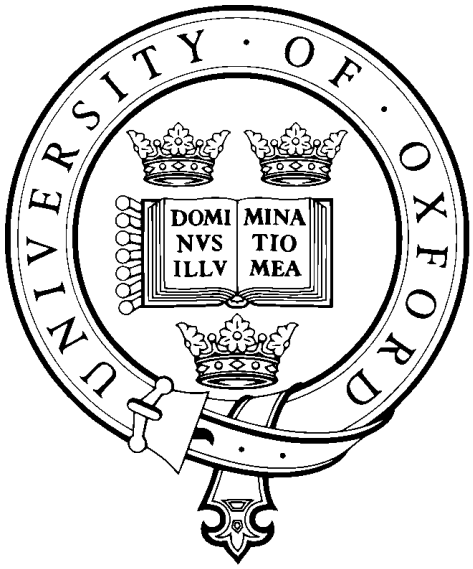
Simulation of, and with, First Generation Quantum Computers



Tyson Jones

Corpus Christi College

Supervised by

Professor Simon C. Benjamin

A thesis submitted to the Department of Materials, University of Oxford,
in Hilary Term, 22nd April, 2022, for the degree of
Doctor of Philosophy

Acknowledgements

It is an intimidating task to adequately thank my supervisor Simon C. Benjamin. Without any hesitation, I testify him to be the most stunningly intelligent, disarmingly charming, inspiringly enthusiastic and stirringly caring leader I have yet had the honour of being influenced by. But in equal measure, I am ungrateful that I may be so cursed as to one day leave his supervision: to leave unsettled our debates on what is moral, what is economic, and what is simulatable; to forego his improvised lectures on physics masterfully rendered in simple terms; to conclude my discipleship in matters bureaucratic and administrative. If not for Simon's endless patience; his understanding reassurances of "these things happen" when disaster strikes; his selflessness in accepting meetings at unsociable hours when timezones constrain; and his general tolerance of my tomfoolery, I would not have survived this doctorate.

I owe thanks also to a string of fantastic educators before him, not limited to: Georgina Tangey for steering me away from medicine; Linda McIver for steering me toward computational science; Kathryn Grainger and Alex Gavrilescu for marrying me to physics; and the inexhaustible John Kermond for infecting me with his contagious passion for mathematics. So too I thank my former research supervisors: Jon McCormack, Tim Garoni, Russell Anderson, Lincoln Turner, Eric Thrane, Stephen Wolfram, Tapio Simula, Frank Wuerthwein, Christian Walder, and the incredible Arun Konagurthu who long ago ascended to the status of academic mentor and life coach. I must also thank David Albrecht, Neil Carmona-Vickery, and Peter Corkill for their profound effects on my academic trajectory. With only modest embarrassment, I further thank both Peter and Elly Corkill for permitting me their fold-out couch during my final highschool exams. Anything confined within the margins of this page is insufficient to capture my indebtedness to, gratitude toward, and admiration of the Corkill family.

Next, the score to my peers: I thank Chris Whittle in ways inexpressible by even the pithiest thesis acknowledgement. I thank Shi Qiu for sharing my journey from a collapsing Anglosphere through to crumbling Western European institutions. I thank Patrick Inns on behalf of the human race for being a saintly bastion of unending kindness, and

on my own behalf for his many proofreadings of my manuscript drafts, recited aloud. Had we been together at the completion of this thesis, I am sure it too would have echoed down the Banbury staircase in his magnificent accent. I thank Alex Kenny for his meticulous reading of the 358 drafted pages of this thesis in an act of astonishing diligence and dedication paralleled only by his directorial contributions to the Life at Banbury sitcom special (link). I thank Anna Moloney for every motivating "gleeba" toward the needlessly stylised diagrams herein. Many of us struggle to find purpose, helplessly adrift in this Nihilistic late stage capitalistic hellscape. I only wish everyone could experience the existential validation of a beautifully rendered vector graphic and Anna's admiration. I thank every philosopher (even the profanely many Heideggerians) who enriched beyond measure my time in Oxford and whose engagements did more for my intellectual development than my doctoral studies. Andrea Vitangeli, Elisabeth Huh, Luke Luttmann, Erick Spahr, Henry Straughan, and the honorary philosopher Sinan Shi share equal blame for my collapse of character. And in proportion, I thank Leo Trotz–Liboff for his efforts to undo these enlightenments as a proponent of the *guile* and *relentless* dichotomy through which I have since vowed to measure all things.

I thank the fellow inaugural babies of Banbury 2017 (not forgetting Jessie Lim, Adrienne Propp, James Famelton, Chris Fuller, Allen Zhang, Joshua Carter, Tony Liu, Jiate Luo, and Liban Alcantara), the fellow victims of St Mary's Rd 2018 (especially the upliftingly kind Robert Laurella), the fellow pandemic survivors of Banbury 2021 (Marius Emanuel, Christos Konstantopoulos, Sean Johnston, Kae Ono and the ceremoniously recognised Jane Lau), the fellow MCR layabouts (Kira Schützenhofer, Nora Kelemen, Max Jenkins, Freddy Trinh, Hailey Trier, Tara Lee), my fellow Clarendon Scholars (especially Linda Qian, Matthias Aengenheyster, Alessandro Lodi, Mark Brooke and Emma Blümke), my fellow colleagues of QTechTheory (Suguru Endo, Sam McArdle, Xiao Yuan, Zhenyu Cai, Xiaosi Xu, Takahiro Tsunoda, Richard Meister, Sam Jaques, Bálint Kozcor, Arthur Rattew, Cica Gustiani, Carlos Outeiral, Armands Strikis, Matt Goh, Hamza Jnane, Hans Chan, and the newer torchbearers from whom this thesis prevented my meeting), and my fellow debauchers of the House of Delphi.

I offer little thanks to the conspiring hands of fate which through my DPhil saw my stay in a ramshackled Cowley hovel, a pandemic induced suspension of my internship, my victimisation of a violent crime, and my stranding in the United Arab Emirates. The gratitude must instead go to the altruistic Penny Meallin, without whom I may still be stuck in Dubai International Airport, or writing these words among the street cats of Istanbul, Turkey. I am thus fortunate to thank the staff and fellow Melbourne patrons of the State Library Victoria, the Monash Caulfield library, Marche Board Game Cafe, Starbucks on Swanston St, the Soap Bar Launderette, and the Fortress videogame bar wherein this thesis was primarily written over three gruelling months. I am proud to have earned the title therein as "the English tea guy".

I thank the Clarendon Fund for their generous award of the Clarendon Scholarship, Corpus Christi College for the A. E. Haigh Scholarship, and the Department of Materials for additional funding. So too I thank Quantum Motion Technologies, and IBM Research UK. I similarly thank Udon Yasan on Bourke St for their barely edible seaweed udon which at $6 a bowl - only 30% of the present minimum hourly wage - enabled the timely end of this thesis and prevented the untimely end of its author.

Finally, I thank my family; my nana Jan Cross for her endless praise and inexhaustible kindness; my mother Lianne Jones for her unyielding dedication through a tumultuous upbringing; my uncle Boo Cross for his early lessons in matters academic and cinematographic. Finally, I thank my adored twin sister Kea Jones and the long distance oxytocin she remotely administered through regular updates from her two furry children; Puffington and Cooper.

Abstract

The year is 2022. Scientists and engineers inch ever closer to building a practical quantum computer. The excitement in the research community, that we might soon fulfil Feynman's dream to leverage quantum mechanics in machines capable of exponentially quicker computation [1], is steadily growing. With promised revolutions in chemistry [2], condensed matter physics [3] and machine learning [4], and an expected market of 1.8 billion USD by 2026 [5], quantum computing fights for the spotlight amid international pandemics and climate catastrophe. But the journey ahead is not an easy one. Quantum computers, requiring precise control of extraordinarily delicate quantum systems, are unsurprisingly challenging to engineer and equally difficult to design. A demonstration of *quantum advantage* [6] through the quantum solving of a practical problem faster than available classical means, remains a research ambition.

At the forefront of this research are classical computers: the machines which crunch the numbers in our calculations, which interface with our quantum experiments, and which render this very document. Without them, quantum computation would have remained a fanciful whim on Feynman's chalkboard. Behind the rapidly growing repertoire of quantum algorithms lies an equally impressive and expanding corpus of classical simulation strategies.

This thesis is about utilising first generation quantum computers and predicting their behaviour using classical computers. It develops novel quantum algorithms to perform variational minimisation, Hamiltonian diagonalisation, and approximate circuit recompilation, all intendedly compatible with near-future machines. It also devises classical algorithms for efficiently simulating quantum variational algorithms, emulating quantum computers using high-performance supercomputing facilities, and showcases the author's efforts in scientific software development. Incidentally, this thesis makes no direct use of current day quantum hardware facilities. We hope to convince the disappointed reader that such an endeavour is presently pointless.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Foreword

This thesis presents novel quantum algorithms designed for near-future quantum computers, and novel classical algorithms to accelerate their numerical study. It is split equally between these topics: Chapters 2-4 present quantum algorithms and Chapters 5-7 present classical simulation algorithms and software. For a reader interested in only one of these areas, it will suffice to read this introductory chapter which assumes little expertise in either area, then proceed to Chapter 2 or 5.

The thesis will visit in-turn the work of each of the published manuscripts:

- S. McArdle, **T. Jones**, S. Endo, Y. Li, S. C. Benjamin, X. Yuan
  Variational ansatz-based quantum simulation of imaginary time evolution
  npj Quantum Inf **5**, 75 (2019) (link)

- **T. Jones** and S. Endo, S. McArdle, X. Yuan, S. C. Benjamin
  Variational quantum algorithms for discovering Hamiltonian spectra
  Phys. Rev. A **99**, 062304 (2019) (link)

- **T. Jones**, S. C. Benjamin
  Robust quantum compilation and circuit optimisation via energy minimisation
  Quantum **6**, 628 (2022) (link)

- **T. Jones**, A. Brown, I. Bush, S. C. Benjamin
  QuEST and high performance simulation of quantum computers
  Scientific Reports **9**, 10736 (2019) (link)

- **T. Jones**, S. C. Benjamin
  QuESTlink - Mathematica embiggened by a hardware-optimised quantum emula-

tor

Quantum Sci. Technol. **5**, 3 (2020) (link)

in addition to the works presently under review:

- **T. Jones**, J. Gacon

  Efficient calculation of gradients in classical simulations of variational quantum algorithms

  arXiv:2009.02823 (2020) (link)

- **T. Jones**

  Efficient classical calculation of the quantum natural gradient

  arXiv:2011.02991 (2020) (link)

and the works in preparation:

- **T. Jones**, C. Beaudoin, B. Koczor, S. C. Benjamin

  Distributed algorithms for simulating quantum computers

- **T. Jones**

  Faster emulation of noisy quantum variational algorithms

We will also present or make mention of the software projects:

- **T. Jones**, A. Brown, S. C. Benjamin, I. Bush, B. Koczor, R. Meister, *et. al.*

  The Quantum Exact Simulation Toolkit

  DOI: 10.5281/zenodo.**6475156** (2017 - 2022)

  Hosted on Github `QuEST-Kit/QuEST` (link)

- **T. Jones**, B. Koczor

  QuESTlink

DOI: 10.5281/zenodo.**6475200** (2018 - 2022)

Hosted on Github `QTechTheory/QuESTlink` (link)

- **T. Jones**

  Dissipative Recompiler

  DOI: 10.5281/zenodo.**6475224** (2018)

  Hosted on Github `QTechTheory/DissipativeRecompiler` (link)

Without the space to include the necessary background material, we regretfully exclude the thematically relevant published, submitted and developing works:

- B. Koczor, S. Endo, **T. Jones**, Y. Matsuzaki, S. C. Benjamin

  Variational-state quantum metrology

  New J. Phys. **22**, 083038 (2020) (link)

- H. H. S. Chan, R. Meister, **T. Jones**, D. P. Tew, S. C. Benjamin

  Grid-based methods for chemistry modelling on a quantum computer

  arXiv:2202.05864 (2022) (link)

- C. Gustiani, **T. Jones**, F. Gonzalez-Zalba, S. C. Benjamin

  Circuit synthesis tailored to quantum machines via virtual quantum devices

Though this thesis will attempt to be cohesive and self-contained, some sections will feature grey, italicized text which clarifies the thematic relevance of the passage and the author's research contributions. These are primarily for the benefit of my examiners. All forewords (such as this one) can be considered text of this kind, and are thus not styled.

1.2 Introduction

What is quantum computation and why is it powerful? The author has enjoyed frequent debates provoked by this well-intentioned question, had with physicists, computer scientists and the public alike. To attempt an answer is to navigate a minefield of epistemic nuance about physical systems and their models [7]; to conjecture about the surmountability of engineering barriers [8]; or to gesture toward yet-maturing computational resource theories [9]. Avoiding any such nuisance, this introductory chapter will answer through a rapid historical review beginning in Section 1.3. It begins with the development of classical computation before describing the analogous strides in the younger quantum field. We summarise the goals of next-generation quantum computers, the brief history of experiment, and the limitations of current day prototypes. Section 1.4 introduces the mathematical tools to describe digital quantum computers as used in this thesis. We finally highlight several specific quantum algorithms of particular relevance to this thesis. These are Trotterisation in Section 1.5, then three variational algorithms in Section 1.6 which includes a general introduction to the growing field of quantum variational algorithms.

Portions of this chapter are taken verbatim from a literature review previously written (solely) by the author as an internal departmental document.

1.3 History

This section provides a historical review of research in classical and quantum computing. We begin with *yesterday's* theory of classical computation (Sec. 1.3.1), focusing on that relevant to the subsequent quantum developments. We then look at the early theory

of quantum computation (Sec. 1.3.2) which predominantly assumed perfect large-scale quantum devices. Such machines are generally accepted to be experimentally inaccessible today. As such, yesterday's quantum theory somewhat whimsically anticipates *tomorrow's* quantum computers. Finally, we survey *today's* experiments in building quantum computers (Sec. 1.3.3) which are regrettably imperfect, small-scale and highly constrained. This latter section is the main motivation for the research direction of this thesis.

1.3.1 Yesterday

The theory of digital computation begins with Turing's seminal 1937 paper [10] which introduces a mathematical model for a computer. Later dubbed a *Turing machine*, this computer stores its program like data. In this sense it is universal, unlike machines with fixed programs. The real design of such a machine became known as the Von Neumann architecture based on Von Neumann's 1945 report [11] of the EDVAC machine in development by Eckert and Mauchly. This is despite Zuse's patent for a similar design in 1936 to extend his electromechanical *Versuchsmodell* computer [12]. The first implementation of the Von Neumann architecture is accepted to have been completed in 1948 [13], though it suffered from spontaneous errors unsolvable by improved engineering alone. So two years later, Hamming described classical error correcting codes which employ more computer memory to detect and remedy logical errors in such a computer [14].

In 1960, Rabin *et al.* introduced the theory of computational complexity [15] (named as such 5 years later [16]): a framework for studying and describing the runtime scalability of algorithms. Several years later, Cobham described the class of problems P

which have solving algorithms that scale polynomially with the problem size [17] and which are considered tractable. For example, computing the square root of a decimal number. The same year, Edmonds introduced NP [18], the class containing problems with exponential scaling or worse, such as finding the shortest path which connects a set of coordinates. In 1971, Cook described the class NP-complete [19], a subclass of NP containing problems for which the solutions can be transformed in polynomial time to solve any other NP problem. His example problem was the boolean satisfiability (3SAT) problem, where one seeks the satisfying boolean assignment of a many-variable propositional formula. Karp exemplified the significance of the NP-complete class the following year [20], realising that finding a polynomial time solution to an NP-complete problem offers a route to polynomial time solving *all* NP problems, and would imply $P \equiv NP$. It is widely believed $P \neq NP$ [21]. The next 7 years would see Gary *et al.* compile a list of over 300 problems established to be NP-complete [22]. In 1973, Bennett would demonstrate that the logically irreversible Turing machine could be made reversible [23]. This implies that even machines constrained to be thermodynamically reversible can exhibit the same universal computational power of the Turing machine.

All the meanwhile, many notable classical algorithms were developed which would later become relevant to quantum computation. This includes the fast Fourier transform to compute the discrete Fourier transform of an n-length vector in $\mathcal{O}(n \log n)$ steps [24]. An improved method for finding the prime factors of any integer was devised and rigorously proven to be scale exponentially with the magnitude of the factorised integer [25]. This assured that the recently established RSA cryptosystem [26] was secure, provided the secret key to be factorised was kept very large. A drastic improvement [27] saw a restored excitement in using neural networks [28] to perform algorithmically complicated tasks. Simulated annealing combined with a Metropolis algorithm, whereby a candidate solution is perturbed in order to cool an analogous physical system, was

shown to be an efficient heuristic technique for solving optimisation problems [29].

When the close of 1983, the theoretical foundation of classical computational science had been laid, and musings of a new quantum technology had already begun.

1.3.2 Tomorrow

As early as 1959, Feynman was conceptualising machines which exploit the laws of quantum mechanics to perform computation [30]. This was first formalised by Benioff in 1980 [31] with a quantum extension of the Turing machine [10], which Bennett had already shown could be made thermodynamically reversible [23]. Feynman used Bell's theorem [32] to argue that Benioff's quantum Turing machine could not be simulated by classical means [1], though this had been reasoned by Poplavskii 7 years earlier [33] and would be proven rigorously by Lloyd 14 years later [34].

Deutsch was first to recognise a *universal* quantum computer in 1985 [35], demonstrating Benioff's quantum Turing machine could model any continuous system with finite memory and time: an impossible task for Turing's discrete classical machine. By 1992, Deutsch and Jozsa [36] had discovered a class of problems which were exponentially faster to solve with a quantum machine than with classical means, and yet would be further improved upon [37].

While the Deutsch–Jozsa algorithm could be dismissed as having no practical consequence, the community was shaken by Shor's 1994 description of a quantum algorithm to find the prime factors of integers in polynomial time [38]. Being capable of breaking RSA cryptosystems, Shor's algorithm ignited the field of quantum cryptography and demonstrated the revolutionary power of the qubit: the fundamental memory block

of a quantum computer which would receive its name the next year [39]. By the time Grover discovered a quadratic speedup for list searches using quantum memory [40], the community was convinced of an ideal quantum computer's usefulness and had turned its attention toward the problems in its physical instantiation.

Decoherence, whereby a quantum system interacts with its environment and becomes classical [41], was shown to limit the possible speedup of a quantum machine [42] and returned the practicality of quantum computation to a matter of debate [43, 44]. Then in 1996, Shor *et al.* devised quantum error correcting schemes which used additional physical qubits and operations to repair logical qubits which have decohered [45, 46, 47, 48, 49, 50]. These codes were realised to be employable for fault-tolerance even when the correcting operations themselves contain errors [51, 52, 53], provided such errors are sufficiently infrequent [54]. The community promptly set to estimating [55] and lowering [56] this fault-tolerance *threshold*, which when surpassed, was shown to enable arbitrarily accurate quantum computation [57]. These correction protocols were quickly improved by Steane [58, 59] and their resource costs reduced by Gottesman and Chuang [60] by utilising the phenomenon of quantum teleportation discovered only 6 years earlier [61].

In the meantime, it was shown that general quantum unitaries could be enacted with small universal operation sets [62, 63] whilst maintaining fault tolerance [64], and well approximated with short sequences of operations from a fixed set [65]. A quantum calculation of the Fourier transform of a length-n vector was discovered by Kitaev [66] and improved by Hales *et al.* [67] to be $\mathcal{O}((\log n)^2)$ in runtime and use only $\log_2 n$ qubits: a logarithmic improvement over the classical fast Fourier transform. The quantum Fourier transform (QFT) would be used in many future quantum algorithms which achieve exponential speedups over their classical counterparts. Introducing quantum

fluctuations in lieu of the thermal fluctuations in simulated annealing was shown to improve convergence to the solution of optimisation problems [68, 69]. In 1997, Bernstein and Vazirani would supply the first formal evidence that Deutsch's quantum Turing machine violates the complexity-theoretic Church-Turing thesis [70], demonstrating that not even a *probabilistic* classical Turing machine can efficiently simulate a quantum Turing machine.

At the turn of the millennium, *adiabatic* quantum computing was conceived [71]. In this new paradigm, a quantum system was kept cool and adiabatically changed so that the groundstate of its describing Hamiltonian was transformed into the solution state of some optimisation problem. Adiabatic algorithms were first developed to solve the NP-complete 3SAT [71] and exact cover [72] problems, although not in polynomial quantum resources, and an adiabatic approach to Grover's algorithm with the same complexity soon followed [73]. This adiabatic paradigm was soon proven to be equivalent to the circuit based model: digital quantum computers were proven capable of polynomially simulating adiabatic computers [74], and the converse proven 7 years later [75]. Their equivalence meant the mathematics of studying an adiabatic algorithm's performance by its Hamiltonian spectrum could be used for studying circuit algorithms. In 2002, Childs *et al.* [76] showed that exponential speedups are not limited to algorithms which use the QFT, but can be achieved with oracular algorithms based on the quantum walk, previously shown exponentially faster than the classical random walk [77].

The next two decades saw proposals for quantum acceleration of machine learning with neural networks [78], the solving of general linear equations [79] and Monte Carlo methods [80], and a wealth of other quantum applications. Such far-future quantum algorithms remain an active area of research. An appetite for fault-tolerant large-scale quantum computers on which to deploy them has been well and truly whet.

1.3.3 Today

The experimental history of general purpose quantum computation began around the anticipated Y2K downfall of classical computation [81]. Thankfully, both technologies survived. By 1998, qubits had been experimentally realised [82], quantum logical gates had been implemented, and the Deutsch–Jozsa algorithm had been run in a 2-qubit NMR experiment [83]. Quantum teleportation was demonstrated using one [84] and two [85] photons, and quantum error correction [86] and other strategies [87] to fight decoherence performed with few qubit NMR machines. Such machines soon realised Shor's algorithm to factorise 15 into 3×5 [88], surpassed 10 years later by the factorisation of 21 into 3×7 with a photonic computer [89]. These efforts have not yet approached the classically feasible regime of $\approx 10^{240}$ [90].

By 2011, Paul's ion trap design [91] - first demonstrated performing a controlled-NOT gate in 1995 [92] - had suspended 14 entangled qubits [93]. And the same year, the first quantum platform of the Von Neumann architecture was realised via two superconducting qubits [94]. An adiabatic evolution algorithm to compute Ramsey numbers was performed in 2013 using 28 logical qubits represented by 84 physical ones [95]. The gate fidelity threshold for fault tolerant quantum computing would be reached with superconducting qubits in 2014 [96], though with significantly fewer qubits than needed to implement error correction.

We have reached 2017, when the research of this thesis began. Today's quantum computers evidently leave much to be desired. Indeed, none of the previously mentioned quantum architectures have yet been proven viable for practical computation. Ion trap, neutral atom, silicon chip, semiconductor, topological and a range of other hardware remain candidate platforms with their own noise profiles, native operations, qubit

connectivity constraints and obstacles to scaling. They are all presently incapable of executing any one of the aforementioned quantum algorithms in classically intractable regimes. If a quantum algorithm is soon to outperform its classical counterpart, it will be one specifically tailored for today's noisy intermediate-scale quantum (NISQ) computers [6].

1.4 Modelling

This section offers a minimal review of the language and mathematical tools for modelling digital quantum computers, as are relevant to this thesis. We divide this into two parts: the modelling of *pure* states admitted by perfect quantum computers through the statevector formalism, and that of *mixed* states of imperfect computers through density matrices.

We adhere to the following notation and symbol conventions:

notation	**meaning/type**
x	complex scalar, unless stated otherwise
x^*	complex-conjugate of scalar x
$\vec{x}$	vector with scalar elements $\{x_i : i \in \mathbb{N}\}$
$\underline{\underline{x}}$	matrix with scalar elements $\{x_{ij} : i, j \in \mathbb{N}\}$
$\hat{x}$	operator
$\hat{x}^\dagger$	complex-conjugate transpose of $\hat{x}$
$\hat{x}_t$	operator acting upon qubit of index t
i	imaginary unit
$\hat{\mathbb{1}}^{\otimes n}$	identity operator upon n qubits (a $\mathbb{R}^{2^n \times 2^n}$ matrix)

$[\hat{a}, \hat{b}]$	commutator $\hat{a}\hat{b} - \hat{b}\hat{a}$
$\hat{\sigma}$	Pauli operator
$\hat{H}$	Hamiltonian
$\mathcal{H}$	Hilbert space
$\|\psi\rangle$ (Greek alphabet)	general pure quantum state ($\in \mathcal{H}$)
$\|i\rangle$ (English alphabet)	computational basis state ($i \in \mathbb{N}$)
$\|i\rangle_N$	pure state of N qubits
$i_{[j]}$	j-th bit of binary representation of i ($i, j \in \mathbb{N}$)
$\cong$	equivalence up to global phase
$\hat{\rho}$	density operator (disambiguated from equivalent density matrix $\underline{\underline{\rho}}$)
$\mathcal{D}$	completely-positive trace-preserving channel
$\delta_{i,j}$	Kronecker delta function
$\otimes$	Kronecker product
$\mathbb{E}[x]$	expected value of random variable x
$\langle\hat{x}\rangle$	expected measurement outcome of observable operator $\hat{x}$
$\mathcal{O}(f(x))$	function bounded by $f(x)$ as $x \to \infty$ (big O notation)

1.4.1 Pure

Statevectors

The statevector is one of many mathematical representations of a digital quantum state without noise nor classical uncertainty. An N-qubit *pure* state can be described by a

statevector $|\psi\rangle \in \mathcal{H}$,

$$|\psi\rangle = \sum_{i}^{2^N} \alpha_i \, |i\rangle \, , \qquad \alpha_i \in \mathbb{C}, \qquad \sum_i |\alpha_i|^2 = 1, \tag{1.1}$$

where $\mathcal{H}$ is a Hilbert space, or an L^p Lebesgue space with a $(p = 2)$-norm [97]. Here, $|i\rangle$ is one of the 2^N possible classical configurations of the system, formed by one-qubit states

$$|i\rangle = \bigotimes_{j}^{N} |i_{[j]}\rangle \, , \qquad |i_{[j]}\rangle \in \{|0\rangle \, , \, |1\rangle\}, \tag{1.2}$$

and α_i are the normalised complex amplitudes which encode the quantum state's population between the classical possibilities. We have used $i_{[j]}$ to indicate the j-th bit $(j \geq 0)$ of the integer $i \geq 0$. An amplitude is sometimes referred to in terms of its magnitude $m = |\alpha_i| \in \mathbb{R}$ and phase $\phi \in \mathbb{R}$, whereby

$$\alpha_i \equiv m \, e^{\mathrm{i}\phi}. \tag{1.3}$$

We are said to be working in the $\hat{Z}$-basis if our chosen bit states, $|0\rangle$ and $|1\rangle$, are eigenstates of the Pauli $\hat{Z}$ operator, as will be assumed for the remainder of this thesis. The statevector can describe the phenomena of superposition and entanglement as properties of $\{\alpha_i\}$. The Born rule [98] asserts that the probability of a quantum state $|\psi\rangle$ being in a particular N-qubit classical configuration i is given by

$$\text{Prob}\left(|\psi\rangle \cong |i\rangle\right) \quad = |\, \langle i|\psi\rangle \,|^2 \quad = |\alpha_i|^2, \tag{1.4}$$

where we have employed a "bra" vector $\langle i| = |i\rangle^\dagger$ of the conjugate Hilbert space $\overline{\mathcal{H}}$, and the L^2 inner product defined as

$$\langle \Psi|\psi\rangle = \sum_i^{2^N} \beta_i^* \, \alpha_i \; \in \mathbb{C}, \qquad \text{letting } \; |\Psi\rangle = \sum_i \beta_i \, |i\rangle \, . \tag{1.5}$$

We use the symbol $\cong$ here to mean "equal amplitudes up to global phase" which we elaborate on momentarily. The probability of a single qubit of index $q \geq 0$ being in the classical state $s \in \{0, 1\}$ is informed by the superposition,

$$\text{Prob}\left(|\psi\rangle_q \text{ is } s\right) = \sum_i^{2^N} \delta_{s,i_{[q]}} \, \text{Prob}\left(|\psi\rangle \cong |i\rangle\right) \tag{1.6}$$

$$= \sum_i^{2^N} \delta_{s,i_{[q]}} \, |\alpha_i|^2. \tag{1.7}$$

Beyond the normalisation of the amplitudes to yield probabilities $\in [0, 1]$, there is an additional degeneracy hidden in the statevector description: the so-called "global phase". Applying a unit complex prefactor $e^{\mathrm{i}\phi}$ unto every amplitude will not modify the relative phases between amplitudes, nor affect the probability distributions - it will in fact cause no change at all, and is considered a non-physical artefact of the statevector description. As such, states $|\psi\rangle$ and $|\Psi\rangle$ of the same Hilbert space which satisfy $|\psi\rangle \equiv e^{\mathrm{i}\phi} \, |\Psi\rangle$ describe the same physical state. Global phase degeneracy can sometimes prove a minor nuisance of the statevector formalism, as we will witness in this thesis.

Otherwise, statevector descriptions lend themselves well to classical simulation of quantum computers, where the amplitudes can be encoded as floating-point complex numbers, and hence the quantum state as a numerical vector. Of course, the exponentially growing memory costs to store 2^N such amplitudes strictly limit the sizes of the systems

which can be classically simulated. We will explore this further in Chapter 6.

The evolution of a quantum state can be modelled by operators acting upon the statevector which change the relative magnitudes and complex phases of the amplitudes.

Operators

This thesis will make explicit use of only *three* families of operators to describe the transformation of statevectors. These are *unitaries*, *projectors*, and *Hermitian operators*, and are fundamentally linked. Loosely, they respectively relate to "advancing" of the quantum state, measuring the state, and modelling the distribution of the measurement outcomes. We defer discussion of measurement and projective operators to the next section.

A unitary operator $\hat{U}$ satisfies $\hat{U}^{-1} = \hat{U}^\dagger$, where the conjugated-transposed operator $\hat{U}^\dagger$ remains unitary. In a sense, unitary operators are *reversible* and describe coherent operations on a quantum state that do not induce any classical noise nor uncertainty. Any unitary can be decomposed into a family of complete, primitive operators on a quantum computer [62, 63] and as such are considered the basic units of digital quantum computation. In this thesis, we will say the "action" of a unitary $\hat{U}$ (with matrix $\underline{\underline{U}}$ in the computational basis) upon an N-qubit pure state $|\psi\rangle$ is to modify its statevector amplitudes as implied by the matrix-vector multiplication of

$$|\psi\rangle \to \hat{U}\,|\psi\rangle\,, \qquad \underline{\underline{U}} \in \mathbb{C}^{2^N \times 2^N}. \tag{1.8}$$

We will also compactly notate a unitary applied to a *single* qubit using a subscript:

applying $\hat{U}$ to the q-th qubit ($q > 0$) of N qubits is notated by

$$\hat{U}_q \left|\psi\right\rangle = \mathbb{1}^{\otimes N-q-1} \otimes \hat{U} \otimes \mathbb{1}^{\otimes q} \left|\psi\right\rangle . \tag{1.9}$$

We will somewhat flippantly switch between matrix and representation-free treatments of operators, when convenient, of which we are already guilty. Some canonical named unitaries appearing in this thesis (here expressed in the $\hat{Z}$ basis) are

$$\text{Hadamard} \quad \hat{\mathrm{H}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \hat{S} = \begin{pmatrix} 1 & \\ & e^{\mathrm{i}\frac{\pi}{4}} \end{pmatrix}, \tag{1.10}$$

in addition to controlled unitaries (expressed using projectors in the proceeding section) and the Pauli operators $\hat{\sigma} \in \{\mathbb{1}, \hat{X}, \hat{Y}, \hat{Z}\}$. The Pauli operators have the form of the Pauli matrices, and happen to be unitary ($\hat{\sigma}^\dagger = \hat{\sigma}^{-1}$), idempotent ($\hat{\sigma}^2 = \hat{\mathbb{1}}$) *and* Hermitian ($\hat{\sigma}^\dagger = \hat{\sigma}$).

Hermitian operators are self-adjoint, satisfying $\hat{O} = \hat{O}^\dagger$. The postulates of quantum mechanics assert than every measurable observable corresponds to a Hermitian operator. We describe measurement in the proceeding section. The most notable Hermitian operator is the Hamiltonian $\hat{H}$ of the energy observable which Noether's theorem [99] demonstrates is intimately linked with time dynamics of a system. Incidentally, this thesis will only consider Hermitian operators which constitute Hamiltonians (which we label as $\hat{H}$), or which generate unitary gates (which we label $\hat{G}$).

Every Hermitian operator $\hat{O}$ can be expressed with real coefficients in a complete basis of Hermitian operators $\{\hat{O}_j\}$.

$$\forall\, \hat{O} = \hat{O}^\dagger \quad \exists\, \{h_k \in \mathbb{R}\} \quad \text{s.t.} \quad \hat{O} \equiv \sum_k h_k \hat{O}_k. \tag{1.11}$$

For example, the 4^D possible tensor products of $D \in \mathbb{N}$ Pauli matrices form a complete basis for all Hermitian $\hat{O} : \mathbb{C}^{2^D \times 2^D}$. Meanwhile, every fixed unitary $\hat{U}$ can be generated by a corresponding Hermitian operator $\hat{G}$,

$$\forall\, \hat{U} \in \mathrm{SU}(D) \qquad \exists\, \hat{G} \qquad \text{s.t.} \qquad \hat{U} \equiv \exp(\mathrm{i}\,\hat{G}), \qquad \hat{G} = \hat{G}^\dagger. \tag{1.12}$$

This means every unitary can be expressed in terms of Pauli operators and real scalars λ_i, as

$$\hat{U} \equiv \exp\left(\mathrm{i} \sum_{i}^{4^D} \lambda_i \bigotimes_{j}^{D} \hat{\sigma}_{ij}\right). \tag{1.13}$$

In this thesis, we reserve D to indicate the (logarithm of the) dimension of the generating Hermitian operator of unitary gates, to disambiguate it from N which is the (logarithm of the) dimension of the Hamiltonian and corresponding quantum states. In this way, N always refers to the total number of qubits in a system, whereas $D \leq N$ denotes only the number of qubits targeted by a particular unitary operator.

Every possible smooth real-parametrization of a Hermitian operator is expressible in a static complete basis

$$\hat{O}(\vec{\theta}) = \sum_{k} h_k(\vec{\theta})\ \hat{O}_k, \tag{1.14}$$

where $h_k(\vec{\theta}) : \mathbb{R}^{\dim(\vec{\theta})} \mapsto \mathbb{R}$. This means that every real parametrisation of a unitary

$\hat{U}(\vec{\theta})$ can hence be generated through real functions h_k or λ_i,

$$\hat{U}(\vec{\theta}) \equiv \exp(\mathrm{i}\,\hat{G}(\vec{\theta})) \tag{1.15}$$

$$\equiv \exp\left(\mathrm{i}\sum_k h_k(\vec{\theta})\;\hat{G}_k\right) \tag{1.16}$$

$$\equiv \exp\left(\mathrm{i}\sum_i^{4^D} \lambda_i(\vec{\theta})\;\bigotimes_j^D \hat{\sigma}_{ij}\right). \tag{1.17}$$

This also informs us that the derivative of a general unitary (itself, skew-Hermitian and generally non-unitary) can be expressed as a new Hermitian operator upon the unitary,

$$\frac{\partial\hat{U}(\vec{\theta})}{\partial\theta_i} = \mathrm{i}\,\frac{\partial\hat{G}(\vec{\theta})}{\partial\theta_i}\;\exp(\mathrm{i}\,\hat{G}(\vec{\theta})) \tag{1.18}$$

$$= \mathrm{i}\left(\sum_k \frac{\partial h_k(\vec{\theta})}{\partial\theta_i}\hat{G}_k\right)\hat{U}(\vec{\theta}) \tag{1.19}$$

$$= \mathrm{i}\,\hat{\Lambda}(\vec{\theta})\,\hat{U}(\vec{\theta}) \tag{1.20}$$

where $\hat{\Lambda} = \hat{\Lambda}^\dagger$. Note that in general, $\hat{\Lambda} \neq \hat{G}$ and ergo $[\hat{\Lambda},\,\hat{U}] \neq 0$, unless all h_k are linear in all θ_i.

The ability to relate parameterised unitaries (and their derivatives) to their generating Hermitian and specifically Pauli operators in this way will be an important tool in this thesis. It will generally be the case that the actual number of Pauli tensors needed to generate a useful quantum gate is $\ll 4^D$. In fact, most often a *singly*-parametrized *single* term is needed with a linear function $\lambda(\theta) = c\,\theta$ for some constant $c \in \mathbb{R}$ (usually $c = \frac{1}{2}$ or $c = 1$). This still admits D-qubit entangling Pauli gadgets of the form

$$\hat{U}(\theta) = \exp\left(\mathrm{i}\,c\,\theta\bigotimes_j^D \hat{\sigma}_j\right), \tag{1.21}$$

which remain a powerful entangling gate. Furthermore, their derivatives

$$\frac{\partial \hat{U}(\theta)}{\partial \theta} = \mathrm{i}\, c \left(\bigotimes_j^D \hat{\sigma}_j \right) \hat{U}(\theta), \tag{1.22}$$

do have the form of commuting operators, i.e. $[\hat{U}, \otimes_j \hat{\sigma}_j] = 0$.

Measurements

Measurements are our means to obtain classical information about an otherwise inscrutable quantum state. They are the process of non-unitarily (when the measurer is outside the system) transforming a state (or substate) into a single of its possible classical states, randomly, with likelihoods informed by the amplitudes [98]. This thesis will need no more complicated a concept than *projective* measurement whereby measuring an observable yields an eigenvalue of, and "collapses" the state into an eigenstate of, a corresponding Hermitian operator. Any Hermitian operator $\hat{O}$ can be expressed in terms of projectors

$$\hat{O} = \sum_i e_i \left|e_i\right\rangle \left\langle e_i\right|, \tag{1.23}$$

where e_i is the eigenvalue of eigenstate $|e_i\rangle$. After measurement of $\hat{O}$, a quantum state $|\psi\rangle$ is left in state

$$|\psi\rangle \underset{\hat{O}}{\rightarrow} |e_i\rangle \quad \text{with probability} \quad |\langle e_i|\psi\rangle|^2. \tag{1.24}$$

The expected eigenvalue outcome of the observable measurement is the expectation value of its Hermitian operator, and given by the inner product,

$$\mathbb{E}[e_i] = \langle\psi|\hat{O}|\psi\rangle \ \in \mathbb{R}. \tag{1.25}$$

We often speak of measuring qubits in the computational basis, transforming a single-qubit state as

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle \ \rightarrow \begin{cases} |0\rangle, & \text{with probability } |\alpha_0|^2 \\ |1\rangle, & \text{with probability } |\alpha_1|^2 = 1 - |\alpha_0|^2, \end{cases} \tag{1.26}$$

which we notate as

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle \ \text{—}\boxed{\nearrow}\text{—} \ |x\rangle, \qquad \text{where} \quad x \in \{0, 1\}.$$

This process is a projective measurement in the $\hat{Z}$ basis, collapsing the qubit to an eigenstate $|0\rangle$ (or $|1\rangle$) of operator $\hat{Z}$ with corresponding eigenvalue $+1$ (or -1), which have been relabelled to bits. If this were the only native measurement basis available to a quantum device, then other observables could be measured by using additional unitaries which transform the eigenstates of the observable into those of $\hat{Z}$. For example, if $|e_1\rangle = a|0\rangle + b|1\rangle$ and $|e_2\rangle = c|0\rangle + d|1\rangle$ were the eigenstates of $\hat{O}$ with eigenvalues ± 1, then constructing

$$\hat{U} = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

and performing

$$|\psi\rangle \ \text{—}\boxed{\hat{U}^\dagger}\text{—}\boxed{\nearrow}\text{—}\boxed{\hat{U}}\text{—} \ |x\rangle, \qquad |x\rangle \in \{|e_1\rangle, |e_2\rangle\}$$

would measure $|\psi\rangle$ in $\hat{I}$, and obtain an eigenstate.

Such is the obvious protocol for evaluating the expectation values of observables on quantum devices. Assume, for example, we study an N-qubit Hamiltonian which can be decomposed into native gates of an N-qubit quantum device, like Pauli operators; $\hat{H} = \sum_i^T h_i \bigotimes_j^N \hat{\sigma}_{ij}$. Then, the expected energy of state $|\psi\rangle$ is,

$$\langle\psi|\hat{H}|\psi\rangle = \sum_i^T h_i \langle\psi| \bigotimes_j^N \hat{\sigma}_{ij}|\psi\rangle \,, \tag{1.27}$$

and can be obtained through T separate experiments, each one repeatedly preparing $|\psi\rangle$, applying a unitary transformation into the $\{\hat{\sigma}_{ij} : j\}$ Pauli basis (usually requiring N separate single-qubit gates), and measuring the targeted qubits in the $\hat{Z}$ basis. The outcome eigenvalue of the full $\{\hat{\sigma}_{ij} : j\}$ operator is the product of the individual qubit eigenvalues (± 1). These are averaged, so that the expectation values are effectively Monte Carlo sampled. After all experiments, these averages are summed, as weighted by h_i, to produce an experimental estimation of the energy. If each unique circuit being experimentally performed undergoes m_E repetitions to sample its expectation value, then, assuming preparing $|\psi\rangle$ costs P gates, the total number of quantum operations performed to measure the energy is

$$\mathcal{O}(m_E\, T\, (N + P)). \tag{1.28}$$

We note there is an emerging literature on improving upon this basic scheme, including near-optimally grouping Hamiltonian terms into commuting groups which can be simultaneously non-destructively measured [100], expanding these groups through unitary transformations [101], and redistributing samples between terms and groups based on their weights h_i [102].

1.4.2 Mixed

Density matrices

Section 1.4.1 introduced the statevector, a mathematical representation of pure or noise-free quantum states. We now introduce the *density operator*, a more general description which can capture uncertainty about the state(s) that a quantum system is in, as (for example) a result of noise or unknown interactions. An N-qubit density operator can be expressed in terms of computational basis-state projectors,

$$\hat{\rho} = \sum_{i}^{2^N} \sum_{j}^{2^N} \rho_{ij} \left|i\right\rangle \left\langle j\right|, \qquad \rho_{ij} \in \mathbb{C}. \tag{1.29}$$

We often use the operator $\hat{\rho}$ and its matrix representation $\underline{\underline{\rho}}$ (dubbed the *density matrix*) of $\hat{Z}$-basis amplitudes ρ_{ij} interchangeably. A density matrix $\hat{\rho}$ is Hermitian ($\hat{\rho}^\dagger = \hat{\rho}$), positive semi-definite (all eigenvalues of $\underline{\underline{\rho}}$ are ≥ 0) and normalised (trace($\underline{\underline{\rho}}$) $= 1$), and can be expressed as a *mixture* of pure states $|\psi_i\rangle$ with respective probabilities p_i as

$$\hat{\rho} \equiv \sum_{i} p_i \left|\psi_i\right\rangle \left\langle \psi_i\right|, \qquad 0 \leq p_i \leq 1, \qquad \sum_{i} p_i = 1. \tag{1.30}$$

By no coincidence, this equivalently describes the mixed state produced by randomly preparing one of the $\{|\psi_i\rangle\}$ pure states, with respective probabilities $\{p_i\}$. In this way, a density matrix has no greater epistemic meaning than the observable distributions it prescribes. The probability of a qubit $q \geq 0$ of the described mixed system being in

classical state $s \in \{0, 1\}$ is then

$$\begin{aligned} \text{Prob}(\hat{\rho}_q \text{ is } s) &= \sum_i p_i \ \text{Prob}(|\psi_i\rangle_q \ \text{ is } s) \qquad (1.31) \\ &= \sum_i p_i \sum_j^{2^N} \delta_{s,j_{[q]}} \ \text{Prob}\left(|\psi_i\rangle \cong |j\rangle\right), \qquad \text{(as per 1.6)} \end{aligned}$$

or, in terms of the diagonal elements of $\underline{\underline{\rho}}$ which are always real,

$$\text{Prob}(\hat{\rho}_q \text{ is } s) = \sum_i^{2^N} \rho_{ii} \ \delta_{s,i_{[q]}}. \qquad (1.32)$$

The density matrix describing a pure state $|\psi\rangle$ satisfies

$$\hat{\rho} = |\psi\rangle \langle\psi|, \qquad \text{trace}(\underline{\underline{\rho}}^2) = 1, \qquad (1.33)$$

while for a mixed state, this latter quantity (called the "purity") satisfies

$$\frac{1}{2^N} \ \leq \ \text{trace}(\underline{\underline{\rho}}^2) \ < \ 1, \qquad (1.34)$$

with minimum achieved by the maximally mixed state $\hat{\rho} = \mathbb{1}/2^N$. Another common quantity is the "fidelity" with respect to a pure state $|\psi\rangle$, defined as

$$\mathcal{F}(\rho, |\psi\rangle) = |\langle\psi|\rho|\psi\rangle|^2, \qquad (1.35)$$

which we will often invoke in this thesis to measure the effect of mixing (through noise and decoherence) during a quantum algorithm. We caution however that this can often be an overly simplistic and misleading measure of accuracy [103, 104], and should at times be substituted with more precise metrics for bounding state distinguishability like the diamond-norm [105].

Unlike statevectors, density matrices are gauge invariant and do not encode an analog of the non-physical global phase discussed in Section 1.4.1. However, a *dense* density matrix contains *square* as many complex elements as a equal-dimension statevector has amplitudes. Ergo, density matrix descriptions quadratically more classically expensive to numerically model. This is sometimes a price worth paying for the powerful analyses possible using the density matrix formalism. We explore this further in Chapter 6.

Operators

A coherent operation which would be modelled by a unitary $\hat{U}$ upon a statevector $|\psi\rangle \to \hat{U}\,|\psi\rangle$, will transform a density operator as

$$\hat{\rho} \to \hat{U}\ \hat{\rho}\ \hat{U}^{\dagger}, \tag{1.36}$$

and will not change its purity. The expectation value of an observable corresponding to Hermitian matrix $\hat{O}$ is

$$\langle O \rangle = \text{trace}\left(\hat{O}\,\hat{\rho}\right) \quad \in \mathbb{R}. \tag{1.37}$$

The algebraic instantiations of Hermitian and unitary operators devised for statevectors will typically be compatible with density matrices of the same dimension.

Channels

The utility of density matrices is their ability to model changes in our knowledge of the quantum state as a result of decoherence and interactions with external systems. We express these processes as channels: completely positive trace-preserving maps from

density matrices *to* density matrices, expressible in the operator-sum representation as

$$\hat{\rho} \;\rightarrow\; \mathcal{D}(\hat{\rho}) = \sum_i \hat{K}_i\,\hat{\rho}\,\hat{K}_i^\dagger, \qquad \text{where} \quad \sum_i \hat{K}_i^\dagger\,\hat{K}_i = \hat{\mathbb{1}}. \tag{1.38}$$

Here, $\{\hat{K}_i\}$ are *Kraus operators* which characterise the channel, and their collection is a Kraus *map*. Some canonical Kraus maps (letting p be their error probability) include

$$\text{amplitude damping:} \qquad \hat{K}_1 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix}, \qquad \hat{K}_2 = \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix}, \tag{1.39}$$

$$\text{dephasing:} \qquad \hat{K}_1 = \sqrt{p}\,\hat{Z}, \qquad \hat{K}_2 = \sqrt{1-p}\,\hat{\mathbb{1}}, \tag{1.40}$$

$$\text{depolarising:} \qquad \hat{K}_{i\in 1,2,3} = \sqrt{\frac{p}{3}}\,\hat{\sigma}^{(i)}, \qquad \hat{K}_4 = \sqrt{1-p}\,\hat{\mathbb{1}}, \tag{1.41}$$

where $\hat{\sigma}^{(i)} \in \{\hat{X}, \hat{Y}, \hat{Z}\}$. The dephasing (depolarising) channels are model the probabilistic erroneous application of the $\hat{Z}$ operator (each of the Pauli operators). The one-qubit homogeneous depolarising channel can be equivalently expressed as

$$\mathcal{D}_\Delta(\hat{\rho}) = \left(1 - \frac{4}{3}\,p\right)\hat{\rho} + \left(\frac{4}{3}\,p\right)\frac{\hat{\mathbb{1}}}{2}, \tag{1.42}$$

revealing $p = 3/4$ induces maximal mixing. This thesis will also feature the two-qubit analogues of the homogeneous dephasing and depolarising channels, acting upon qubits q_1 and q_2, respectively given by

$$\mathcal{D}_\phi(\hat{\rho}) = (1-p)\,\hat{\rho} + \frac{p}{3}\left(\hat{Z}_{q_1}\hat{\rho}\hat{Z}_{q_1} + \hat{Z}_{q_2}\hat{\rho}\hat{Z}_{q_2} + \hat{Z}_{q_1}\hat{Z}_{q_2}\hat{\rho}\hat{Z}_{q_1}\hat{Z}_{q_2}\right), \tag{1.43}$$

$$\mathcal{D}_\Delta(\hat{\rho}) = \left(1 - \frac{14}{15}p\right)\hat{\rho} + \frac{p}{15}\left(\sum_{\sigma,\omega\,\in\{X,Y,Z,\mathbb{1}\}} \hat{\sigma}_{q_1}\hat{\omega}_{q_2}\;\hat{\rho}\;\hat{\sigma}_{q_1}\hat{\omega}_{q_2}\right). \tag{1.44}$$

1.5 Trotterisation

This section briefly introduces Trotterisation for the purpose of real-time quantum simulation, which will serve as a comparative benchmark for other simulation techniques presented in this thesis. As such, it is not original research, although all simulations and their conclusions are my own.

With our small mathematical tool set, we are ready to explore the one and only far-future quantum algorithm explored in this thesis. Its purpose is to simulate the unitary time dynamics of a closed quantum system, as is the frequent duty of a researcher. Given some initial state $|\Psi(0)\rangle$ and a Hamiltonian of interest $\hat{H}$, we seek future evolution states which satisfy

$$|\Psi(t)\rangle = \exp(-\mathrm{i}\, t\, \hat{H})\, |\Psi(0)\rangle\,, \tag{1.45}$$

as prescribed by the non-dimensionalised Schrödinger equation [106]. There is an enormous, mature literature on methods to utilise classical computers for this endeavour, though they are ultimately limited by the exponentially growing resources necessary to represent quantum states. Meanwhile, there is a young but growing literature on utilising quantum computers for this task which boast polynomially scaling quantum resources. Indeed, simulating quantum systems was Feynman's first proposed application of such a computer [1]. Arguably the most straightforward quantum algorithm for real-time simulation is *Trotterisation*, whereby we leverage the Suzuki–Trotter decompositions [107] of the unitary evolution operator seen in Equation 1.45. We assume our Hamiltonian is efficiently diagonalised in a basis of operators which are native or

tractable to effect on our quantum device - we here choose the Pauli basis,

$$\hat{H} \equiv \sum_{i}^{T} h_i \bigotimes_{j} \hat{\sigma}_{ij}, \qquad h_i \in \mathbb{R}, \qquad \hat{\sigma}_{ij} \in \{\mathbb{1}, \hat{X}, \hat{Y}, \hat{Z}\}. \tag{1.46}$$

Then, the first order Suzuki–Trotter decomposition of $r \in \mathbb{N}$ repetitions prescribes a series of smaller unitary operators

$$|\Psi(t)\rangle \approx \prod_{i}^{r} \prod_{i}^{T} \exp\left(-\mathrm{i}\, t\, h_i / r \bigotimes_{j} \hat{\sigma}_{ij}\right) |\Psi(0)\rangle\,, \tag{1.47}$$

with an accuracy informed by r and the commutators between the terms of $\hat{H}$. Notice each operator has the form of the Pauli gadget in Equation 1.21, and can hence likely be effected on a quantum computer through a parametrized gate, with parameter $\theta \sim t\, h_i / r$. A family of higher order decompositions exist which better reduce this commutation error, but which involve a greater total number of operators [108]:

$$\hat{S}[t, 1, 1] = \prod_{i}^{T} \exp\left(-\mathrm{i}\, t\, h_i \bigotimes_{j} \hat{\sigma}_{ij}/2\right) \prod_{T}^{i} \exp\left(-\mathrm{i}\, t\, h_i \bigotimes_{j} \hat{\sigma}_{ij}/2\right) \tag{1.48}$$

$$\hat{S}[t, n, r] = \prod^{r} \hat{S}[t/r, n, 1] \tag{1.49}$$

$$S[t, n, 1] = \left(\prod^{2} S[\mu\, t, n-2, 1]\right) S[(1-4\mu)t, n-2, 1] \left(\prod^{2} S[\mu\, t, n-2, 1]\right), \tag{1.50}$$

where $\hat{S}[t, n, r]$ is an n-th order (n is 1 or even) r-repetition approximation to the time-t unitary evolution operator. With a classically inaccessible approximation to the evolved state $|\Psi(t)\rangle$ now obtained in our quantum register, we then either perform a series of measurements of interest on the state, or feed it into another algorithm. While there is a growing repertoire of techniques to shrink the costs of Trotterisation [109] and improve its performance [110, 111] , it is generally regarded as incompatible with near-

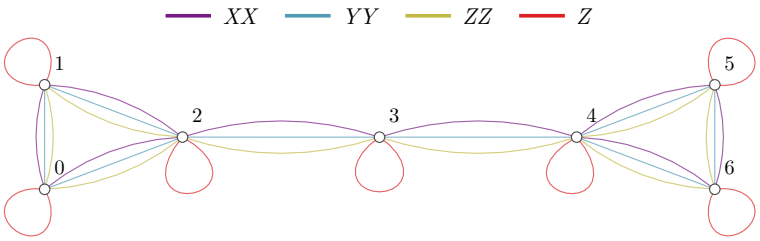


Figure 1.1: The qubit connectivity and their interactions suggested by the 7-spin Hamiltonian in Equation 1.51. Nodes represent qubits, and edges indicate the connected qubits interact. This diagram was generated with QuESTlink's `DrawCircuitTopology[]` function [112], as described in Chapter 7.

term devices due to its prescription of deep circuits involving precise gate parameters in order to achieve practical accuracies.

We offer now a quick sketch of a simulated implementation of Trotterisation. We use the 7-spin Hamiltonian considered in Reference [113], of the form

$$\hat{H} = \sum_q B_q \hat{Z}_q + \sum_{q_1 q_2} \sum_{\hat{\sigma} \in \hat{X}, \hat{Y}, \hat{Z}} J^{\sigma}_{q_1 q_2} \hat{\sigma}_{q_1} \hat{\sigma}_{q_2}, \qquad B_i < 0, \quad J^{\sigma}_{q_1 q_2} > 0, \tag{1.51}$$

where subscripts are qubit indices. The topology of this Hamiltonian is shown in Figure 1.1. For illustration, its first order Trotterisation (Equation 1.47) with a single repetition $r = 1$ is shown in Figure 1.2. We simulate evolution of the initial state $|\Psi(0)\rangle = |1\rangle\,|+\rangle^{\otimes 6}$ to fixed time $t = 2$ using a variety of Trotterisation orders and repetitions, comparing them to a direct matrix exponentiation of the Hamiltonian (Equation 1.45). The infidelities as functions of the total number of required Pauli gadget gates is shown in Figure 1.3. We also compare the pure evolution with *noisy* evolution determined by density matrix simulation of 1 and 2 qubit depolarising errors (of rates 10^{-4}) following every Trotter gate.

Even this modest 7-qubit 31-term Hamiltonian of nearest-neighbour two-qubit interactions required as many as 400 Trotter gates to achieve an algorithmic fidelity of 99%,

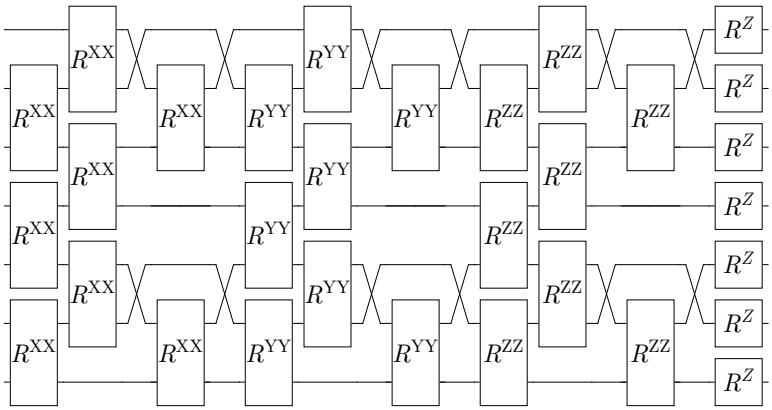


Figure 1.2: The first order single-repetition Suzuki–Trotter decomposition of the 7-spin Hamiltonian in Equation 1.1, as a circuit of one- and two-qubit Pauli gadgets of the form $R^{XX} = \exp(\mathrm{i}\,\beta\,\hat{X} \otimes \hat{X}), \quad \beta \in \mathbb{R}$ (similarly for R^{YY} and R^{ZZ}). Note that while these gates do not generally commute, the additive terms of the original Hamiltonian naturally do, hence the ultimate ordering of gates in the Trotter circuit is arbitrary. We have chosen a visually compact ordering. This diagram was generated with QuESTlink's `DrawCircuit[]` function [112].

which in the presence of modest noise, could not exceed 95%. The inaccuracies could not be circumvented by using more repetitions or higher order Trotterisations due to the increased decoherence, revealing a trade-off between algorithmic error (exponential truncation error) and gate error. We have demonstrated that canonical Trotterisation is ill-suited for near-future quantum devices. Fortunately, this does not preclude the use of such devices for the application of quantum simulation. Section 1.6.2 will introduce a method which utilises many samples of short depth circuits, considered a promising application of first generation quantum computers, and which will inspire much of the research of this thesis.

1.6 Variational algorithms

With fault-tolerant quantum computers out of the present picture, there is a growing appetite for algorithms compatible with NISQ devices. A prevalent NISQ architec-

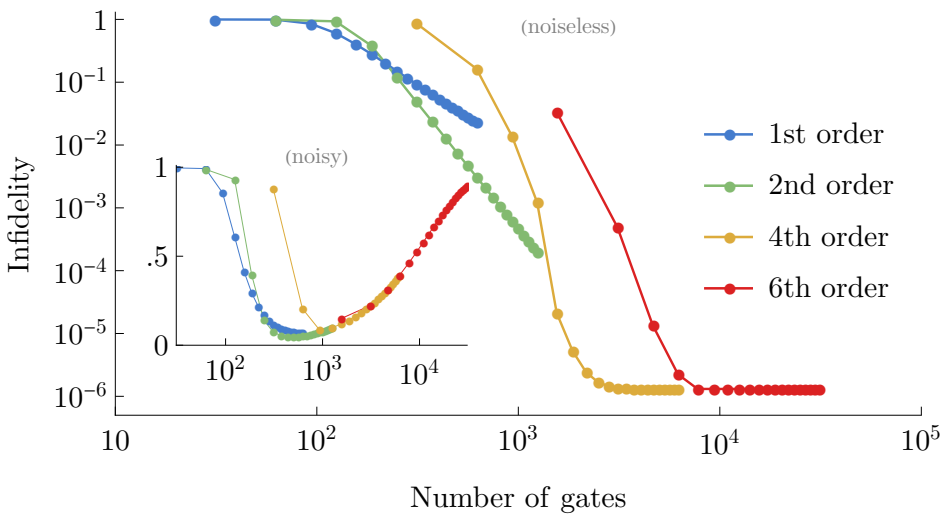


Figure 1.3: The infidelity $1 - |\langle\Psi(t)|\psi(t)\rangle|^2$ between the true evolution $|\Psi(t)\rangle$ and the approximation $|\psi(t)\rangle$ at $t = 2$ admitted by Trotterisation of the 7-spin dynamics of Hamiltonian 1.51. We show several orders (colours) and repetitions (dots) of Suzuki–Trotter decompositions [107] against the total number of gates they require (horizontal axis). The statevector simulations shown in the outer plot were noiseless (demonstrating only algorithmic, or Trotter truncation error), while the density matrix simulations of the inner plot assumed 1 and 2 qubit depolarising channels (error rate 10^{-4} per-gate) after every Trotter gate (demonstrating algorithmic *and* physical errors). Simulations used QuESTlink's `ApplyCircuit` and `CalcFidelity` facilities [112].

ture is the so-called hybrid computer whereby a classical processor works alongside a quantum co-processor. In a sense, the classical processor aims to reduce the computational burden of its quantum counterpart, which can then focus on only the classically intractable tasks like measuring quantum observables. If such tasks can be made to require only short depth circuits and tolerate imprecise operations, then they may be compatible with near-future imperfect hybrid classical-quantum computers.

Quantum variational algorithms are a family of techniques to leverage these hybrid devices. The classical computer maintains a tractable set of real parameters $\vec{\theta} = \{\theta_1, \theta_2, \dots\}$ which inform the quantum states $|\psi(\vec{\theta})\rangle$ produced and processed by the quantum computer. This is through an "ansatz" circuit $\hat{U}(\vec{\theta})$ acting on some (typically,

fixed) input state $|\text{in}\rangle$.

$$|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\ |\text{in}\rangle\,. \tag{1.52}$$

Such a circuit can be as simple as a sequence of $P \in \mathbb{N}$ separable uniquely singly-parameterised unitary gates

$$\hat{U}(\vec{\theta}) \ = \ \hat{U}_P(\vec{\theta}_P)\ \ldots\ \hat{U}_2(\theta_2)\ \hat{U}_1(\theta_1). \tag{1.53}$$

To be of any utility, we expect a classical description (e.g. $\vec{\psi} \in \mathbb{C}^{2^N}$) of its N-qubit output ansatz state $|\psi(\vec{\theta})\rangle \in \mathcal{H}_N$ to be intractable, and that the state has a reasonable sensitivity to each parameter θ_i. This is possible even when there are exponentially fewer parameters than that needed to generate the full $SU(2^N)$ space. Yet, we simultaneously desire that the ansatz submanifold of the full N-qubit Hilbert space includes states close to those of interest, although we will not know their parameters *a priori*. There is ergo a trade-off between ansatz *expressibility* (and the consequential accuracy of the algorithm output) and its *efficiency* (its consequential resource costs) [114].

In an iterative variational algorithm, measurements on $|\psi(\vec{\theta})\rangle$ build a corpus of information which is used by the classical computer to update $\vec{\theta}$, and hence change the ansatz state; the new state is studied afresh. This process is repeated such that the visited ansatz states $|\psi(\vec{\theta}_t)\rangle$ advance under some measure. For example, as approximations to a time-evolving physical system $|\psi(\vec{\theta}_t)\rangle \approx |\Psi(t)\rangle$, or toward time-independent states of interest like observable eigenstates, $\lim_{t\to\infty} |\psi(\vec{\theta}_t)\rangle \approx |e_0\rangle$. Many variational algorithms navigate some observable space. For example, given a Hamiltonian H under which it is sufficiently efficient to measure the energy of a quantum state, the ansatz circuit allows us to navigate a parameterised energy space $\langle E(\vec{\theta})\rangle$. We visualise this in Figure 1.4.

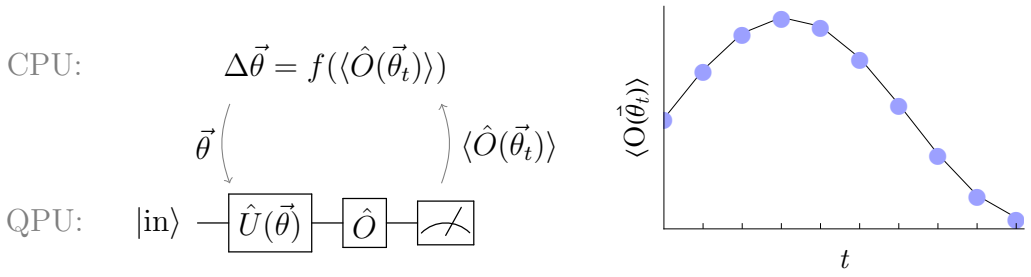


Figure 1.4: The iterative protocol of a hybrid quantum-classical computer running a variational algorithm. The quantum computer accepts a set of parameters $\vec{\theta}$, and measures an observable $\hat{O}$ of the ansatz state $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\text{in}\rangle$, which is used by the classical computer to determine the next generation of parameters.

Quantum variational algorithms are believed to be promising contenders for the first practical algorithms to exhibit quantum advantage. This is because they show some innate resilience to quantum errors and are often integrable with *ad hoc* error mitigation techniques [115, 116, 117]. Their general utilisation of relatively short quantum circuits, frequent classical feedback, and their estimations of quantities like $\langle E\rangle$ through repeated sampling imply their compatibility with imperfect quantum devices. Proving this property remains a challenge since variational quantum algorithms can be difficult to analyse, and the 'black box' treatment of the ansatz circuit can preclude analytic study. For these reasons, the expected runtime and resource scalings of proposed quantum variational algorithms are often empirically motivated through numerical simulation. Alas, this restricts us to the study of these algorithms in their *impotent* regimes whereby their classical alternatives are strictly faster. At present, we can only tenuously extrapolate how today's proposed quantum variational algorithms will perform in the classically intractable regime which defies simulation. And in many cases, we can instead only show they *won't* perform sufficiently well on even the modestly sized machines available today. Still, variational methods appear the best application for near-future quantum computers. We will now review three existing quantum variational algorithms of particular relevance to the original research in this thesis.

1.6.1 Quantum gradient descent

An early ubiquitous quantum variational algorithm, first appearing in Peruzzo *et al's* quantum variational eigensolver [118], is quantum gradient descent. We illustrate it here, since it serves as a simple minimisation technique for which we will develop an improved replacement later in this thesis.

In quantum gradient descent, we seek to find the parameters $\vec{\theta}_0$ which produce an ansatz state $|\psi(\vec{\theta}_0)\rangle$ for which some observable of interest is minimised. This is typically the energy $\langle E(\vec{\theta})\rangle$ under a Hamiltonian $\hat{H}$, such that

$$\langle E(\vec{\theta}_0)\rangle = \min_{\vec{\theta}} \ \langle E(\vec{\theta})\rangle = \min_{\vec{\theta}} \ \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle\,. \tag{1.54}$$

If our ansatz state $|\psi(\vec{\theta})\rangle$ is sufficiently well parametrized, we expect this quantity to lie close to the true ground-state of the Hamiltonian, i.e.

$$\langle E(\vec{\theta}_0)\rangle \approx e_0, \quad \text{where} \quad \hat{H} \equiv \sum_i e_i \,|e_i\rangle\,\langle e_i|\,, \quad e_0 = \min_i e_i. \tag{1.55}$$

Gradient descent is the simple prescription that our parameters should evolve opposite to the local gradient $\nabla\,\langle E(\vec{\theta})\rangle$ as

$$\dot{\vec{\theta}} = -\nabla\,\langle E(\vec{\theta})\rangle\,, \qquad (\text{notating } \dot{\theta}_i = \mathrm{d}\theta_i/\mathrm{d}t). \tag{1.56}$$

We iteratively update the ansatz parameters by repeatedly evaluating the energy gradient in parameter-space, and traversing in the opposite direction by an Euler step [119] of length Δt,

$$\Delta\vec{\theta} = -\nabla\,\langle E(\vec{\theta})\rangle\,\Delta t, \tag{1.57}$$

in the hope that $\vec{\theta} + \Delta\vec{\theta} \to \theta_0$. This is a simple quantum analogue of classical gradient descent; we are merely offloading evaluation of the cost function (energy) and its gradient to a quantum coprocessor, to avoid prohibitively expensive classical evaluation of the Hamiltonian.

Evaluating the energy gradient with a quantum computer can sometimes be as simple as evaluating the energy itself, although there are number of prescriptions in the literature. While we can always invoke a finite difference approximation,

$$\delta\theta \, \frac{\partial \langle E(\vec{\theta}) \rangle}{\partial \theta_i} \approx \langle E(\vec{\theta}|_{\theta_i \to \theta_i + \delta\theta}) \rangle - \langle E(\vec{\theta}) \rangle \, , \tag{1.58}$$

there are many scenarios conditioned upon the form of the ansatz circuit $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta}) \, |\text{in}\rangle$ where more accurate strategies are possible. For example, when the ansatz circuit is separable, $\hat{U}(\vec{\theta}) = \prod_i \hat{U}_i(\theta_i)$, and the i-th gate is a Pauli gadget of the form $\hat{U}_i(\theta_i) = \exp\left(\mathrm{i}\, \theta_i \, c \, \hat{\sigma}^{\otimes}\right)$ (a re-expression of Equation 1.21) then the derivative is given by the parameter-shift rule [120] precisely as

$$\frac{1}{c} \frac{\partial \langle E(\vec{\theta}) \rangle}{\partial \theta_i} = \langle E(\vec{\theta}|_{\theta_i \to \theta_i + \frac{\pi}{4c}}) \rangle - \langle E(\vec{\theta}|_{\theta_i \to \theta_i - \frac{\pi}{4c}}) \rangle \, . \tag{1.59}$$

We direct the interested reader to the concise proof offered in Reference [121], and note the many extensions of the parameter-shift rule for more general unitary gates [122]. To the best of our knowledge, there is yet no such rule for the general paramaterisation of a unitary gate given Equation 1.16, of the form $\hat{U}(\theta) = \exp(\mathrm{i} \sum_k h_k(\theta) \hat{G}_k)$, with general Hermitian generator basis $\{\hat{G}_k\}$. While noting that handling such a general case is almost always unnecessary,we will nonetheless see a more complicated treatment of this general gate in Chapter 2.

There are many heuristics which can be inherited from the classical optimisation lit-

erature to decide the initial states of quantum gradient descent, and to measure when convergence has been reached. Naturally these include randomising the initial parameters over the period of their corresponding gates, and stopping when the energy has observed to plateau. We caution however that the parameterized observable landscape often features large regions where $\Delta \langle E(\vec{\theta}) \rangle \ll 1$ referred to as"barren plateaus" [123, 124] which like local minima, can trap the ansatz evolution and imitate convergence, thwarting attempts to minimise. The research community has proposed a wealth of creative solutions to address this problem, including parameter initialisation strategies to at least initially avoid plateaus [125, 126], devising alternate costs functions without plateaus [127], quantum metropolis algorithms [128, 129] and through motivating ansätze designs [114, 130, 131]. We will later present a novel dynamic timestep protocol to address a related problem.

We now illustrate quantum gradient descent attempting to minimise the 7-spin Hamiltonian of Equation 1.51 (Figure 1.1). We employ the 56-parameter ansatz circuit shown in Figure 1.5 upon the arbitrary initial state $|\text{in}\rangle = |1\rangle |+\rangle^{\otimes 6}$ and plot the resulting parameter and energy evolution in Figure 1.6. Naturally, the routine is unsuccessful; in general, gradient descent is a poor minimisation routine in high dimensional settings with barren plateaus such as these.

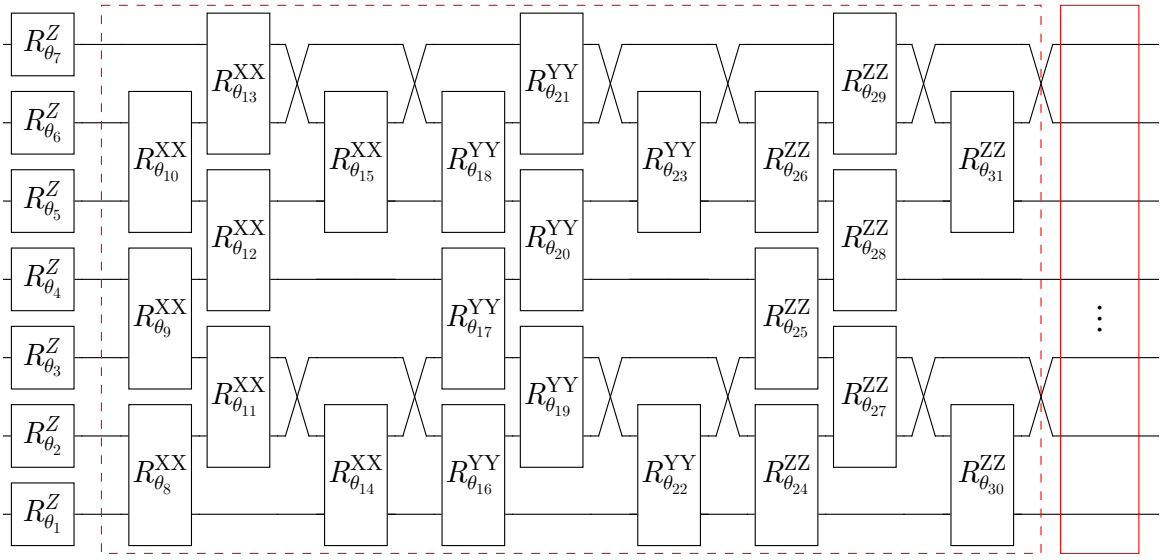


Figure 1.5: The 56-parameter ansatz circuit used in the example quantum gradient descent (and several other variational algorithms presented in this introductory chapter) of the 7-spin Hamiltonian given in Equation 1.51. The dashed outlined section is repeated twice albeit with unique parameters. Not drawn is the final unitary of the form $e^{i\theta_{56}}\hat{\mathbb{1}}$ which modulates the non-physical global phase. This is irrelevant to gradient descent, but is an important gate in Li's real-time simulation algorithm of the proceeding section, and explained in Chapter 2. Finally, we note the resemblence with the Trotter circuit of Figure 1.2 is no coincidence; we have premeditatedly chosen our ansatz to use the same gate set and topology as the Trotter–Suzuki decomposition, to simplify a comparison in a proceeding section.

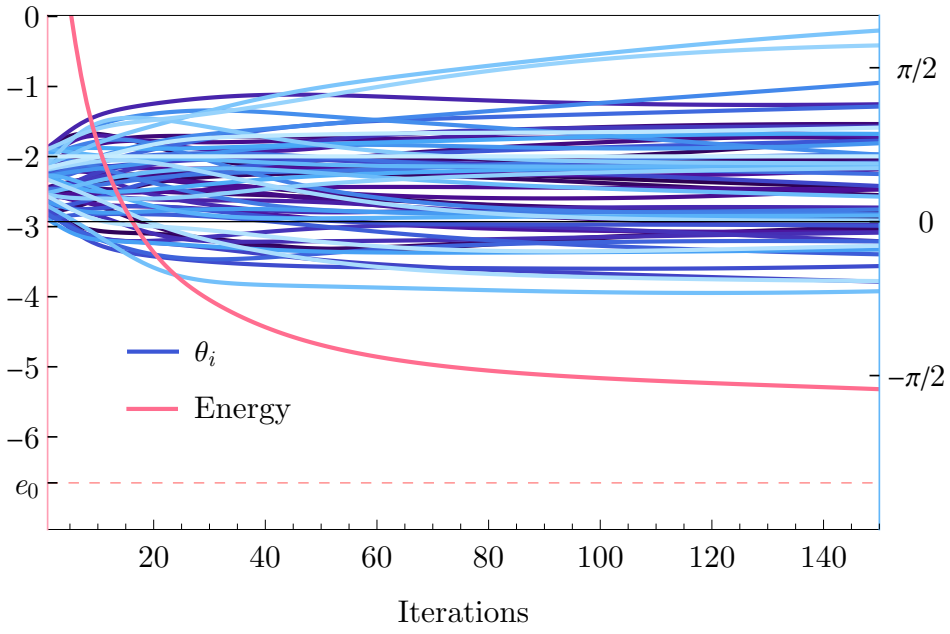


Figure 1.6: Quantum gradient descent unsuccessfully seeking the ground-state energy e_0 of the 7-spin Hamiltonian 1.51, using the 56-parameter ansatz of Figure 1.5, and starting from uniformly random parameters $\theta_i(0) \sim U_{[0,1]}$. The timestep was assigned its order of magnitude maximum stable (that which still exhibited monotonic energy decrease) at $\Delta t = 0.1$. Simulation was rapidly deployed using QuESTlink's `CalcQuregDerivs` function [112].

1.6.2 Quantum real-time simulation

This section briefly introduces an existing variational algorithm for real-time simulation, since it serves as the bedrock for our novel variational imaginary-time minimisation method presented in Chapter 2. We also use it to generate interesting input states to our recompilation algorithm from Chapter 4. This section contains no original research, although the simulations are my own.

This section introduces another existing variational quantum algorithm in the literature; Li's method for real-time simulation [132]. This is, in a sense, a variational alternative to the Trotter simulation reviewed in Section 1.5, and hence expected better suited to near-future quantum computers. While unrelated to the task of energy minimisation, we will see that the main quantum task of the previously introduced gradient descent will appear as a subroutine in Li's method.

We now briefly outline the derivation of Li's method. Note that we will make a rigorous derivation of our own variational algorithm in Section 2.4.1 which, with a small adjustment, will mirror a more substantial derivation of Li's method. Consider the evolution of a state $|\Psi(t)\rangle = \exp(-\mathrm{i}\hat{H}t)\,|\Psi(0)\rangle$ under a time-independent Hamiltonian $\hat{H}$, equivalently described by Lagrangian

$$L = \langle\Psi(t)|\left(\mathrm{i}\frac{\partial}{\partial t} - \hat{H}\right)|\Psi(t)\rangle\,. \tag{1.60}$$

We seek to approximate $|\Psi(t)\rangle$ at discrete intervals using ansatz states $|\psi(\vec{\theta}_t)\rangle$. We will denote the j-th parameter of the full set $\vec{\theta}_t$ as merely θ_j, neglecting its implicit time-dependence. Temporarily assuming that the ansatz can generate the true evolution

$|\Psi(t)\rangle$ precisely, the Langrangian can be expressed as

$$L = \mathrm{i}\,\langle\psi(\vec{\theta}_t)|\frac{\partial}{\partial t}|\psi(\vec{\theta}_t)\rangle \;-\; \langle\psi(\vec{\theta}_t)|\hat{H}|\psi(\vec{\theta}_t)\rangle \tag{1.61}$$

$$= \mathrm{i}\sum_j \langle\psi(\vec{\theta}_t)|\,\frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_j}\frac{\mathrm{d}\theta_j}{\mathrm{d}t} \;-\; \langle\psi(\vec{\theta}_t)|\hat{H}|\psi(\vec{\theta}_t)\rangle\,. \tag{1.62}$$

The prescribed Euler-Lagrange equations, in terms of the parameters θ_j and their time-derivatives $\dot{\theta}_j = \mathrm{d}\theta_j/\mathrm{d}t$, can be shown to imply parameter evolution

$$\frac{\partial L}{\partial\theta_k} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial\dot{\theta}_k}\right) \qquad\Longrightarrow\qquad \underline{\underline{\mathcal{A}}}\,\dot{\vec{\theta}}_t = -\vec{\mathcal{V}}, \tag{1.63}$$

where we have defined the tensors

$$\begin{aligned}\mathcal{A}_{jk} &= \Im\left[\frac{\partial\,\langle\psi(\vec{\theta}_t)|}{\partial\theta_j}\frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_k}\right],\\ \mathcal{V}_k &= \Re\left[\langle\psi(\vec{\theta}_t)|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_k}\right].\end{aligned} \tag{1.64}$$

By now relaxing the assumption that $|\psi(\vec{\theta}_t)\rangle$ is a complete parametrisation of $|\Psi(t)\rangle$, Li's method presents itself as a technique to approximate the true dynamics through an ansatz state $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\Psi(0)\rangle$ with a tractable number of classical parameters. This is by repeated quantum evaluation of quantities $\underline{\underline{\mathcal{A}}}$ and $\vec{\mathcal{V}}$, and classical solving of linear equation 1.63 to update the ansatz parameters $\vec{\theta}_t$ from their initial values $\vec{\theta}_0 \approx \vec{0}$. Despite their intimidating appearance, $\underline{\underline{\mathcal{A}}}$ and $\vec{\mathcal{V}}$ can be evaluated through a series of simple quantum circuits which feature the ansatz gates, their Hermitian generators, and an ancilla qubit. We defer discussion of these circuits until Chapter 2, when we will derive analogous circuits for our own variational method.

For illustration, we now include a classical emulation of Li's method simulating the 7-

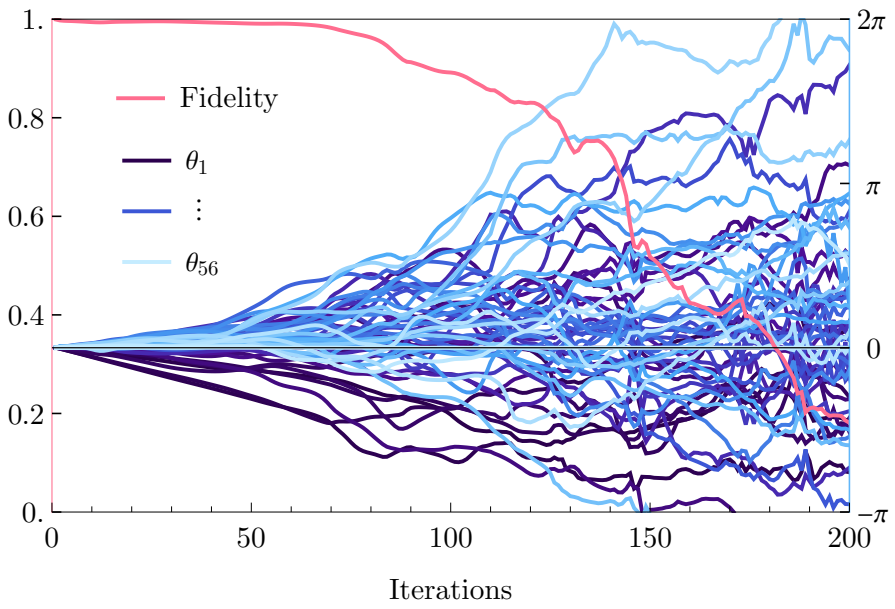


Figure 1.7: Li's variational method simulating the real-time dynamics of the 7-spin system with Hamiltonian 1.51, from initial state $|\Psi(0)\rangle = |1\rangle\,|+\rangle^{\otimes 6}$, using the 56-parameter ansatz $\hat{U}(\vec{\theta})$ of Figure 1.5. The fidelity compares $\hat{U}(\vec{\theta})\,|\Psi(0)\rangle$ (initially offset slightly by $\vec{\theta}_0 = 10^{-4}\vec{}$ to avoid a singularity) to the true evolution given by numerical exponentiation of the time evolution operator. The total time simulated is $t = 2$ ($\Delta t = 10^{-2}$), with a Tikhonov regularisation parameter of $\mu = 10^{-7}$ (explained in Chapter 2).

spin system of Equation 1.51 (with connectivity shown in Figure 1.1). This is the same system we simulated with Trotterisation in Section 1.5. We re-employ the 56-parameter ansatz circuit shown in Figure 1.5, which we previously used in quantum gradient descent. The evolution of the parameters, and the fidelity of the ansatz state against the true evolution (found through numerical matrix exponentiation of $-\mathrm{i}\,t\,\hat{H}$), is shown in Figure 1.7. For the interested reader, we also include a more direct comparison of Li's method against Trotterisation in Figure 1.8, real-time simulating the same system but with a larger ansatz of an identical structure to the Trotter circuit. It reveals that Li's method restores Trotterisation in the early time regime (when the variational parameters are linear and coincide with the Trotter gate strengths), but can deviate and outperform Trotterisation at later times.

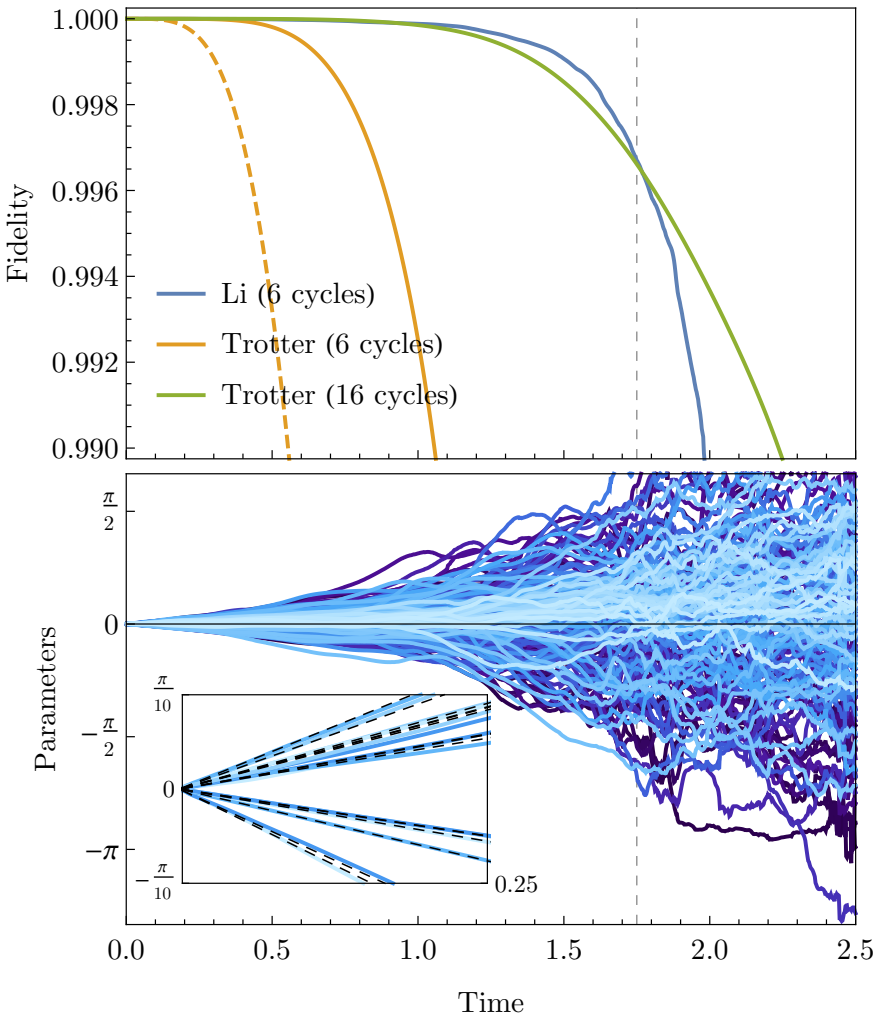


Figure 1.8: Li's variational method and Trotterisation simulating the real-time dynamics of the same 7-spin system as Figure 1.7. Li's method uses the 186-parameter ansatz formed by 6 repetitions of the first order Trotter circuit. The fidelity compares Li and Trotter methods to the true evolution given by numerical exponentiation of the time evolution operator.

1.6.3 Quantum natural gradient

Here we introduce another variational minimisation algorithm, quantum natural gradient [133], superior to gradient descent of Section 1.6.1. This method was developed subsequent to our novel imaginary-time minimisation technique presented in Chapter 2, and later realised to be equivalent. We summarise it here for completeness.

The quantum natural gradient is a quantum adaptation of the classical natural gradient [134], an improvement upon gradient descent which uses information about the local geometry of the cost function (and in the quantum case, of the parameterized observable manifold). The geometric tensor $\underline{\underline{G}}$ [135], of which the real component coincides with the Fubini-Study tensor [136, 137] and the classical Fisher information matrix [138], is used to update the parameters under

$$\Re\left[\underline{\underline{G}}\right]\,\dot{\vec{\theta}} = -\nabla\langle E\rangle\,, \tag{1.65}$$

$$\text{where}\quad G_{ij} = \frac{\partial\langle\psi|}{\partial\theta_i}\frac{\partial|\psi\rangle}{\partial\theta_j} - \frac{\partial\langle\psi|}{\partial\theta_i}|\psi\rangle\langle\psi|\frac{\partial|\psi\rangle}{\partial\theta_j}. \tag{1.66}$$

Here, $\nabla\langle E\rangle$ is the gradient of the parameterised energy $\langle E(\vec{\theta})\rangle = \langle\psi(\vec{\theta})|\hat{H}|\,|\psi(\vec{\theta})\rangle\rangle$ under the system Hamiltonian $\hat{H}$, as appeared in gradient descent of the previous chapter, and as measured by the quantum coprocessor. Interestingly, these variational equations feature terms similar to those appearing in Li's real-time simulation algorithm, though they here perform minimisation. We will demonstrate the origin of this similarity in Chapter 2. Provided that the real component of the geometric tensor can also be obtained through measurements on the quantum computer, then natural gradient can replace gradient descent in minimisation tasks for improved convergence. For illustration, we here repeat the minimisation task in Section 1.6.1 of the 7-spin Ising Hamiltonian given in Equation 1.51 (with connectivity in Figure 1.1). We once

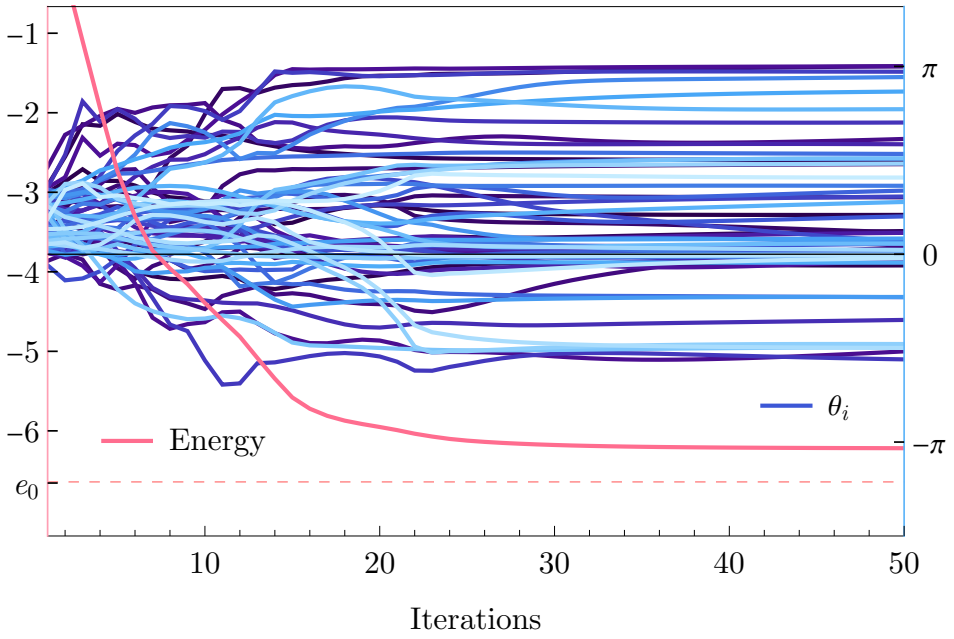


Figure 1.9: Quantum natural gradient approaching the ground-state energy e_0 of the 7-spin Hamiltonian 1.51, using the 56-parameter ansatz of Figure 1.5, and the same hyperparameters as used for gradient descent in Figure 1.6.

again employ the 56-parameter ansatz circuit shown in Figure 1.5 upon the arbitrary initial state $|\text{in}\rangle = |1\rangle\,|+\rangle^{\otimes 6}$, and plot the resulting parameter and energy evolution in Figure 1.9. Resource costs aside, we see quantum natural gradient outperform quantum gradient descent on the same problem, using the same hyperparameters.

Chapter 2

Quantum Imaginary-time Minimisation

Contents

2.1 Foreword

This chapter presents the published manuscript

- S. McArdle, **T. Jones**, S. Endo, Y. Li, S. C. Benjamin, X. Yuan
  Variational ansatz-based quantum simulation of imaginary time evolution
  npj Quantum Inf **5**, 75 (2019) (link)

which develops a novel quantum algorithm for variationally performing imaginary-time minimisation. The quantum component of this algorithm was devised by my co-authors, and its classical component by myself, extending that of its algorithmic predecessor. I also performed the quantum resource accounting and the small-scale analytic investigation, developed novel and bespoke classical simulation strategies, and performed all substantial numerical investigation. This thesis chapter will also contain a significant amount of original derivation, and all text and figures are my own.

2.2 Introduction

Finding the groundstate energy of a Hamiltonian is a common goal of quantum variational algorithms [118, 139, 140, 141, 142, 143, 144]. The task is central to many problems of quantum chemistry [145], but precludes practical classical optimisation due to the exponentially growing Hilbert space [1, 146]. There is growing excitement that variational minimisation of chemistry Hamiltonians will demonstrate quantum advantage; the faster solution of a practical problem with a quantum computer than possible of available classical resources [6]. But Section 1.6.1 already illustrated the performance barriers of simple quantum gradient descent [118], and at the time of the research in this chapter, the improved method of natural gradient [133] of Section 1.6.3 had not yet been developed.

Meanwhile, the classically-equipped quantum simulation community has developed a plethora of bespoke minimisation strategies. One fascinating method is that of *imaginary time evolution* [147, 148, 149] whereby tractably small quantum systems are classically simulated to evolve under the Wick rotated [150] Schrödinger equation. As we will formalise in Section 2.3, this drives the system toward solutions of the time-independent Schrödinger equation, i.e. low-lying energy eigenstates. Imaginary time evolution has broad utility in the study of finite temperature properties [151, 152, 153], in the modelling of solids [154], and for quelling excitations in simulated quantum turbulence [155, 156, 157], even in the author's previous research [158].

Simulation of imaginary-time evolution using a quantum computer may beget significant, potentially exponential speedup [159]. Alas, the imaginary-time propogator $\exp(-\hat{H}t)$ is non-unitary, precluding the use of Trotterisation introduced in Section 1.5. However, one may notice that the variational real-time simulation technique of Sec-

tion 1.6.2 did not leverage unitarity of the modelled evolution. If a similar technique is possible to variationally simulate imaginary-time evolution, then it may serve as a superior minimisation technique in quantum variational algorithms.

This chapter presents a novel algorithm which in a sense combines Li's variational algorithm [160] with imaginary-time simulation, to variationally simulate the latter. We utilise this scheme for cost minimisation in quantum chemistry settings as a superior performing alternative to quantum gradient descent. To investigate the performance of our method, we develop bespoke classical simulation strategies for embedding noise in variational routines. We also explore the classical matrix inversion component of Li's algorithm, and motivate the use of regularisation. Because our algorithm uses low-depth variational circuits compatible with noisy quantum-classical computers, it suggests an exciting application of noisy intermediate-scale quantum hardware [6] and a step toward achieving quantum advantage.

2.3 Motivation

Here we mathematically motivate the potential utility of variational imaginary-time simulation on a quantum computer. Consider the real-time propagation under Hamiltonian $\hat{H}$ of an arbitrary quantum state $|\Psi(0)\rangle$ with amplitudes α_i in the energy eigenstate basis $|e_i\rangle$ (with eigenvalues e_i). The spectral theorem [161] admits

$$|\Psi(t)\rangle = \exp(-\mathrm{i}\, t\, \hat{H})\, |\Psi(0)\rangle = \sum_i \alpha_i\, \exp(-\mathrm{i}\, t\, e_i)\, |e_i\rangle\,. \tag{2.1}$$

Imaginary time evolution refers to the propagation produced by Wick rotating (substituting $t = -\mathrm{i}\tau$, where $\tau \in \mathbb{R}$) this equation to produce

$$|\Psi(\tau)\rangle = \exp(-\tau \hat{H})\,|\Psi(0)\rangle = \sum_i \alpha_i \,\exp(-\tau\, e_i)\,|e_i\rangle\,, \tag{2.2}$$

though we stress $|\Psi(\tau)\rangle$ is *not* an L^2 vector despite its ket notation. This form prescribes a non-unitary evolution whereby the probability of an eigenstate

$$|\langle e_i|\Psi(\tau)\rangle|^2 = |\alpha_i|^2 \exp(-2\, e_i\, \tau), \tag{2.3}$$

vanishes exponentially at a rate proportional to the energy eigenvalue. Ergo when driving $\tau \to \infty$ or "evolving in imaginary time", a renormalisation of $|\Psi(\tau)\rangle$ rapidly and deterministically approaches the minimum energy eigenstate, assuming it began with a non-zero probability.

$$\lim_{\tau\to\infty} \frac{|\Psi(\tau)\rangle}{\|\,|\Psi(\tau)\rangle\,\|} \sim |e_0\rangle\,, \quad \text{assuming } \; |\alpha_0| > 0. \tag{2.4}$$

This makes imaginary time evolution an excellent minimisation scheme for discovering ground states, provided the action of the imaginary-time propagator $\exp(-\tau\hat{H})$ upon an arbitrary state can be efficiently approximated. Indeed in many classical simulation settings, such evolution can be effected by avoiding the relatively expensive exponentiation of a dense $\hat{H}$ matrix representation, and instead through Wick rotations of numerical solvers of the time-dependent Schrödinger equation. Still, the classical simulation must maintain an exponentially costly description of the evolving state.

It is principally possible to formulate a variational scheme whereby an evolution of ansatz parameters $\vec{\theta}_\tau$ prescribe a projection of imaginary-time evolution into the L^2

parameterised manifold generated by an ansatz state $|\psi(\vec{\theta}_\tau)\rangle$. Such a scheme would yield all the benefits of variational simulation, such as reduced quantum resource costs over direct evolution schemes and compatibility with near-term quantum devices. We derive such a scheme in the following section.

2.4 Derivation

The below derivations are my own, although the results of Sections 2.4.1 and 2.4.2 are known and published by my co-authors. The latter is also an alternate derivation to the Langrangian method of Reference [160].

2.4.1 Parameter evolution

We here derive the variational equations which inform how to update the parameters $\vec{\theta}_\tau$ of our ansatz state $|\psi(\vec{\theta}_\tau)\rangle$ in order to evolve the state as prescribed by imaginary-time evolution. This derivation will admit a similar form as Li's method in Section 1.6.2, albeit under a Wick rotation $t \to -\mathrm{i}\,\tau$.

The real-time Schrödinger propogator under a Hamiltonian $\hat{H}$ can be approximated to first-order as

$$|\Psi(t+\delta t)\rangle = \exp(-\mathrm{i}\,\delta t\,\hat{H})\,|\Psi(t)\rangle \tag{2.5}$$

$$\approx (1-\mathrm{i}\,\delta t\,\hat{H})\,|\Psi(t)\rangle\,. \tag{2.6}$$

Assume that the true state at time t can be represented exactly through our ansatz state, $|\Psi(t)\rangle \equiv |\psi(\vec{\theta}_t)\rangle$. This suggests the optimum choice of the "next" parameters

$\vec{\theta}_{t+\delta t}$ are those which best approximate

$$|\psi(\vec{\theta}_{t+\delta t})\rangle \approx |\psi(\vec{\theta}_t)\rangle - \mathrm{i}\,\delta t\,\hat{H}\,|\psi(\vec{\theta}_t)\rangle\,. \tag{2.7}$$

Along with the full derivative, the calculus of variations informs us (notating $\dot{\theta}_i = \mathrm{d}\theta_i/\mathrm{d}t$) that

$$|\psi(\vec{\theta}_{t+\delta t})\rangle \approx |\psi(\vec{\theta}_t)\rangle + \delta\,|\psi(\vec{\theta}_t)\rangle \tag{2.8}$$

$$= |\psi(\vec{\theta}_t)\rangle + \sum_i \frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_i}\;\dot{\theta}_i\;\delta t. \tag{2.9}$$

Developing a method of variational simulation of the Schrödinger equation then reduces to formulating a gradient of the parameters $\dot{\vec{\theta}}$ which well approximate

$$-\mathrm{i}\,\hat{H}\,|\psi(\vec{\theta}_t)\rangle \approx \sum_i \frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_i}\;\dot{\theta}_i. \tag{2.10}$$

McLachlan's variational principle [162] motivates $\dot{\vec{\theta}}$ by minimising the norm of the difference (a vector) of the left and right hand sides,

$$\min_{\delta\vec{\theta}} \left\| \mathrm{i}\,\hat{H}\,|\psi(\vec{\theta}_t)\rangle + \sum_i \frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial\theta_i}\,\dot{\theta}_i \right\| \tag{2.11}$$

at which the variation must be zero;

$$\implies \delta \left\| \mathrm{i}\,\hat{H}\,|\psi(\vec{\theta}_t)\rangle + \frac{\partial\,|\psi(\vec{\theta}_t)\rangle}{\partial t} \right\| = 0. \tag{2.12}$$

We have temporarily restored the explicit time derivative in order to Wick rotate. By now expanding $\|\,|\Psi\rangle\,\| = \sqrt{\langle\Psi|\Psi\rangle}$, we would arrive at the same equations prescribed by Li's method for real-time simulation. Instead, we first perform a Wick rotation, substi-

tuting $t = -\mathrm{i}\tau$ (where $\tau \in \mathbb{R}$) so that we now track the evolution of the system $\psi(\vec{\theta_\tau})$ in "imaginary time". McLachlan's variational principle then takes the form (noting the i coefficient of $\hat{H}$ has cancelled)

$$0 = \delta \left\| \hat{H} \left|\psi(\vec{\theta_\tau})\right\rangle + \frac{\partial \left|\psi(\vec{\theta_\tau})\right\rangle}{\partial \tau} \right\| = \delta\, d(\vec{\theta_\tau}), \tag{2.13}$$

where we have defined $d(\vec{\theta_\tau})$ to be the norm, which we can expand (simplifying via $\langle\Psi|\phi\rangle + \langle\phi|\Psi\rangle = 2\,\Re\,\langle\Psi|\phi\rangle$ and $\hat{H} = \hat{H}^\dagger$) as

$$d(\vec{\theta_\tau}) = \sqrt{\langle\psi(\vec{\theta_\tau})|\hat{H}^\dagger\hat{H}|\psi(\vec{\theta_\tau})\rangle + 2\,\Re\left[\frac{\partial\,\langle\psi(\vec{\theta_\tau})|}{\partial\tau}\hat{H}\,|\psi(\vec{\theta_\tau})\rangle\right] + \frac{\partial\,\langle\psi(\vec{\theta_\tau})|}{\partial\tau}\frac{\partial\,|\psi(\vec{\theta_\tau})\rangle}{\partial\tau}} \tag{2.14}$$

$$= \sqrt{\eta(\vec{\theta_\tau})}, \tag{2.15}$$

defining $\eta(\vec{\theta_\tau})$ for concision. Returning our time derivatives to explicit dependence on our ansatz parameters, we have

$$\eta(\vec{\theta_\tau}) = \langle\psi(\vec{\theta_\tau})|\hat{H}^\dagger\hat{H}|\psi(\vec{\theta_\tau})\rangle + 2\sum_j \dot{\theta}_j\,\Re\left[\langle\psi(\vec{\theta_\tau})|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta_\tau})\rangle}{\partial\theta_j}\right] + \sum_{i,j}\dot{\theta}_i\dot{\theta}_j\frac{\partial\,\langle\psi(\vec{\theta_\tau})|}{\partial\theta_i}\frac{\partial\,|\psi(\vec{\theta_\tau})\rangle}{\partial\theta_j}. \tag{2.16}$$

By treating $\delta\theta_i$ and $\delta\dot{\theta}_i$ as independent differentials, McLachlan's variational principle becomes

$$0 = \delta d(\vec{\theta_\tau}) = \sum_i \frac{\partial d(\vec{\theta_\tau})}{\partial\theta_i}\delta\theta_i + \sum_i \frac{\partial d(\vec{\theta_\tau})}{\partial\dot{\theta}_i}\delta\dot{\theta}_i, \tag{2.17}$$

which by independence between all parameters, can be separated into simultaneously

satisfied

$$0 = \frac{\partial d(\vec{\theta}_\tau)}{\partial \theta_i} \quad \text{and} \quad 0 = \frac{\partial d(\vec{\theta}_\tau)}{\partial \dot{\theta}_i} \quad \forall\, i. \tag{2.18}$$

It will turn out sufficient to use only the rightmost constraint, invoking the chain-rule to express

$$0 = \frac{\partial d(\vec{\theta}_\tau)}{\partial \dot{\theta}_i} = \frac{1}{2} \frac{1}{d(\vec{\theta}_\tau)} \frac{\partial \eta(\vec{\theta}_\tau)}{\partial \dot{\theta}_i}, \tag{2.19}$$

which, since $d(\vec{\theta}_\tau) > 0$, implies

$$0 = \frac{\partial \eta(\vec{\theta}_\tau)}{\partial \dot{\theta}_i} = 2\,\Re\left[\langle\psi(\vec{\theta}_\tau)|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial \theta_i}\right] + 2\,\Re\left[\sum_j \dot{\theta}_j \frac{\partial\,\langle\psi(\vec{\theta}_\tau)|}{\partial \theta_j}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial \theta_i}\right]. \tag{2.20}$$

We finally rearrange these equations over i to the form

$$\underline{\underline{\mathcal{M}}}\,\dot{\vec{\theta}} = -\vec{\mathcal{V}}, \tag{2.21}$$

$$\mathcal{M}_{ij} = \Re\left[\frac{\partial\,\langle\psi(\vec{\theta}_\tau)|}{\partial \theta_i}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial \theta_j}\right], \tag{2.22}$$

$$\mathcal{V}_i = \Re\left[\langle\psi(\vec{\theta}_\tau)|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial \theta_i}\right]. \tag{2.23}$$

Equation 2.21 is an imaginary-time adaptation of Equation 1.63, the variational equation utilised in Li's real-time simulation algorithm. They differ only by the complex component featured in $\underline{\underline{\mathcal{M}}}$, here using the real component over the imaginary component used in Li's. There is also an interesting parallel between Equation 2.21 and the constraint on the parameter evolution imposed by gradient descent, Equation 1.56.

Observe the gradient satisfies

$$\nabla \langle E(\vec{\theta}) \rangle_i = \frac{\partial}{\partial \theta_i} \langle \psi(\vec{\theta}) | \hat{H} | \psi(\vec{\theta}) \rangle = 2 \, \Re \left[\langle \psi(\vec{\theta}) | \, \hat{H} \frac{\partial \, | \psi(\vec{\theta}) \rangle}{\partial \theta_i} \right] = 2 \, \mathcal{V}_i. \tag{2.24}$$

Since the factor 2 would be absorbed into the timestep hyperparameter when numerically performing gradient descent, we can express its parameter evolution as

$$\dot{\vec{\theta}} = -\vec{\mathcal{V}}. \tag{2.25}$$

This suggests that variational imaginary time evolution could be understood as gradient descent with the introduction of the coefficient matrix $\underline{\underline{\mathcal{M}}}$ which captures information about the parametrized ansatz-accessible Hilbert manifold not embedded in the energy manifold. It also clarifies their relative resource costs, as quantities $\underline{\underline{\mathcal{M}}}$ must be experimentally evaluated - we will later see at a modest cost. In fact, we will argue in Section 2.7 that imaginary-time reduces to gradient descent as the quantum evaluation of $\underline{\underline{\mathcal{M}}}$ is subject to increasing noise. Finally, we note the close resemblance with the subsequently developed quantum natural gradient technique introduced in 1.6.3; we will show explicitly in Section 2.4.5 that these algorithms are equivalent.

Although not proven here, positivity of $\underline{\underline{\mathcal{M}}}$ implies $\underline{\underline{\mathcal{M}}}^{-1} \vec{\mathcal{V}}$ is non-negative, and hence that the energy decreases under $\delta \vec{\theta}$, much like under gradient descent (see the supplementary information of Reference [163]). This means variational imaginary-time evolution prescribes a monotonically decreasing energy $\langle \psi(\vec{\theta}_\tau) | \hat{H} | \psi(\vec{\theta}_\tau) \rangle$ when the timestep $\Delta\tau \approx \delta\tau$ is sufficiently small. This will prove a useful property to gauge the stability under other hyperparameters and additional constraints beyond the variational principle.

2.4.2 Quantum subroutine

We now derive the quantum circuits which can experimentally evaluate the quantities $\underline{\underline{\mathcal{M}}}$ and $\vec{V}$ of Equation 2.21 at each step of variational imaginary time evolution. This is via an almost identical process to that of Li's method in Section 1.6.2 - indeed the prescription to evaluate $\vec{\mathcal{V}}$ is unchanged.

Assume our parametrized state $|\psi(\vec{\theta})\rangle$ is generated by an ansatz circuit $\hat{U}(\vec{\theta})$ of P parameters upon a fixed input state $|\text{in}\rangle$, of the form

$$\hat{U}(\vec{\theta}) = \prod_{k=P}^{1} \hat{U}_k(\theta_k), \qquad |\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\text{in}\rangle\,, \tag{2.26}$$

where each $\hat{U}_k(\theta_k)$ is a single-parameter unitary gate acting upon some (undisclosed in our notation) set of qubits, generated by some Hermitian operator expressible in the Pauli basis as

$$\hat{U}_k(\theta_k) \equiv \exp\left(\mathrm{i}\sum_l \lambda_{lk}(\theta_k)\,\vec{\sigma}_{lk}^{\otimes}\right), \quad \vec{\sigma}_{lk}^{\otimes} = \bigotimes_m \hat{\sigma}_{mlk}, \qquad \lambda_{lk}(\theta) : \mathbb{R} \mapsto \mathbb{R}.$$

(re-expression of 1.17)

Besides that our ansatz circuit is separable in the parameters, this is so far a general prescription. For concision, we adopt the notation

$$\hat{U}_{m:n} = \prod_{k=n}^{m} \hat{U}_k(\theta_k), \qquad n > m, \tag{2.27}$$

to indicate the subcircuit consisting of all gates between m and n inclusive. A partial

derivative of our ansatz state is now expressible as

$$\frac{\partial \left|\psi(\vec{\theta})\right\rangle}{\partial \theta_j} = \hat{U}_{j+1:P} \, \frac{\mathrm{d}\hat{U}_j(\theta_j)}{\mathrm{d}\theta_j} \, \hat{U}_{1:j-1} \tag{2.28}$$

and a gate derivative, as the skew-Hermitian but generally non-unitary operator

$$\frac{\mathrm{d}\hat{U}_j(\theta_j)}{\mathrm{d}\theta_j} = \mathrm{i} \sum_l {\lambda_{lj}}'(\theta_j) \; \vec{\sigma}_{lj}^{\otimes} \;\; \hat{U}_j(\theta_j). \tag{2.29}$$

Note $\vec{\sigma}_{lj}^{\otimes}$ and $\hat{U}_j(\theta_j)$ commute so their relative ordering is unimportant. An element satisfying $i < j$ of $\underline{\underline{\mathcal{M}}}$ expands to

$$\begin{aligned} \mathcal{M}_{ij} &= \Re\left[\frac{\partial \left\langle\psi(\vec{\theta}_\tau)\right|}{\partial \theta_i} \frac{\partial \left|\psi(\vec{\theta}_\tau)\right\rangle}{\partial \theta_j}\right] && \text{(reiteration of 2.22)} \\ &= \sum_{l_1, l_2} \lambda_{l_1 i}{}'(\theta_i) \, \lambda_{l_2 j}{}'(\theta_j) \;\; \Re\left[\langle \mathrm{in}| \;\; U_{1:i-1}^{\dagger} \;\; \vec{\sigma}_{l_1 i}^{\otimes} \;\; \hat{U}_{i:P}^{\dagger} \;\; \hat{U}_{j:P} \;\; \vec{\sigma}_{l_2 j}^{\otimes} \;\; \hat{U}_{1:j-1} \;\; |\mathrm{in}\rangle\right] && (2.30) \end{aligned}$$

where $\hat{U}_{i:P}^{\dagger} \, \hat{U}_{j:P}$ simplifies to $\hat{U}_{i:j-1}^{\dagger}$. Since $\underline{\underline{\mathcal{M}}}$ is symmetric, $\mathcal{M}_{ji}$ has the same form, as do the diagonals $\mathcal{M}_{ii}$ (in which case $\hat{U}_{i:j-1}^{\dagger} = \hat{\mathbb{1}}$). We point out that in the exceptionally common case of a Pauli gadget $\hat{U}_k(\theta_k) \equiv \exp(\mathrm{i}\,\theta_k \, c_k \, \vec{\sigma}_k^{\otimes})$, where a gate is a simple rotation about a single Pauli tensor (and $\lambda_k(\theta_k) = \theta_k c_k$ for some fixed $c_k \in \mathbb{R}$), idempotency of the Paulis means the diagonals are known and fixed as $\mathcal{M}_{ii} = {c_k}^2$ for all iterations. Meanwhile, the gradient vector $\vec{\mathcal{V}}$ has elements

$$\begin{aligned} \mathcal{V}_i &= \Re\left[\left\langle\psi(\vec{\theta}_\tau)\right| \hat{H} \frac{\partial \left|\psi(\vec{\theta}_\tau)\right\rangle}{\partial \theta_i}\right] && \text{(reiteration of 2.23)} \\ &= \sum_{l_1} \lambda_{l_1 i}{}'(\theta_i) \; \Re\left[\mathrm{i} \;\; \langle \mathrm{in}| \;\; \hat{U}_{1:P}^{\dagger} \;\; \hat{H} \;\; \hat{U}_{i:P} \;\; \vec{\sigma}_{l_1 i}^{\otimes} \;\; \hat{U}_{1:i-1} \;\; |\mathrm{in}\rangle\right], && (2.31) \end{aligned}$$

where i can be cancelled by changing the complex component. Assume our problem Hamiltonian $\hat{H}$ is tractable in a unitary basis - we are likely to employ the Pauli basis,

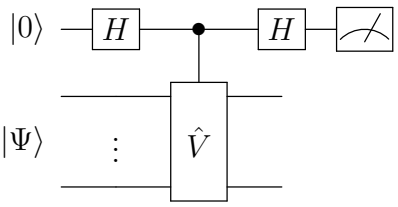


Figure 2.1: The generic Hadamard test [164] which samples the expected value of the ancilla (top most) qubit of $\langle Z \rangle = \Re[\langle\psi|\hat{V}|\psi\rangle]$, for some unitary $\hat{V}$. The quantity $\Im[\langle\psi|\hat{V}|\psi\rangle]$ is evaluated in the same fashion, albeit with the introduction of an ancilla $\hat{S} = \exp(\mathrm{i}\pi\hat{Z}/4)$ gate before the final Hadamard.

whereby

$$\hat{H} = \sum_l h_l \, \vec{\sigma}_l^{\otimes}, \qquad h_l \in \mathbb{R}, \tag{2.32}$$

and such that $\vec{\mathcal{V}}$ expands to

$$\mathcal{V}_i = -\sum_{l_1,l_3} \lambda_{l_1 i}{}'(\theta_i) \; h_{l_3} \; \Im\left[\langle \mathrm{in}| \;\; \hat{U}_{1:P}^{\dagger} \;\; \vec{\sigma}_{l_3}^{\otimes} \;\; \hat{U}_{i:P} \;\; \vec{\sigma}_{l_1 i}^{\otimes} \;\; \hat{U}_{1:i-1} \;\; |\mathrm{in}\rangle\right]. \tag{2.33}$$

Each inner product within $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ now contain only unitary operators, and as such, can be evaluated through repeated sampling on a quantum computer. We naturally assume that if unitary ansatz gate $\hat{U}_k(\theta_k)$ can be performed natively, so too can its inverse $\hat{U}_k(\theta_k)^{\dagger}$, which itself is often as simple as its negation $\hat{U}_k(-\theta_k)$. We opt to employ the Hadamard test [164], for which a simple circuit to evaluate a complex component of $\langle\Psi|\hat{V}|\Psi\rangle$ is shown in Figure 2.1. In our application, the input quantum state $|\Psi\rangle$ and operator of interest $\hat{V}$ for terms of $\mathcal{M}_{ij}$ and $\vec{V}_i$ are

$$\begin{array}{cc} \text{for } \mathcal{M}_{ij} & \text{for } \mathcal{V}_i \\ |\Psi\rangle = \hat{U}_{1:i-1}\,|\mathrm{in}\rangle & |\Psi\rangle = \hat{U}_{1:i-1}\,|\mathrm{in}\rangle \\ \hat{V} = \vec{\sigma}_{l_1 i}^{\otimes} \;\; \hat{U}_{i:j-1}^{\dagger} \;\; \vec{\sigma}_{l_2,j}^{\otimes} \;\; \hat{U}_{i:j-1} & \hat{V} = \hat{U}_{i:P}^{\dagger} \;\; \vec{\sigma}_{l_3}^{\otimes} \;\; \hat{U}_{i:P} \;\; \vec{\sigma}_{l_1,i}^{\otimes}. \end{array} \tag{2.34}$$

The Hadamard test requires effecting an ancilla-controlled form of $\hat{V}$ onto ansatz register $|\Psi\rangle$, which may appear prohibitively expensive. Fortunately our $\hat{V}$ operators feature the adjoint $\hat{U}_k^\dagger$ of every contained ancilla gate $\hat{U}_k(\theta_k)$. Since each contribute opposite phase changes upon the ancilla, their contributions cancel, and the gates can instead be effected non-controlled. In addition, any of these gates performed on the ansatz register *after* the second Pauli tensor will also not determine the ancilla phase, and need not be performed; this makes $\hat{U}_{i:P}^\dagger$ in $\hat{V}$ redundant. Only the Pauli tensors need to be controlled, where each operator can be applied in turn as a single-qubit controlled Pauli gate.

$$C_{\text{ancilla}}\left[\vec{\sigma}^{\otimes}\right] = \prod_m C_{\text{ancilla}}\left[\hat{\sigma}_m\right]. \tag{2.35}$$

Ergo our application of the Hadamard test requires only *two* controlled n-qubit Pauli tensors per evaluated inner product, each of which can be expanded into n controlled single-qubit Pauli gates. We illustrate the consequential quantum circuits to evaluate $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ in Figure 2.2. At every iteration of variational imaginary time evolution, an experimentalist must use these circuits to evaluate these quantities as functions of the current parameters $\vec{\theta}_\tau$. We note that subsequent works removed the need for an ancilla qubit and the expected relatively cumbersome controlled Pauli gates, through destructive measurements directly upon the ansatz register [165]. In principle however, our non-destructive measurements through the Hadamard test permit the incorporation of error mitigation strategies, explored in subsequent works.

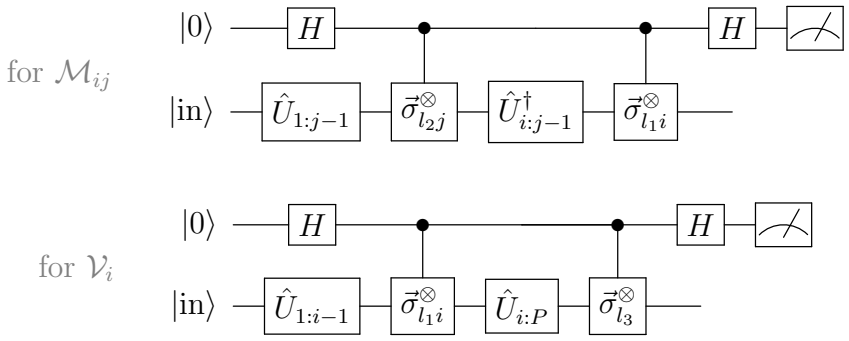


Figure 2.2: The circuits to evaluate the (l_1, l_2)-th real inner product of $\mathcal{M}_{ij}$ (Equ 2.30), and the (l_1, l_3)-th imaginary inner product of $\mathcal{V}_i$ (Equ 2.33), through repeated measurement to determine $\langle \hat{Z} \rangle$ of the ancilla qubit. This is an optimised adaptation of the Hadamard test [164] shown in Fig. 2.1, as employed by Li and Benjamin in their real-time variational simulator [132], from which our imaginary-time routine is inspired.

2.4.3 Quantum resources

We now sketch the quantum resource costs of experimentally implementing one iteration of variational imaginary-time evolution. This is the number of quantum gates and measurements required to populate $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$. Evaluating $\mathcal{M}_{ij}$ (and $\mathcal{V}_i$) requires repeating the Hadamard test circuits in Figure 2.2, approximating each inner product within Equations 2.30 (and 2.33) in-turn as the average of $m_{\mathcal{M}}$ (and $m_{\mathcal{V}}$) measurements on the ancilla qubit. A new measurement requires repeating the full Hadamard test and every gate therein.

We first study $\underline{\underline{\mathcal{M}}}$. Recall that each ansatz gate can be generated by a Pauli string of L terms upon D qubits. That is

$$\hat{U}_k(\theta_k) = \exp\left(\mathrm{i} \sum_{l}^{L} \lambda_{lk}(\theta_k) \bigotimes_{q}^{D} \hat{\sigma}_{qlk} \right). \qquad \text{(re-expression of 1.17)}$$

This means each $\mathcal{M}_{i<j}$ is composed of L^2 inner products, each one evaluated through $m_{\mathcal{M}}$ invocations of its corresponding Hadamard test. Equations 2.34 and 2.35 reveal a

single invocation requires $2(j-1)$ ansatz gates and $2D$ controlled (single-target) Pauli gates, in addition to the 2 Hadamards and 1 measurement upon the ancilla. Diagonal $\mathcal{M}_{ii}$ feature $L^2 - L$ inner products (avoiding those for which $l_1 = l_2$, which yield *a priori* known norms), each of which prescribe $m_{\mathcal{M}}$ repetitions of a Hadamard test with $i-1$ ansatz gates, $2D$ controlled Paulis, 2 Hadamards and 1 measurement. In total, evaluating $\underline{\underline{\mathcal{M}}}$ requires

$$\frac{1}{2} m_{\mathcal{M}} \, L \, P (L \, P + L - 2) = \mathcal{O}(m_{\mathcal{M}} L^2 P^2) \quad \text{measurements,} \tag{2.36}$$

$$\frac{1}{6} \, m_{\mathcal{M}} \, L \, P (P-1)(4 \, L \, P + L - 3) = \mathcal{O}(m_{\mathcal{M}} L^2 P^3) \quad \text{ansatz gates,} \tag{2.37}$$

$$m_{\mathcal{M}} L \, D \, P (L \, P + L - 2) = \mathcal{O}(m_{\mathcal{M}} D L^2 P^2) \quad \text{controlled Paulis,} \tag{2.38}$$

in addition to twice as many ancilla Hadamards as measurements.

We now consider $\vec{\mathcal{V}}$. Let T be the number of Pauli tensors in our Hamiltonian, each of which target at most N qubits. That is,

$$\hat{H} = \sum_{l}^{T} h_l \bigotimes_{q}^{N} \sigma_{ql}. \tag{re-expression of 2.32}$$

Equation 2.33 informs us that evaluating element $\mathcal{V}_i$ requires $L \, T$ invocations of the imaginary-component variant of the Hadamard test. And Equations 2.34 and 2.35 reveal each of $m_{\mathcal{V}}$ invocations involve P ansatz gates, $N + D$ controlled Paulis, in addition to 2 Hadamards, 1 $\hat{S}$ gate, and 1 measurement. In total, evaluating $\vec{\mathcal{V}}$ requires

$$m_{\mathcal{V}} \, P \, L \, T \quad \text{measurements and } \hat{S} \text{ gates,} \tag{2.39}$$

$$m_{\mathcal{V}} \, P^2 \, L \, T \quad \text{ansatz gates,} \tag{2.40}$$

$$m_{\mathcal{V}} \, P \, L \, T (N + D) \quad \text{controlled Paulis,} \tag{2.41}$$

in addition to twice as many ancilla Hadamards as measurements. Finally, if we treat the costs of measurement, Hadamard gates, $\hat{S}$ gates, ansatz gates, and controlled (single-target) Pauli gates as equivalent, then the experimentalist's total quantum costs of simulating a single iteration of imaginary-time evolution would become

$$\mathcal{O}\left(m_{\mathcal{M}}\, L^2\, P^2(P+D) \;+\; m_{\mathcal{V}}\, P\, L\, T(P+N+D)\right) \quad \text{operations.} \tag{2.42}$$

As an example, in the reasonable scenario whereby each ansatz gate is a two-qubit Pauli gadget ($L = 1,\ D = 2$), and the Hamiltonian contains the physically typical $T = N^4$ terms (implying $N + 1$ total working qubits in a quantum register), the total costs are well approximated as

$$\approx m_{\mathcal{M}} P^3 + m_{\mathcal{V}}(PN^5 + P^2N^4) \quad \text{operations.} \tag{2.43}$$

This suggests that in typical applications, the costs of performing imaginary-time evolution will be dominated by the Hamiltonian-related costs of populating $\vec{\mathcal{V}}$, equivalent to performing quantum gradient descent. We thus expect that imaginary-time evolution can replace gradient descent as a minimisation scheme while incurring only marginal increases in quantum costs. Indeed we will later see this to be true, exaggerated by other factors including shot noise resilience and convergence improvements.

2.4.4 Classical subroutine

We have so far described all quantum subroutines involved in variational imaginary time evolution, enabling us to populate quantities $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ at every iteration. We now

turn our attention to the classical subroutine; solving

$$\underline{\underline{\mathcal{M}}}\,\dot{\vec{\theta}} = -\vec{\mathcal{V}} \qquad \text{(reiteration of 2.21)}$$

for $\dot{\vec{\theta}}$, which inform the updated parameters of the next iteration. We will generally be unable to directly invert $\underline{\underline{\mathcal{M}}}$ to obtain

$$\dot{\vec{\theta}} = -\,\underline{\underline{\mathcal{M}}}^{-1}\,\vec{\mathcal{V}}, \tag{2.44}$$

as we cannot expect $\underline{\underline{\mathcal{M}}}$ to remain invertible at every assignment of $\vec{\theta}$. We should instead seek

$$\min_{\dot{\vec{\theta}}} \quad \left\| \underline{\underline{\mathcal{M}}}\,\dot{\vec{\theta}} + \vec{\mathcal{V}} \right\| \tag{2.45}$$

through numerical schemes sensitive to the expected properties of the parametrized norm space. We can already appreciate a major source of singularity is periodicity in our ansatz state $|\psi(\vec{\theta})\rangle$, and hence in $\underline{\underline{\mathcal{M}}}$ as a function of $\vec{\theta}$. This implies Equation 2.21 is underspecified, and suggests the stability of inversion can be improved by introducing constraints on the solution $\dot{\vec{\theta}}$. We can do this through regularisation of Equation 2.45, for instance using Tikhonov regularisation [166] whereby we seek

$$\min_{\dot{\vec{\theta}}} \quad \left\| \underline{\underline{\mathcal{M}}}\,\dot{\vec{\theta}} + \vec{\mathcal{V}} \right\|^2 + \mu \left\| \dot{\vec{\theta}} \right\|^2, \tag{2.46}$$

for some Tikhonov hyperparameter $\mu > 0$. This pressures the change in parameters to be small, such that the evolution of the parameters $\vec{\theta}_\tau$ is smooth. An increasing μ suggests a stronger pressure for smoothness at a reduced accuracy against true imaginary-time evolution. Fortunately in our application, we care little about strong

faithfulness to imaginary-time evolution, and instead only wish to utilise its rapid convergence to the ground state. This permits us to use a larger μ (or choose its value more flippantly) than when regularising Li's real-time method [132]. For the same reasons however, we should not preclude large leaps in parameter space ($\|\dot{\vec{\theta}}\| \gg 1$) if they improve convergence, and hence should not over-constrain evolution through a large μ. We resolve these tensions by dynamically choosing μ as the corner of an L-curve [167] formed by a mere *three* samples, though we note this problem remains an active area of classical research [168, 169, 170].

Equation 2.46 can now be solved for $\dot{\vec{\theta}}$ with any number of multi-parameter least squares regression routines. While its analytic solution is given by the normal form

$$\dot{\vec{\theta}} = -\left(\underline{\underline{\mathcal{M}}}^2 + \mu^2 \mathbb{1}\right)^{-1} \underline{\underline{\mathcal{M}}} \vec{\mathcal{V}}, \tag{2.47}$$

it is prudent to instead leverage a numerically stable solving routine, such as through singular value decomposition [171]. With a solution $\dot{\vec{\theta}}$ obtained, we can update the ansatz parameters via a simple Euler step [119]

$$\vec{\theta}_{\tau+\Delta\tau} = \vec{\theta}_\tau + \Delta\tau\, \dot{\vec{\theta}}, \tag{2.48}$$

where $\Delta\tau$ is the timestep, and is a hyperparameter of variational imaginary-time evolution, as it was for quantum gradient descent. Presently, we fix this parameter, and can leverage the expected monotonic decrease of the energy $\langle\psi(\vec{\theta}_\tau)|\hat{H}|\psi(\vec{\theta}_\tau)\rangle$ to heuristically determine a suitable timestep ahead of simulation, choosing the largest which maintains this property. In subsequent works presented in this thesis, we will develop more advanced techniques to dynamically modify $\Delta\tau$.

2.4.5 Relation to natural gradient

Subsequent to our development of variational imaginary-time evolution, the quantum adaptation of the natural gradient [133], as introduced in Section 1.6.3, was developed. This variational algorithm is almost identical to our imaginary-method above, although replaces our matrix $\underline{\underline{\mathcal{M}}}$ with $\underline{\underline{g}}$ satisfying

$$g_{ij} = \mathcal{M}_{ij} - \Re\left[\frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\langle\psi|\frac{\partial|\psi\rangle}{\partial\theta_j}\right]. \qquad \text{(re-expression of 1.66)}$$

There is some confusion in the literature over the relationship and relative performances of quantum natural gradient and variational imaginary-time evolution. In this section, we will show $\underline{\underline{g}}$ and $\underline{\underline{\mathcal{M}}}$ are equivalent when the ansatz is designed such that the global phase is effectively unconstrained in the variational equations. This equivalence will motivate an important ansatz design consideration used subsequently in this chapter.

Interestingly, the equivalence was first discovered through an unrelated attempt of my co-authors to reconcile a density matrix derivation of imaginary-time [172] with the statevector formulation used in this thesis. It was realised that the variational expression of pure imaginary-time did not admit the global phase gauge symmetry inherent to the density matrix formalism. That is, Equation 2.21 gratuitously constrains the ansatz parameters toward states with specific non-physical global phases (so too does the original Li's method [160]), as would a naive classical simulation of the pure real-time dynamics. This suggests a "waste" of the parameters, and that a derivation of our method without a global phase prescription could more accurately simulate the probability dynamics of imaginary-time, and hence converge to the ground-state faster. To address this, we merely add an additional parameter to our ansatz circuits which determines only the global phase. Being otherwise unconstrained, it can freely "undo"

the total global phase prescription of the other parameters, effectively removing their constraint. This heuristic solution proved very effective, improving the convergence reliability of variational imaginary-time at the cost of a single ansatz parameter. We will show now that this heuristic incidentally yields equivalent variational equations to quantum natural gradient.

As before, let $|\psi(\vec{\theta})\rangle$ notate the ansatz state produced from P parameters $\vec{\theta} = \{\theta_1,\ \ldots,\ \theta_P\}$. We continue to let $\underline{\underline{\mathcal{M}}} : \mathbb{R}^{P\times P}$ and $\vec{\mathcal{V}} : \mathbb{R}^P$ notate the variational imaginary-time objects of this system. Then, let

$$|\Phi(\vec{\theta}, \phi)\rangle = e^{\mathrm{i}\,\phi}\,|\psi(\vec{\theta})\rangle \tag{2.49}$$

be an ansatz state of $P+1$ parameters, where the final parameter ϕ modulates only the global phase. Let $\underline{\underline{\mathcal{M}}}' : \mathbb{R}^{(P+1)\times(P+1)}$ and $\vec{\mathcal{V}}' : \mathbb{R}^{P+1}$ denote the variational imaginary-time objects of this larger system. These satisfy

$$(i < P+1) \qquad \mathcal{V}'_i = \Re\left[\langle\Phi|\,\hat{H}\frac{\partial\,|\Phi\rangle}{\partial\theta_i}\right] = \Re\left[\langle\psi|\,\hat{H}\frac{\partial\,|\psi\rangle}{\partial\theta_i}\right] = \mathcal{V}_i, \tag{2.50}$$

$$\mathcal{V}'_{P+1} = \Re\left[\langle\Phi|\,\hat{H}\frac{\partial\,|\Phi\rangle}{\partial\phi}\right] = \Re\left[\mathrm{i}\,\langle\psi|\,\hat{H}\,|\psi\rangle\right] = 0, \tag{2.51}$$

by Hermitivity of $\hat{H}$. Meanwhile, $\underline{\underline{\mathcal{M}'}}$ remains symmetric with elements

$$(i,j\ < P+1) \qquad \mathcal{M}'_{ij} = \Re\left[\frac{\partial\langle\Phi|}{\partial\theta_i}\frac{\partial|\Phi\rangle}{\partial\theta_j}\right] = \Re\left[\frac{\partial\langle\psi|}{\partial\theta_i}\frac{\partial|\psi\rangle}{\partial\theta_j}\right] \tag{2.52}$$
$$= \mathcal{M}_{ij},$$

$$(i < P+1) \qquad \mathcal{M}'_{i,P+1} = \Re\left[\frac{\partial\langle\Phi|}{\partial\theta_i}\frac{\partial|\Phi\rangle}{\partial\phi}\right] = \Re\left[\mathrm{i}\frac{\partial\langle\psi|}{\partial\theta_i}|\psi\rangle\right] \tag{2.53}$$
$$= -\Im\left[\frac{\partial\langle\psi|}{\partial\theta_i}|\psi\rangle\right],$$

$$\mathcal{M}_{P+1,P+1} = \Re\left[\frac{\partial\langle\Phi|}{\partial\phi}\frac{\partial|\Phi\rangle}{\partial\phi}\right] = \Re\left[-\mathrm{i}^2\langle\psi|\psi\rangle\right] \tag{2.54}$$
$$= 1,$$

by normalisation of $|\psi(\vec{\theta})\rangle$. We have shown that inclusion of the global phase parameter ϕ has admitted

$$\underline{\underline{\mathcal{M}'}} = \left(\begin{array}{ccc:c} & & & -\Im\left[\frac{\partial\langle\psi|}{\partial\theta_1}|\psi\rangle\right] \\ & \underline{\underline{\mathcal{M}}} & & \vdots \\ & & & -\Im\left[\frac{\partial\langle\psi|}{\partial\theta_P}|\psi\rangle\right] \\ \hdashline -\Im\left[\frac{\partial\langle\psi|}{\partial\theta_1}|\psi\rangle\right] & \dots & -\Im\left[\frac{\partial\langle\psi|}{\partial\theta_P}|\psi\rangle\right] & 1 \end{array}\right), \quad \vec{\mathcal{V}}' = \left(\begin{array}{c} \vec{\mathcal{V}} \\ \\ \hdashline 0 \end{array}\right). \tag{2.55}$$

where $-\Im\left[\frac{\partial\langle\psi|}{\partial\theta_i}|\psi\rangle\right]$ are the so-called Berry connections [173]. Expanding the imaginary-

time variational equation $\underline{\underline{\mathcal{M}'}}\left(\vec{\dot{\theta}} \otimes \{\dot{\phi}\}\right) = -\mathcal{V}'$ suggests

$$(i < P+1) \qquad -\mathcal{V}'_i = \sum_j^P \mathcal{M}'_{ij}\,\dot{\theta}_j \;+\; \mathcal{M}'_{i,P+1}\,\dot{\phi} \tag{2.56}$$

$$\implies -\mathcal{V}_i = \sum_j^P \mathcal{M}_{ij}\,\dot{\theta}_j \;-\; \Im\left[\frac{\partial\,\langle\psi|}{\partial\theta_i}\,|\psi\rangle\right]\,\dot{\phi}, \tag{2.57}$$

$$\text{meanwhile} \qquad -\mathcal{V}'_{P+1} = \sum_j^P \mathcal{M}'_{P+1,j}\,\dot{\theta}_j \;+\; \mathcal{M}'_{P+1,P+1}\,\dot{\phi} \tag{2.58}$$

$$\implies 0 = -\sum_j^P \Im\left[\frac{\partial\,\langle\psi|}{\partial\theta_j}\,|\psi\rangle\right]\,\dot{\theta}_j \;+\; \dot{\phi}. \tag{2.59}$$

Equation 2.59 isolates a solution for $\dot{\phi}$ which we substitute into Equation 2.57 and factorise with respect to $\dot{\theta}_j$ to arrive at

$$-\mathcal{V}_i = \sum_j^P \dot{\theta}_j \left(\mathcal{M}_{ij} \;-\; \Im\left[\frac{\partial\,\langle\psi|}{\partial\theta_i}\,|\psi\rangle\right]\,\Im\left[\frac{\partial\,\langle\psi|}{\partial\theta_j}\,|\psi\rangle\right]\right). \tag{2.60}$$

Our next step is to replace the imaginary components by observing the enclosed scalars are strictly imaginary. Specifically, by asserting the ansatz state $|\Phi\rangle = \hat{U}\,|\text{in}\rangle$ is separable between unitaries,

$$\text{i.e.} \qquad \hat{U}(\vec{\theta},\phi) = \hat{U}_{P+1}(\phi)\prod_{k=P}^{1}\hat{U}_k(\theta_k) \qquad \text{(re-expression of 2.26)}$$

$$= e^{\mathrm{i}\,\phi}\,\hat{U}_{1:P}, \qquad \text{(as per notation 2.27)}$$

and leveraging that each unitary gate can be expressed in terms of a Hermitian generator (Equation 1.17) of which the derivative introduces a skew-Hermitian term $\mathrm{i}\,\hat{\Lambda}_i(\theta_i)$

(Equation 1.20), we can observe that

$$\frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle = -\mathrm{i}\ \langle \mathrm{in}|\ \hat{U}^\dagger_{1:i}\ \hat{\Lambda}_i\ \hat{U}^\dagger_{i+1:P}\ \hat{U}_{1:P}\ |\mathrm{in}\rangle \tag{2.61}$$

$$= -\mathrm{i}\ \langle \mathrm{in}|\ \hat{U}^\dagger_{1:i}\ \hat{\Lambda}_i\ \hat{U}_{1:i}\ |\mathrm{in}\rangle \tag{2.62}$$

is strictly imaginary, due to Hermitivity of Λ_i. Therefore, we can express

$$\Im\left[\frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\right]\Im\left[\frac{\partial \langle\psi|}{\partial\theta_j}|\psi\rangle\right] = \langle \mathrm{in}|\,\hat{U}^\dagger_{1:i}\,\hat{\Lambda}_i\,\hat{U}_{1:i}\,|\mathrm{in}\rangle\ \ \langle \mathrm{in}|\,\hat{U}^\dagger_{1:j}\,\hat{\Lambda}_j\,\hat{U}_{1:j}\,|\mathrm{in}\rangle \tag{2.63}$$

$$= -\frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\ \ \frac{\partial \langle\psi|}{\partial\theta_j}|\psi\rangle \tag{2.64}$$

$$= \frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\ \ \langle\psi|\frac{\partial\,|\psi\rangle}{\partial\theta_j}. \tag{2.65}$$

Since this product is real, we can arbitrarily notate its real component, to match our form of natural gradient. Finally, we repeat our observation in Equation 2.24 that $\mathcal{V}_i = \nabla \langle E\rangle_i /2$, discarding the constant factor which will absorb into the time-step, and thus rearrange Equation 2.60 into

$$-\nabla \langle E\rangle_i = \sum_j^P \dot{\theta}_j \left(\mathcal{M}_{ij} - \Re\left[\frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\ \ \langle\psi|\frac{\partial\,|\psi\rangle}{\partial\theta_j}\right]\right). \tag{2.66}$$

This is recognised as the expanded form of the quantum natural gradient with an energy cost function, $-\nabla \langle E\rangle = \underline{\underline{g}}\,\dot{\vec{\theta}}$ (Equation 1.65). Notice $\vec{\theta}$ does not contain the global phase parameter ϕ, and $|\psi(\vec{\theta})\rangle$ does not necessarily contain a global phase gate nor impose an unconstrained global phase. We have ergo shown that introducing an additional global phase parameter into variational imaginary time admits variational equations equivalent to natural gradient with one fewer parameters. The total resource costs are unchanged - the additional parameter in imaginary-time introduces P more quantum observables to evaluate (the Berry connections [173]), while the method avoids the

additional product terms in each natural gradient tensor element, which are themselves formed of P total Berry connections. The variational imaginary-time equations (when a global phase is included) differ from the natural gradient equations only in their matrix form, where one of the P simultaneous equations have been separated into two.

This analysis has demonstrated that ansätze designed for variational imaginary-time evolution should always include a parameter determining only the global phase gate. This can be achieved with a single qubit gate acting upon an eigenstate, and performed at the start of the ansatz circuit upon the known initial state. For example, if $|\text{in}\rangle$ includes a qubit q in a computational basis state, then one can leverage a single-qubit rotation gate on that qubit to invoke,

$$\exp(\mathrm{i}\,\theta\,\hat{Z}_q)\,|\text{in}\rangle = e^{\pm\,\mathrm{i}\,\theta}\,|\text{in}\rangle\,. \tag{2.67}$$

For the remainder of this thesis, we will ensure our ansätze include such a gate.

2.5 Demonstration

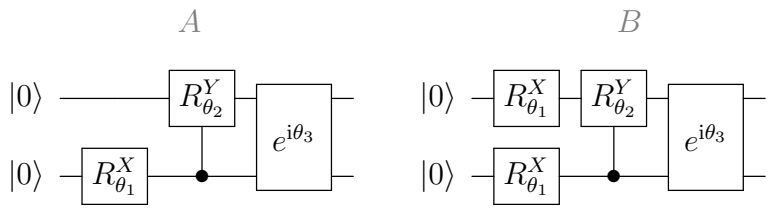
Sections 2.4.2 and 2.4.4 developed the quantum and classical subroutines of variational imaginary-time evolution. Before we proceed to a large-scale numerical investigation, we demonstrate a small illustrative analytic study.

We highlight two interesting examples; one where imaginary-time evolution greatly outperforms gradient descent as a minimisation routine (labelled A), and an adversarial

example with similar performances (labelled B). These are diagonal systems

$$\hat{H}_A = \begin{pmatrix} 1 & & & \\ & 2 & & \\ & & 3 & \\ & & & 0 \end{pmatrix}, \qquad \hat{H}_B = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 2 & \\ & & & 0 \end{pmatrix}, \tag{2.68}$$

with two-qubit three-parameter ansätze



where both include a global phase of θ_3. They differ by ansatz B having a separable ansatz gate $\hat{U}_1(\theta_1) = \exp(\mathrm{i}\theta_1\hat{X}/2) \otimes \exp(\mathrm{i}\theta_1\hat{X}/2)$. These ansätze generate states

$$|\psi_A\rangle = e^{\mathrm{i}\theta_3} \begin{pmatrix} \cos\left(\frac{\theta_1}{2}\right) \\ -\mathrm{i}\cos\left(\frac{\theta_2}{2}\right)\sin\left(\frac{\theta_1}{2}\right) \\ 0 \\ -\mathrm{i}\sin\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_2}{2}\right) \end{pmatrix}, \qquad |\psi_B\rangle = e^{\mathrm{i}\theta_3} \begin{pmatrix} \cos^2\left(\frac{\theta_1}{2}\right) \\ \sin^2\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_2}{2}\right) - \frac{1}{2}\mathrm{i}\sin\left(\theta_1\right)\cos\left(\frac{\theta_2}{2}\right) \\ -\frac{1}{2}\mathrm{i}\sin\left(\theta_1\right) \\ -\sin^2\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_2}{2}\right) - \frac{1}{2}\mathrm{i}\sin\left(\theta_1\right)\sin\left(\frac{\theta_2}{2}\right) \end{pmatrix}, \tag{2.69}$$

with simple analytic expectation energies visualised in Figure 2.3. For both systems,

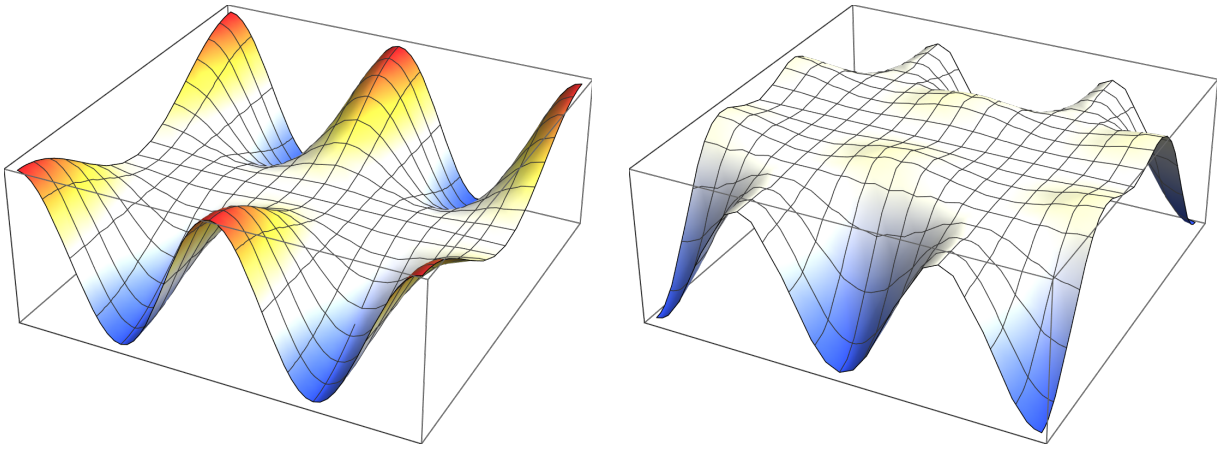


Figure 2.3: The parametrized energy landscapes accessible to our toy analytic ansätze. Colour emphasises the energy (z axis) relative to the extrema of system A, and is consistent with Figure 2.4.

$\underline{\underline{\mathcal{M}}}$ is invertible for $\theta_1 \neq 2\pi n$ ($n \in \mathbb{Z}$), and admits the forms

$$\vec{\mathcal{V}}_A = \begin{pmatrix} \frac{1}{4}\sin(\theta_1)\cos(\theta_2) \\ -\frac{1}{2}\sin(\theta_2)\sin^2\left(\frac{\theta_1}{2}\right) \\ 0 \end{pmatrix}, \qquad \vec{\mathcal{V}}_B = \begin{pmatrix} \frac{1}{4}\cos^2(\theta_2/2)\left(\sin(2\theta_1) - \sin(\theta_1)\right) \\ -\frac{1}{4}\cos(\theta_1)\sin(\theta_2)\sin^2(\theta_1/2) \\ 0 \end{pmatrix},$$

$$\underline{\underline{\mathcal{M}_A}}^{-1}\,\vec{\mathcal{V}}_A = \begin{pmatrix} \sin(\theta_1)\cos(\theta_2) \\ -2\sin(\theta_2) \\ 0 \end{pmatrix}, \qquad \underline{\underline{\mathcal{M}_B}}^{-1}\,\vec{\mathcal{V}}_B = \begin{pmatrix} \frac{1}{2}\cos^2\left(\frac{\theta_2}{2}\right)\left(\sin\left(2\theta_1\right) - \sin\left(\theta_1\right)\right) \\ -\frac{8\sin(\theta_2)\cos(\theta_1)}{\cos(\theta_1)+2\cos(2\theta_1)-\cos(3\theta_1)+6} \\ \frac{2\sin^2(\theta_1/2)\sin(2\theta_1)\sin(\theta_2)}{\cos(\theta_1)+2\cos(2\theta_1)-\cos(3\theta_1)+6} \end{pmatrix}. \tag{2.70}$$

Recall that variational imaginary-time evolution and gradient descent respectively evolve the parameters $\vec{\theta} = \{\theta_1, \theta_2, \theta_3\}$ under

$$\Delta\vec{\theta} = -\Delta t\, \underline{\underline{\mathcal{M}}}^{-1}\vec{\mathcal{V}}, \qquad \Delta\vec{\theta} = -\,\Delta t\, \vec{\mathcal{V}}. \qquad \text{(reiteration of 2.21 and 1.57)}$$

For system A, $\underline{\underline{\mathcal{M}}}^{-1}$ upon $\vec{\mathcal{V}}$ eliminates a $\sin^2(\theta_1/2)$ term from $\Delta\theta_2$, decoupling it from θ_1. This suggests imaginary time can independently optimise these parameters. Curiously,

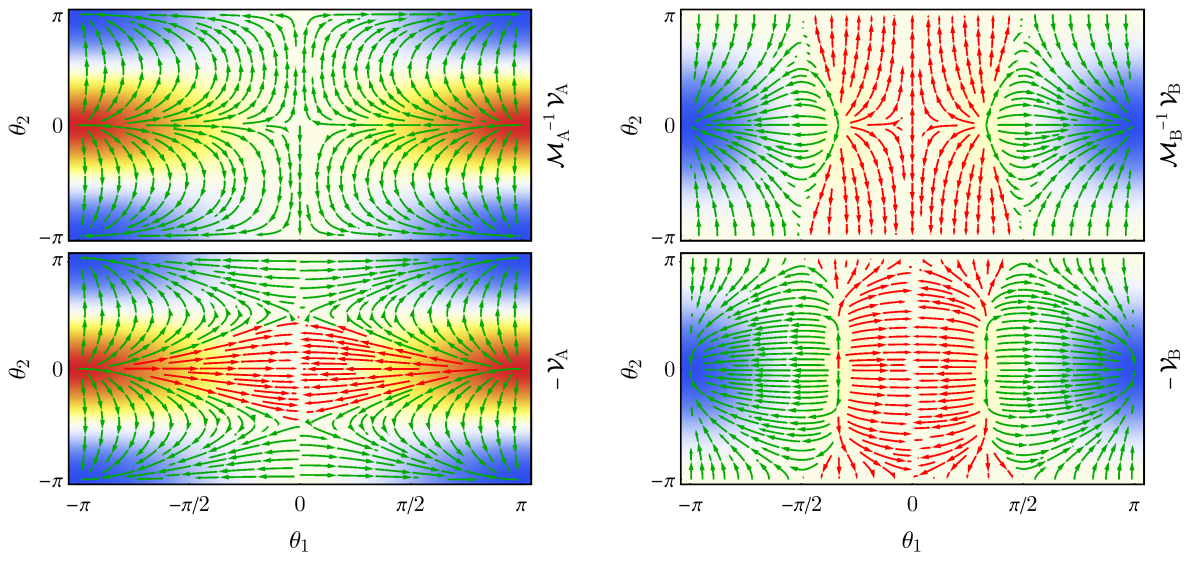


Figure 2.4: Minimisation of $\langle\psi_A|\hat{H}_A|\psi_A\rangle$ (left) and $\langle\psi_B|\hat{H}_B|\psi_B\rangle$ (right) of Fig. 2.3 via imaginary-time (top) and gradient descent (bottom). Arrows indicate $\Delta\vec{\theta}$ and are green if evolving from their (θ_1, θ_2) starting point ultimately leads to convergence to a global minima (blue area), else red. While $\Delta t = 1$ for 100 iterations, the simulations were stably insensitive to these hyperparameters.

it makes no prescription on the global phase ($\Delta\theta_3 = 0$). For the adversarial system B, the imaginary-time equations replace $\sin^2(\theta_1/2)$ in $\Delta\theta_2$ with a reciprocal transcendental of θ_1. We illustrate simulation of both methods in Figure 2.4, and see variational imaginary-time greatly outperforms gradient descent when minimising system A. In contrast, under the measure of the total area of (θ_1, θ_2) initial states which ultimately converge to global minima, gradient descent outperforms imaginary-time, by a small margin.

2.6 Simulation

We now have a complete quantum algorithm to perform variational imaginary-time evolution, which we have seen outperform gradient descent as a minimisation routine

in uncharacteristically small, noise-free settings. An experimentalist could now deploy these routines to a real quantum device, provided our circuits can be implemented without hassle. Alas, it is yet unclear whether the limited accuracy of today's machines would render such an effort pointless, since we have not yet investigated its resilience to device imperfections and noise. An analytic treatment of noise in variational algorithms can be significantly challenging, owed to our black box treatment of the ansatz state. So, we will instead numerically simulate our algorithm under varying noise conditions.

A precise full-state classical simulation of our algorithm under noise naively suggests the use of channels upon density matrices. But density matrix simulation requires *square* as many classical resources (memory and time) than the statevector simulation of the same number of qubits. In our tested regime, statevector simulation of pure states is on the threshold of tractability, while density matrix simulation of mixed states is beyond it. To perform our tests, we first developed a bespoke optimisation for statevector simulation of our algorithm, and an additional strategy to approximate the effect of noise upon it by randomly perturbing the output variational observables. We derive these methods below. We assure the surprised reader that the unexpected costs of classically simulating variational algorithms will be explained in Section 5.3.

2.6.1 Pure

To perform variational imaginary-time evolution, an experimentalist must perform the Hadamard test circuits upon variational states, according to Equation 2.34 (and visualised in Figure 2.2). This is to ultimately arrive at approximate experimental evalua-

tions of the quantities

$$\mathcal{M}_{ij} = \Re\left[\frac{\partial \langle\psi(\vec{\theta}_\tau)|}{\partial\theta_i}\frac{\partial |\psi(\vec{\theta}_\tau)\rangle}{\partial\theta_j}\right], \qquad \mathcal{V}_i = \Re\left[\langle\psi(\vec{\theta}_\tau)|\,\hat{H}\frac{\partial |\psi(\vec{\theta}_\tau)\rangle}{\partial\theta_i}\right].$$

(reiteration of 2.22 and 2.23)

In this pursuit, the experimentalist must invoke per-gate derivatives of the P-parameter ansatz state, and decompositions of the D-qubit gate derivatives into weighted sums of L unitary operators;

$$\frac{\partial |\psi(\vec{\theta})\rangle}{\partial\theta_j} = \hat{U}_{j+1:P}\,\frac{\mathrm{d}\hat{U}_j(\theta_j)}{\mathrm{d}\theta_j}\,\hat{U}_{1:j-1}, \qquad \frac{\mathrm{d}\hat{U}_j(\theta_j)}{\mathrm{d}\theta_j} = \mathrm{i}\sum_l^L \lambda_{lj}\bigotimes_i^D \hat{\sigma}_{jli}\,\hat{U}_j(\theta_j).$$

(reiteration of 2.28 and 2.29)

We showed in Section 2.4.3 that this endeavour costs as many gates as

$$\mathcal{O}\left(m_{\mathcal{M}}\,L^2\,P^2(P+D)\;+\;m_{\mathcal{V}}\,P\,L\,T(P+N+D)\right) \qquad \text{(reiteration of 2.42)}$$

per every iteration, where T is the number of terms in the N-qubit Hamiltonian, and $m_{\mathcal{M}}$ (and $m_{\mathcal{V}}$) are the number of measurements and Hadamard test repetitions afforded to estimating each observable necessary in the computation of an element $\mathcal{M}_{ij}$ (and $\mathcal{V}_i$),

However, in classical simulations we need not respect these experimental limitations. We can instead compute $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ through any means admitted by their analytic forms above, provided doing so remains tractable for our target system sizes. We opt to treat $|\psi(\vec{\theta}_\tau)\rangle$ as a computable black-box, and obtain numerical expressions for the derivative vectors (which are *not* L^2 statevectors) through finite difference approximations. Specifically, by leveraging the fourth-order central finite difference formula [174] with a

stepsize $\Delta\theta \ll 1$,

$$\frac{\partial \left|\psi(\vec{\theta})\right\rangle}{\partial \theta_i} \approx \frac{1}{\Delta\theta} \sum_{n=1}^{4} \pm c_n \left|\psi(\vec{\theta}|_{\theta_i \to \theta_i \pm n\Delta\theta})\right\rangle , \tag{2.71}$$

$$c_1 = \frac{4}{5}, \quad c_2 = -\frac{1}{5}, \quad c_3 = \frac{4}{105}, \quad c_4 = -\frac{1}{280}, \tag{2.72}$$

we can approximate and record each of the P derivative vectors via *eight* evaluations of the P-gate ansatz circuit, each time changing one parameter. This is a fixed-factor more expensive and approximate alternative to the parameter shift rule [120] but one which permits any form of ansatz gate. We then populate the $P \times P$ symmetric matrix $\underline{\underline{\mathcal{M}}}$ through $(P^2 + P)/2$ inner products between the computed derivative vectors. To populate $\vec{\mathcal{V}}$, we first obtain another non-L^2 vector

$$\left|\Psi\right\rangle \coloneqq \left|\psi(\vec{\theta}_\tau)\right\rangle , \tag{2.73}$$

$$\left|\Phi\right\rangle \coloneqq \hat{H} \left|\psi(\vec{\theta}_\tau)\right\rangle = \sum_{k}^{T} h_k \left(\bigotimes_{i}^{N} \hat{\sigma}_{ki} \left|\Psi\right\rangle \right) , \tag{2.74}$$

through an additional ansatz circuit evaluation (cloning the output state), and a simulation of each Hamiltonian Pauli tensor effected as operators upon that cloned state, weighted-summed. We then compute $\vec{\mathcal{V}}$ through P inner products between the fixed $\left|\Phi\right\rangle$ and the derivative vectors. By assuming each inner product and vector-sum has roughly the same cost as simulating a single ansatz gate (all are $\mathcal{O}(2^N)$), then evaluating $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ in this manner has a similar cost to simulating a total of

$$\approx \frac{17}{2}P^2 + \frac{5}{2}P + T(1 + N) \quad \text{gates.} \tag{2.75}$$

This is regardless of the number of Pauli tensors L in the generator of each ansatz gate. Ergo, even in the experimentally easiest setting of $L = 1$, our scheme effectively reduces

the number of gate simulations needed by $\mathcal{O}(P^3 + P\,T\,N)$, from the number necessary to simulate each circuit (with a single shot) in the experimental procedure as prescribed by Equation 2.42. We leverage this scheme to simulate noise-free variational imaginary time evolution on quantum computers of sizes which would be classically intractable to precisely emulate. And in the proceeding section, we describe how this method can be extended still to enable noisy simulation.

We finally remark that the total classical cost of simulating P-parameter N-qubit variational imaginary-time minimisation in this manner is $\mathcal{O}(P^2\,2^N)$ time and $\mathcal{O}(P\,2^N)$ memory. In Chapter 5 of this thesis, we will develop a superior method for simulation of the imaginary-time and natural gradient processes, enabling simulation in the same time but with a P-independent memory cost of $\mathcal{O}(2^N)$. It remains as the asymptotically fastest noise-free precise simulator of quantum natural gradient and quantum imaginary-time evolution in the literature to date. Alas, it was not ready for deployment in the study of this section's work.

2.6.2 Noisy

The preceding section outlined an efficient noise-free classical simulation scheme of variational imaginary-time evolution. We now seek to introduce noise as produced both by a rudimentary model of per-gate error, and by shot noise in the finite sampling estimation of expectation values. We wish to avoid prohibitively expensive density matrix simulation. We will now show how to incorporate both kinds of error by cheaply performing noise-free statevector simulation as above to determine quantities $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$, before pseudorandomly perturbing them. To derive our method, we propagate the probability distributions of each experimental measurement to distributions unto the

final observables.

We begin by first considering only the effect of shot noise. Let $\underline{\underline{M}}$ and $\vec{V}$ be the matrix and vector of random variables approximating $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$, populated by an experimentalist within an iteration of imaginary time evolution. We first focus on $\underline{\underline{M}}$. Recall that

$$\mathcal{M}_{ij} = \sum_{l_1,l_2}^{L} \lambda_{l_1 i}\lambda_{l_2 j} \ \langle i,j,l_1,l_2\rangle \,, \qquad \text{(re-expression of 2.30)}$$

where we have defined

$$\langle i,j,l_1,l_2\rangle = \Re\left[\langle \text{in}| \ U^\dagger_{1:i-1} \ \vec{\sigma}^{\otimes}_{l_1 i} \ \hat{U}^\dagger_{i:j-1} \ \vec{\sigma}^{\otimes}_{l_2 j} \ \hat{U}_{1:j-1} \ |\text{in}\rangle\right], \qquad (2.76)$$

and where $\lambda_{lj} \in \mathbb{R}$ appear in the Pauli generator of our ansatz gates $\hat{U}_i(\theta_i) = \exp(\mathrm{i}\,\theta_i \sum_l \lambda_{li}\,\vec{\sigma}^{\otimes}_{li})$. We demonstrated in Equation 2.34 that each $\langle i,j,l_1,l_2\rangle$ can be experimentally evaluated using the Hadamard test, through repeated measurement of the ancilla qubit in the circuit of Figure 2.2. Let $\{A_k^{(i,j,l_1,l_2)} \ : \ 1 \le k \le m_{\mathcal{M}}\}$ be the random outcomes of $m_{\mathcal{M}}$ independent measurements of this ancilla, which satisfy

$$A_k^{(i,j,l_1,l_2)} \in \{-1,\ 1\}, \qquad (2.77)$$

$$\mathbb{E}[A_k^{(i,j,l_1,l_2)}] = \langle i,j,l_1,l_2\rangle \,, \qquad (2.78)$$

$$\therefore \ \text{Var}[A_k^{(i,j,l_1,l_2)}] = 1 - \langle i,j,l_1,l_2\rangle^2 \,. \qquad (2.79)$$

Let $B^{(i,j,l_1,l_2)}$ be the average of these measurements, by which an experimentalist ap-

proximates $\langle i, j, l_1, l_2 \rangle$, and ergo computes $M_{ij} \approx \mathcal{M}_{ij}$.

$$B^{(i,j,l_1,l_2)} = \frac{1}{m_{\mathcal{M}}} \sum_{k}^{m_{\mathcal{M}}} A_k^{(i,j,l_1,l_2)}, \tag{2.80}$$

$$\implies M_{ij} = \sum_{l_1,l_2}^{L} \lambda_{l_1 i}\, \lambda_{l_2 j}\, B^{(i,j,l_1,l_2)}. \tag{2.81}$$

The central limit theorem [175] asserts that for sufficiently many measurements $m_{\mathcal{M}} \gg 1$, the random variable $B^{(i,j,l_1,l_2)}$ approximates a variate of the normal distribution

$$B^{(i,j,l_1,l_2)} \sim \mathcal{N}\left(\langle i, j, l_1, l_2 \rangle\,,\ \frac{1}{m_{\mathcal{M}}} \left(1 - \langle i, j, l_1, l_2 \rangle^2\right) \right). \tag{2.82}$$

By propagating $\mathrm{Var}(\sum_l \alpha_l B^{(l)}) = \sum_l {\alpha_l}^2 \mathrm{Var}(B^{(l)})$, we can treat M_{ij} as a variate of the normal distribution

$$M_{ij} \sim \mathcal{N}\left(\sum_{l_1,l_2}^{L} \lambda_{l_1 i}\, \lambda_{l_2 j}\, \langle i, j, l_1, l_2 \rangle\,,\ \frac{1}{m_{\mathcal{M}}} \sum_{l_1,l_2}^{L} {\lambda_{l_1 i}}^2\, {\lambda_{l_2 j}}^2 \left(1 - \langle i, j, l_1, l_2 \rangle^2\right) \right). \tag{2.83}$$

Ergo, M_{ij} has mean $\mathcal{M}_{ij}$, and a variance determined by the expectation values which constitute $\mathcal{M}_{ij}$. We can therefore produce $\underline{\underline{M}}$ in our simulations by first computing the experimentally inaccessible $\underline{\underline{\mathcal{M}}}$ (finding each $\langle i, j, l_1, l_2 \rangle$) via noise-free statevector simulation, and then pseudo-randomly sampling the normal distribution of Equation 2.83, for example making use of the Box–Muller method [176]. By then substituting $\underline{\underline{M}}$ in-place of $\underline{\underline{\mathcal{M}}}$ (and a similarly sampled $\vec{V}$ in-place of $\vec{\mathcal{V}}$) in our simulations, we would effectively incorporate shot noise error. Each element of $\underline{\underline{M}}$, and each iteration of imaginary-time evolution, experiences independent noise. Hence empirical study of the effect of shot noise will require repetition of our simulations.

Note that this technique is not yet compatible with the optimised statevector simulation strategy introduced in Section 2.6.1, which avoided the need to compute individual

$\langle i, j, l_1, l_2 \rangle$. However, for the common special case whereby each ansatz gate is generated by a single Pauli tensor $\hat{U}_i(\theta_i) = \exp(\mathrm{i}\,\theta_i\,\lambda_i\,\vec{\sigma}_i^{\otimes})$ (that is when $L = 1$), Equation 2.83 simplifies to

$$M_{ij} \sim \mathcal{N}\left(\mathcal{M}_{ij},\ \frac{1}{m_{\mathcal{M}}}\left({\lambda_i}^2{\lambda_j}^2 - {\mathcal{M}_{ij}}^2\right)\right). \tag{2.84}$$

This allows the incorporation of shot noise into $\underline{\underline{M}}$ knowing only $\underline{\underline{\mathcal{M}}}$ and the ansatz gate coefficients λ_i. We can ergo compute $\underline{\underline{\mathcal{M}}}$ entirely through the method of Section 2.6.1, and thereafter produce the noise-aware matrix M_{ij} with no additional quantum simulation.

Modelling shot noise in $\vec{V}$ is slightly harder. Through similar working, we can show in general that sufficiently many measurements-per-Hadamard test $m_{\mathcal{V}}$ admits distribution

$$V_i \sim \mathcal{N}\left(\mathcal{V}_i,\ \frac{1}{m_{\mathcal{V}}}\sum_{l_1}^{L}\sum_{l_3}^{T}{\lambda_{l_1 i}}^2{h_{l_3}}^2(1 - \langle i, l_1, l_3\rangle^2)\right), \tag{2.85}$$

where we have notated the inner-product of Equation 2.33 as

$$\langle i, l_1, l_3\rangle = \Im\left[\langle \mathrm{in}|\ \hat{U}_{1:P}^{\dagger}\ \vec{\sigma}_{l_3}^{\otimes}\ \hat{U}_{i:P}\ \vec{\sigma}_{l_1 i}^{\otimes}\ \hat{U}_{1:i-1}\ |\mathrm{in}\rangle\right]. \tag{2.86}$$

Even when $L = 1$, we must assume there are many Hamiltonian terms ($T \gg 1$) such that the sample distribution resembles

$$V_i \sim \mathcal{N}\left(\mathcal{V}_i,\ \frac{{\lambda_{1i}}^2}{m_{\mathcal{V}}}\sum_{l_3}^{T}{h_{l_3}}^2(1 - \langle i, 1, l_3\rangle^2)\right), \tag{2.87}$$

the variance of which cannot be expressed as a function of only λ_{1i}, $\{h_{l_3}\}$ and $\mathcal{V}_i$. This suggests there is no precise way to model the shot noise in V_i without computing the T individual inner products that our optimised simulation of Section 2.6.1 avoided. So

instead, we model an upper bound of shot noise severity,

$$\mathrm{Var}(V_i) \;<\; \frac{1}{m_{\mathcal{V}}} \sum_{l_1}^{L} \sum_{l_3}^{T} {\lambda_{l_1 i}}^2 {h_{l_3}}^2, \tag{2.88}$$

which is independent of the ansatz state and which can be pre-computed before simulation. Our numerical simulations will thusly exaggerate the shot noise error in $\vec{V}$. We will happily later see that this strengthens an observation of shot error resilience in quantum variational imaginary time evolution.

So far, we have seen how to take $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ found via the optimised statevector simulation of Section 2.6.1, and imitate shot noise error by replacing them with normally-sampled random variables $\underline{\underline{M}}$ and $\vec{V}$. We now incorporate a rudimentary gate error model, resembling the depolarising channel of Equation 1.42. Let us develop a non-granular noisy channel modelling a unitary action $\hat{U}$ upon N-qubit pure state $|\mathrm{in}\rangle$ which can fail with probability $0 \leq \epsilon \leq 1$, inducing total decoherence of the register to the maximally mixed state.

$$\mathcal{D}_\epsilon \left(|\mathrm{in}\rangle \langle \mathrm{in}| \right) = (1-\epsilon)\, \hat{U} \, |\mathrm{in}\rangle \langle \mathrm{in}| \, \hat{U}^\dagger + \epsilon \frac{\mathbb{1}}{2^N}. \tag{2.89}$$

The expectation value of a subsequent Hermitian operator $\hat{V}$ can be expressed as

$$\langle \hat{V} \rangle_\epsilon = \mathrm{trace} \left(\hat{V} \; \mathcal{D}_\epsilon \left(|\mathrm{in}\rangle \langle \mathrm{in}| \right) \right) \tag{2.90}$$

$$= (1-\epsilon)\, \mathrm{trace} \left(\hat{V} \; \hat{U} \, |\mathrm{in}\rangle \langle \mathrm{in}| \, \hat{U}^\dagger \right) + \frac{\epsilon}{2^N} \mathrm{trace} \left(\hat{V} \; \mathbb{1} \right). \tag{2.91}$$

If $\hat{V}$ has a symmetric spectrum (as do the Pauli matrices) then it is also traceless;

$\text{trace}(\hat{V}) = 0$. The noisy expectation value has become

$$\langle \hat{V} \rangle_\epsilon = (1-\epsilon)\ \langle \text{in}|\, \hat{U}^\dagger\, \hat{V}\, \hat{U}\, |\text{in}\rangle \tag{2.92}$$

$$= (1-\epsilon)\ \langle \hat{V} \rangle\,. \tag{2.93}$$

Therefore our simple decoherence channel of Equation 2.89 has the effect of scaling the noise-free expectation values by $1-\epsilon$. For simplicity, let us further assume that ϵ is fixed for every one of our quantum circuits, since they have roughly equal gate depths. Then, the effect of scaling every expectation value (ultimately described by that of $\hat{Z}_{\text{ancilla}}$) in Equations 2.30 and 2.33 is to scale

$$\underline{\underline{\mathcal{M}}} \to (1-\epsilon)\, \underline{\underline{\mathcal{M}}}, \qquad \vec{\mathcal{V}} \to (1-\epsilon)\, \vec{\mathcal{V}}. \tag{2.94}$$

We crudely motivate the full circuit success rate $1-\epsilon$ as the product of probabilities that each circuit gate succeeds. Denote the single gate error rate as ξ. Then approximating each circuit involved in the evaluation of $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ to have P gates (the asymptotic upper bound) of equal error rates, we can express

$$1-\epsilon \approx (1-\xi)^P. \tag{2.95}$$

It is straightforward to integrate our two error models. The effect of decoherence is to simply scale the expectation values featured in the formulation of the shot distributions. We formalize this process in Algorithm 2.1.

We also perform a quick numerical verification that our approximate simulation strategy is sufficiently accurate in the regime of our subsequent tests. Figure 2.5 indicates that our simple depolarising model yields a $\approx 1\%$ error in the variational quantities for the

Algorithm 2.1: Classical simulation of noisy P-parameter variational imaginary-time minimisation of a Pauli Hamiltonian $\hat{H} = \sum_l^T h_l \, \vec{\sigma}_l^{\otimes}$, with shot noise informed by $m_{\mathcal{V}}$ experimental measurements per observable forming $\hat{\mathcal{V}}$ (similarly for $m_{\mathcal{M}}$ vs $\underline{\underline{\mathcal{M}}}$), and a depolarising error rate of ξ per gate. Each ansatz gate is assumed expressible as $\hat{U}_k(\theta_k) = \exp(\mathrm{i} \sum_l^L \lambda_{lk}(\theta_k) \ \otimes_q^D \hat{\sigma}_{qlk})$, as per Eq. 1.17.

1. Precompute the parameter-independent constant:

$$v = \frac{1}{m_{\mathcal{V}}} \sum_l^T {h_l}^2$$

2. For each iteration of imaginary-time evolution:
   2.1. Compute $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ as per Section 2.6.1
   2.2. For each (i, j), set M_{ij} as a Box–Muller sample of:

$$M_{ij} \sim \mathcal{N}\left((1-\xi)^P \mathcal{M}_{ij}, \ \ \frac{1}{m_{\mathcal{M}}} \left({\lambda_i}^2 {\lambda_j}^2 - (1-\xi)^{2\,P} {\mathcal{M}_{ij}}^2 \right) \right)$$

   2.3. For each i, set V_i as a Box–Muller sample of:

$$V_i \sim \mathcal{N}\Big((1-\xi)^P \, \mathcal{V}_i, \ \ v \, {\lambda_i}^2 \Big)$$

   2.4. Solve the Tikhonov condition by a least squares regression:

$$\min_{\vec{\dot{\theta}}} \ \left\| \underline{\underline{M}} \, \vec{\dot{\theta}} + \vec{V} \right\|^2 + \mu \left\| \vec{\dot{\theta}} \right\|^2$$

   2.5. Update the parameters by the Euler step:

$$\vec{\theta} \to \vec{\theta} \ + \ \Delta t \, \vec{\dot{\theta}}$$

next section's tests with gate error probabilities $\xi = 10^{-4}$, and is occluded by shot noise.

2.7 Testing

We are ready to test our variational imaginary-time evolution algorithm on more substantial systems in the presence of noise, armed with our bespoke simulation strategies.

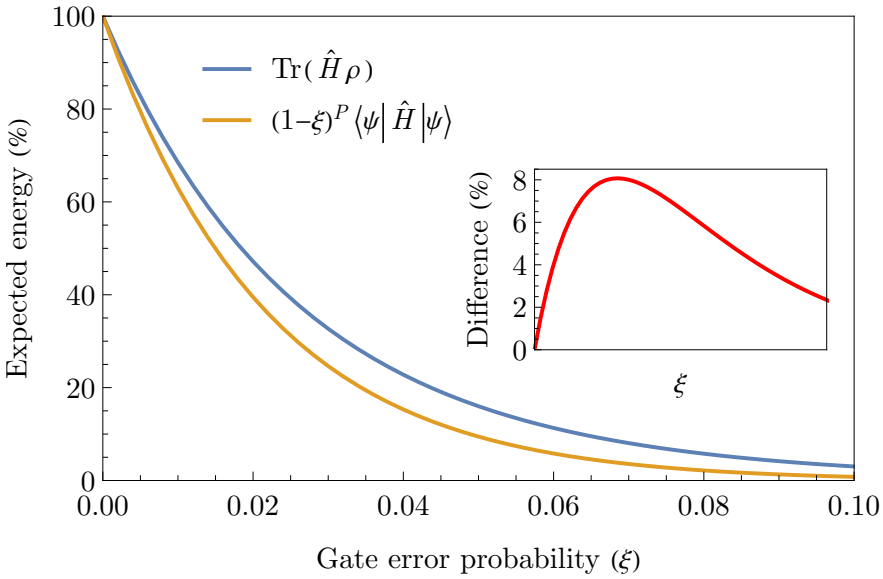
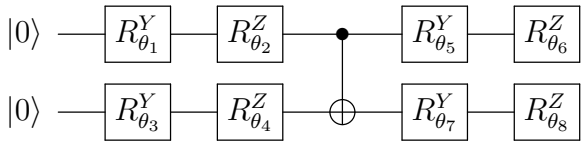


Figure 2.5: A comparison of exact density matrix simulation (blue curve) with our statevector approximation method (yellow curve) when studying the expected energy under a random Hamiltonian (a 4-qubit 20-term uniformly random Pauli string with coefficients in $[-1, 1]$) of states produced by a 4-qubit instance of the ansatz shown in Figure 2.7 with random parameters. The main plot shows the percentage deviation from the ideal expected value as noise is introduced, as prescribed by our exact and approximate methods, and the subplot shows their difference.

We have only to choose the system and the ansatz circuit. We opt to explore many-body quantum optimisation problems typical in quantum chemistry [145] which are considered in the literature as a promising application of variational routines [6]. In particular, we choose the electronic structure problem of the Hydrogen (H_2) and Lithium Hydride (LiH) molecules, and task ourselves to find the energy of the ground state electron configuration. The H_2 test is a single noise-free check performed by my coauthors to verify our variational equations indeed approximate imaginary-time evolution under a universal ansatz. The LiH tests performed by myself are substantial measures of the method's general performance and resilience to noise, under restrictive ansätze.

The encoding of these electronic problems into quantum-hardware-compatible Hamiltonians is a substantial topic, primarily investigated by my coauthors, which we will



(2.97)

Figure 2.6: The ansatz $\hat{U}^{(\mathrm{H}_2)}$ used to variationally minimise the H_2 electronic structure Hamiltonian, taken from Reference [163] Figure S2.

only briefly outline here. We consider Hydrogen H_2 in the STO-6G basis, whereby each atomic wavefunction is described by Slater-type orbitals (STO), each approximated as a linear sum of six Gaussian (6G) functions. A Hydrogen *atom* has a single spin orbital ($1S$), and spin multiplicity yields a *molecular* wavefunction of four spin orbitals. In the Bravyi–Kitaev (BK) representation [177], whereby orbitals map simply to qubits, this system is described by a four-qubit Pauli Hamiltonian, although its action on two qubits is diagonal [140, 144] and so permits a reduction to a two-qubit ($N = 2$) Pauli Hamiltonian

$$\hat{H}^{(\mathrm{H}_2)} = g_0\hat{\mathbb{1}} + g_1\hat{Z}_0 + g_2\hat{Z}_1 + g_3\hat{Z}_0\hat{Z}_1 + g_4\hat{Y}_0\hat{Y}_1 + g_5\hat{X}_0\hat{X}_1, \tag{2.96}$$

where coefficients g_i are determined by the internuclear distance R. We choose $R = 0.75\,\text{Å}$ and hence $g_0 = 0.2252$, $g_1 = 0.3435$, $g_2 = -0.4347$ $g_3 = 0.5716$, $g_4 = 0.0910$, $g_5 = 0.0910$. We make use of the so-called "hardware efficient ansatz" [118] with $P = 8$ parameters of simple rotation gates ($L = 1$), which we label $\hat{U}^{(\mathrm{H}_2)}$ and draw in Figure 2.6.

Meanwhile, the Lithium Hydride (LiH) molecule is considered in the STO-3G basis. The three electrons of the Li atom populate the $1S$, $2S$, $2P_x$, $2P_y$ and $2P_z$ orbitals. In combination with the H's $1S$ orbital, the LiH molecule is thusly described by twelve spin orbitals in this basis. This can be compressed to eight spin orbitals by asserting

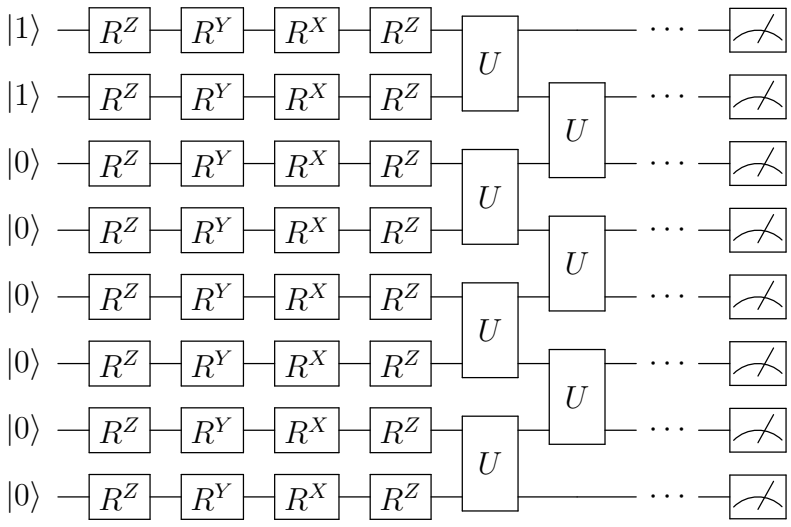


Figure 2.7: The general structure of the ansatz $\hat{U}^{(\mathrm{LiH})}$ used in minimisation of the Lithium Hydride electronic structure Hamiltonian. The form of the U gate is $U = e^{i\alpha YX}e^{i\beta XY}e^{i\gamma ZZ}e^{i\delta YY}e^{i\epsilon XX}$, and the pictured U columns are repeated 3 times for a total of 137 parameters. This figure is taken from Reference [163] Figure S4.

that two specific orbitals (a spin up and down) together have a unit probability, and another two have a zero probability; this is motivated by their respective high and low occupation numbers in a reduced density matrix analysis [178]. The BK transformation then produces an ($N = 8$)-qubit ($T = 302$)-term Hamiltonian, where our Pauli coefficients correspond to a modelled internuclear separation of $R = 1.45\,\text{Å}$. These steps were carried out by my co-authors using OpenFermion [179], an electronic structure package for transforming computational chemistry problems into forms native to quantum computers. To minimise this Hamiltonian, we use a hardware efficient ansatz $\hat{U}^{(\mathrm{LiH})}$ with nearest neighbouring gates which are simply parametrized ($L = 1$), inspired by the low depth circuit ansatz (LDCA) [180]. We draw this in Figure 2.7.

We begin with a noise-free simulation of variational imaginary-time evolution of $\hat{H}^{(\mathrm{H_2})}$ via ansatz $\hat{U}^{(\mathrm{H_2})}$, and compare it to unrestricted imaginary time evolution. The latter is found through direct matrix exponentiation of the operator $\exp(-\hat{H}^{(\mathrm{H_2})}\tau)$ and repeated

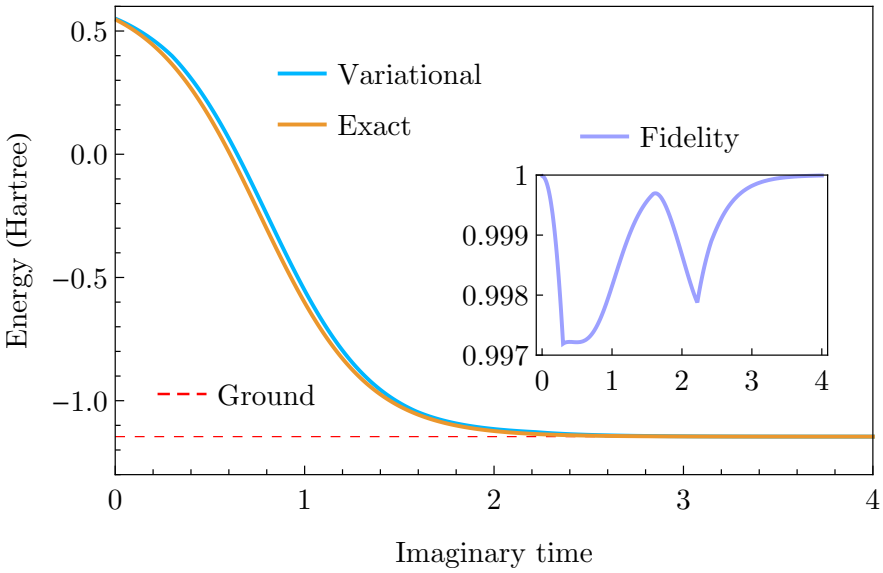


Figure 2.8: Comparison of exact imaginary time evolution $\exp(-\tau \hat{H}^{(\mathrm{H}_2)}) |\mathrm{in}\rangle$ against the successive states of variational evolution $\hat{U}^{(\mathrm{H}_2)}(\vec{\theta}_\tau) |\mathrm{in}\rangle$. This simulation was performed by co-authors Sam McArdle and Xiao Yuan, and rendered myself for Figure 2 of Reference [163].

renormalisation of evolving state. The results are shown in Figure 2.8. From the same initial state (as produced by a random assignment of the ansatz parameters), variational imaginary-time exhibits the exponential reduction in energy prescribed by exact imaginary-time evolution. This is a promising demonstration, albeit one of a charitably small system.

Next, we perform noise-free simulations of variational imaginary-time evolution of $\hat{H}^{(\mathrm{LiH})}$ via ansatz $\hat{U}^{(\mathrm{LiH})}$, and compare them to quantum gradient descent. Simulations are performed using the statevector optimisations developed in Section 2.6.1. We assign each method its greatest stable timestep Δt, determined as the maximum for which the energy monotonically decreases under evolution. We consider two kinds of starting states; those produced from a random initialisation of the parameters, and those which are randomly perturbed (by at most $\Delta\theta = \pi/50$) from the Hartree–Fock (HF) estimation of the groundstate. Our measures of performance include the total number

out of 1280 independent simulations which have converged to within 1 mHartree of true ground by a given iteration, and the precise proximity to ground of those converged. The results are shown in Figure 2.9, revealing variational imaginary-time significantly outperforms gradient descent as a minimisation routine in these settings, achieving closer proximity to the true ground among the converged states, and orders of magnitude greater convergence rates and converged populations.

With such promising performance, our final investigation is into the noise resilience of variational imaginary-time, and its sampling costs relative to gradient descent. These considerations ultimately determine its utility in near-future error-prone quantum computers. We efficiently integrate shot noise and gate errors into our simulations via Algorithm 2.1. With a fixed per-gate error rate of $\xi = 10^{-4}$, we vary the number of measurements involved in evaluating the variational observables ($\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$). Gradient descent experiences only the error in $\vec{\mathcal{V}}$, while imaginary-time evolution experiences error in both $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$. We count how many minimisations have converged to the groundstate, shown in Figure 2.10. Imaginary-time evolution is shown to perform impressively well in noisy settings, significantly outperforming gradient descent, even when suffering substantially more shot noise. Furthermore, it demands relatively little additional resources. For our utilised ansatz ($D = L = 1$, $P = 137$) and Hamiltonian ($N = 6$, $T = 302$), the total experimental costs of evaluating $\underline{\underline{\mathcal{M}}}$ is $1.7 \times 10^6\, m_{\mathcal{M}}$, and that of $\underline{\mathcal{V}}$ is $6.1 \times 10^6\, m_{\mathcal{V}}$. Figure 2.10 demonstrates, for example, that the 5% convergence rate of $m_{\mathcal{V}} = 10^6$ gradient descent can be improved to 96% by adopting $m_{\mathcal{M}} = 5 \times 10^4$ additional measurements per imaginary-time observable; a total resource increase of a mere 1.4%. We happily note that a subsequent work would analytically prove the noise resilience of variational imaginary time evolution suggested by our numerical results [181].

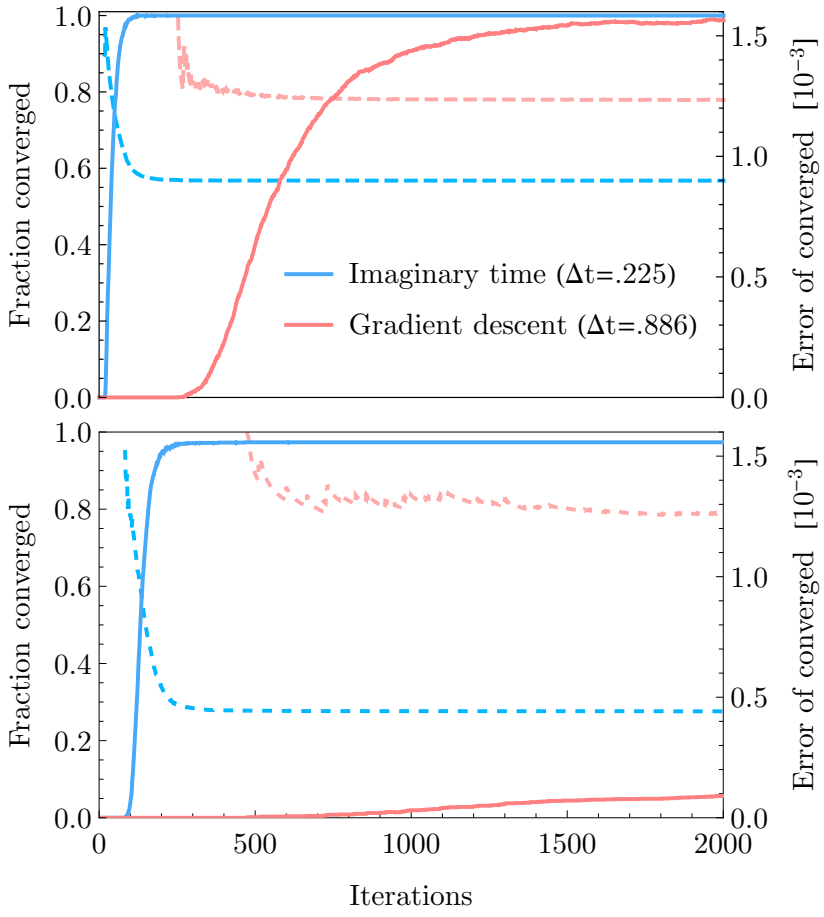


Figure 2.9: Variational imaginary-time evolution (blue) and gradient descent (pink) minimising $\hat{H}^{(\mathrm{LiH})}$ with noise-free ansatz $\hat{U}^{(\mathrm{LiH})}$ of Figure 2.7, each with their maximum stable timestep. Simulations in the top plot begin with parameters randomly deviating from those which produce the Hatree-Fock stock (and hence start *close* to the solution), while those of the bottom plot begin totally randomly. The solid lines (against the left axis) indicate the fraction of 1280 simulations which, by the given iteration, have converged to within 1 mHartree of the true ground state. The dashed lines (against the right axis) indicate the average proximity to the true ground state of only the so-far converged simulations.

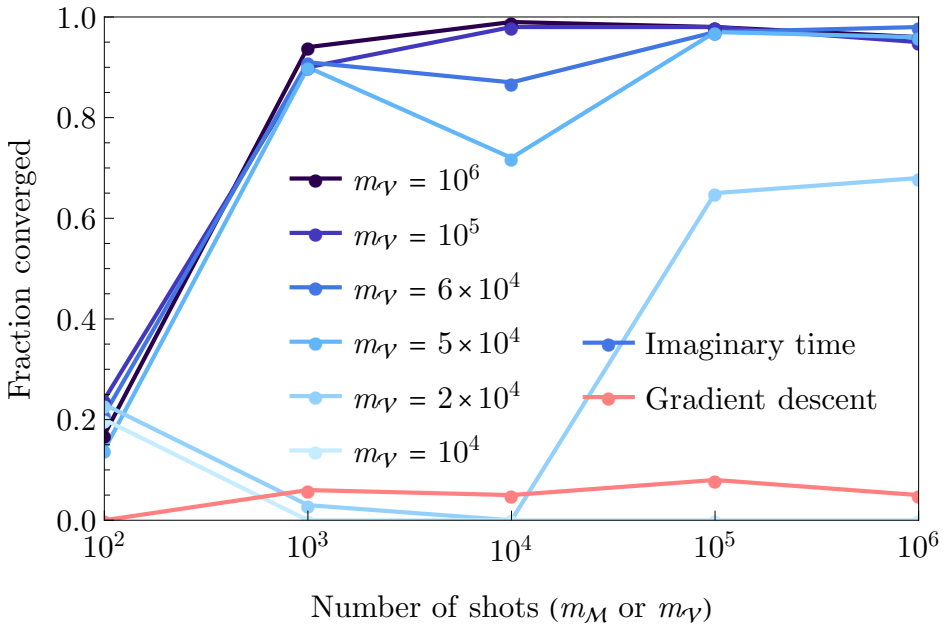


Figure 2.10: Simulated minimisation of $\hat{H}^{(\mathrm{LiH})}$ via noisy variational imaginary time evolution with ansatz $\hat{U}^{(\mathrm{LiH})}$, in the presence of a fixed 10^{-4} error rate per gate, and varying shot noise. Each point indicates the fraction of 100 trials which, after 2000 iterations from uniformly random initial parameter states, finished within 1 mHartree of the true ground state. For imaginary time (the blue lines), the horizontal axis indicates the number of shots $m_{\mathcal{M}}$ used in sampling each *element* of observable matrix $\underline{\underline{\mathcal{M}}}$, every iteration. Estimation of the gradient $\vec{\mathcal{V}}$ also experienced shot noise during imaginary-time, as per the legend. For gradient descent, the horizontal axis is $m_{\mathcal{V}}$. As prescribed by $\hat{H}^{(\mathrm{LiH})}$ and $\hat{U}^{(\mathrm{LiH})}$, the total costs (number of quantum gates) of evaluating $\underline{\underline{\mathcal{M}}}$ is $1.7 \times 10^6\, m_{\mathcal{M}}$, and that of $\underline{\mathcal{V}}$ is $6.1 \times 10^6\, m_{\mathcal{V}}$.

2.8 Discussion

This chapter derived a method to variationally simulate imaginary-time evolution using a hybrid classical-quantum computer. Inspired by Li's method for real-time simulation, it involves a quantum processor iteratively evaluating a polynomial number of observables (which relate to energy derivatives), and a classical processor solving a polynomially large underspecified linear equation, using regularisation. We rigorously established the quantum resource costs to perform a single iteration. Our algorithm was then demonstrated to be an excellent variational minimisation technique, greatly outperforming canonical quantum gradient descent on modest quantum chemistry problems. This was ascertained through classical emulation of the algorithm, specifically minimising electronic structure Hamiltonians of Hydrogen gas and the Lithium Hydride molecule. We observed impressive shot noise resilience, which was later analytically verified. These conducted tests were only possible through our development of a novel classical simulation protocol, which propagated the effects of depolarising and shot noise into the probability distributions of the measured variational observables, which were then pseudorandomly sampled.

Through this chapter, we also discovered that Li's method can be improved by the addition of a global phase parameter, which when introduced into variational imaginary-time, was proven to yield the subsequently developed quantum natural gradient algorithm. The impressive success of variational imaginary-time evolution secured its place as the minimisation subroutine of choice for the succeeding research of this thesis.

Chapter 3

Quantum Diagonalisation

Contents

3.1 Foreword

This chapter presents the published manuscript

- **T. Jones** and S. Endo, S. McArdle, X. Yuan, S. C. Benjamin
  Variational quantum algorithms for discovering Hamiltonian spectra
  Phys. Rev. A **99**, 062304 (2019) (link)

which devises a novel quantum algorithm to perform Hamiltonian diagonalisation. The principle of the algorithm was suggested by my co-author Suguru Endo and evolved under our joint investigation. The numerical research of this chapter is my own.

Much of this work was completed in parallel with the research of Reference [163] as presented in the previous chapter. This meant several of its research revelations were more naturally placed in the former manuscript and chapter, simplifying the investigation of the present chapter. Furthermore, we note with humour that much of the collaborative work between Suguru Endo and myself involved disproving the other's attempted analytic statements about the presented method. The result is a work heavy in numerics, and one which can be viewed as an interesting application of variational imaginary time minimisation, and a comparison of variational heuristics, ansatz circuits and minimisers.

3.2 Introduction

Chapter 2 introduced variational imaginary time minimisation to determine the ground-state energy of a given Hamiltonian. However, many physical properties of a quantum system are determined by more information about its energy spectrum [182], such as how molecular spectra inform their dynamics [183] and reaction rates [184]. Classically diagonalising a quantum Hamiltonian is harder still than finding merely its minimum, which we already saw to be exponentially and prohibitively expensive. This precludes the precise study of many interesting chemical compounds [185]. But just as they might for the ground-state problem, quantum computers may have the potential to accelerate the discovery of higher excited states.

At the time of this chapter's work, the utilisation of a quantum computer to find the higher excited states of a problem or chemical Hamiltonian had received relatively little attention. A handful of proposals based on quantum eigensolving, such as the quantum subspace expansion method [186, 187] and the von-Neumann entropy method [188], required either many measurements, or deep quantum circuits, for instance to perform quantum phase estimation as a subroutine. This challenged their ultimate compatibility with near-future quantum computers.

This chapter presents a novel quantum variational algorithm to sequentially calculate the energy eigenvalues of a given Hamiltonian. The algorithm employs the imaginary-time minimisation technique presented in Section 2 as a sub-routine, and makes use of the shallow swap test [189, 190] to evaluate the overlap of two quantum states [191]. Our method makes use of only shallow circuits, at the cost of additional measurements, and can leverage a simple error detection routine to mitigate qubit-flip errors when used in quantum chemistry settings.

During preparation of our original manuscript, a similar work by Higgott *et al* [191] introduced the use of the swap test with variational minimisation to discover the energy eigenstates of the diatomic Hydrogen molecule. We additionally contrast the performance of such methods based on direct descent with our imaginary time approach, and find that the former is prone to becoming stuck in non-physical local minima of the parameter space [192, 163]. This likely renders them unsuitable for probing the full spectra of even modestly bigger systems, which we numerically demonstrate with a 6-qubit molecular Hamiltonian.

In contrast, we find that variational imaginary time evolution tends to converge to energy eigenstates of the Hamiltonian, regardless of the initial state. This mechanism proves crucial for our algorithm to reliably discover the physical energy spectrum. We test our method on 18-qubit Hamiltonians which encode the boolean satisfiability problem (3SAT), and to find the electronic spectrum of the previously seen Lithium Hydride (LiH) molecule when encoded into both 6 and 10 qubit Hamiltonians, with varying interatomic distances.

3.3 Derivation

This illustrative section derives the quantum components of our algorithm, almost all of which were originally performed by my coauthors (primarily Suguru Endo), and the cited references. My contributions herein are only the formalisation of the method, and motivation of the penalisation parameter. All text below is my own.

3.3.1 Energy penalisation

We here derive the simple mechanism core to our algorithm. Just like in variational imaginary-time minimisation of Chapter 2, assume we have chosen an ansatz circuit $\hat{U}(\vec{\theta})$ with a tractable set of P real parameters $\vec{\theta}$, which produces state $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\text{in}\rangle$ when applied to a fixed input state $|\text{in}\rangle$. Recall that under variational imaginary time evolution, the parameters are repeatedly modified as

$$\underline{\underline{\mathcal{M}}}\;\Delta\vec{\theta} = -\Delta t\,\vec{\mathcal{V}}, \qquad \text{(re-expression of 2.21)}$$

where every iteration, measurements inform the classical quantities

$$\mathcal{M}_{ij} = \Re\left[\frac{\partial\,\langle\psi(\vec{\theta}_\tau)|}{\partial\theta_i}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial\theta_j}\right], \qquad \text{(reiteration of 2.22)}$$

$$\mathcal{V}_i = \Re\left[\langle\psi(\vec{\theta}_\tau)|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta}_\tau)\rangle}{\partial\theta_i}\right], \qquad \text{(reiteration of 2.23)}$$

and the ansatz state approaches the ground-state $|e_0\rangle$ of the Hamiltonian $\hat{H} = \sum_{i=0} e_i\,|e_i\rangle\,\langle e_i|$.

We begin with a hypothetical. Imagine if it were possible to produce the augmented Hamiltonian

$$\hat{H}' = \hat{H} + \alpha\,|e_0\rangle\,\langle e_0|\,, \qquad (3.1)$$

where scalar $\alpha > 0$ is sufficiently large such that $|e_0\rangle$ is *not* the ground-state of $\hat{H}'$. Instead, the state $|e_1\rangle$ has become the ground state of $\hat{H}'$, and $|e_0\rangle$ is an excited state of $\hat{H}'$ with an energy eigenvalue $e_0 + \alpha$. This would suggest that imaginary time evolution under $\hat{H}'$ would see $|e_0\rangle$ decay exponentially faster than $|e_1\rangle$ while the rest of the spectrum, orthogonal to $|e_0\rangle$, is unaffected. The evolving ansatz state would thus

approach $|e_1\rangle$;

$$\text{Under } \underline{\underline{\mathcal{M}}}^{-1}\vec{\mathcal{V}}\,|_{\hat{H}\to\hat{H}'}, \qquad \lim_{\tau\to\infty}|\psi(\vec{\theta}_\tau)\rangle \sim |e_1\rangle\,. \qquad \text{(adaptation of 2.4)}$$

Imagine next that the first excited state $|e_1\rangle$ was subsequently "excited" in the Hamiltonian like the original groundstate, effecting

$$\hat{H}'' = \hat{H} + \alpha\,|e_0\rangle\langle e_0| + \alpha\,|e_1\rangle\langle e_1|\,. \tag{3.2}$$

We might expect imaginary-time evolution under $\hat{H}''$ to converge to the next low-lying eigenstate state of the original Hamiltonian, $|e_3\rangle$. If this process were repeated n times, then the n-th augmented Hamiltonian

$$\hat{H}'^{(n)} = \hat{H} + \sum_{j=0}^{n-1}\alpha\,|e_j\rangle\langle e_j|\,, \qquad \alpha > e_n - e_0, \tag{3.3}$$

would drive imaginary-time evolution into the n-th eigenstate $|e_n\rangle$. The energy of each converged ansatz state can be measured to discover an energy eigenvalue. In-principle, one could obtain the full energy spectrum of $\hat{H}$ in this manner. Naturally we have no *a priori* knowledge of the eigenstates nor how to construct their projectors. But if variational minimisation successfully converges into an eigenstate, we at least obtain a tractable parameter set $\vec{\lambda}$ through which to later reproduce it. Concretely, perfect minimisation under $\hat{H}'^{(n)}$ produces parameters $\vec{\lambda}^{(n)} = \lim_{\tau\to\infty}\vec{\theta}_\tau$ which we can use to later generate the ansatz state $|\psi(\vec{\lambda}^{(n)})\rangle = |e_n\rangle$. This concludes the exercise of our imaginations.

Being able to experimentally reproduce an eigenstate $|e_n\rangle$ through an ansatz circuit $|\psi(\vec{\theta})\rangle$ (using parameters $\vec{\lambda}^{(n)}$ admitted by perfect minimisation) does not obviate the

challenges of then effecting projector $|e_n\rangle\langle e_n| = |\psi(\vec{\lambda}^{(n)})\rangle\langle\psi(\vec{\lambda}^{(n)})|$ in the augmented Hamiltonian. Indeed, directly modifying the Hamiltonian would require exponentially-costly tomography [193] of each converged state. Fortunately, we need not directly modify the Hamiltonian. Observe that substitution of $\hat{H} \rightarrow \hat{H}'^{(n)}$ affects only the classical vector $\vec{\mathcal{V}}$ evaluated each iteration of imaginary time evolution. Showing $n = 1$ for clarity,

$$\mathcal{V}_i\Big|_{\hat{H}\rightarrow\hat{H}'} = \Re\Bigg(\frac{\partial\langle\psi(\vec{\theta})|}{\partial\theta_i}\hat{H}|\psi(\vec{\theta})\rangle \;+\; \alpha\frac{\partial\langle\psi(\vec{\theta})|}{\partial\theta_i}|e_0\rangle\langle e_0|\psi(\vec{\theta})\rangle\Bigg) \tag{3.4}$$

$$= \mathcal{V}_i \;+\; \alpha\,\frac{1}{2}\frac{\partial}{\partial\theta_i}|\langle\psi(\vec{\theta})|e_0\rangle|^2 \tag{3.5}$$

$$\simeq \mathcal{V}_i \;+\; \frac{\alpha}{2\,\Delta\theta}\Big(|\langle\psi(\vec{\theta}|_{\theta_i\rightarrow\theta_i+\Delta\theta})|e_0\rangle|^2 \;-\; |\langle\psi(\vec{\theta})|e_0\rangle|^2\Big), \tag{3.6}$$

where $\Delta\theta$ is some sufficiently small change in parameter θ_i. We have employed a first order finite difference order approximation to express the additional term introduced into $\vec{\mathcal{V}}$ as a difference in "overlaps" $|\langle\psi(\vec{\theta}|_{\theta_i\rightarrow\theta_i+\Delta\theta})|e_0\rangle|^2$ and $|\langle\psi(\vec{\theta})|e_0\rangle|^2$, where $|e_0\rangle = |\psi(\vec{\lambda}^{(0)})\rangle$ is an ansatz state. Naturally, higher order approximations can be used, admitting more terms [174]. These quantities can each be evaluated by a quantum computer using the low depth swap test circuit [190, 189] or an alternative circuit which permits a rudimentary error mitigation strategy [194] outlined in the next section. As such, Equation 3.6 reveals a method to augment the effected Hamiltonian during experimental implementation of variational imaginary time evolution on a quantum computer. In essence, we successively obtain ansatz states close to eigenvectors and effectively energetically penalise them through incorporating additional quantum measurements into our variational equations. We formalise this method in Algorithm 3.1.

The hyperparameter α is ideally chosen to be as large as the gap between the ground and highest excited state sought, to translate discovered states beyond the sought band

Algorithm 3.1: Computing the modified variational energy gradient $\vec{\mathcal{V}}_i\,'$, under energy penalisation of previously discovered eigenstates, given the original gradient $\vec{\mathcal{V}}$. This algorithm is invoked at every iteration of imaginary time evolution.

1. Let $\vec{\theta}$ be the current ansatz parameter-set during this iteration of imaginary time evolution.
2. Let $|\psi(\vec{\phi})\rangle$ be the ansatz state produced by applying the ansatz circuit, with gate parameters set to $\vec{\phi}$, upon some fixed input state.
3. Let $\vec{\mathcal{V}}$ be the energy gradient at the current $\vec{\theta}$, as computed without the introduction of any excited terms (i.e. Equation 2.22).
4. Let $\{\vec{\lambda_k}\}$ be the list of ansatz parameter-sets which each produce one of the previously converged states (assumed to be energy eigenstates).
5. Let α and $\Delta\theta$ be sufficiently large and small (respectively) real hyperparameters.
6. Set $\vec{\mathcal{V}}\,' = \vec{0}$.
7. For every parameter-set $\vec{\lambda}$ in $\{\vec{\lambda_k}\}$ ...
   - 7.1. Prepare state $|\psi(\vec{\lambda})\rangle$ in quantum register 1.
   - 7.2. Prepare state $|\psi(\vec{\theta})\rangle$ in quantum register 2.
   - 7.3. Measure $x = |\langle\psi(\vec{\theta})|\psi(\vec{\lambda})\rangle|^2$ via the swap-test on registers 1 and 2.
   - 7.4. For every parameter θ_i in parameter-set $\vec{\theta}$...
     - 7.4.1. Re-prepare state $|\psi(\vec{\lambda})\rangle$ in quantum register 1.
     - 7.4.2. Prepare state $|\psi(\vec{\theta}|_{\theta_i \to \theta_i + \Delta\theta})\rangle$ in quantum register 2.
     - 7.4.3. Measure $y = |\langle\psi(\vec{\theta}|_{\theta_i \to \theta_i + \Delta\theta})|\psi(\vec{\lambda})\rangle|^2$ via the swap-test on registers 1 and 2.
     - 7.4.4. Set $\vec{\mathcal{V}}_i\,' = \vec{\mathcal{V}}_i\,' + \frac{\alpha}{2\Delta\theta}(y - x)$.
8. Return $\vec{\mathcal{V}}_i\,'$.

and prevent their rediscovery. This gap can often be a-priori estimated for quantum chemistry Hamiltonians using efficient classical methods [195]. Keep in mind that α appears only in the classical component of variational minimisation, and can be changed at any stage during the algorithm to adjust the effective augmented spectrum or the search space. Further, there is no requirement that each discovered eigenstate is excited by the same amount in the modified Hamiltonian; should we wish to, we may vary α for each excited state. In that scenario, one could add

$$+\sum_{j}^{n} \alpha_j \, \Re\left(\frac{\partial \, \langle \psi(\vec{\theta}(\tau)) |}{\partial \theta_i} \, |e_j\rangle \, \langle e_j | \psi(\vec{\theta}) \rangle \right) \tag{3.7}$$

to the un-penalised $\mathcal{V}_i$ as per Equation 3.4, to effect a Hamiltonian

$$\hat{H}'^{(n)} = \sum_{j}^{n} (e_j + \alpha_j) \, |e_j\rangle \, \langle e_j| \; + \; \sum_{j=n} e_j \, |e_j\rangle \, \langle e_j| \, , \tag{3.8}$$

with modulable spectrum $\{e_0 + \alpha_0, \; \ldots, \; e_n + \alpha_n, \; e_{n+1}, \; \ldots\}$. This may prove a useful utility in adaptations of Algorithm 3.1 to real-time simulation settings, as we elaborate upon in Section 3.6.

Armed with a quantum algorithm to effectively augment Hamiltonians by energetically penalising previously converged ansatz states, we can proceed to iteratively discover (and subsequently penalise) approximate energy eigenstates. We present our scheme in Algorithm 3.2.

3.3.2 State overlap

This section *very* rapidly outlines the process of performing the aforementioned swap test to compute the *overlap* $|\langle\psi|\phi\rangle|^2$ of states $|\psi\rangle$ and $|\phi\rangle$, as stored in separate quan-

Algorithm 3.2: Discovering parameter-sets which produce approximate eigenstates from the ansatz circuit. Each output parameter-set $\vec{\lambda}_k$ can be used to produce an approximate eigenstate, $|e_k\rangle \approx \hat{U}(\vec{\lambda}_k)\,|\text{in}\rangle$, via the ansatz circuit $\hat{U}$, and in-turn, have its energy measured.

1. Let N be the number of sought excited eigenstates.
2. Perform imaginary-time minimisation as per Chapter 2.
3. Set $\vec{\lambda}_0$ to the converged ansatz parameter-set $\vec{\theta}$.
4. For k in $\{1, \ldots, N\}$ ...
   - 4.1. Randomise the ansatz parameter-set $\vec{\theta}$.
   - 4.2. Perform imaginary-time minimisation, where for every iteration ...
     - 4.2.1. Evaluate $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ as per Section 2.4.2.
     - 4.2.2. Produce $\vec{\mathcal{V}}'$ from $\vec{\mathcal{V}}$ via Algorithm 3.1, passing parameter-set list $\{\vec{\lambda}_0, \ldots, \vec{\lambda}_{k-1}\}$.
     - 4.2.3. Update $\vec{\theta}$ as per Section 2.4.4, substituting $\vec{\mathcal{V}}$ for $\vec{\mathcal{V}}'$.
   - 4.3. Set $\vec{\lambda}_k$ to the converged parameter-set $\vec{\theta}$.
5. Return $\{\vec{\lambda}_0, \ldots, \vec{\lambda}_N\}$.

tum registers. We mention Buhrman *et al*'s original non-destructive swap-test [196] and Carlos *et al*'s fixed-depth destructive swap-test [190]. Our algorithm may employ either, although the former albeit more expensive method enables rudimentary error mitigation. Since untested in this work, we mention it only in passing.

The non-destructive swap test repeatedly performs the circuit of Figure 3.1, sampling the expected ancilla outcome of $\frac{1}{2} - \frac{1}{2}|\,\langle\phi|\psi\rangle\,|^2$. The pictured controlled-SWAP gate is a simple sequence of controlled two-qubit SWAP gates (i.e. *Fredkin* gates [197]) between corresponding qubits in each register; hence the circuit *depth* grows linearly with the number of qubits. Thereafter, the two registers together remain in state

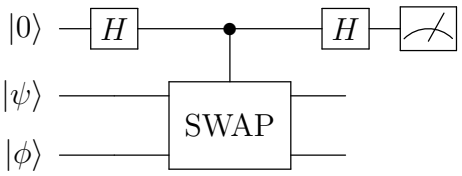


Figure 3.1: The non-destructive swap-test circuit [196].

$$|\Psi\rangle = \frac{1}{\sqrt{2}} \left(|\phi\rangle |\psi\rangle \pm |\psi\rangle |\phi\rangle \right), \tag{3.9}$$

with sign determined by the ancilla measurement outcome, and are now unneeded by the core algorithm. But they may remain useful; if input states $|\psi\rangle$ and $|\phi\rangle$ are invariant under some symmetry, so too is $|\Psi\rangle$. If a final measurement upon $|\Psi\rangle$ reveals this symmetry is violated, then it indicates an error has occurred during the preparation of $|\psi\rangle$ and $|\phi\rangle$, and/or the swap-test, and this swap-test sample should be discarded. Such symmetries are common of the ansatz states used in quantum variational algorithms for molecular Hamiltonians, like the unitary coupled cluster (UCC) ansatz's conservation of encoded electron populations [198].

More general ansätze without such symmetries should forego the non-destructive swap-test for a destructive fixed-depth formulation [190]. The circuit, presented in Figure 3.2, contains no ancilla nor SWAP gates, and instead features controlled-NOT gates between disjoint pairs of qubits. Each final pair-wise measurement contributes a factor $m_i = \pm 1$ (-1 if both qubits are in outcome 1, else $+1$) to a final product over all pairs, which yields value 1 with probability $\frac{1}{2} - \frac{1}{2} |\langle\phi|\psi\rangle|^2$. This again offers a method to compute the overlap through repeated quantum sampling.

3.4 Simulation

We investigate the performance of our algorithm through classical simulation of its deployment upon a perfect (that is, fault tolerant) quantum computer. We do not study more realistic noisy settings because the classical facilities for doing so (the algorithm of Section 2.6.2, and the high-performance numerical tools of Chapter 6) had not yet been developed at the time of this research. We are instead restricted to the bespoke

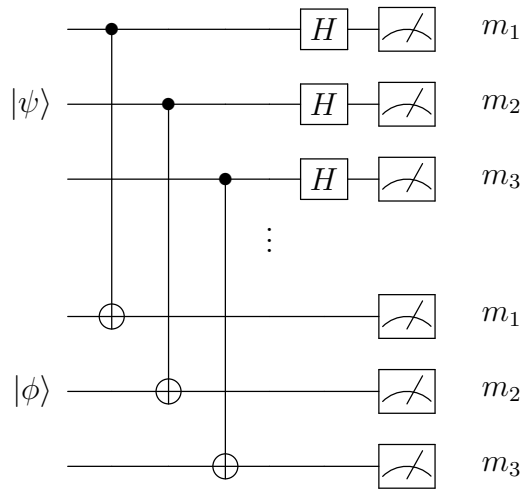


Figure 3.2: The destructive fixed-depth swap-test circuit [190].

methods for pure statevector simulation described in Section 2.6.1. This present section will motivate the Hamiltonians and ansätze selected for our classical simulations.

3.4.1 LiH

Our tests of variational imaginary-time evolution in Section 2.7 included solving the electronic structure problem of the Lithium Hydride (LiH) molecule at a fixed interatomic spacing of $R = 1.45$Å, encoded as an 8-qubit 302-term Pauli Hamiltonian $\hat{H}_8^{(\mathrm{LiH})}$. In this work, we will explore two related chemical Hamiltonians; a compression (exploiting symmetries in the Bravyi–Kitaev transformation [177]) to a 6-qubit 401-term Hamiltonian $\hat{H}_6^{(\mathrm{LiH})}$, and a 10-qubit 631-term encoding, $\hat{H}_{10}^{(\mathrm{LiH})}$. This latter Hamiltonian preserves the higher excited states which are corrupted by the previous compressions, and allows us to vary the interatomic separation to probe a meaningful continuous family of Hamiltonians. These Hamiltonians were produced by my co-authors using OpenFermion [179]

3.4.2 3SAT

The spectra of $\hat{H}_{10}^{(\mathrm{LiH})}(R)$ over interatomic separation R are non-analytic and known to be somewhat complicated, with level-crossings and varying measures of entanglement across even the groundstate eigenvector. As a simpler initial check of our algorithm, we probe Hamiltonians which are diagonal in the $\hat{Z}$-basis and which have analytically known, integer-spaced, highly-degenerate eigenvalues. They encode *Boolean satisfiability* problems, specifically 3SATs [199], using the prescription of Reference [200] to which the author made a minor contribution.

The Boolean satisfiability problem is a central problem in classical computer science, and involves finding a satisfying assignment of variables constrained in a propositional formula [199]. For 3SAT, this formula is a set of clauses, each consisting of three terms, which are Boolean variables with or without negation. A clause is satisfied by containing at least one true term, and every clause must be simultaneously satisfied to satisfy the formula. For example, the four-variable seventeen-clause formula

$$
\begin{array}{rcrc}
(a \text{ or } \neg b \text{ or } c) & \text{and} & (d \text{ or } a \text{ or } \neg b) & \text{and} \\
(\neg a \text{ or } b \text{ or } c) & \text{and} & (d \text{ or } \neg b \text{ or } \neg c) & \text{and} \\
(\neg d \text{ or } \neg a \text{ or } c) & \text{and} & (\neg d \text{ or } b \text{ or } \neg c) & \text{and} \\
(d \text{ or } b \text{ or } c) & \text{and} & (d \text{ or } a \text{ or } b) & \text{and} \\
(d \text{ or } \neg a \text{ or } \neg b) & \text{and} & (\neg d \text{ or } \neg a \text{ or } b) & \text{and} \\
(d \text{ or } \neg a \text{ or } \neg c) & \text{and} & (d \text{ or } b \text{ or } \neg c) & \text{and} \\
(d \text{ or } \neg a \text{ or } b) & \text{and} & (\neg a \text{ or } \neg b \text{ or } \neg c) & \text{and} \\
(\neg d \text{ or } a \text{ or } c) & \text{and} & (d \text{ or } \neg a \text{ or } c) & \text{and} \\
(\neg a \text{ or } \neg b \text{ or } c) & & &
\end{array}
$$

is uniquely satisfied by variable assignment $a = 0$ (or *False*), $b = 1$ (or *True*), $c = 1$ and $d = 1$. Finding a satisfying assignment to a 3SAT problem is NP-complete [199], and we restrict ourselves to problems with unique clauses, where each variable within a clause is unique, and which admit a single satisfying solution; this motivates our approximate ratio ≈ 4.267 of total clauses to total variables [201]. These are the 3SAT problems which, in some sense, are *hardest* to solve, although the only relevance to our application here is its non-degenerate groundstate.

We build a diagonal Hamiltonian $\hat{H}^{(\mathrm{SAT})}$ from a 3SAT formula by treating each computational basis state as a Boolean assignment and energetically penalising it by the number of clauses it fails to satisfy. Each Boolean variable maps to the basis states of a qubit, producing a Hamiltonian with a unique zero-energy groundstate (encoding the 3SAT solution), and highly-degenerate integer eigenvalues. Ergo, groundstate $|abcd\rangle = |0111\rangle$ has energy 0 of the example 3SAT above, and state $|0000\rangle$ has energy 2 since it violates two clauses; (d or b or c) and (d or a or b).

In our classical simulations below, we will merely count clause failures to form $\hat{H}^{(\mathrm{SAT})}$ and measure energy as a real-weighted sum of a state's amplitudes (squared). In principle, a quantum experiment may efficiently realise these Hamiltonians using a number of $\hat{Z}$ operators linear in the number of clauses [200], leveraging $|0\rangle\langle 0| = (\hat{\mathbb{1}} + \hat{Z})/2$ and $|1\rangle\langle 1| = (\hat{\mathbb{1}} - \hat{Z})/2$ to produce a clause-penalisation operator:

$$|\varphi\rangle\langle\varphi| = \frac{1}{2^3}\left(\hat{\mathbb{1}} + \sum_v n_v Z_{q_v} + \sum_{v,u} n_u Z_{q_v} Z_{q_u} + \prod_v n_v Z_{q_v}\right), \tag{3.10}$$

where $n_v = \pm 1$ encode the clause's variable negations. This would permit them to minimise $\hat{H}^{(\mathrm{SAT})}$ to drive toward a classical solution, obtaining the satisfying Boolean assignment. We illustrate such a protocol on a 29-qubit Hamiltonian in Figure 3.3.

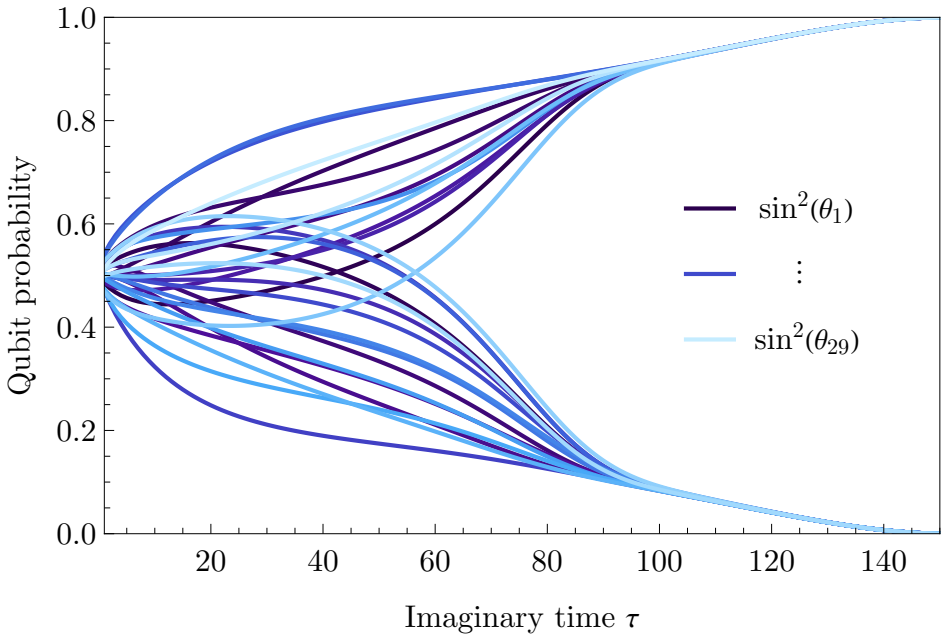


Figure 3.3: Iterative minimisation of a 29-qubit 124-clause 3SAT Hamiltonian with a unique groundstate. Parameter θ_i informs the angle between outcomes of a non-orthogonal projective measurement upon qubit i, as per Reference [200].

There is evidence an iterative approach using projective measurements [200] could outperform Schöning's classical solver [202], but we stress this is beyond the scope of our investigation here. Our quantum diagonalisation algorithm is *not* suggested as an efficient 3SAT solver; we employ them only as simply structured examples. Our study employs $\hat{H}_n^{(\mathrm{SAT})}$ of up to $(n = 18)$ qubits, the maximum tractable under the restrictive resource costs of classically simulating variational algorithms, as will be substantiated in Chapter 5

3.4.3 Ansatz

The choice of ansatz is very important in variational simulation, and in this work, we explore the use of two. We try the recently proposed and chemically motivated *low depth* circuit ansatz (LDCA) [180], shown to outperform the somewhat canonical

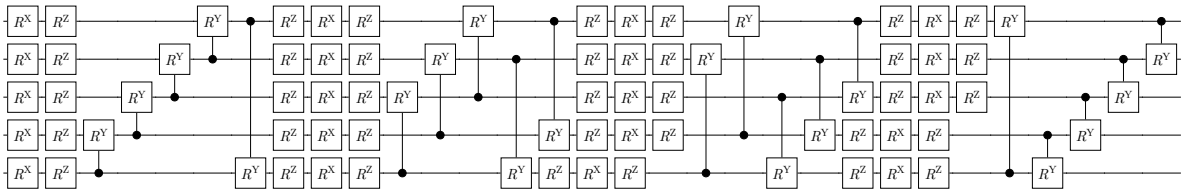


Figure 3.4: A 5-qubit example of the low depth circuit ansatz (LDCA) [180], of four layers. Notice the gap between the control c and target t qubit of $C_c[R_t^Y]$ widens each layer.

unitary coupled cluster ansatz (UCC) [198]. We draw the LDCA in Figure 3.4. We also re-employ the $\hat{U}^{(\mathrm{LiH})}$ ansatz of Figure 2.7 used in our previous imaginary-time simulations [163], which we subsequently refer to as the *compact ansatz*. Both ansätze scale linearly with the number of qubits, and can be considered hardware efficient [180, 163]. While only a single ansatz need be used in an application, we tested both and found interesting benefits to each, so we compare them in our tests.

3.5 Testing

This section tests our quantum Hamiltonian diagonalisation algorithm through classical simulation, upon the systems described in the previous section. We compare the energy spectrum reported by our algorithm with those found of direct, exponentially expensive classical diagonalisation.

As a preliminary test, we study the regular integer-spaced highly-degenerate spectrum of Hamiltonians $\hat{H}_n^{(3\mathrm{SAT})}$, encoding n-Boolean 3SAT problems. Figure 3.5 present the typical behaviour of our algorithm, for $n = 16$ and 18 qubits. For illustration, we also include an example parameter evolution in Figure 3.6. The core mechanism is successful; reported energies (orange) are indeed true eigenvalues (dashed lines), and the converged states are excited and correctly avoided in subsequent minimisation.

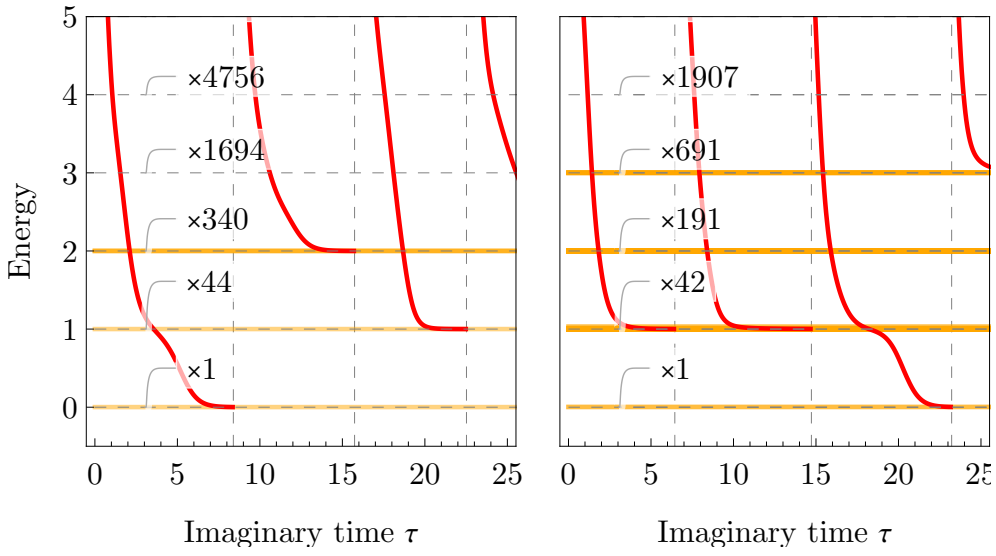


Figure 3.5: Our variational quantum diagonalisation algorithm discovering some low lying eigenstates of the 18 boolean (left) and 16 boolean (right) 3SAT Hamiltonians, using the compact ansatz (with 126 and 112 parameters respectively). Vertical dashed lines indicate iterations when the Hamiltonian was excited as per Eq. 3.3, and the ansatz parameters re-randomised, before resuming imaginary-time minimisation. Horizontal dashed and orange-coloured lines indicate the true eigenvalues and those found by our method respectively. Labels indicate the eigenvalue degeneracy.

But we note an unexpected result; our algorithm does not converge into eigenstates of strictly increasing energy, as prescribed by perfect imaginary-time evolution, even when the initial population of the next lowest-lying eigenstate is non-negligible. This is a violation of Equation 2.4 due to the variational restriction on imaginary evolution. It implies that the gradient $\underline{\underline{\mathcal{M}}}^{-1} \nabla \langle E \rangle$ of the transformed energy manifold tends to contain local minima proximate to Hamiltonian eigenstates, in addition to the global minima near the true groundstate. Converging to eigenstates "out of order" in this way does not jeopardise our algorithm's ability to discover such states, but may preclude its utility in applications where *only* the lowest lying states are sought. The strong deviation of *variational* from *unrestricted* imaginary-time evolution bespeaks a strong sensitivity to the parameter space, highlighting the importance of careful ansatz design.

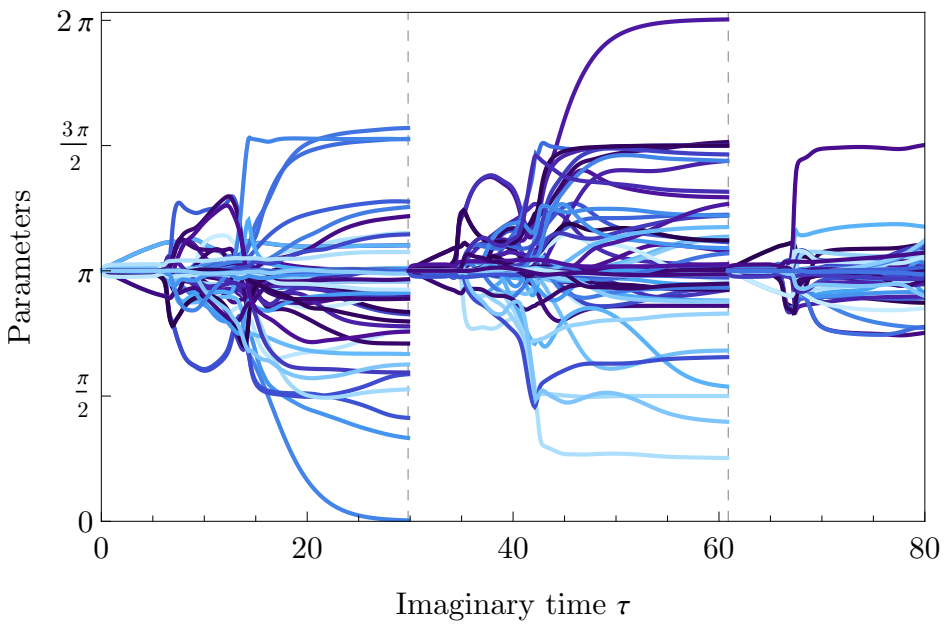


Figure 3.6: An example of the parameter evolution during our variational diagonalisation algorithm. Here, the ansatz parameters $\vec{\theta}$, once converging to $\vec{\lambda}^{(n)}$, are restored to $\theta_i = \pi$ during every penalisation of $\hat{H}'^{(n+1)} = \hat{H}'^{(n)} + \alpha \, |\psi(\vec{\lambda}^{(n)})\rangle \langle\psi(\vec{\lambda}^{(n)})|$, indicated at the dashed vertical lines.

Reassuringly, it may also suggest that even ansatz circuits insufficiently expressive to capture precise evolution may can still minimise toward eigenstates.

We now perform more practical tests; tasking our algorithm with diagonalisation of a compressed Lithium Hydride Hamiltonian $\hat{H}_6^{(\mathrm{LiH})}$. This time we verify that the converged ansatz states are indeed (very close to) eigenstates, and contrast the performance of imaginary-time minimisation with gradient descent. Figure 3.7a shows our algorithm successfully discover the spectrum, and Figure 3.7b shows gradient descent converging to non-eigenstates which when subsequently excited, perturb the Hamiltonian in a non-trivial way. This leads to errors in the discovered spectrum, shown in Figure 3.8.

Next, we study $\hat{H}_{10}^{(\mathrm{LiH})}(R)$ as a function of the interatomic separation R. The results for both ansätze are shown in Figure 3.9. The compact ansatz with 70 parameters shows reasonable agreement with the true spectrum, despite generating only a small

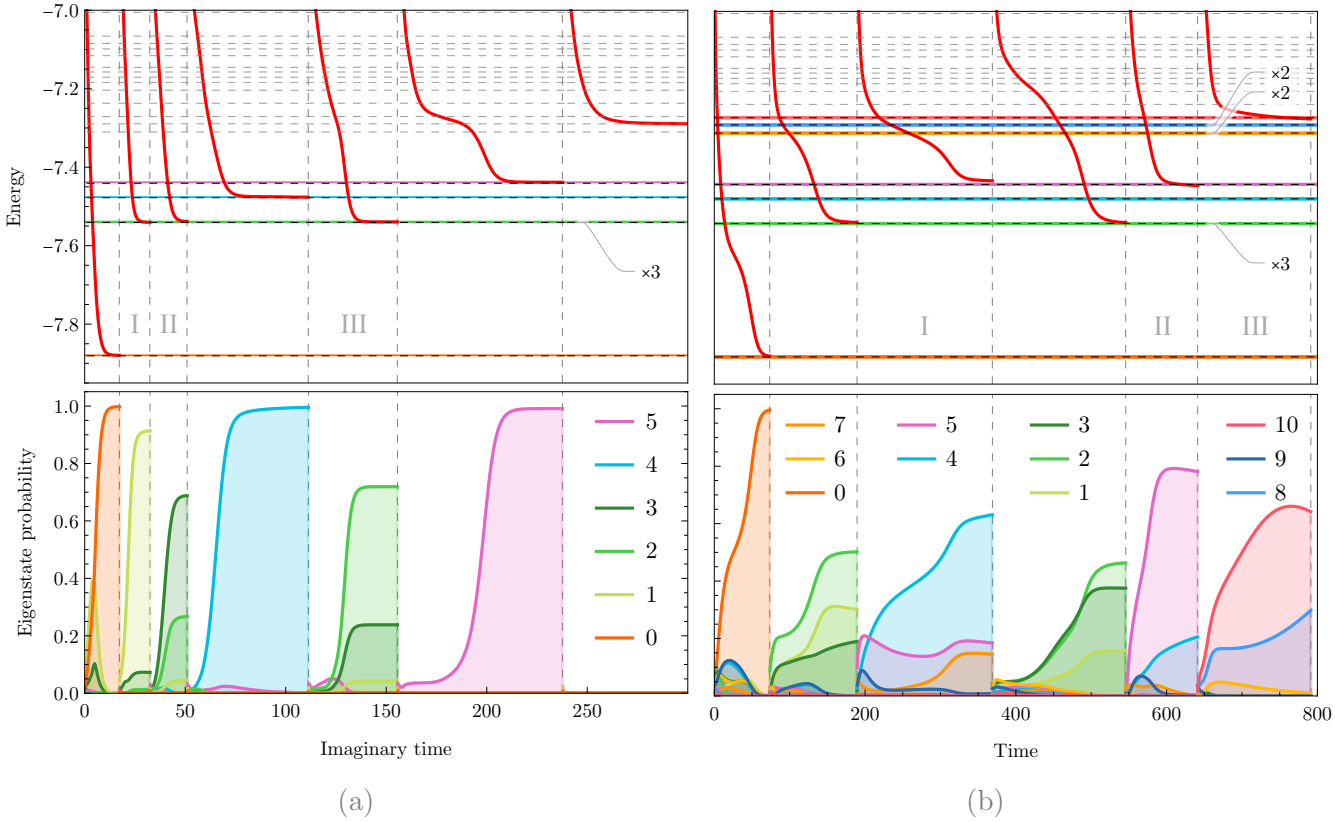


Figure 3.7: Our diagonalisation algorithm (left) discovering then exciting several low lying eigenstates of $\hat{H}_6^{(\mathrm{LiH})}$ in comparison to an alternate formulation [203] using gradient descent (right). Both use the low-depth ansatz [180] of 56 parameters, shown in Figure 3.4. The top plot shows the expected energy (red) of the evolving ansatz state under the latest augmented Hamiltonian; when converging (coloured horizontal lines) to an eigenstate (dashed horizontal lines), the ansatz state is excited (vertical dashed lines). The bottom plot shows the population of the true eigenstates. The ultimate spectra reported by these algorithms is shown in Figure 3.8. (a) **Imaginary time**. Regions I, II and III converge to orthogonal superpositions of the triply degenerate first-excited states (shown in green), which are themselves eigenstates. (b) **Gradient descent**. The green (labelled 1, 2 and 3), orange (labelled 6 and 7) and blue (labelled 8 and 9) states are degenerate. Regions I, II and III show gradient descent converging to non-eigenstates.

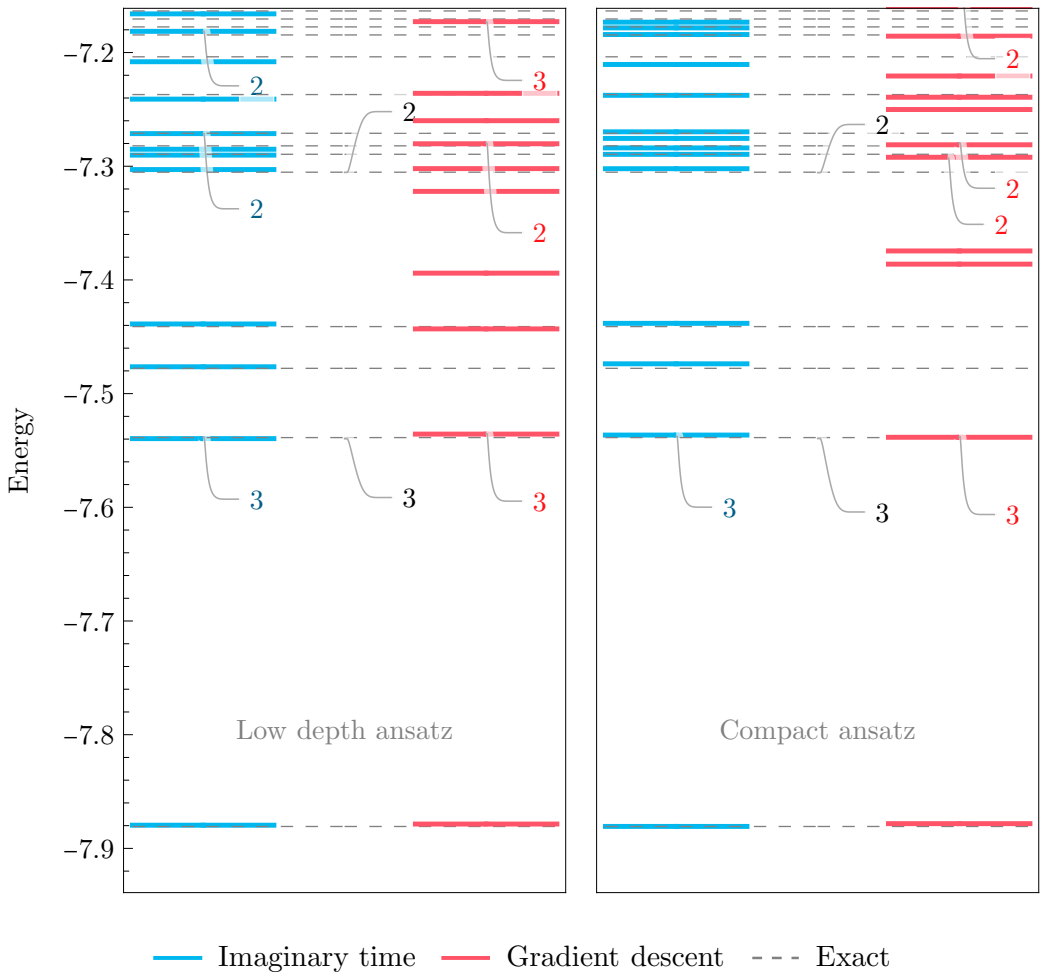


Figure 3.8: The sub-spectra of $\hat{H}_6^{(\mathrm{LiH})}$ discovered by our diagonalisation algorithm, when using variational imaginary-time (blue) and gradient descent (red) as the minimisation subroutine, and two choices of ansatz circuit. Grey lines show the true spectrum. The low depth and compact ansätze use 56 and 42 parameters respectively. The low depth ansatz simulations extend those in Figs. 3.7a and 3.7b. Energies closer than 5×10^{-3} apart are combined and their degeneracy labelled.

submanifold of the full 2^{10} Hilbert space. The smooth deviation of the lowest discovered energy with the true ground energy is understood to result from the ansatz's inability to reliably approximate the ground state. The low depth ansatz with 145 parameters shows a marked improvement in accuracy, and a better discovery of the degenerate energy eigenvalues. Interestingly, both ansätze show decreased accuracy around bond length $l \approx 2.5$Å. This was also seen in recent variational eigensolver experiments [144], and attributed to the insufficient power of the low depth ansatz to generate these anomalously highly entangled eigenstates [118].

3.6 Discussion

This chapter proposed a novel variational algorithm for a hybrid quantum-classical computer to iteratively discover the spectra of classically intractable Hamiltonians. It prescribes repeated quantum minimisation under augmented variational equations which feature additional terms measurable through the simple SWAP test. This means *double* the number of qubits than in the ansatz circuit alone are ultimately required, and an additional ancilla qubit is needed if we wish to leverage ansatz symmetries for error mitigation. The additional terms in the variational equations approximate energy penalisation of the converged ansatz states, and in-turn, shifting of specific eigenvalues in the energy spectrum. We tested our method through classical simulation, using the physical Lithium Hydride electronic structure Hamiltonian, and scalable fictitious Hamiltonians which encode 3SAT problems. We tested our algorithm with both the LDCA [180] and the hardware efficient "compact ansatz" [163], finding the former was superior in all cases. We employed both our variational imaginary-time evolution algorithm of Chapter 2, and canonical quantum gradient descent [118] (as per the par-

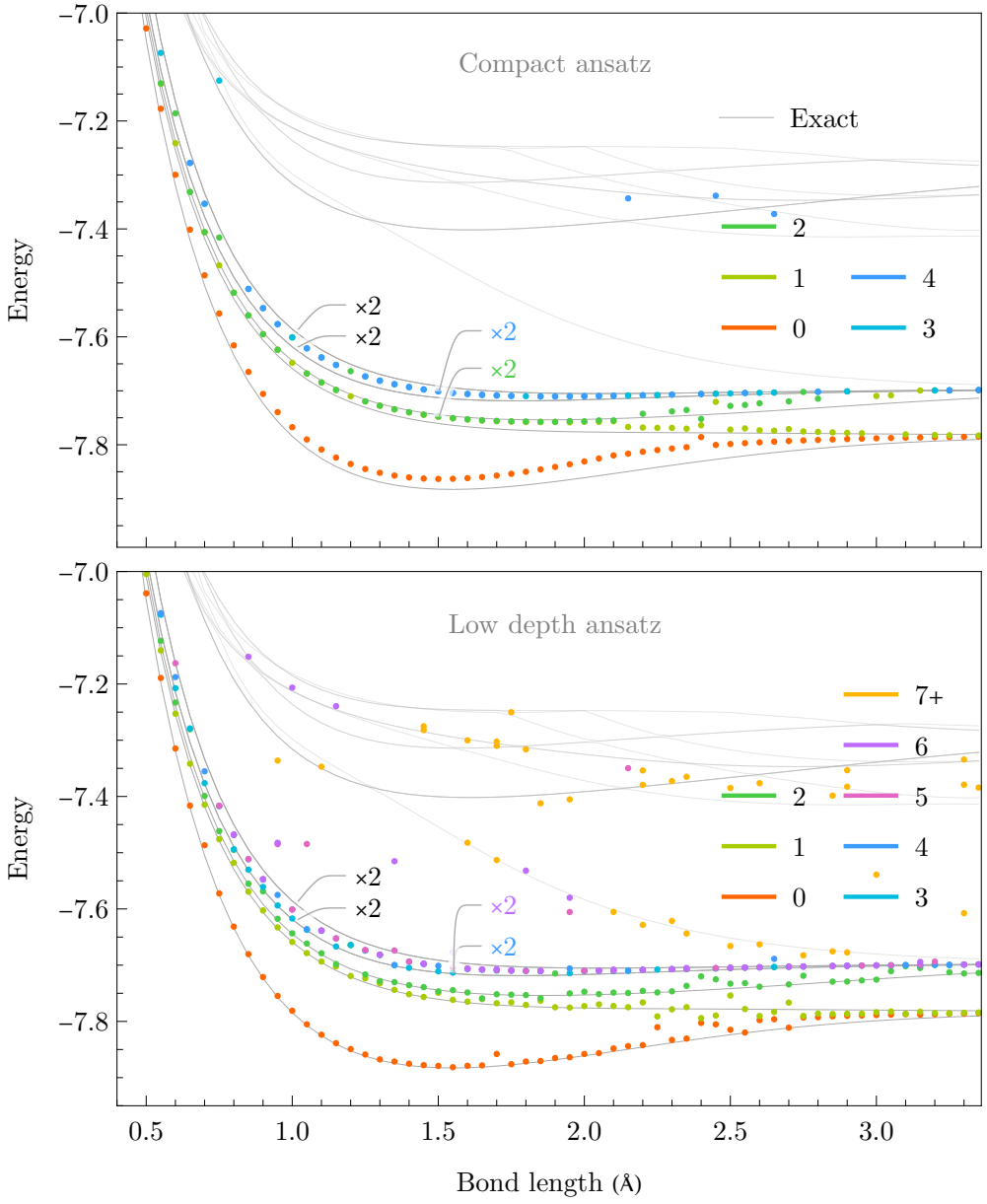


Figure 3.9: Our diagonalisation algorithm discovering (coloured points) the energy spectrum (gray lines) of the 10-qubit Lithium Hydride electronic structure Hamiltonian $\hat{H}_{10}^{(\mathrm{LiH})}$ over varying bond length. Inset labels indicate degeneracy in both the true and discovered states. The compact ansatz (top) is used with 70 parameters and for 10^4 iterations at every bond length. The low depth ansatz uses 145 parameters for 4×10^4 iterations.

allel work [191]), as minimisation subroutines. In our tests, imaginary-time reliably converged to eigenstates, while gradient descent frequently converged to energetically insignificant local minima, causing rapidly accumulating errors in the reported spectrum. Our study also revealed an interesting property of variational imaginary time minimisation; that while it seems to reliably converge to eigenstates, it does not necessarily approach them in order of the lowest lying. This behaviour deviates from precise imaginary time evolution, owed to the ansatz's limited expressibility, but does not jeopardise our scheme.

Our somewhat introductory study here begets a number of interesting questions. How should the goal of generating eigenstates inform the design of the ansatz? Why does the imaginary-transformed energy space, as traversed along the gradient $\underline{\underline{\mathcal{M}}}^{-1}\vec{\mathcal{V}}$, contain local minima at the eigenstates, as opposed to non-attractive stationary states? Could a changing ansatz design controllably influence the order of discovered eigenstates? We must also note several important limitations of our present study. While our testing demonstrated that algorithmic error (due to the excited ansatz states only approximating the true eigenstates) did not appear to catastrophically accumulate, we made no similar study in noisy conditions. Section 3.3.2 described how incorporation of a simple error mitigation strategy, through post-selection of the swap-test measurements, could nullify (some of) the errors in the estimations of the state overlaps. But even decoherence present only during minimisation will likely corrupt our Hamiltonian penalisation, and gradually increase the error of the reported eigenvalues. Further, as a variational algorithm, our method must navigate the many associated pitfalls such as barren plateaus [204]. Figure 3.10 shows an anomalous simulation of our method becoming trapped in a suspected barren plateau toward a shallow local minima. We save further discussion of these standard variational considerations for Chapter 4.

If such obstacles can be overcome, our algorithm has many potential interesting applications beyond learning the spectrum of chemistry Hamiltonians. We are not limited to exciting only eigenstates - we can excite *any* states for which the generating ansatz parameters are known. We could ergo eliminate unwanted subspaces from the searched spectrum of minimisation, such as those which break symmetries or indicate errors, by energy penalising them. Since our Hamiltonian augmentation is performed through the classical variational equations, we could even modulate the energy spectrum of real-time variational simulations, like Li's method [132] of Section 1.6.2. Eigenstates can have their eigenvalues shifted to create or remove energy degeneracies, change resonances, or even introduce time-dependence; all without prohibitive classical description. This remains an exciting direction of future research, from which the author regrets his distraction by a Snicket-esque series of unfortunate events.

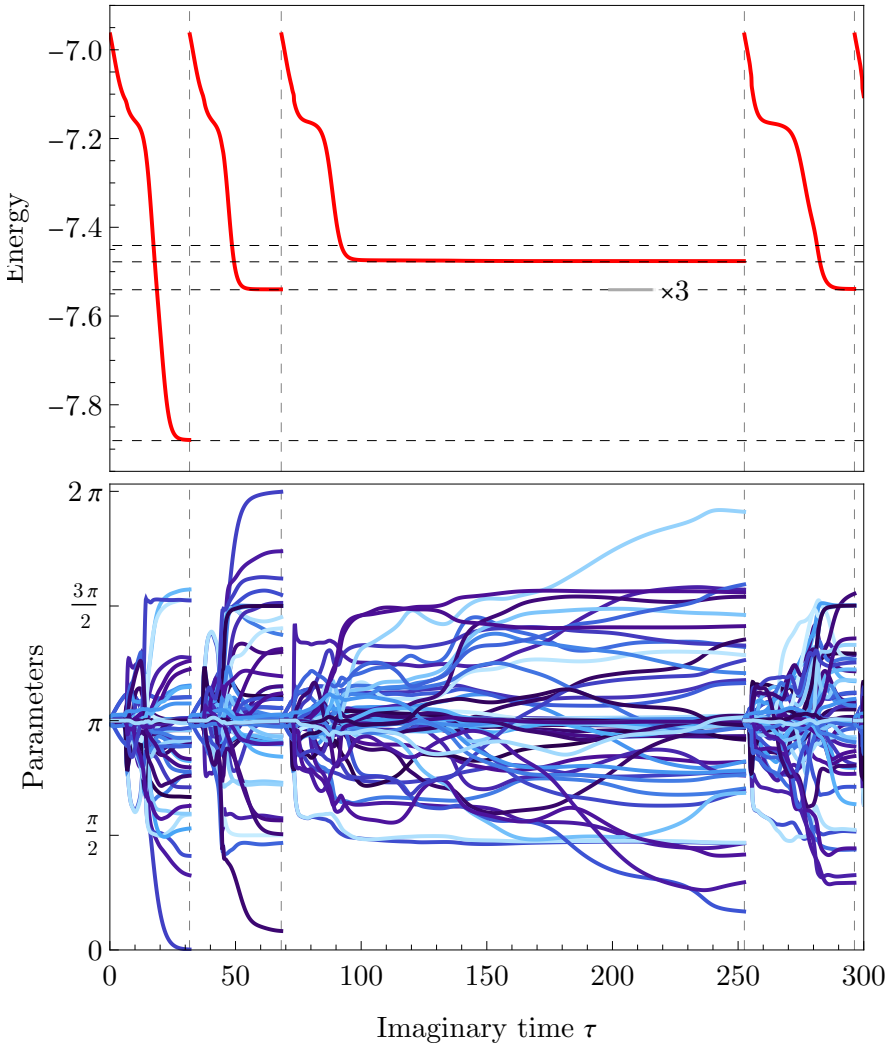


Figure 3.10: Our diagonalisation routine becoming momentarily stuck in a region with energy (top) very close to an eigenvalue of $\hat{H}_6^{(\mathrm{LiH})}$, suspected a barren plateau [204]. Notice the 56 parameters (bottom) of the low depth ansatz continue to vary drastically.

Chapter 4

Quantum Recompilation

Contents

4.1 Foreword

This section presents the published manuscript

- **T. Jones**, S. C. Benjamin

  Robust quantum compilation and circuit optimisation via energy minimisation

  Quantum **6**, 628 (2022) (link)

which develops a novel quantum algorithm for approximately recompiling quantum states. My coauthor identified this area for research and suggested an approach; we jointly developed the resulting algorithm. Work that is not my own is identified in text, though my contribution can be summarised as the development, analytic and numerical investigation of the algorithm in noise-free settings. This work also culminated in an open-source `C` package, available on Github.

- **T. Jones**

  Dissipative Recompiler

  DOI: 10.5281/zenodo.**6475224** (2018)

  Hosted on Github `QTechTheory/DissipativeRecompiler` (link)

All derivation in this chapter is my own.

4.2 Introduction

This thesis has so far presented quantum variational algorithms for hybrid classical-quantum computers to discover the energy groundstates and low-lying eigenstates of classically intractable Hamiltonians. We will now leverage these schemes as subroutines to an orthogonal problem; quantum-assisted approximate recompilation of quantum circuits. In particular, we explore recompilation of variational circuits themselves.

In conventional computing, compilers are essential to translate programs into efficient low-level instructions for execution at the hardware level. Quantum computers will also benefit greatly from efficient compilation, but the nature of the compilation goal depends on the nature of the machine. Even with tomorrow's fault-tolerant devices, only a limited non-universal family of operations can be performed directly on encoded data, so that other (usually non-Clifford) operations must be performed with the use of additional resources such as magic states [205, 206, 207]. Compilation of quantum circuits for these platforms will involve minimising the number of these expensive resource-consuming gates. Substantial efforts have been made to understand how to minimise the number of non-Clifford gates, such as T-gates, that are required to perform a given task [208, 209, 210, 211, 212].

But in the present day and near-future of NISQ devices, the priorities for compilation are different. Without error correction, one expects a wider family of operations can be performed upon a quantum computer, albeit with a substantial error burden. Typically, two-qubit gates (and higher degree gates) fair worse, while single-qubit gates are higher fidelity [213, 214, 215, 216]. Different emergent noise characteristics will distinguish which of many gate-level decompositions to implement a non-native operator are optimal. Moreover a device will have a native connectivity: with global qubit commu-

nication assumed an engineering impossibility, qubits may be constrained to undergo multi-qubit operations in only two-dimensional nearest-neighbour topologies, or other network structures [217]. Without the safe abstraction that any interaction can be efficiently effected with a series of restricted interactions, compilation of a generic circuit into one optimised for a particular NISQ device is indispensable.

Yet it is not without complications. Even when targeting a universal gate set, it may be *impossible* to exactly recompile a circuit to one of sufficiently shallow depth to be useful on NISQ devices. And we may not even wish to impose precise equivalence, if we are already treating our circuit as an imprecise black-box, as we do in variational applications. *And* regardless of whether exact recompilation is possible, classically obtaining the recompiled circuit is expected to be NP and generally intractable. Even recompiling the action of a circuit upon a fixed input state (as repeatedly performed in variational schemes), which one expects substantially more permissive than recompiling the full circuit unitary, remains intractable.

This motivates our endeavour to develop an algorithm to leverage a *quantum* device to perform *approximate* recompilation of NISQ circuits on fixed inputs. This chapter presents a novel method of translating one quantum circuit into another, and then automatically shrinking it; in effect, recompiling and optimising it. This will, along the way, involve the exploration and development of several minor phenomena and subroutines, including the early-evolution behaviour of Li's real-time simulation, experimental proposals to estimate recompilation fidelity, and a method to dynamically update the minimisation timestep hyperparameter. We test our scheme on up to 20 qubits, and perform a demonstration of a possible application; extending variational real-time simulation. We also make a comprehensive, density matrix based analysis of the impact of gate noise for a scenario involving over 200 gates.

4.2.1 Previous work

We note other recent efforts to perform approximate recompilation of quantum circuits. For example, a scheme presented by Khatri *et al* describes a method to recompile a (possibly unknown) n-qubit unitary U into a parametrised circuit V [218], by maximising their *overlap*. Their cost function is $|\mathrm{Tr}(V^\dagger U)|^2$, evaluated via the Hilbert-Schmidt test, requiring $2n$ qubits and $2n$ additional controlled gates. They demonstrate their technique with noise-free and noisy simulations of up to 9 qubits. They thus focus on the less permissive task of recompiling the entire unitary as opposed to its action on a fixed input state, while noting the possibility of fixed-input schemes. Compared to the present work, that study [218] presents a broader range of techniques and includes 1-qubit demonstrations on cloud-based quantum hardware, but investigates considerably more shallow circuits as the input (e.g. 4 rather than 60 layers) and smaller registers (simulating a maximum of 9 qubits rather than the 20 considered here). To perform our larger and more complex compilation tasks, we introduce certain enabling techniques described presently, including adaptive timesteps, elimination of redundant gates, and a 'lure' method to improve convergence.

Meanwhile an experimentally-focused effort [219] performed a similar scheme on a photonic processor of 4 qubits, with numerical testing on up to 5 qubits. There, they recompile an unknown unitary U acting on a fixed, restrictedly seperable state $|\mathrm{in}\rangle$, and use an ansatz divided into multiple layers V_l, each targeting one fewer qubits. The parameters in each layer are successively trained by driving the layer's nominated qubit toward its initial state in the input separable state, i.e. toward parameters which satisfy $(\prod_l V_l)^\dagger U\,|\mathrm{in}\rangle \approx |\mathrm{in}\rangle$. In contrast to these works, our fixed-input method requires only n qubits, with a cheap cost function evaluated via measuring a fictitious Hamiltonian. We make no conditions on the input state. When our input unitary happens to be pa-

rameterized, our scheme is compatible with an additional novel *luring* technique, which can significantly improve convergence on systems larger than tested by the previous works. We elaborate more explicitly on the features and limitations of our method in Section 4.6.

4.3 Derivation

This section presents derivations of our main algorithm and its related subroutines.

4.3.1 Compilation

Consider the state produced by a fixed unitary $\hat{\mathcal{A}}$ upon fixed input state $|\text{in}\rangle$. Suppose we are given a parameterized unitary circuit $\hat{\mathcal{B}}(\vec{\theta})$, with P unspecified parameters $\vec{\theta}$, and gates from the native set of our target quantum hardware. The task of approximately compiling $\hat{\mathcal{A}}$ into $\hat{\mathcal{B}}(\vec{\theta})$ is that of finding $\vec{\theta}_0$ which satisfy

$$\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle \approxeq \hat{\mathcal{A}}\,|\text{in}\rangle \qquad \Longrightarrow \qquad \hat{\mathcal{B}}^\dagger(\vec{\theta}_0)\,\hat{\mathcal{A}}\,|\text{in}\rangle \approxeq |\text{in}\rangle\,. \tag{4.1}$$

Here, we assign $\approxeq$ to loosely mean that the states are proximate in some space which ignores global phase. Assume that $\hat{\mathcal{B}}(\vec{\theta})$ has the form of a separable ansatz,

$$\hat{\mathcal{B}}(\vec{\theta}) = \prod_{i=P}^{1} \hat{\mathcal{B}}_i(\theta_i) = \hat{\mathcal{B}}_{1:P}, \tag{4.2}$$

re-adopting our usual subcircuit notation. We will further assume that the adjoint gate $\hat{\mathcal{B}}_i^\dagger(\theta_i)$ is experimentally feasible to perform, as we did in Section 2.4.2. This will almost certainly be the case if $\hat{\mathcal{B}}_i$ are rotation gates or Pauli gadgets (Equation 1.21) in which

case, $\hat{\mathcal{B}}_i^\dagger(\theta_i) = \hat{\mathcal{B}}_i(-\theta_i)$. Then, the inverse circuit

$$\hat{\mathcal{B}}^\dagger(\vec{\theta}) = \prod_{i=1}^{P} \hat{\mathcal{B}}_i^\dagger(\theta_i) = \hat{\mathcal{B}}_{1:P}^\dagger \tag{4.3}$$

is equally as easy to experimentally perform as a sequence of gates, in the reverse direction as they appear un-conjugated in $\hat{\mathcal{B}}(\vec{\theta})$. This enables us to consider the product

$$\hat{U}(\vec{\theta}) = \hat{\mathcal{B}}^\dagger(\vec{\theta})\, \hat{\mathcal{A}} \tag{4.4}$$

as an experimentally feasible P-parameter ansatz circuit for which we seek to obtain $\vec{\theta}_0$ satisfying

$$|\psi(\vec{\theta}_0)\rangle = \hat{U}(\vec{\theta}_0)\ |\text{in}\rangle \quad \approxeq |\text{in}\rangle\,. \tag{4.5}$$

Note this is an intuitively more permissive goal than to seek $\hat{U}(\vec{\theta}_0) \approx \hat{\mathbb{1}}$, since we require only a *single* state $|\text{in}\rangle$ (rather than the entire Hilbert space) to undergo a specific transformation. Our next step is to introduce some strategy to obtain $\vec{\theta}_0$, such as an iterative variational strategy. In principle, we could create a scalar cost function of $\vec{\theta}$, such as $1 - |\langle \text{in}|\psi(\vec{\theta})\rangle|$, which may be experimentally measurable using the swap-test utilised in Chapter 3, and integrate it with a generic minimisation routine, like gradient descent. Alas, this would require either deriving new expressions for the parameter derivatives of the cost function and developing experimental procedures to quantum evaluate them (like we did in Section 2.4.2), or introducing yet-unstudied additional algorithmic error through finite-difference approximations. It would further preclude utilising the performance advantages we saw of imaginary-time evolution in Chapter 2.

Instead, we recast the problem. We assume that $|\text{in}\rangle$ is a well-understood state and trivial to prepare, as is almost always the case in variational algorithms; often $|\text{in}\rangle$ is separable, or the simple initial state to some real-time simulation. We reason then that it is easy to produce some tractable Hamiltonian for which $|\text{in}\rangle$ is the unique groundstate, such as

$$\hat{H}_{\text{rec}} = - |\text{in}\rangle \langle \text{in}| . \tag{4.6}$$

For example, if $|\text{in}\rangle \equiv |0\rangle^{\otimes}$, then a different but equally obvious Hamiltonian is $\hat{H}_{\text{rec}} = \sum_q \hat{Z}_q$. If we wished, we could further create favourable properties of this Hamiltonian such as a well-defined gap and intentional eigenstate degeneracies, to aid our next step. Armed with a suitable Hamiltonian, we can recast the problem of recompilation into one of energy minimisation; we seek $\vec{\theta}_0$ which admit energy $\langle E(\vec{\theta}_0)\rangle$ satisfying

$$\langle E(\vec{\theta}_0)\rangle = \min_{\theta} \langle \psi(\vec{\theta})|\hat{H}_{\text{rec}}|\psi(\vec{\theta})\rangle . \tag{4.7}$$

We can now employ any one of the variational energy minimisation techniques explored in this thesis. We naturally select imaginary time evolution with which we have greatest familiarity. We employ it as a subroutine to find $\vec{\theta}_0$ which minimises $\hat{H}_{\text{rec}}$ with ansatz $\hat{U}(\vec{\theta}) = \hat{\mathcal{B}}^{\dagger}(\vec{\theta})\,\hat{\mathcal{A}}$ upon input state $|\text{in}\rangle$, just like we would any other problem Hamiltonian. We emphasise that $\hat{H}_{\text{rec}}$ is in a sense, *fictitious*; we design it merely to drive compilation, and it need not encode any physical system. As such, we can guarantee it is tractable to incorporate into variational minimisation. If energy minimisation succeeds in obtaining $\vec{\theta}_0$ which satisfy (or sufficiently well approximately satisfy) Equation 4.7, then we have simultaneously discovered the ideal (or approximately ideal) parameters to populate our target circuit $\hat{\mathcal{B}}(\vec{\theta}_0)$. Ergo, we will have succeeded in recompilation. As we will soon see, there are further tricks we can employ.

An interested reader may wonder why we did not leverage a variational principle to develop a direct parameter optimisation strategy for iterative recompilation, like we did in our use of McLachlan's principle for imaginary-time in Section 2.4.1. This is because the proximity of $\hat{\mathcal{B}}(\vec{\theta})$ to $\mathcal{A}$ is independent of the time (or, *interpolated iteration*) derivative of the parameters. As such, we will not obtain a prescription for *updating* the parameters per-iteration unless we manually add a target time-evolution, which we have essentially performed above (by incorporating imaginary time evolution).

We quickly demonstrate this mathematically. Define $d(\vec{\theta})$ as the norm distance between the input and recompiled states,

$$d(\vec{\theta}) = \left\| \left(\hat{\mathcal{B}}(\vec{\theta}) - \hat{\mathcal{A}} \right) |\text{in}\rangle \right\| \tag{4.8}$$

$$= \sqrt{2 - 2\,\Re\left[\langle \text{in}|\, \hat{\mathcal{B}}^\dagger(\vec{\theta})\, \hat{\mathcal{A}}\, |\text{in}\rangle \right]}, \tag{4.9}$$

invoking $\| \, |\Psi\rangle \, \| = \sqrt{\langle \Psi | \Psi \rangle}$, and observing $x + x^* = 2\,\Re[x]$. McLachlan's variational principle seeks the minimum at the extrema of the variation,

$$\min_{\vec{\theta}} d(\vec{\theta}) \quad \Longrightarrow \quad 0 = \delta d(\vec{\theta}) \tag{4.10}$$

$$= \sum_i \frac{\partial d(\vec{\theta})}{\partial \theta_i}\, \delta\theta_i \tag{4.11}$$

$$\Longrightarrow \quad 0 = \frac{\partial d(\vec{\theta})}{\partial \theta_i} \;\; \forall \;\; i, \tag{4.12}$$

as per the full derivative and independence of θ_i. Note we made no inclusions of higher

order derivatives of $\vec{\theta}$, because clearly $\frac{\partial d(\vec{\theta})}{\partial(\partial_t^{(n)}\theta_i)} = 0 \; \forall \; n \geq 1$. Continuing,

$$0 = \frac{\partial}{\partial\theta_i}\sqrt{2 - 2\,\Re\left[\langle \text{in}|\,\hat{\mathcal{B}}^\dagger(\vec{\theta})\,\hat{\mathcal{A}}\,|\text{in}\rangle\right]} \tag{4.13}$$

$$\implies 0 = \frac{\partial}{\partial\theta_i}\Re\left[\langle \text{in}|\,\hat{\mathcal{B}}^\dagger(\vec{\theta})\,\hat{\mathcal{A}}\,|\text{in}\rangle\right] \tag{4.14}$$

$$= \Re\left[\langle \text{in}|\,\frac{\hat{\mathcal{B}}^\dagger(\vec{\theta})}{\partial\theta_i}\,\hat{\mathcal{A}}\,|\text{in}\rangle\right]. \tag{4.15}$$

There is little point continuing to substitute $\hat{\mathcal{B}}(\vec{\theta})$ for a separable ansatz in terms of unitaries generated by Hermitian operators, since we will maintain a similar form to that above, which simply asserts that the optimum parameters satisfy Equation 4.15. In a sense, we have binarised our cost function to those parameters which do and do not satisfy this equation, with no prescription for how to obtain them. We would obtain a similar conclusion through a cost function in greater parallel with our decided method above, like

$$\max_{\vec{\theta}}\left|\langle \text{in}|\,\hat{\mathcal{B}}^\dagger(\vec{\theta})\,\hat{\mathcal{A}}\,|\text{in}\rangle\right|. \tag{4.16}$$

4.3.2 Optimisation

Section 4.3.1 derived our main compilation method, which is to perform minimisation under a fictitious Hamiltonian resembling $\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |\text{in}\rangle\langle\text{in}|$, designed to drive the parameters $\vec{\theta}$ toward $\vec{\theta}_0$ which satisfy $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle \approxeq \hat{\mathcal{A}}\,|\text{in}\rangle$, for a user-designed target circuit $\hat{\mathcal{B}}(\vec{\theta})$. We now develop a post-compilation routine to *optimise* the output circuit by removing superfluous (or approximately superfluous) gates from $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle$. This is an optional gate-elimination optimisation step in a sense, though we find it greatly worthwhile to shrink the gate costs of $\hat{\mathcal{B}}$ without tangible compromise in the recompilation

fidelity.

Recall the target circuit is safely assumed separable in parameters,

$$\hat{\mathcal{B}}(\vec{\theta}) = \prod_{i=P}^{1} \hat{\mathcal{B}}_i(\theta_i). \qquad \text{(re-expression of 4.3)}$$

For notational simplicity, assume that each gate is parametrised such that $\theta_i = 2\pi n$, $n \in \mathbb{Z}$ produces $\hat{\mathcal{B}}_i(\theta_i) = \hat{\mathbb{1}}$ (up to global phase). This is already the case for the ubiquitous Pauli gadgets, but is possible for any periodic parametrized gate by shifting and scaling the parameter labels. We can then reason that any parameter $\theta_i \in \vec{\theta}_0$ satisfying

$$d_i = \min_{n\in\mathbb{Z}} |\theta_i - 2\pi\, n| \ll 1 \qquad \text{effects} \qquad \hat{\mathcal{B}}_i(\theta_i) \approxeq \hat{\mathbb{1}}, \tag{4.17}$$

and may hence make a negligible contribution in $\hat{\mathcal{B}}(\vec{\theta}_0)$ toward producing the state $\hat{\mathcal{A}}\,|\text{in}\rangle$. Ergo, its gate is a candidate for removal from the circuit. Let $\hat{\mathcal{B}}_x(\theta_x)$ be the *closest* of such gates, where (for simplicity) $\theta_x = \delta x$, and $\delta x \ll 1$. That is,

$$\delta x = \min_{i\in\{1,\ldots,P\}} d_i\,. \tag{4.18}$$

To eliminate this gate, we resume imaginary-time minimisation under $\hat{H}_{\text{rec}}$ with ansatz $\hat{\mathcal{B}}^\dagger(\vec{\theta})\hat{\mathcal{A}}\,|\text{in}\rangle$, starting from $\vec{\theta} = \vec{\theta}_0$, but add an additional constraint to our variational equations to enforce that

$$\Delta\tau\ \dot{\theta}_x = -\frac{\delta x}{K}, \tag{4.19}$$

where $K \in \mathbb{N}^+$ is a hyperparameter to force θ_x to smoothly reach value 0 in K iterations. Constraint 4.19 is incorporated by removing the x-th column (or setting it to zero) of

matrix $\underline{\underline{\mathcal{M}}}$ from the imaginary-time variational equations

$$\underline{\underline{\mathcal{M}}}\,\dot{\vec{\theta}} = -\vec{\mathcal{V}}, \qquad \text{(reiteration of 2.21)}$$

and adding it (times the forced $\dot{\theta}_x$) to the $\vec{\mathcal{V}}$ vector. That is, we transform

$$\mathcal{V}_i \to \mathcal{V}_i + \frac{\delta x}{K\,\Delta\tau}\mathcal{M}_{i,x}, \qquad \mathcal{M}_{i,x} \to 0 \qquad \forall i \in \{1,\ldots,P\}. \tag{4.20}$$

Imaginary-time minimisation under these modified equations drives the system toward a state where $\theta_x = 0$ and the other parameters are pressured to return to their minimum possible energy configuration, maximising the fidelity. After $> M$ iterations (since we should continue minimising afterward to ensure the other parameters converge to their reconfiguration), we reach new parameters $\vec{\theta}_0'$ whereby $\theta_x = 0$ and $\hat{\mathcal{B}}_x(\theta_x) = \hat{\mathbb{1}}$, allowing us to remove the now superfluous x-th gate from the recompiled circuit. Minimizing under constraint Equation 4.19 will increase the energy (and thus decrease the recompiled fidelity) by some small amount. Ergo, we are removing a gate at the cost of a small fidelity defect. We will see in Section 4.3.3 how an experimentalist may monitor this fidelity.

With θ_x eliminated, and circuit $\hat{\mathcal{B}}(\vec{\theta}_0')$ now shrunk to one of $P-1$ parameters, we can repeat this process; identify the next parameter satisfying Equation 4.18, and eliminate it by augmented imaginary-time minimisation. We can continue to shrink $\hat{\mathcal{B}}$ in this way, gate-by-gate, until the energy has increased beyond some hyperparameter threshold. At this point, we deem further optimisation will too greatly compromise the recompilation fidelity, and we adopt the latest parameter set. By using one's full knowledge of $\hat{H}_{\text{rec}}$ and its energy spectrum (in addition to the fidelity bounds of the proceeding section), an experimentalist could carefully choose this energy threshold in order to bound the

infidelity, as necessary for their application. In our testing however, we will heuristically assign this threshold to be double the initial energy gap reached after recompilation, before optimisation. In other words, we permit the energy defect to double in exchange for a reduction in the output circuit. We will denote the reduced output circuit, post-optimisation, by $\hat{\mathcal{B}}_{\text{elim}}$, where its parameters have finally been fixed to $\vec{\theta}_0'$. If optimisation was successful, $\dim(\vec{\theta}_0') < \dim(\vec{\theta}_0)$.

4.3.3 Fidelity

Since we will test our algorithm through classical emulation of quantum devices, we will be able to numerically compute the fidelity of the partially recompiled state during minimisation as

$$F(\vec{\theta}) = \left| \langle \text{in} |\, \hat{\mathcal{B}}^\dagger(\vec{\theta})\, \hat{\mathcal{A}}\, | \text{in} \rangle \right|^2 . \tag{4.21}$$

While useful as method to track the progress of recompilation, this quantity will be generally experimentally inaccessible. Fortunately, we can exploit our knowledge of, and freedom over, the fictitious Hamiltonian $\hat{H}_{\text{rec}}$ to offer an experimental bound to the fidelity.

First consider when an experimentalist is able to realise Hamiltonian

$$\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |\text{in}\rangle \langle \text{in}|\,, \qquad \implies 0 \leq \langle E(\vec{\theta}) \rangle \leq 1. \tag{4.22}$$

In this scenario, we find that the fidelity is related precisely to the fidelity F;

$$\langle E(\vec{\theta})\rangle = \langle \text{in}|\, \hat{\mathcal{A}}^\dagger\, \hat{\mathcal{B}}\left(\hat{\mathbb{1}} - |\text{in}\rangle\langle \text{in}|\right) \hat{\mathcal{B}}^\dagger\, \mathcal{A}\, |\text{in}\rangle \tag{4.23}$$

$$= 1 - F(\vec{\theta}). \tag{4.24}$$

For the general case of a Hamiltonian with eigenstates $|e_i\rangle$ and minimum $|\text{in}\rangle$,

$$\hat{H}_{\text{rec}} \equiv e_0\, |\text{in}\rangle\langle \text{in}| + \sum_{i>0} e_i\, |e_i\rangle\langle e_i|\,, \qquad e_{i+1} > e_i \tag{4.25}$$

we consider our variational state in the energy basis,

$$|\psi(\vec{\theta})\rangle = \hat{\mathcal{B}}^\dagger(\vec{\theta})\, \mathcal{A}\, |\text{in}\rangle \quad \equiv \quad \beta_0\, |\text{in}\rangle + \sum_{i>0} \beta_i\, |e_i\rangle\,, \tag{4.26}$$

with amplitudes satisfying

$$\beta_i = \langle \psi(\vec{\theta})|e_i\rangle\,, \qquad \sum_{i=0} |\beta_i|^2 = 1, \tag{4.27}$$

and can ergo express the fidelity of recompilation merely as the probability of the groundstate, $F(\vec{\theta}) = |\beta_0|^2$. The energy of our variational state becomes

$$\langle E(\vec{\theta})\rangle = F(\vec{\theta})\, e_0 + \sum_{i>0} |\beta_i|^2\, e_i. \tag{4.28}$$

To derive bounds between F and $\langle E\rangle$, imagine F is fixed and we wish to obtain the bounds of $\langle E\rangle$. The maximum and minimum energies respectively correspond to the maximum occupation of the highest excited state, and the maximum occupation of the

first excited state (since the groundstate population is fixed by F). That is,

$$\max_{\{|\beta_{i>0}|^2\}} \langle E(\vec{\theta})\rangle = F(\vec{\theta})\, e_0 + (1 - F(\vec{\theta}))\, e_{\max} \qquad \leq e_{\max} \tag{4.29}$$

$$\min_{\{|\beta_{i>0}|^2\}} \langle E(\vec{\theta})\rangle = F(\vec{\theta})\, e_0 + (1 - F(\vec{\theta}))\, e_1 \qquad \leq e_1. \tag{4.30}$$

We rearrange these to obtain bounds of the fidelity;

$$\frac{e_1 - \langle E(\vec{\theta})\rangle}{e_1 - e_0} \leq F(\vec{\theta}) \leq \frac{e_{\max} - \langle E(\vec{\theta})\rangle}{e_{\max} - e_0}. \tag{4.31}$$

In other words, we have inferred the extrema probabilities of the groundstate (i.e. F) of a general superposition, in terms of its energy. Since $|\text{in}\rangle$ and consequentially $\hat{H}_{\text{rec}}$ is well understood, we will know its energy spectrum (or good approximations thereof) in advance, including the ground energy e_0, the first excited energy e_1, and the maximum energy $e_{\max}$. Therefore, Equation 4.31 is a means to infer the fidelity of recompilation from the experimentally accessible energy, which is likely already being evaluated during the minimisation subroutine. Finally, we remind again that $\hat{H}_{\text{rec}}$ is fictitious and controllable by the experimentalist, and even if they cannot achieve $\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |\text{in}\rangle\langle\text{in}|$ which identifies the fidelity uniquely, they may be able to modulate the gaps $e_1 - e_0$ and $e_{\max} - e_0$ to tighten these bounds.

4.3.4 Luring

Quantum minimisation, as we have witnessed in this thesis, can go awry in a multitude of ways. In anticipation of potential difficulties in the minimisation subroutines of recompilation, we explore additional meta-heuristics or techniques that can exploit the context of recompilation. In particular, we will leverage that our input quantum circuit

$\hat{\mathcal{A}}$ is likely itself a parametrised ansatz circuit, $\hat{\mathcal{A}}(\vec{\phi})$, where $\vec{\phi} = \vec{\phi}_0$ produces the state to be re-expressed through $\hat{\mathcal{B}}$. For this scenario, we develop a technique we refer go as "*luring*", whereby we undergo N successive recompilations of intermediate input states $\hat{\mathcal{A}}(\vec{\phi}_0\, n/N)\,|\text{in}\rangle$, with $n \in \{1, \ldots, N\}$. This is a meta-routine above recompilation to improve performance, and is generally not necessary for the results of this chapter, although it will be invoked later for an exceptionally tricky recompilation example.

Assume that $\hat{\mathcal{A}}(\vec{\phi})$ is parametrised such that $\vec{\phi} = \vec{0}$ produces the identity, i.e. $\hat{\mathcal{A}}(\vec{0}) \cong \hat{\mathbb{1}}$. This is a matter of labelling, and leaves $\hat{\mathcal{A}}(\vec{\phi}_0)$ general. We next require that the unitaries

$$\hat{\mathbb{1}}, \quad \hat{\mathcal{A}}(\vec{\phi}_0\, 1/N), \quad \hat{\mathcal{A}}(\vec{\phi}_0\, 2/N), \quad \ldots \quad \hat{\mathcal{A}}(\vec{\phi}_0(N-1)/N), \quad \hat{\mathcal{A}}(\vec{\phi}_0), \tag{4.32}$$

interpolate between the identity and the full action of $\hat{\mathcal{A}}$. If this were not the case, then we could instead fix $\vec{\phi} = \vec{\phi}_0$ and introduce our own global parametrisation α which does have this property, such as $\hat{\mathcal{A}}^{\alpha}(\vec{\phi}_0)$ for $\alpha \in [0, 1]$.

Next, we reason that since $\hat{\mathcal{A}}(\vec{\phi}_0/N)\,|\text{in}\rangle$ is consequentially "closer" (in some measure) to $|\text{in}\rangle$ than $\hat{\mathcal{A}}(\vec{\phi}_0)\,|\text{in}\rangle$, it is *easier* to minimise $\hat{\mathcal{B}}^{\dagger}(\vec{\theta})\hat{\mathcal{A}}(\vec{\phi}_0/N)\,|\text{in}\rangle$ than $\hat{\mathcal{B}}^{\dagger}(\vec{\theta})\hat{\mathcal{A}}(\vec{\phi}_0)\,|\text{in}\rangle$ (the original problem). We use "easiness" here to loosely refer to the convergence rate and likelihood of ultimately reaching the solution state, assumed inverse to the state "complexity". Our logic here is somewhat hand-wavy, and stems from independent observations that short-time evolved states are easier to recompile than long-time ones, known to be more complex as measurable by unitary compressibility [220]. This suggests that the circuits of Equation 4.32, acting upon $|\text{in}\rangle$, are progressively more difficult to recompile into from $\hat{\mathcal{B}}(\vec{\theta})$ (and thus, vice versa).

Our final logical assertion is that it is easier to recompile to state $\hat{\mathcal{A}}(\vec{\phi}_0/N) \approxeq \hat{\mathbb{1}}$ from

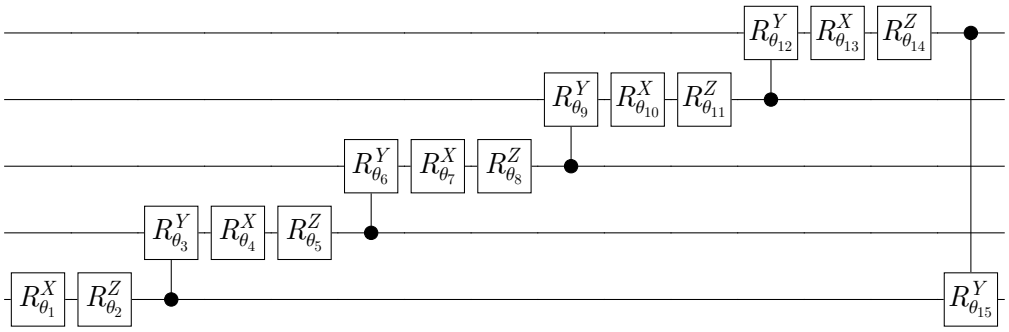


Figure 4.1: The simple ansatz circuit used both as input state $\hat{\mathcal{A}}(\vec{\phi}_0)\,|\text{in}\rangle$ and target state $\hat{\mathcal{B}}(\vec{\theta})\,|\text{in}\rangle$ in our demonstration of recompilation with luring.

$\hat{\mathcal{B}}(\vec{0}) = \hat{\mathbb{1}}$, than from a state of randomly initialised parameters $\hat{\mathcal{B}}(\vec{\theta}) \not\approx \hat{\mathbb{1}}$, due to the close proximity of the states. This would suggest that recompiling and converging to each intermediate state $\hat{\mathcal{A}}(\vec{\phi}_0\, n/N)$ in-turn for $n \in \{1, \ldots, N\}$, beginning each recompilation from the final parameters $\vec{\theta}$ of the previous recompilation, is easier than recompiling directly to $\hat{\mathcal{A}}(\vec{\phi}_0)$.

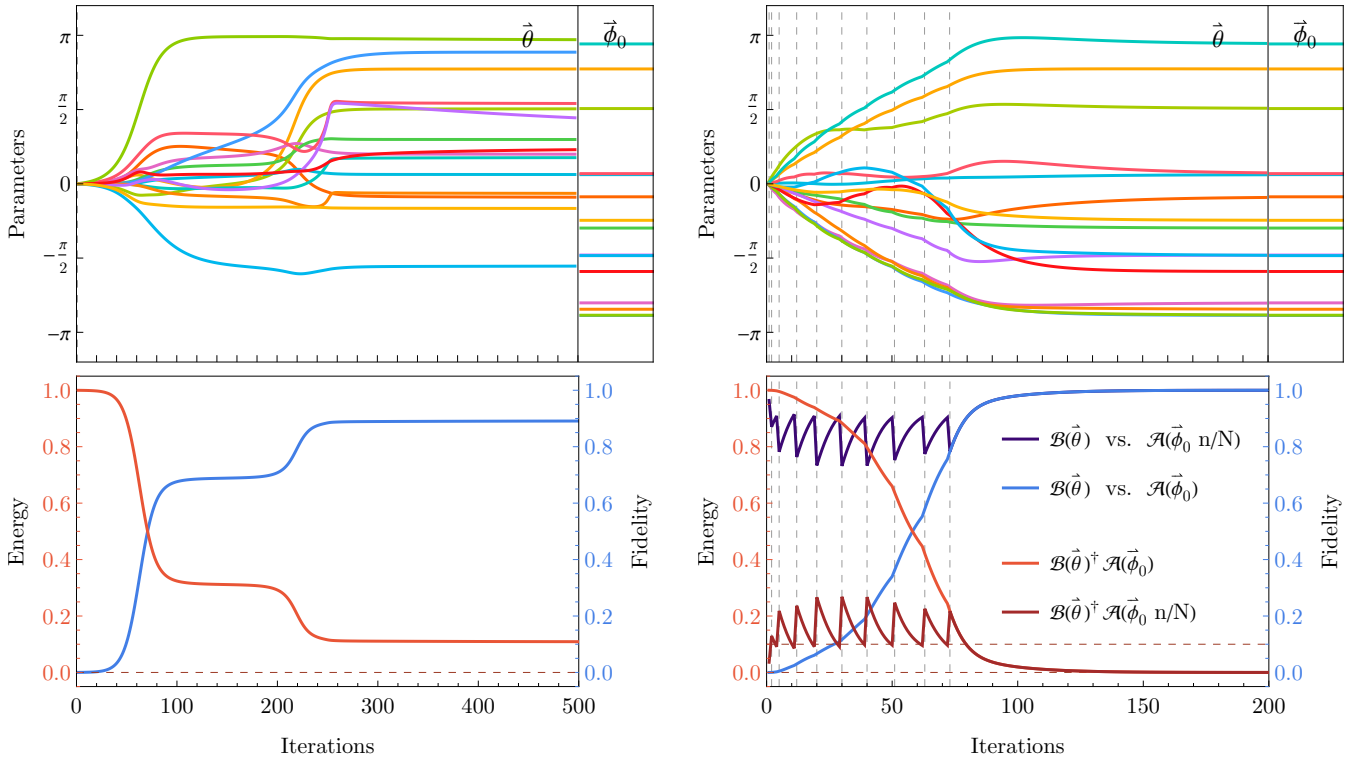
In effect, we propose to progress the parameters $\vec{\theta}$ to their final values $\vec{\theta}_0$ which produce

$$\hat{\mathcal{B}}\,(\vec{\theta}_0)\,|\text{in}\rangle \approxeq \hat{\mathcal{A}}(\vec{\phi}_0)\;|\text{in}\rangle \tag{4.33}$$

by successively "luring" them to intermediate values $\vec{\theta}^{(n)}$ which produce

$$\hat{\mathcal{B}}\,(\vec{\theta}^{(n)})\,|\text{in}\rangle \approxeq \hat{\mathcal{A}}(\vec{\phi}_0\, n/N)\;|\text{in}\rangle\,. \tag{4.34}$$

We do not claim these above properties are universally true; indeed one can construct adversarial examples for which luring of the recompilation parameters through intermediate states will direct them into local minima avoided by direct evolution from a random initial state. However, we generally observe luring to benefit convergence on arbitrary examples, *even* in instances where exact recompilation is possible. We will now demonstrate this.



(a) Without luring.

(b) With luring.

Figure 4.2: The left subfigure shows direct recompilation of $\hat{\mathcal{B}}(\vec{\theta})\,|\text{in}\rangle$ into $\hat{\mathcal{A}}(\vec{\phi}_0)\,|\text{in}\rangle$, where both circuits have the structure shown in Figure 4.1. The right subfigure shows the same recompilation, albeit via 10 successive recompilations of $\hat{\mathcal{B}}(\vec{\theta})\,|\text{in}\rangle$ into $\hat{\mathcal{A}}(\vec{\phi}_0\,n/N)\,|\text{in}\rangle$, where n is incremented whenever the experimentally accessible energy falls below 0.1. Where direct recompilation fails, luring succeeds in obtaining the precise parameters $\vec{\theta}_0 = \vec{\phi}_0$ whereby $\hat{\mathcal{A}} \equiv \hat{\mathcal{B}}$.

Consider the 5-qubit 15-parameter recompilation target $\hat{\mathcal{B}}(\vec{\theta})$ given in Figure 4.1. As an impractical example only for demonstration, we seek $\vec{\theta}$ which recompile input state $\hat{\mathcal{A}}(\vec{\phi}_0)\,|\text{in}\rangle$, whereby $\hat{\mathcal{A}}(\vec{\phi}_0) \equiv \hat{\mathcal{B}}(\vec{\phi}_0)$. That is, $\hat{\mathcal{A}}$ has an identical structure to $\hat{\mathcal{B}}$ and is ergo perfectly recompiled by $\hat{\mathcal{B}}(\vec{\theta})$ when $\vec{\theta} = \vec{\phi}_0$. We choose $\vec{\phi}_0$ to be uniformly random in $[-\pi, \pi]$, and begin recompiling from $\vec{\theta} = \vec{0}$. Imaginary-time minimisation will use Hamiltonian $\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |0\rangle\langle 0|^{\otimes 5}$ and fixed timestep $\Delta\tau = 0.1$. Because a perfect recompilation is possible, it will only fail due to issues in converging to the global groundstate of the Hamiltonian. Figure 4.2a shows the result of this effort after 500 iterations; $\vec{\theta}$ are driven into a final configuration $\vec{\theta}_0 \neq \vec{\phi}_0$, achieving a recompilation fidelity of only 89%. So we repeat this test, this time using $N = 10$ rounds of luring, incrementing n (and thus the input circuit parameters $\vec{\phi}_0\, n/N$) whenever the energy falls below 0.1 from the known groundstate $e_0 = 0$. The exceptional results are shown in Figure 4.2b. With no changes to the quantum resource costs, we now reach a recompiled fidelity of 99.9% by only 125 iterations, and stably continue to reach $\vec{\theta}_0 \equiv \vec{\phi}_0$ up to numerical precision by 200 iterations.

4.3.5 Adaptive timestep

Chapters 2 and 3 performed imaginary-time minimisation with a fixed timestep hyperparameter $\Delta\tau$, motivated as the largest for which energy monotonically decreased during evolution. This value is highly contingent on the geometry of the energy landscape, within which vast energy plateaus are known to emerge as the dimension increases. Because this chapter will later perform recompilation benchmarking at varying dimensions, we must first develop a protocol to automatically decide $\Delta\tau$ for the minimisation subroutines. In this section, we will derive an experimental strategy to cheaply and dynamically vary $\Delta\tau$ during evolution, which will not only remove the burden of manually motivating a fixed $\Delta\tau$, but will also considerably accelerate convergence in drastically changing energy environments.

We first motivate that our technique will introduce a negligible overhead in quantum resources. Recall that every iteration of P-parameter/gate imaginary-time evolution involves quantum evaluation of quantities $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}} \propto \nabla \langle E \rangle$ in a total number of

$$\mathcal{O}\left(m_{\mathcal{M}}\, L^2\, P^2(P+D) \;+\; m_{\mathcal{V}}\, P\, L\, T(P+N+D)\right) \quad \text{operations,} \quad \text{(reiteration of 2.42)}$$

where L and D are the number of terms and qubits in the Hermitian generator of each parametrized gate (typically $L = 1$, $D \in \{1, 2\}$), $T \gg 1$ is the number of terms in the N-qubit problem Hamiltonian, and $m_{\mathcal{M}}$ and $m_{\mathcal{V}}$ are the number of experimental shots used to evaluate each expectation value involved in the estimation of $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ quantities respectively. Meanwhile, the cost of even the most naive and inefficient method to evaluate the expected energy $\langle E(\vec{\theta}) \rangle$ alone is only

$$\mathcal{O}(m_E\, T\,(D+P)) \quad \text{operations,} \quad \text{(reiteration of 1.28)}$$

using m_E samples per Hamiltonian term. Ergo, evaluating $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ together is a dominating factor

$$\mathcal{O}\left(\frac{m_{\mathcal{M}}}{m_E} L P + \frac{m_{\mathcal{V}}}{m_E} L^2 P^2\right) \times \tag{4.35}$$

more expensive than evaluation of $\langle E(\vec{\theta})\rangle$. Even primitive quantum gradient descent with single-Hermitian-generator-term gates ($L = 1$) requires $P\times$ as many operations to evaluate the energy gradient over just the energy. This reveals that performing some additional energy estimations $\{\langle E(\vec{\varphi}_i)\rangle : i\}$ in every iteration of minimisation will only negligibly affect the total runtime. Or equivalently, we can perform as many as P additional energy measurements per iteration without doubling the total quantum costs of quantum gradient descent (and affect even less the total costs of imaginary-time). Evaluating the energy at a discrete set of points will be the only quantum requirement of our adaptive timestep technique.

Next, recall that after quantum evaluation of $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$, we classically obtain a solution $\dot{\vec{\theta}}$ to the regularised equation

$$\min_{\dot{\vec{\theta}}} \quad \left\| \underline{\underline{\mathcal{M}}}\, \dot{\vec{\theta}} + \vec{\mathcal{V}} \right\|^2 + \mu \left\| \dot{\vec{\theta}} \right\|^2, \tag{reiteration of 2.46}$$

which is used to update the parameters in a simple Euler step

$$\vec{\theta} \;\to\; \vec{\theta} \;+\; \Delta\tau\, \dot{\vec{\theta}}, \tag{re-expression of 2.48}$$

with fixed timestep $\Delta\tau$. At the close of Section 2.4.1, we reasoned that a sufficiently

small $\Delta\tau$ will ensure the energy monotonically decreases, and hence that

$$\langle E(\vec{\theta} + \Delta\tau\, \dot{\vec{\theta}})\rangle \;<\; \langle E(\vec{\theta})\rangle\,. \tag{4.36}$$

Yet, it is known that too small a timestep will unnecessarily slow minimisation, especially in regions with slowly varying energy landscapes. We have demonstrated above that an experimentalist can very much afford to test several values of $\Delta\tau$ and determine that which admits the greatest decrease in energy within that iteration, and hence in a sense, obtain the "most value" from that iteration. So we task the experimentalist with an additional subroutine each iteration, which is to efficiently discover

$$\min_i \langle E(\vec{\theta} + \eta_i\, \dot{\vec{\theta}})\rangle\,, \tag{4.37}$$

through a tractable number of candidate timesteps $\{\eta_i : i\}$. This is a gradient-free line-search [221] along an arbitrary one-dimensional path in our energy landscape. Since this happens to be tangent to the gradient in quantum gradient descent, we can imagine our technique as a means to discover the bottom of the current energy slope within a single iteration, though such a picture is less clear for imaginary-time minimisation.

We next motivate the choices of η_i to try, which form the crux of our scheme. We should consider that typical parametrised energy landscapes are expected to have both regions of significant variation, and of barren plateaus. This suggests the timestep should be rapidly adaptable between very small and very large values between iterations. We will thus develop a strategy to vary η among exponentially separated values

$$\eta_i \in \{\, 2^i\, \Delta\tau \;:\; i \in \mathbb{Z}\}, \tag{4.38}$$

where $\Delta\tau$ remains the previous iteration's timestep (informed by its own choice of η).

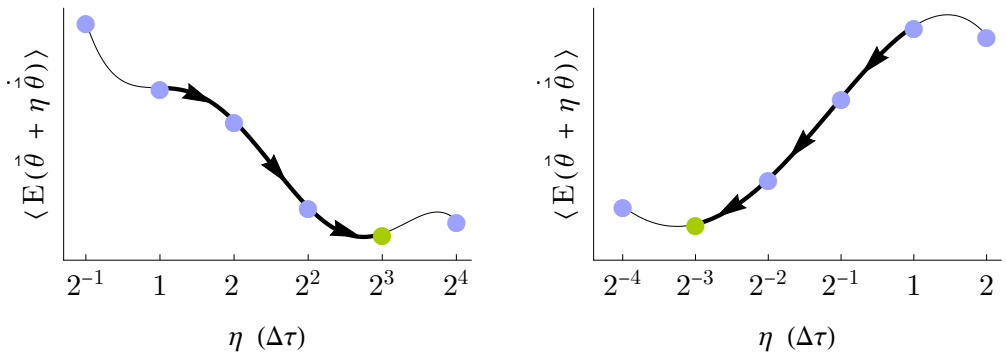


Figure 4.3: The ideal behaviour of our exponentially-adapting recursive timestep procedure, enlarging (left) and shrinking (right) the timestep in order to reach a local minimum (green point) of $\langle E(\vec{\theta} + 2^i \Delta\tau \dot{\vec{\theta}}) \rangle$ from $i = 0$, within a single iteration of imaginary time minimisation. The curved line is the experimentally unknown energy landscape, the points are measured energies, and the arrows indicate the direction of the changing center timestep argument to our adaption routine during recursion.

The initial timestep remains an insignificant hyperparameter to tune the numerically accessible grid space of η_i. We point out that we are *not* restricting our visited parameters $\vec{\theta}$ to an exponential grid, because $\dot{\vec{\theta}}$ may still freely vary continuously, just like when $\Delta\tau$ is fixed.

Our core technique is simple; during every iteration of imaginary-time, once $\dot{\vec{\theta}}$ has been evaluated, we experimentally measure the energy of ansatz states produced by parameters $\vec{\theta} + \eta_i \dot{\vec{\theta}}$ for $i \in \{-1, 0, 1\}$. This offers a first test of whether the timestep should shrink, stay the same, or enlarge, based on which achieves the minimum energy. If the timestep should change, we recurse and test the next adjacent η_i value ($i = -2$ or $i = 2$), continuing until we find a local minima along our discrete line-search. In this way, our candidate timesteps are changing exponentially, and we need only $\log_2(\delta\theta)$ additional energy measures to adapt the timestep to reflect a changing scale of variation $\delta\theta$ in the energy landscape. We picture this in Figure 4.3. Note there are several additional edge-cases which must also be considered, as better visualised in Figure 4.4.

For instance, since our unitary ansatz $\hat{\mathcal{B}}(\vec{\theta})$ must be periodic, there is a natural maxi-

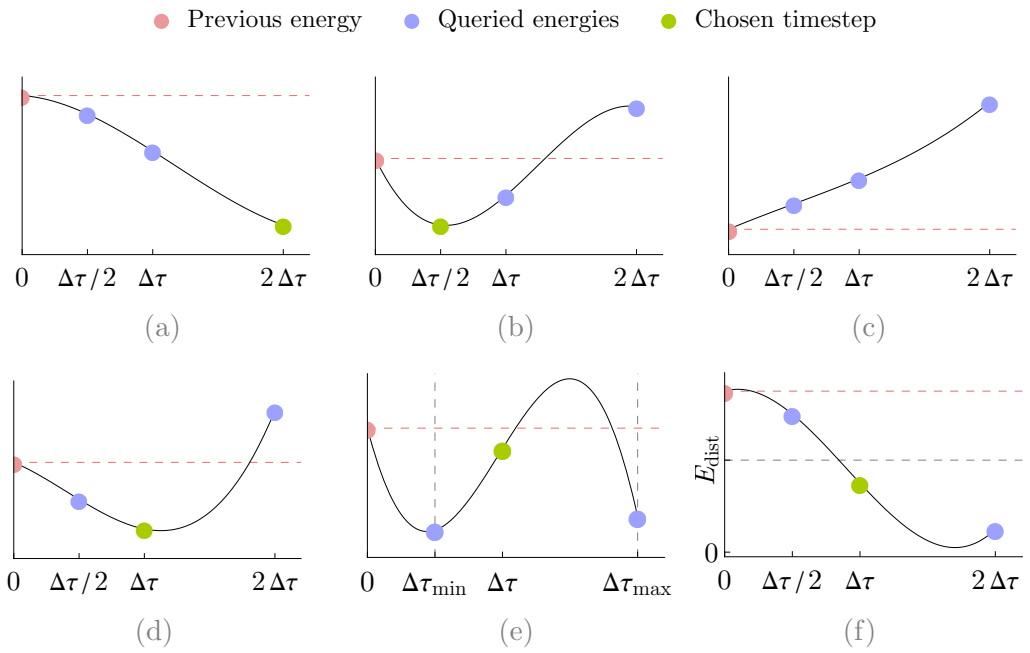


Figure 4.4: Example configurations which trigger the base cases and recursions of our adaptive time-step technique, presented in Algorithm 4.1. The horizontal axes are η, and the vertical axes are the energy of $\langle E(\vec{\theta} + \eta\,\vec{\dot{\theta}})\rangle$. The red point shows the energy of the previous imaginary-time iteration, $\langle E(\vec{\theta})\rangle$. The other points show the experimentally queried energies of this iteration, where $\Delta\tau$ is the previous imaginary-time iteration's time-step (or that of the previous recursion). In these examples, the green point corresponds to the chosen timestep (or that centered in the next recursion). The curved lines are merely examples of possible energy landscapes not known to the experimentalist. Each subfigure shows an instance whereby:

(a) The largest tested step is best; recurse and test $4\Delta\tau$.

(b) The smallest tested step is best, suggesting the present stepsize may be unstably large. Recurse and test $\Delta\tau/4$.

(c) All tested steps were unstably large, increasing the energy from the previous iteration. Recurse and rapidly shrink the step, testing $\Delta\tau/4$, $\Delta\tau/8$ and $\Delta\tau/16$.

(d) The previous iteration's timestep remained locally optimal; preserve $\Delta\tau$.

(e) The previous iteration's timestep has approached either $\Delta\tau_{\text{min}}$ of Eq. 4.46, or $\Delta\tau_{\text{max}}$ of Eq. 4.40, and cannot be meaningfully shrunk or enlarged respectively; preserve $\Delta\tau$.

(f) The previous iteration's timestep was sufficient to converge within our target distance E_{dist} of the known groundstate; halt minimisation.

mum parameter change loosely bound by

$$\Delta\theta_{\max} \;\leq\; \prod_i \Big(\text{period of } \hat{\mathcal{B}}_i(\theta_i)\Big)\,, \tag{4.39}$$

suggesting a maximum meaningful dynamic timestep of

$$\Delta\tau_{\max} \sim \frac{\Delta\theta_{\max}}{\max_i \dot{\theta}_i} \tag{4.40}$$

which, when all gate periods are equal (conventionally 2π or 4π), admits a tighter

$$\Delta\tau_{\max} = \frac{4\pi}{\max\limits_i \dot{\theta}_i}\,. \tag{4.41}$$

Meanwhile, a minimum meaningful timestep can be obtained by first bounding the energy variation

$$\delta|\,\langle E(\vec{\theta})\rangle\,| = \sum_i \frac{\partial\,|\,\langle E(\vec{\theta})\rangle\,|}{\partial\theta_i}\delta\theta_i \quad\leq\quad P\,\delta\theta \max_i \left|\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i}\right|. \tag{4.42}$$

Recall that in Chapter 2, we showed a Hamiltonian expressible as $\hat{H}_{\text{rec}} = \sum_j^T h_j\,\hat{\sigma}_j^{\otimes}$ and general ansatz gates expressible as $\hat{\mathcal{B}}_i(\theta_i) = \exp(\mathrm{i}\sum_k \lambda_{ik}(\theta_i)\hat{\sigma}_{ik}^{\otimes})$ admit energy derivatives of the form (where $|\,\langle\Psi|\Phi\rangle\,| \leq 1$)

$$\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i} \;=\; -\sum_{k,j} \lambda_{ki}{}'(\theta_i)\; h_j\; \Im\left[\langle\Psi|\Phi\rangle\right], \qquad \text{(re-expression of 2.33)}$$

$$\implies \qquad \left|\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i}\right| \;\leq\; \sum_j |h_j| \sum_k |\lambda_{ki}{}'(\theta_i)|, \tag{4.43}$$

$$\implies \qquad \max_i \left|\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i}\right| \;\leq\; \sum_j |h_j| \sum_k \max_{i,\phi} |\lambda_{ki}{}'(\phi)|. \tag{4.44}$$

When every parametrized gate of $\hat{\mathcal{B}}(\vec{\theta})$ is a linearly-parameterized Pauli gadget ($\sum_k \lambda_{ik}(\theta_i) \equiv \theta_i$), Equation 4.42 rearranges to

$$\delta\theta \;\geq\; \frac{\delta \left\langle E(\vec{\theta}) \right\rangle}{P \sum\limits_j |h_j|}. \tag{4.45}$$

By finally introducing a minimum meaningful energy change ΔE_{min} we endeavour to achieve each iteration, we can thusly conclude a loose minimum timestep of

$$\Delta\tau_{\text{min}} \;\sim\; \frac{\Delta E_{\text{min}}}{P \sum\limits_j |h_j| \;\max_i \dot{\theta}_i}\,. \tag{4.46}$$

We include these and several other considerations in our formalisation of our adaptive time-step technique, presented in Algorithm 4.1. We also formalise how to incorporate this routine into variational imaginary-time minimisation in Algorithm 4.2.

We remark that there are a plethora of further interesting improvements possible. For example, when the timestep recursion ends, having found a $\Delta\tau$ satisfying $E_{\Delta\tau} < E_{\Delta\tau/2},\; E_{2\Delta\tau}$, it is *not* generally true that

$$E_{\Delta\tau} = \min_{\Delta\tau/2 \,<\, \eta \,<\, 2\Delta\tau} \left\langle E(\vec{\theta} + \eta\,\dot{\vec{\theta}}) \right\rangle. \tag{4.47}$$

That is, we are *not* guaranteed to have found the *continuous* local minima of energy along η in that region, since we were restricted to discrete $\eta \in \{2^i \Delta\tau : i \in \mathbb{Z}\}$. Ergo, instead of returning $\Delta\tau$, we could begin a new exponentially-advancing search within the space *around* $\Delta\tau$, such as through the bisection method [222], investigating the energies produced by steps $\Delta\tau 3/4$ and $\Delta\tau 3/2$. This would significantly increase the complexity of the code and its number of edge-cases. We could also integrate our technique with the simulated annealing meta-heuristic [29], whereby we accept a timestep

Algorithm 4.1: Our adaptive time-step technique, invoked during each iteration of imaginary-time minimisation once $\vec{\dot{\theta}}$ has been obtained, to inform the step by which to update the parameters $\vec{\theta}$. Our pseudocode assumes the following global constants and hyperparameters:

ΔE_{min} an underbound of the energy change in an imaginary-time iteration
P the dimension of $\vec{\theta}$
χ the abs-sum of the $\hat{H}_{\text{rec}}$ Pauli-basis coefficients
e_0 the ground-state energy of $\hat{H}_{\text{rec}}$
E_{dist} the target distance to the groundstate of minimisation

---

adaptTimestep($\Delta\tau,\ E_{\Delta\tau/2},\ E_{\Delta\tau},\ E_{2\Delta\tau},\ E_{\textbf{prev}},\ \vec{\theta},\ \vec{\dot{\theta}}$ **)**

1. Set $\Delta\tau_{\text{max}} = 4\pi \,/\, \max_i \dot{\theta}_i$
2. Set $\Delta\tau_{\text{min}} = \Delta E_{\text{min}} / (P\,\chi\, \max_i \dot{\theta}_i)$
3. Set $E_{\text{min}} = \min\{E_{\Delta\tau/2},\ E_{\Delta\tau},\ E_{2\Delta\tau}\}$
4. If $(E_{\text{min}} - e_0) \leq E_{\text{dist}}$
   4.1. Return $\Delta\tau$
5. If $E_{\text{min}} > E_{\text{prev}}$:
   5.1. Measure $E_{\Delta\tau/4} = \langle E(\vec{\theta} + \Delta\tau/4\ \vec{\dot{\theta}})\rangle$
   5.2. Measure $E_{\Delta\tau/8} = \langle E(\vec{\theta} + \Delta\tau/8\ \vec{\dot{\theta}})\rangle$
   5.3. Measure $E_{\Delta\tau/16} = \langle E(\vec{\theta} + \Delta\tau/16\ \vec{\dot{\theta}})\rangle$
   5.4. Return
   adaptTimestep$(\Delta\tau/8,\ E_{\Delta\tau/16},\ E_{\Delta\tau/8},\ E_{\Delta\tau/4},\ E_{\text{prev}},\ \vec{\theta},\ \vec{\dot{\theta}})$
6. If $E_{\text{min}} = E_{\Delta\tau}$:
   6.1. Return $\Delta\tau$
7. If $E_{\text{min}} = E_{\Delta\tau/2}$:
   7.1. If $\Delta\tau/2 \ <\ \Delta\tau_{\text{min}}$:
      7.1.1. Return $\Delta\tau$
   7.2. Measure $E_{\Delta\tau/4} = \langle E(\vec{\theta} + \Delta\tau/4\ \vec{\dot{\theta}})\rangle$
   7.3. Return
   adaptTimestep$(\Delta\tau/4,\ E_{\Delta\tau/4},\ E_{\Delta\tau/2},\ E_{\Delta\tau},\ E_{\text{prev}},\ \vec{\theta},\ \vec{\dot{\theta}})$
8. If $E_{\text{min}} = E_{2\Delta\tau}$:
   8.1. If $2\Delta\tau \ >\ \Delta\tau_{\text{max}}$:
      8.1.1. Return $\Delta\tau$
   8.2. Measure $E_{\Delta 4\tau} = \langle E(\vec{\theta} + 4\Delta\tau\ \vec{\dot{\theta}})\rangle$
   8.3. Return
   adaptTimestep$(4\Delta\tau,\ E_{\Delta\tau},\ E_{2\Delta\tau},\ E_{4\Delta\tau},\ E_{\text{prev}},\ \vec{\theta},\ \vec{\dot{\theta}})$

Algorithm 4.2: A reformulation of variational imaginary-time minimisation to incorporate our adaptive time-step routine, as presented in Algorithm 4.1. We assume our recompilation Hamiltonian can be effected as $\hat{H}_{\text{rec}} = \sum_j h_j \, \hat{\sigma}_j^{\otimes}$ with a known groundstate energy e_0. Note that Line 5. merely initialises E_{prev} to a value larger than the first measured energy, and can be replaced with a flag.

1. Set $\Delta\tau = 1$ (arbitrary initial timestep).
2. Set hyperparameter E_{dist} (stopping distance from groundstate e_0).
3. Set hyperparameter ΔE_{min} (minimum meaningful energy change)
4. Set constant $\chi = \sum_j |h_j|$
5. Set initial value $E_{\text{prev}} = 999$
6. Set initial value $\vec{\theta}$ randomly
7. For each iteration of imaginary-time evolution:
   - 7.1. Evaluate $\underline{\underline{\mathcal{M}}}$ and $\vec{\mathcal{V}}$ as per Section 2.4.2
   - 7.2. Obtain $\dot{\vec{\theta}}$ as per Section 2.4.4.
   - 7.3. Measure $E_{\Delta\tau/2} = \langle E(\vec{\theta} + \Delta\tau/2 \, \dot{\vec{\theta}}) \rangle$
   - 7.4. Measure $E_{\Delta\tau} = \langle E(\vec{\theta} + \Delta\tau \, \dot{\vec{\theta}}) \rangle$
   - 7.5. Measure $E_{2\Delta\tau} = \langle E(\vec{\theta} + 2\Delta\tau \, \dot{\vec{\theta}}) \rangle$
   - 7.6. If $E_{\Delta\tau} < E_{\text{dist}}$
     - 7.6.1. Halt
   - 7.7. Evaluate
     $$\Delta\tau = \textbf{adaptTimestep}(\Delta\tau, \; E_{\Delta\tau/2}, \; E_{\Delta\tau}, \; E_{2\Delta\tau}, \; E_{\text{prev}}, \; \vec{\theta}, \; \dot{\vec{\theta}})$$
   - 7.8. Update the parameters by the Euler step:
     $$\vec{\theta} \to \vec{\theta} \; + \; \Delta\tau \, \dot{\vec{\theta}}$$
   - 7.9. Measure $E_{\text{prev}} = \langle E(\vec{\theta} + \Delta\tau \, \dot{\vec{\theta}}) \rangle$

which increases the energy with some probability inversely proportional to the energy change. This may assist our adapted imaginary-time algorithm to escape *tiny* local minima avoided with a fixed timestep. We will not however encounter the need for such extensions for the systems tested in this thesis.

4.4 Demonstration

We are now armed with a method of recompiling an input circuit $\hat{\mathcal{A}}\,|\text{in}\rangle$ into a template $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle$ and further shrinking it to $\hat{\mathcal{B}}_{\text{elim}}(\vec{\theta}'_0)\,|\text{in}\rangle$, by using imaginary-time minimisation with an adaptive timestep, an experimentally monitorable fidelity, and an optional luring meta-heuristic especially relevant to recompiling variational circuits. Let us demonstrate our method, albeit through classical emulation. We consider the recompilation of an interesting and practically relevant system, demonstrating a fascinating potential application of our scheme. Recall Section 1.6.2 introduced Li's method for variational real-time simulation, which we saw greatly outperform Trotterisation for the same quantum resource costs, but eventually reach a time of rapidly deteriorating fidelity ($t = 1.75$ in Figure 1.8). In this section, we will utilise recompilation to *extend* the longevity of Li's real-time simulation, by recompiling the ansatz state during simulation.

For real-time simulation, we adopt the 7-spin Hamiltonian

$$\hat{H}_{\text{sim}} = \sum_q B_q\,\hat{Z}_q + \sum_{q_1 q_2}\sum_{\hat{\sigma}\in\hat{X},\hat{Y},\hat{Z}} J^{\sigma}_{q_1 q_2}\,\hat{\sigma}_{q_1}\,\hat{\sigma}_{q_2}, \qquad B_i < 0,\;\; J^{\sigma}_{q_1 q_2} > 0,$$

(reiteration of 1.51)

where here subscript indices are qubit indices. We denote this physical Hamiltonian

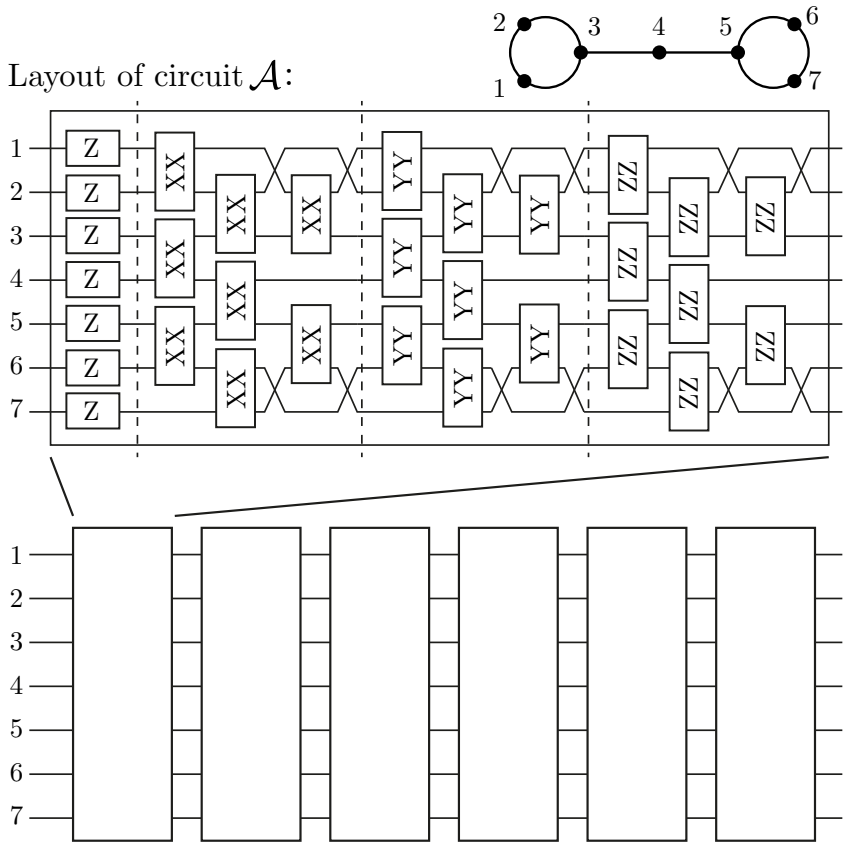


Figure 4.5: The circuit $\hat{\mathcal{A}}(\vec{\phi})$ which produces the state $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})|\,|\text{in}\rangle$ that we will recompile into $\hat{\mathcal{B}}(\vec{\theta})$ (Figure 4.6). The upper right figure summarises the two-qubit gate connections. The circuit structure is a 1st-order 6-repetition Trotter circuit of the 7-spin Hamiltonian 1.51, though with gate strengths produced by Li's variational real-time simulation method to time $t = 1.75$. For the purposes of recompilation, one can regard it as an arbitrary pattern of 186 unique non-Clifford gates (including 144 two-qubit gates). All gates are Pauli gadgets of the form $Z(\theta) = \exp(-i\frac{\theta}{2}\hat{Z})$, $ZZ(\theta) = \exp(-i\frac{\theta}{2}\hat{Z}\otimes\hat{Z})$, and similarly for the Y and Z gates. *This figure was produced by my co-authors in Reference [113].*

$\hat{H}_{\text{sim}}$ to disambiguate it from the fictitious Hamiltonian $\hat{H}_{\text{rec}}$ driving recompilation. The topology of $\hat{H}_{\text{sim}}$ is that shown in Figure 1.1. In order to create an interesting evolution, we select the initial state $|\Psi(0)\rangle = |1\rangle\,|+\rangle^{\otimes 6}$, where one qubit is orientated such that it has maximum energy with respect to its local field and the others have zero expected energy. We simulate this state to $t = 1.75$ using an ansatz with the same structure as a 6-repetition 1st order Trotter circuit (six layers of the circuit shown in Figure 1.2), which is around the time when the fidelity of Li's method begins to wane. We refer to this state as $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})|\,|\text{in}\rangle$, which is visualised by the dashed vertical line in Figure 1.8. This real-time state is highly non-trivial, and of significantly greater fidelity to the true evolution state (0.995) than the Trotter circuit of an equal number of gates ($\ll 0.1$). An alternative visualisation of $\hat{\mathcal{A}}(\vec{\phi})\,|\text{in}\rangle$ is shown in Figure 4.5.

We will attempt to recompile $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})|\,|\text{in}\rangle$ into a markedly different circuit structure $\hat{\mathcal{B}}(\vec{\theta})$, which we show in Figure 4.6, which is generally experimentally cheaper to effect and less expressive. Indeed, $\hat{\mathcal{B}}(\vec{\theta})$ happens to perform significantly worse as an ansatz to Li's real-time simulation from $t = 0$. We choose a simple recompilation Hamiltonian for which $|\Psi(0)\rangle$ is the ground state, namely

$$\hat{H}_{\text{rec}} = \hat{Z}_0 - \sum_{q>0}^{6} \hat{X}_q. \tag{4.48}$$

For now, we will not need to make use of luring nor an adaptive timestep. We now perform variational imaginary-time minimisation under $\hat{H}_{\text{rec}}$ with ansatz $\hat{\mathcal{B}}(\vec{\theta})^\dagger\,\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})\;|\text{in}\rangle$ to discover $\vec{\theta}_0$, as per Section 4.3.1. This process is shown in Figure 4.7, and produces a recompiled state $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle$ with an impressive fidelity of 0.998. Approximate recompilation has been successful despite the significant differences of the circuits.

Yet, we recompile further; we employ the optional post-recompilation optimisation

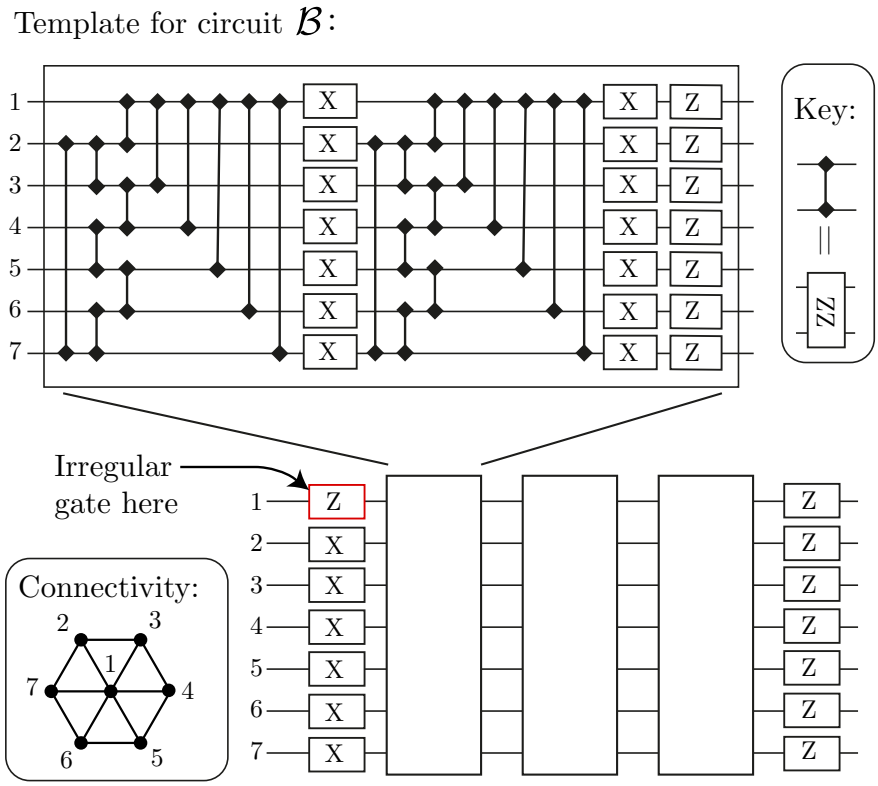


Figure 4.6: The structure of the recompiled circuit $\hat{\mathcal{B}}(\vec{\theta})$, before additional gate elimination, produced from $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})|\,|\text{in}\rangle$ of Figure 4.5. $\hat{\mathcal{B}}$ has a smaller family of gate types (YY and XX type gates are omitted), half as many two-qubit gates in total (72 rather than 144), but a larger number of single-qubit gates (77 versus 42). Moreover the connectivity of the two-qubit gates is markedly different, forming the triangular lattice shown in the inset. Notice $\hat{\mathcal{B}}$ includes a $\hat{Z}$ rotation on qubit state $|0\rangle$ which produces only a global phase, which we proved in Chapter 2 provides an important benefit to imaginary-time minimisation. *This figure was produced by my co-authors in Reference [113].*

routine of Section 4.3.2 to automatically eliminate additional gates from $\hat{\mathcal{B}}(\vec{\theta}_0)$. This is shown in Figure 4.8, and produces $\hat{\mathcal{B}}_{\text{elim}}(\vec{\theta}'_0)$. Remarkably, this removes a further 30 gates (11 single-qubit, 19 two-qubit) while retaining a final recompilatiom fidelity 0.995. The final circuit includes only 119 gates (whereas the original $\hat{\mathcal{A}}$ has 186) and moreover the number of two-qubit gates has been reduced almost to a third (from 144 to 53). Recall that all remaining two-qubit gates are all of a single type, $\exp(-i\frac{\theta}{2}\hat{Z}\otimes\hat{Z})$. We emphasise that these recompilation fidelities have been achieved with a non-trivial circuit $\hat{\mathcal{A}}(\vec{\phi}_0)$ that is *already* optimised in a sense, with respect to a simple Trotter circuit, through the use of Li's algorithm. Applying our recompilation to the same circuit with gate strengths prescribed by Trotterisation produces even *more* dramatic compression to a $\sim 50\%$ depth circuit with a negligible fidelity loss of $\Delta F = 0.0002$.

This concludes our efforts to recompile this 7-spin example. We now explore whether such efforts were worthwhile towards our goal of extending variational real-time simulation. The output circuit $\hat{\mathcal{B}}_{\text{elim}}(\vec{\theta}'_0)\,|\text{in}\rangle$ approximates the true $t = 1.75$ evolution state with fidelity ~ 0.992, while requiring only 64% of the gates (37% of the two-qubit gates) of the original $\hat{\mathcal{A}}\,|\text{in}\rangle$ circuit. We hence have room to 'append' new parametrized gates onto $\hat{\mathcal{B}}$ or $\hat{\mathcal{B}}_{\text{elim}}$, initially producing the identity, to recover similar quantum costs as $\hat{\mathcal{A}}$. We choose to append subcircuits admitted by single-cycle Trotterisation, albeit where the gate strengths become independent parameters, which we denote by $\hat{\mathcal{T}}$. We task these compressed and padded circuits with *continuing* variational real-time simulation for $t > 1.75$, letting the parameters freely vary under the prescription of Li's method. The results are shown in Figure 4.9. We see that indeed Li's method can continue to accurately simulate the dynamics of the 7-spin system beyond $t > 1.75$ when using ansatz circuits of the form $\hat{\mathcal{T}}^3\,\hat{\mathcal{B}}$ and $\hat{\mathcal{T}}^4\,\hat{\mathcal{B}}_{\text{elim}}$. The right panel of Figure 4.9 reveals such circuits involve only modest increases of single-qubit gates, but no more two-qubit gates than originally employed by $\mathcal{A}$. One may wonder whether this improvement was

due to a fault in our original selection of $\hat{\mathcal{A}} = \hat{\mathcal{T}}^6$ as the ansatz circuit for Li's real-time simulation, and that $\hat{\mathcal{B}}$ was always a superior choice. The red lines in Figure 4.9 show this is *not* the case; simulation using $\hat{\mathcal{B}}$ from either $t = 0$ or $t = 1.75$ is grossly inaccurate. Indeed, that circuit $\hat{\mathcal{A}}$ leverages the Trotter gate strengths in the early time evolution (revealed by Figure 1.8) suggests it is a potentially ideal circuit to capture the early dynamics.

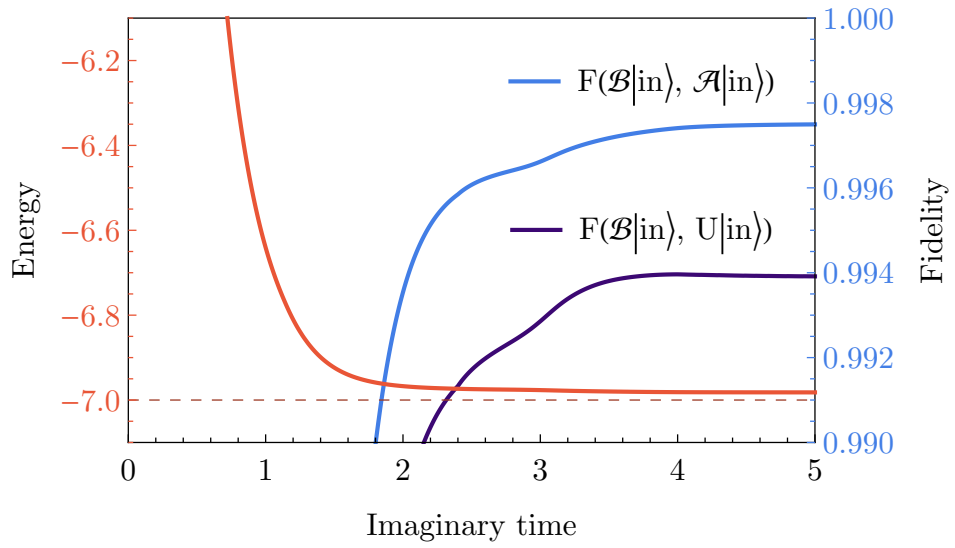


Figure 4.7: The fidelity and energy (under $\hat{H}_{\text{rec}}$, with minimum $e_0 = -7$) during recompilation of $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})\,|\text{in}\rangle$ (Figure 4.5) into $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle$ (Figure 4.6). While not directly measurable, an experimentalist can bound the fidelity from the measurable energy as per Equation 4.31. We also include the darker blue curve to show the fidelity between the recompiling state and the *true* experimentally-inaccessible real-time state of the 7-qubit spin system 1.51.

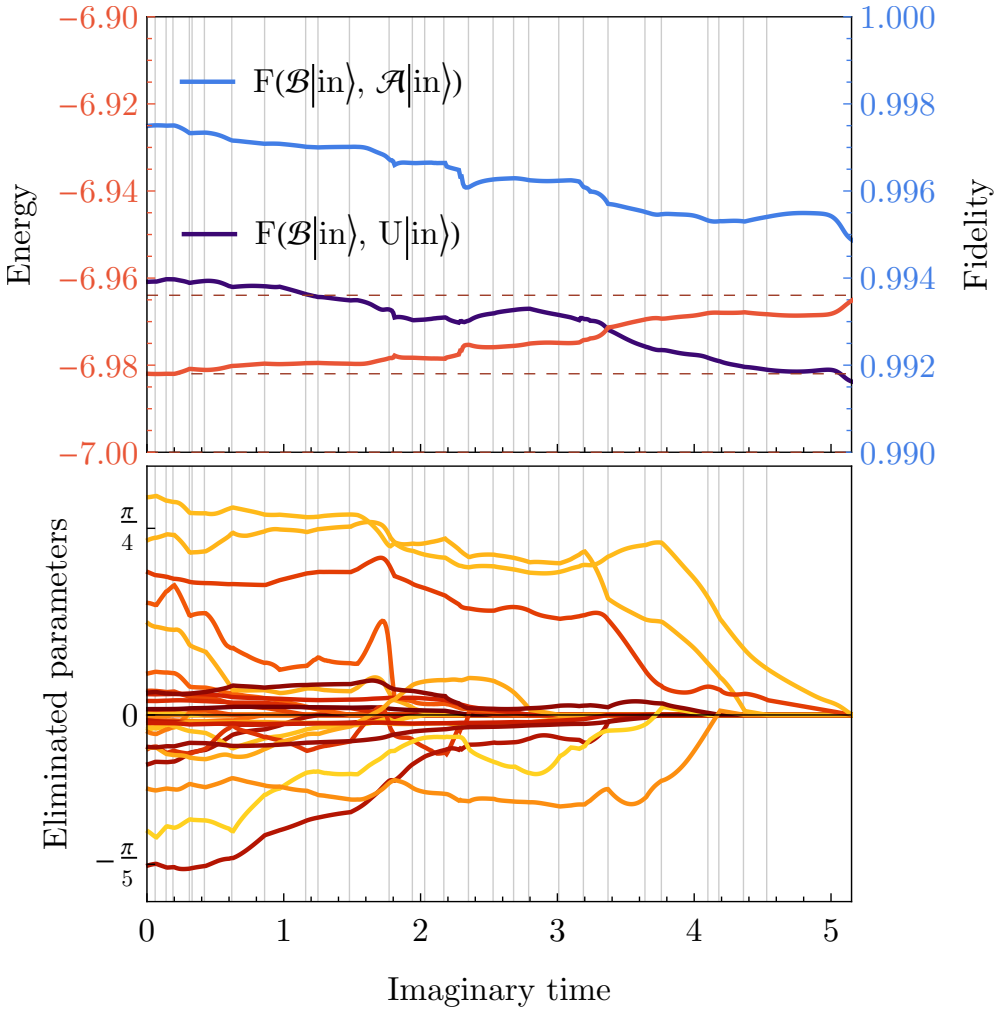


Figure 4.8: Additional optimisation of $\hat{\mathcal{B}}(\vec{\theta}_0)\,|\text{in}\rangle$ (Figure 4.6) into $\hat{\mathcal{B}}_{\text{elim}}(\vec{\theta}'_0)\,|\text{in}\rangle$ (of 30 fewer gates), by the post-recompilation gate elimination strategy of Section 4.3.2. We slowly deactivate 'weak' gates during further imaginary-time minimisation, removing them once they produce the identity operation (here, reaching $\theta_i = 0$ as indicated by a vertical line). We continue until the experimentally measurable energy defect has doubled, reducing the fidelity of recompilation by only $\Delta F = 0.003$. The darker blue curve shows that $\hat{\mathcal{B}}_{\text{elim}}(\vec{\theta}'_0)\,|\text{in}\rangle$ now has a 0.992 fidelity with the *true* inaccessible real-time evolution state $\hat{U}\,|\text{in}\rangle$, which the original ansatz state $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})\,|\text{in}\rangle$ produced with fidelity 0.995.

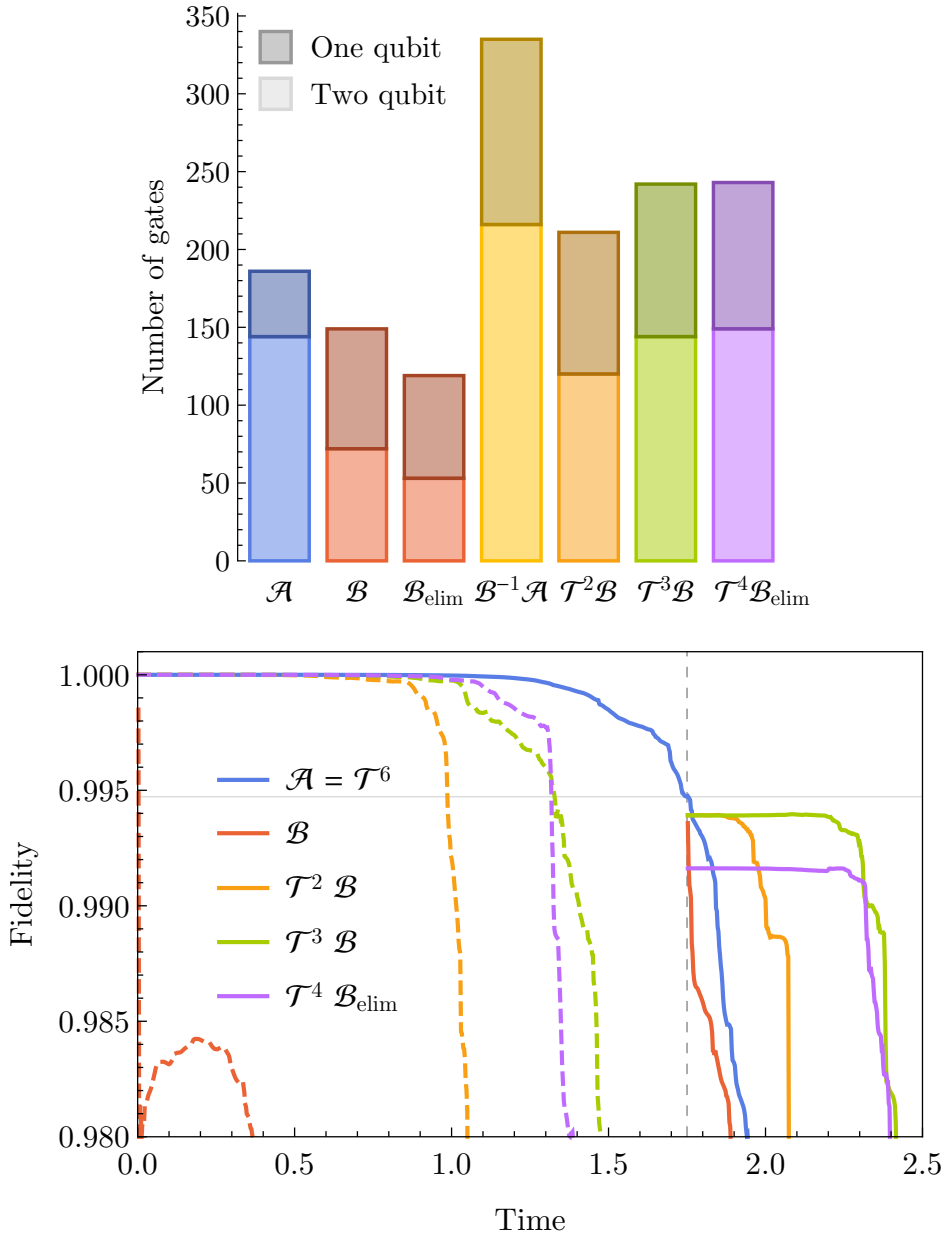


Figure 4.9: Real-time simulation of the 7-spin system 1.51 using Li's variational algorithm [160] (as introduced in Section 1.6.2) with various ansatz circuits, from initial state $|\text{in}\rangle = |\Psi(0)\rangle = |1\rangle\,|+\rangle^{\otimes 6}$. In blue, the original structure $\hat{\mathcal{A}}(\vec{\phi})$ which we pause at $t = 1.75$ before its fidelity drops precipitously. We then recompile $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})\,|\text{in}\rangle$ into $\hat{\mathcal{B}}(\vec{\phi}_0)\,|\text{in}\rangle$ and $\hat{\mathcal{B}_{\text{elim}}}(\vec{\phi}'_0)\,|\text{in}\rangle$ using our recompilation algorithm, and pad them with Trotter cycles $\hat{\mathcal{T}}$ (where gate strengths are free parameters). Li's real-time simulation resumes, using these circuits as as ansätze, to sustain high fidelity simulation for a further 0.4 time units. The right subplot shows these circuit costs.

Our results here have several interesting implications, both about our recompilation algorithm and generally about variational simulation.

- Different time regimes of real-time simulation using Li's variational method appear to benefit from different ansatz circuits. That is, ansätze which accurately generate early-time dynamics may be outperformed by similarly sized but structurally distinct ansätze at later times. Counter-intuitively, this does not appear to result from an increased capacity to produce complicated entangled states, since our recompiled and padded ansätze had fewer entangling gates than $\hat{\mathcal{A}}$.

- Furthering the previous observation, ansätze which mimic the circuit structure prescribed by Trotterisation can reproduce the Trotter strengths then outperform them for early dynamics, suggesting their utility for a "first" ansatz.

- Variational real-time simulation can be paused, have its ansatz gate recompiled into a different structure, and resumed for a significant benefit to the fidelity and range of accurate simulation. We point out this action demands no additional quantum facilities, since this process is itself a variational routine.

4.5 Testing

We have seen a promising success of our recompilation algorithm on a challenging albeit small 7-qubit system. We now perform more systematic investigations into the scaling and noise resilience of our recompilation algorithm.

4.5.1 Scaling

In this section, we will explore how our recompilation algorithm scales to larger system sizes in noise-free settings. We will adopt the adaptive timestep technique of Section 4.3.5, but we will *not* perform the additional optimisation routine of Section 4.3.2, nor will we immediately employ luring (Section 4.3.4).

As inputs, we generate the easily scalable squeezed states [223, 224],

$$\mathcal{A}(\phi_1, \phi_2, \phi_3) = \prod_t Y_t(\pi/2) \prod_{c,t>c} \mathrm{C}_c[Z_t(\phi_1)] \times \prod_{t,c>t} \mathrm{C}_c[Z_t(\phi_1)] \prod_t X_t(\phi_2) Z_t(\phi_3), \tag{4.49}$$

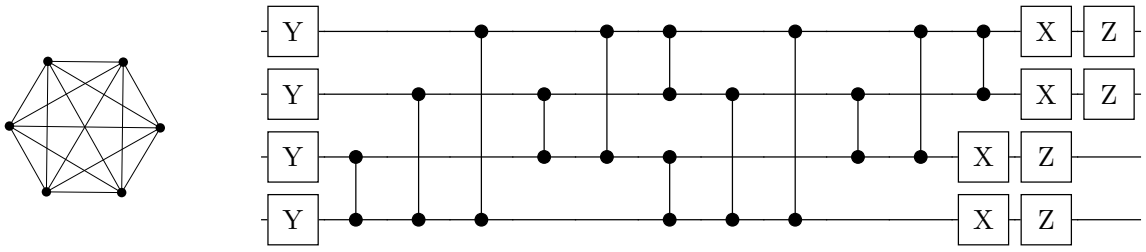
where U_t notates gate U acting on qubit t, and $C_c[U_t]$ notates the same gate with additional control qubit c. As before, X, Y, Z compactly denote rotations. For each test of a given circuit size, the parameters ϕ_1, ϕ_2, ϕ_3 are chosen uniformly randomly in $[0, 2\pi)$. An example circuit which prepares a 4-qubit squeezed state is shown in Figure 4.10. We choose a scalable family of target template circuits with distinct gate topologies and gate sets from the input circuits;

$$\hat{\mathcal{B}}(\vec{\theta}) = \hat{V}\,\mathrm{reverse}(\hat{V}), \tag{4.50}$$

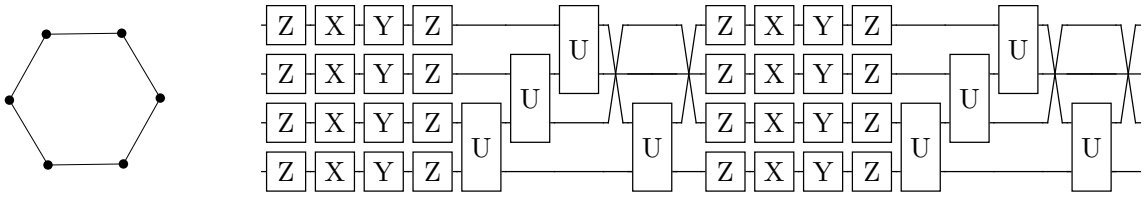
$$\text{where} \quad \hat{V} = \left(\prod_t Z_t X_t Y_t Z_t \prod_t U_{t,t+1} \right)^2, \tag{4.51}$$

$$\text{and} \quad \hat{U} = \begin{pmatrix} 1 & & & \\ & e^{i\theta} & & \\ & & e^{i\theta} & \\ & & & 1 \end{pmatrix}, \tag{4.52}$$

where every gate features a unique parameter, and U is the parameterised SWAP gate. This circuit is also visualised in Figure 4.10.



(a) $\hat{\mathcal{A}}(\phi_1, \phi_2, \phi_3)$ (Eq. 4.49)



(b) half of $\hat{\mathcal{B}}(\vec{\theta})$ (Eq. 4.50)

Figure 4.10: Qubit connectivities (left) and circuits (right) of a four-qubit example input (top) and output (bottom) of our recompilation scaling tests. The top circuit acting on $|0\rangle^{\otimes 4}$ generates a squeezed state.

Note that $\hat{\mathcal{A}}(\phi_1, \phi_2, \phi_3)$ and $\hat{\mathcal{B}}(\vec{\theta})$ differ significantly, and there is no exact assignment $\vec{\theta}$ for which $\hat{\mathcal{A}}(\phi_1, \phi_2, \phi_3)\,|0\rangle \equiv \hat{\mathcal{B}}(\vec{\theta})\,|0\rangle$ besides when both are the identity. Furthermore, the *optimum* value of $\vec{\theta}$ for which $|\,\langle 0|\,\hat{\mathcal{B}}^\dagger(\vec{\theta})\hat{\mathcal{A}}(\phi_1, \phi_2, \phi_3)\,|0\rangle\,|$ is maximum, is dimension dependent and may hence may prove a progressively worsening metric of recompilation quality. Rigorously qualifying the performance of our recompilation scheme in a scale-invariant remains a challenge which we do not presently address.

Recompilation will be driven by the deceptively simple but very useful n-qubit Hamiltonian

$$\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |0\rangle\langle 0|^{\otimes n}\,. \tag{4.53}$$

Notice its spectral gap remains 1 as the system size n grows; the first excited state is $(2^n - 1)$-degenerate, and the fidelity of the recompiled state is exactly $1 - \langle E(\vec{\theta})\rangle$, as per Equation 4.24. Furthermore, for the sake of numerical simulation, this Hamiltonian

offers rapid classical evaluation of the energy and its derivatives, as involved in the variational imaginary-time subroutines. Recall from Section 2.6.1 that simulating these subroutines involves numerically obtaining derivatives of the ansatz state $|\psi(\vec{\theta})\rangle = \hat{\mathcal{B}}^\dagger(\vec{\theta})\hat{\mathcal{A}}\,|0\rangle$ in the parameter space, $\{\frac{\partial|\psi(\vec{\theta})\rangle}{\partial\theta_i} : i\}$. Letting $[\Psi]_0$ denote the *first* complex amplitude of vector $|\Psi\rangle$, our simple recompilation Hamiltonian permits the evaluation of the energy and its derivative in time $\mathcal{O}(1)$ and $\mathcal{O}(2^n)$ respectively, via

$$\langle E(\vec{\theta})\rangle = \langle\psi(\vec{\theta})|H_{\text{rec}}|\psi(\vec{\theta})\rangle = 1 - [\psi(\vec{\theta})]_0 \tag{4.54}$$

$$\begin{aligned}\mathcal{V}_i &= \Re\left[\langle\psi|H_{\text{rec}}|\frac{\partial\psi(\vec{\theta})}{\partial\theta_i}\rangle\right] \\ &= \Re\left[\langle\psi|\frac{\partial\psi(\vec{\theta})}{\partial\theta_i}\rangle - [\psi(\vec{\theta})]_0{}^*\left[\frac{\partial\psi(\vec{\theta})}{\partial\theta_i}\right]_0\right].\end{aligned} \tag{4.55}$$

That is, by a *single* inner product and a mere product of scalars. For a general Hamiltonian, these quantities could otherwise take time $\mathcal{O}(2^{2n})$ to evaluate.

We fix the number of iterations to be sufficiently large such that all tests have converged (in our tests, 200 iterations), and measure the final fidelity achieved. We found that use of adaptive time-step over a heuristically-motivated fixed time-step improves convergence drastically. The cost is very modest: on average about 1% more energy measurements. A demonstration of the time-step evolving adaptively during one of the subsequent scaling tests is presented in Figure 4.11.

We simulate recompilation of squeezed states from 3 to 20 qubits, each time performing 20 tests with random squeezed state parameters $\vec{\phi}$, and random initial ansatz parameters $\vec{\theta}$. At each of the 200 iterations of variational imaginary-time evolution, the linear equations were solved using singular value decomposition (SVD), with no truncation, in lieu of Tikhonov regularisation, merely as a simulation convenience. The results are

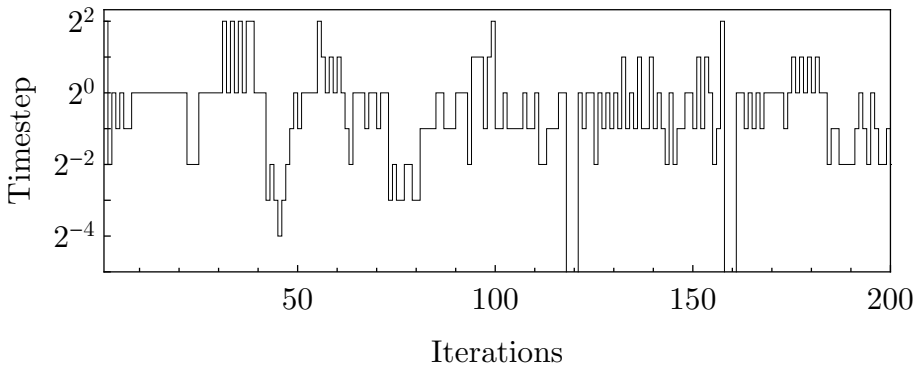


Figure 4.11: An example of our adaptive timestep technique updating the step size during recompilation (and therein, imaginary-time minimisation) of squeezed states, as part of our scaling tests.

presented in Figure 4.12. Despite excellent recompilation of the 16-qubit state to 99% fidelity, larger squeezed states performed significantly worse. The maximum fidelity achieved for 19 qubits (highlighted along the red line in Figure 4.12) is 59%, though the average is 11%. We speculate this particular squeezed state may be anomalously difficult to recompile due to, for example, prescribing strong entanglement between non-neighbouring qubits, for which there may be a small space of proximate ansatz states. Converging into such a space from an initial random set of parameters becomes decreasingly likely with increasing number of qubits [125].

As cautioned, a test of scaling performance may be more difficult than recompiling a particular state. We demonstrate this by recompiling the anomalously difficult 19 qubit circuit, using the same total number of iterations (200), but introducing 10 stages of luring, and truncation into SVD. The ansatz parameters now begin at zero, and all gate strengths in $\mathcal{A}\left|0\right\rangle(\phi_1, \phi_2, \phi_3)$ (including the $Y(\pi/2)$ terms) are initially scaled to $\times 0.1$, and then incrementally restored to their full values over the course of recompilation. We furthermore introduce truncation of singular values below 10^{-5} when solving the linear equations involved in variational imaginary-time evolution. At no extra algorithmic

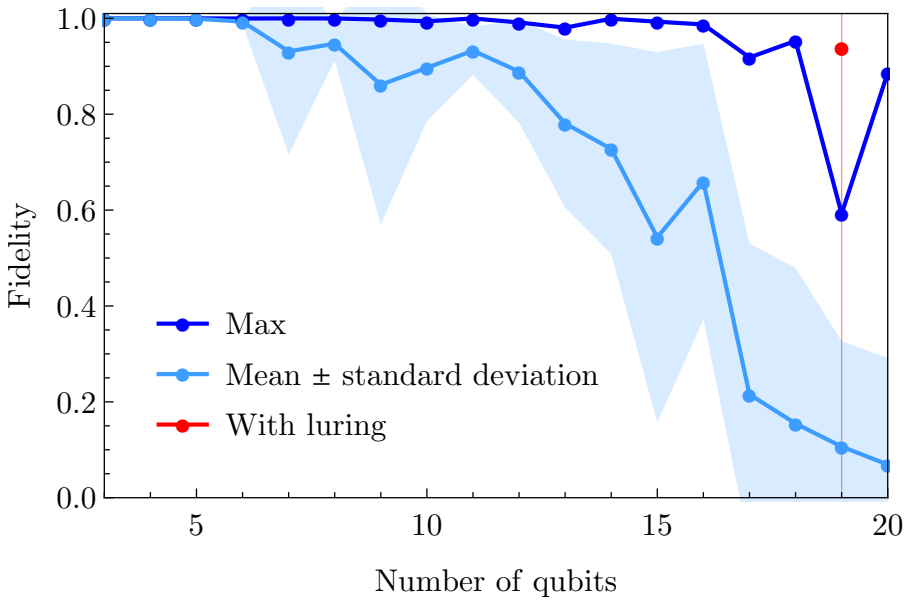


Figure 4.12: The performance of squeezed state recompilation ($\hat{\mathcal{A}}$ to $\hat{\mathcal{B}}$ of Figure 4.10) driven by 200 iterations of imaginary time minimisation, tested 20 times for each system size. All tests employ no luring *except* the red point, which applies 10 stages of luring to increase the fidelity from 59% to 94% (shown in Figure 4.13).

cost, the same 19 qubit case is now recompiled to a final 94% fidelity, and is denoted by the red point in Figure 4.12. The changing fidelity achieved by recompilation against the lured state is shown in Figure 4.13.

4.5.2 Noise

*This section only summarises work performed by my co-author in our joint manuscript. We direct the interested reader to a more thorough treatment in Section **5** of Reference [113], which includes expanded motivations and incorporations of error mitigation.*

We now return to compilation of the demonstrated 7-spin system from Section 4.4, this time in the presence of depolarising decoherence and error mitigation. This means

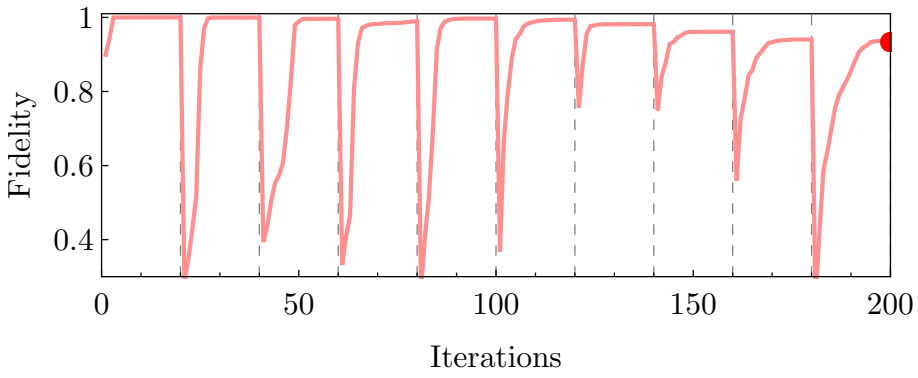


Figure 4.13: Recompilation of the anomalously difficult 19-qubit squeezed state in Fig 4.12, using 10 stages of luring. The ultimate fidelity of 94% is denoted in Figure 4.12 as a red dot.

making the follow changes to our earlier simulations.

1. We migrate from statevector to density matrix simulation, inducing correlated two-qubit depolarising errors of error rate ξ after every two-qubit gate, via channel

$$\mathcal{D}_{\Delta}[\xi](\rho) = \left(1 - \frac{16}{15}\xi\right)\rho + \left(\frac{16}{15}\xi\right)\frac{\hat{\mathbb{1}}^{\otimes 2}}{2}. \qquad \text{(re-expression of Eq. 1.44)}$$

   Single-qubit depolarising of strength $\xi/10$ follows every single-qubit gate, to reflect their typically higher fidelity in today's experiments. We do not include the error due to shot noise and finite sampling of the variational observables, since efficient simulation requires a bespoke strategy like that presented in Section 2.6.2. A realistic emulation would also require our quantum algorithm to make use of frugal sampling techniques [225, 226].

2. We substitute the imaginary-time matrix ($\underline{\underline{\mathcal{M}}}$ of Equation 2.22), which we showed equivalent to the quantum natural gradient in Section 2.4.5, with a recently developed noise-aware generalisation of the latter quantity [227].

3. We employ error mitigation, specifically the extrapolation technique [160, 228] which is a well-established and experimentally validated [229] method with a simple protocol: Each observable required for the optimisation process is estimated by measuring at noise level ξ, and again at noise level 2ξ. The recorded value is then the observable extrapolated to $\xi \to 0$ using the presumption of exponential decay [230].

We performed simulations with noise severity ξ ranging between 0.1% and 1%, as presented in Fig. 4.14. At the upper limit, even the circuit $\hat{\mathcal{A}}$ alone produces untenably noisy outputs. Yet, despite the total recompilation circuit $\hat{\mathcal{B}}(\vec{\theta})^\dagger\hat{\mathcal{A}}$ involving ~ 250 gates, recompilation succeeds remarkably well in its determination of final parameters $\vec{\theta}_\xi^{\,n}$. That is, the effect of noise *during* the recompilation process only negligibly worsens the error of state $\hat{\mathcal{B}}(\vec{\theta}_\xi^{\,n})$, itself in the presence of noise. This suggests that in circumstances where the noise is not too high for circuits to be of some value, the recompilation remains useful and indeed remarkably accurate. We conclude that recompilation with noisy circuits is entirely practical. This exercise also suggests that it may ultimately be profligate sampling requirements, rather than any inherent limitations of the scheme, that will bound the use of noisy circuits in practical compilation scenarios.

4.6 Discussion

In this chapter, we developed a novel technique to employ an imperfect quantum computer to approximately recompile a quantum circuit upon a fixed input state. The input circuit is re-expressed into a user-specified circuit structure which we have seen can be wildly different from the input circuit in its depth, topology, and gate family. We then produced an optional post-recompilation optimisation routine to automatically remove

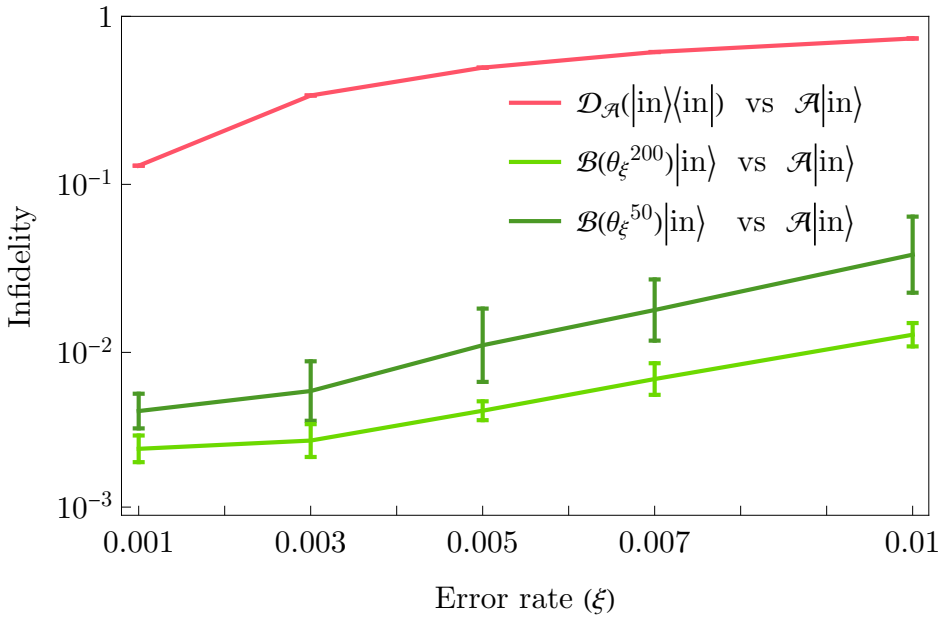


Figure 4.14: Recompilation of the 7-spin system in the presence of significant (classically simulated) experimental noise. We recompile $\hat{\mathcal{A}}(\vec{\phi}_{t=1.75})\,|\text{in}\rangle$ (Figure 4.5) into $\hat{\mathcal{B}}(\vec{\theta}_\xi^{\,n})\,|\text{in}\rangle$ (structure of Figure 4.6) under $\hat{H}_{\text{rec}} = \hat{\mathbb{1}} - |\text{in}\rangle\langle\text{in}|$. The horizontal axis ξ is the severity of two-qubit depolarising noise following each two-qubit gate (single qubit gates receive severity $\xi/10$) in all circuits. The green lines show the infidelity between input $\hat{\mathcal{A}}\,|\text{in}\rangle$ and the pure state generated by $\mathcal{B}$ with parameters $\vec{\theta}_\xi^{\,n}$, as learned from n iterations of ξ-severe noisy quantum natural gradient. Specifically, this is $1 - |\,\langle\text{in}|\,\mathcal{B}(\vec{\theta}_\xi^{\,n})^\dagger\mathcal{A}\,|\text{in}\rangle\,|^2$, and is the error introduced by the noisy compilation process itself. The red line (uppermost) is included for context, and shows the infidelity between the ideal target state $\mathcal{A}\,|\text{in}\rangle$ and its noisy counterpart $\mathcal{D}_{\mathcal{A}}(|\text{in}\rangle\langle\text{in}|)$, where $\mathcal{D}_{\mathcal{A}}$ is the channel prescribed by inserting noise into $\mathcal{A}$. Concretely, the red line shows $1 - |\,\langle\text{in}|\,\mathcal{A}^\dagger\mathcal{D}_{\mathcal{A}}(|\text{in}\rangle\langle\text{in}|)\mathcal{A}\,|\text{in}\rangle\,|^2$. One observes that the green lines are orders of magnitude lower than the red, confirming that recompilation is remarkably noise-robust, since the error of noisy recompilation is occluded by the noise in the ultimate circuits. Generally 9 independent runs were collated for each plotted value of ξ, except $\xi = 0.01$ for which there were 14 runs. *The data for this plot was generated by my co-authors in Reference [113], though visualised and captioned myself.*

the least important gates from the circuit, again leveraging a quantum device. To assist recompilation, we developed two novel meta-algorithms which we expect to be of wider utility in general variational algorithms; luring to boost convergence reliability, and exponentially-adapting timestep to accelerate convergence and relieve the nuisance of selecting a fixed timestep.

We numerically demonstrated our recompilation algorithm on an interesting problem; extending the accuracy of Li's real-time variational simulation [160] (as introduced in Section 1.6.2) by on-the-fly recompilation of the ansatz. Despite being a challenging 7-qubit example, recompilation was remarkably effective. We next tested the scalability of our algorithm on squeezed states up to 20 qubits. Although our results are hard to interpret due to the unmeasured but known increasing difficulty of the recompilation task, we demonstrated that our luring and adaptive timestep techniques can drastically improve recompilation with no resource penalty. Finally, we re-tested our 7-qubit example in the presence of decoherence, demonstrating that physical noise totally occludes the resulting algorithmic error of recompilation. As such, our recompilation strategy is practically robust to noise.

Our results are very encouraging, but there are a multitude of caveats about our analysis and known limitations of our scheme, including:

- Compilation from the original circuit $\mathcal{A}$ is not to an equivalent unitary circuit, but rather to a target circuit $\mathcal{B}$ that (ideally) has the same effect on just one specific input state $|\text{in}\rangle$, so that $\mathcal{B}|\text{in}\rangle = \mathcal{A}|\text{in}\rangle$.

  This is a profoundly more permissive goal, but is in fact the *right* goal for many quantum algorithms including so-called hybrid quantum-classical approaches [2].

- While the specific input state $|\text{in}\rangle$ can have any form, it is necessary that we

'understand' it well enough to be able to write down a (fictitious) Hamiltonian for which it is the ground state.

- The approach we describe here has unproven scaling as the circuits grow beyond 20 qubits. We do not expect an imperfect 20 qubit quantum device to perform our algorithm faster than our classical simulations herein. But compilation of circuits beyond the classical simulation limit will require quantum hardware, and will consume considerable time on that hardware; we have not motivated this threshold. We note that all-classical software to recompile circuits involving parameterised gates does exist [212] and can make significant savings. However no classical compiler can be expected to approach optimality for general circuits since even the task of verifying that two circuits are near-identical is QMA-complete [231]. One may wonder whether classical simulation of our recompilation algorithm, using all the algorithmic high-performance techniques to be introduced in Chapters 5 and 6, would outperform the aforementioned all-classical software. This is ultimately a fascinating question about asymptotically *worsening* algorithms to improve their *parallelisation*, and one we hope to answer in future research.

- We have here restricted our attention to circuits formed of unitary gates, so that our complete circuits $\mathcal{A}$ and $\mathcal{B}$ are themselves unitary. Generalising to non-unitary circuits would appear possible however.

- A more comprehensive compiler might automatically propose and test different templates for $\mathcal{B}$, rather than requiring the user to specify one. This would be a higher-level process operating above the compilation we describe; prior work on all-classical optimisation could be employed here [212], or variational studies which use ansätze that auto-evolve [232, 233, 234].

Ultimately the recompilation method we describe here, being dependent on variational energy-minimisation, will be subject to essentially the same scaling as all variational methods. The corresponding issues of 'barren plateaus' [125, 204] and local versus global cost functions [235], etc, are under very active investigation in the field [172, 133, 227]. Its practicality will also depend on the total prescribed measurement costs. While not here formalised, we do at least know that the cost of the metric tensors involved in imaginary time evolution and noisy quantum natural gradient are marginal [227], as numerically demonstrated in Chapter 2.

If indeed our algorithm becomes experimentally feasible, it has several interesting, potential applications, such as in the preparation of Gibbs states [128, 129], appearing in machine learning via Boltzmann machines [236], and in the emerging field of quantum semi-definite programming [237, 238]. It may also serve to shrink the energy sampling costs of general variational minimisation methods, as employed in Chapters 2 and 3, provided approximate recompilation is achievable with fewer measurements than needed of the energy estimation. In the near-term, it serves as a useful classical tool for researchers to approximately recompile their circuits by emulating our algorithm in their programming language and tools of choice.

Chapter 5

Classical Variational Simulation

Contents

5.1 Foreword

This section presents the unpublished works presently under review

- **T. Jones**, J. Gacon

  Efficient calculation of gradients in classical simulations of variational quantum algorithms

  arXiv:2009.02823 (2020) (link)

- **T. Jones**

  Efficient classical calculation of the Quantum Natural Gradient

  arXiv:2011.02991 (2020) (link)

and the manuscript in preparation

- **T. Jones**

  Faster emulation of noisy quantum variational algorithms

which devise novel classical algorithms for simulating variational quantum algorithms. I developed these algorithms, wrote all three manuscripts, and performed all derivation. In the first manuscript, my co-author Julien Gacon implemented the algorithm and performed all benchmarking. The submitted manuscripts were developed during short internships with IBM Research UK and Quantum Motion London, and submitted for publication with permission. Their respective algorithms have been implemented in IBM's Qiskit simulator [239] by my co-author, and in Oxford's QuESTlink simulator [112] by myself. The final manuscript in preparation generalises the previous schemes to density matrices and parameterised channels, and its prescribing algorithms have already been implemented in QuESTlink by myself.

5.2 Introduction

Classical simulation of variational quantum algorithms has been an indispensable tool in the research of this thesis so far. This is in part due to the limited analytic investigation that such algorithms typically permit. In lieu of classically emulating the experimental protocols, we have so far described a variety of bespoke variational simulation methods. Section 2.6.1 described how to avoid decomposing imaginary-time quantities into individually simulated observables while still treating the ansatz as a black-box. Section 2.6.2 introduced noise into statevector simulations by random perturbation of variational observables, and Section 4.5.1 saw the strategic use of a recompilation Hamiltonian which admitted cheap variational observables.

These bespoke optimisations were necessary because direct classical simulation of the studied variational algorithms was untenably inefficient. Somewhat surprisingly, simulation was limited to few qubits because of the relative expense of population the variational quantities, rather than that of a one-shot circuit evaluation. This is an unintuitive result, which we formalise in the next section.

The wider research community has developed implementation-level strategies to combat the insidious expense of classically simulating quantum variational algorithms. So-called "batch" strategies have emerged for parallel evaluation of entire circuits, allowing simultaneous evaluation of an ansatz circuit for different parameter values. [240, 241]. Though they admit the same asymptotic costs, they can use parallel or distributed hardware to, for example, simultaneously evaluate $\partial \langle E \rangle / \partial \theta_i$ for several values of i.

Such methods turn out unnecessary. By dissecting the recurrent forms of variational quantities, it is possible to asymptotically accelerate simulation of quantum gradient descent and natural gradient (and therefore, imaginary-time evolution). In this chapter,

we will derive fixed-memory algorithms which are a factor $\approx P$ faster than their typical forms when simulating the aforementioned P-parameter variational algorithms. Our algorithms will only prescribe a change to the usual order of simulated gates and inner products invoked by their naive counterparts. As such, we do not specify *how* to simulate such operations, which we instead defer to Chapter 6.

5.3 Motivation

It is easy to assume that the exponentially growing costs of classically simulating a single gate will always dominate the runtime in practical settings. After all, if the circuit depth is increased polynomially, the number of qubits can be reduced logarithmically to recover a similar simulation time. This may lead one to erroneously expect that variational quantum algorithms can be simulated in a similar runtime to that of one-shot circuits of only slightly more qubits. In other words, variational simulation can be substituted by slightly shrinking the simulation size. We will demonstrate this is not the case, and that classical simulation of variational algorithms can be insidiously expensive.

Consider a family of circuits with fixed depth c upon m qubits (with ergo $c\,m$ gates). Imagine m is the maximum which can be simulated using statevector methods (presented in the next chapter) within some acceptable time limit;

$$\Delta t_{\max} \sim c\,m\,2^m. \tag{5.1}$$

Next, consider the same family of circuits as n-qubit $(c\,n)$-parameter ansätze, used in variational quantum natural gradient. Assuming the expected value of an observable

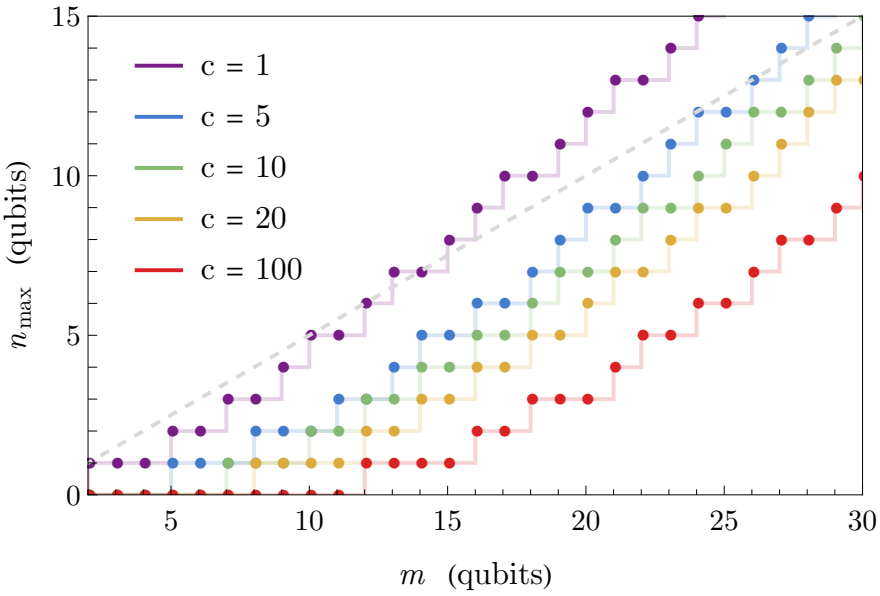


Figure 5.1: The maximum ansatz size ($n_{\max}$ qubits, depth c) for which an iteration of quantum natural gradient is simulatable in a similar time as a single m-qubit c-depth circuit, as per Eq. 5.3. The dashed gray line shows $n_{\max} = m/2$.

can be calculated through a single simulation of a circuit, the runtime to obtain the full geometric tensor $\underline{\underline{G}} : \mathbb{C}^{cn \times cn}$ (and ergo emulate a single iteration of natural gradient) is

$$\Delta t \sim c^3 \, n^3 \, 2^n. \tag{5.2}$$

The maximum n for which we could perform this in $\Delta t \leq \Delta t_{\max}$ is

$$n_{\max} = \left\lfloor \frac{3}{\log(2)} W\left(\frac{\log(2)}{3} \left(\frac{m \, 2^m}{c^2} \right)^{1/3} \right) \right\rfloor, \tag{5.3}$$

where W is the Lambert W function [242]. We visualise this in Figure 5.1 which reveals that for tractable desktop simulation scales, an iteration of quantum natural gradient would take as long as simulating a single circuit of more than *twice* as many qubits. Concretely, the time to simulate a 30-qubit 600-gate circuit (of modest depth $c = 20$) approximates that to simulate a 13-qubit 260-gate ansatz circuit through a

single iteration of quantum natural gradient.

The unexpected expense of simulating variational processes is a consequence of the upper bound of tractable quantum emulation being insufficiently large to outweigh the gate complexity of variational circuits. It is therefore crucial that we seek an asymptotic algorithmic improvement where possible.

5.4 Quantum gradient descent

Quantum gradient descent, as first introduced in Section 1.6.1, seeks the minimum energy $\langle E(\vec{\theta})\rangle = \langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle$ accessible to the parametrized ansatz state $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\text{in}\rangle$ under problem Hamiltonian $\hat{H}$. It does this by iteratively advancing the parameters using

$$\dot{\vec{\theta}} = -\nabla\,\langle E(\vec{\theta})\rangle\,, \qquad \text{(reiteration of 1.56)}$$

where an element of the energy gradient can be experimentally found using the parameter-shift rule [120]

$$\frac{1}{r_i}\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i} = \langle E(\vec{\theta}|_{\theta_i\to\theta_i+\frac{\pi}{4r_i}})\rangle - \langle E(\vec{\theta}|_{\theta_i\to\theta_i-\frac{\pi}{4r_i}})\rangle\,, \qquad \text{(reiteration of 1.59)}$$

but is in any case analytically expressible as

$$\frac{\partial\,\langle E(\vec{\theta})\rangle}{\partial\theta_i} = 2\,\Re\left[\langle\psi(\vec{\theta})|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta})\rangle}{\partial\theta_i}\right]. \qquad \text{(reiteration of 2.24)}$$

As throughout this thesis, assume there are $P = \dim(\vec{\theta})$ parameters in an N-qubit ansatz circuit. We showed in Section 2.6.1 (whereby $\nabla\,\langle E\rangle = 2\,\vec{\mathcal{V}}$) that under the nat-

ural assumption that each ansatz gate (determining constant r_i of Eq. 1.59) features a single unique parameter, the gradient can be classically obtained in $\mathcal{O}(P^2)$ gate simulations, plus the cost of obtaining the non-L^2 vector $\hat{H}\left|\psi(\vec{\theta})\right\rangle$ which we now label $\mathfrak{H}$. That technique also required maintaining P registers at a total memory cost of $\mathcal{O}(P2^N)$. We will now develop a classical algorithm to compute $\nabla\left\langle E\right\rangle(\vec{\theta})$ (or equivalently, $\vec{\mathcal{V}}$) in a total of only $\mathfrak{H}+\mathcal{O}(P)$ gate simulations and $\mathcal{O}(1)$ registers.

Although our derivation is novel, it prescribes a method of equal complexity to an existing technique which leverages automatic differentiation of reversible cost functions [243], used extensively in classical machine learning [244]. Such a technique has already been applied to simulation of quantum variational circuits [245]. However, our own derivation below may be more illustrative to a quantum information theory audience.

5.4.1 Derivation

We will now derive a recurrence relationship between the energy derivatives which will later permit a rolling evaluation of the full gradient vector. We re-adopt our assumption of a separable ansatz, and our previous notation

$$\hat{U}_{m:n}=\prod_{k=n}^{m}\hat{U}_k(\theta_k),\qquad n>m, \tag{reiteration of 2.27}$$

so that $\left|\psi\right\rangle=\hat{U}_{1:P}\left|\text{in}\right\rangle$, and a derivative thereof is compactly expressible as

$$\frac{\partial\left|\psi(\vec{\theta})\right\rangle}{\partial\theta_j}=\hat{U}_{j+1:P}\,\frac{\mathrm{d}\hat{U}_j(\theta_j)}{\mathrm{d}\theta_j}\,\hat{U}_{1:j-1}. \tag{reiteration of 2.28}$$

For laterclarity, will write inner products explicitly as $\text{prod}\big[\,|a\rangle\,,|b\rangle\,\big] = \langle a|b\rangle$. Then

$$\frac{\partial\,\langle E\rangle}{\partial\theta_i} = 2\,\Re\,\text{prod}\big[\hat{U}_{1:P}\;|\text{in}\rangle\,,\;\;\hat{H}\;\hat{U}_{i+1:P}\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\hat{U}_{1:i-1}\;\;|\text{in}\rangle\,\big], \tag{5.4}$$

$$= 2\,\Re\,\text{prod}\big[\hat{U}^\dagger_{i+1:P}\;\hat{H}\;\hat{U}_{1:P}\;|\text{in}\rangle\,,\;\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\hat{U}^\dagger_{i:P}\;\hat{U}_{1:P}\;\;|\text{in}\rangle\,\big], \tag{5.5}$$

$$= 2\,\Re\,\text{prod}\big[\hat{U}^\dagger_{i+1:P}\;\hat{H}\;\;|\psi\rangle\,,\;\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\hat{U}^\dagger_{i:P}\,|\psi\rangle\,\big], \tag{5.6}$$

By defining vectors $|\phi\rangle_i$ and $|\lambda\rangle_i$ (noting the latter will not generally be L^2) which admit recurrent forms,

$$|\phi\rangle_i = \hat{U}^\dagger_{i:P}\,|\psi\rangle \quad\implies |\phi\rangle_i = \hat{U}^\dagger_i\,|\phi\rangle_{i+1}\,, \tag{5.7}$$

$$|\lambda\rangle_i = \hat{U}^\dagger_{i+1:P}\;\hat{H}\;\;|\psi\rangle \quad\implies |\lambda\rangle_i = \hat{\hat{U}}^\dagger_{i+1}\,|\lambda\rangle_{i+1}\,, \tag{5.8}$$

we can make explicit an underlying recurrence in the energy derivatives;

$$\frac{\partial\,\langle E\rangle}{\partial\theta_i} = 2\,\Re\,\text{prod}\big[\,|\lambda\rangle_i\,,\;\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\;|\phi\rangle_i\,\big] \tag{5.9}$$

$$= 2\,\Re\,\text{prod}\big[\hat{U}^\dagger_{i+1}\,|\lambda\rangle_{i+1}\,,\;\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\hat{U}^\dagger_i\,|\phi\rangle_{i+1}\,\big]. \tag{5.10}$$

By iterating in order $i \in \{P,\,P-1,\,\ldots,\,1\}$ and in-turn evaluating $|\phi\rangle_i$ and $|\lambda\rangle_i$ from their previous evaluations (Equations 5.7 and 5.8) in a fixed number of gates, we are able to evaluate every element of the gradient vector (via Equation 5.9) in a total $\mathcal{O}(P)$ gates and inner products. We have avoided applying each gate in the ansatz more than a fixed number of times (precisely, *twice*), and have not had to create more than a fixed number of working registers.

5.4.2 Gate derivatives

Equation 5.9 features the derivative of an ansatz gate. While an experimentalist must decompose this into unitaries (Equation 2.29), our classical simulations have so far avoided this nuisance through, for example, finite-difference approximation of the ansatz state (Equation 2.71). It will prove more convenient now to effect this derivative gate as a non-unitary operator, modifying

$$|\phi\rangle \to \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} |\phi\rangle \,, \tag{5.11}$$

which will be no obstacle for even a modestly-equipped simulator. For the exceptionally common case of a D-qubit Pauli gadget, we know

$$\hat{U}_i(\theta_i) = \exp\left(\mathrm{i}\,\theta_i\,\lambda_i \bigotimes_j^D \hat{\sigma}_{ij}\right) \qquad \Longrightarrow \qquad \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} = \mathrm{i}\,\lambda_i \left(\bigotimes_j^D \hat{\sigma}_{ij}\right) \hat{U}_i(\theta_i),$$

(reiteration of 1.22)

and can hence apply $\hat{U}_i(\theta_i)$ before or after applying each of its D generating Pauli operators in-turn. The state could then be rescaled with coefficient $\mathrm{i}\lambda_i$, though it is more efficient to defer treatment of the coefficient until the ultimate inner product of Equation 5.9. Should a more advanced ansatz gate be naturally expressed through an L-term generating Pauli string as,

$$\hat{U}_i(\theta_i) = \exp\left(\mathrm{i}\,\theta_i \sum_k^L \lambda_{ik} \bigotimes_j^D \hat{\sigma}_{ijk}\right) \quad \Longrightarrow \quad \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} = \mathrm{i} \sum_k^L \lambda_{ik} \left(\bigotimes_j^D \hat{\sigma}_{ijk}\right) \hat{U}_i(\theta_i), \tag{5.12}$$

then each $\{\hat{\sigma}_{ijk} : k\}$ can be simulated in-turn as above at a total factor L to the runtime, and their individual inner products summed;

$$\frac{\partial \langle E \rangle}{\partial \theta_i} = 2 \, \Re \sum_k^L \mathrm{i} \, \lambda_{ik} \, \mathrm{prod} \left[|\lambda\rangle_i , \quad \left(\bigotimes_j^D \hat{\sigma}_{ijk} \right) \hat{U}_i \, |\phi\rangle_i \right]. \tag{5.13}$$

Some common gates admit derivatives which are more naturally expressed in terms of non-unitary constituents. For example, the derivative of the one-qubit phase gate,

$$\frac{\mathrm{d}}{\mathrm{d}\theta_i} \begin{pmatrix} 1 & 0 \\ 0 & e^{\mathrm{i}\,\theta_i} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & \mathrm{i}\, e^{\mathrm{i}\,\theta_i} \end{pmatrix} = \mathrm{i} e^{\mathrm{i}\,\theta_i} \, |1\rangle\langle 1|, \tag{5.14}$$

can be effected by a projection of the target qubit into its $|1\rangle$ state, and the scaling of $\mathrm{i}\, e^{\mathrm{i}\,\theta_i}$ again deferred to an inner product. Projection is likely an existing efficient facility in a simulator as a subroutine used in simulating measurement. Similarly, controlled gates become

$$\frac{\mathrm{d}}{\mathrm{d}\theta_i} \left(|1\rangle \, \langle 1| \otimes \hat{V}_i + |0\rangle \, \langle 0| \otimes \mathbb{1} \right) = |1\rangle \, \langle 1| \otimes \frac{\mathrm{d}\hat{V}_i}{\mathrm{d}\theta_i}. \tag{5.15}$$

Finite difference approximations can be used for all remaining gate forms, such as those specified symbolically or element-wise, e.g.

$$\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} \approx \frac{1}{2\,\Delta\theta} \left(\hat{U}_i(\theta_i + \Delta\theta) - \hat{U}_i(\theta_i - \Delta\theta) \right). \tag{5.16}$$

A high degree approximation should be used since $\hat{U}_i$ is expected small and cheap to evaluate for different θ_i.

Algorithm 5.1: Efficient classical evaluation of the energy gradient at variational state $\hat{U}(\vec{\theta})\,|\text{in}\rangle$ under N-qubit Hamiltonian $\hat{H}$. Let denote g the cost of effecting a single fixed-size unitary gate upon a statevector.

Input : $|\lambda\rangle$, $|\phi\rangle$, $|\mu\rangle$: Temporary N-qubit statevectors which should persist between invocations of the algorithm, and which may be safely modified in-between.
Input : $|\text{in}\rangle$: An immutable N-qubit statevector, initialised as the input state to the ansatz circuit.
Input : $\hat{U}_{1:P}$: An iterable representation of the ansatz circuit, with a single unique parameter θ_i in each gate $\hat{U}_i$.
Input : $\hat{H}$: An N-qubit Hamiltonian which can be effected upon a state in time $\mathfrak{H}$.
Output: $\nabla\langle E\rangle$: the energy gradient vector specified element-wise.

$|\lambda\rangle \coloneqq |\text{in}\rangle$ // clone state in g
$|\lambda\rangle \leftarrow \hat{U}_{1:P}\,|\lambda\rangle$ // apply P gates in Pg
$|\phi\rangle \coloneqq |\lambda\rangle$ // clone state in g
$|\lambda\rangle \leftarrow \hat{H}\,|\lambda\rangle$ // apply $\hat{H}$ in $\mathfrak{H}$
for $i \in \{P, \ldots, 1\}$ **do**
    $|\phi\rangle \leftarrow \hat{U}_i^\dagger\,|\phi\rangle$ // apply gate in g
    $|\mu\rangle \coloneqq |\phi\rangle$ // clone state in g
    $|\mu\rangle \leftarrow (\mathrm{d}\hat{U}_i/\mathrm{d}\theta_i)\,|\mu\rangle$ // apply non-unitary in g, as per Sec. 5.4.2
    $\nabla\langle E\rangle_i = 2\,\Re\,\langle\lambda|\mu\rangle$ // compute inner product in g
    if $i > 1$ **then**
        $|\lambda\rangle \leftarrow U_i^\dagger\,|\lambda\rangle$ // apply gate in g
    end
end

5.4.3 Algorithm

We formally present our strategy in Algorithm 5.1. It involves a total of $\mathcal{O}(P)$ gate (and derivative) simulations and inner product calculations. The latter is typically cheaper than a single gate. It also requires obtaining the vector $\hat{H}\,|\lambda\rangle \equiv \hat{H}\,|\psi(\vec{\theta})\rangle$ at an assumed cost $\mathfrak{H}$, which depends on the specific Hamiltonian representation. For example, when $\hat{H}$ is a T-term N-qubit Pauli Hamiltonian, the naive cost is $\mathcal{O}(T\,N\,2^N)$. Despite such Hamiltonians having in principle as many as $T = 4^N$ terms, typical physical Hamiltonians of interest grow as $T = \mathcal{O}(N^4)$ [118]. We finally note that the cost of evaluating $\hat{H}\,|\lambda\rangle$ is likely already being paid in each iteration of simulation, in order to compute

the expected energy of the current parameter assignment. Ergo, Line 4 of Algorithm 5.1 may be superfluous, and our total method cost is $\mathcal{O}(P)$ gate simulations.

Our algorithm is compatible with any parallelisation and distribution techniques to accelerate the simulation of the operators and inner products. Our $\mathcal{O}(P)$ speedup is thus, in a sense, *free*. We also required only a fixed (in terms of the number of parameters) memory overhead; three N-qubit statevectors which can persist between invocations of the algorithm (and hence, over the full course of gradient descent minimisation).

5.5 Quantum natural gradient

Section 1.6.3 introduced the quantum natural gradient as a minimisation technique superior to quantum gradient descent. It evolves the parameters via

$$G_{ij} = \frac{\partial \langle\psi|}{\partial\theta_i}\frac{\partial |\psi\rangle}{\partial\theta_j} - \frac{\partial \langle\psi|}{\partial\theta_i}|\psi\rangle\langle\psi|\frac{\partial |\psi\rangle}{\partial\theta_j}, \qquad \Re[\underline{\underline{G}}]\,\dot{\vec{\theta}} = -\nabla\langle E\rangle$$

(re-expression of 1.66 and 1.65)

where $\langle E\rangle = \langle\psi|\hat{H}|\psi\rangle$ is the energy of the P-parameter ansatz state $|\psi(\vec{\theta})\rangle = \hat{U}(\vec{\theta})\,|\text{in}\rangle$ under some problem Hamiltonian $\hat{H}$. We showed in Section 2.4.5 that this is equivalent to our prior developed variational imaginary time technique, when the latter includes an explicit global phase gate. Note that we can evaluate the full complex matrix $\underline{\underline{G}}$, rather than just its real component, in the same time, and will hence do so for notational simplicity.

In this section, we derive a novel algorithm to classically evaluate $\underline{\underline{G}}$ in a factor $\mathcal{O}(P)$ faster than the state-of-the-art seen in the literature, again using only a fixed memory overhead.

5.5.1 Derivation

For clarity, we illustrate recurrent forms of the individual terms of the tensor $\underline{\underline{G}}$. We will begin with the Berry connection [173], which we label as

$$T_i(\vec{\theta}) = \langle \psi(\vec{\theta}) | \frac{\partial |\psi(\vec{\theta})\rangle}{\partial \theta_i}. \tag{5.17}$$

Note subscripts below (or vectors above) the symbol T (and the soon introduced L) distinguish them from the previously reserved symbols T and L, as related respectively to the number of Pauli tensors in the Hamiltonian, and in a gate's Hermitian generator. As before, we assume the ansatz circuit is separable into parameter-unique gates $\hat{U} = \hat{U}_{1:P}$, adopting notation

$$\hat{U}_{m:n} = \prod_{k=n}^{m} \hat{U}_k(\theta_k), \qquad n > m; \tag{reiteration of 2.27}$$

Recall $P = \dim(\vec{\theta})$. Then we have

$$T_i(\vec{\theta}) = \langle \text{in} | \ \hat{U}_{1:P}^\dagger \ \hat{U}_{i+1:P} \ \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} \ \hat{U}_{1:i-1} \ | \text{in} \rangle \tag{5.18}$$

$$= \langle \text{in} | \ \hat{U}_{1:i}^\dagger \ \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} \ \hat{U}_{1:i-1} \ | \text{in} \rangle \tag{5.19}$$

$$= \langle \psi_i | \ \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} \ | \psi_{i-1} \rangle \, , \tag{5.20}$$

where we have defined intermediate states produced by prefixes of the ansastz circuit as

$$|\psi_i\rangle = \hat{U}_{1:i} |\text{in}\rangle \qquad \Longrightarrow \qquad |\psi_{i+1}\rangle = \hat{U}_{i+1} |\psi_i\rangle \, . \tag{5.21}$$

Equation 5.20 suggests a simple iterative strategy to calculate T_i for all $i \in \{1, \ldots, P\}$, in a total of $\mathcal{O}(P)$ gates, equivalent to our method of calculating the energy gradient in Section 5.4 under $\hat{H} \to \mathbb{1}$. However, we need not do this directly, and can instead integrate its calculation into the more sophisticated evaluation of the first term of $\underline{\underline{G}}$. We label this term

$$L_{ij}(\vec{\theta}) = \frac{\partial \langle\psi(\vec{\theta})|}{\partial\theta_i} \frac{\partial |\psi(\vec{\theta})\rangle}{\partial\theta_j}, \tag{5.22}$$

noting that its real component $\underline{\underline{\mathcal{M}}} = \Re[\underline{\underline{L}}]$ happens to be that used in imaginary time evolution of Chapter 2. Through similar working, we can show

$$(i < j) \qquad L_{ij} = \langle \text{in}| \ \hat{U}^\dagger_{1:i-1} \ \frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i} \ \hat{U}^\dagger_{i+1:P} \ \hat{U}_{j+1:P} \ \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_i} \ \hat{U}_{1:j-1} \ |\text{in}\rangle \tag{5.23}$$

$$= \langle\psi_{i-1}| \ \frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i} \ \hat{U}^\dagger_{i+1:j} \ \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_i} \ |\psi_{j-1}\rangle \,, \tag{5.24}$$

noting that $\underline{\underline{L}}$ is Hermitian. Let us define a matrix of vectors, each not necessarily normalised, as

$$(i < j) \qquad |\phi_{ij}\rangle = \hat{U}^\dagger_{i:j} \ \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j} \ |\psi_{j-1}\rangle \,, \tag{5.25}$$

so that we may express

$$(i < j) \qquad L_{ij} = \langle\psi_{i-1}| \ \frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i} \ |\phi_{i+1,j}\rangle \,. \tag{5.26}$$

By observing the recurrent forms

$$\langle\psi_{i-1}| = \langle\psi_i| \, \hat{U}_i, \qquad \text{and} \qquad \begin{aligned} |\phi_{i-1,j}\rangle &= \hat{U}^\dagger_{i-1} \, |\phi_{i,j}\rangle \,, \\ |\phi_{j+1,j+1}\rangle &= \frac{\mathrm{d}\hat{U}_{j+1}}{\mathrm{d}\theta_{j+1}} \, |\psi_j\rangle \,, \end{aligned} \tag{5.27}$$

and for later clarity, notating $\text{prod}\big[\,|a\rangle\,,|b\rangle\,\big] = \langle a|b\rangle$, we make explicit a recurrency in $\underline{\underline{L}}$;

$$L_{ij} = \text{prod}\left[\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;|\psi_{i-1}\rangle\,,\;\;|\phi_{i+1,j}\rangle\right], \tag{5.28}$$

$$\therefore\;\; L_{i-1,j} = \text{prod}\left[\frac{\mathrm{d}\hat{U}_{i-1}}{\mathrm{d}\theta_{i-1}}\,\hat{U}_{i-1}^\dagger\;|\psi_{i-1}\rangle\,,\;\;\hat{U}_i^\dagger\;|\phi_{i+1,j}\rangle\right]. \tag{5.29}$$

This form reveals that given $|\psi_j\rangle$, we can produce $|\phi_{j+1,j+1}\rangle$ in one (derivative) gate operation, and from that iteratively produce $|\phi_{i,j}\rangle$ for every $i < j$, one gate operation at a time (similarly for $|\psi_i\rangle$), without any caching or creating additional memory. This means all L_{ij} values, for $i < j$, can be computed in a total $\mathcal{O}(j)$ gate operations and $\mathcal{O}(1)$ space, in decreasing i order. Since $|\psi\rangle_{j+1}$ is furthermore obtained from $|\psi_j\rangle$ by a single gate operation, all L_{ij}, for $i < j \leq P$, may be calculated in $\mathcal{O}(P^2)$ gates.

With $L_{ji} = {L_{ij}}^*$, this leaves the diagonal elements of the form

$$L_{ii} = \langle \text{in}|\;\hat{U}_{1:i-1}^\dagger\;\frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i}\;\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\;\hat{U}_{1:i-1}\;|\text{in}\rangle \tag{5.30}$$

$$= \langle\psi_{i-1}|\,\frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i}\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}\,|\psi_{i-1}\rangle \tag{5.31}$$

$$= \langle\phi_{ii}|\phi_{ii}\rangle\,, \tag{5.32}$$

where although $\hat{U}^\dagger\hat{U} = \mathbb{1}$, in general $\frac{\mathrm{d}\hat{U}}{\mathrm{d}\theta_i}$ is non-unitary, and hence $\langle\phi_{ii}|\phi_{ii}\rangle$ is not necessarily unity. This term can be simply computed through an invocation of $\text{prod}\big[\,|\phi_{ii}\rangle\,,|\phi_{ii}\rangle\,\big]$ like the other terms. However, common Pauli gadgets of the form $\hat{U}_i(\theta_i) = \exp(\mathrm{i}\,\lambda_i(\theta_i)\,\vec{\sigma}_i^{\otimes})$ (where $\vec{\sigma}_i^{\otimes}$ is some tensor product of Pauli operators) admit the trivial $L_{ii} = \lambda_i'(\theta_i)^2$ which can be determined *a priori* to avoid P total inner products during simulation.

We point out that *controlled* Pauli gadgets lose this property,

$$\hat{U}_i(\theta_i) = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \exp(\mathrm{i}\,\lambda_i(\theta_i)\,\vec{\sigma}_i^{\otimes}) \tag{5.33}$$

$$\implies \frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i}\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} = \lambda_i'(\theta_i)^2\ |1\rangle\langle 1| \otimes \mathbb{1}, \tag{5.34}$$

$$\implies L_{ii} = \lambda_i'(\theta_i)^2\ |\,\langle\psi_{i-1}|1\rangle\,|^2, \tag{5.35}$$

instead involving the probability of the control qubit being in state 1. Having evaluated every L_{ij} and T_i by the above recurrences, the full tensor of the natural gradient is then simply constructed as

$$G_{ij}(\vec{\theta}) = L_{ij}(\vec{\theta}) - T_i^*(\vec{\theta})\,T_j(\vec{\theta}). \tag{5.36}$$

5.5.2 Algorithm

Our classical algorithm to compute $\underline{\underline{G}}$ is to simply iteratively evaluate the recurrences of Equations 5.20 and 5.28 to determine $\underline{\underline{L}}$ and $\vec{T}$, before finally constructing $\underline{\underline{G}}$ through Equation 5.36. We formalise this in Algorithm 5.2, which invokes the same gate derivatives shown in Section 5.4.2.

Once again, a generic simulator is expected to support all primitive operations therein. Each can make use of the parallelisation and high-performance computing techniques which we will introduce in Chapter 6. Requiring only a fixed memory overhead, our scheme enables a factor $\mathcal{O}(P)$ faster simulation of precise quantum natural gradient (and ergo, variational imaginary time minimisation) than the methods used in the present day literature.

Algorithm 5.2: Calculating the complete Fisher information matrix G_{ij} in a total $\mathcal{O}(P^2)$ gates / clone operations, and $\mathcal{O}(1)$ temporary statevectors. Here, $|a\rangle \coloneqq |b\rangle$ denotes cloning state $|b\rangle$ into $|a\rangle$, while $|a\rangle \leftarrow \hat{U}$ denotes modifying $|a\rangle$ under the action of operator $\hat{U}$. Comments on the right-hand-side indicate the state of the modified register after an operation.

Input : Initial state, which is input to the ansatz, $|\text{in}\rangle$
Input : Temporary statevectors $|\chi\rangle$, $|\psi\rangle$, $|\phi\rangle$, $|\lambda\rangle$, $|\mu\rangle$
Input : Ansatz circuit $\prod_i \hat{U}_i(\theta_i)$
Output: Fisher information matrix $G_{ij} \,\forall\, 1 \leq i, j \leq P$
Output: $|\psi\rangle$ is left in ansatz state $\hat{U}_P \dots \hat{U}_1 |\text{in}\rangle$

// Handle edge-cases
$|\chi\rangle \coloneqq |\text{in}\rangle$
$|\chi\rangle \leftarrow \hat{U}_1$ // $|\chi\rangle = \hat{U}_1 |\text{in}\rangle$, permanently
$|\psi\rangle \coloneqq |\chi\rangle$
$|\phi\rangle \coloneqq |\text{in}\rangle$
$|\phi\rangle \leftarrow \frac{\mathrm{d}\hat{U}_1}{\mathrm{d}\theta_1}$
$T_1 = \langle\chi|\phi\rangle$
$L_{1,1} = \langle\phi|\phi\rangle$
// Compute $L_{i\leq j}$ (Eq. 5.28) and T_k (Eq. 5.20)
for $j \in \{2, \dots, P\}$ **do**
    $|\lambda\rangle \coloneqq |\psi\rangle$ // $|\lambda\rangle = \hat{U}_{j-1} \dots \hat{U}_1 |\text{in}\rangle$
    $|\phi\rangle \coloneqq |\psi\rangle$
    $|\phi\rangle \leftarrow \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j}$ // $|\phi\rangle = \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j} \hat{U}_{j-1} \dots \hat{U}_1 |\text{in}\rangle$
    $L_{j,j} = \langle\phi|\phi\rangle$ // Skippable if $\hat{U}_j = \exp(\alpha\hat{\sigma})$
    for $i \in \{j-1, \dots, 1\}$ **do**
        $|\phi\rangle \leftarrow \hat{U}_{i+1}^\dagger$ // $|\phi\rangle = \hat{U}_{i+1}^\dagger \dots \hat{U}_j^\dagger \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j} \hat{U}_{j-1} \dots \hat{U}_1 |\text{in}\rangle$
        $|\lambda\rangle \leftarrow \hat{U}_i^\dagger$ // $|\lambda\rangle = \hat{U}_{i-1} \dots \hat{U}_1 |\text{in}\rangle$
        $|\mu\rangle \coloneqq |\lambda\rangle$
        $|\mu\rangle \leftarrow \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i}$ // $|\mu\rangle = \frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_i} \hat{U}_{i-1} \dots \hat{U}_1 |\text{in}\rangle$
        $L_{ij} = \langle\mu|\phi\rangle$ /* $\langle\mu|\phi\rangle = \langle\text{in}| \hat{U}_1^\dagger \dots \hat{U}_{i-1}^\dagger \frac{\mathrm{d}\hat{U}_i^\dagger}{\mathrm{d}\theta_i} \cdot \hat{U}_{i+1}^\dagger \dots \hat{U}_j^\dagger \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j} \hat{U}_{j-1} \dots \hat{U}_1 |\text{in}\rangle$ */
    end
    $T_j = \langle\chi|\phi\rangle$ /* $\langle\chi|\phi\rangle = \langle\text{in}| \hat{U}_1^\dagger \cdot \hat{U}_2^\dagger \dots \hat{U}_j^\dagger \frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_j} \hat{U}_{j-1} \dots \hat{U}_1 |\text{in}\rangle$ */
    $|\psi\rangle \leftarrow \hat{U}_j$ // $|\psi\rangle = \hat{U}_j \dots \hat{U}_1 |\text{in}\rangle$
end
// Unpack T and L values into the Hermitian quantum tensor G
for $i \in \{1, \dots P\}$ **do**
    for $j \in \{1, \dots P\}$ **do**
        if $i \leq j$ **then**
            $G_{ij} = L_{ij} - T_i^* T_j$
        else
            $G_{ij} = L_{ji}^* - T_i^* T_j$
        end
    end
end

5.6 Extensions

Both algorithms made several simplifying assumptions about the ansatz circuit and Hamiltonian which can be relaxed without asymptotic penalty. For instance:

- **Multi-parameter gates** like $\hat{U}_i(\theta_a, \theta_b)$ are trivial to support, since the preconditions to evaluate the variational terms with respect to θ_a and θ_b are the same. Ergo, lines 7- 9 of Alg. 5.1 are merely looped for each θ_j in $\hat{U}_i$ (analogously for Alg. 5.2).

- **Repeated parameters** like θ_a featured in ansatz $\hat{U}_i(\theta_a)\,\hat{U}_j(\theta_b)\,\ldots\,\hat{U}_k(\theta_a)$. One can show by the chain rule that Equ. 5.9 would become

$$\frac{\partial\,\langle E\rangle}{\partial\theta_a} \;=\; 2\,\Re\,\mathrm{prod}\big[\,|\lambda\rangle_i\,,\,\frac{\mathrm{d}\hat{U}_i}{\mathrm{d}\theta_a}\,|\phi\rangle_i\,\big] \;+\; 2\,\Re\,\mathrm{prod}\big[\,|\lambda\rangle_j\,,\,\frac{\mathrm{d}\hat{U}_j}{\mathrm{d}\theta_a}\,|\phi\rangle_j\,\big], \tag{5.37}$$

  suggesting an independent treatment of the gates, and only later combining their contributions into a gradient element. The complexity is now $\mathcal{O}(Q)$ where $Q > P$ is the number of *gates* in the ansatz (previously assumed P).

- **Non-unitary (but invertible) gates** like $\hat{U}_i$ with $\underline{\underline{U}} = \begin{pmatrix} 1 & 2\mathrm{i} \\ 3 & 4\mathrm{i} \end{pmatrix}$. Unitarity was leveraged in order to "undo" gate $\hat{U}_i$ by effecting $\hat{U}_i^\dagger \equiv \hat{U}_i^{-1}$, so Line 6 of Alg. 5.1 would merely become $|\lambda\rangle \leftarrow \hat{U}_i^{-1}\,|\lambda\rangle$. Note that Line 11 features $\hat{U}_i^\dagger$ to remove $\hat{U}_i$ from the next iteration's inner-product of Line 9 through $(\hat{U}_i^\dagger)^\dagger = \mathbb{1}$. As such, it need not be changed.

- **Non-Hermitian cost functions**, like $\hat{H} \neq \hat{H}^\dagger$. In this case, the quantity $\langle E\rangle$ is

now complex, and Equ. 2.24 is no longer satisfied. Instead, we leverage

$$\frac{\partial \langle E \rangle}{\partial \theta_i} \neq 2\,\Re\left[\langle\psi(\vec{\theta})|\,\hat{H}\frac{\partial\,|\psi(\vec{\theta})\rangle}{\partial\theta_i}\right], \tag{5.38}$$

$$\frac{\partial \langle E \rangle}{\partial \theta_i} = \langle \text{in}|\,\frac{\partial \hat{U}^\dagger(\vec{\theta})}{\partial\theta_i}\hat{H}\hat{U}(\vec{\theta})\,|\text{in}\rangle + \langle \text{in}|\,\hat{U}^\dagger(\vec{\theta})\hat{H}\frac{\partial \hat{U}(\vec{\theta})}{\partial\theta_i}\,|\text{in}\rangle \tag{5.39}$$

$$= \langle \text{in}|\,\frac{\partial \hat{U}^\dagger(\vec{\theta})}{\partial\theta_i}\hat{H}\hat{U}(\vec{\theta})\,|\text{in}\rangle + \langle \text{in}|\,\frac{\partial \hat{U}^\dagger(\vec{\theta})}{\partial\theta_i}\hat{H}^\dagger\hat{U}(\vec{\theta})\,|\text{in}\rangle^*, \tag{5.40}$$

performing our algorithms *twice*: once with cost-operator $\hat{H}$ and again for $\hat{H}^\dagger$, being sure to retain the *full* inner-product scalars, rather than just their real components. We finally sum their output gradients.

5.7 Discussion

This chapter derived asymptotically improved algorithms for classically simulating quantum gradient descent and quantum natural gradient. They offer an $\mathcal{O}(P)$ speedup for P-parameter ansätze without a scaling memory penalty. This will enable the research community to simulate deeper ansatz circuits and study previously intractable systems. For example, Reference [246] which simulated natural gradient with a modest 40 parameters could leverage our technique to simulate ≈ 250 parameters in the same time. Both algorithms have since been incorporated into the IBM Qiskit simulator [239], and the Oxford QuESTlink simulator [112] of the proceeding chapter.

It is natural to hope that an extension of these techniques would enable accelerated simulation of *noisy* variational algorithms. In Appendix C of Reference [247], the author erroneously laments that since the analytic statevector expressions leveraged by Algorithms 5.1 and 5.2 do not appear in the analogous expressions with precise den-

sity matrices, an extension of these algorithms for noisy parameterised channels is not possible. Much too hasty! Both algorithms have since been adapted by the author for asymptotically faster simulation using *density matrices* to handle ansätze specified as noisy but trace-preserving parameterised channels. In essence, these adaptations leverage that; the trace of a density matrix represented in the Choi–Jamiolkowski formalism [248, 249] (introduced in the next chapter) resembles a state-vector inner-product; a Pauli Hamiltonian may be efficiently transformed into a dense matrix and subsequently into an Choi-vector representation of an unnormalised density matrix; the quantum Fisher information matrix is well approximated by the Hilbert–Schmidt metric tensor in typical experimental settings [227]. Alas, like the margins of *Arithmetica*, there is no room here to derive these methods explicitly.

Chapter 6

Classical High-Performance Simulation

Contents

6.1 Foreword

This section is relevant to the published work

- **T. Jones**, A. Brown, I. Bush, S. C. Benjamin
  QuEST and High Performance Simulation of Quantum Computers
  Scientific Reports **9**, 10736 (2019) (link)

and the manuscript currently under preparation

- **T. Jones**, C. Beaudoin, B. Koczor, S. C. Benjamin
  Distributed Algorithms for Simulating Quantum Computers

My contribution to the first manuscript is elaborated upon in Chapter 7, and included writing the text and performing benchmarking of an early prototype of the QuEST simulator, then developed primarily by my co-author Ania Brown. The second developing manuscript is a collection of twenty novel asymptotically-superior distributed algorithms for simulating quantum operations, fourteen of which I developed, and the remaining proposed by my coauthors. At present, I have authored all text of twenty five thousand words.

This chapter is scoped well beyond these manuscripts however, and describes significant classical computational efforts throughout my doctorate. Without the space to be comprehensive, this chapter seeks to introduce the core concepts of high-performance classical simulation of quantum computers. It will provide only examples of my original research, which has manifested in the software of the proceeding chapter.

6.2 Introduction

We hope to have so far convinced the reader that classical simulation of quantum computers is a vital and irreplaceable step in the development of quantum algorithms. Chapter 5 developed asymptotically improved methods of simulating *variational* routines by assuming existing facilities for simulation of individual gates upon a statevector. This chapter addresses that assumption. Classical emulation of digital quantum systems is known to grow exponentially in either time or memory (or both) with increasing qubits [33]. Simulators of quantum computers must therefore utilise the many facilities available in high performance classical computing (HPC).

This chapter will become somewhat multidisciplinary in nature, borrowing concepts from both classical and quantum information processing. To remain palatable, sections will have the following structure; we will review the HPC considerations of a particular classical computing platform, then we will derive a ubiquitous algorithm for quantum statevector simulation whose implementation demonstrates the aforementioned considerations. In all, we will discuss serial single-CPU, multithreaded multi-CPU, GPU-accelerated and network distributed simulation of statevectors and density matrices, and example simulations of single-target, controlled, many-target, SWAP and Pauli gadget gates, dephasing, depolarising and damping decoherence channels, and Pauli Hamiltonian expectation values. These facilities form a small subset of the full HPC efforts conducted throughout the research of this thesis, of which we will give a better indication in Chapter 7.

6.2.1 Model

There are many practical models and paradigms of simulation used by the research community. These include, but are not limited to, tensor networks [250, 251] well suited for many-qubit short-depth circuits, matrix product state (MPS) [252] for weakly entangling circuits, bespoke simulators of (limited gate-type) stabilizer circuits [253, 254, 255], and decision diagrams [256, 257, 258, 259] for noise-free logical circuits.

Perhaps the conceptually simplest paradigm is full-state simulation [260, 261], also known as brute-force and Schrödinger-style, which maintains a dense representation of a quantum state as a vector or matrix of precisely-known complex amplitudes. An N-qubit pure state is encoded as a vector of floating-point complex numbers

$$|\psi\rangle = \sum_{i}^{2^N} \alpha_i |i\rangle \quad \rightarrow \quad \vec{\psi} = \{\alpha_0, \ \ldots, \ \alpha_{2^N-1}\}, \qquad \alpha_i \in \mathbb{C}. \tag{6.1}$$

There is some suggestion that polar complex forms are worthwhile [262], although this chapter will mostly abstract away such nuance. The array $\vec{\psi}$ can in principle represent unnormalised non-physical states whereby

$$\sum_{i}^{2^N} |\alpha_i|^2 \neq 1, \tag{6.2}$$

which we saw useful for bespoke variational simulation in Chapter 5, and which will prove crucial for a clever representation of density matrices through the Choi–Jamiolkowski isomorphism [248, 249], presented later in this chapter. The amplitudes α_i are queryable and modifiable at any stage during simulation, hence full-state simulators are instances of "strong simulators", known to admit precision $1/2^{\epsilon+1}$ in $2^{\epsilon-\mathcal{O}(\epsilon)}$ time [260]. The time and memory costs of a full-state simulation scale as $\mathcal{O}(2^N)$, though time scales linearly

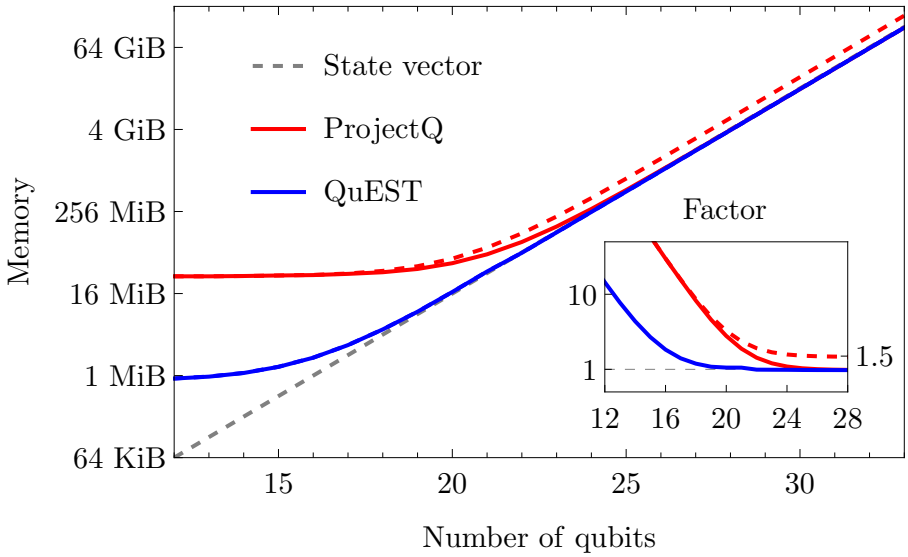


Figure 6.1: The memory costs of simulating quantum registers (using double precision floating-point Cartesian complex numbers) in full-state simulator implementations QuEST [249] and ProjectQ [263]. Deviation from the dashed line shows language and runtime specific program overheads.

with the number of operations, which typically incur no memory penalties. Full-state simulators are therefore well suited for simulation of deep circuits with limited qubits $(1-30)$, though they can very effectively leverage parallelisation and distribution techniques to remain useful for larger simulations $(30-45)$. We helpfully gauge the typical cost of statevectors representation in Figure 6.1.

Because of their algorithmic simplicity, their complete description of the quantum state, and their excellent utilisation of classical hardware, full-state simulators are a safe first choice when simulating a generic quantum circuit. In this thesis, we restrict ourselves to full-state simulation of closed quantum systems evolving under discrete digital operations.

6.2.2 Notation

We summarise the notation used in this chapter. Assume the first indexed qubit (with index 0) is the right-most and least significant qubit, which is the convention already adopted in this thesis. Thus

$$\textit{(single qubit kets)} \qquad |0\rangle \otimes |1\rangle \otimes |1\rangle \quad \equiv \quad |2^1 + 2^0\rangle = |3\rangle \qquad \textit{(many qubit kets)}$$

where the first qubit (with index 0) is in single-qubit state $|1\rangle$.

Our algorithm pseudocode will make use of symbols:

$\%$	integer modulo
$\ll$	bitwise left-shift
$\gg$	bitwise right-shit
$\mid$	bitwise `OR`
$\oplus$	bitwise `XOR`
$\&$	bitwise `AND`
$\times$	scalar multiplicaition (often implicit)
$\neg$	bit flip and logical negation
conj	complex conjugation
^{T}	matrix transposition
$\{\,\}$	empty list
$\{1, \ldots, n)$	list of integers up to and excluding n
$i_{[n]}$	n-th bit of unsigned binary integer i
$\vec{\psi}[i]$ or ψ_i	i-th element or amplitude of state $\|\psi\rangle$

6.3 Serial

Incorporating HPC techniques into scientific simulation often means taking an existing implementation of an algorithm, and rewriting the code into an asymptotically equivalent (and unfortunately often, less readable) form which somehow runs significantly faster. To the unfamiliar eye, these changes - which take advantage of low-level hardware features and implementation nuances - and their immense performance benefits can look like magic. To understand them, we must first understand the modern computer architecture and compiler stack. This section provides a rapid introduction to the computing facilities essential for writing high performance code on single CPU systems, as relevant to statevector simulation of quantum computers.

6.3.1 Facilities

Code written by a programmer in a (relatively) high-level programming language like `C` is processed by a *compiler* and translated into native instructions for the computer's central processing unit (CPU), called *machine code*. This process includes *optimisation*, whereby the compiler may notice and exploit meta-properties of the input code to produce faster machine code. For example, it may notice a repeatedly called function is small and worth *inlining*, whereby its code definition is duplicated in the output machine code to reduce invocation overheads [264]. Or, the compiler may notice that a loop in the code is modifying contiguous elements of an array in an agnostic fashion, and automatically *vectorise* them (we will elaborate upon this later) [265]. When machine code is executed, the operating system dispatches the instructions therein in-turn to the CPU. This becomes a ballet of information flow.

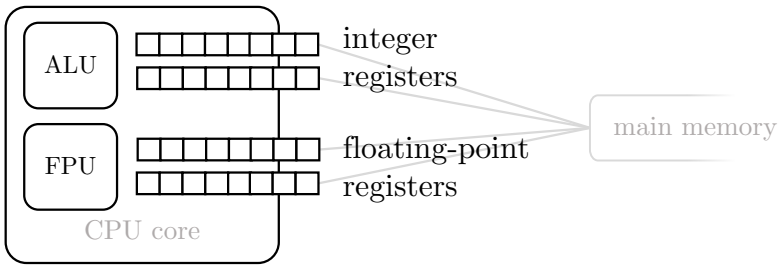


Figure 6.2: An illustration of a single CPU core which receives binary data from main memory (such as RAM) into its registers, processes the data within sub-processing units via digital logical gates, and returns the output to the registers. Since bitwise and arithmetic processing is much faster than floating-point, modern CPUs contain separate ALU and FPU units and registers.

During an instruction, the data processed by the machine code (such as variable instances) are loaded into the *registers* of the CPU (or of one of its potentially many *cores*). So too is the instruction. Computation upon the values in these registers is performed by processing units within the core, performing digital logical gates upon the binary data. Indeed, these are the classical counterpart to the quantum computational gates seen throughout this thesis. The output of the classical gates is returned to the registers. Generally the number of gates to perform a *floating-point operation* (a *flop*, such as the division of two decimal numbers) significantly exceeds the number to perform an integer operation (such as the addition of two natural numbers), or a bitwise operation (such as a bit-shift). For the sake of speed, modern CPU cores will ergo contain separate registers to store integers and floating-point numbers [266], respectively feeding them to separate arithmetic logical units (ALU) and floating-point units (FPU). We visualise this in Figure 6.2. Furthermore, bitwise operations within the ALU (like logic operations, bit shifts and increments) are typically significantly faster than general arithmetic operations (like multiplication of large integers). High performance code is ergo *type aware*, and strives to minimise the superfluous use of floating-point operations while maximising the substitution of integer arithmetic with bitwise operations.

Modern CPU registers are typically larger than necessary to contain a single primitive like an integer. In fact, the author writes this very thesis on a machine with a dual-core Intel i7 CPU, containing 128-bit vector registers [267], each of which can simultaneously store up to *four* 4-byte integer primitives. High performance code will make use of *vectorisation* [265], whereby the CPU processing units will perform the same operation upon all of the primitives stored within a register simultaneously. This is an instance of single-instruction multiple-data (SIMD) processing. We note that some architectures involve a separate vector processing unit (VLU) for such facilities. While vectorisation can be explicitly invoked by the user code through advanced vector extension (AVX) instructions [268], a modern compiler can often automatically detect scenarios in the user code amenable to vectorisation [269]. These include loops with explicit bounds which modify contiguous array elements in a simple, predictable manner without *branching* or *control flow* (which we introduce below) [270]. High performance code will hence make use of one loop paradigm over an asymptotically equivalent alternative in order to give the compiler the best chance of auto-vectorisation.

We have so far described how the CPU performs an instruction - but efficiently executing instructions is only half the battle, especially in data-dominated applications. Even *loading* data into the registers can be a complicated matter. A modern classical computer features a hierarchy of progressively smaller and lower-latency memory banks between the "main" random access memory (RAM) storing all of a running program's data, and the CPU registers which ultimately receive this data. These intermediate banks are called *caches* [271], which we visualise in Figure 6.3.

Consider the scenario where a CPU core's next task is to operate upon some primitive in memory which must hence be loaded into a register. In the ideal case, this primitive is stored in the core's level one data cache (L1d), and can be loaded immediately; this

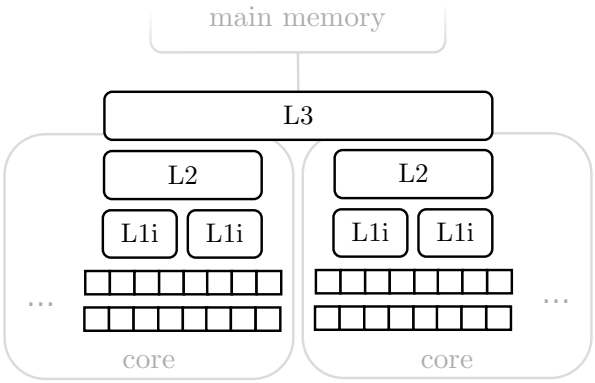


Figure 6.3: An illustration of the cache hierarchy of a dual-core CPU. Data, which includes both program runtime variables and the program machine code instructions themselves, cascade from main memory (like RAM) through the progressively smaller-capacity but smaller-latency cache, to the registers of the CPU cores.

is a *cache hit*. In the worst case, the primitive resides only in RAM; a *cache miss*. In that case, it must then be cumbersomely transferred into the CPU-wide L3 cache, then into the core-specific L2 cache, then into the L1d cache, and finally into the core's register. In lieu of propogating single primitives, contiguous regions of cache memory called *cache-lines* are overwritten at once. This allows, for example, adjacent elements of an array to be simultaneously loaded between caches, and ergo that a looped array-access will maximise its cache hits. Ensuring that the caches are effectively pre-fetching array elements in this way, by iterating elements in-order (*striding*), is crucial for high performance processing of large datasets. When a primitive is modified, this change must be propagated back through the caches to main memory, to maintain *coherence* between the cache and the true program state. We will see this is an important nuance in multithreaded settings.

All operations in a classical computer are discretised in time, through *clock cycles*, and their ultimate costs are hardware dependent. For illustration, the latencies of the L1, L2 and L3 caches of the Intel i7-4770 (3.4GHz) are 4, 12, and 36 cycles re-

spectively [267]. During a cycle, some independent CPU components may operate simultaneously; caches may be loaded while processing units perform gates, or the next machine code instruction may be pre-fetched. When encountering a conditional (`if`) statement, the next instruction to load is not yet determined. In lieu of waiting, a CPU may perform *branch prediction* [272], predicting the condition outcome and pre-fetching the determined instruction so that, if correct, it can be evaluated without delay. If the prediction is wrong however, the CPU must undo its state changes and fetch the correct instruction with some *misprediction penalty* (15-20 clock cycles on the i7-4770 [267]). Failed branch predictions can severely undermine performance, so high-performance code avoids superfluous conditional statements inside large performance-critical loops.

6.3.2 Simulation

We have so far mentioned every major performance-related implementation consideration relevant to the serial (i.e. non-parallel) local (i.e. non-distributed) simulation of quantum computers as performed through the development of this thesis. To illustrate these considerations, we now incorporate them into some example simulation algorithms. We visit the single-qubit general unitary to demonstrate type-awareness and striding, and the single-controlled unitary to demonstrate efficient caching, branching and vectorisation.

Single-qubit gate

Let us first derive a simple algorithm to simulate a single qubit gate $\hat{U}_t$ (with general elements $u_{rc} \in \mathbb{C}$) upon qubit t of an N-qubit statevector $|\psi\rangle$. For notational clarity,

we temporarily index from 1 such that $1 \leq t \leq N$. The action of $\hat{U}_t$ upon $|\psi\rangle$ is

$$\hat{U}_t |\psi\rangle = \mathbb{1}^{\otimes(N-t)} \otimes \hat{U} \otimes \mathbb{1}^{\otimes(t-1)} \ |\psi\rangle \,, \tag{6.3}$$

although we naturally avoid any wasteful instantiation of the identity matrices. Without loss of generality, let

$$|\psi\rangle = \sum_{i}^{2^{N-t}} \sum_{j}^{2^{t-1}} \alpha_{ij} |i\rangle |0\rangle |j\rangle + \beta_{ij} |i\rangle |1\rangle |j\rangle \,, \tag{6.4}$$

where $\alpha_{ij},\ \beta_{ij} \in \mathbb{C}$ together form 2^N amplitudes of $|\psi\rangle$, and the kets represent the computational basis states of $N-t$, 1 and $t-1$ -qubit substates (when read left to right) respectively. The middle ket will be targeted by $\hat{U}$. Since $\hat{U}|0\rangle = u_{11}|0\rangle + u_{21}|1\rangle$ and $\hat{U}|1\rangle = u_{12}|0\rangle + u_{22}|1\rangle$, then applying $\hat{U}$ to qubit t produces

$$\begin{aligned} \hat{U}_t |\psi\rangle = \sum_{i}^{2^{N-t}} \sum_{j}^{2^{t-1}} & \left(u_{11}\,\alpha_{ij} + u_{12}\,\beta_{ij}\right) |i\rangle |0\rangle |j\rangle + \\ & \left(u_{22}\,\beta_{ij} + u_{21}\,\alpha_{ij}\right) |i\rangle |1\rangle |j\rangle \,. \end{aligned} \tag{6.5}$$

Each amplitude in the statevector has undergone a complex-weighted sum with a pair amplitude, at an index 2^{t-1} apart. Without yet considering any implementation nuances, this admits a simple local, serial, iterative method to update $|\psi\rangle$ (as represented by an array $\vec{\psi}$ with n-th amplitude $\vec{\psi}[n]$) which we present in Algorithm 6.1. Notice that since the amplitudes are modified in unique pairs, this algorithm is highly parallelizable; each of the 2^{N-1} pairs of amplitudes could be modified by independent and parallel processes. This pattern of memory access and modification is visualised in Figure 6.4.

Since the single-qubit gate potentially modifies every one of the 2^N amplitudes, as

Algorithm 6.1: Unoptimised local serial in-place simulation of a single-qubit gate $\hat{U}$ (with elements u_{ij}), on qubit $t \geq 0$ of an N-qubit vector $\vec{\psi}$. The comments make reference to the states made explicit in Equation 6.5.

local_singleTargetGate($\vec{\psi}$, N, $\{u_{ij}\}$, t):
    numPairs $= 2^N/2$
    pairSpace $= 2^t$
    // $|k\rangle = |0\rangle\,|i\rangle\,|j\rangle$
    for k **in** $\{0,\ 1,\ \ldots,\ \text{numPairs} - 1\}$:
        // index of $|i\rangle\,|0\rangle\,|0\rangle^{\otimes}$
        offset $= \lfloor k/\text{pairSpace} \rfloor\ 2^{t+1}$
        // index of $|i\rangle\,|0\rangle\,|j\rangle$
        indα $=$ offset $+\ k\,\%\,$pairSpace
        // index of $|i\rangle\,|1\rangle\,|j\rangle$
        indβ $=$ indα $+$ pairSpace
        $\alpha = \vec{\psi}[\text{ind}\alpha]$
        $\beta = \vec{\psi}[\text{ind}\beta]$
        $\vec{\psi}[\text{ind}\alpha] = u_{11}\,\alpha + u_{12}\,\beta$
        $\vec{\psi}[\text{ind}\beta] = u_{22}\,\beta + u_{21}\,\alpha$

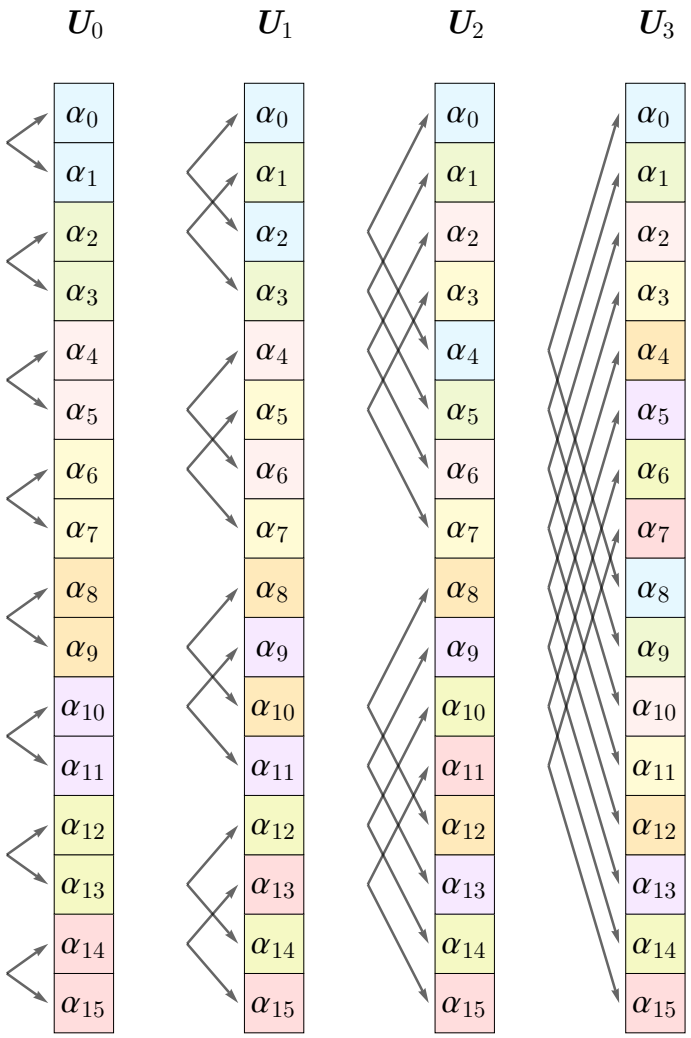


Figure 6.4: The pattern of amplitude combination during simulation of a single-qubit unitary $\hat{U}_t$ upon qubit $t \geq 0$ of a 4-qubit statevector $\sum_i \alpha_i |i\rangle$. Amplitudes which share a colour (and are joined by an arrow) are modified to become a complex-weighted sum of each other in the output register, as indicated by Equation 6.5 and prescribed by Algorithms 6.1 and 6.2.

Algorithm 6.2: Optimised (with bitwise operations) local serial in-place simulation of a single-qubit gate. This is asymptotically equivalent to Algorithm 6.4, although involves fewer non-bitwise operations and executes significantly faster.

local_singleTargetGate($\vec{\psi}$, N, $\{u_{ij}\}$, t):
    numPairs $= 2^N/2$
    // $|k\rangle = |0\rangle\,|i\rangle\,|j\rangle$
    for k **in** $\{0,\ 1,\ \ldots,$ numPairs$)$:
        // index of $|i\rangle\,|0\rangle\,|0\rangle^{\otimes}$
        ind$i = (k \gg t) \ll (t+1)$
        // index of $|0\rangle^{\otimes}\,|0\rangle\,|j\rangle$
        ind$j = k\,\&\,((1 \ll t) - 1)$
        // index of $|i\rangle\,|0\rangle\,|j\rangle$
        ind$\alpha =$ ind$i\,|\,$ind$j$
        // index of $|i\rangle\,|1\rangle\,|j\rangle$
        ind$\beta =$ ind$\alpha \oplus (1 \ll t)$
        $\alpha = \vec{\psi}$[indα]
        $\beta = \vec{\psi}$[indβ]
        $\vec{\psi}$[indα] $= u_{11}\,\alpha + u_{12}\,\beta$
        $\vec{\psi}$[indβ] $= u_{22}\,\beta + u_{21}\,\alpha$

does our Algorithm 6.2 without repetition, it is asymptotically optimal. But its implementation can be improved from that directly suggested by our pseudocode, via our considerations of the previous section. Notice each of the $2^N/2$ iterations invokes an integer division or a floor operation (Line 6.1), a mod operation (Line 6.1), and arithmetic of large integers (Lines 6.1 and 6.1). This is because we have not fully appreciated the connection between the computational space of qubits, and classical bit sequences. All of these operations can be replaced with relatively inexpensive bitwise interleaving operations, as demonstrated in Algorithm 6.2 (and more thoroughly described in the proceeding section). These seemingly innocuous changes provide a factor $\approx 3.6\times$ speedup for a 24-qubit statevector with random amplitudes, under a random single-qubit gate, simulated on the i7-4770 [267]. For the remainder of this chapter, we will use bitwise operations for all arithmetic concerning the indices of computational basis states.

Controlled gate

We next consider the controlled gate, which will involve considerations of caching, branching and automatic vectorisation. We begin with an algorithmic derivation. The introduction of a control qubit $c \geq 1$ (and purely for notational simplicity, $c > t$) into Equation 6.3 resembles

$$C_c[\hat{U}_t] = \mathbb{1}^{\otimes(N-c)} \otimes |0\rangle\langle 0| \otimes \mathbb{1}^{c-1} \tag{6.6}$$

$$+\mathbb{1}^{\otimes(N-c)} \otimes |1\rangle\langle 1| \otimes \mathbb{1}^{\otimes(c-t-1)} \otimes \hat{U} \otimes \mathbb{1}^{\otimes(t-1)}. \tag{6.7}$$

This prescribes an identical action as $\hat{U}_t$ upon the state *except* that we do not modify any amplitude for which the corresponding computational basis state has the c-th qubit in state $|0\rangle$. We will ergo concentrate on the determination of which amplitudes to modify, and for clarity, denote the modification of an amplitude α_i (of N-qubit basis state $|i\rangle$) as merely $\alpha_i \to f(\alpha_i)$. These will be amplitudes with indices satisfying $i_{[c]} = 1$, which denotes that the c-th bit of unsigned integer i is one. We also define $\Lambda = 2^N$ as the number of amplitudes in the statevector.

Introducing this "control condition" into our gate simulation (replacing lines 6.2-6.2 of Alg. 6.2) can be as simple as pseudocode

```
for every index i                                    (method A)
    if the c-th bit of i is 1
        α_i → f(α_i)
```

Our `for` loop undergoes Λ iterations, and modifies $\Lambda/2$ total amplitudes. Unfortunately, we have introduced a branching condition into an exponentially large loop. In

principle, the runtime penalty of an occasional failed branch prediction could exceed the cost of evaluating f and modifying the amplitude α_i in global memory. In multi-threaded simulation, it may furthermore reduce thread uniformity, elaborated upon in the proceeding section.

It is easy to naively re-express the same scheme without branching, as

```
for every index i                                    (method B)
    let b be the c-th bit of i
    α_i → (1 − b) α_i + b f(α_i)
```

where our loop still iterates Λ times, but we now (attemptedly) modify every one of Λ amplitudes - half are incidentally assigned their existing value. Note though that every amplitude modification (and resulting memory access) will trigger either a cache hit, else an expensive cache miss in which the cache will be updated with a new contiguous sub-array of the statevector amplitudes. Since the distance between amplitude sub-arrays which actually need modification (2^c) can be quite large, and the cache line quite small ($\approx$ 2 to 16 double-precision complex amplitudes [273]), method B is expected to cause *twice* as many cache misses than method A. In instances where evaluating $f(\alpha_i)$ is very simple (a few complex floating-point operations, like in the previous section's single-target gate), this caching penalty will likely dominate the total runtime and slow simulation by up to a factor two. And since f itself is needlessly evaluated every iteration in this scheme, the method will suffer even when f is relatively costly.

Fortunately, we can restructure our loop to avoid both branching *and* superfluous memory accesses. We iterate all local basis states $|j\rangle$ of a register with *one fewer qubits*, and then interleave the control bit in its 1 state. That is, we iterate all $(N-1)$-bit indices

$j \in \{0, 1, \ldots, \Lambda/2)$, each corresponding to basis state

$$|j\rangle_{N-1} = |j_{[N-2]}\rangle \ldots |j_{[1]}\rangle\, |j_{[0]}\rangle\,,$$

and into each, insert the $|1\rangle$ state at index c to produce the index of basis state

$$|i\rangle_{N} = |j_{[N-2]}\rangle \ldots |j_{[c]}\rangle\, |1\rangle\, |j_{[c-1]}\rangle \ldots |j_{[1]}\rangle\, |j_{[0]}\rangle\,.$$

Doing so allows us to arrive directly at each index satisfying the control condition, corresponding to an amplitude needing modification:

```
for every value j of (N − 1) bits
    i = insert 1 as the c-th bit into j
    α_i → f(α_i)                                   (method C)
```

We iterate only $\Lambda/2$ times, yet modify only $\Lambda/2$ amplitudes, avoiding any branching. Since i evaluates to batches of ordered contiguous indices, we have minimised the number of cache misses. Note too that evaluation of i can be done rapidly with $\mathcal{O}(1)$ bitwise operations, as we saw possible for the single-qubit gate.

The final optimisation possible is to guide the compiler toward automatic vectorisation of the arithmetic in our loops. We observe that there being a single control qubit c implies a consistent distance 2^{c+1} between the batches (of size 2^c) of modified amplitudes. So we could iterate every *second* batch of 2^c contiguous indices which we know ahead of time all satisfy the control condition.

```
for every batch where c is 1
    for every index i in that batch
        α_i → f(α_i)                                    (method D)
```

We again perform a total of only $\Lambda/2$ iterations and amplitude modifications without branching, although we have used *two* `for` loops. Like method C, all logic to determine the ultimately modified indices can be performed bitwise, and the methods are asymptotically identical. However, it is now explicit to the compiler that the inner `for` loop modifies only contiguous array elements in sequence; a clever compiler can then integrate automatic vectorisation, performing the array modifications of several independent iterations simultaneously on SIMD-supporting hardware. In serial simulation, this can run significantly faster than method C, as much as $2\times$ on the i7-4770. The nested loops pose only a minor obstacle to local parallelisation strategies like multithreading.

We formalise the four aforementioned single-control simulation methods in Algorithm 6.3, and present their benchmarked performance in serial and multithreaded local simulation upon a 31-qubit statevector in Figure 6.5. Unsurprisingly, the cache penalties of method D make it the worst performing in all settings. Curiously, the superfluous branching of method A did not impact it in serial simulation where it outperformed method C, attributable to robust branch prediction, although it parallelised inferiorly. Though not shown here, the natural multi-control extension of method C exponentially outperforms the others, since it leverages that exponentially fewer amplitudes need modification.

Algorithm 6.3: Methods of incorporating a single control qubit $c \geq 0$ into serial, local simulation of a process which otherwise modifies amplitudes under $\alpha_i \to f(\alpha_i)$. We define $\Lambda = 2^N$ as the number of amplitudes in the N-qubit statevector $\vec{\psi}$. Though platform specific, we present their relative performance in characteristic HPC settings in Figure 6.5.

```
// in-line convenience functions
getBit(n, i):
    return (n ≫ i) & 1

flipBit(n, i):
    return n ⊕ (1 ≪ i)

insertZeroBit(n, i):
    l = (n ≫ i) ≪ (i + 1)
    r = n & ((1 ≪ i) − 1)
    return l | r

getPrefixZeroSuffix(j, t, k)
    return (j ≪ (t + 1))|(1 ≪ t)|k

// method A
for i in {0, ..., Λ − 1}
    if getBit(i, c) is 1
        ψ⃗[i] = f(ψ⃗[i])

// method B
for i in {0, ..., Λ − 1}
    b = getBit(i, c)
    ψ⃗[i] = (1 − b) ψ⃗[i] + b f(ψ⃗[i])

// method C
for j in {0, ..., Λ/2 − 1}
    i = insertZeroBit(j, c)
    i = flipBit(i, c)
    ψ⃗[i] = f(ψ⃗[i])

// method D
for k in {0, ..., Λ/2^(c+1) − 1}
    for j in {0, ..., 2^c − 1}
        i = getPrefixZeroSuffix(k, c, j)
        i = flipBit(i, c)
        ψ⃗[i] = f(ψ⃗[i])
```

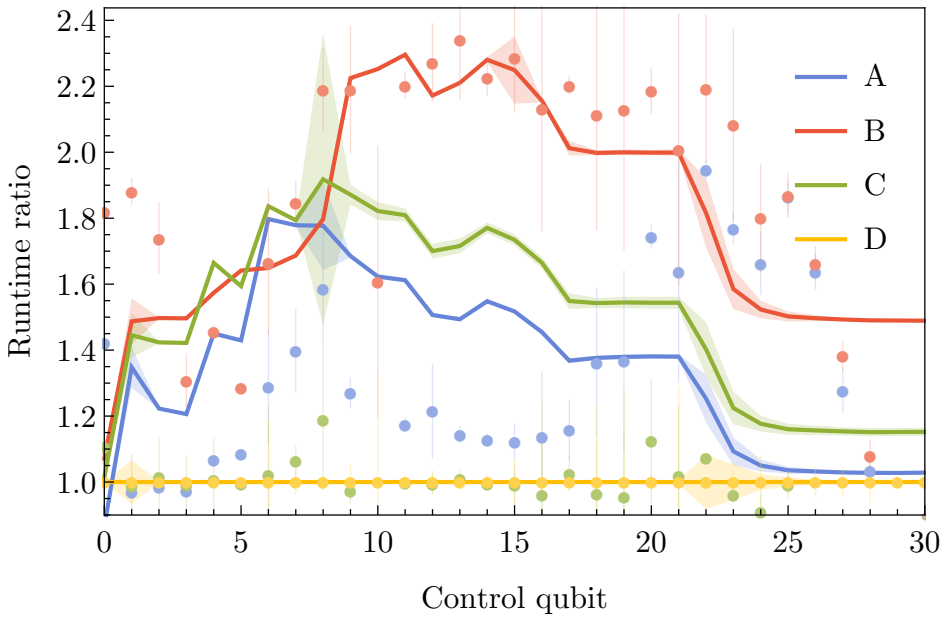


Figure 6.5: Runtime performance of the local single-control qubit simulation methods of Algorithm 6.3, upon a 31-qubit statevector. The horizontal axis is the index of the control qubit (c), and the vertical axis is the average factor runtime versus method D. Lines indicate serial simulation, and points indicate multithreaded simulation with 96 threads. Shaded regions and vertical lines indicate 3 standard deviations of uncertainty among the 100 repetitions for each control qubit. Benchmarks were run at double precision on a two-socket 48-core 348 GiB machine, such that the 32 GiB statevector spanned one socket. The testing code was compiled using the most aggressive optimisation flags possible (`gcc`'s `O3`), although relative un-optimised behaviour was similar.

6.4 Multithreading

6.4.1 Facilities

Multithreaded applications leverage multiple CPUs or cores to perform multiple tasks in parallel, using *shared memory* [274]. Typically, each CPU resides in a separate *socket* with its own local random-access memory space. These spaces together constitute parallel RAM (PRAM), all of which is accessible to any CPU through an interconnecting bus. This is abstracted at the level of the user's code, so that accessing local vs another socket's memory is distinguishable only by the latency penalty due to the bus exchange;

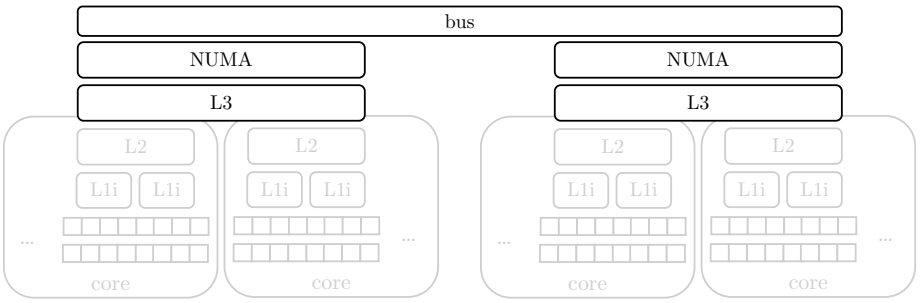


Figure 6.6: The memory hierarchy of a multi-CPU shared-memory system (in this example, two CPUs each with two cores). The main memory of the program is divided between NUMA registers connected by a high-throughput bus, together constituted a variable latency PRAM. In grey are the single-core diagrams of Figure 6.3.

this constitutes single non-uniform memory access (NUMA) [275]. The local memory of each socket is sometimes itself referred to as a NUMA node. We illustrate the general structure in Figure 6.6.

Effective use of multithreading means dividing a task into independent equally-costly subtasks which make use of separate partitions of the main memory, and dispatching them to separate CPUs cores. Each independent process performing a subtask is referred to as a *thread*. The allocation of threads to cores is typically a duty of the compiler when given explicit details about the data dependencies of the subtasks. Large loops which process independent array elements are easily parallelised and are excellent candidates for automatic multithreaded acceleration. Still, high-performance code should diligently monitor the parallelisation granularity, thread load and caching behaviour. These in-turn mean giving each thread as large a task as possible to shrink invocation costs, giving threads uniform loads by avoiding early terminations and branching, and assigning threads to write to separate cache-lines. This latter endeavour avoids *false sharing* [276] whereby separate threads modify data within the same cache-line and prompt the entire cache to be superfluous propagated and reloaded to avoid collision. False sharing in large loops processing large amounts of memory can be catastrophic

to multithreaded performance. As such, high-performance code ensures the *stride* between data accessed by adjacent threads exceeds the cache-line size [277]. This sometimes mean duplicating code at compile-time with static stride configurations, as we will soon demonstrate.

In typical settings, threads are virtual and are not strictly bound to physical CPU cores. The number of simultaneous threads executing a program will not be known at compile time, and is instead dynamically chosen with regard to the executing hardware. Since each thread has necessarily private local memory, the combined memory cost of all threads allocated to a core is *a priori* unknown. If high-performance code is not careful to limit it, the combined thread memory may exceed the L1d and L2 caches of a CPU core and trigger caching penalties which exceed the benefit of parallelism. This places pressure to use fewer simultaneous threads. Meanwhile, almost all operating systems impose an upperbound on the per-thread memory, through page limits and finite stack sizes. If this is exceeded at runtime, a stack overflow error will halt execution. This places pressure to use less memory per threads. As such, diligent high-performance code will manage an equilibrium of thread number, thread task size and thread memory.

We finally mention that independent simultaneous threads which contribute to the same output datum (like a mean value) must modify its intermediate variable strictly sequentially, else encounter a *race condition* [278] and a resulting erroneous output. If there are many logically parallel threads all contributing to the same datum, they should employ a hierarchal parallel reduction in a logarithmic number of sequential rounds, to avoid a totally sequential linear-time modification. Such facilities are often abstracted in the multithreading framework, but their performance implications must be considered by the programmer.

6.4.2 Simulation

Many-qubit gate

To illustrate these considerations, we describe simulating a general n-qubit gate $\hat{U} : \mathbb{C}^{2^n \times 2^n}$ with elements u_{kl}, acting upon an N-qubit statevector $|\psi\rangle$ (where $N \geq n$). This a generalisation of the single-qubit gate of Algorithm 6.2. Let $\vec{t} = \{t_1, \ldots, t_n\}$ be a (not necessarily sorted) list of target qubits indexed (least significant) from zero ($t_i \in \{0, \ldots, N-1\}$). Without loss of generality, let

$$|\psi\rangle = \sum_{i}^{2^{N-n}} \sum_{j}^{2^n} \alpha_{ij} |i, j\rangle , \qquad \alpha_{ij} \in \mathbb{C}, \tag{6.8}$$

where $|i, j\rangle$ denotes the computational basis state whereby the target qubits (as per the ordering in t) produce j, and the remaining $N - n$ qubits produce i. By definition

$$\hat{U} |j\rangle = \sum_{k}^{2^n} u_{jk} |k\rangle , \tag{6.9}$$

and so we have

$$\hat{U}_{\vec{t}} |\psi\rangle = \sum_{i}^{2^{N-n}} \sum_{j}^{2^n} \alpha_{ij} \sum_{k}^{2^n} u_{jk} |i, k\rangle , \tag{6.10}$$

which by swapping labels $i \leftrightarrow k$, can be expressed as

$$\hat{U}_{\vec{t}} |\psi\rangle = \sum_{i}^{2^{N-n}} \sum_{j}^{2^n} \left(\sum_{k}^{2^n} \alpha_{ik} u_{kj} \right) |i, j\rangle , \tag{6.11}$$

to make clear that the amplitudes are modified under $\hat{U}_{\vec{t}}$ to

$$\alpha_{ij} \to \sum_{k}^{2^n} \alpha_{ik}\, u_{kj}. \tag{6.12}$$

We see that each amplitude becomes the result of multiplying 2^n amplitudes (including itself) with a column of $\hat{U}$. The 1D statevector indices of α_{ij} for all 2^n values of j can be obtained in sequence by bitwise modification of i. This informs a local iterative algorithm for applying $\hat{U}_{\vec{t}}$, using $\mathcal{O}(2^n)$ additional memory to store α_{ij} during modification. We present this in Algorithm 6.4, and the resulting amplitude access and modification patterns in Figure 6.7.

The implementation-level nuances enter Algorithm 6.4 at the comments of lines 6.4 and 6.4, which imitate multithreading pre-compiler directives. We can leverage that each iteration of the outer loop (variable k) modifies a unique subset of the amplitudes, with new values determined by the same subset; iterations are hence independent and can be run in parallel. Note though that each thread must retain its own private copy of the 2^n-length temporary arrays. This exponentially-growing memory overhead would quickly cause cache invalidation and stack overflows, hence our code implicitly assumes n is small; for example, fewer than 16 qubits which would otherwise overflow (at double precision) a 1 MiB stack. This is a precondition that a rigorous implementation must check in advance.

Mitigating false sharing is a harder challenge. For a general set of target qubits $\vec{t}$, it is not possible to structure our code such that distinct iterations modify disjoint cache-lines; we cannot even ensure that a *single* iteration modifies only a single cache-line. Instead, a single iteration (and hence potentially, a single thread) may modify exponentially distant amplitudes and prompt $\mathcal{O}(2^n)$ cache misses. While there is ergo

Algorithm 6.4: Local simulation of an n-qubit general unitary $\hat{U}_{\vec{t}}$ with elements u_{ij}, operating upon qubits $\vec{t} = \{t_j \geq 0 : 0 \leq j < n\}$ of an N-qubit pure state $\vec{\psi}$, in $\mathcal{O}(2^{N+n})$ time and $\mathcal{O}(2^n)$ additional memory. Multithreading will require every of p threads have private arrays $\vec{\text{ampInds}}$ and $\vec{\text{amps}}$, and hence imposes a total memory overhead $\mathcal{O}(p\,2^n)$. Subroutine **assignBits** must be in-lined.

assignBits$(n, \vec{t}, v)$:
    for j **in** $\{0, \ldots,$ **length**$(\vec{t}))$:
        $b =$ **getBit**(v, j)
        $n = n \,\&\, (b \ll \vec{t}[j])$
    return n

multiTargetGate$(\vec{\psi},\ \{u_{ij}\},\ \vec{t})$:
    colSize $= 2^n$
    sorted$(\vec{t})$
    create array $\vec{s} =$ **sorted**$(\vec{t})$
    // assign each thread a private copy of below arrays
    create array $\vec{\text{ampInds}}$ **of length** colSize
    create array $\vec{\text{amps}}$ **of length** colSize
    // assign each thread independent iterations (values of k)
    for k **in** $\{0,\ 1,\ \ldots,\ 2^N/\text{colSize} - 1\}$:
        // index of $|i, j = 0\rangle$
        ind0 = k
        for s **in** $\vec{s}$:
            ind0 = **insertZeroBit**(ind0, s)
        // copy the target amplitudes
        for r **in** $\{0, \ldots,$ colSize $- 1\}$:
            // index of $|i, j = r\rangle$
            ind = **assignBits**(ind0, $\vec{t}$, r)
            $\vec{\text{ampInds}}[r]$ = ind
            $\vec{\text{amps}}[r] = \vec{\psi}$[ind]
        // modify the target amplitudes
        for r **in** $\{0, \ldots,$ colSize $- 1\}$:
            ind = $\vec{\text{ampInds}}[r]$
            $\vec{\psi}$[ind] $= 0$
            for c **in** $\{0, \ldots,$ colSize $- 1\}$:
                $\vec{\psi}$[ind] $= \vec{\psi}$[ind] $+ u_{rc}$ $\vec{\text{amps}}[c]$
    free $\vec{s}$
    free $\vec{\text{ampInds}}$
    free $\vec{\text{amps}}$

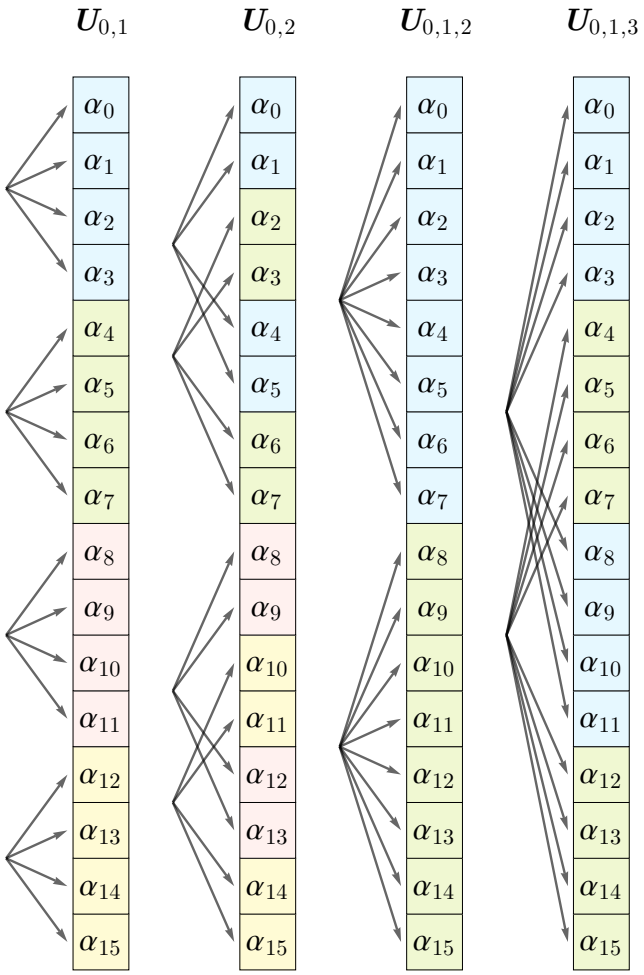


Figure 6.7: The pattern of amplitude combination during local simulation of a many-qubit unitary $\hat{U}_{\vec{t}}$ upon qubits $\vec{t}$ of a 4-qubit statevector $\sum_i \alpha_i |i\rangle$. Amplitudes which share a colour (and are joined by an arrow) are modified to become a complex-weighted sum of each other in the output register, as indicated by Equation 6.12 and prescribed by Algorithm 6.4.

no universally optimum thread allocation scheme, we can still avoid false sharing in many scenarios (like when n is few, or targets the first n qubits $\{0, \dots, n)$) by exploiting several helpful properties of our algorithm.

- Each iteration has an identical workload without branching.
- There are a power-of-2 number of iterations.
- A cache-line can typically contain a power-of-2 number of amplitudes.
- There are typically a power-of-2 number of available threads.
- The number of total iterations is expected to be significantly exceed the number of threads.

These properties allows us to statically pre-allocate each thread to a contiguous, (approximately) uniform subset of the iterations, which we attempt to make as large as possible to minimise new thread invocations. Unfortunately we cannot know this ideal configuration at compile-time as necessary for using (for example) `OpenMP` scheduling, since it depends on runtime variables like the number of amplitudes and threads. We can circumvent this by defining *multiple* duplicates of Algorithm 6.4 which each assume different static multithreaded configurations. This duplication can be hidden to a caller through a wrapper function which chooses which backend function to invoke at runtime, at a negligible runtime penalty, and an unimportant penalty to the compiled code size. These duplicated definitions can be generated automatically with compiler macros.

We hope to have demonstrated that the task of parallelising a simple simulation algorithm with multithreading, even when loop iterations are independent, can be somewhat complicated. The author has contributed over 100 multithreaded facilities to the QuEST project, which show excellence scaling runtime performance in Figure 7.9 of the proceeding chapter.

6.5 Hardware acceleration

Hardware acceleration refers to the dispatching of tasks to specialised hardware in order to run faster than if dispatched to a generic CPU. We will focus on general-purpose graphical-processing unit (GPGPU) acceleration, whereby we repurpose the highly-parallel architecture of GPUs as are optimised for the linear algebra common in graphical processing. We further limit ourselves to consideration of NVIDIA GPUs with the Compute Unified Device Architecture (CUDA) [279]. Such devices are massively parallel and can significantly accelerate the homogeneous data processing [280] characteristic of statevector simulation. While rewarding, CUDA programming introduces many constraints and complications to memory and thread management not seen in other parallelisation techniques.

6.5.1 Facilities

Section 6.3.1 described the core of a CPU which has simple, heirarchal caches, and which can operate in almost total independence from the other CPU cores. This is not true of CUDA cores, each of which have many registers and share a modest L1 cache with other cores in the same *stream multiprocessor* (SM). A GPU contains many SMs which all share an L2 cache as an intermediate bank against the large and relatively slow global GPU memory, the latter of which (dubbed *VRAM*) can exchange with RAM. We visualise this architecture in Figure 6.8. For a sense of scale, the Quadro P6000 (a high-end GPGPU) has 3840 cores between 30 SMs, and 24 GiB of VRAM [281]. The allocation of virtual tasks to processes executing upon the hardware of a GPU can be substantially complicated since the GPU simultaneously employs efficient parallelisation, pipelining and context switching [282]. We offer a bottom-up explanation.

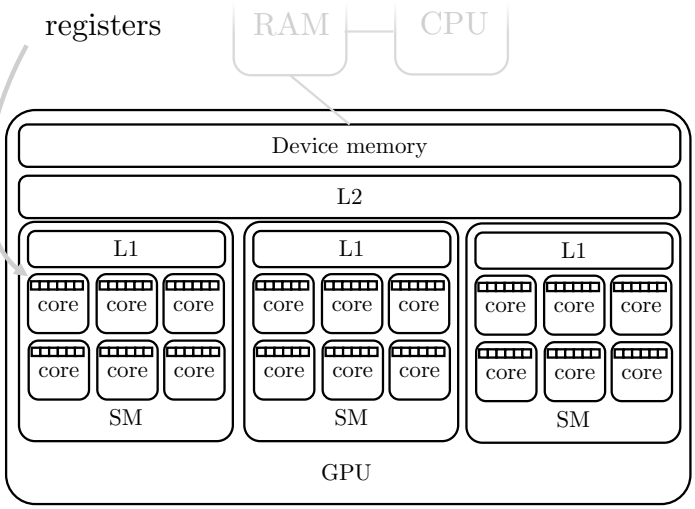


Figure 6.8: The typical architecture of an NVIDIA GPU as interfaced through CUDA (simplified, showing only the facilities relevant to this chapter).

A GPU application typically consists of some memory preparation code and one or more kernel functions to be invoked in (logical) parallel many times over (*tens of millions* in the case of quantum simulation). The smallest meaningful process is a *thread* which (virtually) performs one kernel invocation, and which ideally processes independent data from other simultaneous threads. Unlike in a multicore CPU, GPU threads cannot be treated asynchronously. This is because 32 threads are bound together in a *warp* which is indivisibly dispatched to a single CUDA core, and which sees the threads therein strictly synchronised by a shared instruction clock. A single warp can ergo only perform SIMD, although its unusual mechanism for doing so through lock-step threads is referred to as single-instruction multiple-thread (SIMT). Even when a kernel function contains a branching condition satisfied by only some threads, the threads within a warp will not diverge, but instead together explore the branch outcomes sequentially (with some threads *masked*, waiting idly). High-performance GPU code will ergo minimise intra-warp thread branching[1].

[1]We caution that recent GPU architectures *can* diverge threads within a warp, and ergo newer

Warps are virtually organised, but when dispatched to a core, become *active*. Each thread has private memory which (in an active warp) is physically distributed between the registers of a core. Warps are virtually grouped into *blocks*, each of which are indivisibly deployed to a single SM. Threads can access block-wide (i.e. inter-warp) shared memory through the L1 cache of an SM, though this is not without complication. Firstly, the active warps within an SM are *not* synchronised, so a programmer must manually prevent race conditions. This is especially important since a block may contain more warps than there are physical cores in the SM, and hence some threads (in distinct warps) are effectively running sequentially. Secondly, every thread cannot simultaneously access (even when only *reading*) the shared memory, which is actually split into separate *banks* which can each service only a single thread at a time. Independent of their warp, only threads (within a block) accessing distinct shared memory banks may do so in parallel, else encounter a *bank conflict* and consequential sequential access. While bank sizes are fixed on old hardware, they are compile-time configurable on modern GPUs, to produce at-most 32 banks per SM. We finally note that multiple threads requesting the same address in L1 (rather distinct addresses within the same bank) can do so without collision via a broadcast. High-performance GPU code will ergo deliberately schedule and stagger shared memory access to minimise bank conflicts, configure optimal bank sizes, and synchronise thread reading of the same data.

Typically, *multiple* blocks will be enqueued to run sequentially on an SM. However, an SM can *context switch* between blocks with almost no overhead, and pipeline processing of multiple blocks; as such, there can sometimes be said to be multiple active blocks per SM. Still, at any time, there is expected to be some number of *active blocks* (allocated to a SM though not necessarily with all warps actively running on cores), and a larger

CUDA versions ($\geq$ 9, released 2017) offer warp-level primitives such as intra-warp synchronisation (link). Such granular facilities appear irrelevant to the homogeneous memory-bound nature of quantum statevector simulation, and will not be further considered.

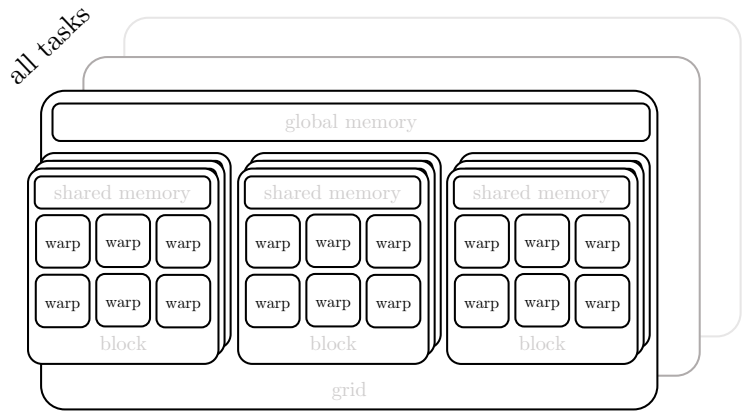


Figure 6.9: The virtual hierarchy of processes in CUDA. This illustration suggestively mimics the GPU architecture of Figure 6.9, but uses stacked boxes to indicate queueing of virtual tasks. Note this illustration makes several simplifications for clarity; it erroneously suggests all warps within a block are simultaneously deployed to cores (in reality, some may be *inactive*); it implies blocks are enqueued to specific SMs, though allocation from a global queue is instead decided at runtime; it suggests grids are executed sequentially, though they can in principal occur concurrently.

number of inactive blocks enqueued. The collection of all blocks corresponding to the same kernel invocation is called a *grid*, and all threads there-in share access to *global memory*, instantiated physically between the L2 cache and device memory. Since global memory access is much slower than intra-grid shared memory, a high-performing CUDA application imposes tight data locality to threads where possible. And when data must be aggregated between threads, it is done so ideally through hierarchal parallel reductions. In principle, the grids of distinct kernels can be simultaneously deployed on the GPU, but it is more likely these will be enqueued, and ergo that one kernel is being actively executed at any given time.

In summary, the invocations of a kernel (threads within the same grid) are spawned at once but are divided into blocks, each enqueued to be deployed to an available SM, and there-in further divided into warps, each enqueued to be deployed to a core within their block's SM. We visualise this virtual hierarchy in Figure 6.9, which attempts to

parallel the hardware architecture of Figure 6.8. For a sense of scale, the aforementioned 30-SM 3840-core Quadro P6000 can (in principle) run 122,880 simultaneous threads, although the *resident threads* to an SM is software capped at 2048 [281]. Hence if each thread processed one amplitude of a 30 qubit statevector (16 GiB total at double precision) and were divided uniformly into blocks containing 128 warps (as many cores per SM, of which only half can run concurrently), then the total processing would involve $262,144$ blocks and require $17,477$ sequential rounds. This is however an insufficient picture to estimate the runtime which will be affected by bank conflicts, pipelining and asynchronous context switching.

We have avoided any mention of the geometric layout and indexing scheme of grids, blocks and threads, since mostly irrelevant to the one-dimensional statevector processed by quantum simulation. We have further neglected description of several other kinds of specialised memory available to a CUDA application, like texture and surface memory, which can prove convenient for efficiently propagating dynamic but tractable runtime inputs (like few-qubit unitary matrix descriptions) to kernels. Finally, we have foregone discussion of optimising the parallelisation granularity using properties of the deployed GPU hardware, referred to as *tuning* [283, 284].

6.5.2 Simulation

Since quantum statevector simulation often involves only a few operations per amplitude of which there are exponentially many, it is generally *memory bound* [285]. That is, the runtime is dominated by the costs of transferring memory [286], rather than those of performing numerical operations. This manifests in the short bodies of the `for` loops in Algorithms 6.2 and 6.3. Being memory bound relaxes our adherence to the general

aforementioned performance considerations, and can yield simpler compromising code. For example, consider the initialisation of an N-qubit statevector (where $N \gg 1$) kept permanently in the GPU device memory, into the plus state $|+\rangle^{\otimes N} = \sum_i 1^{-N/2} |i\rangle$. It is tempting to mitigate bank collisions by distributing amplitude-modifying threads sparsely between blocks. However, in the simulation regime whereby GPU acceleration is worthwhile in the first instance (i.e. there are sufficiently many amplitudes to outweigh the overheads of moving memory between RAM and VRAM), we can safely assume the number of amplitudes greatly exceeds the total number of independent shared memory banks on the GPU. Ergo even naively allocating a thread to modify each amplitude will not increase the total number of serial memory accesses, and we only need to choose as many threads per block as we believe there are shared memory banks. We demonstrate this in Algorithm 6.5.

While the author has developed over 100 low-level CUDA functions for the QuEST [249] software project (git blame up to 13th Oct 2021), it is a challenge to describe any which navigate the previous high-performance considerations without first introducing significant new material non-specific to quantum simulation (such as tree-based parallel reduction [287]). Furthermore, many QuEST facilities do not abide by all previous high-performance considerations due to software development trade-offs more relevant to Chapter 7. In lieu of more illustrative examples of high-performance GPU algorithms, we offer benchmarks of QuEST's early GPU facilities simulating random circuits (described in Reference [288]) in Figure 7.10 of Chapter 7.

Algorithm 6.5: GPU-agnostic memory-bound initialisation of an N-qubit statevector (where N is assumed large) stored in global device memory, into the plus state $|+\rangle^{\otimes N}$. Since we expect only 32 concurrently-queryable banks of shared memory per SM, we assign only $4\times$ threads per block (> 1 so as to trade thread invocation overheads for warp context switches). We adopt the conventional good practice of ceiling our determination of the number of blocks, and explicitly checking the thread index legitimacy in the kernel, in case a later change to the simulation granularity yields block dimensions which do not evenly divide the number of amplitudes.

```
// a precompiler directive declares this function as a device kernel
initPlusStateKernel(dψ⃗, nA, c):
    // these are built-in thread-private variables
    i = blockId * blockDim + threadId
    if i ≥ nA
        return
    dψ⃗[i] = c

initPlusState(ψ⃗):
    Let dψ⃗ be the persistent copy of ψ⃗ in GPU device memory
    Let nA be the number of amplitudes in dψ⃗
    Let c = 1/√(2^nA)

    // fix this as a small multiple of 32, the max banks per SM
    Let threadsPerBlock = 128
    Let numBlocks = ⌈nA / threadsPerBlock⌉
    With configuration (numBlocks, threadsPerBlock):
        Launch initPlusStateKernel(dψ⃗, nA, c)
```

6.6 Distribution

We have reached the most substantial and challenging classical high-performance simulation paradigm of this thesis, to which the author has given most of their research attention. Unfortunately, a comprehensive treatment of these efforts would greatly exceed the limits of this thesis, and as such, we offer only a cursory introduction to the topic, and some examples and summary of my contributions. Table 6.1 summarises my full efforts.

6.6.1 Facilities

Distributed computing involves a large number of independent computers cooperating over a network to solve a common problem [289]. These are typically compute nodes in supercomputers connected via low-latency high-bandwidth networks, but which still suffer considerable inter-node communication costs relative to the latencies and bandwidths between local CPUs and RAM. Ergo, high-performance distributed computing often means the decomposition of a problem into independent, parallel sub-problems requiring minimal inter-node communication.

Distribution is a crucial deployment paradigm for large simulations for two main reasons. Firstly, it provides an easily *scalable* form of parallelisation whereby additional modular compute nodes can help accelerate a task by connecting to the network. Secondly, the total process memory available to the computation, as aggregated between all participating machines, can greatly exceed the RAM available to any one machine. Provided that a single machine is never required to locally store all data at once, distributed computing networks can process ludicrously large amounts of data. Indeed, full-state simulation of quantum statevectors above a certain size (about 34 qubits at 256 GiB) can *only* be performed using distribution.

Developing a distributed application can be challenging. The paradigm is decentralised and often unintuitive, and requires every independent process to asynchronously execute identical code albeit with different initial data, synchronising when communication is necessary. Yet, the independently executing machines or *nodes* must have consensus on program-wide decisions, such as outcomes of random processes like measurement. Furthermore, memory management becomes a fundamental algorithmic design, as opposed to a performance consideration. In the previous sections, we emphasised the

need for economical access of the shared memory on multicore and GPU platforms. On a distributed system however, there is no shared memory space whatsoever; processes can *only* exchange data through explicit *message passing* over the network, for which the programmer must manage the communication buffers and synchronisation.

6.6.2 Partitioning

The core design decision of a distributed simulator is how to partition and distribute the data. An N-qubit statevector is described by 2^N amplitudes, evenly divisible only into 2^k groups for $k \in \mathbb{N}$ [290]. We have seen (e.g. in Algorithms 6.2 and 6.4) that the local runtime cost of modifying an amplitude under the action of a gate is uniform across amplitudes. Ergo, one should uniformly distribute the statevector amplitudes between a power-of-2 number of nodes (constraining the *world size*), else wastefully incur additional serial tasks on one or more nodes.

The next essential design decision is the size of the *communication buffers*. These are persistent arrays on each node which receive data from other nodes to be processed before incorporation into local data. In this thesis, we assume communication buffers of size equal to the statevector partition on a node, i.e. of 2^{N-k} elements. This is in contrast to qHipster [291], a distributed simulator currently maintained by Intel, which uses buffers of *half* this size in order to fit a larger statevector partition per-node. It also enables them to overlap communication and local amplitude updating in simple gates. Our contrary choice of an equal-sized buffer has three motivations.

1. It means the later-presented distributed algorithms will require only a *single* (the minimum) serial round of communication, when nodes exchange their statevector partitions. This admits significantly cleaner code, simpler analyses, and in some

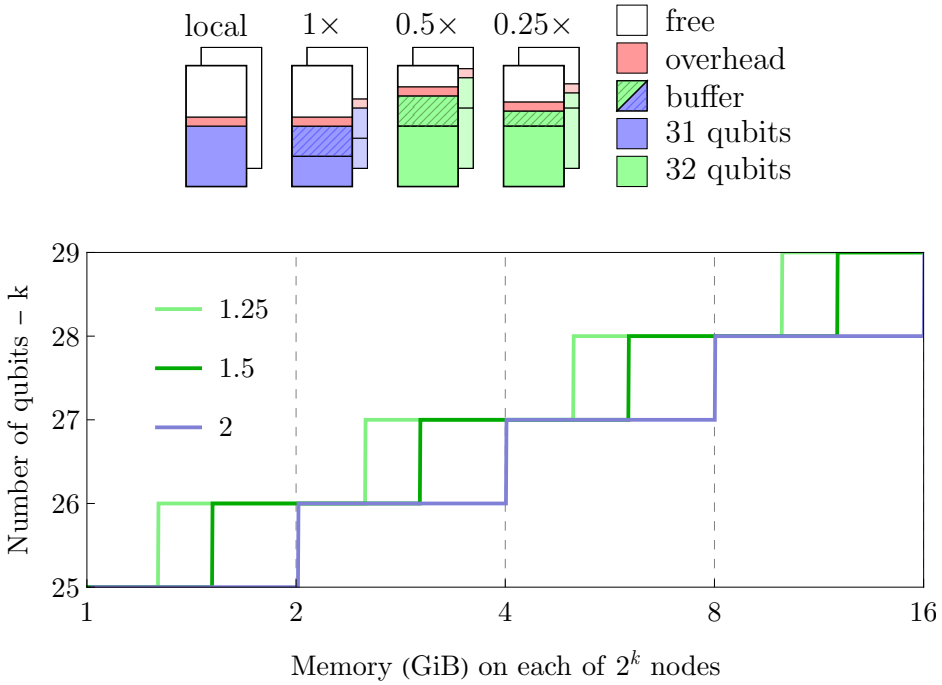


Figure 6.10: Relationship between communication buffer size and maximum simulation size. The top plot shows how the statevector partition (solid colour) and buffers (hashed) - of equal, half and quarter size - fill the node memory. The bottom plot shows the maximum number of qubits which can fit on 2^k nodes of varying memory, assuming a 50 MiB overhead. The 0.5× buffer is used by qHipster [291] to simulate a maximum of one more qubit over QuEST [249] which uses 1×.

cases, fewer algorithmic edge-cases.

2. It enables a family of advanced calculations, like that of fidelity between a statevector and density matrix, precluded by a half-size buffer.

3. It will mean at most *one* fewer qubits are simulatable on given hardware, than possible of the half-buffer configuration, which we deem an insignificant penalty. This is because modern RAM capacity is typically a power-of-2, but cannot all be dedicated to the statevector partition and buffer, due to the overheads of the operating system. As such, the partition itself cannot exceed 25% of RAM. A buffer of half size would however allow the partition to double in size (i.e. the

addition of a qubit) while leaving at least 25% of RAM free for overheads. Any subsequent reductions in buffer size would yield no benefit. We visualise this in Figure 6.10.

Each node in a distributed application is uniquely identified by its integer *rank*, which in our case satisfies $r \in \{0,\ \ldots,\ 2^k - 1\}$. This means that node r contains only amplitudes $\alpha_i \in \mathbb{C}$ with indices

$$i \in \{r2^{N-k},\ \ r2^{N-k} + 1,\ \ \ldots,\ \ (r+1)2^{N-k} - 1\}. \tag{6.13}$$

Equivalently, this is the amplitude of every N-qubit basis state $|i\rangle_N$ satisfying

$$|i\rangle_N \in \{\ |r\rangle_k \, |j\rangle_{N-k}\ :\ j \in \{0,\, 1,\, \ldots,\, 2^{N-k})\ \}. \tag{6.14}$$

6.6.3 Simulation

The constraints of distributed memory invalidate all gate simulation algorithms so-far seen. One must devise new distributed algorithms, but even doing so *inefficiently* in a way compatible with the memory and communication constraints of Section 6.6.2 can be challenging. This thesis of (attemptedly) forty thousand words lacks the space to present all of the author's novel distributed algorithms which, in a manuscript currently under preparation, take up twenty five thousand words. Instead, we will demonstrate a derivation of a basic operation - the single-qubit general gate - then rapidly describe how some more complicated and seemingly impossible examples can be efficiently performed, showcasing their emerging communication patterns.

Single-qubit gate

Let $\hat{U}_t$ be a single-qubit gate described by a general 2×2 complex matrix $\underline{\underline{U}}$ targeting qubit t of an N-qubit statevector, which is uniformly distributed between 2^k nodes. In Section 6.3.2, we observed that $\hat{U}_t$ combines amplitudes α_{ij} and β_{ij}, which are separated by 2^t indices (indexing qubit $t \geq 0$). For all i, j, these are the amplitudes of N-qubit basis states

$$\alpha_{ij} \; |i\rangle \, |0\rangle \, |j\rangle \, , \qquad\qquad \beta_{ij} \; |i\rangle \, |1\rangle \, |j\rangle \, , \tag{6.15}$$

where $|j\rangle$ are t-qubit substates. Because blocks of 2^t contiguous amplitudes are combined with only the next 2^t, contiguous blocks of $B = 2^{t+1}$ amplitudes remain independent of one another. And because the number of amplitudes contained in each node is $\Lambda = 2^{N-k}$, two communication scenarios emerge.

1) When $t < N - k \implies 2^t < \Lambda$, the index distance between every α_{ij} and β_{ij} is smaller than the number of amplitudes in each node. And since this implies $B \leq \Lambda$, we know that *every* α_{ij} has its corresponding β_{ij} amplitude contained within the same node. Hence, every node already contains all the amplitudes which determine its statevector partition under the operation of $\hat{U}_t$. No network exchange of amplitudes (nor communication of any kind) is needed, and we say that the simulation of $\hat{U}_t$ is "*embarrassingly parallel*" [292]. Furthermore, the relative local ordering of amplitudes is identical to their ordering in local non-distributed simulation; we can hence simply task each node with locally simulating $\hat{U}$ via Algorithm 6.2, as if each contained the entire statevector of only $N - k$ amplitudes.

2) When $t \geq N - k \implies 2^t \geq \Lambda$, it implies α_{ij} and β_{ij} are necessarily contained within separate nodes. Every node therefore cannot update its local statevector without first

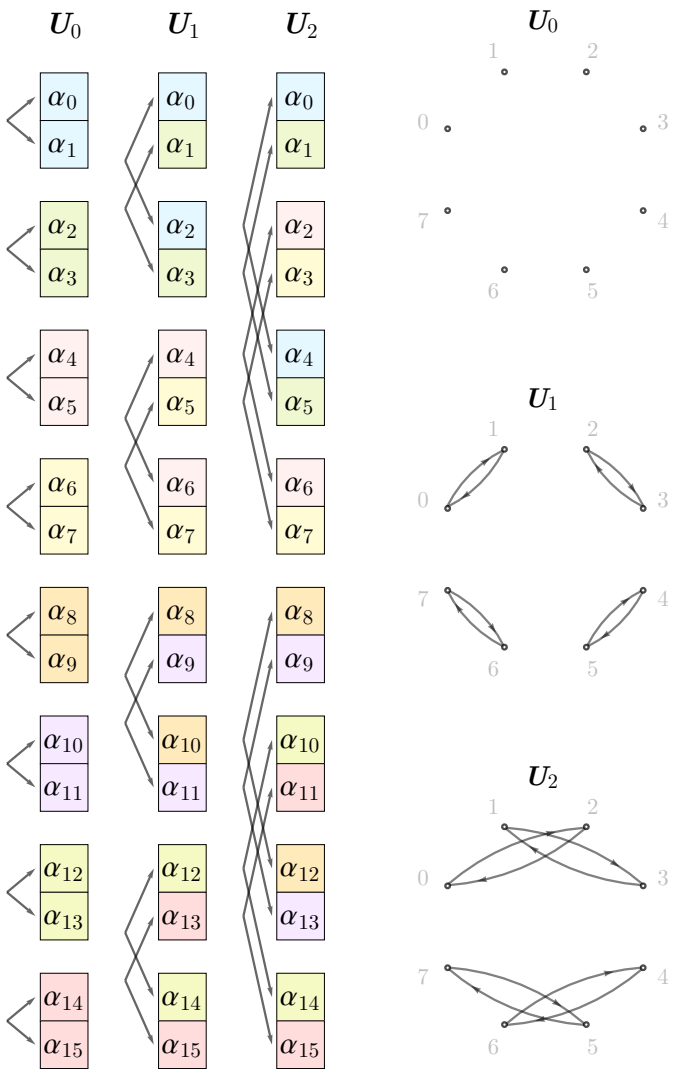


Figure 6.11: Example distributed communication patterns (right column) necessary to simulate single-qubit gates $\hat{U}_t$ with amplitudes modified as per the left three columns, when a 4-qubit statevector is distributed between $2^k = 8$ nodes (each containing two amplitudes). In the three diagrams of the right column, these nodes (with ranks 0-7) are illustrated by labelled vertices, and communication between them (the sending of amplitudes) signified by arrows. The simulations of $\hat{U}_1$ and $\hat{U}_2$ use communication pattern **a)** of Figure 6.12.

receiving amplitudes from other nodes - we visualise this necessity in Figure 6.11. This scenario implies strict inequality $\Lambda < B$, meaning each node contains only amplitudes of states $|i\rangle|0\rangle|j\rangle$ *or* of states $|i\rangle|1\rangle|j\rangle$, but never both. Therefore, all amplitudes within a node (with rank $r \geq 0$) combine with those in a single other node, which we refer to as its "*pair node*". Each node and its pair must exchange their full statevector partitions with each other. The pair node has rank

$$r' = r_{\neg(t-(N-k))} \equiv r \pm \left\lfloor \frac{2^t}{\Lambda} \right\rfloor, \tag{6.16}$$

with sign indicating whether the node contains α_{ij} $(+)$ or β_{ij} $(-)$ amplitudes. By using our Λ-sized communication buffers, each node can receive the entirety of its pair node's statevector partition in a single round of communication. And since all nodes are uniquely paired, these communications can happen simultaneously, assuming the network does not saturate. Each node is then only responsible for updating its local statevector partition, which it does in an iterative fashion similar to the local simulation of Algorithm 6.1, but with a minor change: Equation 6.5 informs us that the amplitudes are modified to become

$$\begin{cases} \alpha_{ij} \rightarrow u_{11}\,\alpha_{ij} + u_{12}\,\beta_{ij}, \\ \beta_{ij} \rightarrow u_{22}\,\beta_{ij} + u_{21}\,\alpha_{ij}, \end{cases} \tag{6.17}$$

where any given node contains either only α_{ij} amplitudes in its $\vec{\psi}$ partition (with β_{ij} in buffer $\vec{\varphi}$), or vice-versa. We determine which are contained by simply checking qubit t of any contained basis state, which is equivalent to the t-th bit of the global index of any contained amplitude - so for simplicity, we check the *first* contained, with global index $r\,\Lambda$. Hence if $(r\,\Lambda)_{[t]}$ is 0 or 1, then node r contains α_{ij} or β_{ij} amplitudes respectively.

Our final consideration is the ordering between local amplitudes in partition $\vec{\psi}$ and those they will combine with in buffer $\vec{\varphi}$. It may be tempting to iterate every local amplitude $\vec{\psi}[l] \; \forall \; l \in \{0, \; \ldots, \; \Lambda - 1\}$ and algebraically determine the index l' of its combining amplitude $\vec{\varphi}[l']$, via

$$l' = (r\,\Lambda + l)_{\neg t} \;\%\; \Lambda \tag{6.18}$$

$$= (r\,\Lambda + l)_{\neg t} \;\&\; (\Lambda - 1) \tag{6.19}$$

where we leverage Λ is a power-of-2 to replace the mod operator with a relatively fast bitwise `AND` operation. This expression produces the global index of each local statevector amplitude, flips the t-th qubit to obtain the global index of the pair amplitude, and finally maps it to a local buffer index by modding. While this obtains the correct index, it neglects a key observation; that the relative ordering of $|i\rangle\,|0\rangle\,|j\rangle \;\; \forall j$ and $|i\rangle\,|1\rangle\,|j\rangle \;\; \forall j$ is fixed between nodes, as hence is that between α_{ij} and $\beta_{ij} \; \forall j$. This means $l' = l \; \forall \; l$, and the local updating of amplitudes after communication is straightforward with an efficient memory stride. Further, all nodes can concurrently perform this procedure after their initial communicating, hence this phase of the algorithm is embarrassingly parallel.

We do not know in advance which of scenarios **1**) or **2**) will be invoked during distributed simulation, since N, k and t are all user input parameters. So our algorithm must incorporate both. We conclude that distributed simulation of a one-qubit general gate requires at most a *single* "serial round" of pair-wise statevector exchange before embarrassingly parallel local simulation. The total data transferred through the network is either zero or 2^N amplitudes (the full statevector), and each node thereafter performs $\mathcal{O}(2^{N-k})$ floating-point operations. We formalise this strategy in Algorithm 6.6, and some typical resulting communication patterns were shown Figure 6.11.

Algorithm 6.6: Distributed in-place simulation of a single-qubit general gate $\hat{U}$, represented by matrix $\underline{\underline{U}} : \mathbb{C}^{2\times 2}$ with elements u_{ij}, targeting qubit $t \geq 0$ of an N-qubit statevector $|\psi\rangle$ uniformly distributed between 2^k nodes. Each node has a unique rank $r \geq 0$, a statevector partition $\vec{\psi} : \mathbb{C}^{1\times 2^{N-k}}$, and an equally sized buffer $\vec{\varphi}$. This algorithm prescribes $\mathcal{O}(2^{N-k})$ floating-point operations and $\mathcal{O}(1)$ serial rounds of pairwise communication, via a call to **exchangeArrays** of Algorithm 6.7.

```
singleTargetGate(ψ⃗, N, U⃗⃗, t):
    if t < N − k
        local_singleTargetGate(ψ⃗, N − k, U⃗⃗, t)
    else
        // exchange amplitudes with pair node
        Λ = 2^{N−k}
        r′ = flipBit(r Λ, t) ≫ (N − k)
        exchangeArrays(ψ⃗, φ⃗, r′)

        // determine if node contains α or β
        if getBit(r Λ, t) is 0
            x = u11
            y = u12
        else
            x = u22
            y = u21

        // modify all local amplitudes
        for l in {0, ..., Λ − 1}
            ψ⃗[l] = x ψ⃗[l] + y φ⃗[l]
```

As arguably a quantum simulator's most ubiquitous operation, the one-qubit general gate sets a salient performance threshold. We should strive to develop algorithms for more advanced operations which yield similar resource costs to the one-qubit gate, for two reasons; firstly, for the user experience. Normalising simulation function runtimes prevents a user from encountering an unexpectedly large runtime or memory consumption due to a few invocations of an esoteric and unusually expensive operation. We will expand upon this in Chapter 7. Secondly, an untenably inefficient function may negate the runtime benefit of optimised functions in practical simulation settings. For these reasons, we seek to develop further distributed algorithms which modify the statevector *in-place*, in at most $\mathcal{O}(2^{N-k})$ local operations, and in a *fixed number* of serial rounds of strictly-pairwise communication.

These requirements seem untenably restrictive; yet a surprising number of more advanced gates are possible which admit the same resource costs as the single-qubit gate. They use one or more of the distributed communication patterns of Figure 6.12.

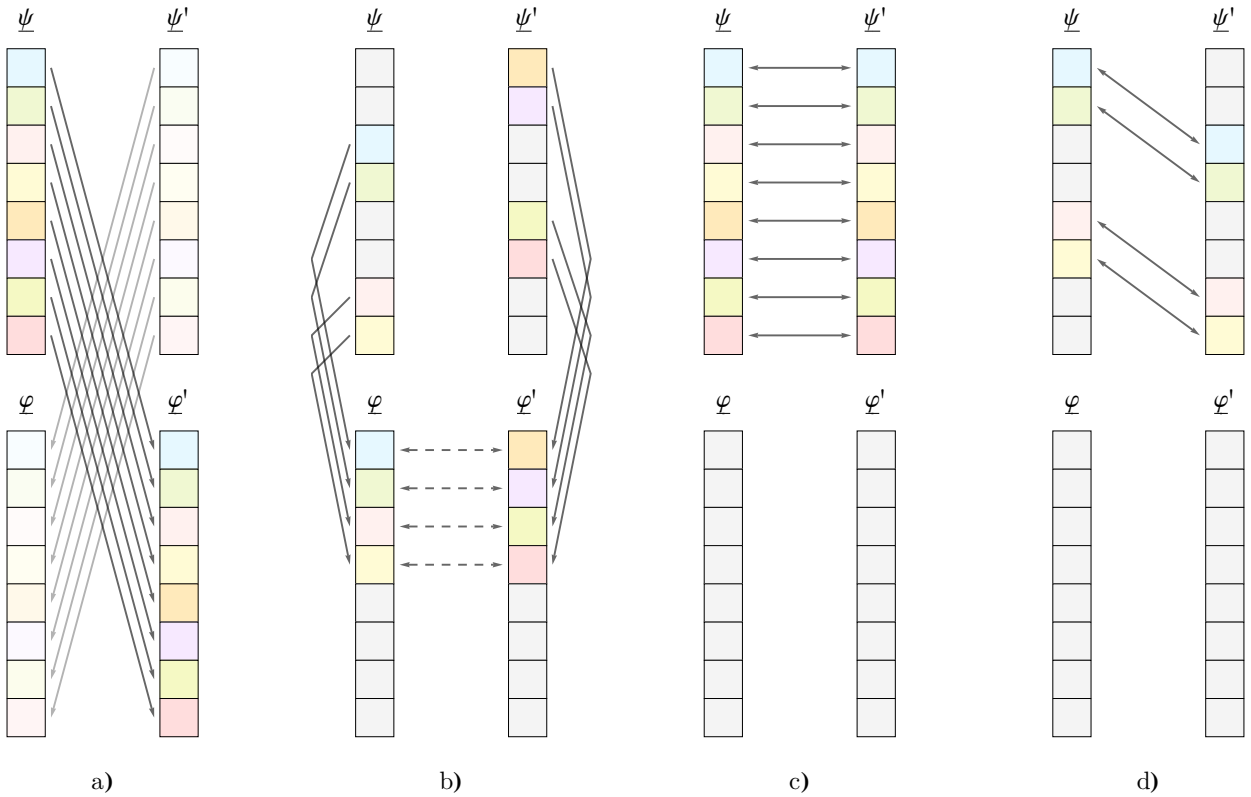


Figure 6.12: Methods of distributed amplitude exchange between pairs of nodes, used in the author's distributed algorithms for simulating quantum statevectors. Boxes represent amplitudes in arrays $\vec{\psi}$ and $\vec{\varphi}$, which within the same column, are the statevector partition and communication buffer (respectively) of a single node; $\vec{\psi}'$ and $\vec{\varphi}'$ are the partition and buffer of a communicating pair node. Arrows indicate the transfer of amplitudes between nodes and/or arrays (also guided by colour). **a)** demonstrates the most common pattern in this manuscript, where each node sends (via message passing) its full statevector partition into its pair node's buffer, so that the communicated amplitudes can be modified or rearranged before modifying the state. **b)** shows a less frequent scenario whereby each node first "packs" a (possibly disjoint) subset of its statevector amplitudes into a contiguous subarray of its communication buffer, before the buffers are exchanged through message passing (indicated by dashed double-ended arrows). Thereafter, nodes locally update their statevector partitions by modification and/or reordering of the buffer amplitudes. This scheme is favourable over **a)** when the runtime penalties of local memory copying and cache missing are outweighed by the benefit of communicating fewer total amplitudes and/or messages. **c)** shows an infrequent scenario whereby nodes can directly exchange their statevector amplitudes through a single message pass. This is only possible when non-local amplitudes require no modification or reordering, and appears as a behaviour of only a subset of nodes during a given distributed algorithm. **d)** shows a rare scenario where nodes can directly exchange a subset of their statevector amplitudes, at possibly offset indices. This requires multiple message passes (at least one for each contiguous stride of amplitudes), and is ideal when the size of each is similar to (or exceeds) the maximum (as constrained by the `MPI` implementation, or by the network).

Controlled gates

For instance, we saw in Section 6.3.2 (subsection Controlled gate) that the introduction of control qubits to a gate shrinks simulation costs. Indeed, the controlled single-qubit unitary reduces the total number of amplitudes needing communication from Algorithm 6.7, and often precludes the need for some nodes to communicate at all. While we do not derive the prescription, we visualise the resulting communication in Figure 6.13.

Pauli gadget

As another example, consider the n-qubit Pauli gadget of the form

$$\hat{U}_{\vec{t}}(\theta) = \exp\left(\mathrm{i}\,\theta \bigotimes_{j}^{n} \hat{\sigma}_{t_j}\right), \qquad \text{(re-expression of 1.21)}$$

which has appeared in every chapter of this thesis and is ubiquitous in variational quantum algorithms. While this might appear a complicated $2^n \times 2^n$ unitary, it can be expressed in Euler form [293] as

$$\hat{U}_{\vec{t}}(\theta) \equiv \cos(\theta)\mathbb{1}^{\otimes n} + \mathrm{i}\sin(\theta) \bigotimes_{j}^{n} \hat{\sigma}_{t_j}, \tag{6.20}$$

where the Pauli tensor upon an N-qubit basis state $|i\rangle$ can be shown to produce

$$\bigotimes_{j}^{n} \hat{\sigma}_{t_j} |i\rangle = (-1)^{p(i)} \ \mathrm{i}^{\dim(\vec{t}^{(Y)})} \ |i_{\neg\vec{t}^{(X,Y)}}\rangle\,. \tag{6.21}$$

Here, $\vec{t}^{(Y)}$ is the subset of the qubits $\vec{t}$ targeted by $\hat{Y}$ operators ($\vec{t}^{(X,Y)}$ is the subset targeted by $\hat{X}$ *or* $\hat{Y}$), $i_{\neg\vec{k}}$ is the integer i with its k-th bit flipped ($\forall k \in \vec{k}$), and $p(i)$ is

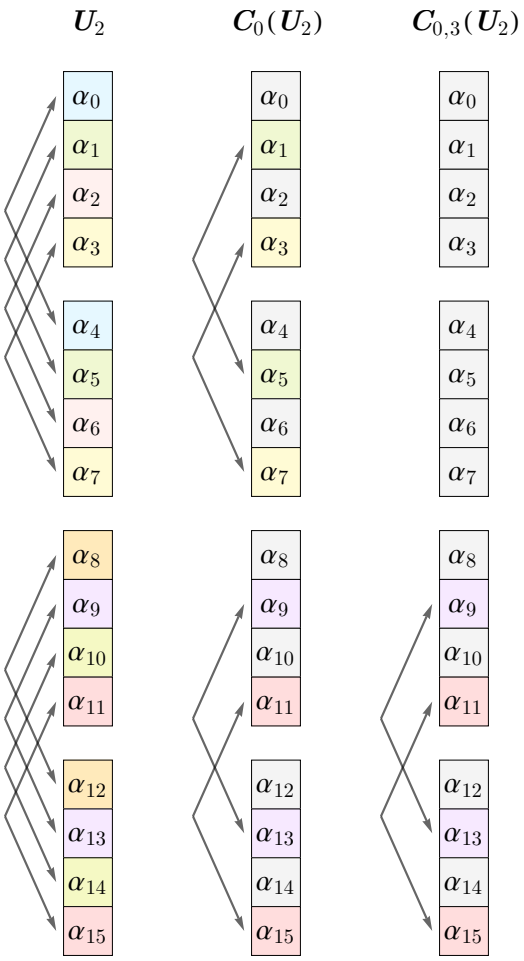


Figure 6.13: The pattern of amplitude combination during simulation of a controlled single-target unitary $C_{\vec{c}}[\hat{U}_t]$ upon a 4-qubit statevector $\sum_i \alpha_i |i\rangle$ distributed between four nodes. Amplitudes which undergo no modification (and hence need not be exchanged over the network) are shown in grey.

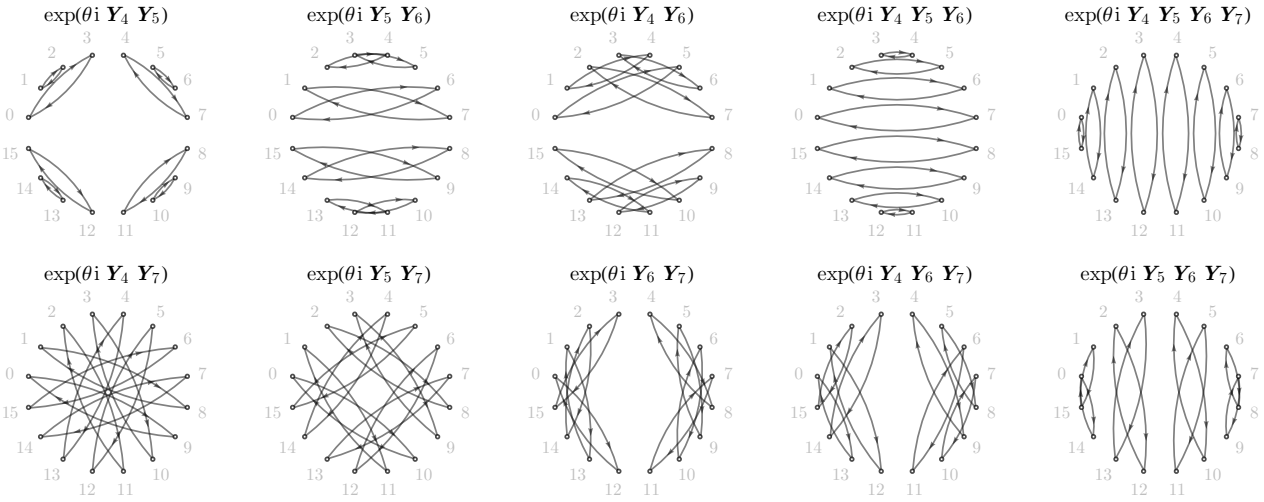


Figure 6.14: Communication patterns of multi-qubit Pauli gadgets $\exp(\theta\,\mathrm{i}\bigotimes_j \hat{\sigma}_{t_j})$ upon an $(N = 8)$-qubit statevector (with 256 amplitudes) distributed between $2^k = 16$ nodes. Simulation is embarrassingly parallel whenever $\max_j(t'_j) < N-k$, where t'_j are the target indices of only the $\hat{X}$ and $\hat{Y}$ operators; $\mathbb{1}$ and $\hat{Z}$ operators do not affect communication. Otherwise, simulation requires pairwise communication, as pictured.

defined as $p(i) = \dim(\{q \in \vec{t}^{(Y,Z)} : b_q(i) = 1\})$. These quantities arise from the -1 and i scalars in the $\hat{Z}$ and $\hat{Y}$ matrices. Ergo, the Pauli tensor alone would modify amplitude α_i of the full statevector $|\psi\rangle = \sum_i \alpha_i |i\rangle$ by

$$\alpha_i \;\to\; \beta_{i_{\neg\vec{t}(X,Y)}}\; \alpha_{i_{\neg\vec{t}(X,Y)}}, \qquad \text{where} \qquad \beta_i = (-1)^{p(i)}\;\mathrm{i}^{\dim(\vec{t}^{(Y)})} \tag{6.22}$$

which remarkably, permits integer and bitwise evaluation (since $\beta_i \in \{\pm 1, \pm\mathrm{i}\}$), and can be shown to require at most a single round of pairwise statevector exchange. Equation 6.20 informs us that the full Pauli gadget similarly prescribes

$$\alpha_i \;\to\; \cos(\theta)\,\alpha_i \;+\; \mathrm{i}\,\sin(\theta)\;\beta_{i_{\neg\vec{t}(X,Y)}}\;\alpha_{i_{\neg\vec{t}(X,Y)}}, \tag{6.23}$$

and is in fact *cheaper* to simulate than the single-qubit general gate. The emerging communication patterns are shown in Figure 6.14.

SWAP gate

Next, consider the humble SWAP gate

$$\text{SWAP} = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{pmatrix}. \tag{6.24}$$

In principle, we could naively leverage the known equivalence

$$\text{SWAP}_{t_1,t_2} \equiv C_{t_1}[\hat{X}_{t_2}]\, C_{t_2}[\hat{X}_{t_1}]\, C_{t_1}[\hat{X}_{t_2}] \tag{6.25}$$

to simulate SWAP through three invocations of a controlled single-qubit gate, $\hat{U} = \hat{X}$. Depending on the target qubits t_1 and t_2, this could invoke as many as three serial rounds of pairwise communication, and a total of $3 \times 2^{N-1}$ amplitude modifications. But a superior strategy is possible. Without loss of generality, we represent an N-qubit statevector as

$$\begin{aligned} |\psi\rangle = & \sum_{x}^{2^{N-t_2-1}} \sum_{y}^{2^{t_2-t_1-1}} \sum_{z}^{2^{t_1}} \\ & \alpha_{xyz} |x\rangle |0\rangle |y\rangle |0\rangle |z\rangle + \beta_{xyz} |x\rangle |0\rangle |y\rangle |1\rangle |z\rangle + \\ & \gamma_{xyz} |x\rangle |1\rangle |y\rangle |0\rangle |z\rangle + \lambda_{xyz} |x\rangle |1\rangle |y\rangle |1\rangle |z\rangle , \end{aligned} \tag{6.26}$$

where α_{xyz}, β_{xyz}, γ_{xyz} and λ_{xyz} together make up the 2^N amplitudes of $|\psi\rangle$ in the computational basis. This makes clear that the SWAP operation, effecting:

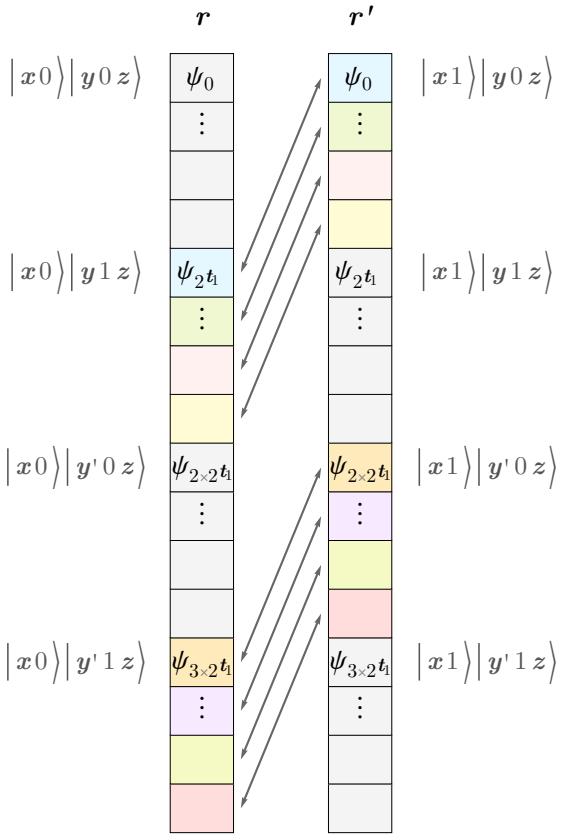


Figure 6.15: One of three communication patterns (occurring when $t_1 < N - k \leq t_2$) invoked by the distributed SWAP gate between qubits $t_2 > t_1 \geq 0$. Shown are the amplitudes among two paired nodes of ranks r (where the prefix t_2 qubit is 0) and r' (where t_2 is 1). $\psi_i \equiv \vec{\psi}[i]$ indicates the i-th local statevector amplitude, and the kets on either side indicate the corresponding global basis state of the amplitude (where x, y, z refer to the substates of Equation 6.26). Observe that there are $2^{N-k}/2^{t_1+1}$ sequences of 2^{t_1} contiguous amplitudes (and each, spaced by 2^{t_1} amplitudes) which are mapped contiguously to the pair node's statevector, albeit offset by $\pm 2^{t_1}$.

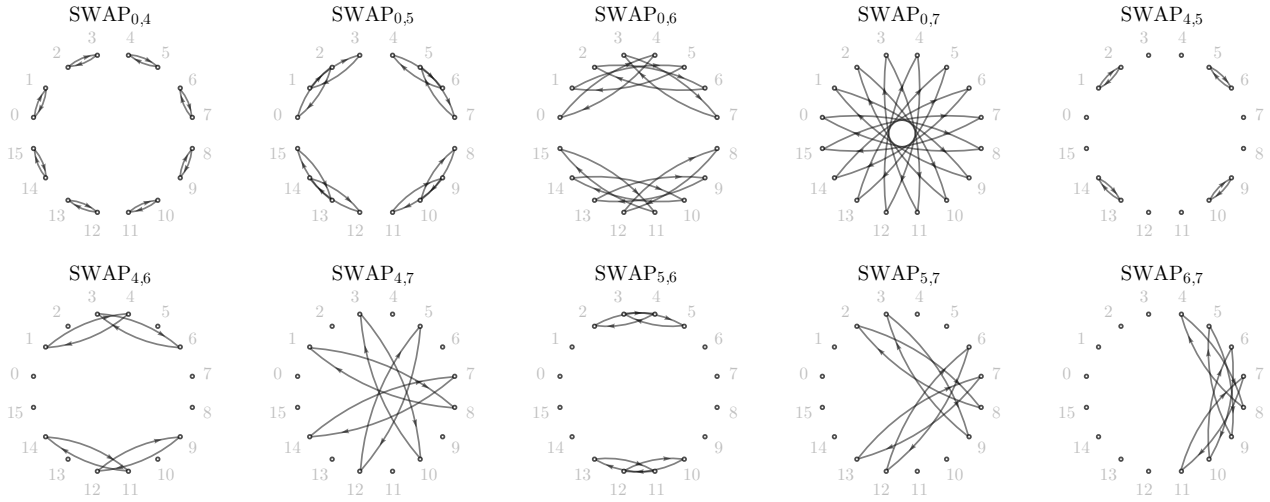


Figure 6.16: Communication patterns of SWAP_{t_1,t_2} gates upon an $(N = 8)$-qubit statevector (with 256 amplitudes) distributed between $2^k = 16$ nodes. Simulation is embarrassingly parallel whenever $\max(t_1, t_2) < N - k$, and otherwise requires pairwise communication between a possible subset of nodes.

$$\text{SWAP } |\psi\rangle = \sum_{x}^{2^{N-t_2-1}} \sum_{y}^{2^{t_2-t_1-1}} \sum_{z}^{2^{t_1}} \tag{6.27}$$

$$\alpha_{xyz} |x\rangle |0\rangle |y\rangle |0\rangle |z\rangle + \gamma_{xyz} |x\rangle |0\rangle |y\rangle |1\rangle |z\rangle +$$

$$\beta_{xyz} |x\rangle |1\rangle |y\rangle |0\rangle |z\rangle + \lambda_{xyz} |x\rangle |1\rangle |y\rangle |1\rangle |z\rangle ,$$

merely swaps the amplitudes $\beta_{xyz} \leftrightarrow \gamma_{xyz}$ (of states $|x\ 0\ y\ 1\ z\rangle$ and $|x\ 1\ y\ 0\ z\rangle$) and leaves half of the amplitudes unchanged. The number of qubits in these kets informs the relative positions of these amplitudes in the full statevector. That is, β_{xyz} is situated 2^{t_1} elements beyond α_{xyz}, and γ_{xyz} is situated 2^{t_2} elements beyond α_{xyz}. Therefore, γ_{xyz} is $2^{t_2} - 2^{t_1}$ positions ahead of its pair-amplitude β_{xyz}, independent of x, y and z.

In distributed simulation, swapping β_{xyz} and γ_{xyz} can be shown to invoke one of *three* distinct communication patterns, each with at most *one* round of pairwise statevector exchange compatible with our distributed constraints, and at most 2^{N-k-1} amplitudes

sent per-node. The amplitude modification of the more complicated of these scenarios is shown in Figure 6.15, which can be effected through the communication pattern **(d)** of Figure 6.12. Consequently, distributed simulation of the SWAP gate is *cheaper* than a one-qubit general gate; we illustrate its emerging communication patterns in Figure 6.16. It will prove an important utility to effect other operations.

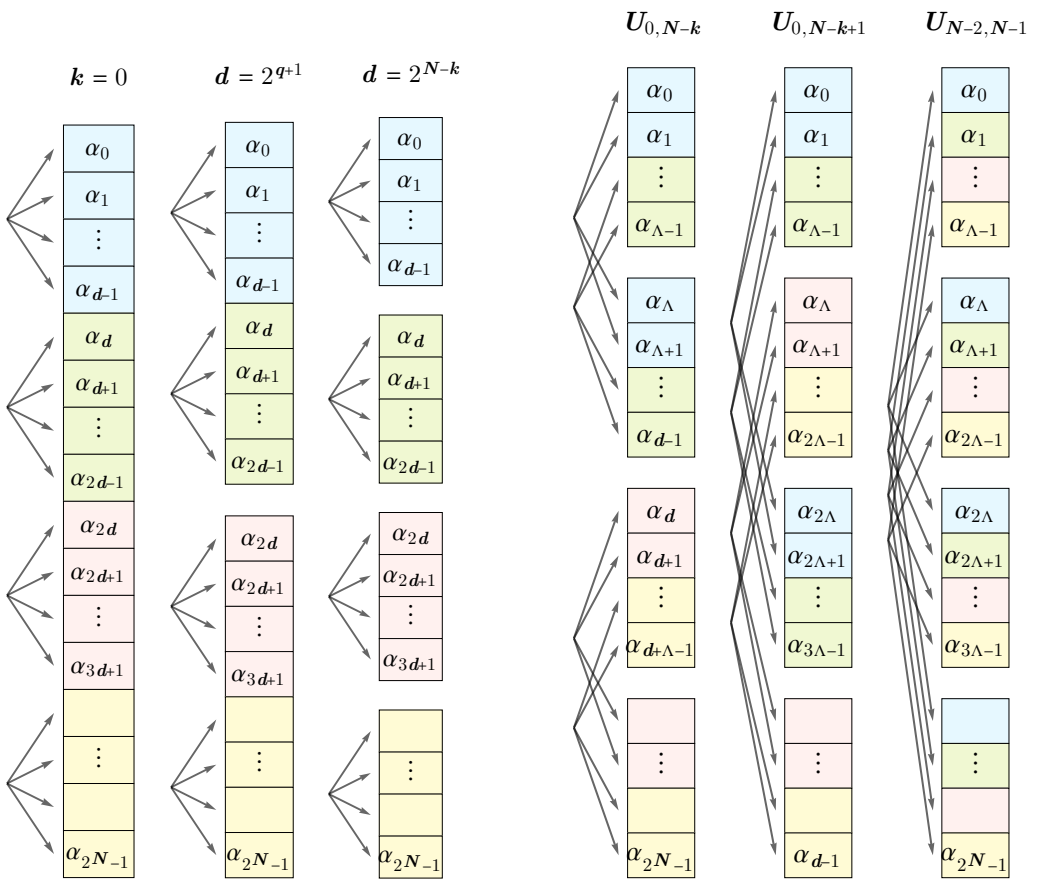
Many-qubit gate

Section 6.4.2 derived Algorithm 6.4 to simulate the n-qubit general unitary in local, multithreaded shared-memory settings. Each individual amplitude became a weighted sum of 2^n amplitudes. With the amplitudes of an N-qubit statevector distributed between 2^k nodes, the n-qubit unitary appears generally impossible since it may (in the worst case) require each node to simultaneously store and exchange $2^{N-k} \times 2^n$ amplitudes, greatly exceeding the 2^{N-k} communication buffer. Only when the gate targets qubits strictly within the subset $\{0, \ldots, N-k)$, can it be simulated embarrassingly parallel. We illustrate this in Figure 6.17.

However, by invoking an additional $\mathcal{O}(n)$ sequence of pairwise-communicating SWAP gates, we can effect *any* n-qubit (though where $n < N-k$) general unitary targeting *any* subset of qubits $\vec{t}$, exploiting that

$$\hat{U}_{\vec{t}}\Big|_{t_j \geq N-k \; \forall j} \equiv \left(\bigotimes_j^n \text{SWAP}_{j,t_j} \right) \hat{U}_{\{0,\,\ldots,\,n-1\}} \left(\bigotimes_j^n \text{SWAP}_{j,t_j} \right). \tag{6.28}$$

The limit $n < N-k$ arises from the 2^{N-k} buffer capacity. We demonstrate the simulation of a 4-qubit general gate via this decomposition in Figure 6.18.



(a) $\hat{U}_{\{0,\ldots,q\}}$ where $q < N - k$ (b) $\hat{U}_{\{t_1,t_2\}}$

Figure 6.17: Amplitude modification under a multi-qubit unitary $\hat{U}$ upon an N-qubit statevector distributed between $2^k \in \{1, 2, 4\}$ nodes (i.e. $\Lambda = 2^{N-k}$ amplitudes per-node). We define $d = 2^{q+1}$ as the number of contiguous amplitudes mixed by the unitary (q is the largest targeted qubit). The left graphic shows that targeting the first $N-k$ qubits induces no communication. The right graphic shows that *any* multi-qubit gate (even *two* qubit) which includes a single qubit $\geq N-k$ will trigger communication. When two or more qubits are $\geq N-k$, direct simulation would require larger communication buffers than permitted by our distributed scheme (in the right-most column, $\Lambda(2^k - 1) \geq \Lambda$).

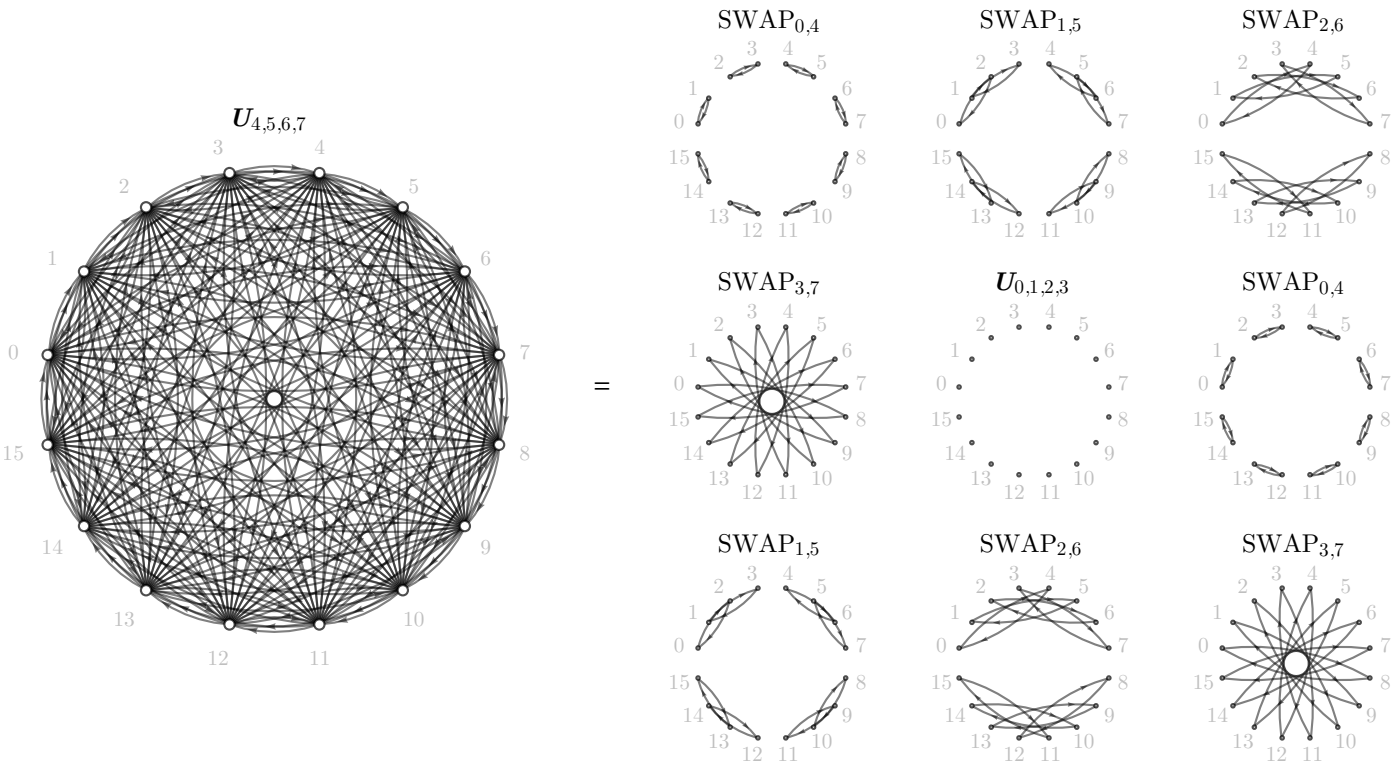


Figure 6.18: Decomposition of a 4-qubit general unitary acting upon the upper qubits of an $(N = 8)$-qubit statevector (with 256 amplitudes) distributed between $2^k = 16$ nodes, into a sequence of SWAP gates and a unitary upon the lower qubits. While direct in-place $\hat{U}_{4,5,6,7}$ simulation would require all-to-all communication with incompatibly large communication buffers, each gate of the decomposition is either embarrassingly parallel or prescribes pairwise communication compatible with our buffers.

6.7 Algorithms

Until now, this chapter presented (but mostly, alluded to) a variety of platform-specific high-performance algorithms for simulating statevectors. We conclude with a few examples of novel *platform-agnostic* algorithms, which repurpose relatively simple existing facilities (like those just demonstrated) to simulate more advanced states and operations. We retain our local and distributed performance requirements.

6.7.1 Density matrices

Every algorithm so far presented in this thesis has operated upon statevectors, as introduced in Section 1.4.1. A normalised statevector can only represent noise-free pure states, although we have seen their bespoke utilisation to study noisy systems, like in Section 2.6.2. General and precise study of decohering mixed states however requires a numerical implementation of the density matrices introduced in Section 1.4.2.

It is obvious that the matrix description of a unitary operator $\hat{U}$ modifying a statevector $|\psi\rangle \to \hat{U}\,|\psi\rangle$ is unchanged when modelling the same operation upon an equivalent sized density matrix $\rho \to \hat{U}\rho\,\hat{U}^\dagger$. However, efficient evaluation of the left-multiplying matrix, to leverage that $\hat{U}^\dagger$ is a tensor product of mostly identities (Equation 1.9), is *not* obvious. We would need to develop a new data structure encoding $\rho : \mathbb{C}^{2^N \times 2^N}$ which permits partitioning in distributed settings, and admits tractable communication when operated upon. Such a radical change to the fundamental state type is likely a prohibitive core refactor in an established software project.

Fortunately, a clever choice of representation enables us to leverage all the existing statevector simulation facilities and repurpose them to simulate density matrices. Consider

a general N-qubit density matrix

$$\rho = \sum_{i=0}^{2^N-1} \sum_{j=0}^{2^N-1} \alpha_{ij} \, |i\rangle \langle j|_N \,, \quad \alpha_{ij} \in \mathbb{C}. \tag{6.29}$$

Let $\|\rho\rangle\rangle : \mathbb{C}^{1\times 2^{2N}}$ denote an *unnormalised* $2N$-qubit statevector formed by concatenating the 2^N columns of amplitudes of density matrix ρ;

$$\|\rho\rangle\rangle = \sum_{n=0}^{2^{2N}-1} \alpha_{n\%2^N, \lfloor n/2^N \rfloor} \; |n\rangle_{2N} \,. \tag{6.30}$$

Initialising such a structure into pure states is easy,

$$|+\rangle \langle +|^{\otimes N} = \frac{1}{2^N} \sum_{i}^{2^N} \sum_{j}^{2^N} |i\rangle \langle j| \qquad \cong \qquad \frac{1}{2^N} \sum_{i}^{2^{2N}} |i\rangle \,, \tag{6.31}$$

as is, as remarkably prescribed under the Choi–Jamiolkowski isomorphism [248], effecting a unitary $\hat{U} : \mathbb{C}^{2^N \times 2^N}$;

$$\hat{U} \rho \hat{U}^\dagger \qquad \cong \qquad \left(\hat{U}^* \otimes \mathbb{1}^{\otimes N}\right) \left(\mathbb{1}^{\otimes N} \otimes \hat{U}\right) \|\rho\rangle\rangle \,. \tag{6.32}$$

Notice that $\left(\mathbb{1}^{\otimes N} \otimes \hat{U}\right) \|\rho\rangle\rangle$ is merely $\hat{U}$ effected upon $\|\rho\rangle\rangle$ in the typical manner of a gate upon the lower N qubits of a $2N$-qubit statevector. Then, $\left(\hat{U}^* \otimes \mathbb{1}^{\otimes N}\right)$ merely effects the complex conjugation of $\hat{U}$ upon the upper N qubits; we can effect it by simply adding N to the index of each target qubit and simulating it as general gate (conjugated) upon a statevector. Controlled gates are similarly simple; the additional elements of a controlled gate matrix are all real, so we can simply add N to each control qubit too. We illustrate this connection in Figure 6.19.

This reveals a strategy to leverage existing statevector simulators, even distributed

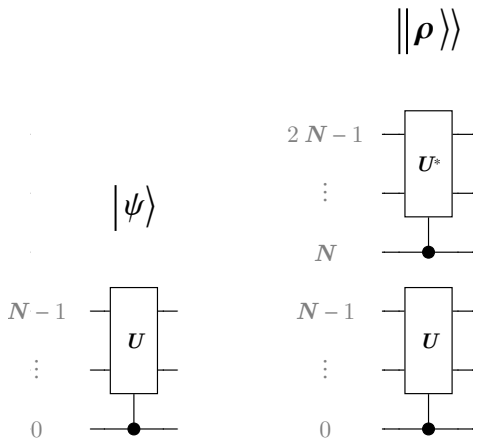


Figure 6.19: Prescription of a full-Hilbert general unitary upon an N-qubit statevector $|\psi\rangle$ (left) and an N-qubit density matrix ρ (right) which is represented as an unnormalised $2N$-qubit vector $\|\rho\rangle\rangle$.

ones, for the unitary simulation of N-qubit density matrices. In a sense, we pretend they are statevectors of $2N$ qubits, where we restrict the logical application of gates to the first N qubits. Such a technique is compatible with every unitary simulation algorithm presented in this thesis.

6.7.2 Channels

The main utility of the density matrix formalism is the description of mixing operators like noise channels. These have no statevector analogue and require new algorithms, likely specific to the computing platform. While the author has developed and/or contributed to 10 novel channel simulations (each with distinct multithreaded, GPU-accelerated and distributed paradigms), a comprehensive derivation of even a single channel algorithm exceeds the space here available. We instead briefly sketch two examples.

Dephasing

Recall that the dephasing channel describes a $\hat{Z}$ error occurring on qubit $0 \leq q < N$ with probability λ;

$$\mathcal{D}_{\phi_q}(\rho) = (1-\lambda)\rho + \lambda\, \hat{Z}_q\, \rho\, \hat{Z}_q. \qquad \text{(re-expression of 1.40)}$$

Upon a general state $\rho = \sum_{ij} \alpha_{ij} \left|i\right\rangle \left\langle j\right|$, this can be shown to modify amplitudes via

$$\alpha_{ij} \to \begin{cases} \alpha_{ij}, & i_{[q]} = j_{[q]} \\ \alpha_{ij}(1-2\lambda), & i_{[q]} \neq j_{[q]}, \end{cases} \tag{6.33}$$

suggesting a simple, bitwise, diagonal (no overlapping of process data) and embarrassingly parallel (when distributed) algorithm to simulate $\mathcal{D}_{\phi_q}(\rho) \cong \mathcal{D}_{\phi_q}(\|\rho\rangle\!\rangle)$.

The two-qubit dephasing channel is no more complicated, enabling a similar surprisingly simple embarrassingly-parallel simulation:

$$\mathcal{D}_{\Delta_{q_1,q_2}}(\rho) = (1-\lambda)\rho + \frac{\lambda}{3}\begin{pmatrix} \hat{Z}_{q_1}\rho\hat{Z}_{q_1} + \hat{Z}_{q_2}\rho\hat{Z}_{q_2} \\ + \hat{Z}_{q_1}\hat{Z}_{q_2}\rho\hat{Z}_{q_1}\hat{Z}_{q_2} \end{pmatrix}, \qquad \text{(re-iteration of 1.43)}$$

$$\implies \quad \alpha_{ij} \to \begin{cases} \alpha_{ij}, & i_{[q_1]} = j_{[q_1]} \ \wedge \ i_{[q_2]} = j_{[q_2]} \\ \alpha_{ij}\left(1 - \frac{4}{3}\lambda\right), & \text{otherwise.} \end{cases} \tag{6.34}$$

Depolarising

The one-qubit uniform depolarising channel which induces an erroneous Pauli gate upon qubit $q \geq 0$ with probability λ can be expressed as

$$\mathcal{D}_{\Delta q}(\rho) = (1-\lambda)\rho + \frac{\lambda}{3}\left(\hat{X}_q\rho\hat{X}_q + \hat{Y}_q\rho\hat{Y}_q + \hat{Z}_q\rho\hat{Z}_q\right). \qquad \text{(re-expression of 1.42)}$$

The action upon a general state $\rho = \sum_{ij} \alpha_{ij} |i\rangle \langle j|$ can be shown to induce

$$\alpha_{ij} \to \begin{cases} \alpha_{ij}\left(1 - \frac{2\lambda}{3}\right), & i_{[q]} = j_{[q]} \\ \alpha_{ij}\left(1 - \frac{4\lambda}{3}\right) + \frac{2\lambda}{3}\alpha_{i_{\neg q}, j_{\neg q}}, & i_{[q]} \neq j_{[q]} \end{cases} \tag{6.35}$$

which, upon an equivalent unnormalised $2N$-qubit statevector $\|\rho\rangle\!\rangle = \sum_x^{2^{2N}} \beta_x \|x\rangle\!\rangle$, induces

$$\beta_x \to \begin{cases} \beta_x\left(1 - \frac{2\lambda}{3}\right), & x_{[q]} = x_{[q+N]}, \\ \beta_x\left(1 - \frac{4\lambda}{3}\right) + \frac{2\lambda}{3}\beta_{x_{\neg q \neg (q+N)}}, & \text{otherwise.} \end{cases} \tag{6.36}$$

In distributed simulation, whereby the 2^{2N} amplitudes are uniformly partitioned between 2^k nodes, this fortunately admits pairwise communication with no larger a cost than that to simulate the one-qubit gate on a $2N$-qubit statevector (though through *three* distinct communication patterns). One can also show that a *two*-qubit depolarising channel upon qubits q_1 and q_2,

$$\mathcal{D}_{\Delta q_1, q_2}(\rho) = \left(1 - \frac{16\lambda}{15}\right)\rho + \frac{\lambda}{15}\sum_{\substack{\hat{\sigma},\hat{\sigma}' \in \\ \{\mathbb{1},X,Y,Z\}}} \hat{\sigma}'_{q_2}\hat{\sigma}_{q_1}\ \rho\ \hat{\sigma}'_{q_2}\hat{\sigma}_{q_1}, \qquad \text{(re-iteration of 1.44)}$$

induces an amplitude modification of

$$\beta_x \to \beta_x \left(1 - \frac{16\lambda}{15}\right) + \frac{4\lambda}{15} \times \begin{cases} \beta_x & \\ & x_{[q_1]} = x_{[q_1+N]} \wedge x_{[q_2]} = x_{[q_2+N]} \\ \beta_{x_{\neg q_1 \neg (q_1+N)}} & \\ & x_{[q_1]} \neq x_{[q_1+N]} \wedge x_{[q_2]} = x_{[q_2+N]} \\ \beta_{x_{\neg q_2 \neg (q_2+N)}} & \\ & x_{[q_1]} = x_{[q_1+N]} \wedge x_{[q_2]} \neq x_{[q_2+N]} \\ \beta_{x_{\neg q_1 \neg q_2 \neg (q_1+N) \neg (q_2+N)}} & \\ & x_{[q_1]} \neq x_{[q_1+N]} \wedge x_{[q_2]} \neq x_{[q_2+N]}, \end{cases} \tag{6.37}$$

which admits *five* unique communication patterns, the most expensive of which can be effected with *three* rounds of pairwise communication. We visualise this in Figure 6.20, which also includes the interesting and tractable one-way communication of the amplitude damping channel, which we do not explicitly derive here.

6.7.3 Mixed state energy

Mixed states are only worth simulating if we can extract information from them such as probabilities and expectation values. The expected energy under a Hamiltonian in the Pauli basis,

$$\hat{H} = \sum_i^T h_i \bigotimes_j^N \hat{\sigma}_j^{(i)}, \quad h_i \in \mathbb{R}, \quad \hat{\sigma}_j^{(i)} \in \{\hat{X}, \hat{Y}, \hat{Z}, \mathbb{1}\}, \tag{6.38}$$

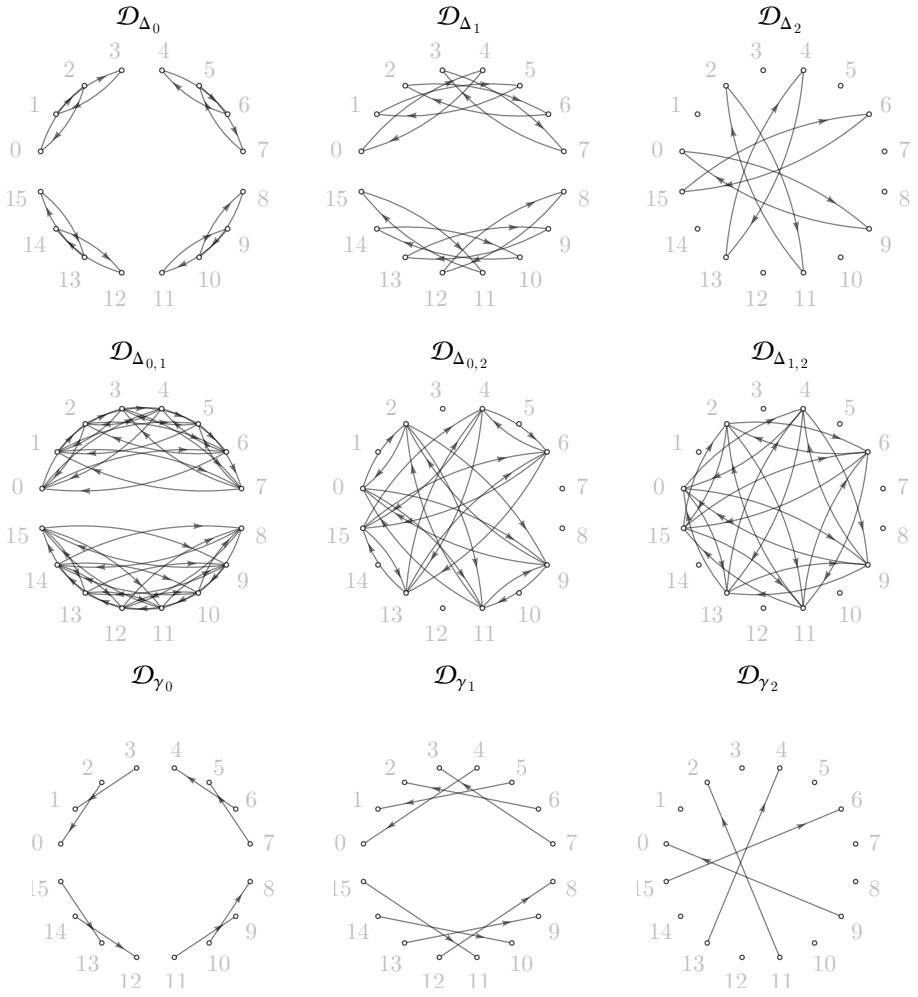


Figure 6.20: Communication during 16-node distributed simulation of one-qubit depolarising (top row), two-qubit depolarising (middle row), and one-qubit amplitude damping (bottom row) channels upon the subscripted qubits of a 3-qubit density matrix. Interestingly, two-qubit depolarising can be performed with at most *three* rounds of pairwise exchange, and damping requires only one-way communication.

has been a staple of this thesis. To classically calculate it for a mixed state ρ, one may be tempted to utilise that

$$\langle E \rangle = \text{trace}\left(\hat{H}\,\rho\right) \tag{6.39}$$

$$= \sum_{i}^{T} h_i \,\text{trace}\left(\bigotimes_{j}^{N} \hat{\sigma}_j^{(i)}\,\rho\right) \tag{6.40}$$

$$= \sum_{i}^{T} h_i \sum_{x=0}^{2^N-1} \left(\mathbb{1}^{\otimes N} \otimes \bigotimes_{j}^{N} \hat{\sigma}_j^{(i)}\; \|\rho\rangle\!\rangle\right)_{x\,(2^N+1)} \tag{6.41}$$

and evaluate each of T traces (as a partial sum) after applying a sequence of N one-qubit Pauli gates via Algorithm 6.2 (when local) or Algorithm 6.6 (when distributed) upon the $2N$-qubit vector $\|\rho\rangle\!\rangle$. In both deployments, this would gratuitously require $\mathcal{O}(TN2^{2N})$ floating-point operations, $\mathcal{O}(TN2^{2N})$ amplitude modifications (and the associated caching costs) and when distributed between 2^k nodes, a total of Tk rounds of pairwise communication. It would also either necessitate a $\mathcal{O}(2^{2N})$ memory overhead to restore $\|\rho\rangle\!\rangle$ after the Pauli operators, or double all runtime costs by exploiting Pauli idempotency.

Yet, an alternate calculation is possible in $\mathcal{O}(TN2^{2N})$ *integer* operations (and factor N fewer floating-point operators), $\mathcal{O}(2^{2N})$ amplitude modifications (the minimum), *no* memory overhead and *no* statevector exchange whatsoever when distributed. This seemingly impossible improvement is discovered by expanding the trace into

$$\text{trace}(\hat{H}\rho) = \sum_{k=0}^{2^N-1}\sum_{l=0}^{2^N-1}\left(\sum_{i}^{T} h_i \left[\bigotimes_{j}^{N} \hat{\sigma}_j^{(i)}\right]_{kl}\right)\rho_{lk} \tag{6.42}$$

and, utilising that each $\hat{\sigma}_j^{(i)}$ are square $d \times d$ matrices,

$$\left[\bigotimes_j^N \hat{\sigma}_j\right]_{kl} = \prod_{j=0}^{N-1} \left[\hat{\sigma}_{N-j}\right]_{\lfloor \frac{k}{d^j} \rfloor \% d,\, \lfloor \frac{l}{d^j} \rfloor \% d} \tag{6.43}$$

In fact, because $d = 2$ for the Pauli operators, this is equivalently

$$= \prod_{j=0}^{N-1} \left[\hat{\sigma}_{N-j}\right]_{k_{[j]},\, l_{[j]}} \quad \in \{\pm 1, \pm \mathrm{i}\}, \tag{6.44}$$

observing a natural 2D extension of our ket indexing. This reveals an $\mathcal{O}(N)$ bitwise-wise method to calculate a single element of an N-qubit tensor product of Pauli matrices. Hence, we can calculate a coefficient

$$\mu_{kl} = \sum_i^T h_i \left[\bigotimes_j^N \hat{\sigma}_j^{(i)}\right]_{kl} \tag{6.45}$$

in $\mathcal{O}(T)$ floating-point operations and $\mathcal{O}(NT)$ integer operations. The entire expected value is then

$$\mathrm{trace}(\hat{H}\rho) = \sum_k^{2^N} \sum_l^{2^N} \mu_{kl}\ \rho_{lk} \tag{6.46}$$

$$\equiv \sum_{n=0}^{2^{2N}-1} \mu_{\lfloor n/2^N \rfloor,\, n\%2^N}\ \ \left\|\rho\right\rangle\!\right\rangle_n . \tag{6.47}$$

Every 2^{2N} value of μ is independent and hence computable in parallel;

$$\mu(n) = \mu_{\lfloor n/2^N \rfloor,\, n\%2^N} = \sum_i^T h_i \left(\prod_{j=0}^{N-1} \left[\hat{\sigma}_{N-j}^{(i)}\right]_{n_{[N+j]},\, n_{[j]}}\right). \tag{6.48}$$

In distributed simulation, because our Pauli Hamiltonian description is present on every

node, the scheme is embarrassingly parallel before a final global reduction;

$$\text{trace}(\hat{H}\rho) = \sum_{r=0}^{2^k-1} \left(\sum_{m=0}^{2^{2N-k}-1} \mu(m + r\,2^{2N-k})\; \|\rho\rangle\!\rangle_{m+r\,2^{2N-k}} \right), \tag{6.49}$$

where everything in brackets is local to a node.

6.8 Discussion

This chapter featured only illustrative examples of the many algorithmic and implementation-level innovations developed through my doctorate. These include over 60 high-performance functions defined with *separate* multithreaded (with `OpenMP`), GPU-accelerated (with `CUDA`), and distributed (with `MPI`) backends, in addition to an exhausting level of other software facilities described in the next chapter. The distributed facilities in particular required substantial novel algorithmic development, which we attempt to succinctly summarise in Table 6.1. The impact of this chapter's research is illuminated in Chapter 7.

Table 6.1: Asymptotic performance of the novel distributed algorithms developed during the course of the research in this thesis. Columns Bops and Flops give the worst-case total (aggregated between all nodes) number of bitwise and floating-point operations (respectively). Comm. rounds is the number of serial rounds of pairwise communication invoked by the algorithm, and Comm. amps is the total number of amplitudes communicated across the network. The Memory column gives the additional memory overhead needed by the algorithm, not including that of the input states (which are modified in-place) nor their permanent buffers. Note that while single-target $\hat{U}_t$ is conventional and included for comparison, all other algorithms are novel and asymptotically superior to the those reported in the statevector simulation literature.

(a) Upon an N-qubit statevector, whereby $|\psi\rangle \to \hat{U}\,|\psi\rangle$.

Operation	Description	Bops	Flops	Comm. rounds	Comm. amps	Memory
$\hat{U}_t$	single-target	$\mathcal{O}(2^N)$	$\mathcal{O}(2^N)$	0 or 1	2^N	$\mathcal{O}(1)$
$\hat{C}_c[\hat{U}_t]$	single-control	$\mathcal{O}(2^{N-1})$	$\mathcal{O}(2^{N-1})$	0 or 1	2^{N-1}	$\mathcal{O}(1)$
$\hat{C}_{\vec{c}}[\hat{U}_t]$	many-control	$\mathcal{O}(2^{N-\dim(\vec{c})})$	$\mathcal{O}(2^{N-\dim(\vec{c})})$	0 or 1	$2^{N-\dim(\vec{c})}$	$\mathcal{O}(1)$
$\exp(\mathrm{i}\theta\hat{Z}^{\otimes})$	phase gadget	$\mathcal{O}(N2^N)$	$\mathcal{O}(2^N)$	0	0	$\mathcal{O}(1)$
$\hat{\sigma}^{\otimes}$	Pauli tensor	$\mathcal{O}(N2^N)$	0	0 or 1	2^N	$\mathcal{O}(1)$
$\exp(\mathrm{i}\theta\hat{\sigma}^{\otimes})$	Pauli gadget	$\mathcal{O}(N2^N)$	$\mathcal{O}(2^N)$	0 or 1	2^N	$\mathcal{O}(1)$
SWAP	swap gate	2^{N-1}	0	0 or 1	2^{N-1}	$\mathcal{O}(1)$
$\hat{U}_{\vec{t}}$	many-target	$\mathcal{O}(N2^{N+\dim(\vec{t})})$	$\mathcal{O}(2^{N+\dim(\vec{t})})$	$\mathcal{O}(\dim(\vec{t}))$	$\mathcal{O}(\dim(\vec{t})\,2^N)$	$\mathcal{O}(2^{\dim(\vec{t})})$
QFT	quantum Fourier transform	$\mathcal{O}(N2^N)$	$\mathcal{O}(N2^N)$	$\log_2(\#\text{nodes})$	$\log_2(\#\text{nodes})\,2^N$	$\mathcal{O}(1)$

(b) Upon an N-qubit density matrix, whereby $\rho \to \hat{U}\rho\hat{U}^\dagger$ or $\mathcal{D}(\rho)$ or $\sum_i \hat{K}^{(i)}\rho\hat{K}^{(i)\dagger}$. Generally the unitary operations tabled in (a) can be effected on the equivalent-dimension density matrix with a worst-case complexities of 2^N times their statevector complexity, and require at most a single additional round of communication.

Operation	Description	Bops	Flops	Comm. rounds	Comm. amps	Memory
$\hat{U}_t$	single-target	$\mathcal{O}(2^{2N})$	$\mathcal{O}(2^{2N})$	0, 1 or 2	2^{2N}	$\mathcal{O}(1)$
$\hat{U}_{\vec{t}}$	many-target	$\mathcal{O}(N2^{2N+\dim(\vec{t})})$	$\mathcal{O}(2^{2N+\dim(\vec{t})})$	$\mathcal{O}(\dim(\vec{t}))$	$\mathcal{O}(\dim(\vec{t})\,2^{2N})$	$\mathcal{O}(2^{\dim(\vec{t})})$
$\mathcal{D}_{\phi_a}$	1-qubit dephasing	$\mathcal{O}(2^{2N-1})$	$\mathcal{O}(2^{2N-1})$	0	0	$\mathcal{O}(1)$
$\mathcal{D}_{\Delta a}$	1-qubit depolarising	$\mathcal{O}(2^{2N})$	$\mathcal{O}(2^{2N})$	0 or 1	2^{2N-1}	$\mathcal{O}(1)$
$\mathcal{D}_{\phi_{a,b}}$	2-qubit dephasing	$\mathcal{O}(2^{2N-2})$	$\mathcal{O}(2^{2N-2})$	0	0	$\mathcal{O}(1)$
$\mathcal{D}_{\Delta a,b}$	2-qubit depolarising	$\mathcal{O}(2^{2N})$	$\mathcal{O}(2^{2N})$	0, 1, 2 or 3	2^{2N-3}	$\mathcal{O}(1)$
$\mathcal{D}_{\gamma_a}$	1-qubit damping	$\mathcal{O}(2^{2N})$	$\mathcal{O}(2^{2N})$	0 or 1	2^{2N-2}	$\mathcal{O}(1)$
$\{\hat{K}^{(i)}_{\vec{t}} : i\}$	many-qubit Kraus map	$\mathcal{O}(N2^{2N+2\dim(\vec{t})})$	$\mathcal{O}(2^{2N+2\dim(\vec{t})})$	$\mathcal{O}(\dim(\vec{t}))$	$\mathcal{O}(\dim(\vec{t})\,2^{2N})$	$\mathcal{O}(2^{4\dim(\vec{t})})$
$\mathrm{tr}(\rho^2)$	purity	$\mathcal{O}(2^{2N})$	$\mathcal{O}(2^{2N})$	$\log_2(\#\text{nodes})$	$\mathcal{O}(\#\text{nodes}^2)$	$\mathcal{O}(1)$
$\mathrm{tr}(\sum_i^T h_i\hat{\sigma}_i^{\otimes}\rho)$	Pauli string expectation	$\mathcal{O}(NT2^{2N})$	$\mathcal{O}(T2^{2N})$	$\log_2(\#\text{nodes})$	$\mathcal{O}(\#\text{nodes}^2)$	$\mathcal{O}(1)$

Algorithm 6.7: An overloaded MPI implementation [294] of the statevector exchange methods visualised in Figure 6.12. Some require repeated but tractable calling of the below functions. The reader may assume $\vec{\psi}$ is a node's (2^{N-k})-amplitude partition of an N-qubit statevector, $\vec{\varphi}$ is its equal-sized communication buffer, and $0 \leq r' < 2^k$ is the rank of the paired node.

```
exchangeArrays(v⃗₁, i₁, v⃗₂, i₂, r′, n)
    MPI_Sendrecv(
        (address of) v⃗₁[i₁], n, MPI_COMPLEX,
        r′, MPI_ANY_TAG,
        (address of) v⃗₂[i₂], n, MPI_COMPLEX,
        r′, MPI_ANY_TAG,
        MPI_COMM_WORLD, status);

exchangeArrays(ψ⃗, φ⃗, r′)
    exchangeArrays(ψ⃗, 0, φ⃗, 0, r′, 2^(N−k))
```

Chapter 7

Classical Software Development

Contents

7.1 Foreword

This section presents the published works

- **T. Jones**, A. Brown, I. Bush, S. C. Benjamin
  QuEST and High Performance Simulation of Quantum Computers
  Scientific Reports **9**, 10736 (2019) (link)

- **T. Jones**, S. C. Benjamin
  QuESTlink – Mathematica embiggened by a hardware-optimised quantum emulator
  Quantum Sci. Technol. **5**, 3 (2020) (link)

in addition to the two subject software projects.

- **T. Jones**, A. Brown, S. C. Benjamin, I. Bush, B. Koczor, R. Meister, J. Wilkins, N. Vogt, D. Silcock, K. Chhabra, G. Struchalin, M. Prokop, C. J. Anders
  The Quantum Exact Simulation Toolkit
  DOI: 10.5281/zenodo.**6475156** (2017 - 2022)
  Hosted on Github `QuEST-Kit/QuEST` (link)

- **T. Jones**, B. Koczor
  QuESTlink
  DOI: 10.5281/zenodo.**6475200** (2018 - 2022)
  Hosted on Github `QTechTheory/QuESTlink` (link)

I wrote the first manuscript and performed all numerical work, benchmarking early prototypal QuEST code which was primarily written by my co-author Anna Brown under the algorithmic direction of Simon C. Benjamin and the supervision of Ian Bush.

I would later lead development of QuEST, graduating it to a complete software library. I then created QuESTlink and wrote the second manuscript. I humbly acknowledge that the title's reference to The Simpsons was the masterstroke of my coauthor. Where not explicitly stated otherwise, all content of this chapter is my original work.

7.2 Introduction

Algorithms and implementation optimisations to advance simulations of quantum computers, as presented in Chapter 6, are invaluable to the research community - but not in their abstract form. A quantum computer scientist should not be expected to have the necessary expertise in high-performance computing nor in scientific programming to implement these methods, nor the time. It is crucial that researchers instead have access to *software* which implements these schemes behind a usable interface in a familiar language. Ergo, most of the classical simulations of the previous chapters have been implemented into two software projects, the Quantum Exact Simulation Toolkit (QuEST) [249] and QuESTlink [112]. The process of developing and maintaining a software project is markedly different to those of devising algorithms and implementing them in scientific codes.

This chapter serves to document the development of QuEST and QuESTlink. It will showcase their software architectures, interface designs, and meta-program facilities, which are justified with well-known software development practices.

7.3 History

This section rapidly summarises the history of the projects, my contributions to them, and their engagement with the research community.

7.3.1 QuEST

QuEST is a HPC simulator of quantum computers, written in C and C++, which hybridises OpenMP, MPI and CUDA. Its founding design motivation addresses the researcher's need to change the scale of simulation (from few to tens of qubits) and correspondingly redeploy to better-suited computing platforms (from CPUs to GPUs and supercomputing networks), and avoids the tedium of switching libraries or adopting new programming languages.

QuEST began as a collection of local multithreaded and distributed codes, and separate GPU codes, written by my coauthor Anna Brown in March 2017, under algorithmic direction of Simon Benjamin. My contributions began in January 2018 and included example programs and benchmarking for the February whitepaper [249], and an overhaul of the documentation system to include maths and circuit diagrams. Through May 2018 to Dec 2021, I would make a significant number of substantial changes to QuEST which included; merging of the multithreaded, distributed and GPU codes into a single code-base and build system; refactoring the code and build for `C` and `C++` agnosticism and wide compiler compatibility; overhauling the software architecture; overloading all functions for agnostic density matrix support; adding a comprehensive user-input validation and error handling system; substantially developing the documentation; adding generation of quantum assembly language (QASM), an emerging standard for specifying gate-level quantum programs [295, 296]; adding a comprehensive automated unit-testing system; adding a collection of new types and data structures; adding low-level hooks for easier integration of QuEST into larger software stacks; adding a multitude of new high-performance simulation functions. In that time, I made 672 commits editing 796k lines of code which added 234 new functions and types, 124 unit tests, and a multitude of bug patches. I liaised with external contributors, maintained the Github repository,

and created the website quest.qtechtheory.org.

QuEST has received extensive engagement from the research community. As of April 2022, its Github repository has 270 stars and 95 forks, which include derivative projects by research companies Rahko, HQS, Riverlane and CQC, and proposals for integration into NASA's QuaSar stack. The Github repository has seen 169 active or closed issues, and 12 pull requests from external collaborators in the last year alone. QuEST is known to be used by companies including IBM Research UK, Tencent, Quantum Motion Ltd and Amazon Web Services, and was the subject of the final challenge in the Asian Supercomputing Competition (ASC2020). QuEST's whitepaper [249] has an estimated 132 citations, and was Scientific Report's 11th most downloaded *Physics* paper of 2019 (to date; 12k accesses). To the best of the author's knowledge, QuEST remains the only available simulator which supports deployment to all of GPUs, multi-CPU systems and distributed systems. It is furthermore the only publicly available distributed simulator of density matrices.

7.3.2 QuESTlink

Because QuEST prioritises performance, it is written and interfaced with `C` and `C++`. These are low-level languages which can be tedious to program, especially for scientific applications. A researcher typically seeks slower higher-level tools and resorts to languages like `C` only when encountering a performance barrier. Within my research group, this produced a workflow of developing *ad hoc* C codes for interfacing with QuEST, and analysing the output data in the high-level scientific language Mathematica.

In December 2018, the author created QuESTlink to invoke the high-performance QuEST utilities directly from within Mathematica, substantially shrinking program-

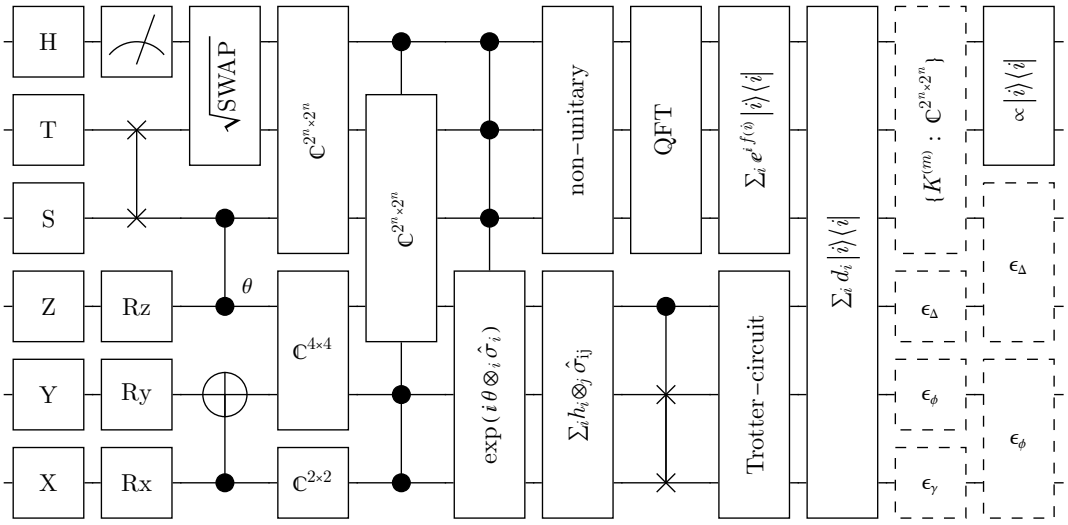
ming times and accelerating research. QuESTlink involves 5488 additional lines of `C++`, Mathematica and Wolfram templating code in software layers above QuEST. Since then, QuESTlink was invited for publication in the IOPscience Quantum Science and Technology issue on quantum software, and the resulting whitepaper [112] was advertised by the Wolfram Publication Watch Team. QuESTlink was also invited to the Wolfram collection of Quantum packages as published by Wolfram Media, and as a Staff Pick on the Wolfram Community Forum.

7.4 Facilities

This section summarises the facilities of QuEST and QuESTlink, the significant majority of which were implemented by the author. We will not provide a complete catalogue of their application programming interfaces (API), but will attempt to illustrate the utility and diversity of the libraries through some examples.

7.4.1 QuEST

QuEST boasts a wide range of powerful facilities between its ≈ 200 user-facing functions, agnostic to whether it is deployed in multithreaded, GPU or distributed applications. This includes simulation of statevectors and density matrices with customisable precision (using single, double or quadruple floating-point precision), starting from numerous initial states, under the actions of multi-controlled multi-target Pauli gadgets; multi-controlled multi-target general unitaries (specified element-wise); forced and random measurements; one- and two-qubit depolarising and dephasing channels; multi-qubit general Kraus maps; QFT and Trotterisation circuits. QuEST can also calculate;



$$\langle\psi_1|\psi_2\rangle \quad |\langle 0|\psi\rangle|^2 \quad \langle\psi|\Big(\sum_i h_i\otimes_j\hat{\sigma}_{ij}\Big)|\psi\rangle \quad \langle\psi|\Big(\sum_i d_i\,|i\rangle\langle i|\Big)|\psi\rangle$$

$$\{|\langle i|\psi\rangle|^2 \,:\, i<2^n\} \quad \sum_i |\langle i|\psi\rangle|^2 \quad \mathrm{tr}\left({\rho_1}^\dagger\rho_2\right) \quad \mathrm{tr}\left(|0\rangle\langle 0|\,\rho\right) \quad \langle\psi|\rho|\psi\rangle$$

$$\mathrm{tr}\big(\rho_1-\rho_2)(\rho_1-\rho_2)^\dagger\big) \quad \{\mathrm{tr}(|i\rangle\langle i|\,\rho) \,:\, i<2^n\} \quad \mathrm{tr}(\rho^2) \quad \mathrm{tr}(\rho)$$

$$\mathrm{tr}\Big(\rho\sum_i h_i\otimes_j\hat{\sigma}_{ij}\Big) \quad \mathrm{tr}\Big(\rho\sum_i d_i\,|i\rangle\langle i|\Big)$$

Figure 7.1: A visual summary of QuEST's core simulation facilities. Essentially all are also available in QuESTlink's wider API.

expectation values in a variety of observable basis including real-weighted Pauli sums and diagonal matrices; purities, fidelities, inner products and distances of states; and measurement probabilities. We summarise QuEST's core simulation facilities in Figure 7.1.

QuEST has a variety of additional non-simulation facilities for logging output, parsing files, generating data-structures (like diagonal operators) from other descriptions (like Ising Hamiltonians), and generating QASM [295, 296] programs. For example, the `C` code

```
startRecordingQASM(qureg);
initPlusState(qureg);
hadamard(qureg, 0);
controlledRotateY(qureg, 0, 1, 1.5);
ComplexMatrix2 m = {.real={{1,0},{0,0}}, .imag={{0,0},{0, 1}} };
unitary(qureg, 0, m);
printRecordedQASM(qureg);
```

produces QASM output

```
OPENQASM 2.0;
qreg q[10];
creg c[10];
// Initialising state |+>
reset q;
h q;
h q[0];
cRy(1.5) q[0],q[1];
U(0.78539816339745,0,0.78539816339745) q[0];
```

Because QuEST's API offers a significantly larger number of primitive operations than expressible in the QASM standard, many will log *multiple* QASM commands. For instance, the `C` code:

```
ComplexMatrix2 m = {.real={{1,0},{0,0}}, .imag={{0,0},{0, 1}} };
controlledUnitary(qureg, 2, 3, m);
multiStateControlledUnitary(qureg, (int[])0,1, (int[])0,1, 2, 3, m);
```

produces output

```
cU(0.78539816339745,0,0.78539816339745) q[2],q[3];
// Restoring the discarded global phase of the previous controlled unitary
Rz(0.78539816339745) q[3];
// NOTing some gates so that the subsequent unitary is controlled-on-0
x q[0];
ccU(0.78539816339745,0,0.78539816339745) q[0],q[1],q[3];
// Restoring the discarded global phase of the previous multicontrolled unitary
Rz(0.78539816339745) q[3];
// Undoing the NOTing of the controlled-on-0 qubits of the previous unitary
x q[0];
```

Notice the heavy use of comments. Because the QASM output cannot be comprehensive, it intends to communicate to the reader rather than another parser. When the QuEST operation cannot be decomposed into QASM primitives, the output will include only instructive comments. For example, attempting to produce QASM for a three-subregister diagonal operator with a custom exponential-polynomial phase function $\theta(x, y, z) = x^2 + 2y + 4y^{-1} - 3.14\sqrt{z}$, as simulated with `C` code

```
int qubits[] = {0,1, 3,4,5, 7};
int numQubitsPerReg[] = {2, 3, 1};
int numRegs = 3;
qreal coeffs[] = {1, 2, 4, -3.14};
qreal exponents[] = {2, 1, -1, 0.5};
int numTermsPerReg[] = {1, 2, 1};
applyMultiVarPhaseFunc(qureg, ...);
```

produces the following debug-friendly QASM output;

```
// Here, applyMultiVarPhaseFunc() multiplied a complex scalar of the form
//     exp(i (
//          + 1 x^2
//          + 2 y + 4 y^(-1)
//          - 3.14 z^0.5 ))
//   upon substates informed by qubits (under an unsigned binary encoding)
//     |x> = {0, 1}
//     |y> = {3, 4, 5}
//     |z> = {7}
```

7.4.2 QuESTlink

QuESTlink's founding facility is its ability to off-load the expensive numerical task of quantum emulation to the high-performing QuEST backend, from behind a high-level Mathematica interface. This backend includes most of QuEST's core functions, but can additionally utilise hardware *remote* to the user's Mathematica kernel. By repurposing Github's automated continuous integration (CI) system, we precompile the QuESTlink source code for several operating systems to offer rapid installation-free continuous deployment (CD). We will elaborate further on this in Section 7.6.2.

QuESTlink has grown to be much more than an interface to QuEST, and leverages Mathematica's powerful symbolic capabilities to offer: compact, symbolic specification of circuits akin to operator syntax in the quantum literature; plotting and diagramming of circuits, circuit topologies and density matrices; analytic calculation of circuits and Hamiltonians as symbolic matrices; expansions of operator expressions into the Pauli basis; calculation of variational circuit derivatives; automatic insertion of noise into a circuit or gate schedule according to a hardware specification. At present, QuESTlink has 69 functions with 15 optional arguments, and 22 symbolic operators.

Since we will next demonstrate some of these facilities, we first offer a quick review of

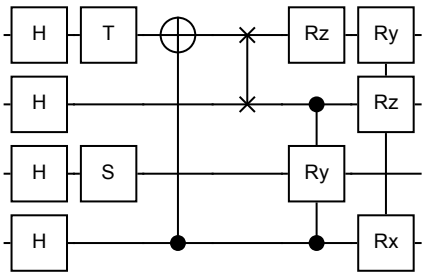
the relevant Mathematica syntax. Evaluation of function `f` with input `a` is denoted by `f[a]`, or with postfix notation as `a //f`. Expressions with a trailing `;` suppress their otherwise displayed result. Elements of a list `x = {a, b, c}` are accessed as `x[[i]]` where index `i` ≥ 1, and `x[[m;;n]]` returns the sublist spanning indices `m` to `n`. The shortcut `ex /.a -> b` replaces sub-expressions of `ex` which match pattern `a`, with `b`. In the subsequent code snippets, context should make clear what is *input* to the Mathematica notebook, and what is a rendered *output*.

QuESTlink can be deployed without any setup nor installation, by simply opening a Mathematica notebook or kernel and running:

```
Import["https://qtechtheory.org/questlink.m"];
CreateDownloadedQuESTEnv[];
```

We explain how this is possible in Section 7.6.2 (and in Figure 7.4). Specifying, diagramming and analytically evaluating circuits is concise.

```
c = Circuit[ H0 H1 H2 H3 S1 T3 C0[X3] SWAP2,3 ×
        C2,0[Ry1[θ]] Rz3[ϕ] R[λ, X0 Z2 Y3] ];
DrawCircuit[c]
```




```
CalcCircuitMatrix[c][[1, 1]]
```

$$\frac{1}{4}\, e^{-\frac{3i}{2}} \operatorname{Cos}\left[\frac{\lambda}{2}\right] - \frac{1}{4}\, e^{\frac{3i}{2}+\frac{i\pi}{4}} \operatorname{Sin}\left[\frac{\lambda}{2}\right]$$

When a quantum register (a `Qureg`) is created, its expensive numerical representation is stored in the backend QuEST process, the memory for which may be on a remote machine. The Mathematica kernel retains only a pointer to this data structure. When applied to a register, the circuit is dispatched to the backend for simulation.

```
ψ = InitPlusState @ CreateQureg[4];
ApplyCircuit[ψ, c /. {θ → π/3, ϕ → π/4, λ → π/5}];
GetQuregMatrix[ψ][[ ;; 4]] // MatrixForm
```

$$\begin{pmatrix} 0.0672751 - 0.948674\, i \\ 0. + 0.\, i \\ 0. + 0.\, i \\ 0. + 0.\, i \end{pmatrix}$$

By employing Mathematica's powerful symbolic engine, QuESTlink can efficiently simplify operators without representing them in any basis.

```
h = SimplifyPaulis[ ω (X2 X3 +  σ Y3 Y0 + Z0 Z1 + X1 X2) +
        (α X0 + α Z0 X1)^3 (γ X0 Y1 - δ Z2 Z3 + λ Y0 Y1 Z2 X3)^2 ]
```

$$\omega\, X_1 X_2 + \omega\, X_2 X_3 + \sigma\,\omega\, Y_0 Y_3 + 2\,\alpha^3 \left(\gamma^2 + \delta^2 + \lambda^2\right) X_1 Z_0 + \omega\, Z_0 Z_1 - 4\,\alpha^3\,\gamma\,\delta\, Y_0 Z_1 Z_2 Z_3$$

```
CalcExpecPauliSum[
     ψ, h /. {α → 1, ω → 2, γ → 3, σ → 4, δ → 5, λ → 6}, CreateQureg[4]]
```

```
-3.08425
```

There, the instruction `CalcExpecPauliSum` evaluated $\langle\psi|h|\psi\rangle$ using a temporary workspace created with `CreateQureg`. QuESTlink also has a remarkably expressive method of specifying realistic quantum hardware properties, which can capture gate constraints (including of gate family, topology, parameters, and all conceivable by the author), active noise of erroneous gates, passive noise of idle qubits, and unitary interactions with a bath (as represented by "hidden qubits"). These *device specifications* can be extremely substantial, whereby noise and gate translations can depend on dynamic circuit variables and execution times. We offer a very simple and modest example:

```
opt = BaseStyle → {FontFamily → "CMU Serif"};
ViewDeviceSpec[device, opt][[1]] // Row
```

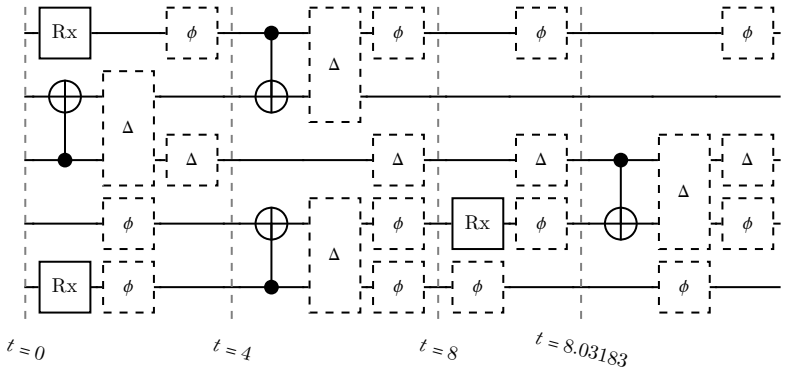
Fields	
Number of accessible qubits	5
Number of hidden qubits	0
Number of qubits (total)	5
Duration symbol	Δt
Description	A line of 5 nearest−neighbour decohering qubits, with imperfect gates

Gates			
Gate	Conditions	Noisy form	Duration (Δt)
$\mathrm{Rx}_{q\_}[\theta\_]$		$\mathrm{Rx}_{q}\left[\frac{\pi}{10}+\theta\right]$	$\frac{\mathrm{Abs}[\theta]}{2\pi}$
$\mathrm{C}_{c\_}[\mathrm{X}_{q\_}]$	Abs[q − c] === 1	$\mathrm{C}_{c}[\mathrm{X}_{q}]$ $\mathrm{Depol}_{c,q}[0.01]$	1 + q

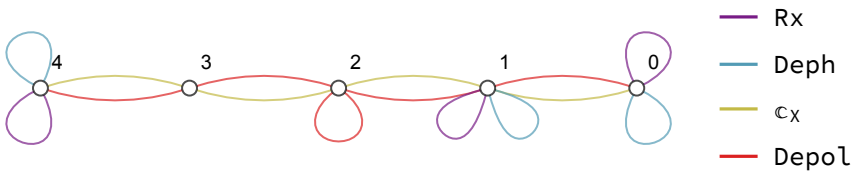
Qubits	
Qubit	Passive noise
0	$\mathrm{Deph}_{0}\left[0.01\left(1-e^{-\Delta t}\right)\right]$
1	$\mathrm{Deph}_{1}\left[0.0225\left(1-e^{-\Delta t}\right)\right]$
2	$\mathrm{Depol}_{2}\left[0.01\left(1-e^{-\Delta t}\right)\right]$
3	
4	$\mathrm{Deph}_{4}\left[0.09\left(1-e^{-\Delta t}\right)\right]$

Device specifications are useful for other users to painlessly transform their input logical circuits into scheduled channels of noisy imperfect operations, as would be realistically effected on the specified device, without themselves understanding the device complexities. The syntax for specifying these virtual devices has evolved iteratively with feedback from experimental groups including those with trapped ion, superconducting, neutral atom, NV-center, and silicon platforms, and can currently capture canonical models like bath interactions, cross-talk, over-rotations, leakage errors, and passive decoherence. Further development of experiment-specific interfaces to device specifications is in an early stage, and is currently lead by my colleague Cica Gustiani.

```
in = Circuit[Rx0[.1] C0[X1] Rx1[.2] Rx4[.3] C2[X3] C2[X1] C4[X3]];
sched = InsertCircuitNoise[in, device];
DrawCircuit[sched, opt]
```




```
DrawCircuitTopology[ExtractCircuit[sched]]
```

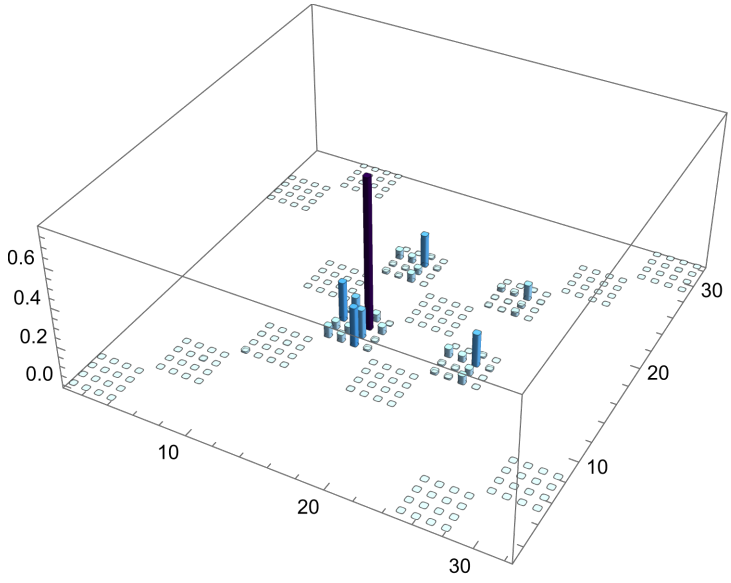


Though not shown here, all parameters of the device specification can remain *symbolic* so that the prescribed circuit contains conditional, symbolic times and noise intensities. This is immensely useful for understanding and optimising dispatched experimental operations. The noisy schedules can be plugged straight into QuESTlink's many density matrix simulation facilities.

```
ρ = CreateDensityQureg[device[NumTotalQubits]];
InitClassicalState[ρ, 4];
ApplyCircuit[ρ, ExtractCircuit[sched]];
CalcPurity[ρ]
```

```
0.833606
```

```
PlotDensityMatrix[ρ]
```



7.5 Interface

The previous section offered a glimpse at the interfaces of QuEST and QuESTlink. This section will provide a more thorough taxonomy and summarise the design considerations made when developing their respective APIs. The prioritised principles and standards of both projects include

- **Least astonishment** [297], whereby functions should behave as users expect, and indicate so unambiguously in their naming. Astonished users create bugs,

and as we will later passionately argue, bugs in simulators are very bad. To this end, both projects offer only functions with no hidden side-effects, which do not fail silently (instead throwing errors as will be described in Section 7.7), and which have only negligible hidden memory costs. Furthermore, they use *imperative* function names like `createQureg` so that the user understands the explicit action performed.

- **Backward compatibility** [298], whereby new project releases minimise changes to previous functions. This ensures, for example, that a new QuEST release does not introduce software regressions into larger stacks. If a backward-breaking change is necessary, it *must* result in an error or warning as opposed to a silent change of behaviour. The latter scenario would likely introduce a bug into a dependent system.

- **Defensive design** [299], whereby functions must explicitly check that all their input preconditions are satisfied. As research tools, QuEST and QuESTlink take every precaution to mitigate bugs and user errors through comprehensive unit testing and stringent user input validation, which will be visited in detail in Sections 7.8 and 7.7.

We now describe some specific instantiations of these principles.

7.5.1 QuEST

The QuEST API is divided into submodules which group similar behaving functions with similar naming prefixes, in-line with the principle of least astonishment. These are

- **Data structures** containing functions like `createQureg`, `createPauliHamil`, `createDiagonalOp` with corresponding `destroy`-prefixed functions. The `create` prefix warns the user that a persistent heap-memory object is returned which they must later destroy.

- **Non-trace-preserving operators** containing functions like `applyProjector`, `applyDiagonalOp`, `applyPauliHamil`. The `apply` prefix warns that the effect on the register may break its validity as a physical state.

- **Decoherence channels** like `mixDamping`, `mixDepolarising`, `mixKrausMap`. The `mix` prefix highlights that since the function induces mixing, it can only be applied to density-matrices.

- **Calculations** like `calcInnerProduct`, `calcExpecPauliHamil`, `calcProbOfOutcome`. The `calc` prefix assures that the function will not modify the register, and will return a scalar or low-dimensional non-memory-intensive output. It warns the user however that some exponentially-intensive computation will be invoked, as opposed to a *getter*.

- **Getters** like `getAmp` and `getNumQubits`. The `get` prefix suggests the operation is trivial and likely obtaining a pre-computed result or property.

- **Setters** like `setAmps` and `setWeightedQureg`. The `set` warns that these functions permit the user to freely modify subsets of the state data and hence corrupt it.

- **State initialisations** like `initPlusState`, `initPureState`, `initStateFromAmps`. The `init` prefix assures that the states are thereafter in valid physical states, unlike the `set` functions. It warns that all amplitudes of the state will be overwritten

regardless of their existing values. Furthermore, the generic `State` infix suggests both statevector and density matrix registers are accepted.

- **QASM utilities** like `startRecordingQASM`, `writeRecordedQASMToFile` and `clearRecordedQASM`. Unlike the other groups, these have consistent *suffixes* for three reasons; to retain the imperative-function-name convention; to retain the camel-case convention (eliminating a `QASM` prefix; to retain the acronyms-as-capitals convention (eliminating a `qasm` prefix).

- **Unitary gates** like `unitary`, `controlledNot`, `multiRotateZ`. While arguably the *main* facility of a quantum computing simulator, these groups have no universally consistent naming convention for several reasons; their early basic forms were named before design constraints were imposed, and persisted for too long to be changed without imposing a significant nuisance for dependent projects; they are divided into subgroups with their own consistent prefixes, like `controlled` and `multiControlled` functions, or the `Rotate` infixed singly-parameterised functions; they include exceptions like `sGate` and `tGate`, which are short and difficult to make consistent.

In addition to the function name, QuEST's function arguments also follow a deliberate convention.

- The argument order is consistent between functions, and matches that suggested by the name, and the convention in the quantum computing community. For example, `multiControlledMultiQubitUnitary` accepts arguments in order `qureg`, `controls`, `numControls`, `targets`, `numTargets`, `unitaryMatrix`, which is faithful to the familiar literature syntax of $C_{\text{controls}}[U_{\text{targets}}]$ (at least in regard to the qubits).

- When a function requires additional working space that scales exponentially with the number of qubits, it does *not* create this space internally; this would incur unexpected memory overheads and possible saturations upon the user, violating least astonishment. Such functions instead accept an additional *working register* from the user, of the same dimension as the primary input register. For example, `calcExpecPauliHamil` accepts arguments `qureg`, `hamil`, `workspace`. Note this is *not* true of functions like `multiQubitUnitary` which *do* involve internal stack-arrays, but since these have a fixed size bound, their memory overhead is negligible relative to that of the input register.

7.5.2 QuESTlink

The naming and argument conventions of the QuESTlink API inherit the previous QuEST considerations, albeit using title-case over camel-case in function names, consistent with inbuilt Mathematica functions. Furthering the principle of least astonishment, QuESTlink imposes several additional design conventions:

- While prefix `Calc` is still used for functions with exponentially-growing numerical costs (like `CalcPauliExpressionMatrix`), some functions can become unexpectedly intractable due to the involved symbolic expansions of their internal Mathematica functions. In these instances, their QuESTlink name hints at the internal bottleneck. For example, `SimplifyPaulis` which internally calls `Simplify`.

- In-line with Mathematica convention, optional function parameters, those which provide supplemental information, or those which cause deviation from the expected behaviour, should be supplied as `Options`. For example, `ApplyCircuit` accepts `ShowProgress` $\rightarrow$ `True` (to render an otherwise hidden progress-bar during

evaluation), in-lieu of a positional argument like `ApplyCircuit[qureg, circuit, True]`. This has several benefits; it keeps function signatures compact; it allows inexperienced users to invoke default behaviour without worrying about all arguments; it self-documents the role of the parameter.

- Functions which invoke internal Mathematica facilities which the user may wish to customise should propagate user-options to them. For example, `DrawCircuit` invokes internal function `Graphics` and as such, forwards all compatible options like `BaseStyle`. This allows a user to programmatically modify the output of `DrawCircuit` like they would any Mathematica evaluation, for instance to change colours or fonts.

7.6 Architecture

This section describes and motivates the software architecture of QuEST, and the derivative structure of QuESTlink. We neglect the less interesting discussion of their *build* systems, except to say that QuEST uses `CMake` (build developed by Ania Brown) to ease integration into larger software stacks, and QuESTlink uses `GNUMake` (build developed by the author) for easy dependency-free standalone compilation.

7.6.1 QuEST

The QuEST API strives to be agnostic to precision (4-16 byte floating-point numbers), to language (`C` and `C++`), to register types (statevectors and density-matrices) and to deployed hardware (serial systems, multi-CPUs, GPUs and distributed supercomputers). Achieving this simplicity for the user requires a somewhat complicated software

architecture from the designer, which we visualise and describe in Figure 7.2.

To clarify the complicated network of dependencies, internal function names are typically prefixed with either `validate`, `qasm_`, `statevec_`, `densmatr_` or `agnostic_` (and in the CPU backend, suffixed with `Local` or `Distributed`). The latter three prefixes are fundamental to QuEST's clear use of the Choi isomorphism of Section 6.7.1 to represent density matrices while minimising code duplication (a generally good practise [299]). For instance, the multi-controlled phase gadget $C_{\vec{c}}[\exp(\mathrm{i}\,\theta \otimes_{q\in\vec{t}} \hat{Z}_q)]$ modifies an N-qubit density matrix represented as a $2N$-qubit Choi vector $\|\rho\rangle\!\rangle$ via

$$\|\rho\rangle\!\rangle \;\rightarrow\; C_{\vec{c}+N}\left[\exp\left(-\mathrm{i}\,\theta \bigotimes_{q\in\vec{t}} \hat{Z}_{q+N}\right)\right]\; C_{\vec{c}}\left[\exp\left(\mathrm{i}\,\theta \bigotimes_{q\in\vec{t}} \hat{Z}_{q}\right)\right]\; \|\rho\rangle\!\rangle\,, \tag{7.1}$$

i.e. as *two* unitary statevector operations. As such, its code need only invoke backend functions which expect statevectors, as demonstrated in Algorithm 7.1. In contrast, some facilities make use of bespoke density matrix routines prefixed with `densmatr_`, while other internal functions make no assumptions of the state type (because they invoke high-level facilities like entire gates) and are prefixed with `agnostic_`.

An important component of the QuEST architecture is the user input validation, which will be described in Section 7.7. When validation fails, QuEST invokes the `invalidQuESTInputError()` function which by default terminates execution after displaying an error message specific to the problem. For example, `"QuEST error in applyTrotterCircuit: The number of Trotter repetitions must be >=1. Exiting..."`. This function is declared as a *weak* symbol [300] (or as an *alternate name* on MSVC compilers [301]) so that a user may override it with a custom definition which receives the name of the function and the explanatory message. This *error hook* [302] is useful for larger software stacks to utilise QuEST's validation system and gracefully handle

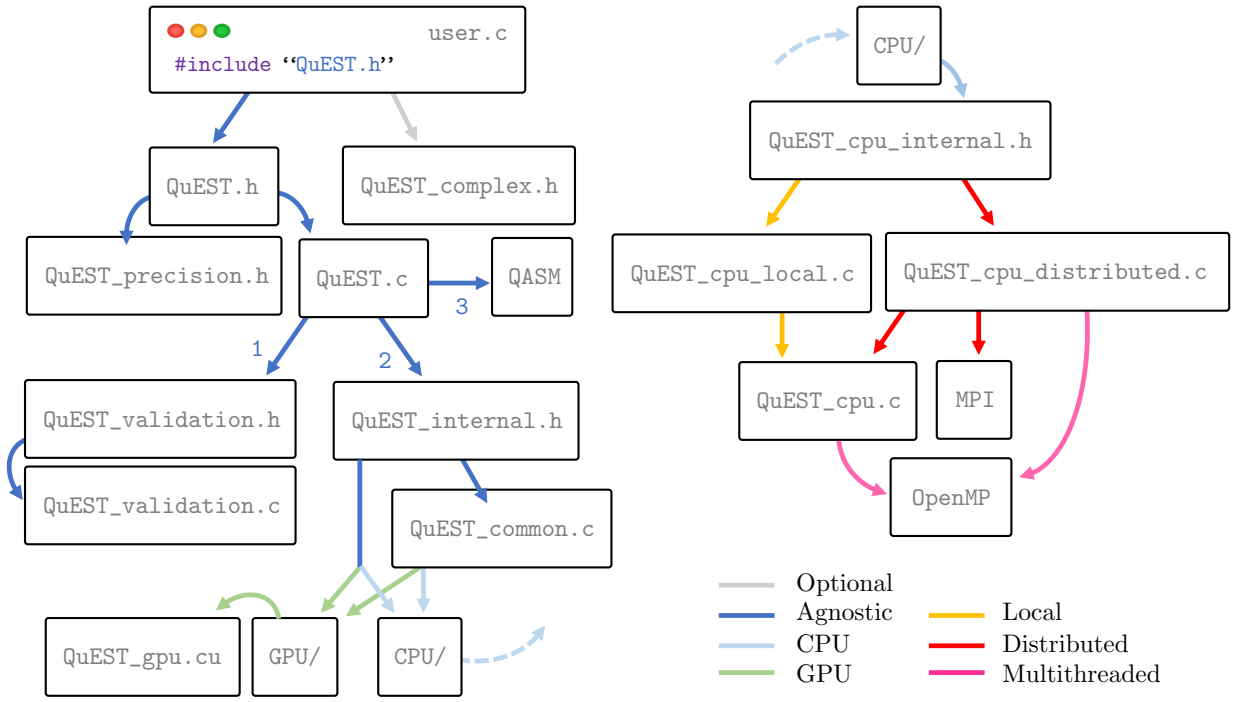


Figure 7.2: The QuEST software architecture, as semantically modularised between `C` and `C++` source and header files. The optionally included `QuEST_complex.h` contains a precision-agnostic complex type and quality-of-life functions for parsing complex scalars to the QuEST API. `QuEST_precision.h` contains a precision-agnostic real type used by the entire project. `QuEST.c` contains the front-facing API, and in-turn triggers input validation, the backend, then QASM logging. In this way, the interface is platform agnostic, and no hardware-specific code is invoked before the user input is validated, minimising the risk of un-debuggable crashes. `QuEST_internal.h` declares all front-end accessible internal functions, though some re-purpose the backends of other functions, and are thus defined without any platform code in `QuEST_common.c`. The remaining functions have distinct CPU and GPU backends, the latter of which is defined entirely in `C++` CUDA code within `QuEST_gpu.cu` To maximise code reuse, the CPU backend features almost all low-level serial code (with optional multithreaded precompiler directives) in `QuEST_cpu.c`, which are called by either `QuEST_cpu_local.c` or `QuEST_cpu_distributed.c` (only one of which is compiled and linked). This is because the distributed algorithms typically invoke the local algorithms as subroutines, but perform extra communications using the linked MPI library.

Algorithm 7.1: An example function within `QuEST.c` which simulates the multi-controlled phase gadget upon both N-qubit statevectors and density-matrices represented as $2N$-qubit Choi vectors (as per Equation 7.1). Notice all backend functions expect a statevector. The qubit indices are concisely translated by N via bit-shifts upon their bit-masks.

multiControlledMultiRotateZ($\vec{\psi}$, N, isDensityMatrix, $\vec{c}$, $\vec{t}$, θ):
    validateMultiControlsMultiTargets($\vec{\psi}$, $\vec{c}$, $\vec{t}$)
    cm = **getBitMask**($\vec{c}$)
    tm = **getBitMask**($\vec{t}$)
    statevec_multiControlledMultiRotateZ($\vec{\psi}$, cm, tm, θ)
    if (isDensityMatrix)
        statevec_multiControlledMultiRotateZ($\vec{\psi}$, cm $\ll N$, tm $\ll N$, $-\theta$)
    qasm_recordMultiControlledMultiRotateZ($\vec{\psi}$, $\vec{c}$, $\vec{t}$, θ):

user input errors. For instance, QuESTlink redefines `invalidQuESTInputError()` to throw a `C++` exception which is caught at a higher level and propagated to a Mathematica `Message[]` error (elaborated upon in the next section). QuEST has similar hooks for random number generation, and reporting hardware facilities.

We have neglected discussion of the unit testing framework, which we defer to Section 7.8.

7.6.2 QuESTlink

We now describe the QuESTlink software architecture, which makes extensive use of the Wolfram Symbolic Transfer Protocol (WSTP [303]), formerly known as MathLink [304], to communicate between independent `C` and Mathematica processes. Notice QuESTlink adopts the `Link` suffix conventional for linked Mathematica processes, though capitalisation changed for visual clarity. All platform specific and high-performance codes reside in the core QuEST project, appearing as a `git` submodule [305] of the QuESTlink repos-

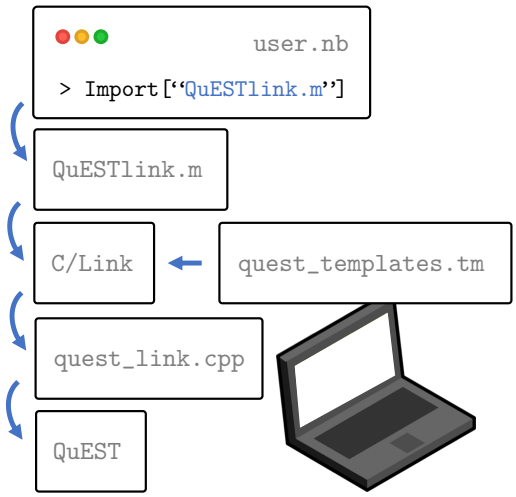


Figure 7.3: The QuESTlink software stack, from the user's code (`user.nb`) to the driving QuEST framework. This stack can run entirely on a user's local machine, achieving an interactive Mathematica interface to a `C/C++` based emulator. Note the distinction between WSTP (the protocol for communicating between Mathematica and an external process) and `C/Link` (an implementation for C) is unimportant, and hereafter disregarded.

itory. The remaining code is divided between three source files; `QuESTlink.m`, written in Mathematica (also known as the *Wolfram Language*) which provides high-level analytic and graphing facilities, and encodes user inputs like circuits into a primitive language understood by the lower layers; `quest_link.cpp` written in `C++` which invokes the QuEST backend, and `quest_templates.tm`, written in a bespoke Wolfram template language parsed by WSTP, which maps `QuESTlink.m` symbols to `quest_link.cpp` signatures. This stack is visualised in Figure 7.3.

The structure of `QuESTlink.m` is very simple, consisting predominantly of high-level code and patterns mapping user inputs to WSTP symbols. These symbols trigger `quest_link.cpp` functions which nearly all return WSTP symbolic expressions (in lieu of primitives like integers), allowing runtime errors to be propagated back to the Mathematica frontend as `Message[]` invocations. QuESTlink overrides QuEST's `invalidQuESTInputError` hook to throw a `C++` exception which is caught within `quest_link.cpp`,

and propagated. This allows QuESTlink to utilise QuEST's comprehensive input validation (elaborated upon in Section 7.7) without code duplication. It means however that the code of `quest_link.cpp` can be quite complicated, containing the memory management and checkpointing necessary to prevent crashes and memory leaks upon invalid user input.

We now elaborate on the process of deploying QuESTlink, which in a sense, details its *running* software architecture. The call `Import["https://qtechtheory.org/questlink.m"]` is redirected by the `qtechtheory.org` server to the latest version of the QuESTlink Mathematica package on Github. This immediately enables all analytic and graphing facilities, and defines the subsequently callable `CreateDownloadedQuESTEnv[]` which downloads the latest precompiled QuESTlink backend executable (containing QuEST) from the Github repository, informed by the user's operating system. The user may alternatively call `CreateLocalQuESTEnv[exec]` to use a local executable perhaps compiled for their specific hardware, or `CreateDownloadedQuESTEnv[ip, port1, port2]` to connect to a *remote* server like a supercomputer. We visualise this file serving in Figure 7.4. The resulting communication pattern between the active Mathematica and `C++` processes constituting QuESTlink are shown in Figure 7.5.

7.7 Input validation

In an idyllic alternate reality, users digest the full documentation (to be seen in Section 7.9) and write perfect code which satisfies all preconditions of an API. But there is no time for such endeavours in our own reality. Researchers move fast and break things, and unknowingly request nonsensical tasks of their simulation tools. If such mistakes are left undiscovered, they risk wasting significant time, effort and money (think su-

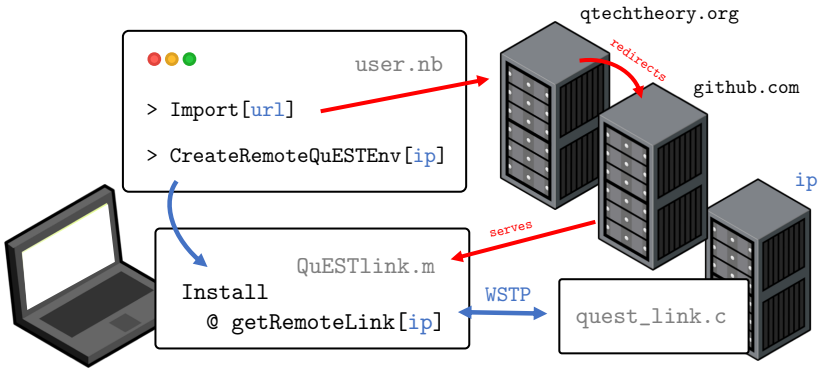


Figure 7.4: The protocol for stand-alone deployment of QuESTlink. First, a user calls Mathematica's `Import[url]` function to fetch a copy of the `QuESTlink.m` package, hosted on Github, but redirected from qtechtheory.org. This provides the `QuEST'` namespace, and defines functions to connect to a `quest_link` backend; this backend can exist on the calling machine, or remotely as here pictured.

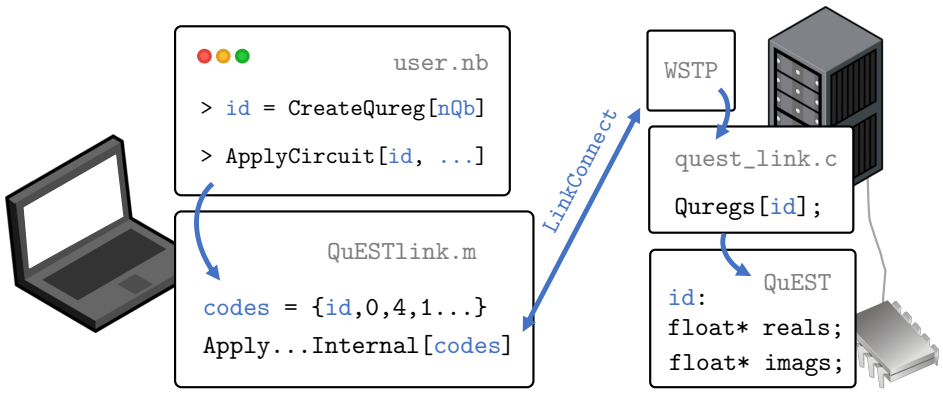


Figure 7.5: Protocol for off-loading an emulation task onto a remote QuEST environment. Simulation structures declared in Mathematica are actually allocated remotely, possibly even in accelerated hardware memory. Quantum circuits, by first being encoded into arrays of numbers, are communicated to the server over the internet (via TCP/IP).

percomputing credits), corrupting research conclusions and/or making their way into wider software stacks. It is therefore crucial that scientific libraries feature stringent *input validation* [299] whereby all assumptions made of the API input are checked and if not satisfied, yield informative error messages and fatally stop execution. A user submitting unexpected inputs which silently corrupt their simulation is a failure of the simulator, and a violation of the founding software design principles.

In typical applications, the rigour of validation forms a trade-off with a performance penalty; thankfully in statevector simulation, validation overheads are completely outweighed by the exponential cost of the API backend functions. At present date, the QuEST API features 66 unique checks upon the user input (such as whether qubit indices are valid, and that decoherence probabilities decrease purity), and QuESTlink adds another 98 (such as whether Pauli operator expressions contain standalone scalars, and that noisy circuit schedules feature monotonically increasing times). Consistent with the defensive design principle of Section 7.5, both projects lean to over-validation, throwing errors even when a user error is principally recoverable.

We offer an illustrative example; the input validation within QuEST's function `multiControlledMultiQubitUnitary`. This function simulates the multi-controlled multi-target general unitary gate $C_{\vec{c}}[\hat{U}_{\vec{t}}]$ with m control qubits $\vec{c} = \{c_1, \dots, c_m\}$ (type forced to be integers), n target qubits $\vec{t} = \{t_1, \dots, t_n\}$ (type forced to be integers), a forcedly-square $2^\eta \times 2^\eta$ complex matrix $\underline{\underline{U}}$, and acting upon an N-qubit register (a statevector or density matrix). All of $\vec{c}$, m, $\vec{t}$, n, $\underline{\underline{U}}$, η and N (provided through a register) are inputs with the following validated preconditions:

- The control and target qubit indices are in the valid domain;

$$0 \leq c_i < N \;\; \forall \;\; 1 \leq i \leq m \quad \text{and} \quad 0 \leq t_i < N \;\; \forall \;\; 1 \leq i \leq n.$$

If unsatisfied and uncaught, this precondition would cause a runtime segmentation-fault.

- There are a valid number of declared qubit indices;

$$m < N \quad \text{and} \quad n \leq N.$$

This includes the possibility of *no* control qubits ($n = N$) but excludes *no* target qubits ($m \neq N$). This scenario will actually imply other caught validations like those of uniqueness, but remains a worthwhile quality-of-life check to alert users that their input is *already* invalid, without looking at the values in $\vec{c}$ or $\vec{t}$. Note that in distributed settings, there are additional runtime restrictions on the size of n due to communication buffers, as described in Section 6.6.2;

$$n \leq N - \log_2(\#\text{nodes}).$$

- The control and target qubits are disjoint;

$$\vec{c} \ \cap \ \vec{t} \ = \ \vec{\emptyset}.$$

An overlap of the control and target qubits would not cause a runtime error, but would produce a logically incorrect and non-interpretable change to the register. This precondition can be checked in $\mathcal{O}(N(n+m))$ bitwise operations.

- The control and target qubits are each unique;

$$c_i \neq c_j \ \forall \ i \neq j \quad \text{and} \quad t_k \neq t_l \ \forall \ k \neq l.$$

Duplication in the target qubits would silently produce an incorrect result. Interestingly, duplicates in the control qubits would cause no errors, but they are likely to be unintentional and indicate a potential issue in the user's prepared inputs.

These checks can be performed in $\mathcal{O}(n+m)$ bitwise operations.

- The matrix $\underline{\underline{U}} : \mathbb{C}^{2^\eta \times 2^\eta}$ has the correct dimensions for the number of qubits, i.e. $\eta = n$. Otherwise, the routine would segmentation-fault. Furthermore, that η is not too large as to cause stack overflows in the temporary memory internally created by the function.

- The matrix $\hat{\underline{\underline{U}}}$ is unitary, as can be assessed under numerical uncertainty;

$$\underline{\underline{U}}^\dagger \underline{\underline{U}} \approx \mathbb{1}^{\otimes \eta} \quad \implies \quad \left| \sum_i u_{ri}\, u^*_{ci} - \delta_{rc} \right| < \varepsilon \;\; \forall \;\; 1 \leq r, c \leq 2^\eta.$$

  This property can only be checked against some accuracy threshold $\varepsilon \ll 1$ in time $\mathcal{O}(2^{3\eta})$. While this appears to be an exponentially growing overhead, recall that the size of η is upperbounded, and certainly satisfies $\eta \ll N$. This check of unitarity protects the user from unintentionally breaking the normalisation of their register, but may prove a barrier to other users who *intend* to affect a non-unitary; hence it is important the API include other functions without such checks, which appropriately warning names. In QuEST, this is `applyMultiControlledMatrixN`, with the `apply` prefix discussed in Section 7.5.

Note that several additional preconditions did not need to be checked (like that qubit indices were integers), and several could *not* be checked (like that $\vec{c}$ was an m-length array as the user claims), due to properties of the `C` programming language (strong-typing and dimensionless heap-array pointers). Further, several contextual assumptions need not be enforced, like prior normalisation of the input register, since advanced may users may wish to manipulate non-physical states.

7.8 Testing

Use of tools like QuEST and QuESTlink save significant time for researchers because they abstract away the tedium and complications of simulation. To remain useful, these tools must be robust; bugs in the tools serve to waste the researcher's time and jeopardise their research. Section 7.7 argued that all API input *preconditions* must be validated by the simulator, but this pales against the importance of validating the *postconditions*. It is *essential* that scientific tools perform as they are described, and it is not the job of the user to check a tool's validity for their application domain. For a scientific tool to be of any use, all facilities must be rigorously checked against all conceivable inputs (where feasible), *systematically* on each target platform and operating system, and *automatically* whenever any change is made to the code base. That is, scientific libraries need continuously-integrated regression and multi-platform unit testing. We describe the incorporation of these into QuEST.

7.8.1 Unit tests

Unit tests are codes which aim to verify the behaviour of every individual *unit* of software [306]. They seek bugs at the lowest level of the software architecture, typically individual functions, and give users confidence that the software performs correctly on the platforms to which the tests have been deployed. They also help scaffold the behaviour of a function; in *test-driven development* (TDD), occasionally used in the development of QuEST, the unit tests of a function are written before the tested function itself, which is considered complete once the tests pass [307].

Comprehensive, automated, high-coverage unit tests are especially important in QuEST

for several reasons:

- As visible in Figure 7.2, QuEST has multiple platform-specific backends with algorithmically-distinct implementations of the same function (semantically speaking), which are built by an ecosystem of different compilers. Each combination of settings can produce unique bugs, and so each must be tested in-turn.

- As QuEST's implementations use the high-performance paradigms of Chapter 6, they differ markedly from the unoptimised form of their core algorithms. As such, the code can be difficult to analyse impractical to formally verify. Unit testing is crucial to ensure that optimisation did not introduce bugs.

- Some functions contain a surprising multitude of edge-cases. For example, two-qubit depolarising in distributed settings admits *five* distinct communication scenarios, and within each, multiple groups of nodes with distinct behaviours. These behaviours (and corresponding codes) are triggered depending on the register size, target qubits and the number of nodes.

- Many functions permit inconspicuous bugs easily missed by manual tests. For instance, consider if line 6.2 of Algorithm 6.2 (serial, local simulation of a simple single-qubit unitary), which reads

$$\vec{\psi}[\text{ind}\beta] = u_{22}\,\beta + u_{21}\,\alpha,$$

was erroneously modified to

$$\vec{\psi}[\text{ind}\beta] = u_{22}\,\beta + u_{12}\,\alpha.$$

This bug would be undetected by any test with an input matrix $\underline{\underline{U}}$ with elements satisfying $u_{21} = u_{12}$, such as $\underline{\underline{U}} = \frac{1}{\sqrt{2}}\begin{pmatrix} \mathrm{i} & 1 \\ 1 & \mathrm{i} \end{pmatrix}$. Even Algorithm 6.4 (local, multi-

threaded simulation of a multi-qubit unitary) contains a stack-overflow triggered for different inputs depending on the number of active threads, the operating system page limits, and performance penalties determined by the CPU cache sizes.

For these reasons, unit tests must verify every possible input where feasible. The next sections will describe the different aspects and considerations of the unit tests.

7.8.2 Verification

This sections describes to what the unit tests compare QuEST's output. Often unit tests merely check that a function yields a set of analytically known results [308]. For example, whether function `calcProbOfOutcome` called upon a symmetric state admits $|\langle 0|+\rangle|^2 = 1/2$. But having just argued the need for comprehensive tests of all inputs, we will need to create an automated reference output, or *oracle*. Fortunately, this thesis has demonstrated that there are an abundance of methods to skin a quantum cat and compute the same function via distinct algorithms. We can verify that QuEST's complicated algorithmically-optimised high-performance code produces the same outputs as simple, un-optimised, algorithmically-naive and human-readable oracles. The QuEST source-code file `tests/utilities.cpp` contains 128 developer-accessible user-inaccessible `C++` functions to inefficiently but robustly evaluate linear-algebra routines in-analogously to the methods of QuEST's backend. This is often through decomposition into many simple operations. For example, in lieu of methods similar to the in-place high-performance routines in Algorithm 6.3 to simulate a controlled-gate upon an N-qubit statevector, `tests/utilities.cpp` performs a multi-controlled multi-target gate $C_{\vec{c}}[\hat{U}_{\vec{t}}]$ by constructing full-state matrices costing $\mathcal{O}(2^{2N})$ additional memory. Letting

$m = \dim \vec{c}$ and $n = \dim \vec{t}$, we leverage

$$\begin{aligned} C_{\vec{c}}[\hat{U}_{\vec{t}}] \;\equiv\; & \text{SWAPS}_{\vec{t}\leftrightarrow\{0,\,\ldots,\,n-1\}}\;\; \text{SWAPS}_{\vec{c}\leftrightarrow\{n,\,\ldots,\,n+m-1\}}\;\; \times \\ & C_{\{n,\,\ldots,\,n+m-1\}}[U_{\{0,\,\ldots,\,n-1\}}]\;\; \times \\ & \text{SWAPS}_{\{n,\,\ldots,\,n+m-1\}\leftrightarrow\vec{c}}\;\; \text{SWAPS}_{\{0,\,\ldots,\,n-1\}\leftrightarrow\vec{t}}, \end{aligned} \tag{7.2}$$

where SWAPS is a product of individual SWAP gates, and the $(m+n)$-qubit unitary in the centre is simply an identity-matrix with a rightmost $2^n \times 2^n$ submatrix $\underline{\underline{U}}$. That is (showing $n = 1$ for clarity),

$$C_{\{1,\ldots,m\}}[\hat{U}_0] = \mathbb{1}^{\otimes(m+1)} - |1\rangle\langle 1|^{\otimes m} \otimes (\hat{U} - \mathbb{1}) \tag{7.3}$$

$$= 2^m \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \begin{matrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{matrix} \end{pmatrix} \tag{7.4}$$

which is then left-tensored with $\mathbb{1}^{\otimes(N-m-n)}$ to the full Hilbert space. Each two-qubit SWAP gate is produced by Lemma 3.1 of Reference [309], as

$$\text{SWAP}_{q_2,q_1} \;=\; \text{SWAP}_{q_2-q_1,0} \;\otimes\; \mathbb{1}^{\otimes q_1} \tag{7.5}$$

$$\equiv \begin{pmatrix} \mathbb{1}^{\otimes(q_2-q_1)} \otimes \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right) & \mathbb{1}^{\otimes(q_2-q_1)} \otimes \left(\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right) \\ \mathbb{1}^{\otimes(q_2-q_1)} \otimes \left(\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right) & \mathbb{1}^{\otimes(q_2-q_1)} \otimes \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right) \end{pmatrix} \;\otimes\; \mathbb{1}^{\otimes q_1}. \tag{7.6}$$

Through this process, we obtain a $2^N \times 2^N$ matrix encoding $C_{\vec{c}}[\hat{U}_{\vec{t}}]$ which we matrix-multiply onto the statevector. The result is the action of a multi-controlled multi-target unitary upon the register, through a method totally dissimilar to the tested API routine,

and hence one which is extremely unlikely to admit an identical error.

The `C++` code in `tests/utilities.cpp` makes extensive use of *operator-overloading* [310] and complex types to improve readability, and all utility functions explicitly check their own preconditions at every invocation. They include quality-of-life functions to assist with writing unit tests, such as pseudo-random generators of n-qubit unitary matrices via Gram-Schmidt orthogonalisation [311] of random complex row-vectors.

Finally, the unit tests verify that the API throws an error (invoking `invalidQuESTInputError`) for every possible form of erroneous user input, and that the error message is correct (or contains a desired sub-string). For example, the unit tests for `multiControlledMultiRotateUnitary` upon an N-qubit register include invalid input scenarios when

- the number of target or control qubits is ≤ 1 or $\geq N$.
- the target and control qubits contain a repetition, or overlap, or contain invalid indices (< 0 or $\geq N$).
- the unitary matrix is non-unitary, or has incorrect dimensions ($\neq 2^n \times 2^n$), or cannot fit into the node in distributed settings ($n > N - \log_2(\#\text{nodes})$).

The thrown error messages are checked for substrings such as `"invalid matrix size"`.

7.8.3 Generation

Having described how we verify a QuEST *output*, we now discuss how we generate the test *input*. We previously argued that we must test all possible inputs. To do so, we employ `Catch2` [312], a light-weight standalone `C++` unit-testing framework which is able to concisely generate combinations of parameters. For example, the m-controlled n-

target general unitary $C_{\vec{c}}[\hat{U}_{\vec{t}}]$ is simulated upon an N-qubit register for all permutations of inputs

$$n\text{Max} = \min(N - \log_2(\#\text{nodes}), N - 1), \tag{7.7}$$

$$\forall\ n \in \{1,\ \ldots,\ n\text{Max}\}, \tag{7.8}$$

$$\forall\ m \in \{1,\ \ldots,\ N - n\}, \tag{7.9}$$

$$\forall\ \vec{t} \in \{\text{ all length-}n\text{ permutations of }\{0,\ \ldots,\ N-1\}\ \}, \tag{7.10}$$

$$\forall\ \vec{c} \in \{\text{ all length-}m\text{ permutations of }\{0,\ \ldots,\ N-1\} \setminus \vec{t}\ \}, \tag{7.11}$$

$$\underline{\underline{U}} = \text{a pseudorandom } 2^n \times 2^n \text{ unitary matrix.} \tag{7.12}$$

When non-distributed, this generates a staggering $\sum_{n=1}^{N-1} \sum_{m=1}^{N-m} P^N_{m+n} = \mathcal{O}((N+2)!)$ number of unique parameter sets. For a modest $N = 5$ qubits, this yields 980 unique tests. Performing this tractably required developing bespoke templated classes which implement the `Catch2 IGenerator` interface to produce k-permutations. Each test is performed separately on a statevector and density matrix. By abstracting away this generation through convenience functions in `tests/utilities.cpp`, the resulting input-preparing portion of the unit testing code becomes remarkably concise. The example above (full code here) is generated by

```
// try all possible numbers of targets and controls
int numTargs = GENERATE_COPY( range(1,maxNumTargs+1) );
int maxNumCtrls = NUM_QUBITS - numTargs;
int numCtrls = GENERATE_COPY( range(1,maxNumCtrls+1) );

// generate all possible valid qubit arrangements
int* targs = GENERATE_COPY(
sublists(range(0,NUM_QUBITS), numTargs) );
int* ctrls = GENERATE_COPY(
sublists(range(0,NUM_QUBITS), numCtrls, targs, numTargs) );

// for each qubit arrangement, use a new random unitary
QMatrix unitary = getRandomUnitary(numTargs);

// test multiControlledMultiQubitUnitary(
qureg, targs, numTargs, ctrls, numCtrls, unitary);
```

7.8.4 Automation

With the *why*, *what* and *how* of QuEST's unit testing described, we now address the *when*. It is not enough to conduct the tests once. After all, between the three numerical precision modes (single, double, quadruple), the two calling languages (`C` and `C++`), the five deployment modes (serial, local & multithreaded, distributed, distributed & multithreaded, GPU-accelerated), the three main supported operating systems (Windows, MacOS and Linux) and the four main supported compiler types (Intel, GNU, Clang and MSVC), there are already 360 distinct environments with their own capacities for bugs. And yet, a once-off test in all these settings in insufficient. Every change to the source code can, in principle, introduce a new bug into an existing facility. We cannot task a code reviewer to ensure against this, being as fallible as the original programmer.

We instead re-utilise our unit tests for *regression testing* [298] by automatically re-running them whenever changes are proposed of the primary source code; this is *con-*

tinuous integration (CI) [313]. Since QuEST is hosted on Github, we use Github Actions [314] to trigger workflows specified in `YAML` files [315] which recompile QuEST and run the full suite of unit tests whenever a pull request is made. The output resembles:

```
Start   1: calcDensityInnerProduct
1/127 Test   #1: calcDensityInnerProduct ..............   Passed    0.44 sec
Start   2: calcExpecDiagonalOp
2/127 Test   #2: calcExpecDiagonalOp ..................   Passed    0.41 sec
Start   3: calcExpecPauliHamil
3/127 Test   #3: calcExpecPauliHamil ..................   Passed   10.78 sec

...

Start 126: twoQubitUnitary
126/127 Test #126: twoQubitUnitary .....................   Passed    1.19 sec
Start 127: unitary
127/127 Test #127: unitary .............................   Passed    0.30 sec

100% tests passed, 0 tests failed out of 127

Total Test time (real) = 3253.31 sec
```

Ideal unit tests trigger every distinct behaviour of software, which is measurable via *code coverage* [316]; a metric of how many lines of the source code are executed during testing. QuEST uses `LCOV` [317] to monitor coverage changes, also triggered in the CI, and reported by `codecov.io`.

7.9 Documentation

Between Sections 7.5 and 7.8, we argued that functions must do "what they say on the tin". We now describe the tin. Documentation (doc) is paramount in deciding the utility of a tool. Beyond being a catalogue of available facilities, documentation

stringently specifies the precise behaviour of a function, against which all testing is compared [299]. Good doc includes:

- Clear, concise summarises of the basic utility of functions, allowing users to efficiently search through the facility catalogue. In scientific libraries, this is aided by illustrative diagrams and maths rendering, and short code examples [318]. Browsable doc also include links to other relevant doc items, to aid discovery of software facilities.

- Longer, precise and comprehensive descriptions of all nuances in each function. This must include the behaviour of the function upon all input types, all preconditions, all outputs and exceptions thrown by the function, and crucially, all undetectable but invalid inputs which cause runtime errors. In scientific code, it should document the runtime and memory complexities of the function, to aid the user's performance accounting, especially if the facility is more expensive than the API average. If the code is open-source, the doc should link to the function source-code and its unit tests, so that the user may see its formally defined behaviour.

- Autogeneration. Writing doc and updating it as the code base changes can be laborious. This is why the doc should be written as close to the source code as possible, as a *docstring*, to be automatically rendered into browsable documentation through a doc generator [318].

The QuEST doc fulfils most (attemptedly *all*) of these characteristics, and is generated at each release from `doxygen` docstrings which include `LaTeX` and `tikz` markup, and which are styled with custom `HTML` and `CSS`. We show an example in Figure 7.6. Note that while such API doc is useful as a *lookup reference*, it is a poor introduction to

the library for a first-time user [318]. So in addition, the author has created a series of tutorials and demonstrations at quest.qtechtheory.org/docs. We direct the interested reader to the tutorial on Grover's Search (link).

Meanwhile, the documentation for the QuESTlink project is less developed, since there is presently no official Mathematica doc generator. Instead, the QuESTlink API catalogue can be explored by running `?QuEST‘*` directly within a notebook. A short doc snippet for function `f` is viewable by `?f`, which we demonstrate in Figure 7.7. With future plans to emulate Wolfram's official external doc style using third-party tools, we have in the meantime provided a multitude of demonstrations at website questlink.qtechtheory.org.

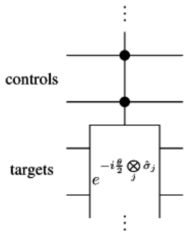
```
void multiControlledMultiRotatePauli(Qureg          qureg,
                                     int *          controlQubits,
                                     int            numControls,
                                     int *          targetQubits,
                                     enum pauliOpType * targetPaulis,
                                     int            numTargets,
                                     qreal          angle
                                    )
```

Apply a multi-controlled multi-target multi-Pauli rotation, also known as a controlled Pauli gadget.

This is the unitary

$$|1\rangle\langle 1|^{\otimes \text{numControls}} \otimes \exp\left(-i\frac{\theta}{2}\bigotimes_{j}^{\text{numTargets}} \hat{\sigma}_j\right) + \sum_{k=0}^{2^{\text{numControls}}-2} |k\rangle\langle k| \otimes \mathbf{I}$$

where $\hat{\sigma}_j$ are the Pauli operators (pauliOpType) in `targetPaulis`, which operate upon the corresponding qubits in `targetQubits`.



> All qubits not appearing in `targetQubits` and `controlQubits` are assumed to receive the identity operator.

For example:

```
int numCtrls = 1;
int numTargs = 4;
int ctrls[] = {4};
int targs[] = {0,1,2,3};

pauliOpType paulis[] = {PAULI_X, PAULI_Y, PAULI_Z, PAULI_I};

multiControlledMultiRotatePauli(
    qureg, ctrls, numCtrls, targs, paulis, numTargs, 0.1);
```

effects

$$|1\rangle\langle 1| \otimes \exp\left(-i\,(0.1/2)\,X_0 Y_1 Z_2\right)\mathbf{I}_3 \;+\; |0\rangle\langle 0| \otimes \mathbf{I}^{\otimes 4}$$

on `qureg`, where unspecified qubits (along with those targeted by `PAULI_I`) are assumed to receive the identity operator (excluded from exponentiation).

> This means specifying `PAULI_I` does *not* induce a global phase factor $\exp(-i\theta/2)$. Hence, if all `targetPaulis` are identity, then this function does nothing to `qureg`. Specifying `PAULI_I` on a qubit is superfluous but allowed for convenience.

This function effects the controlled Pauli gadget by first (controlled) rotating the qubits which are targeted with either `X` or `Y` into alternate basis, performing multiControlledMultiRotateZ() on all target qubits, then restoring the original basis.

See also

- multiControlledMultiRotateZ()
- multiRotatePauli()
- multiRotateZ()
- rotateX()
- rotateY()
- rotateZ()
- rotateAroundAxis()

Parameters

[in,out]	**qureg**	object representing the set of all qubits
[in]	**controlQubits**	list of the indices of qubits to control upon
[in]	**numControls**	length of length `controlQubits`
[in]	**targetQubits**	a list of the indices of the target qubits
[in]	**targetPaulis**	a list of the Pauli operators around which to rotate the target qubits
[in]	**numTargets**	length of list `targetQubits`
[in]	**angle**	the angle by which the multi-qubit state is rotated around the Z axis

Exceptions

invalidQuESTInputError()	• if any qubit in `controlQubits` and `targetQubits` is invalid, i.e. outside **[0,** `qureg.numQubitsRepresented` **)** • if `controlQubits` or `targetQubits` contain any repetitions • if any qubit in `controlQubits` is also in `targetQubits` (and vice versa) • if `numTargets` **< 1** • if `numControls` **< 1** (use multiRotateZ() for no controls) • if any element of `targetPaulis` is not one of `PAULI_I`, `PAULI_X`, `PAULI_Y`, `PAULI_Z`
segmentation-fault	• if `controlQubits` contains fewer elements than `numControls` • if `targetQubits` contains fewer elements than `numTargets` • if `targetPaulis` contains fewer elements than `numTargets`

Author
Tyson Jones

Definition at line 705 of file QuEST.c.

```
705
706 {
    validateMultiControlsMultiTargets(qureg, controlQubits, numControls,
targetQubits, numTargets, __func__);
707     validatePauliCodes(targetPaulis, numTargets, __func__);
708
709     int conj=0;
710     long long int ctrlMask = getQubitBitMask(controlQubits, numControls);
711     statevec_multiControlledMultiRotatePauli(qureg, ctrlMask,
targetQubits, targetPaulis, numTargets, angle, conj);
712     if (qureg.isDensityMatrix) {
713         conj = 1;
714         int shift = qureg.numQubitsRepresented;
715         shiftIndices(targetQubits, numTargets, shift);
716         statevec_multiControlledMultiRotatePauli(qureg, ctrlMask<<shift,
targetQubits, targetPaulis, numTargets, angle, conj);
717         shiftIndices(targetQubits, numTargets, -shift);
718     }
719
720     // @TODO: create actual QASM
721     qasm_recordComment(qureg,
722         "Here a %d-control %d-target multiControlledMultiRotatePauli of
angle %g was performed (QASM not yet implemented)",
723         numControls, numTargets, angle);
724 }
```

References getQubitBitMask(), Qureg::isDensityMatrix, Qureg::numQubitsRepresented, qasm_recordComment(), shiftIndices(), statevec_multiControlledMultiRotatePauli(), validateMultiControlsMultiTargets(), and validatePauliCodes().

Referenced by TEST_CASE().

Figure 7.6: Example QuEST documentation (divided into two columns) of the function `multiControlledMultiRotatePauli` (link). This is the example for which the input validation was discussed in Section 7.7.

```
?DrawCircuit
```

Symbol

DrawCircuit[circuit] generates a circuit diagram. The circuit can contain symbolic parameters.

DrawCircuit[circuit, numQubits] generates a circuit diagram
with numQubits, which can be more or less than that inferred from the circuit.

DrawCircuit[{circ1, circ2, ...}] draws the total circuit, divided
into the given subcircuits. This is the output format of GetCircuitColumns[].

DrawCircuit[{{t1, circ1}, {t2, circ2}, ...}] draws the total circuit, divided into the given subcircuits,
labeled by their scheduled times {t1, t2, ...}. This is the output format of GetCircuitSchedule[].

DrawCircuit[{{t1, A1,A2}, {t2, B1,B2}, ...}] draws the total circuit, divided into subcircuits {A1 A2, B1 B2,
...}, labeled by their scheduled times {t1, t2, ...}. This is the output format of InsertCircuitNoise[].

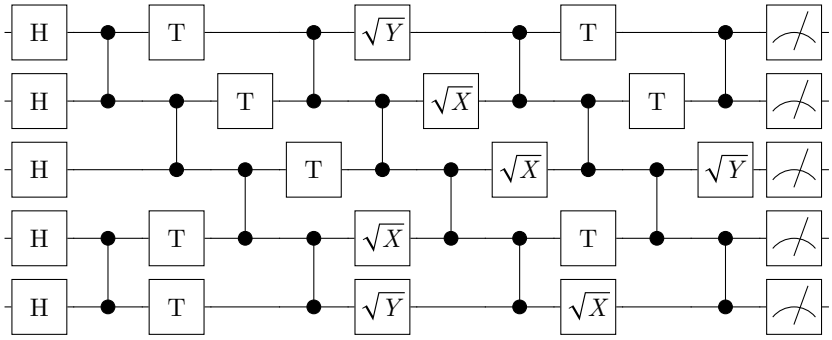
DrawCircuit accepts optional arguments Compactify, DividerStyle, SubcircuitSpacing, SubcircuitLabels, LabelDrawer
and any Graphics option. For example, the fonts can be changed with 'BaseStyle –> {FontFamily –> "Arial"}'.

Figure 7.7: Example QuESTlink documentation, accessed directly within a Mathematica notebook

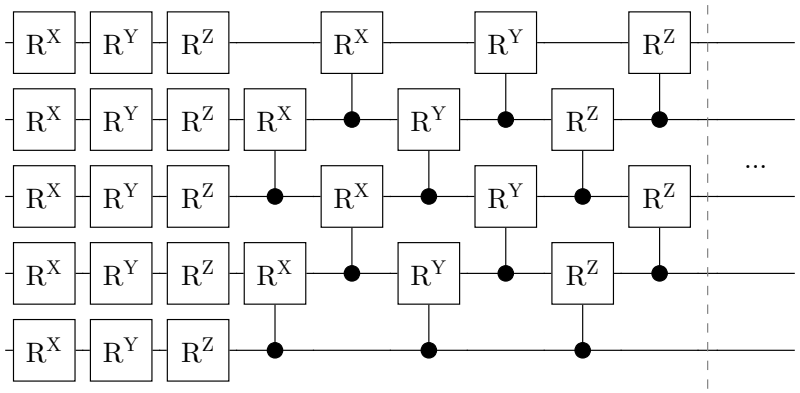
7.10 Benchmarking

This section collates the results of QuEST and QuESTlink benchmarking from the author's works of Reference [249] and [112]. This involves repeated timed simulation of the circuits of Figure 7.8 with varying qubit numbers (up to 38) and gate depths, until the results show an acceptable variance. In addition to the basic runtime plots of Figures 7.10, 7.12 and 7.13, we also present *weak-* and *strong-scaling* in Figures 7.9 and 7.11. Strong-scaling measures the relative speedup as the degree of parallelisation (e.g. the number of threads or distributed nodes) is increased, and ergo captures the parallelisability of the simulation (as in-part informed by Amdahl's law [319]). Good strong-scaling is evidenced by a unity-gradient straight-line. Weak-scaling shows the slowdown as the degree of parallelisation increases proportionally to an increase of simulation difficulty; in our scenario, as the simulated register gains one qubit while *twice* as many distributed nodes are employed. Good weak-scaling is evidenced by a flat line.

Our tests exhibit exceptional strong and weak scaling, in all of multithreaded, GPU-accelerated and distributed settings, implying excellent utilisation of the deployed hardware. They include competitive comparisons to multithreaded ProjectQ [263], although we note the latter shows superior serial performance through a variety of circuit-level algorithmic optimisations. Moving from 34 to 37 simulated qubits (16 to 128 nodes) slows distributed QuEST by a mere 9% in comparison to the 148% slowdown of qHipster [291], an alternative distributed library maintained by Intel. Communication induced a $\approx 10^1$ slowdown, in contrast to a 10^6 slowdown reported in distributed tests of other tools [320]. Our final results show that the Mathematica layer in QuESTlink invokes only a minor and often negligible performance overhead to that of core QuEST.

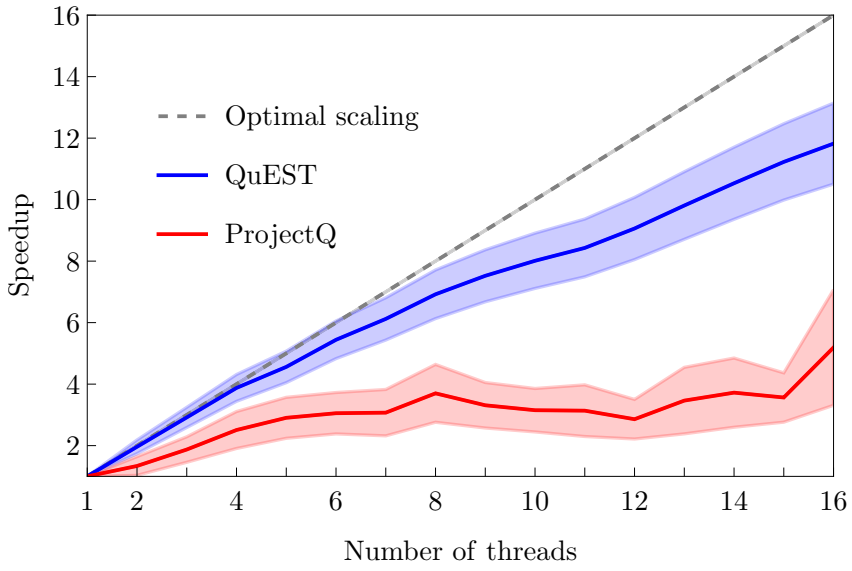


(a) A 5-qubit depth-10 example of the random circuit used to benchmark QuEST (of the linear topology described in Ref. [288]). For measure, a depth 100 circuit of 30 qubits features 1020 single qubit gates and 967 controlled phase gates.

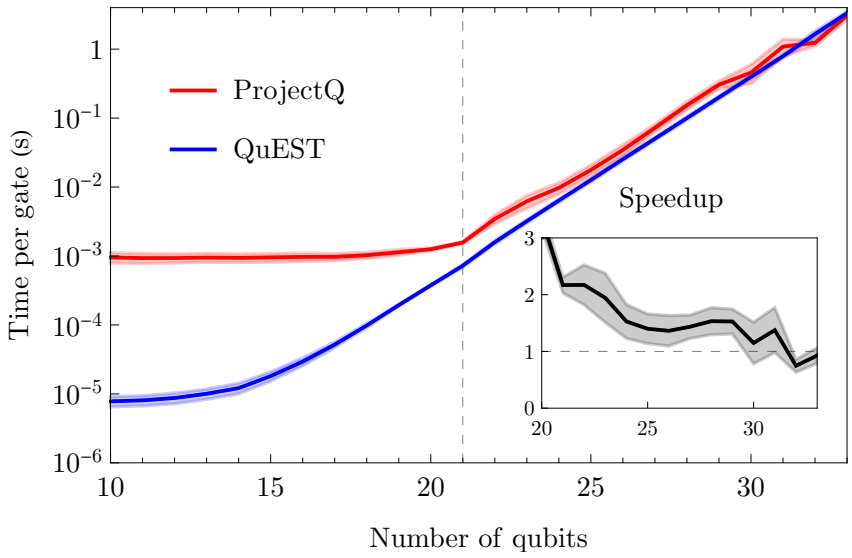


(b) A 5-qubit 1-repetition example of the arbitrary circuit used for comparative benchmarking of QuESTlink and QuEST. Every gate involves a rotation of a uniformly random angle between 0 and 4π, which generate all rotations. The 15-qubit circuits employed for benchmarking feature 87 gates per repetition.

Figure 7.8: The arbitrary circuits simulated in benchmarking of QuEST (top) and QuESTlink (bottom).



(a) Strong-scaling (30 qubits)



(b) Absolute runtime (16 threads)

Figure 7.9: The performance of multithreaded QuEST v0.10 [249] and ProjectQ v0.3.5 [263], simulating the random circuits of Fig. 7.8a. Solid lines and shaded regions indicate the mean and a standard deviation either side (respectively) of ∼7 k simulations of circuit depths between 10 and 100. The dashed line on the right indicates indicates when the statevector has filled the L3 cache.

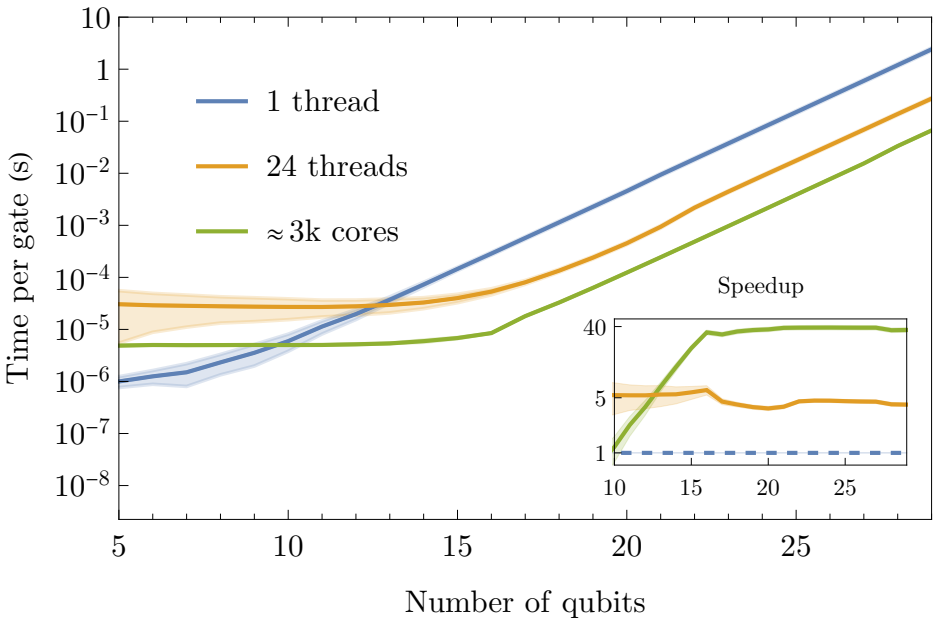
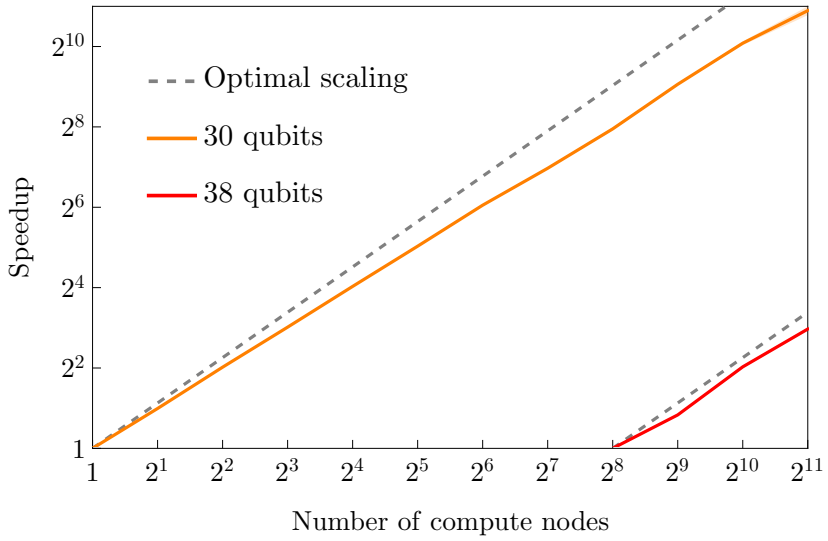
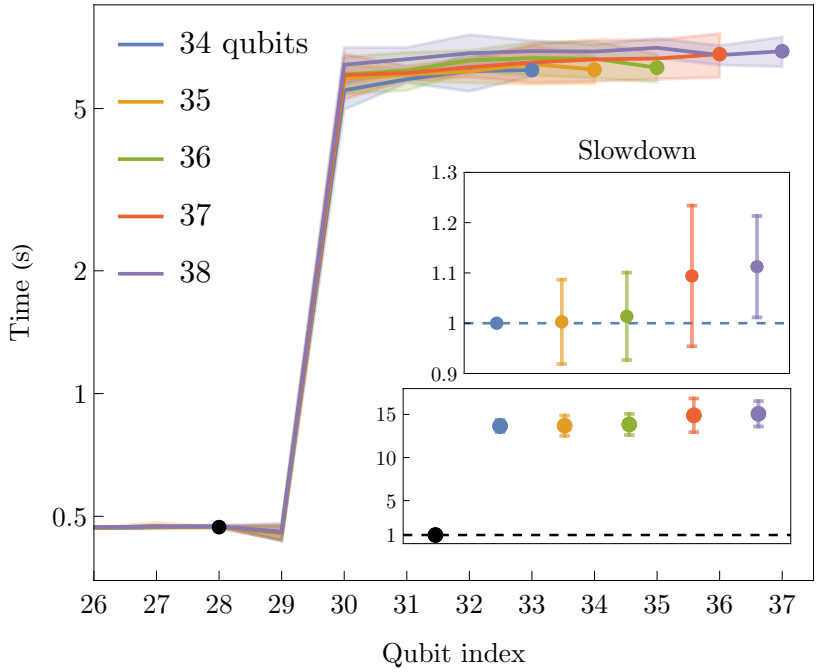


Figure 7.10: Benchmarked QuEST performance of serial (blue), multithreaded (orange) and GPU-accelerated (green) simulation of random circuits (Fig. 7.8a).



(a) Strong-scaling



(b) Weak-scaling

Figure 7.11: Performance of distributed QuEST on the ARCHER supercomputer [321] with 24-thread 2-socket 64 GiB nodes. (a) simulates the random circuits of Fig. 7.8a. (b) simulates single qubit gates distributed on $\{16, 32, 64, 128, 256\}$ respective nodes (i.e. 2^k for increasing k). Communication occurs for qubits at indices ≥ 30. Recall that good weak-scaling is characterised by unchanged runtime (flat lines, when communicating) as the problem difficulty increases proportionally to the computing resources, as shown. The bottom subplot shows the slowdown caused by communication, while the top subplot shows the slowdown of rotating the final (communicated) qubit as the total number of qubits simulated increases.

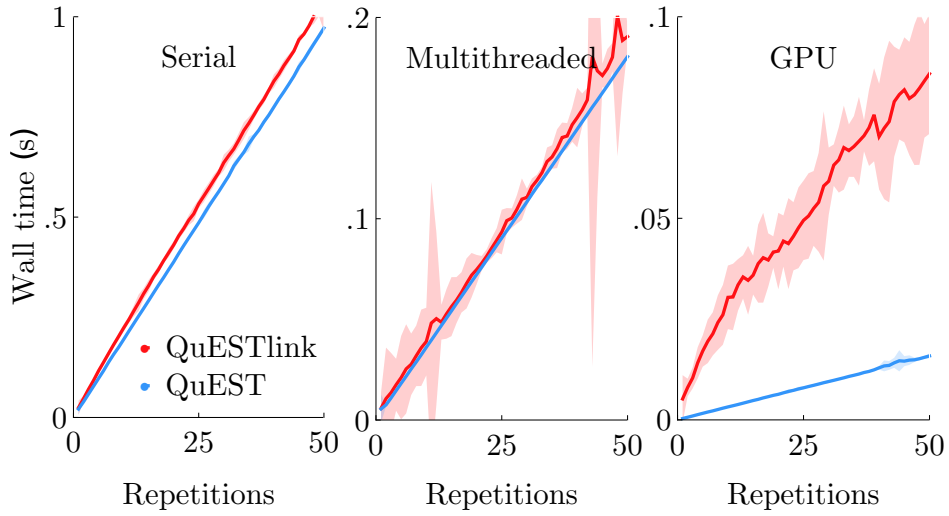


Figure 7.12: Comparitive performance of QuEST and (local) QuESTlink emulating the arbitrary 15-qubit circuit of Fig. 7.8b with a varying circuit depth (repetitions of the base circuit), in each of QuESTlink's supported parallelisation modes. Multithreaded emulation used 8 of 12 available threads, though were shared with the Mathematica kernel. Solid lines indicate the mean runtime of 10 random simulations, and shaded regions indicate three standard deviations. The circuit of 50 repetitions contains a total of 4350 gates.

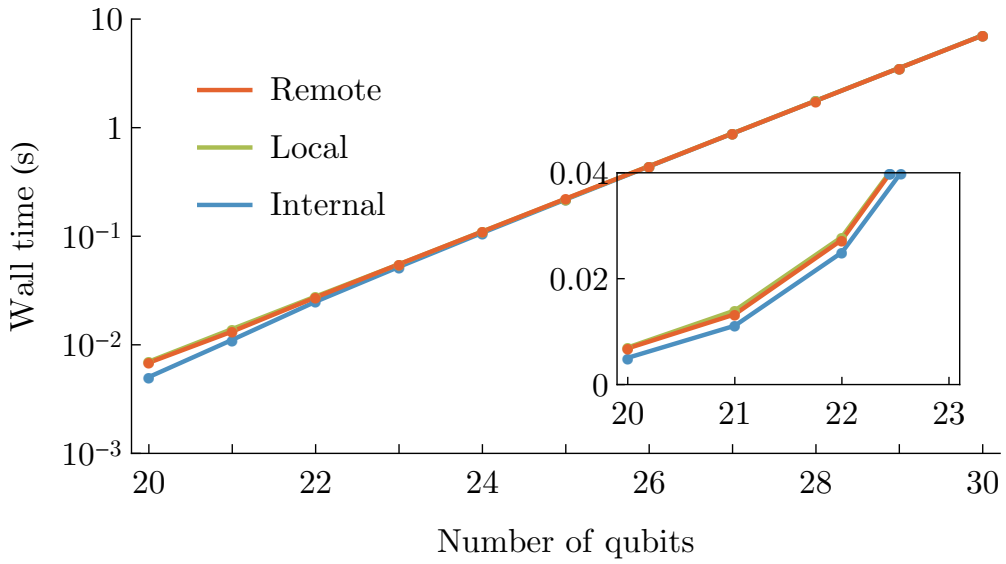


Figure 7.13: Benchmarking of a fixed-length circuit (Fig. 7.8b) simulation in QuESTlink, using different environment configurations, though all ultimately running on the GPU from Fig. 7.12. *Remote* used `CreateRemoteQuESTEnv[]` to connect from a 13-inch 2017 MacBook Pro, with an average latency of 0.38 ms, with wall time measured in Mathematica with `AbsoluteTiming[]`. *Local* ran in a notebook on the Xeon machine, also using `AbsoluteTiming[]`. *Internal* measured the time spent in the QuESTlink `C++` backend, and so excludes any Mathematica and WSTP overheads. Each datum is the average of 50 tests with re-randomised gate parameters.

7.11 Discussion

This chapter has neglected the more mundane tasks of maintaining software such as those of version control, managing external contributions (example), authoring release notes (example), bug triage (example) and communicating with the user base. Instead, this chapter motivated the founding principles and facilities of QuEST and QuESTlink and their manifestation in the software architecture and interface design. We emphasised the responsibility of scientific codes to perform rigorous user input validation and unit testing, and demonstrated how this can be done with software platforms like Github. We then showcased the excellent benchmarked performance of the software.

As will by now be self-evident, QuEST and QuESTlink were instrumental for the research in this thesis. They have furthermore emerged as valuable software tools for the wider research community, and attracted laundry lists of requested features. The author expects to contribute further to these projects, to maintain them, to update them with new algorithmic innovations, and to streamline the transfer of these projects to their next generation of developers.

The list of future features for QuEST include:

- Tools to automatically optimally build and configure QuEST on candidate hardware. For example, choosing the number of `OpenMP` threads and `MPI` processes based on the number of CPU cores and sockets, or the `CUDA` parallelisation granularity based on the GPU specification.

- A refactored GPU backend for tight integration with NVIDIA's recent cuQuantum SDK [322].

- Support for multiple GPUs, locally connected via NVLink [323] on IBM Power9

platforms [324]. Further, support for multiple GPUs between distributed machines, potentially using emerging GPU architectures with embedded networks cards [325] or high-performance abstractions [326].

- Incorporation of recent algorithmic optimisations such as adaptive polar encodings of complex amplitudes [262], and adaptive compression of density matrices during channel simulation [327].

Chapter 8

Conclusion

Quantum computers beckon toward a revolution in computation. Precisely when this revolution will occur remains a matter of intense debate, argued between pessimistic experimentalists and optimistic corporate interests. In the interim before error-corrected fault-tolerant practical-scale universal quantum computers, it is prudent to scrutinise the utility of their near-future non-corrected architecturally-constrained predecessors. Quantum algorithms carefully designed to be inherently resilient to the imperfections and limitations of these devices may unlock their power. If so, near-future quantum computers may prove to be more than mere prototypes intended to sustain the patience of investors during the journey to the error correction threshold. They may instead become powerful non-universal machines capable of accelerating particular scientific simulations, like those of quantum chemistry. This thesis sought to advance this endeavour.

We developed three novel quantum algorithms which seek to harness the potential power of first generation quantum computers. Chapter 2 presented variational imaginary-time

evolution [163], a quantum adaptation of the system-cooling imaginary-time method ubiquitous in classical simulations of quantum many-body physics. We showed variational imaginary-time evolution was a superior minimisation routine to the state of the art, at the time of its development, and demonstrated its utility in quantum chemistry applications. We proved an equivalence to the subsequently developed quantum natural gradient method [133], and proposed a relevant improvement to Li's real-time algorithm [132].

Chapter 3 presented an efficient quantum algorithm to re-purporse quantum minimisers in order to diagonalise classically intractable Hamiltonians [194]. That is, to iteratively discover the energy eigenvalues and ansatz parameters which generate close approximations to the eigenstates. We demonstrated promising performance of our method upon chemistry problems, studied through classical simulation, and made additional revelations about the convergence behaviour of imaginary-time evolution when used as a variational minimiser. We highlighted the utility of our scheme for cheap augmentation of spectra in variational methods like real-time simulation.

Chapter 4 presented a variational quantum algorithm for approximately recompiling and optimising quantum circuits [113], with particular focus on variational ansätze. This is an important task for adapting classically intractable circuits for candidate quantum hardware. Through classical simulation, we demonstrated an impressive noise resilience of our algorithm and illustrated its potential application in extending real-time simulation. We also devised meta-algorithms for adapting the time-step and luring minimisers toward global convergence, useful for wider variational routines.

Throughout these works, we argued the necessity of strong classical simulation, meanwhile lamenting the unintuitively steep resource costs of simulating quantum variational routines. Chapter 5 clarified these costs, and developed two novel classical algorithms

for asymptotically faster simulation. These enable researchers to investigate quantum gradient descent [118], quantum *natural* gradient [133] and variational imaginary-time evolution [163] in significantly larger settings than previously possible. Both schemes have since been implemented in IBM's Qiskit simulator [239].

Chapter 6 introduced a plethora of classical high-performance computing techniques relevant to the simulation of quantum computers [249]. This included low-level CPU heuristics, multithreading, GPU-acceleration and distribution, and platform-agnostic algorithms to significantly broaden simulator facilities. The chapter gave only examples of the substantial computational efforts made throughout the author's doctorate, compromising fourteen novel asymptotically-improved distributed algorithms, and equally as many implementation and algorithmic optimisations on various hardware platforms.

Chapter 7 presented two software projects which emerged from these efforts: the Quantum Exact Simulation Toolkit (QuEST) [249] in `C` and `C++`, and QuESTlink [112] in Mathematica. The facilities, interfaces and architectures of these projects were motivated by well-known software development practices, and the chapter established somewhat of a criterion for useful quantum simulators. These projects have seen immense engagement with the scientific community and have become integrated into several notable software stacks.

Through these works, the author aspires to have made some assistance to the research community and progress toward achieving quantum advantage. We hope the distant world which realises practical quantum computation will be one worthy to receive it: a world of societal reform; of equity; of mitigated climate disaster; of social justice; and of the same generosity to enable the disadvantaged to assume their duties in science.

Bibliography

[1] Richard P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21:467–488, June 1982.

[2] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[3] Adam Smith, MS Kim, Frank Pollmann, and Johannes Knolle. Simulating quantum many-body dynamics on a current digital quantum computer. *npj Quantum Information*, 5(1):1–13, 2019.

[4] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[5] Quantum computing market with COVID-19 impact - global forecast to 2026, Feb 2021.

[6] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[7] Clare Horsman, Susan Stepney, Rob C Wagner, and Viv Kendon. When does a physical system compute? *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 470(2169):20140182, 2014.

[8] Igor L Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147–154, 2014.

[9] Eric Chitambar and Gilad Gour. Quantum resource theories. *Reviews of Modern Physics*, 91(2):025001, 2019.

[10] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[11] John Von Neumann. First draft of a report on the EDVAC. *Pennsylvania: University of Pennsylvania*, 1945.

[12] Herman Lukoff. *From Dits to Bits: A personal history of the electronic computer*. Robotics Press, 1979.

[13] F. C. Williams and T. Kilburn. Electronic digital computers. *Nature*, 162(4117):487, 1948.

[14] Richard W. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J.*, 29:147–160, 1950.

[15] Michael Oser Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. *Technical Report 2*, 1960.

[16] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306,

1965.

[17] Alan Cobham. The Intrinsic Computational Difficulty of Functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964*, Amsterdam, 1965. North-Holland.

[18] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[19] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[20] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[21] William I. Gasarch. Guest column: The P=? NP poll. *Complexity Theory Column*, 36, 2002.

[22] Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of NP-completeness (series of books in the mathematical sciences), ed. *Computers and Intractability*, page 340, 1979.

[23] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[24] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[25] John D Dixon. Asymptotically fast factorization of integers. *Mathematics of computation*, 36(153):255–260, 1981.

[26] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[27] Paul Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University*, 1974.

[28] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[29] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[30] Richard Feynman. There's plenty of room at the bottom. *Journal of Microelectromechanical Systems, American Physical Society Meeting*, 2:1, 1993.

[31] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[32] John Stewart Bell. On the Einstein Podolsky Rosen paradox. *Physics*, 1(3):195–200, 1964.

[33] R P Poplavski. Thermodynamic models of information processes. *Soviet Physics Uspekhi*, 18(3):222, 1975.

[34] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[35] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A*, 400(1818):97–117, 1985.

[36] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 439(1907):553–558, 1992.

[37] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 454, pages 339–354. The Royal Society, 1998.

[38] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Nov 1994.

[39] Benjamin Schumacher. Quantum coding. *Physical Review A*, 51(4):2738, 1995.

[40] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[41] H Dieter Zeh. On the interpretation of measurement in quantum theory. *Foundations of Physics*, 1(1):69–76, 1970.

[42] G Massimo Palma, Kalle-Antti Suominen, and Artur K Ekert. Quantum computers and dissipation. *Proc. R. Soc. Lond. A*, 452(1946):567–584, 1996.

[43] Serge Haroche and Jean-Michel Raimond. Quantum computing: dream or nightmare? *Physics Today*, 49(8):51–54, 1996.

[44] Christopher Monroe and David Wineland. Future of quantum computing proves to be debatable. *Physics Today*, 49:107–108, 1996.

[45] Peter W. Shor. Scheme for reducing decoherence in quantum memory. *Phys. Rev. A 52*, 52:2493–2496, 1995.

[46] Andrew M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77(5):793–797, Jul 1996.

[47] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996.

[48] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane. Quantum error correction and orthogonal geometry. *Phys. Rev. Lett.*, 78(3):405–408, Jan 1997.

[49] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum hamming bound. *Phys. Rev. A*, 54:1862–1868, Sep 1996.

[50] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. PhD thesis, California Institute of Technology, 1997.

[51] David P. DiVincenzo and Peter W. Shor. Fault-tolerant error correction with efficient quantum codes. *Phys. Rev. Lett.*, 77(15):3260–3263, Oct 1996.

[52] Alexi Kitaev. Quantum error correction with imperfect gates. In *Proceeding of the Third International Conference on Quantum Communication and Measurement*, 1997.

[53] John Preskill. Fault-tolerant quantum computation. In *Introduction to Quantum Computation*. World Scientific, 1998.

[54] Emanuel Knill, Raymond Laflamme, and W Zurek. Threshold accuracy for quantum computation. *arXiv preprint quant-ph/9610011*, 1996.

[55] Christof Zalka. Threshold estimate for fault tolerant quantum computation. *arXiv preprint quant-ph/9612028*, 1996.

[56] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188. ACM, 1997.

[57] Emanuel Knill, Raymond Laflamme, and Wojciech Zurek. Resilient quantum computation: Error models and thresholds. *Proc. R. Soc. London, Ser. A*, 454:365–384, 1998.

[58] Andrew M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Phys. Rev. Lett.*, 78(11):2252–2255, Mar 1997.

[59] Andrew M. Steane. Space, time, parallelism and noise requirements for reliable quantum computing. *Fortsch. Phys.*, 46:443, 1998.

[60] Daniel Gottesman and Isaac L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402:390–393, 1999.

[61] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

[62] Michael Reck, Anton Zeilinger, Herbert J. Bernstein, and Philip Bertani. Experimental realization of any discrete unitary operator. *Phys. Rev. Lett.*, 73(1):58–61, Jul 1994.

[63] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52(5):3457–3467, Nov 1995.

[64] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan. On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for Shor's basis. In *FOCS 1999: 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 486–494, New York, NY, October 1999.

[65] Alexi Yu Kitaev. Quantum computations: algorithms and error correction. *Russ. Math. Surv.*, 52(6):1191–1249, 1997.

[66] Alexi Yu Kitaev. Quantum measurements and the Abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[67] Lisa Hales and Sean Hallgren. An improved quantum Fourier transform algorithm and applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 515–525. IEEE, 2000.

[68] AB Finnila, MA Gomez, C Sebenik, C Stenson, and JD Doll. Quantum annealing: a new method for minimizing multidimensional functions. *Chemical physics letters*, 219(5-6):343–348, 1994.

[69] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355, 1998.

[70] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[71] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[72] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292(5516):472–475, 2001.

[73] Jérémie Roland and Nicolas J. Cerf. Quantum search by local adiabatic evolution. *Phys. Rev. A*, 65:042308, Mar 2002.

[74] Wim Van Dam, Michele Mosca, and Umesh Vazirani. How powerful is adiabatic quantum computation? In *Proceedings 42nd IEEE symposium on foundations of computer science*, pages 279–287. IEEE, 2001.

[75] Dorit Aharonov, Wim Van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM review*, 50(4):755–787, 2008.

[76] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 2003.

[77] Edward Farhi and Sam Gutmann. Quantum computation and decision trees. *Physical Review A*, 58(2):915, 1998.

[78] Dan Ventura and Tony Martinez. Quantum associative memory. *Information Sciences*, 124(1-4):273–296, 2000.

[79] Seth Lloyd. Quantum algorithm for solving linear systems of equations. In *APS March Meeting Abstracts*, 2010.

[80] Ashley Montanaro. Quantum speedup of Monte Carlo methods. *Proc. R. Soc. A*, 471(2181):20150301, 2015.

[81] Joel Achenbach. Not just waiting for Y2K to come. *The Washington Post*, page A1, 1998.

[82] David P DiVincenzo. Quantum computation. *Science*, 270(5234):255–261, 1995.

[83] Jonathan A Jones, Michele Mosca, and Rasmus H Hansen. Implementation of a quantum search algorithm on a quantum computer. *Nature*, 393(6683):344, 1998.

[84] Danilo Boschi, Salvatore Branca, Francesco De Martini, Lucien Hardy, and Sandu Popescu. Experimental realization of teleporting an unknown pure quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 80(6):1121, 1998.

[85] Harald Weinfurter. Experimental bell-state analysis. *EPL (Europhysics Letters)*, 25(8):559, 1994.

[86] David G Cory, MD Price, W Maas, E Knill, Raymond Laflamme, Wojciech H Zurek, Timothy F Havel, and SS Somaroo. Experimental quantum error correction. *Physical Review Letters*, 81(10):2152, 1998.

[87] Lorenza Viola and Seth Lloyd. Dynamical suppression of decoherence in two-state quantum systems. *Physical Review A*, 58(4):2733, 1998.

[88] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883, 2001.

[89] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L O'brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nature Photonics*, 6(11):773, 2012.

[90] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. In *Annual International Cryptology Conference*, pages 62–91. Springer, 2020.

[91] Wolfgang Pauli. Electromagnetic traps for charged and neutral particles. *Reviews of Modern Physics*, 62(3):531, 1990.

[92] Juan I Cirac and Peter Zoller. Quantum computations with cold trapped ions. *Physical Review Letters*, 74(20):4091, 1995.

[93] Thomas Monz, Philipp Schindler, Julio T Barreiro, Michael Chwalla, Daniel Nigg, William A Coish, Maximilian Harlander, Wolfgang Hänsel, Markus Hennrich, and Rainer Blatt. 14-qubit entanglement: Creation and coherence. *Physical Review Letters*, 106(13):130506, 2011.

[94] Matteo Mariantoni, Haohua Wang, Tsuyoshi Yamamoto, Matthew Neeley, Radoslaw C Bialczak, Yu Chen, Michael Lenander, Erik Lucero, Aaron D OConnell, Daniel Sank, et al. Implementing the quantum von neumann architecture with superconducting circuits. *Science*, 334(6052):61–65, 2011.

[95] Zhengbing Bian, Fabian Chudak, William G Macready, Lane Clark, and Frank Gaitan. Experimental determination of Ramsey numbers. *Physical Review Letters*, 111(13):130505, 2013.

[96] R Barends, J Kelly, A Megrant, A Veitia, D Sank, E Jeffrey, TC White, J Mutus, AG Fowler, B Campbell, et al. Logic gates at the surface code threshold: Superconducting qubits poised for fault-tolerant quantum computing. *arXiv preprint arXiv:1402.4848*, 2014.

[97] Paul Adrien Maurice Dirac et al. *The principles of quantum mechanics*. Number 27. Oxford University Press, 1981.

[98] Abraham Pais. Max Born's statistical interpretation of quantum mechanics. *Science*, 218(4578):1193–1198, 1982.

[99] E Noether. Invariante variationsprobleme. *Nachrichen von der Knigliche Gesellschaft der*, 1918.

[100] Vladyslav Verteletskyi, Tzu-Ching Yen, and Artur F Izmaylov. Measurement optimization in the variational quantum eigensolver using a minimum clique cover. *The Journal of chemical physics*, 152(12):124114, 2020.

[101] Artur F Izmaylov, Tzu-Ching Yen, Robert A Lang, and Vladyslav Verteletskyi. Unitary partitioning approach to the measurement problem in the variational quantum eigensolver method. *Journal of chemical theory and computation*, 16(1):190–195, 2019.

[102] Ophelia Crawford, Barnaby van Straaten, Daochen Wang, Thomas Parks, Earl Campbell, and Stephen Brierley. Efficient quantum measurement of pauli operators in the presence of finite sampling error. *Quantum*, 5:385, 2021.

[103] Michael Beverland, Earl Campbell, Mark Howard, and Vadym Kliuchnikov. Lower bounds on the non-Clifford resources for quantum computations. *Quantum Science and Technology*, 5(3):035009, 2020.

[104] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. Characterizing quantum gates via randomized benchmarking. *Physical Review A*, 85(4):042311, 2012.

[105] Alexei Yu Kitaev, Alexander Shen, Mikhail N Vyalyi, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47. American Mathematical Soc., 2002.

[106] Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Physical Review*, 28(6):1049, 1926.

[107] Masuo Suzuki. Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations. *Physics Letters A*, 146(6):319–323, 1990.

[108] Naomichi Hatano and Masuo Suzuki. Finding exponential product formulas of higher orders. In *Quantum annealing and other optimization methods*, pages 37–68. Springer, 2005.

[109] Ian D Kivlichan, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Wei Sun, Zhang Jiang, Nicholas Rubin, Austin Fowler, Alán Aspuru-Guzik, et al. Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via trotterization. *Quantum*, 4:296, 2020.

[110] Andrew M Childs, Aaron Ostrander, and Yuan Su. Faster quantum simulation by randomization. *Quantum*, 3:182, 2019.

[111] Earl Campbell. Random compiler for fast Hamiltonian simulation. *Physical Review Letters*, 123(7):070503, 2019.

[112] Tyson Jones and Simon C Benjamin. QuESTlink – Mathematica embiggened by a hardware-optimised quantum emulator. *Quantum Science and Technology*, 2020.

[113] Tyson Jones and Simon C Benjamin. Robust quantum compilation and circuit optimisation via energy minimisation. *Quantum*, 6:628, 2022.

[114] Zoë Holmes, Kunal Sharma, Marco Cerezo, and Patrick J Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1):010313, 2022.

[115] Suguru Endo, Simon C. Benjamin, and Ying Li. Practical quantum error mitigation for near-future applications. *Phys. Rev. X*, 8:031027, Jul 2018.

[116] Suguru Endo, Zhenyu Cai, Simon C Benjamin, and Xiao Yuan. Hybrid quantum-classical algorithms and quantum error mitigation. *Journal of the Physical Society of Japan*, 90(3):032001, 2021.

[117] Armands Strikis, Dayue Qin, Yanzhu Chen, Simon C. Benjamin, and Ying Li. Learning-based quantum error mitigation. *PRX Quantum*, 2:040330, Nov 2021.

[118] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L Obrien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5, 2014.

[119] Leonhard Euler. *Institutionum calculi integralis volumen primum...*, volume 2. 1769.

[120] Jun Li, Xiaodong Yang, Xinhua Peng, and Chang-Pu Sun. Hybrid quantum-classical approach to quantum optimal control. *Physical Review Letters*, 118(15):150503, 2017.

[121] Gavin E Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv preprint arXiv:1905.13311*, 2019.

[122] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, 2022.

[123] Andrew Arrasmith, Marco Cerezo, Piotr Czarnik, Lukasz Cincio, and Patrick J Coles. Effect of barren plateaus on gradient-free optimization. *Quantum*, 5:558, 2021.

[124] Samson Wang, Enrico Fontana, Marco Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature Communications*, 12(1):1–11, 2021.

[125] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, 2019.

[126] Stefan H. Sack, Raimel A. Medina, Alexios A. Michailidis, Richard Kueng, and Maksym Serbyn. Avoiding barren plateaus using classical shadows. *PRX Quantum*, 3:020365, Jun 2022.

[127] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost

function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1–12, 2021.

[128] Kristan Temme, Tobias J Osborne, Karl G Vollbrecht, David Poulin, and Frank Verstraete. Quantum metropolis sampling. *Nature*, 471(7336):87–90, 2011.

[129] Man-Hong Yung and Alán Aspuru-Guzik. A quantum–quantum metropolis algorithm. *Proceedings of the National Academy of Sciences*, 109(3):754–759, 2012.

[130] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. Entanglement-induced barren plateaus. *PRX Quantum*, 2(4):040316, 2021.

[131] Xia Liu, Geng Liu, Jiaxin Huang, and Xin Wang. Mitigating barren plateaus of variational quantum eigensolvers. *arXiv preprint arXiv:2205.13539*, 2022.

[132] Ying Li and Simon C. Benjamin. Efficient variational quantum simulator incorporating active error minimization. *Phys. Rev. X*, 7:021050, Jun 2017.

[133] James Stokes, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. Quantum natural gradient. *Quantum*, 4:269, 2020.

[134] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[135] Alfred Shapere and Frank Wilczek. *Geometric phases in physics*, volume 5. World scientific, 1989.

[136] Guido Fubini. *Sulle metriche definite da una forma hermitiana: nota*. Office graf. C. Ferrari, 1904.

[137] Eduard Study. Kürzeste wege im komplexen gebiet. *Mathematische Annalen*, 60(3):321–378, 1905.

[138] RA Fisher. Accuracy of observation, a mathematical examination of the methods of determining, by the mean error and by the mean square error. *Monthly Notices of the Royal Astronomical Society*, 80:758–770, 1920.

[139] Ya Wang, Florian Dolde, Jacob Biamonte, Ryan Babbush, Ville Bergholm, Sen Yang, Ingmar Jakobi, Philipp Neumann, Alán Aspuru-Guzik, James D Whitfield, et al. Quantum simulation of helium hydride cation in a solid-state spin register. *ACS nano*, 9(8):7769–7774, 2015.

[140] P. J. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, B Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Jeffrey, E. Lucero, A. Megrant, J. Y. Mutus, M. Neeley, C. Neill, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, P. V. Coveney, P. J. Love, H. Neven, A. Aspuru-Guzik, and J. M. Martinis. Scalable quantum simulation of molecular energies. *Phys. Rev. X*, 6:031007, Jul 2016.

[141] Yangchao Shen, Xiang Zhang, Shuaining Zhang, Jing-Ning Zhang, Man-Hong Yung, and Kihwan Kim. Quantum implementation of the unitary coupled cluster for simulating molecular electronic structure. *Phys. Rev. A*, 95:020501, Feb 2017.

[142] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Aln Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[143] S. Paesani, A. A. Gentile, R. Santagati, J. Wang, N. Wiebe, D. P. Tew, J. L.

O'Brien, and M. G. Thompson. Experimental Bayesian quantum phase estimation on a silicon photonic chip. *Phys. Rev. Lett.*, 118:100503, Mar 2017.

[144] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[145] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[146] Tao Shi, Eugene Demler, and J. Ignacio Cirac. Variational study of fermionic and bosonic systems with non-Gaussian states: Theory and applications. *Annals of Physics*, 390:245 – 302, 2018.

[147] M. H. Poincaré. Sur la dynamique de l'électron. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 21(1):129–175, Dec 1906.

[148] Siu A Chin, S Janecek, and E Krotscheck. Any order imaginary time propagation method for solving the Schrödinger equation. *Chemical Physics Letters*, 470(4-6):342–346, 2009.

[149] Philipp Bader, Sergio Blanes, and Fernando Casas. Solving the Schrödinger eigenvalue problem by the imaginary time propagation technique using splitting methods with complex coefficients. *The Journal of chemical physics*, 139(12):124117, 2013.

[150] G. C. Wick. Properties of Bethe-Salpeter wave functions. *Phys. Rev.*, 96:1124–1134, Nov 1954.

[151] F. Verstraete, J. J. García-Ripoll, and J. I. Cirac. Matrix product density operators: Simulation of finite-temperature and dissipative systems. *Phys. Rev. Lett.*, 93:207204, Nov 2004.

[152] Michael Zwolak and Guifré Vidal. Mixed-state dynamics in one-dimensional quantum lattice systems: A time-dependent superoperator renormalization algorithm. *Phys. Rev. Lett.*, 93:207205, Nov 2004.

[153] F. Alexander Wolf, Ara Go, Ian P. McCulloch, Andrew J. Millis, and Ulrich Schollwöck. Imaginary-time matrix product state impurity solver for dynamical mean-field theory. *Phys. Rev. X*, 5:041032, Nov 2015.

[154] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum Monte Carlo simulations of solids. *Rev. Mod. Phys.*, 73:33–83, Jan 2001.

[155] Romain Dubessy, Camilla De Rossi, Thomas Badr, Laurent Longchambon, and Hélene Perrin. Imaging the collective excitations of an ultracold gas using statistical correlations. *New Journal of Physics*, 16(12):122001, 2014.

[156] Monte Carlo methods for impurity physics in ultracold Bose quantum gases, author=Ardila, L. A. P., journal=Nature Reviews Physics, pages=1–1, year=2022, publisher=Nature Publishing Group.

[157] Bo Zhang. Quantum turbulence in two dimensional Bose-Einstein condensates. *PhD thesis, The College of William and Mary*, 2011.

[158] Tyson Jones. Detecting vortex clusters with principal component analysis. *Honours thesis, Monash University, School of Physics and Astronomy*, 2016.

[159] I. M. Georgescu, S. Ashhab, and Franco Nori. Quantum simulation. *Rev. Mod. Phys.*, 86:153–185, Mar 2014.

[160] Ying Li and Simon C Benjamin. Efficient variational quantum simulator incorporating active error minimization. *Physical Review X*, 7(2):021050, 2017.

[161] Nelson Dunford and Jacob T Schwartz. *Linear operators, part 1: general theory*, volume 10. John Wiley & Sons, 1988.

[162] A. D. McLachlan. A variational solution of the time-dependent Schrödinger equation. *Molecular Physics*, 8(1):39–44, 1964.

[163] Sam McArdle, Tyson Jones, Suguru Endo, Ying Li, Simon C Benjamin, and Xiao Yuan. Variational ansatz-based quantum simulation of imaginary time evolution. *npj Quantum Information*, 5(1):1–6, 2019.

[164] Artur K Ekert, Carolina Moura Alves, Daniel KL Oi, Michał Horodecki, Paweł Horodecki, and Leong Chuan Kwek. Direct estimations of linear and nonlinear functionals of a quantum state. *Physical Review Letters*, 88(21):217901, 2002.

[165] Kosuke Mitarai and Keisuke Fujii. Methodology for replacing indirect measurements with direct measurements. *Physical Review Research*, 1(1):013006, 2019.

[166] Andrei Nikolaevich Tikhonov, AV Goncharsky, VV Stepanov, and Anatoly G Yagola. *Numerical methods for the solution of ill-posed problems*, volume 328. Springer Science & Business Media, 1995.

[167] D Calvetti, S Morigi, L Reichel, and F Sgallari. Tikhonov regularization and the L-curve for large discrete ill-posed problems. *Journal of computational and applied mathematics*, 123(1-2):423–446, 2000.

[168] Zhu Nan-hai and Zhao Xiao-hua. Optimal calculation of Tikhonov regularization parameter based on genetic algorithm. , 26(5):25–030, 2009.

[169] Eitan Levin and Alexander Y Meltzer. Estimation of the regularization parameter in linear discrete ill-posed problems using the Picard parameter. *SIAM Journal on Scientific Computing*, 39(6):A2741–A2762, 2017.

[170] Hua Guo, Guolin Liu, and Luyao Wang. An improved Tikhonov-regularized variable projection algorithm for separable nonlinear least squares. *Axioms*, 10(3):196, 2021.

[171] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[172] Xiao Yuan, Suguru Endo, Qi Zhao, Ying Li, and Simon C Benjamin. Theory of variational quantum simulation. *Quantum*, 3:191, 2019.

[173] Michael Victor Berry. Quantal phase factors accompanying adiabatic changes. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 392(1802):45–57, 1984.

[174] Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of computation*, 51(184):699–706, 1988.

[175] Pierre Simon de Laplace. *Théorie analytique des probabilités*, volume 7. Courcier, 1820.

[176] George E. P. Box. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29:610–611, 1958.

[177] Jacob T. Seeley, Martin J. Richard, and Peter J. Love. The Bravyi-Kitaev transformation for quantum computation of electronic structure. *The Journal of Chemical Physics*, 137(22):224109, 2012.

[178] Cornelius Hempel, Christine Maier, Jonathan Romero, Jarrod McClean, Thomas Monz, Heng Shen, Petar Jurcevic, Ben P. Lanyon, Peter Love, Ryan Babbush, Alán Aspuru-Guzik, Rainer Blatt, and Christian F. Roos. Quantum chemistry calculations on a trapped-ion quantum simulator. *Phys. Rev. X*, 8:031022, Jul 2018.

[179] Jarrod R McClean, Nicholas C Rubin, Kevin J Sung, Ian D Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E Schuyler Fried, Craig Gidney, Brendan Gimby, et al. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology*, 5(3):034014, 2020.

[180] Pierre-Luc Dallaire-Demers, Jonathan Romero, Libor Veis, Sukin Sim, and Alán Aspuru-Guzik. Low-depth circuit ansatz for preparing correlated fermionic states on a quantum computer. *arXiv preprint arXiv:1801.01053*, 2018.

[181] Barnaby van Straaten and Bálint Koczor. Measurement cost of metric-aware variational quantum algorithms. *PRX Quantum*, 2:030324, Aug 2021.

[182] H.Q. Lin, J.E. Gubernatis, Harvey Gould, and Jan Tobochnik. Exact diagonalization methods for quantum systems. *Computers in Physics*, 7(4):400–407, 1993.

[183] Marco De Vivo, Matteo Masetti, Giovanni Bottegoni, and Andrea Cavalli. Role of molecular dynamics and related methods in drug discovery. *Journal of medicinal chemistry*, 59(9):4035–4061, 2016.

[184] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 2017.

[185] Lea Thøgersen and Jeppe Olsen. A coupled cluster and full configuration interaction study of CN and CN-. *Chemical Physics Letters*, 393(1-3):36–43, 2004.

[186] Jarrod R. McClean, Mollie E. Kimchi-Schwartz, Jonathan Carter, and Wibe A. de Jong. Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states. *Phys. Rev. A*, 95(042308), 2017.

[187] J. I. Colless, V. V. Ramasesh, D. Dahlen, M. S. Blok, M. E. Kimchi-Schwartz, J. R. McClean, J. Carter, W. A. de Jong, and I. Siddiqi. Computation of molecular spectra on a quantum processor with an error-resilient algorithm. *Phys. Rev. X*, 8(011021), 2018.

[188] Raffaele Santagati, Jianwei Wang, Antonio A. Gentile, Stefano Paesani, Nathan Wiebe, Jarrod R. McClean, Sam Morley-Short, Peter J. Shadbolt, Damien Bonneau, Joshua W. Silverstone, David P. Tew, Xiaoqi Zhou, Jeremy L. O'Brien, and Mark G. Thompson. Witnessing eigenstates for quantum simulation of Hamiltonian spectra. *Science Advances*, 4(1), 2018.

[189] Lukasz Cincio, Yiğit Subaşı, Andrew T Sornborger, and Patrick J Coles. Learning the quantum algorithm for state overlap. *New Journal of Physics*, 20(11):113022, 2018.

[190] Juan Carlos, Garcia-Escartin, and Pedro Chamorro-Posada. Swap test and Hong-Ou-Mandel effect are equivalent. *Phys. Rev. A*, 87(052330), 2013.

[191] Oscar Higgott, Daochen Wang, and Stephen Brierley. Variational quantum computation of excited states. *Quantum*, 3:156, 2019.

[192] Dave Wecker, Matthew B Hastings, and Matthias Troyer. Progress towards practical quantum variational algorithms. *Phys. Rev. A*, 92:042303, Oct 2015.

[193] Robin Blume-Kohout. Optimal, reliable estimation of quantum states. *New Journal of Physics*, 12(4):043034, 2010.

[194] Tyson Jones, Suguru Endo, Sam McArdle, Xiao Yuan, and Simon C. Benjamin. Variational quantum algorithms for discovering Hamiltonian spectra. *Phys. Rev. A*, 99:062304, Jun 2019.

[195] Trygve Helgaker, Poul Jorgensen, and Jeppe Olsen. *Molecular electronic-structure theory*. John Wiley & Sons, 2014.

[196] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Phys. Rev. Lett.*, 87:167902, Sep 2001.

[197] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of theoretical physics*, 21(3):219–253, 1982.

[198] Jonathan Romero, Ryan Babbush, Jarrod R McClean, Cornelius Hempel, Peter J Love, and Alán Aspuru-Guzik. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. *Quantum Science and Technology*, 4(1):014008, 2019.

[199] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[200] Simon C Benjamin, Liming Zhao, and Joseph F Fitzsimons. Measurement-driven quantum computing: Performance of a 3-SAT solver. *arXiv preprint arXiv:1711.02687*, 2017.

[201] Dimitris Achlioptas, Assaf Naor, and Yuval Peres. Rigorous location of phase transitions in hard optimization problems. *Nature*, 435(7043):759–764, 2005.

[202] T Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.

[203] Oscar Higgott, Daochen Wang, and Stephen Brierley. Variational quantum computation of excited states, 2018.

[204] Ali Rad, Alireza Seif, and Norbert M Linke. Surviving the barren plateau in variational quantum circuits with Bayesian learning initialization. *arXiv preprint arXiv:2203.02464*, 2022.

[205] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[206] Ying Li. A magic states fidelity can be superior to the operations that created it. *New Journal of Physics*, 17(2):023037, 2015.

[207] Earl T Campbell and Mark Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Physical Review A*, 95(2):022316, 2017.

[208] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. An algorithm for the T-count. *Quantum Info. Comput.*, 14(1516):12611276, Nov 2014.

[209] Neil J Ross and Peter Selinger. Optimal ancilla-free Clifford+ T approximation of z-rotations. *Quantum Inf. Comput.*, 16(11&12):901–953, 2016.

[210] Matthew Amy and Michele Mosca. T-count optimization and Reed–Muller codes. *IEEE Transactions on Information Theory*, 65(8):4771–4784, 2019.

[211] Luke E Heyfron and Earl T Campbell. An efficient quantum compiler that reduces T count. *Quantum Science and Technology*, 4(1):015004, 2018.

[212] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):1–12, 2018.

[213] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[214] T. P. Harty, D. T. C. Allcock, C. J. Ballance, L. Guidoni, H. A. Janacek, N. M. Linke, D. N. Stacey, and D. M. Lucas. High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. *Physical Review Letters*, 113(22):220501, 2014.

[215] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Physical Review Letters*, 117(6):060504, 2016.

[216] John P Gaebler, Ting Rei Tan, Y Lin, Y Wan, Ryan Bowler, Adam C Keith, Scott Glancy, Kevin Coakley, Emanuel Knill, Dietrich Leibfried, et al. High-fidelity

universal gate set for be 9+ ion qubits. *Physical Review Letters*, 117(6):060505, 2016.

[217] Naomi H Nickerson, Joseph F Fitzsimons, and Simon C Benjamin. Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Physical Review X*, 4(4):041041, 2014.

[218] Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T Sornborger, and Patrick J Coles. Quantum-assisted quantum compiling. *Quantum*, 3:140, 2019.

[219] Jacques Carolan, Masoud Mohseni, Jonathan P Olson, Mihika Prabhu, Changchen Chen, Darius Bunandar, Murphy Yuezhen Niu, Nicholas C Harris, Franco NC Wong, Michael Hochberg, et al. Variational quantum unsampling on a quantum photonic processor. *Nature Physics*, 16(3):322–327, 2020.

[220] Adam R. Brown and Leonard Susskind. Second law of quantum complexity. *Phys. Rev. D*, 97:086015, Apr 2018.

[221] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[222] Graham R Wood. The bisection method in higher dimensions. *Mathematical programming*, 55(1):319–337, 1992.

[223] Jian Ma, Xiaoguang Wang, Chang-Pu Sun, and Franco Nori. Quantum spin squeezing. *Physics Reports*, 509(2-3):89–165, 2011.

[224] Duger Ulam-Orgikh and Masahiro Kitagawa. Spin squeezing and decoherence limit in Ramsey spectroscopy. *Physical Review A*, 64(5):052106, 2001.

[225] Jonas M Kübler, Andrew Arrasmith, Lukasz Cincio, and Patrick J Coles. An adaptive optimizer for measurement-frugal variational algorithms. *Quantum*, 4:263, 2020.

[226] Andrew Arrasmith, Lukasz Cincio, Rolando D Somma, and Patrick J Coles. Operator sampling for shot-frugal optimization in variational algorithms. *arXiv preprint arXiv:2004.06252*, 2020.

[227] Bálint Koczor and Simon C Benjamin. Quantum natural gradient generalised to non-unitary circuits. *arXiv preprint arXiv:1912.08660*, 2019.

[228] Kristan Temme, Sergey Bravyi, and Jay M Gambetta. Error mitigation for short-depth quantum circuits. *Physical Review Letters*, 119(18):180509, 2017.

[229] Abhinav Kandala, Kristan Temme, Antonio D Córcoles, Antonio Mezzacapo, Jerry M Chow, and Jay M Gambetta. Error mitigation extends the computational reach of a noisy quantum processor. *Nature*, 567(7749):491–495, 2019.

[230] Suguru Endo, Simon C Benjamin, and Ying Li. Practical quantum error mitigation for near-future applications. *Physical Review X*, 8(3):031027, 2018.

[231] Pawel Wocjan, Dominik Janzing, and Thomas Beth. Two QCMA-complete problems. *arXiv preprint quant-ph/0305090*, 2003.

[232] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications*, 10(1):1–9, 2019.

[233] Arthur G Rattew, Shaohan Hu, Marco Pistoia, Richard Chen, and Steve Wood. A domain-agnostic, noise-resistant, hardware-efficient evolutionary variational

quantum eigensolver. *arXiv preprint arXiv:1910.09694*, 2019.

[234] M Bilkis, M Cerezo, Guillaume Verdon, Patrick J Coles, and Lukasz Cincio. A semi-agnostic ansatz with variable structure for quantum machine learning. *arXiv preprint arXiv:2103.06712*, 2021.

[235] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1–12, 2021.

[236] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[237] Fernando GSL Brandao and Krysta M Svore. Quantum speed-ups for solving semidefinite programs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–426. IEEE, 2017.

[238] Joran van Apeldoorn and András Gilyén. Improvements in quantum SDP-solving with applications. *arXiv preprint arXiv:1804.05058*, 2018.

[239] Andrew Cross. The IBM Q experience and QISKit open-source quantum computing software. *APS*, 2018:L58–003, 2018.

[240] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. Tensorflow quantum: A software framework for quantum machine learning, 2020.

[241] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas PD Sawaya. Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits. *Quantum Science and Technology*, 5(3):034007, 2020.

[242] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert W function. *Advances in Computational mathematics*, 5(1):329–359, 1996.

[243] Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019.

[244] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[245] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. Yao.jl: Extensible, efficient framework for quantum algorithm design, 2019.

[246] David Wierichs, Christian Gogolin, and Michael Kastoryano. Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer. *Physical Review Research*, 2(4):043246, 2020.

[247] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms. *arXiv preprint arXiv:2009.02823*, 2020.

[248] Man-Duen Choi. Completely positive linear maps on complex matrices. *Linear algebra and its applications*, 10(3):285–290, 1975.

[249] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin. QuEST and high performance simulation of quantum computers. *Scientific Reports*, 9(1):1–11, 2019.

[250] Y. Y. Shi, L. M. Duan, and Guifre Vidal. Classical simulation of quantum many-body systems with a tree tensor network. *Physical Review A*, 74(2):022320, 2006.

[251] Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550, 2019.

[252] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):147902, 2003.

[253] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, September 2019.

[254] Yifei Huang and Peter Love. Feynman-path-type simulation using stabilizer projector decomposition of unitaries. *Phys. Rev. A*, 103:022428, Feb 2021.

[255] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[256] George F Viamontes, Igor L Markov, and John P Hayes. *Quantum circuit simulation*. Springer Science & Business Media, 2009.

[257] Vasilis Samoladas. Improved bdd algorithms for the simulation of quantum circuits. In *European Symposium on Algorithms*, pages 720–731. Springer, 2008.

[258] Stefan Hillmich, Igor L Markov, and Robert Wille. Just like the real thing: Fast weak simulation of quantum computation. In *2020 57th ACM/IEEE Design*

Automation Conference (DAC), pages 1–6. IEEE, 2020.

[259] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. Tools for quantum computing based on decision diagrams. *ACM Transactions on Quantum Computing*, 3(3):1–17, 2022.

[260] Cupjin Huang, Michael Newman, and Mario Szegedy. Explicit lower bounds on strong quantum simulation. *IEEE Transactions on Information Theory*, 66(9):5585–5600, 2020.

[261] M Nest. Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond. *arXiv preprint arXiv:0811.0898*, 2008.

[262] Hans De Raedt, Fengping Jin, Dennis Willsch, Madita Willsch, Naoki Yoshioka, Nobuyasu Ito, Shengjun Yuan, and Kristel Michielsen. Massively parallel quantum computer simulator, eleven years later. *Computer Physics Communications*, 237:47–61, 2019.

[263] Damian S. Steiger, Thomas Häner, and Matthias Troyer. ProjectQ: an open source software framework for quantum computing. *Quantum*, 2:49, January 2018.

[264] William Y Chen, Pohua P. Chang, Thomas M Conte, and Wen-mei W. Hwu. The effect of code expanding optimizations on instruction cache design. *IEEE Transactions on Computers*, 42(9):1045–1057, 1993.

[265] Samuel Larsen and Saman Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets. *Acm Sigplan Notices*, 35(5):145–156, 2000.

[266] Stanley F Anderson, John G Earle, Robert Elliott Goldschmidt, and Don M

Powers. The IBM system/360 model 91: Floating-point execution unit. *IBM Journal of research and development*, 11(1):34–53, 1967.

[267] Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, et al. Haswell: The fourth-generation Intel core processor. *IEEE micro*, 34(2):6–20, 2014.

[268] Part Guide. Intel® 64 and ia-32 architectures software developers manual. *Volume 3B: System programming Guide, Part*, 2(11), 2011.

[269] Jing Ge Feng, Ye Ping He, and Qiu Ming Tao. Evaluation of compilers capability of automatic vectorization based on source code analysis. *Scientific Programming*, 2021, 2021.

[270] Jaewook Shin, Mary Hall, and Jacqueline Chame. Superword-level parallelism in the presence of control flow. In *International Symposium on Code Generation and Optimization*, pages 165–175. IEEE, 2005.

[271] James Bottomley. Understanding caching. *Linux Journal*, 2004(117):2, 2004.

[272] James E Smith. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, pages 202–215, 1998.

[273] Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11, 2007.

[274] Bil Lewis and Daniel J Berg. *Multithreaded programming with P-threads*. Prentice-Hall, Inc., 1998.

[275] Nakul Manchanda and Karan Anand. Non-uniform memory access (numa). *New York University*, 4, 2010.

[276] William J Bolosky and Michael L Scott. False sharing and its effect on shared memory performance. In *4th Symposium on Experimental Distributed and Multiprocessor Systems*, pages 57–71. Citeseer, 1993.

[277] Mohammad Alaul Haque Monil, Seyong Lee, Jeffrey S. Vetter, and Allen D. Malony. Understanding the impact of memory access patterns in Intel processors. In *2020 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pages 52–61, 2020.

[278] David A Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954.

[279] NVIDIA. CUDA C programming guide: design guide. 2022.

[280] Duane Storti and Mete Yurtoglu. *CUDA for engineers: an introduction to high-performance parallel computing*. Addison-Wesley Professional, 2015.

[281] NVIDIA Corporation. Datasheet Quadro P6000, Sep 2016.

[282] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. {PipeSwitch}: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 499–514, 2020.

[283] Xiuxia Zhang, Guangming Tan, Shuangbai Xue, Jiajia Li, Keren Zhou, and Mingyu Chen. Understanding the GPU microarchitecture to achieve bare-metal performance tuning. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 31–43, 2017.

[284] Andrew Davidson and John Owens. Toward techniques for auto-tuning GPU algorithms. In *International Workshop on Applied Parallel Computing*, pages 110–119. Springer, 2010.

[285] Bertrand Putigny, Brice Goglin, and Denis Barthou. A benchmark-based performance model for memory-bound hpc applications. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pages 943–950. IEEE, 2014.

[286] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE international symposium on performance analysis of systems and software*, pages 163–174. IEEE, 2009.

[287] Mark Harris et al. Optimizing parallel reduction in CUDA. *NVIDIA developer technology*, 2(4):70, 2007.

[288] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, 2018.

[289] Gregory R Andrews. Foundations of multithreaded, parallel, and distributed programming. *Wesley, University of Arizona, USA*, 2000.

[290] Carl Friedrich Gauss. *Disquisitiones Arithmeticae auctore D. Carolo Friderico Gauss.* in commissis apud Gerh. Fleischer, jun., 1801.

[291] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alãn Aspuru-Guzik. qHiPSTER: The quantum high performance software testing environment, 2016.

[292] Cleve Moler. Matrix computation on distributed memory multiprocessors. *Hypercube Multiprocessors*, 86(181-195):31, 1986.

[293] Leonard Euler and I Often. Introduction to analysis of the infinite: Book ii, 1989.

[294] Ewing Lusk, S Huss, B Saphir, and M Snir. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):623, 2009.

[295] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[296] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel de Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al. OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 2021.

[297] Geoffrey James. Law of least astonishment. *The Tao of Programming*, 4, 1987.

[298] Hareton K. N. Leung and Lee White. A study of integration testing and software regression at the integration level. In *Proceedings. Conference on Software Maintenance 1990*, pages 290–301. IEEE, 1990.

[299] Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[300] Ulrich Drepper. How to write shared libraries. *Red Hat Linux*, 2006.

[301] Raymond Chen. What does the /alternatename linker switch do? *The Old New Thing, Microsoft Dev Blogs*, July 2020.

[302] Binh Nguyen. *Linux Dictionary*. Binh Nguyen, 2003.

[303] WSTP (Wolfram Symbolic Transfer Protocol), *Wolfram*. https://www.wolfram.com/wstp/. Accessed: 2019-12-05.

[304] Todd Gayley. A MathLink tutorial. *Wolfram Research*, 2002.

[305] Eric Pidoux. *Git Best Practices Guide*. Packt Publishing Ltd, 2014.

[306] Dorota Huizinga and Adam Kolawa. *Automated defect prevention: best practices in software management*. John Wiley & Sons, 2007.

[307] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[308] Larry J Morell and Lionel E Deimel. Unit analysis and testing. Technical report, Carnegie-Mellon University Pittsburgh, PA Software Engineering Inst, 1992.

[309] Giuseppe Sergioli. Quantum circuit optimization for unitary operators over non-adjacent qudits. *arXiv preprint arXiv:1711.09765*, 2017.

[310] Björne Stroustrup. Operator overloading in C++. In *Conference on System Implementation Languages: Experience & Assessment (84 Proceedings)*, 1984.

[311] Lyle Pursell and S. Y. Trimble. Gram-Schmidt orthogonalization by Gauss elimination. *The American Mathematical Monthly*, 98(6):544–549, 1991.

[312] Martin Hoeovsk, Phil Nash, Clare Macrae, Jozef Grajciar, Martin Moene, , Kosta, PureAbstract, Offa, , Baruch, Axel Huebl, Coombez, Dvirtz, NeroBurner, David Seifert, Pfiffikus, Daniele E. Domenichelli, Uilian Ries, Mickey Rose, Maciej Patro, Alan Jowett, Alexandr Timofeev, Billy O'Neal, Martin Jebek, Steven

Franzen, VZ, Ashley Hauck, Omer Ozarslan, John Bytheway, Yuriy Kochetkov, and Sean D. Cline. catchorg/catch2: v2.13.9, 2022.

[313] Mathias Meyer. Continuous integration and its tools. *IEEE software*, 31(3):14–16, 2014.

[314] Github Actions. https://github.com/features/actions. Accessed: 2020-03-15.

[315] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. YAML aint markup language, version 1.1. *Working Draft 2008-05*, 11, 2009.

[316] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[317] Paul Larson, Nigel Hinds, Rajan Ravindran, and Hubertus Franke. Improving the linux test project with kernel code coverage analysis. In *Proceedings of the Linux Symposium*, pages 1–12. Citeseer, 2003.

[318] Benjamin D Lee. Ten simple rules for documenting scientific software. *PLoS Computational Biology*, 14(12):e1006561, 2018.

[319] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[320] Ryan LaRose. Distributed memory techniques for classical simulation of quantum circuits. *arXiv preprint arXiv:1801.01037*, 2018.

[321] ARCHER hardware. http://www.archer.ac.uk/about-archer/hardware/. Accessed: Dec 2017.

[322] Sam Stanwyck, Harun Bayraktar, and Tim Costa. cuQuantum: Accelerating quantum circuit simulation on GPUs. *Bulletin of the American Physical Society*, 2022.

[323] Sumit Gupta. What is NVLink? and how will it make the worlds fastest computer possible?". *NVIDIA Blog*, 2014.

[324] Satish Kumar Sadasivam, Brian W Thompto, Ron Kalla, and William J Starke. IBM Power9 processor architecture. *IEEE Micro*, 37(2):40–51, 2017.

[325] Fabrizio; Ammendola Roberto; Biagioni Andrea; Ramusino Angelo Cotta; Fiorini Massimiliano; Frezza Ottorino; Lamanna Gianluca; Cicero Francesca Lo; Martinelli Michele; Neri Ilaria; Paolucci Pier Stanislao; Pastorelli Elena; Pontisso Luca; Rossetti Davide; Simeone Francesco; Simula Francesco; Sozzi Marco; Tosoratto Laura; Vicini Piero; Lonardo, Alessandro; Ameli. A FPGA-based network interface card with GPUDirect enabling realtime GPU computing in HEP experiments.., 2015.

[326] Mark Silberstein, Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, and Emmett Witchel. GPUnet: Networking abstractions for GPU programs. *ACM Transactions on Computer Systems (TOCS)*, 34(3):1–31, 2016.

[327] Yi-Ting Chen, Collin Farquhar, and Robert M Parrish. Low-rank density-matrix evolution for noisy quantum circuits. *npj Quantum Information*, 7(1):1–12, 2021.