Computer Physics Communications 221 (2017) 160–173

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Exploiting the chaotic behaviour of atmospheric models with reconfigurable architectures

Francis P. Russell [a,*], Peter D. Düben [b], Xinyu Niu [a], Wayne Luk [a], T.N. Palmer [b]

[a] Department of Computing, Imperial College London, United Kingdom
[b] Atmospheric Oceanic and Planetary Physics, University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 14 March 2017
Received in revised form 10 August 2017
Accepted 11 August 2017
Available online 19 August 2017

Keywords:
FPGA
Chaotic system
Precision reduction
Weather modelling

ABSTRACT

Reconfigurable architectures are becoming mainstream: Amazon, Microsoft and IBM are supporting such architectures in their data centres. The computationally intensive nature of atmospheric modelling is an attractive target for hardware acceleration using reconfigurable computing. Performance of hardware designs can be improved through the use of reduced-precision arithmetic, but maintaining appropriate accuracy is essential. We explore reduced-precision optimisation for simulating chaotic systems, targeting atmospheric modelling, in which even minor changes in arithmetic behaviour will cause simulations to diverge quickly. The possibility of equally valid simulations having differing outcomes means that standard techniques for comparing numerical accuracy are inappropriate. We use the Hellinger distance to compare statistical behaviour between reduced-precision CPU implementations to guide reconfigurable designs of a chaotic system, then analyse accuracy, performance and power efficiency of the resulting implementations. Our results show that with only a limited loss in accuracy corresponding to less than 10% uncertainty in input parameters, the throughput and energy efficiency of a single-precision chaotic system implemented on a Xilinx Virtex-6 SX475T Field Programmable Gate Array (FPGA) can be more than doubled.

1. Introduction

Climate and weather prediction are computationally intensive; even with high-performance computing resources, it is typically impossible to resolve important convective cloud systems in global models [1]. Numerical models of weather and climate show significant model error due to limited resolution and complexity, necessitating a need for even more resource-intensive models. Performance and power requirements for running such models with hard time constraints, for example in operational weather forecasts, have led to the exploration of hardware accelerators such as GPUs and FPGAs to obtain greater throughput and power efficiency [2,3].

With reconfigurable architectures becoming more prominent and increasingly accessible for running accelerated computation, we choose to investigate how designs may be customised to take advantage of the characteristics of climate simulations. A common approach to enhance throughput of hardware designs is to reduce precision of calculations so that additional hardware resources can be employed to increase parallelism. Excessive precision reduction however, reduces calculation quality and usefulness, so a trade-off must be made between performance and accuracy.

Lorenz showed that weather forecasting involves the prediction of a chaotic system [4]. This implies an exponential growth of errors in any perturbation of the model, such as a slight change in initial conditions; significant divergence between a CPU and hardware implementation is expected simply due to implementation differences, and is not in itself an indicator of error. Diagnostics that rely on solution convergence between implementations for validation are no longer appropriate.

Weather and climate models are known to show significant limitations due to limited resolution or insufficient complexity in the representation of the Earth system. Short-term forecasts will be perturbed by uncertainty in initial conditions due to the nature of measurements and data assimilation. If the model setup is changed, e.g. by a change of software/hardware implementation, model parameters or model forcing (such as CO_2 concentration), it is almost impossible to predict the response of the model due to the chaotic nature of the system and numerical simulation is necessary to identify the impact.

A reduction in precision will reduce computational cost and resource usage but will also alter arithmetic behaviour and influence model simulations. However, given the model limitations that are outlined above, one may consider a reduction in precision

* Corresponding author.
E-mail address: francis.russell02@imperial.ac.uk (F.P. Russell).

http://dx.doi.org/10.1016/j.cpc.2017.08.011

appropriate so long as the behaviour change introduced is insignificant compared to those introduced by other factors. The resources saved through precision reduction may then facilitate an increase of design throughput or permit more complex designs.

To this end, we investigate the reduction of precision in chaotic systems using the Lorenz 1996 model (a.k.a. Lorenz 1995 and referred to hereafter as Lorenz '96), designed by Lorenz [5] to study interactions of atmospheric processes with non-linear, chaotic dynamics. Our analysis has applicability beyond this model — scale interactions within the Lorenz '96 model resemble scale interactive behaviour of various parts of numerical atmosphere models that is typically difficult to capture in idealised systems. Similar scale interactions are also important for turbulent energy cascades relevant to most applications in CFD (computational fluid dynamics).

The Lorenz '96 model has been used in extensive studies in the literature to investigate new methods for data assimilation, the propagation and representation of model error and improved numerical algorithms involving dynamical systems [6–10].

This paper contributes and presents:

- the hardware architecture of a two-scale Lorenz '96 simulation using Runge–Kutta time-stepping;
- a demonstration of how it is possible to make trade-offs between precision and throughput for a system where numerical divergence is expected;
- an analysis of the impact of varying the precision of variables at different scales in the Lorenz '96 implementation using error metrics appropriate to chaotic systems (the Hellinger distance);
- performance, precision and power consumption comparisons of reduced and single-precision FPGA implementations.

This paper is an extended version of work presented at the 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines in 2015 (FCCM'15) [11]. Additional contributions and content include:

- a more detailed presentation of the chaotic properties of Lorenz '96 and the effect of arithmetic precision and parameter changes on the evolution of the system (Section 4.1);
- a detailed overview of the methodology we used to determine the value representations we chose to use for hardware builds (Section 4.3);
- the application of the same methodology with fixed-point representations, and the presentation of power, precision and performance results for the resulting builds (Section 4.7 & 6);
- a more detailed performance analysis of hardware builds, including calculations of arithmetic throughput and bandwidth utilisation (Section 6.2).

2. Field Programmable Gate Arrays as specialised accelerators

Field Programmable Gate Arrays (FPGAs) are integrated circuits whose architecture can be reconfigured. FPGAs are achieving increasing visibility as an accelerator architecture, present in Microsoft, Amazon and IBM data centres — a major benefit being a high compute-to-power ratio. Microsoft's Project Catapult has deployed FPGA accelerators into Microsoft data-centre servers where a performance improvement of 95% in search engine scoring was obtained relative to a software implementation running on 12-core Intel Sandy Bridge CPUs but with a maximum power overhead of only 22.7 W [12]. A revised accelerator architecture is now being deployed at hyperscale in Microsoft's production datacentres worldwide [13].

Alternative accelerator architectures include GPUs (Graphics Processing Units) and the Intel Xeon Phi. Performance and power comparisons typically show highly application-dependent results with arithmetic intensity and regularity of data access being important factors [14,15]. Given the significant power requirement for executing climate models, our work primarily focuses on power efficiency rather than the computational throughput of a given device. High-performance GPUs typically have large power requirements [16], but can be power-efficient when high computational throughput is achieved e.g. for operations such as matrix-multiply [15]. However, finite difference computations like those used in many weather models are typically bandwidth bound and may only achieve a fraction of peak performance. In situations where near peak performance of a GPU cannot be attained, FPGAs can provide a significant advantage over GPUs in terms of energy efficiency [16].

FPGAs typically contain reconfigurable logic blocks, random access memory elements (Block RAM) totalling tens of MBs and DSP (digital signal processing) elements which provide efficient implementation of various numerical primitives. Through configuration of the programmable logic, arbitrary designs can be placed on the chip provided that resource, routing and timing constraints can be satisfied.

Conventional CPU cores contain pipelines designed for the execution of general purpose calculations where operations in a pipeline are dependent on an instruction stream that is unknown until execution of a program. Although it is possible to materialise similar architectures on FPGAs, most efficient utilisation occurs when the architecture chosen is customised to the computation being performed.

In an architecture designed for a specific problem, the pipeline will only involve steps required for the calculation. Since the pipeline is intended solely to execute a specific calculation, there is typically no need for a fetch–decode–execute cycle or techniques such as branch prediction or out-of-order execution which are frequently employed in general purpose architectures. Obtaining effective performance from an FPGA thus depends on the creation of an architecture that can perform a given computation efficiently rather than mapping it to a pipeline chosen in advance.

Optimised FPGA architectures typically make extensive use of pipeline parallelism, and for compute intensive workloads, may perform thousands of operations per cycle. This extensive parallelism enables FPGAs to achieve significant throughput despite having lower clock frequencies (typically hundreds of MHz) than conventional CPU architectures. Key to achieving efficiency is also the choice of number representation — real numbers need not be represented as IEEE floating-point values and using reduced-precision floating or fixed-point representations can significantly reduce resources, making it possible to place higher throughput designs on a device. We exploit the ability for FPGAs to use custom number representations in this work, by choosing representations that possess only the precision necessary to perform calculations of interest. In contrast, a conventional CPU is only capable of performing calculations efficiently with representations chosen during its design.

FPGAs have previously been applied to Limited Area Models such as BOLAM [17] and the global shallow water equations [18] using finite difference schemes though such work has considered short-term simulations that show only limited propagation of model errors due to the chaotic dynamics of the atmosphere. In this work, we consider the influence of reduced precision on a long-term diagnostic for a system with strong chaotic behaviour. Furthermore, the Lorenz '96 model allows us to choose the level of precision reduction to apply at different scales, which we also explore. Previous explorations have been limited to software-based simulations which could only be run at much smaller scale [19,20].

The potential for FPGAs to provide highly power efficient acceleration has led us to explore techniques whereby a design may be customised to take advantage of the numerical properties of chaotic systems that occur in climate modelling. Climate modelling is both power and computation intensive, and effective use of FPGA architectures could provide a significant benefit in terms of power and cost.

3. The Lorenz '96 equations

The two-scale Lorenz '96 model was designed by Lorenz as a simple model to study predictability in a chaotic system. Despite its simplicity, Lorenz '96 possesses important properties of numerical atmosphere models such as scale interactions and chaotic behaviour. This makes Lorenz '96 useful for initial investigation of techniques before application to actual climate models and other chaotic systems [6–10]. Also, its numerical implementation has significant similarities to widely used CFD finite difference models in terms of the discrete grid point representation and time stepping schemes.

The model is defined by a set of coupled equations involving K global-scale variables ($X_{1\leq k\leq K}$), each of which is associated with a set of J local-scale variables ($Y_{1\leq j\leq J,k}$):

$$\frac{dX_k}{dt} = -X_{k-1}(X_{k-2} - X_{k+1}) - X_k + F - \frac{hc}{b}\sum_{j=1}^{J} Y_{j,k} \tag{1}$$

$$\frac{dY_{j,k}}{dt} = -cbY_{j+1,k}(Y_{j+2,k} - Y_{j-1,k}) - cY_{j,k} + \frac{hc}{b}X_k \tag{2}$$

where F is a forcing term, h is the coupling constant, b is the spatial-scale ratio and c is the time-scale ratio. Often $h = 1$, $F = 20$, $b = 10$ and $c = 4$ or $c = 10$. A time-unit in Lorenz '96, hereafter referred to as a Model Time Unit (MTU), is considered equivalent to 5 atmospheric days due to a comparison by Lorenz which matched the error doubling rate to Lorenz '96 to global circulation models [5].

Global and local-scale variables are arranged circularly (Fig. 1) such that $X_k = X_{k+K}$, $Y_{j,k} = Y_{j,k+K}$ and $Y_{j,k} = Y_{j-J,k+1}$. For clarity we show a small system; in practice we look at how to scale J and K to hundreds of thousands. We explore how to scale both J and K, but only run simulations with relatively small values of J (144) since local-scale values only influence global-scale values through a summation so larger values of J result in statistically similar behaviour.

For simplicity we use explicit time stepping. All simulations are temporally discretised using classical fourth-order Runge–Kutta — a higher-order scheme that is frequently used in standard models. It achieves high polynomial accuracy through the combination of multiple intermediate estimates of the future model state.

The choice of time-stepping scheme can significantly impact accelerator design. Runge–Kutta has fourth-order accuracy but requires significantly more operations compared to less accurate schemes. In Runge–Kutta the intermediate estimates are only used for advancing a single timestep, and are not useful otherwise. In a hardware design, calculation and consumption of these intermediate values can be pipelined, so they never need to be explicitly stored, which could only be done off-chip. Hence, adopting Runge–Kutta in a hardware implementation makes it possible to increase the arithmetic intensity of a calculation without increasing bandwidth requirements. By contrast, Adams–Bashforth (another explicit higher-order scheme) reuses system states from previous timesteps in multiple future calculations which would likely force them to be saved off-chip, increasing bandwidth requirements.

4. Precision analysis

In this section we discuss the chaotic properties of Lorenz '96 (Section 4.1), how we evaluate the correctness of a given Lorenz '96 implementation (Section 4.2), our methodology for choosing the precision of hardware designs (Section 4.3), the implementation of the analysis (Section 4.4), and the analysis results for the hardware designs built in this paper (Sections 4.5, 4.6 & 4.7).

4.1. Chaotic behaviour in the Lorenz '96 system

Lorenz '96 exhibits complex wave-like and chaotic behaviour [21]. In chaotic systems, the trajectories of two simulations that only differ by a small perturbation in the initial conditions will diverge, with the mean distance between their trajectories increasing exponentially in time. Eventually, the two trajectories become uncorrelated and the mean distance will no longer increase and converges towards a certain value that is a property of the system.

Despite this, observing the divergence rate for short-term simulations allows us to explore the sensitivity of a simulation to changes in initial conditions, and the relative magnitude of the effects caused by different sources of perturbation.

In this section, we will compare the divergence rates for simulations with perturbations from two different sources:

1. The use of slightly perturbed model parameters.
2. The use of reduced numerical precision in one of the model setups.

We study (1) since weather and climate models (next to many other shortcomings such as limited resolution and complexity and unknown initial conditions) are based on numerous model parameters that are not known exactly such that simulations of future weather will inevitably diverge.

To characterise the impact of this effect, we run our C++ Lorenz '96 implementation (Section 4.4) with different precisions of floating-point variables corresponding to half, single, double, x86-extended, quadruple and octuple precision. We run each system from the same initial state derived by running a simulation in double precision for 2000 iterations (1 MTU).

The average forecast error $E(t)$ at time t for N_f simulations is the average mean difference between the global-scale variables of each model and a reference x^{ref}:

$$E(t) = \frac{1}{KN_f}\sum_{j=1}^{N_f}\sum_{k=1}^{K}\left|X_{k,j}^{\text{ref}}(t) - X_{k,j}^{\text{model}}(t)\right|. \tag{3}$$

For our experiment we choose $N_f = 1$ and choose $K = 1000$ to increase the number of variables we average over. We cannot execute an infinite-precision reference, so we simulate in floating point with 489-bit mantissas used for local and global-scale quantities; we conjecture that a 489-bit mantissa would correspond to the precision obtained by using sexdecuple precision (64-byte) values, were they to be defined by IEEE 754. In practice, little hardware support exists for number representations beyond double precision. Results are shown in Fig. 2.

The precision differences result in exponential increases in error from the reference simulation. Errors reach a peak when the two simulations being compared no longer have any correlation to each other. Although increasing precision does reduce divergence, this demonstrates that numerical divergence is not a useful metric for evaluating the precision of long-term simulations of chaotic systems.

We plot the forecast error when the Lorenz '96 parameters c and F are scaled by various values in Fig. 3. These values are chosen to be close to one to correspond to systems where parameters can only be estimated to within a known degree of certainty.

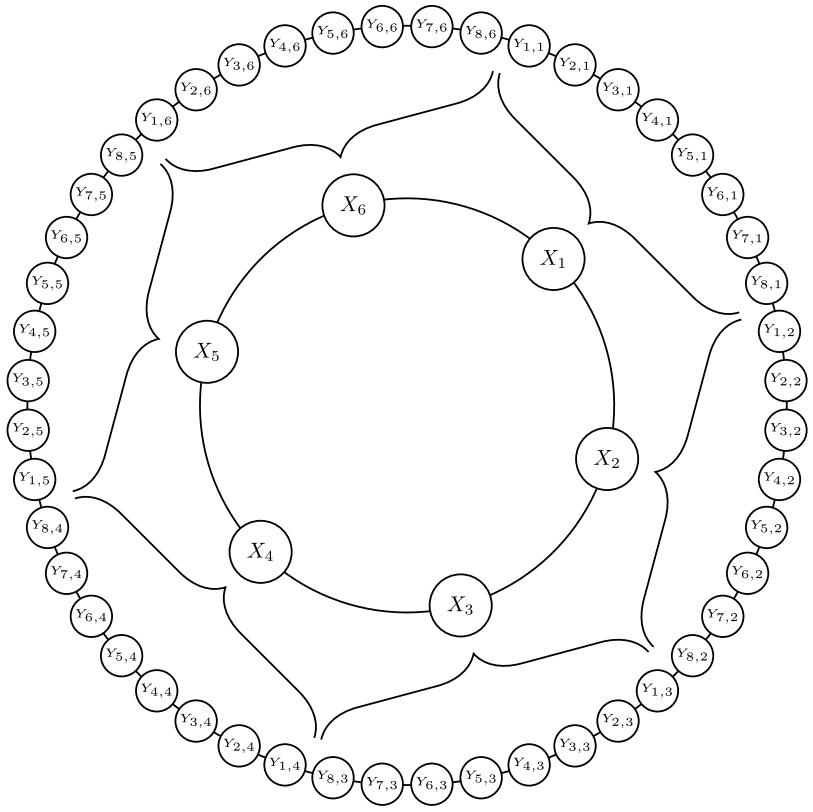


Fig. 1. Visualisation of a two-scale Lorenz '96 system with $J = 8$ and $K = 6$. Global-scale values (X_k) are updated based on neighbouring values and a reduction applied to the local-scale values ($Y_{j,k}$) associated with that value. Local-scale values are updated based on neighbouring values and the associated global-scale value. The neighbourhood topology of both the local and global-scale values is circular.

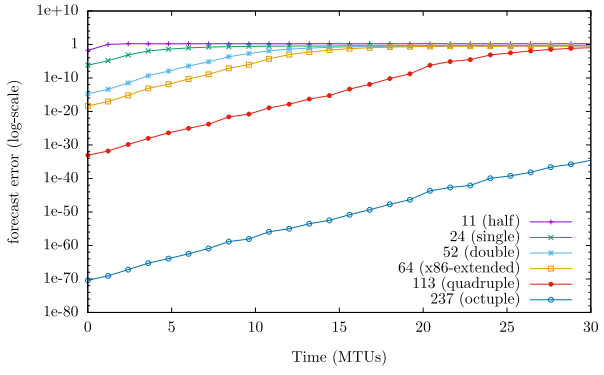


Fig. 2. Forecast error for Lorenz '96 run with different mantissa sizes for both local and global quantities. $J = 120$, $K = 1000$, $F = 40$, $b = 10$ and $c = 10$. All systems were initialised from the same starting state derived by spinning up the system for 2000 iterations (1 MTU) in double precision. Divergence between simulations increases exponentially for all precisions, so we cannot determine what is an acceptable level of numerical precision based on this metric.

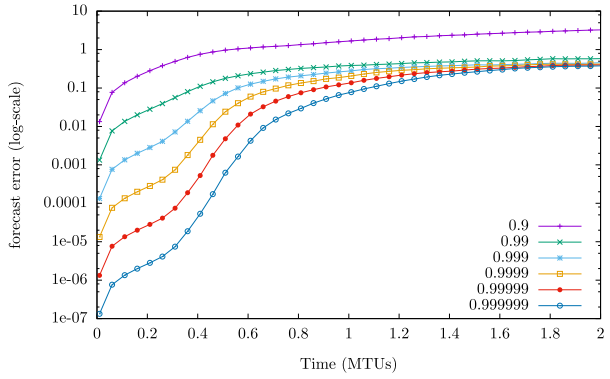


Fig. 3. Forecast error for Lorenz '96 run with parameters c and F multiplied by various coefficients. All systems were initialised from the same starting state derived by spinning up the system for 2000 iterations (1 MTU) in double precision. In comparison to Fig. 2, the rate of divergence caused by changes in system parameters is significantly higher. Note that both the x and y scales are different from Fig. 2 due to the higher rate of increase and magnitude of the divergence. Simulations were performed using 489-bit mantissas for both the local and global quantities (once spin-up was complete).

Again we use 489-bit mantissas for representing local and global-scale quantities. Parameter uncertainties are known to be large for atmospheric applications and 10% uncertainties are not unreasonable [22].

We find that the rate at which forecast error increases due to parameter variation is much greater than that due to changes in arithmetic precision for most of the representations used in Fig. 2. For example, after 1 MTU the forecast error of the Lorenz '96 simulation with c and F scaled by $1-10^{-6}$ is $7.48e-2$, whereas it is $8.84e-6$ for the single-precision implementation with unaltered parameters. This suggests that we can significantly reduce the precision at which a Lorenz '96 simulation is run below that of double or single-precision arithmetic with a change in behaviour comparable to that attributable to parameter uncertainty.

We will consider it acceptable for Lorenz '96 simulations to diverge numerically to the extent attributable to parameter uncertainty. Since Lorenz '96 exhibits non-linear behaviour, the average forecast error is only useful for evaluating short term simulation quality. To evaluate longer term behaviour, we use the Hellinger distance, described in the next section.

4.2. Comparing the long-term dynamics of Lorenz '96 simulations

We use a metric based on the long-term statistical behaviour of the system to compare the similarity of Lorenz '96 simulations in the remainder of this paper. This metric is based on the climatology of Lorenz '96 which is typically defined as the probability density function (PDF) of the X variables, averaged over a long run: 10,000 model time units, equivalent to more than 100 atmospheric years [5]. We note that at least 30 years of weather statistics are typically considered necessary to assess climate and that one Lorenz '96 MTU is considered to correspond to 5 atmospheric days (Section 3). The short term behaviour and correlation of model fields in time will not be considered in such an evaluation. However, we found that results for metrics of short term simulations, such as the forecast error from the previous section, were consistent with metrics that are based on the climatology. For assessing simulation accuracy, we compare the PDFs of *both* local and global-scale variables since we wish to optimise representations for both of these quantities.

We note that the PDF may at first appear to be an overly simple metric for determining whether the long-term dynamics of a precision-reduced Lorenz '96 system are correct. Although Lorenz '96 is only an idealised model to study atmospheric dynamics, the simulation still corresponds to a time-discretised approximation of a continuous system described by a set of partial differential equations. In both Lorenz '96 and real atmospheric simulations, each successive state is highly correlated with the previous one, and we know by construction that each state is derived through application of the Runge–Kutta discretised Lorenz '96 equations. Hence the modified system being compared is fundamentally constrained and the mechanisms whereby such a system could manifest the correct PDF yet be functioning incorrectly are highly limited.

The PDFs of the global-scale variables of a Lorenz '96 system can only be constructed through sampling large numbers of uncorrelated system states, and so is a characterisation of long-term dynamics. We note that the PDF of Lorenz '96 is therefore an emergent property of the system. Most importantly, the PDF of Lorenz '96 is non-trivial to construct — it does not correspond to a Gaussian or simple geometric function [6]. This, combined with the non-linear and chaotic behaviour of Lorenz '96, would magnify any changes in short-term dynamics and makes it extremely improbable that any alteration in either short or long term dynamics would not manifest in changes to the PDF of global-scale values.

The PDFs of global-scale values in Lorenz '96 are considered to be a useful metric for determining correct dynamics in the climate modelling community. To this end, they are often used to analyse veracity of new approaches to *parametrisation* techniques that are studied in Lorenz '96. Parametrisation techniques approximate the effects of unresolved variables in terms of larger-scale, resolved variables e.g. replacing the summation of Y in Eq. (2) by some function of the X variables to avoid having to store or compute on Y (see for example [6]). In contrast, we do not go as far as altering the Lorenz '96 equations themselves, only the precisions of the values used.

A metric often used to compare PDFs is the *Hellinger distance* (see for example [23]). We adopt the Hellinger distance metric for comparing simulations of Lorenz '96. For two probability distributions p and q, the Hellinger distance H is defined as:

$$H(p, q) = \sqrt{\frac{1}{2} \int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 dx}. \tag{4}$$

Calculating the Hellinger distance between two simulations allows us to quantify the statistical similarity between them. Since the Hellinger distance is a comparison of long-term dynamics, we expect two simulations with differing initial conditions but the same parameters to have a Hellinger distance close to zero.

In practice, an empirical measurement of the Hellinger distance between two simulations with identical parameters but differing initial conditions will be non-zero. We use this value as a baseline for estimating when two simulations exhibit the same statistical behaviour.

By varying the parameters of Lorenz '96, we can also calculate Hellinger distances that correspond to the change in behaviour due to changes in parameters to the system. Measurement of real-world parameters for weather modelling are limited by the accuracy and availability of observations from weather instrumentation and therefore subject to uncertainty. By varying Lorenz '96 parameters and measuring the increase in Hellinger distance, we can quantify the change in statistical behaviour that might be considered as being insignificant in comparison due to parameter uncertainties that generate significant model error.

In the following, we will compare Hellinger distances for simulations with reduced precision against Hellinger distances for simulations with parameter changes and differing initial conditions.

4.3. Precision reduction methodology

Reducing calculation precision enables the reduction of hardware resource utilisation, permitting instantiation of additional functional units for increasing throughput. For the Lorenz '96 simulations, we wish to determine the extent to which we can increase throughput while maintaining an acceptable level of accuracy.

We consider both reduced-precision floating point and fixed point. We do not attempt to optimise each intermediate value in our pipeline; rather, we investigate the effects of precision reduction on terms associated with the global and local-scale quantities of Lorenz '96. Düben et al. have shown for an atmosphere model that the precision to calculate small-scale dynamics can be reduced much further than the precision for large-scale dynamics with a smaller influence on model results and forecast quality [20].

We show our methodology in Fig. 4. Each step corresponds to this paper as follows:

1. We modify a CPU implementation of Lorenz '96 to support alternative representations for real values. We describe these changes in Section 4.4.
2. We use the modified Lorenz '96 implementation to profile exponent ranges for local and global-scale quantities. This is described in Section 4.5.
3. We use the original CPU implementation to determine what Hellinger distances we expect between uncorrelated Lorenz '96 simulations with the same parameters, and between Lorenz '96 simulations with altered parameters. This is described in Section 4.6.
4. We run CPU-based simulations of Lorenz '96 with local and global-scale values represented using different value representations. For fixed point, we use the exponent ranges to choose the binary point offset. From these simulations, we produce a mapping between different value representations and the expected Hellinger distance from a reference simulation. This is described in Sections 4.6 and 4.7.
5. We use the Hellinger distances calculated in step 2 to choose what precisions to use for hardware builds (Sections 4.6 and 4.7).
6. We evaluate the performance and power utilisation of our hardware builds and compare Hellinger distances against those estimated in step 2 (Section 6).

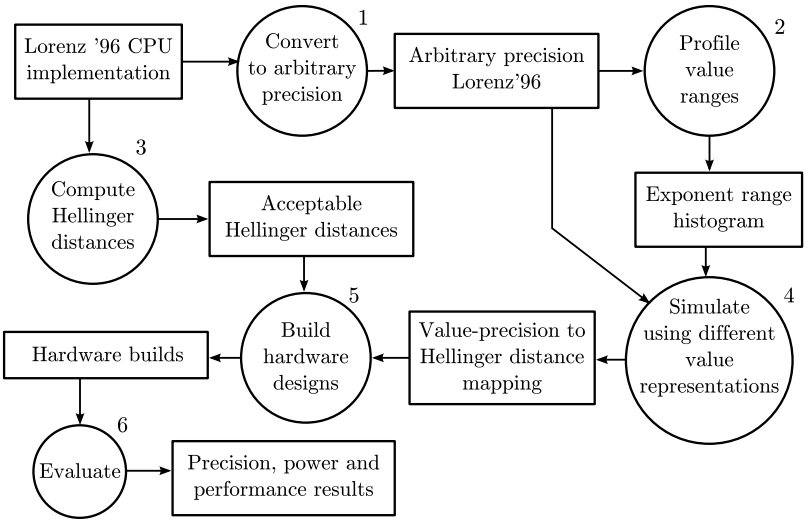


Fig. 4. Our methodology for determining the precision of Lorenz '96 hardware builds. CPU-based simulations are used to determine acceptable Hellinger distances and the expected Hellinger distances of hardware builds at various levels of precision.

4.4. Implementation

We extended the Lorenz '96 C++ implementation to support reduced precision in floating and fixed point. To match the hardware implementation of floating point, we require behaviour equivalent to IEEE 754 with round-to-nearest, no denormalised value support and non-standard mantissa lengths. We originally used an arbitrary precision library but required higher performance for long runs so we approximated reduced precision with round-to-nearest, tie-to-even behaviour through bit manipulation of double-precision values. Since we have no access to the "sticky" bits of the underlying implementation, we will sometimes invoke tie behaviour when a correct implementation would round. We note that the rounding mode has significant impact on the behaviour of this model at low precisions. Using truncation resulted in larger Hellinger distances at lower precision. A previous implementation used round-to-nearest, tie away from zero rounding behaviour (which is not an IEEE 754 rounding mode but is unbiased and was efficiently implementable). However, this resulted in generation of NaNs (not a number) due to the magnitude of coefficients being repeatedly increased via rounding and therefore could not be used for simulations.

For fixed point, we emulate round-to-nearest behaviour with ties rounding away from zero. We note that this behaviour is significantly less problematic for fixed-point implementations than floating-point ones — larger values are less affected by rounding than smaller ones due to the non-normalised representation. If we attempted to optimise the fixed-point representation used at the granularity of each intermediate value we expect we would need to correctly emulate tie behaviour.

Since the hardware and reduced-precision CPU implementations differ in order and implementation of arithmetic operations, the results from each will diverge. Also, the chaotic nature of Lorenz '96 causes even tiny perturbations to lead to divergence. However, we can use the CPU implementation as a tool to analyse the effect of different local and global-scale value mantissa widths on the Hellinger distance.

4.5. Exponent profiling

Runtime profiling of floating point exponent values in the CPU simulation was performed for 8000 time-steps (Fig. 5). Results suggest that base-2 exponent values of up to 10 and 11 are required to capture the largest global and local-scale values, respectively.

For our floating-point hardware implementation, the distribution indicates that exponent widths of 5-bits for both global and local-scale values can represent the majority ($\geq$ 99.999%) of values generated during a simulation. Exponents for custom floating-point types in our hardware design are represented in a biased manner similar to those in IEEE 754, so an exponent field of width 5 can represent exponents in the range -14 to 15, inclusive. As in IEEE 754, the lowest exponent is reserved for representing zero (denormal numbers are unsupported by our hardware synthesis toolchain) and the highest exponent used for representing NaNs and infinities.

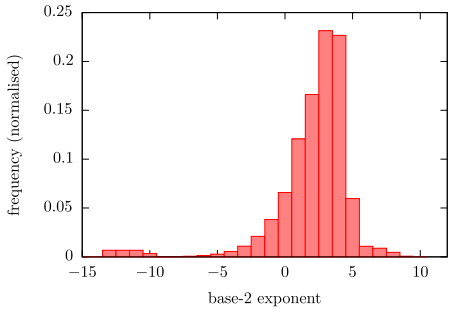
We also use these analyses to determine the ranges of the fixed-point representations. In hardware, fixed-point values have a two's complement representation with a shifted binary point. We choose offsets (the number of bits to the left of the binary point excluding the sign bit) of 11 and 12 for the global and local-scale values, respectively, which provides leeway for representing double the maximum magnitudes observed in the CPU simulations.

4.6. Floating point analysis

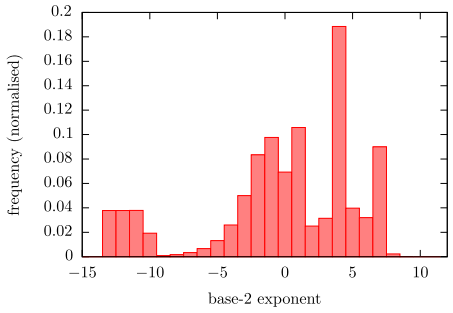
We plot the Hellinger distances between the PDFs of the elements of the global and local-scale states against a double-precision implementation (Fig. 6). We see expected behaviour — that reduced precision leads to greater error. Although the probability distribution of global-scale values appears minimally affected by the use of small ($\leq$ 8 bits) mantissas for local-scale values, this is misleading since the probability distribution of local-scale values has been significantly altered. The results also confirm previous work [20] which concludes that the precision of values at different scales can be optimised independently, which is important for scaling our analysis to more complex models.

To determine the extent to which precision can be reduced, we must choose an acceptable Hellinger distance. We vary c and F in order to determine the Hellinger distances that correspond to a chosen levels of uncertainty in the model parameters. The abstract nature of Lorenz '96 means that parameters cannot be compared directly to a full atmospheric circulation model so it is necessary to make estimates regarding what constitutes realistic levels of uncertainty.

The coupling parameter c can be interpreted as an internal model parameter that is responsible for the coupling between processes (such as the surface layer drag in climate models). Parameter F can be interpreted as boundary condition (such as CO_2 forcing

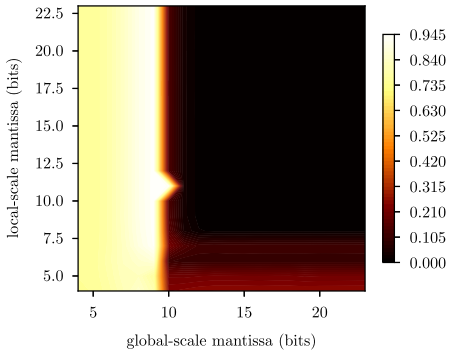


(a) Global-scale.

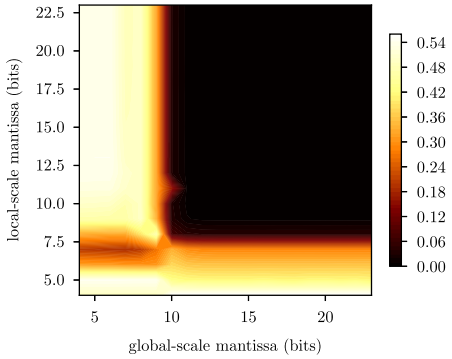


(b) Local-scale.

Fig. 5. Exponent (base-2) distributions for global and local-scale values derived from CPU simulation with $F = 40$, $b = 10$ and $c = 10$. The profile includes all intermediate values calculated in the specified precision.



(a) Global-scale distance.



(b) Local-scale distance.

Fig. 6. Hellinger distances (smaller is better) for X and Y state values between a double-precision CPU implementation and the CPU implementation using reduced-precision floating point for local and global-related values. $F = 40$, $b = 10$, $c = 10$, $J = 64$, $K = 2000$ and the simulation was run for 3,750,000 time-steps with the state sampled every 1000 time-steps. In Lorenz '96, local-scale values only influence global scale values via summation, which is why the Hellinger distance for global-scale values does not increase significantly when local-scale values use mantissa widths less than 10 bits, despite the loss of correct local-scale dynamics.

Table 1
Hellinger distances (4 significant figures) between a double-precision simulation and the same with different initial conditions (averaged from five runs) and altered parameters.

Run	H (global)	H (local)
Changed initial conditions	2.788e−3	1.939e−4
$c \times 0.99, F \times 0.99$	4.605e−3	1.505e−3
$c \times 1.01, F \times 1.01$	5.042e−3	1.719e−3
$c \times 0.9, F \times 0.9$	4.047e−2	1.625e−2
$c \times 1.1, F \times 1.1$	3.620e−2	1.415e−2

Simulations were run for 3,750,000 timesteps with $J = 144$ and $K = 2000$.

in the atmosphere or the heating rate of a combustion engine in Computational Fluid Dynamics). Parameters b and h remain untouched in our tests to keep the complexity of the tests limited. It is a realistic assumption that some model parameters will be uncertain (e.g. viscosity and cloud parameters in atmospheric models) while other parameters are known with high certainty (e.g. heat capacity).

We calculate Hellinger distances between the double-precision reference, the same with changed initial conditions, and with c and F altered by 1% and 10% (Table 1). For reduced-precision floating point we estimate minimum mantissa widths (Table 2). We assume that a change of c is a good representation for parameter uncertainties while a change of F is a good representation for uncertainties in boundary conditions. We consider 1% to be realistic for most numerical models in Computational Fluid Dynamics where uncertainties are low and 10% to be reasonable for atmospheric applications where uncertainties are large due to the complexity of the Earth system [22].

4.7. Fixed point analysis

In order to perform the fixed-point analysis, we must first choose the binary point offsets of the fixed-point representations used for local and global-scale values. We require that the binary point offsets are sufficient to allow the largest local and global-scale values to be represented. The largest values we have observed

Table 2
Estimated minimum floating point mantissa sizes for local and global scale values required to produce a similar Hellinger distances to the corresponding simulations in Table 1.

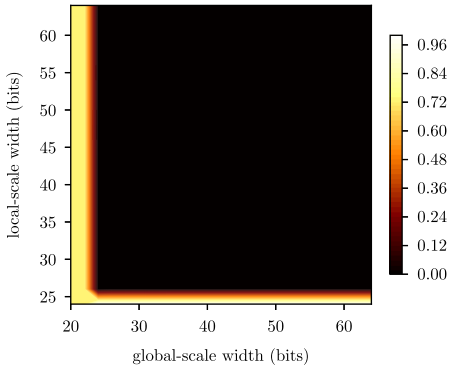
Corresponding run	Estimated H (global)	Estimated H (local)	Estimated min.mantissa (global)	Estimated min.mantissa (local)
Changed initial conditions	2.772e−3	1.587e−4	15	12
$c \times 0.99, F \times 0.99$	3.153e−3	7.094e−4	13	10
$c \times 1.01, F \times 1.01$	3.153e−3	7.094e−4	13	10
$c \times 0.9, F \times 0.9$	3.612e−2	1.108e−2	11	9
$c \times 1.1, F \times 1.1$	3.612e−2	1.108e−2	11	9

We estimate minimum mantissa widths required to produce a simulation with smaller global and local-scale Hellinger distances than those in Table 1. Where multiple combinations of global and local-scale mantissa lengths satisfy the requirements, we choose the configuration with the smallest combined mantissa length. The Hellinger distances shown are our estimates of those that will be produced by the corresponding hardware build.

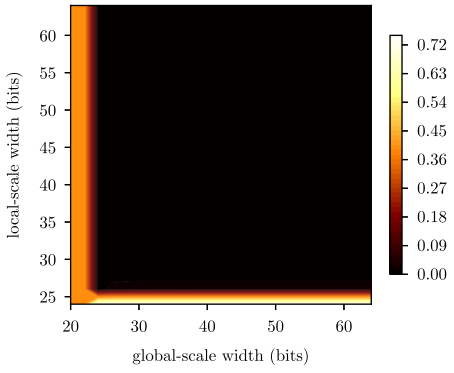
Table 3
Estimated minimum fixed point significand width for local and global scale values required to produce similar Hellinger distances to the corresponding simulations in Table 1.

Run	H (global)	H (local)	Est. min. bits (global)	Est. min. bits (local)
Changed initial conditions	–	–	–	–
$c \times 0.99, F \times 0.99$	3.589e−3	3.957e−4	24	26
$c \times 1.01, F \times 1.01$	3.589e−3	3.957e−4	24	26
$c \times 0.9, F \times 0.9$	3.589e−3	3.957e−4	24	26
$c \times 1.1, F \times 1.1$	3.589e−3	3.957e−4	24	26

We estimate minimum fixed point mantissa widths required to produce a simulation with smaller global and local-scale Hellinger distances based on the results from CPU simulation. Where multiple combinations of global and local-scale significand lengths satisfy the requirements, we choose the configuration with the smallest combined significand length. The Hellinger distances shown are our estimates of those that will be produced by the corresponding hardware build. Even our highest-precision fixed-point simulation (64-bits for local and global-scale significands) does not satisfy the requirement of lower global and local-scale Hellinger distances than those in Table 1 for a simulation with changed initial conditions.



(a) Global-scale distance.



(b) Local-scale distance.

Fig. 7. Hellinger distances (smaller is better) for X and Y state values between a double-precision CPU implementation and the CPU implementation using fixed-point representations for local and global-related values. $F = 40$, $b = 10$, $c = 10$, $J = 64$, $K = 2000$ and the simulation was run for 1,875,000 time-steps with the state sampled every 1000 time-steps.

for local and global scale exponents are 11 and 10, respectively. We increase these by two to account for both representation of the implicit bit in floating point, and to provide some room for representing values outside the range observed. Unlike for floating point, we have no means to detect that an overflow situation has occurred. In a more robust implementation we might signal the host system to handle overflows in software.

In the same manner as the floating-point analysis, we plot Hellinger distances between the PDFs of the elements of the global and local-scale states against a double-precision implementation (Fig. 7). Similar to the floating-point simulations, there is a clear range of representations at which behaviour of the system entirely breaks down.

Using our fixed-point CPU simulations, we estimate the number of bits required for each value type in order to achieve various Hellinger distances (Table 3). In contrast to the floating-point case, we find that above the range that behaviour breaks down, adding additional bits does not strongly correlate with improved Hellinger distances. Corresponding, we end up choosing the same significant widths for all potential fixed-point designs.

Since we have not attempted to optimise fixed-point representations at the level of intermediate values, the width of significand required is primarily a function of value range rather than precision. We theorise that above 26 bits, wider significands are contributing useful precision to only the smallest values represented in the simulation.

Although we fail to find a fixed-point representation for which we consider the simulation to have identical or greater precision to the reference simulation, we note that the desired Hellinger distances are of similar magnitude to the estimated Hellinger distances of the design we choose to build.

5. Architecture

5.1. System overview

The entire Lorenz '96 Runge–Kutta update is moved to a hardware accelerator. The system state is copied from host memory to the accelerator's off-chip memory prior to computation. Arbitrary numbers of Runge–Kutta updates can be performed with the system state sent back to the host when needed.

All computations are moved to hardware, but host involvement remains during updates. Memory controller commands are sent by the host to the accelerator during computation since data access patterns for multiple iterations (Section 5.6) are expensive to compute in hardware. The data-volume is small, so transfer time is not a limiting factor.

The structure of our design is shown in Fig. 8. The padding/stripping steps add/remove padding needed to satisfy memory controller constraints on read/write sizes. The deinterleave/interleave steps split/combine the X and Y state so that sending an X element can be synchronised with sending the first element of the associated Y state. Each Runge–Kutta step is implemented by a single kernel, computing the updated partial X and Y states and intermediate values.

5.2. In-memory representation

We treat the state of a Lorenz '96 system as a matrix (Fig. 9). The matrix is flattened to memory in column-major order so that all Y values associated with a particular X value are located

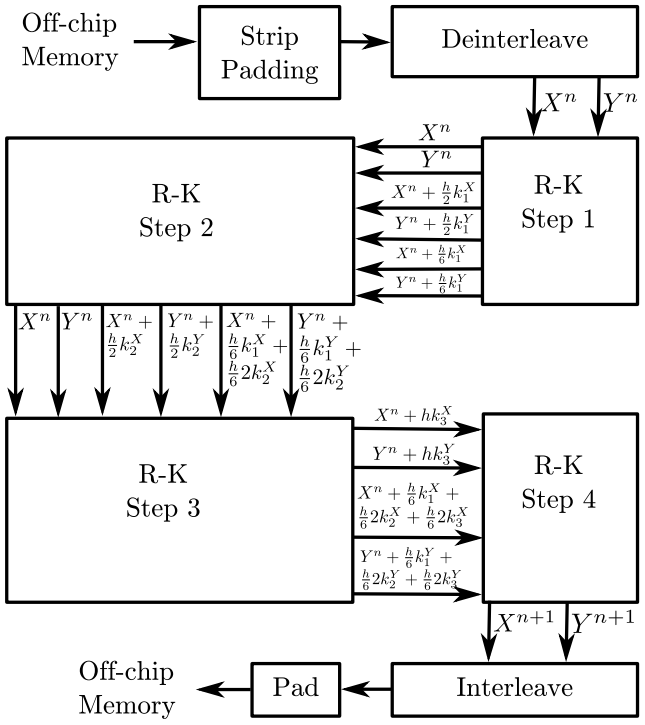


Fig. 8. The hardware structure of a Runge–Kutta time-step for Lorenz '96. X^n and Y^n represent the state of the system at time-step n. k_i^X and k_i^Y represent the ith Runge–Kutta increment for the X and Y states, respectively. Expressions containing the factor $\frac{h}{6}$ represent intermediate values in the computation of $X^{n+1} = X^n + \frac{h}{6}(k_1^X + 2k_2^X + 2k_3^X + k_4^X)$ and $Y^{n+1} = \frac{h}{6}(k_1^Y + 2k_2^Y + 2k_3^Y + k_4^Y)$. All other expressions are Runge–Kutta estimated slopes. Note that the pipeline is linear flow, and that the snaking nature of the figure is solely to maximise space utilisation.

contiguously in memory. We interleave the Y and X state such that each X-value is located before the associated Y values.

5.3. Data path

For clarity, we describe the Runge–Kutta method for calculating y_{n+1} from y_n given the function f that calculates the gradient of y at t such that $\frac{dy}{dt} = f(t, y)$:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5}$$

where:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + 0.5h, y_n + 0.5hk_1) \\ k_3 &= f(t_n + 0.5h, y_n + 0.5hk_2) \\ k_4 &= f(t_n + h, y_n + hk_3). \end{aligned} \tag{6}$$

The kernels for each Runge–Kutta step are similar but differ in the number of inputs and outputs (e.g. all steps except the first take an input containing intermediate values of X^{n+1} and Y^{n+1}). Each step is specialised to avoid unnecessary computation and communication. X and Y remain deinterleaved during processing, but are synchronised so that each X entry is sent/received by kernels simultaneously with the first element of the associated Y-values.

In our design, all values are stored in off-chip memory as IEEE 754 single-precision floating-point values. Global and local-scale quantities are converted to and from lower precision in the "deinterleave" and "interleave" kernels, respectively (Fig. 8).

All intermediate values such as derivatives and slope estimates are maintained in the same precision as the associated X or Y state. For the summation of Y elements needed by the X-derivative, we use an accumulator of the same type as the X elements for improved precision.

$$\begin{pmatrix} X_1 & X_2 & \cdots & X_K \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ Y_{1,1} & Y_{1,2} & \cdots & Y_{1,K} \\ Y_{2,1} & Y_{2,2} & \cdots & Y_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{J,1} & Y_{J,2} & \cdots & Y_{J,K} \end{pmatrix}$$

Fig. 9. State of the Lorenz '96 system arranged as a matrix. Each column contains a single X element and all Y elements associated with the X element. Each column may contain zero or more padding elements after the X element due to vectorisation (Section 5.5).

5.4. Boundary conditions

Calculation of the X and Y derivatives both involve stencil operations with circular boundary conditions. These are problematic to implement since a value to be computed may require information from both the start and end of a stream.

The most significant impact on the design is caused by the X stencil, which requires access to elements from X_{k-2} to X_{k+1} to update X_k. Since X and Y are coupled (and interleaved), anything that causes buffering of an element of X_k will require buffering of the column $Y_{*,k}$ which is undesirable for large J.

The Lorenz '96 state can be visualised as a matrix (Fig. 9) containing both X and Y. We will call the previous Lorenz '96 state S and the new state S'. Three types of value propagate through the Lorenz '96 hardware pipeline — the original state (S), the partially computed value of S' and the Runge–Kutta increments. For ease of reference, we will call the composite intermediate state which enters each Runge–Kutta hardware block S^+. In practice, S^+ is carried by multiple streams between hardware blocks. For the first Runge–Kutta hardware block, $S^+ = S$ and consists of two streams carrying X and Y, but for later blocks multiple input streams are used and each element of S^+ logically corresponds to a tuple of values. Hence, buffering a column of S^+ corresponds to buffering Jn values where n depends on the particular Runge–Kutta hardware block in question.

Handling of stencils in our design is done via two strategies:

5.4.1. Buffering

Since the calculation of dX_k has a dependency on X_{k+1} (which will be read after X_k), the value of dX_k cannot be output until this value has been received. To ensure that X and Y remain in sync throughout the pipeline (Fig. 8) this requires each Runge–Kutta hardware block to buffer a column of S^+.

5.4.2. Pre-reading

The calculation of dX_k also has dependencies on X_{k-1} and X_{k-2} (which will be read before X_k). If we start reading the system state at $k = 1$, then the first result we can output will be for $k = 3$ since we do not have access to the values X_{K-1} and X_K to calculate dX_1 and dX_2. As a consequence of this, each Runge–Kutta hardware block induces a "rotation" of the system in K of size 2. Hence, an entire Runge–Kutta update causes the Lorenz '96 system to be rotated by a factor of 8. Since the system is circular, this has no impact on the calculation and can be undone by the host after the system state is copied back.

This rotation to the system means that the values dependent on dX_1 and dX_2 appear at the end of the stream after the first Runge–Kutta intermediate update. Again, to ensure that X and Y remain in sync, this would appear to require each Runge–Kutta hardware block to buffer two columns of S^+. Instead we implement a strategy of pre-reading, whereby we read some of the initial columns of the system state *twice*.

Each Runge–Kutta hardware block requires that two columns be read twice, meaning that an entire Runge–Kutta update requires that 8 columns be pre-read. On the first read, each Runge–Kutta hardware block discards the Y-associated components of $S^+_{*,1}$ and $S^+_{*,2}$. The second time $S^+_{*,1}$ and $S^+_{*,2}$ are received, they are used for computation.

The first Runge–Kutta block receives $S^+_{*,1}$, …$S^+_{*,K}$, $S^+_{*,1}$, …$S^+_{*,8}$. The second receives $S^+_{*,3}$, …$S^+_{*,K}$, $S^+_{*,1}$, …$S^+_{*,8}$. The final receives $S^+_{*,7}$, …$S^+_{*,K}$, $S^+_{*,1}$ …$S^+_{*,8}$ and produces $S'_{*,9}$, …$S'_{*,K}$, $S'_{*,1}$ …$S'_{*,8}$. Each hardware block produces 2 less columns of S^+ than it receives. Hence, all Runge–Kutta blocks receive more data than strictly necessary but the final block produces output of the correct size.

5.5. Vectorisation

To maximise throughput we vectorise handling of the Y state (by creating multiple in-kernel data-paths). Alternatively we could have replicated the Runge–Kutta pipeline, but the vectorisation approach has a number of benefits (described later).

Our design is constructed such that we can choose an arbitrary vector length v_n representing the number of elements read from a Y-related stream in a single cycle (though the memory controller places limitations on this since only certain read sizes are efficient). We enforce the requirement that J is a multiple of v_n.

The arithmetic required to implement the Y-derivative calculation is duplicated v_n times. Since we do not duplicate the pipeline, we generate no additional counters nor communication channels between kernels. Most importantly, the sizes of the buffers required to store and delay Y-related values are nearly independent of v_n so increasing v_n has minimal effect on on-chip memory usage.

Calculation of the X-derivative involves a summation over each column of Y. We implement an accumulator that accepts one element per cycle, which is relatively costly in terms of adders. With vectorisation we avoid duplicating the accumulator; instead we use another $v_n - 1$ adders to sum the vector elements for input to the accumulator.

One overhead incurred is that each X-state element must be padded to v_n elements. For an arbitrary system, this means that the fraction of padding of an in-memory representation of the system will be $\frac{v_n-1}{J+1}$. For a typical value of $J = 128$, a system where $v_n = 8$ will consist of 5.4% padding and where $v_n = 16$, it will have 11.6% padding.

5.6. Multiple iterations

We can perform multiple Lorenz '96 time-steps with only a single invocation from the host machine. As described in Section 5.4, a "pre-read" region is read at both the start and end of a time-step update. Therefore, the pre-read region cannot be overwritten until it has been read both times.

The memory controller requires all reads and writes to have sizes and occur at addresses which are multiples of a *burst size* (384 bytes). During a time-step update, the new state is written starting at the first burst that does not contain any pre-read data, causing the start of the Lorenz '96 state to move forward in memory each time-step.

The Lorenz '96 state is padded to the memory controller's burst size and circular addressing is used so that the flattened representation of the system state (including padding) "rotates" in memory by multiples of the burst size; rotation is undone by the controller on state transfer to the host. Fig. 10 illustrates the memory layout transformation caused by the first time-step.

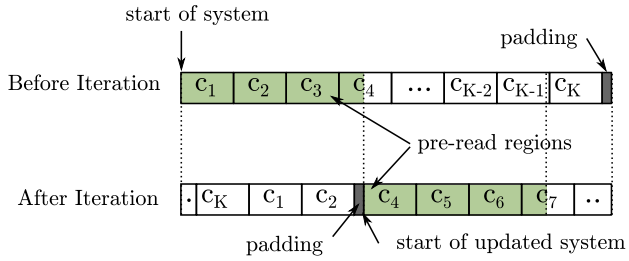


Fig. 10. Rotation of the Lorenz '96 system state caused by a single time-step update. We show a rotation by 3 columns rather than the actual 8. Shaded areas indicate the pre-read regions. Dashed lines indicate burst size aligned addresses. The updated state is written in front of the pre-read region causing the flattened state to rotate in-place. The size of the pre-read region is the minimum burst size multiple capable of holding the number of columns to be pre-read.

6. Results

Hardware simulations[1] were performed on a Maxeler MAX3A Vectis Dataflow Engine (DFE) which contains a Xilinx Virtex-6 SX475T FPGA and 24 GB of DDR3 RAM (distinct from CPU-accessible memory). The SX475T has approximately 4.7 MB of dedicated on-chip memory. Connection to the host was via PCI express. All designs were compiled to run at 150 MHz. All designs were written using the Maxeler's MaxJ language and compiled with version 2013.2.2 of Maxeler's MaxCompiler.

The host machine for the MAX3A card (denoted "System 1") was a hyper-threaded four-core Intel Core i7 870 running at 2.93 GHz with 8 MB L3 cache, 16 GB RAM and a single MAX3A Vectis card. Total idle power consumption of 93 W.

Four floating-point and two fixed-point DFE designs were built for benchmarking and precision analysis. For floating point, designs were built at single precision, one at the precision estimated necessary for similar accuracy to single precision and two at the precision level estimated to have errors of the same order of magnitude as those caused by 1% uncertainty in the input parameters. For fixed point, two designs were built at the precision level estimated to have errors of the same order of magnitude as those caused by 1% uncertainty in the input parameters.

The vector width (v_n) of a build controls how many Y-related elements we can read and write per tick. Aside from resources, v_n is also constrained by the memory controller. An individual stream to/from DRAM can read/write a maximum of 192 bytes per cycle. Other read and write sizes are possible, but must be perfectly divisible into 192 for there not to be a massive increase in design complexity. We store values in 32-bit floating point, so are constrained to vector sizes that are perfectly divisible into 48.

Our architecture requires that J be chosen to be a multiple of the vector width. To enable comparison with the Hellinger distances in Table 1, designs FLOAT3 and FIXED1 were built with a reduced vector width. FLOAT3 and FIXED1 can be run with $J = 64$ permitting comparison with the CPU simulations but FLOAT4 and FIXED2 cannot since 64 is not divisible by the vector width (24).

All designs (except FLOAT3 and FIXED1) were compiled with the maximum supported vector width (Table 4). The notation float(e, m) denotes a floating-point type with an e-bit exponent and m-bit mantissa, including the implicit bit. The notation Q(n,

[1] Throughout this section we use the term "simulation" in the scientific sense. All timing and power utilisation results were collected through execution on actual hardware.

Table 4
Precision, vector widths and resource utilisation of Lorenz '96 DFE builds.

Build	Global-scale type	Local-scale type	Vector width	Utilisation (%)		
				Logic	DSP	BRAM
FLOAT1	float(8, 24)	float(8, 24)	8	55.00	48.16	24.25
FLOAT2	float(6, 15)	float(5, 12)	16	69.85	32.84	28.67
FLOAT3	float(5, 13)	float(5, 10)	16	64.28	32.84	27.63
FLOAT4	float(5, 13)	float(5, 10)	24	79.15	47.92	33.74
FIXED1	Q(11, 12)	Q(12, 13)	16	41.87	63.00	28.95
FIXED2	Q(11, 12)	Q(12, 13)	24	49.31	93.15	36.61

Table 5
Hellinger distances (4 significant figures) against a double-precision CPU implementation.

Build	$J = 64$		$J = 144$	
	H (global)	H (local)	H (global)	H (local)
Changed initial cond.	2.788e−3	1.939e−4	7.029e−3	6.291e−4
$c \times 0.99, F \times 0.99$	4.605e−3	1.505e−3	1.399e−2	1.726e−3
$c \times 1.01, F \times 1.01$	5.042e−3	1.719e−3	1.067e−2	1.861e−3
$c \times 0.9, F \times 0.9$	4.047e−2	1.625e−2	1.167e−1	1.487e−2
$c \times 1.1, F \times 1.1$	3.620e−2	1.415e−2	8.826e−2	1.236e−2
FLOAT1	2.934e−3	2.881e−4	7.472e−3	1.180e−3
FLOAT2	2.776e−3	2.707e−4	4.320e−2	2.443e−3
FLOAT3	3.267e−3	6.742e−4	7.289e−2	1.012e−2
FLOAT4	N/A	N/A	7.404e−2	1.005e−2
FIXED1	3.123e−3	2.361e−4	2.321e−2	1.885e−3
FIXED2	N/A	N/A	2.321e−2	1.885e−3

$F = 40, b = 10, c = 10, J = 120, K = 2000$ and the simulation was run for 3,750,000 time-steps with the state sampled every 1000 time-steps. Results are provided for both $J = 64$ (as in Table 1) and $J = 144$. Since J must be a multiple of the vector width we cannot calculate Hellinger distances for FLOAT4 where $J = 64$.

m) denotes a fixed-point format with n integer bits (excluding the sign bit) and m fractional bits so that Q(11, 12) corresponds to a fixed-point type represented by 24 bits.

The maximum value of J each design could support was reduced to 512 since larger values are of minimal interest to climate scientists and this improves the chances of routing. FLOAT2 was compiled with a 6-bit exponent for global-scale values due to restrictions on mantissa-exponent size combinations imposed by the Maxeler tools. We compiled a design using float(5,13) for both global and local-scale values with a vector width of 16 and a maximum value of J of 128,000, demonstrating we can scale J much higher if necessary.

6.1. Precision

We calculate the Hellinger distances for the local and global-scale state variables between each DFE implementation and a double-precision CPU implementation (Table 5).

For $J = 64$, FLOAT1 and FLOAT2 have Hellinger distances similar to those seen between double-precision simulations with changes to the initial conditions. FLOAT3, which has the lowest precision, exhibits a Hellinger distance comparable to that caused by a 1% variation in c and F (Table 1). FIXED1 (which has the same throughput as FLOAT3) exhibits lower Hellinger distances, particularly for local-scale values.

We also show Hellinger distances for $J = 144$ (the least common multiple of vector widths used that is ≥ 128). For FLOAT4 our Hellinger distances are outside those expected by varying input parameters by 1% (we expect larger distances due to having optimised for a different J) but within those expected by 10% variation. FIXED2 shows significantly lower Hellinger distances compared to FLOAT4 in the case that $J = 144$. Whilst these are larger than those expected by varying the input parameters by 1%, they have similar magnitudes and are much less than those expected by varying the input parameters by 10%.

Table 6
Speed-up factors.

Build	Speedup (3 significant figures) FLOAT1 (DFE, single precision)
FLOAT1	1.00
FLOAT2/FLOAT3/FIXED1	1.90
FLOAT4/FIXED2	2.46

Speed-up factors relative to a single-precision DFE implementation. $J = 144$ and $K = 819{,}200$.

We note that for the case $J = 144$, FIXED1 and FIXED2 have identical Hellinger distances in contrast to FLOAT3 and FLOAT4 which do not. Changing the vector width of a Lorenz '96 floating-point design will cause it to produce different results, even if it is built at the same precision, due to the non-associativity of floating-point addition. Changing the vector width of a Lorenz '96 fixed-point design will not change the result since the associativity of addition is preserved with the fixed-point representation.

Although our fixed-point design has improved accuracy compared to a floating-point design with the same throughput and similar resources, it is difficult to make claims regarding the suitability of fixed point versus floating point for chaotic models. Unlike floating point, where the value bit-width correlates with precision, the width of the fixed-point implementation most strongly correlates with value range since we do not have the ability to optimise each fixed-point value to the dynamic range of the values it represents.

We do observe in the floating-point implementation that increased mantissa sizes lead to improved Hellinger distances. In contrast, in the fixed-point implementations, once the width of fixed-point values are sufficiently high for the system to not behave incorrectly, increased width does not appear to reduce Hellinger distances further.

6.2. Performance

We compare performance between different DFE builds (Fig. 11) with computational throughput and bandwidth shown in Table 7 and speedup factors shown in Table 6. FLOAT3 is excluded since FLOAT4 has the same precision but also higher throughput –FLOAT3 was built to enable a Hellinger distance comparison with the original CPU-based estimates which used $J = 64$ which FLOAT4 does not support due to its vector width.

For reporting arithmetic throughput, we calculate the number of operations required to perform a single time-step of Lorenz '96 using Runge–Kutta time-stepping as: $JK(15m + 23a) + (19m + 23a)K + 21m$, where a and m represent performed additions and multiplies.

6.3. Power consumption

We measure the power requirements of the FPGA implementations on system 1. Total workstation power consumption was measured when running each implementation in different

Table 7
Computational and memory throughput (3 significant figures) for CPU and DFE implementations.

Build	Throughput (elements/s)	GFLOPS	Valuable Band-width (GB/s)	Actual Band-width (GB/s)
FLOAT1 (System 1)	1.14e9	43.5	9.16	9.54
FLOAT2 (System 1)	2.17e9	82.7	17.4	19.1
FLOAT4/FIXED2 (System 1)	2.81e9	107	22.5	25.9

Actual bandwidth refers to the combined rate of communication to and from DRAM. Valuable bandwidth refers to the rate of communication assuming that the amount of data communicated is twice the system size (one read, one write). Valuable bandwidth on the FPGA is reduced compared to actual bandwidth due to the need for padding.

Table 8
System throughput, total power consumption and efficiency (3 significant figures) for CPU and DFE implementations.

Build	Throughput (elements/s)	Power (W)	Efficiency (elements/J)	Relative Efficiency
FLOAT1 (System 1)	1.14e9	137	8.35e6	1.00
FLOAT2 (System 1)	2.17e9	143	1.52e7	1.82
FLOAT4 (System 1)	2.81e9	146	1.92e7	2.30
FIXED2 (System 1)	2.81e9	144	1.95e7	2.33

Throughput is specified in updated elements per second where an element is a single scalar from the Lorenz '96 state produced by a full Runge–Kutta update.
Relative efficiency uses FLOAT1 (the single-precision DFE implementation) as the baseline.

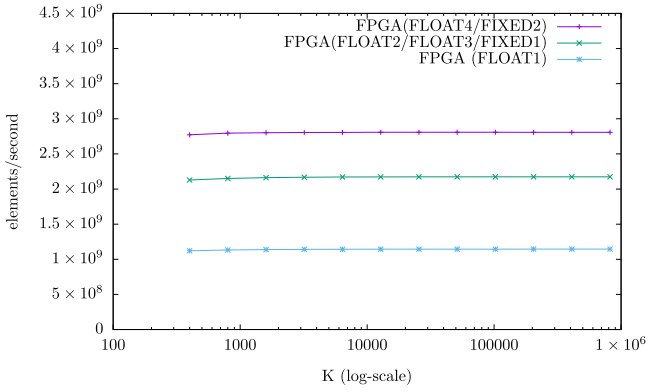


Fig. 11. Throughput of DFE implementations in elements/second for $J = 144$ and varying values of K. We use the term *element* to refer to a single scalar from the Lorenz '96 state produced by a full Runge–Kutta update. The DFE simulations were compiled using the configurations in Table 4.

configurations (Table 8). We subtract the static power requirements of the unused Maxeler cards from our reported figures (the average reported static power requirement per card was 19.6 W).

In terms of computation per Watt, the reduced-precision designs achieve power efficiency of up to 2.3 times higher than a single-precision DFE implementation. We do not observe any significant difference in power requirements between comparable fixed and floating-point builds. Assuming perfect scaling (not unreasonable considering the need in weather forecasting to calculate *ensembles*) System 1 could reach a relative efficiency of 3.7 times over a non-precision optimised FPGA implementation if it contained three fully utilised MAX3A cards.

7. Conclusion and future work

We designed a hardware architecture for the chaotic two-scale Lorenz '96 system. We built FPGA implementations at different precision levels, guided by our CPU-based analysis of Hellinger distances and showed that acceptable behaviour can be achieved at precision significantly lower than single precision. We demonstrated how to reduce precision of a chaotic system to improve throughput of a hardware implementation.

The performance improvements achieved have enabled climate scientists to substantially increase the scales of analyses involving the Lorenz '96 system with and without reduced precision. Previous work in this area was limited to using simulations to explore the latter [20].

We note that Lorenz '96 was chosen as a system primarily to study the effects of precision reduction on chaotic systems and evaluate the ability to exploit reconfigurable hardware for simulating such systems.

Our results indicate that exploiting chaotic behaviour and parameter uncertainty in numerical simulations can facilitate more effective resource utilisation in hardware designs, leading to improved performance and power efficiency.

Impact of future hardware developments

Given the rapid evolution of accelerator technology, we attempt to address the question of how the applicability and effectiveness of our techniques will be impacted by future hardware changes. Although it is difficult to predict future hardware changes, we consider two recent developments: GPU support for half-precision arithmetic and improved FPGA support for single-precision arithmetic.

Computation characteristics can significantly affect the power efficiency and performance comparisons between FPGAs and GPUs [15]. In addition, improvements in fabrication technology also affect power efficiency and resource availability.

This initial work with Lorenz '96 does not explore the granularity at which reduced precision can be applied, nor range of precisions that might be necessary in order to reduce computation requirements of a more physically meaningful model while retaining fidelity. Both will have a significant impact on the performance that may be achieved on a GPU or FPGA.

We note that half precision has a 10-bit mantissa and 5-bit exponent, which happens to match the lowest fidelity representation we explore in this work. If we happened to require slightly higher precision on the GPU, we would need to switch to single precision. In contrast, the reconfigurable nature of the FPGA enables a far more fine-grained approach to precision reduction which may prove beneficial.

We note that GPUs are typically only competitive with FPGAs in terms of power efficiency for compute-bound applications whereas finite difference computations tend to be bandwidth limited. Using half precision on the GPU makes it possible to improve throughput by reducing data-transfer requirements, but we expect that the choice between FPGAs and GPUs will still largely depend on current deciding factors such as how efficient a parallel solution can be mapped to a given architecture.

More recent FPGAs such as the Intel Arria 10 and Stratix 10 families have hardened floating point blocks that provide support

for IEEE 754 single-precision floating point. By incorporating this support directly into DSP (digital signal processing) blocks, single-precision floating point becomes significantly more attractive in terms of performance and power efficiency.

We foresee that improved FPGA single precision support will complement our techniques so long as it is possible to perform reduced-precision arithmetic using the hardened DSP blocks. The primary concern is that of double rounding, but such issues can be avoided provided the interface to the DSP blocks is sufficiently flexible, or can be worked around so long as the DSP blocks provide the appropriate IEEE 754 flags [24].

Even with hardware single-precision arithmetic support, wider floating-point types also impact designs in terms of increased routing resources which lead to congestion and the inability to meet timing constraints. Therefore, we expect reduced precision to still provide benefits for hardware designs.

Methodology extension

Current and future work includes extending our methodology to more physically meaningful systems such as a C-grid shallow water or primitive equation model. In principle, the same analysis that was performed for Lorenz '96 in this paper could be repeated for a more realistic model. However, in order to evaluate the effect of reduced precision on Lorenz '96, we had to use statistical measures that were very compute intensive to evaluate. Performing a similar analysis for more complex systems may require an analysis at every single grid point. The CPU-emulated reduced precision analysis used in this paper is almost certainly impractical for such an analysis due to the overhead of simulating reduced precision.

An alternative approach would be to construct FPGA implementations in which the precision of values can be altered after build time, but prior to execution of the scientific simulation. This would significantly mitigate the cost of the precision analysis phase which requires large numbers of samples be collected in order to produce meaningful statistics.

An interesting research question for atmosphere and ocean models is whether numerical precision can be adjusted locally in both weather and climate simulations in a way that is dependent on, for example, local Reynolds numbers or uncertainties in initial conditions to achieve an even stronger reduction in numerical precision. If it is determined that numerical precision should vary in different regions during a simulation, it will be necessary to investigate efficient runtime reconfiguration of FPGAs in order that load balancing can be performed with reduced precision.

In practice, it is useful to have evidence that the simulation has not been adversely affected by precision reduction. Molecular dynamics (MD) simulations also possess chaotic behaviour that causes precision reduction to significantly alter particle trajectories after only a few collisions. An implementation of reduced-precision MD on FPGAs has adopted the approach of monitoring invariants such as the total energy of the system in order to perform runtime validation [25]. A similar approach would allow us to validate simulation behaviour, but also potentially enable switching to a higher-precision implementation when it is detected that additional precision is required.

Increasing model complexity

We foresee the development of new tools to be an important aspect of future research. In particular, the ability to automatically generate new hardware designs from high-level descriptions of equations will significantly accelerate progress towards implementing more complex systems on reconfigurable hardware. If designs can be generated from high-level descriptions, it opens the possibility of automating modifications to those designs for performing precision-related profiling and automated numerical analysis of model components.

As model complexity increases, the hardware resources required to implement the model increase also and it may not be possible to synthesise a design containing a complete model implementation. Additionally, some aspects of climate simulations may be impractical to implement in a hardware design. We suggest ways to address both these issues.

Various strategies may be employed to address limited FPGA resources. If the computation can be expressed as a pipeline, multiple FPGAs can be directly connected together so that each one can perform a phase of the computation without data needing to be written to memory. As an example, the FPGA boards we use manufactured by Maxeler provide a ring interconnect called "MaxRing". In addition, as described in Section 2, many data centres now support multi-FPGA accelerators, so we envisage that large models can be partitioned and processed by multiple FPGAs effectively as long as overheads in communication among them are minimised.

A second strategy is *partial reconfiguration*. This technique allows subsets of the FPGA design to be reconfigured at run-time. In this way, FPGA resources for one calculation can be re-used for another, but both calculations cannot occur simultaneously. Runtime reconfiguration incurs overhead, but provides an additional way to support more computation stages than having them processing data simultaneously in an FPGA.

Lastly, if some aspects of a model cannot be efficiently implemented using an FPGA architecture, it makes more sense to use a hybrid approach. While this could take the form of a conventional CPU-based system connected to an FPGA board, devices such as Intel's Stratix 10 SoCs contain hard quad-core ARM processors, facilitating power-efficient computation for aspects of the simulation that will not benefit from FPGA acceleration.

Acknowledgments

This work is supported in part by the European Union Horizon 2020 research and innovation programme (Grant Numbers 671653 and 675191), by the UK EPSRC (EP/N031768/1, EP/I012036/1, EP/L00058X/1 and EP/K503733/1), by the European Research Council (Grant Number 291406), by the Maxeler University Programme, by the HiPEAC NoE, by Altera, and by Xilinx.

References

[1] J. Shukla, T.N. Palmer, R. Hagedorn, B. Hoskins, J. Kinter, J. Marotzke, M. Miller, J. Slingo, Bull. Am. Meteorol. Soc. 91 (10) (2010) 1407–1412. http://dx.doi.org/10.1175/2010BAMS2900.1.

[2] L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, G. Yang, Field Programmable Logic and Applications, FPL, 2013 23rd International Conference on, 2013, pp. 1–6. http://dx.doi.org/10.1109/FPL.2013.6645508.

[3] X. Lapillonne, O. Fuhrer, Parallel Process. Lett. 24 (01) (2014) 1450003. http://dx.doi.org/10.1142/S0129626414500030.

[4] E.N. Lorenz, J. Atmos. Sci. 20 (2) (1963) 130–141. http://dx.doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[5] E.N. Lorenz, in: T. Palmer, R. Hagedorn (Eds.), Predictability of Weather and Climate, Cambridge University Press, 2006, pp. 40–58. http://dx.doi.org/10.1017/CBO9780511617652.004.

[6] D.S. Wilks, Q. J. R. Meteorol. Soc. 131 (606) (2005) 389–407. http://dx.doi.org/10.1256/qj.04.03.

[7] E. Ott, B.R. Hunt, I. Szunyogh, A.V. Zimin, E.J. Kostelich, M. Corazza, E. Kalnay, D.J. Patil, J.A. Yorke, Tellus A 56 (5) (2004) 415–428. http://dx.doi.org/10.1111/j.1600-0870.2004.00076.x.

[8] I. Fatkullin, E. Vanden-Eijnden, J. Comput. Phys. 200 (2) (2004) 605–638. http://dx.doi.org/10.1016/j.jcp.2004.04.013.

[9] F. Kwasniok, Phil. Trans. R. Soc. A 370 (2012) 1061–1086. http://dx.doi.org/10.1098/rsta.2011.0384.

[10] T.P. Sapsis, A.J. Majda, Phys. D 252 (2013) 34–45. http://dx.doi.org/10.1016/j.physd.2013.02.009.

[11] F.P. Russell, P.D. Düben, X. Niu, W. Luk, T.N. Palmer, Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 171–178. http://dx.doi.org/10.1109/FCCM.2015.52.

[12] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P.Y. Xiao, D. Burger, SIGARCH Comput. Archit. News 42 (3) (2014) 13–24. http://dx.doi.org/10.1145/2678373.2665678.

[13] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, D. Burger, Microarchitecture, MICRO, 2016 49th Annual IEEE/ACM International Symposium on, IEEE Computer Society, 2016, pp. 1–13. http://dx.doi.org/10.1109/MICRO.2016.7783710.

[14] G. Teodoro, T. Kurc, J. Kong, L. Cooper, J. Saltz, 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 1063–1072. http://dx.doi.org/10.1109/IPDPS.2014.111.

[15] B. Betkaoui, D.B. Thomas, W. Luk, 2010 International Conference on Field-Programmable Technology, 2010, pp. 94–101. http://dx.doi.org/10.1109/FPT.2010.5681761.

[16] S. Mittal, J.S. Vetter, ACM Comput. Surv. 47 (2) (2014) 19:1–19:23. http://dx.doi.org/10.1145/2636342.

[17] D. Oriato, S. Tilbury, M. Marrocu, G. Pusceddu, 2012 Symposium on Application Accelerators in High Performance Computing, 2012, pp. 129–132. http://dx.doi.org/10.1109/SAAHPC.2012.8.

[18] L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, X. Huang, Y. Zhang, G. Yang, ACM Trans. Reconfigurable Technol. Syst. 8 (2) (2015) 11:1–11:16. http://dx.doi.org/10.1145/2629581.

[19] P.D. Düben, T.N. Palmer, Mon. Weather Rev. 142 (2014) 3809–3829. http://dx.doi.org/10.1175/MWR-D-14-00110.1.

[20] P.D. Düben, H. McNamara, T.N. Palmer, J. Comput. Phys. 271 (2014) 2–18. http://dx.doi.org/10.1016/j.jcp.2013.10.042.

[21] S. Herrera, J. Fernández, M.A. Rodríguez, J.M. Gutiérrez, Nonlinear Processes Geophys. 17 (4) (2010) 329–337. http://dx.doi.org/10.5194/npg-17-329-2010.

[22] T. Mauritsen, B. Stevens, E. Roeckner, T. Crueger, M. Esch, M. Giorgetta, H. Haak, J. Jungclaus, D. Klocke, D. Matei, U. Mikolajewicz, D. Notz, R. Pincus, H. Schmidt, L. Tomassini, J. Adv. Modeling Earth Syst. 4 (3) (2012). http://dx.doi.org/10.1029/2012MS000154.

[23] H.M. Arnold, I.M. Moroz, T.N. Palmer, Phil. Trans. R. Soc. A 371 (1991) (2013) 20110479. http://dx.doi.org/10.1098/rsta.2011.0479.

[24] S. Boldo, G. Melquiond, IEEE Trans. Comput. 57 (4) (2008) 462–471. http://dx.doi.org/10.1109/TC.2007.70819.

[25] M. Chiu, M.C. Herbordt, ACM Trans. Reconfigurable Technol. Syst. 3 (4) (2010) 23:1–23:37. http://dx.doi.org/10.1145/1862648.1862653.