OCCAM

OXFORD CENTRE FOR COLLABORATIVE APPLIED MATHEMATICS

Minimizing synchronizations in sparse iterative solvers for distributed supercomputers

by

Sheng-Xin Zhu
Tong-Xiang Gu
Xing-Ping Liu

OCCAM Preprint Number 13/28

Minimizing synchronizations in sparse iterative solvers for distributed supercomputers ☆

Sheng-Xin Zhu[a,1], Tong-Xiang Gu[b,2], Xing-Ping Liu[b,2]

[a] *Oxford Centre for Collaborative and Applied Mathematics, Mathematical Institute, 24-29 St Giles', Oxford, OX1 3LB, U.K.*
[b] *Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, P.O. Box, 8009, Beijing 100088, P.R China.*

Abstract

Eliminating synchronizations is one of the important techniques related to minimizing communications for modern high performance computing. This paper discusses principles of reducing communications due to global synchronizations in sparse iterative solvers on distributed supercomputers. We demonstrates how to minimizing global synchronizations by rescheduling a typical Krylov subspace method. The benefit of minimizing synchronizations is shown in theoretical analysis and is verified by numerical experiments using up to 900 processors. The experiments also show the communication complexity for some structured sparse matrix vector multiplications and global communications in the underlying supercomputers are in the order $P^{1/2.5}$ and $P^{4/5}$ respectively, where P is the number of processors and the experiments were carried on a Dawning 5000A.

Keywords: Minimizing communications, parallel Krylov subspace methods, high performance computing, distributed supercomputers.
2010 MSC: , 65F10, 68W10, 68W15

1. Introduction

Minimizing communications in all kinds of algorithms for multi-core computing platforms is drawing increasing attention. On shared memory computers, reducing computations, exploring structured parallelism and increasing cache hit rate (or data reuse rate) are always given to priorities, while on distributed supercomputers, priority is often given to minimizing all kinds of communications [3]. In this paper, *communication* is referred to as the process of exchanging data between different computing nodes or processors via Message Passing Interface (MPI). Other efficient data movements in local shared memory parts without using MPI, which often require efficient data structure to increase the cache hit rate, go beyond our discussion. Communication time consists of two parts in this paper: the first part is *set uptime*, which is used to prepare available computing resources (computing nodes or processors) or to wait necessary information, it mainly depends on the *latency* of the underlying computer system and the number of *synchronizations*; whereas the second part depends on the ratio of the data volume to the bandwidth of the communication systems, or the so called *volume-to-surface ratio*. Reducing communications should take into account at least of one part of them. This paper focus on improving the performance of Krylov subspace methods on distributed supercomputers mainly by reducing the set up time via eliminating synchronizations.

The fundamental reasons for us to concentrate on reducing the set up time lies in that synchronizations are often unavoidable when computing reliable inner products on a distributed supercomputer, and the latency of an underlying distributed supercomputer is the hardest to be improved. Recent technology has shown that latency almost cannot be improved any more, with a tinny improvement, 5.5%/year, whereas bandwidth increases 26%/year and the speed of

Email addresses: shengxin.zhu@maths.ox.ac.uk (Sheng-Xin Zhu), txgu@iapcm.ac.cn (Tong-Xiang Gu), lxp@iapcm.ac.cn (Xing-Ping Liu)

[1] The authors' research is partly supported by Award no KUK-C1-013-04, made by King Abdullah University of Science of Technology.

[2] The authors' research is partly supported by the NSF of China (No. 61170309, 91130024 and 60973151) and the key project of scientific and technical development of China Academy of Engineering Physics (2012A0202008 and 2011A0202012).

floating-point operations increases 59%/year[19, p.109][40]. And second there is already detailed consideration of minimizing communications by analysing the volume-to-surface ratio for kernels in Krylov iterative methods [2, 5, 18, 29, 30, 37, 38].

The remaining of this paper is organized as follows. Section § 2 first briefly gives the communication complexity of the computational kernels of Krylov subspace methods for some structured sparse matrices. Section § 3 discusses principles to reduce communications. A detailed case study is presented in section § 4, demonstrating how to reconstruct Krylov subspace methods. Section § 5 analyse the performance of the reconstructed methods and the corresponding serial versions. And finally we present some numerical results to verify our analysis and give some conclusions.

2. Communication Complexity

In this paper, P is the number of processors, N is the number of unknowns in an underlying linear system, t_{fl} is the time for one floating point operation and t_s is the latency of an underlying system, and t_w is the volume-to-surface ratio, one unit volume message over the bandwidth of the underlying system. We approximate the total time for on iteration in a Krylov subspace methods as follow

$$T = \underbrace{\phi(N/P)t_{fl}}_{\text{computation time}} + \underbrace{\psi(t_s, t_w)\omega(P)}_{\text{global communication}} + \underbrace{\mu(t_s, t_w, P)}_{\text{local communication}} . \tag{1}$$

Where $\phi(N/P)t_{fl}$ is the computation time for the three basic computational kernels in classical Krylov subspace methods, namely, vector updates, matrix-vector multiplications and inner product computations. Table 1 shows their computation and communication complexity. Vector updates are parallel in nature and there is no need for communication. For general sparse matrices, efficient matrix-vector multiplications need sophisticated techniques [46], whereas for some structured sparse matrices, sparse matrix-vector multiplications usually only need *local communication*—only exchanging data with its neighbours, especially the data structure of the underlying sparse matrices is well-organized. Some time this communication can be overlapped (partially or entire) by other computation times. We denote this part of time as $\mu(t_s, t_w, P)$. Inner product computation needs *global communications*, we denote the complexity as $O(\omega(P))$. There are several theoretical models to describe the possible complexity order $O(\omega(P))$, see [1, 32, 44] for example. The terms $\mu(t_s, t_w, P)$ and $O(\omega(P))$ are left vague at the moment, and will be fitted and verified by real numerical results. The basic assumption is $\omega(P) \gg \mu(t_s, t_w, P)$ for large P.

Table 1 gives the computation and communications time of the three kernels needed on a distributed computer for structured sparse matrices.

Table 1: Communication Complexity of Kernels of Krylov Methods for Structured Sparse Matrices

Name	Operations	Computation time	Communication time
matrix-vector multiplication	$y \leftarrow Ax$	$n_z N t_{fl}/P$	$\mu(t_s, t_w)$
one inner product	$dot \leftarrow x^T y$	$2Nt_{fl}/P$	$2(t_s + t_w)\omega(P)$
k inner products	—	$2kNt_{fl}/P$	$2(t_s + kt_w)\omega(P)$
vector update	$y \leftarrow ax + y$	$2Nt_{fl}/P$	—

N: the number of unknowns; P: the number of processors; n_z: the average number of non-zeros per row of A; t_{fl}: the time per float point operation, t_s: the set up time, t_w: the time of passing per volume (unite) of message. Usually, $t_s \gg t_w$.

3. Strategies for Reducing Communications

It should be mentioned that communication hinders the scalability of iterative Krylov subspace methods has been noticed since 1980s. Pioneer researchers have proposed some original ideas to reduce communications [8, 14, 15, 16, 34, 35, 36]. However, due to the constraint on the memory size at that time, most of these pioneering work

more focused on maximizing floating-point operations per unit memory. With the soar of development of hardware, more and more memory and processors are available, and the gaps between the floating point operations, bandwidth and latency increase dramatically. The importance of reducing communications is getting more public aware. As discussed above, both matrix-vector multiplications and inner product computation need communications. Avoiding communications requires us to reschedule these two computational kernels: minimizing communications in matrix vector multiplications [2, 18, 29, 30]; and minimizing global communications due to the inner product computations [4, 22, 23, 31, 36, 47, 48, 49, 53]. We focus on the late ones for the reasons discussed above.

Three strategies have been considered to minimize communications. First, communications should be overlapped with computations as many as possible, this step is easy to put into practice. Second, perhaps the most attractive one, is to remove inner production computations which need global communications, developing inner product free iterative solvers. For some symmetric linear systems, there are already some results. For example, the inner products in the conjugate gradient(CG) method can be replaced by solving a small linear system as described in the s-step methods [8, 9, 10, 11, 12] and multiple search direction conjugate gradient method [24, 25]. The second remedy is to schedule more inner products to be computed at one synchronization point. In this way, the number of inner products usually increases but more inner products can be at less synchronization points, and thus the set up time (t_s) is reduced. This strategies is valid for all kinds of Krylov methods, and furthermore, such a trick insulate algorithms from the computing platform, which makes the algorithms more portable.

The number of synchronization points ultimately depends on the data dependence in an underlying algorithm. The pattern of data dependence various from algorithm to algorithm, and thus each algorithm has to be consider individually. Luckily, the following strategies can be used as a framework to reduce and breakdown some data dependence in an Krylov subspace methods, namely, *residual split*,*loop shift* and *transpose shift*.

4. Case Study

4.1. Original Algorithm

Consider the GPBiCG(m, ℓ) method [21] (see Algorithm 1): a hybrid generalized product-type methods based on Bi-CG. With a proper choice of the parameters m and ℓ, it can degenerate into 3 other well-known Krylov subspace methods. The GPBiCG$(1, 0)$ is reduce to the well-known BiCGSTAB method [45]; GPBiCG$(1, 1)$ is equivalent to the hybrid BiCGSTAB2 method [26], which is supposed to reduce the "un-luck" breakdown by changing searching direction every other iterate; GPBiCG$(0, 1)$ is identical to the GPBiCG method, which is supposed to be more robust by increasing the search direction space [51], in which, the residual vector r_k in line 16 of Algorithm 1 and the approximation solution x_k in line 19 of Algorithm 1 is updated by three vectors other than two vectors in most Krylov subspace methods. The idea of BiCGSTAB2 and GPBiCG indicates that it is promising to develop breakdown-free Krylov subspace methods by increasing the searching space (for each search) and dynamically changing search directions, which results in the GPBiCG(m, ℓ) method. This method is an typical Krylov subspace method and thus it is an ideal candidate to investigate the performance of its corresponding parallel version.

4.2. Data Dependence Analysis and Algorithm Redesign

There are three synchronization points in Algorithm 1. Global communication are required in line 4, line 8 or line 13, and line 18 in this algorithm. Further, ζ_k depends on s_k and t_k which themselves depends on α_k. β_k in line 18 depends on the residual r_{k+1} which itself depends η_k and ζ_k. Reducing the number of synchronization points need to eliminate the data dependence at first. We also observe that the vector update in line 6 has no relationship with s_k in line 5, and the communication time for the matrix vector multiplication can be overlapped by the vector update in line 6. In this scenario counting the whole term $\mu(t_s, t_w, P)$ in (1) is not appropriate.

4.2.1. Residual Split

The first commonly used trick to breakdown data dependence is *residual split*: the residual r_{k+1} in line 18 of Algorithm 2 can be replaced by the formulae in line 16 2. Therefore the inner product (r_0^*, r_{k+1}) can be computed indirectly by

$$\widetilde{rr}_{k+1} := (r_0^*, r_{k+1}) = (r_0^*, t_k) - \eta_k(r_0^*, y_k) - \zeta_k(r_0^*, s_k). \tag{2}$$

In this way, two more inner products have to be calculated, but these additional inner products can be computed in last synchronization point and thus one synchronization point is reduced.

Algorithm 1 GPBiCG(m, ℓ)

$r_0 = b - Ax_0$, $t_{-1} = w_{-1} = \vec{0}$, choose arbitrary r_0^*, s.t. $(r_0^*, r_0) \neq 0$.
for $k = 0, 1, ...,$ until $\|r_k\| \leq tol$ **do**
  $p_k = r_k + \beta_{k-1}(p_{k-1} - u_{k-1})$; $q_k = Ap_k$
  $\alpha_k = \boxed{\dfrac{(r_0^*, r_k)}{(r_0^*, q_k)}}$
  $t_k = r_k - \alpha_k q_k$; $s_k = At_k$
  $y_k = t_{k-1} - t_k - \alpha_k w_{k-1}$
  if (mod $(k,\ m+\ell) < m$ or $k = 0$) **then**
    $\zeta_k = \boxed{\dfrac{(s_k, t_k)}{(s_k, s_k)}}$
    $u_k = \zeta_k q_k$
    $z_k = \zeta_k r_k - \alpha_k u_k$
    $r_{k+1} = t_k - \zeta_k s_k$
  else
    $\eta_k = \boxed{\dfrac{(s_k, s_k)(y_k, t_k) - (y_k, s_k)(s_k, t_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}}$
    $\zeta_k = \boxed{\dfrac{(y_k, y_k)(s_k, t_k) - (y_k, t_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}}$
    $u_k = \zeta_k q_k + \eta_k (t_{k-1} - r_k + \beta_{k-1} u_{k-1})$
    $z_k = \zeta_k r_k + \eta_k z_{k-1} - \alpha_k u_k$
    $r_{k+1} = t_k - \eta_k y_k - \zeta_k s_k$
  end if
  $\beta_k = \frac{\alpha_k}{\zeta_k} \boxed{\dfrac{(r_0^*, r_{k+1})}{(r_0^*, r_k)}}$
  $x_{k+1} = x_k + \alpha_k p_k + z_k$
  $w_k = s_k + \beta_k q_k$
end for

Algorithm 2 Parallel GPBiCG(m, ℓ)

$r_0 = b - Ax_0$, $t_{-1} = w_{-1} = \vec{0}$, $p_0 = r_0$, $q_0 = Aq_0$, choose arbitrary r_0^*, s.t. $(r_0^*, r_0) \neq 0$, $f = A^T r_0^*$, $\alpha = (r_0^*, r_0)/(f, p_0)$.
for $k = 0, 1, ...,$ until convergence **do**
  $tem = t_{k-1}$
  $t_k = r_k - \alpha_k q_k$;
  $s_k = At_k$
  $y_k = tem - t_k - \alpha_k w_{k-1}$
  if mod$(k,\ m+\ell) < m$ or $k = 0$ **then**
    compute all the inner products:
    $\boxed{\begin{array}{l}(s_k, t_k), (s_k, s_k), (r_0, t_k), (r_0, s_k),\\ (f, s_k), (f, t_k), (f, q_k), (f, p_k)\end{array}}$
    $\zeta_k = \frac{(s_k, t_k)}{(s_k, s_k)}$, $\eta_k = 0$
    $\widetilde{rr}_{k+1} = (r_0^*, t_k) - \zeta_k (r_0^*, s_k)$
    $\widetilde{fu}_k = \zeta_k (f, q_k)$
    $\widetilde{fr}_{k+1} = (f, t_k) - \zeta_k (f, s_k)$
    $u_k = \zeta_k q_k$
    $z_k = \zeta_k r_k - \alpha_k u_k$
    $r_{k+1} = t_k - \zeta_k s_k$
  else
    $h_k = tem - r_k + \beta_{k-1} u_{k-1}$
    compute all the inner products:
    $\boxed{\begin{array}{l}(s_k, s_k), (s_k, y_k), (s_k, t_k), (y_k, t_k), (y_k, y_k),\\ (r_0^*, t_k), (r_0^*, y_k), (r_0^*, s_k), (r_0^*, r_k), (f, t_k),\\ (f, y_k), (f, s_k), (f, h_k), (f, q_k), (f, p_k)\end{array}}$
    $\eta_k = \dfrac{(s_k, s_k)(y_k, t_k) - (y_k, s_k)(s_k, t_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$,
    $\zeta_k = \dfrac{(y_k, y_k)(s_k, t_k) - (y_k, t_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
    $\widetilde{rr}_{k+1} = (r_0^*, t_k) - \eta_k (r_0^*, y_k) - \zeta_k (r_0^*, s_k)$
    $\widetilde{fu}_k = \zeta_k (f, q_k) + \eta_k (f, h_k)$
    $\widetilde{fr}_{k+1} = (f, t_k) - \eta_k (f, y_k) - \zeta_k (f, s_k)$
    $u_k = \zeta_k q_k + \eta_k h_k$
    $z_k = \zeta_k r_k + \eta_k z_{k-1} - \alpha_k u_k$
    $r_{k+1} = t_k - \eta_k y_k - \zeta_k s_k$
  end if
  $\beta_k = \frac{\alpha_k}{\zeta_k} \frac{\widetilde{rr}_{k+1}}{(r_0^*, r_k)}$
  $x_{k+1} = x_k + \alpha_k p_k + z_k$
  $w_k = s_k + \beta_k q_k$
  $p_{k+1} = r_{k+1} + \beta_k (p_k - u_k)$
  $q_{k+1} = Ap_{k+1}$
  $\widetilde{fp}_{k+1} = \widetilde{fr}_{k+1} + \beta_k ((f, p_k) - \widetilde{fu}_k)$
  $\alpha_{k+1} = \frac{\widetilde{rr}_{k+1}}{\widetilde{fp}_{k+1}}$
end for

Table 2: The main computations of GPBiCG(m, ℓ) and PGPBiCG(m, ℓ)

Methods	vector update				inner product				matrix-vector	synchronization points
	H	**if**	**else**	T	H	**if**	**else**	T		
GPBiCG(m, ℓ)	5	3	7	3	1	2	5	2	2	3
PGPBiCG(m, ℓ)	5	3	7	3	0	9	15	0	2	1

The **for** loops in Algorithm 1 and Algorithm 2 are divided into 4 parts: the part before **if** and and after **else** denoted as the heard(H) part and the tail(T) respectively.

4.2.2. Loop Shift and Transpose Shift

Another trick to further breakdown data dependence and reduce synchronization points is referred as *loop shift*. Suppose every loop has 3 statements, say, A, B, C. Then one possible loop pattern is $\{A, B, C\}, \{A, B, C\}, \ldots$. This loop can be shifted to another one, $A, \{B, C, A\}, \{B, C, A\}, \ldots$. According to such a loop shift, the loop in Algorithm 1 from line 2 to line 20 can be arranged as starting with line 5 to line 20 and put line 3 and line 4 at the back of the look with corresponding index changes, in this way, α_{k+1} is considered instead of α_k for each loop. And the synchronization point to compute α_{k+1} is also reduced. See Algorithm 2.

It is noted that computing α_{k+1} needs (r_0^*, q_{k+1}); the update of q_{k+1} depends on a matrix-vector multiplication, $q_{k+1} = Ap_{k+1}$. If p_{k+1} is updated by the residual split technique directly, it will bring two unexpected matrix-vector multiplications. Alternatively, one can write the inner product as

$$(r_0^*, q_{k+1}) = (r_0^*, Ap_{k+1}) = (A^T r_0^*, p_{k+1}) = (f, p_{k+1}), \tag{3}$$

where $f = A^T r_0^*$. Then the inner product (f, p_{k+1}) can be computed indirectly by relatively cheap vector updates and residual placement techniques. Form $p_{k+1} = r_{k+1} + \beta_k(p_k - u_k)$, we get

$$\widetilde{fp}_{k+1} := (f, p_{k+1}) = (f, r_{k+1}) + \beta_k \left((f, p_k) - (f, u_k)\right). \tag{4}$$

The inner products (f, r_{k+1}) and (f, u_k) can be calculated by the residual split tricks.

$$\widetilde{fr}_{k+1} := (f, r_{k+1}) = (f, t_k) - \eta_k (f, y_k) - \zeta_k (f, s_k). \tag{5}$$

Let $h_k = t_{k-1} - r_k + \beta_{k-1} u_{k-1}$ (line 14 in Algorithm 1), then (f, u_k) can be formulated as

$$\widetilde{fu}_k := (f, u_k) = \begin{cases} \zeta_k (f, q_k) & \text{if execute line 9,} \\ \zeta_k (f, q_k) + \eta_k (f, h_k) & \text{if execute line 14.} \end{cases} \tag{6}$$

Equation(4)-(6) show that (f, p_{k+1}) can be computed via (f, t_k), (f, y_k), (f, s_k),(f, q_k) and (f, h_k). These inner products can be calculated when computing inner products associated with ζ_k and η_k in line 8 or line 13 in Algorithm 1. The 3 global synchronization points in Algorithm 1 is reduced to only one in Algorithm 2

We call such a trick in (3) *transpose shift*, which can reduce synchronization points and avoid to increase unnecessary matrix vector multiplications. It should be pointed out that the transpose of matrix should better be avoided in distributed computers, because large data movement is time consuming. Luckily, the transpose only used once.

5. Performance Analysis

The numbers of computational kernels in Algorithm 1 and Algorithm 2 are listed in Table 2. This section compares the performance of the two algorithms according to the number of computational kernels.

Table 3: The time of inner products computation in GPBiCG(m, ℓ) and PGPBiCG(m, ℓ)

Methods	Position	No.	Time
GPBiCG(m, ℓ)	H	1	$2t_{fl}N/P + (t_s + t_w)\,\omega(P)$
	if	2	$4t_{fl}N/P + (t_s + 2t_w)\,\omega(P)$
	else	5	$10t_{fl}N/P + (t_s + 5t_w)\,\omega(P)$
	T	2	$4t_{fl}N/P + (t_s + 2t_w)\,\omega(P)$
PGPBiCG(m, ℓ)	**if**	9	$18t_{fl}N/P + (t_s + 9t_w)\,\omega(P)$
	else	15	$30t_{fl}N/P + (t_s + 15t_w)\,\omega(P)$

5.1. Time Complexity Analysis

As shown in Table 2 the re-designed algorithm requires 4 to 7 more inner products in average but all the inner products need only one global synchronization point rather than three. According to Table 1, it takes

$$t_{inn(k)} = \frac{2kNt_{fl}}{P} + (t_s + kt_w)\,\omega(P) \tag{7}$$

time to compute k inner products at one synchronization point. Since $t_s \gg t_w$, thus reducing the number of global synchronization points can reduce the global communication time. Table 3 reports the time needed for each synchronization point in the two Algorithm in details. Because the number of vector updates and the inner products in **if** branch and **else** branch are different, it is difficult to evaluate the expect solution time for such an algorithm based on only one iteration. Furthermore, the communication time of inner product doesn't linearly depend on the number of inner products, therefore we can not use the average number of $m + \ell$ iterations. Here we suppose m and ℓ are fixed integer and regard $m + \ell$ iterations as a "unit loop". Counting the number of kernels of Krylov subspace methods in Algorithm 1, and substituting the time for each kernels into equation 1, we get a the expected time for a "unit loop"

$$T^{\mathrm{GPBiCG}(m,\ell)} = \{8(m+\ell) + 3m + 7\ell\}t_{vec} + 2(m+\ell)t_{mv} + (m+\ell)(t_{inn(1)} + t_{inn(2)}) + mt_{inn(2)} + \ell t_{inn(5)}, \tag{8}$$

where t_{vec} and t_{mv} is the time for vector update and matrix-vector multiplication given in Table 1, $t_{inn(k)}$ is given in equation (7). The formulae in equation (8) can be simplified as

$$T^{\mathrm{GPBiCG}(m,\ell)} = \frac{\lambda_1}{P} + \lambda_2\omega(P) + 2\,(m+\ell)\,\mu(t_s, t_w, P), \tag{9}$$

where μ is the local communication time defined in Table 1 and

$$\lambda_1 = \{32m + 46\ell + 2\,(m+\ell)\,n_z\}\,Nt_{fl}, \tag{10}$$

$$\lambda_2 = 3\,(m+\ell)\,t_s + (5m + 8\ell)\,t_w. \tag{11}$$

Similarly, the time for one "unit loop" in PGPBiCG(m, ℓ) is

$$T^{\mathrm{PGPbiCG}(m,\ell)} = \frac{\sigma_1}{P} + \sigma_2\omega(P) + 2\,(m+\ell)\,\mu(t_s, t_w, P), \tag{12}$$

where

$$\sigma_1 = \{40m + 60\ell + 2\,(m+\ell)\,n_z\}\,Nt_{fl}, \tag{13}$$

$$\sigma_2 = (m+\ell)\,t_s + (9m + 15\ell)\,t_w. \tag{14}$$

When solving an identical problem using the same number of processors, the improvement in solution time of

PGPBiCG(m, ℓ) compared with that of GPBiCG(m, ℓ) methods is

$$\eta = \frac{T^{\text{GPBiCG}(m,\ell)} - T^{\text{PGPBiCG}(m,\ell)}}{T^{\text{GPBiCG}(m,\ell)}} = \frac{(\lambda_2 - \sigma_2)\, P\omega(P) + \lambda_1 - \sigma_1}{\lambda_2 P\omega(P) + 2\,(m+\ell)\,\mu P + \lambda_1} \tag{15}$$

$$= \frac{\{2\,(m+\ell)\,t_s - (4m + 6\ell)\,t_w\}\, P\omega(P) - (8m + 14\ell)\, N t_{fl}}{\{3\,(m+\ell)\,t_s + (5m + 8\ell)\,t_w\}\, P\omega(P) + 2\,(m+\ell)\,\mu P + \lambda_1}. \tag{16}$$

Because $t_s \gg t_w$, when P is large enough, $\eta \to \frac{2}{3}$.

Conclution 1. *When solving the same problem using the same number of processors, the asymptotic improvement in solution time using PGPBiCG(m, ℓ) compared with using GPBiCG(m, ℓ) is about two thirds.*

5.2. Convergence Analysis

Stability is another concern for parallel algorithms. It has been reported that mathematically equivalent algorithms can have different convergent behaviour. It has been observed that the residual split technique can hinder the convergence rate of the rescheduled algorithms [27, 43]. One possible reason is that the indirectly computing inner product can accumulate the error due to round off. Indirect computing an inner product via several other inner products brings new error sources, which makes the indirectly computed inner product less accurate. Therefore the convergence performance of the re-scheduled algorithm can be poorer compared with the corresponding serial version.

In Algorithm 2, notice that $t_k = r_k - \alpha_k q_k$ in line 4, then the inner product $\widetilde{fr}_{k+1}$ in line 12 and line 22 and inner product $\widetilde{fp}_{k+1}$ in line 32 can be computed in a recursive way:

$$\widetilde{fr}_{k+1} = \widetilde{fr}_k - a_k(f, q_k) - \eta_k(f, y_k) - \zeta_k(f, s_k), \tag{17}$$

$$\widetilde{fp}_{k+1} = \widetilde{fr}_{k+1} + \beta_k \left(\widetilde{fp}_k - (f, u_k)\right). \tag{18}$$

In such a recursive way, if the inner product $\left(f, r_0^*\right) = (f, p_0)$ is provided, then the inner products $\widetilde{fr}_{k+1}$ and $\widetilde{fp}_{k+1}$ can be computed recursively. Two less inner products are not computed in line 8 and line 18. This is the way used in the improved BiCGSTAB [48]. While in Algorithm 2 the inner products (f, r_k) and (r_0, r_k) are updated directly, which reduces the potential error sources due to indirect computations. The convergence performance of the two ways to compute inner product is compared in Figure 2(a). A little further analysis on the accuracy of these two ways to compute the inner product shows that the recursive way to compute the inner product can accumulate the round off error. See Appendix C for details.

6. Numerical Verification

Consider the following test problem

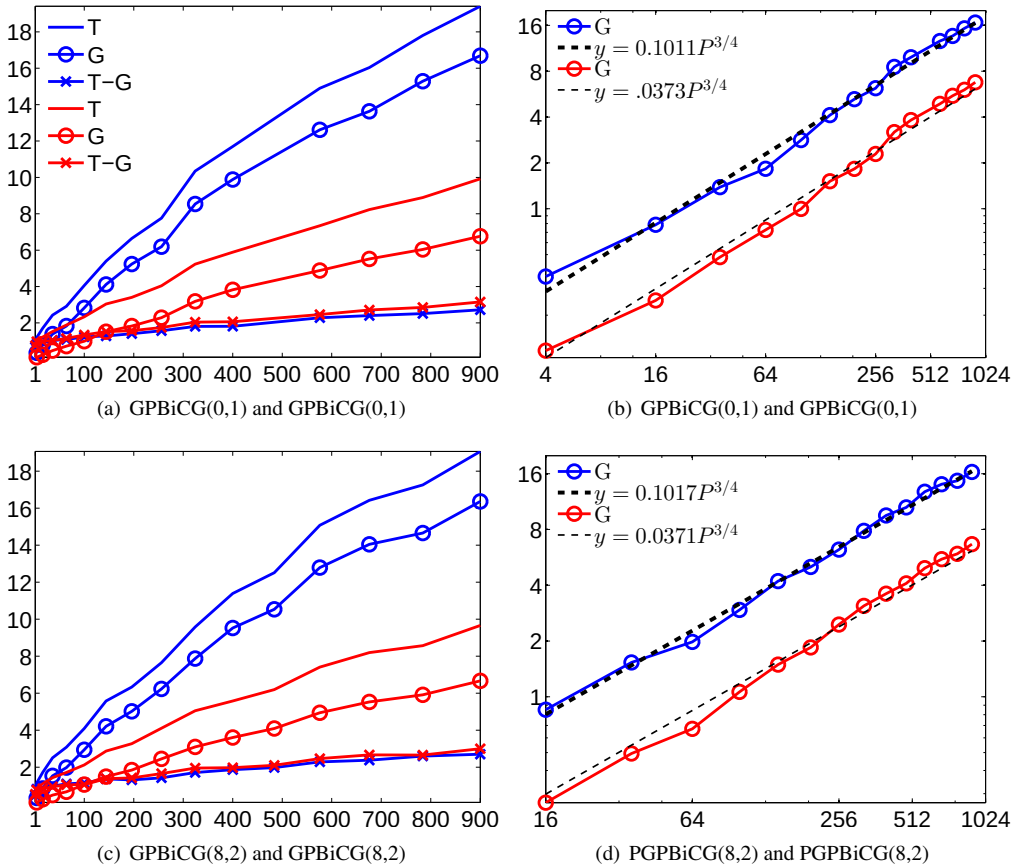
$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + c\frac{\partial u}{\partial x} + d\frac{\partial u}{\partial y} + eu = 0, \quad x, y \in (0, 1), \tag{19}$$

with boundary conditions

$$\begin{cases} \frac{\partial u}{\partial x}|_{x=0} = 0, & \frac{\partial u}{\partial x}|_{x=1} = 0 \\ u|_{y=0} = 0, & u|_{y=1} = 10, \end{cases} \tag{20}$$

where $a = b = 1512.0$, $c = d = 1.0$, $e = 0.0$. The parameters are chosen in such a way that a Krylov subspace method is convergent without preconditioning. A special nine-point difference scheme is used to approximate the test problem; the average non-zero elements of the matrix is 9, $n_z = 9$. The numerical experiments of GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) run on a supercomputer: Dawning 5000A. The experiments are implemented on a modified early version of the software package AZTEC (http://www.cs.sandia.gov/CRF/aztec1.html) for massive linear systems.

The solution time and the communication time for 3000 iterations are collected for each method. Each result is an average of 4 times running results. The problem size on each processor, $\frac{N}{P}$, is fixed as $3,600$.



(a) GPBiCG(0,1) and GPBiCG(0,1)

(b) GPBiCG(0,1) and GPBiCG(0,1)

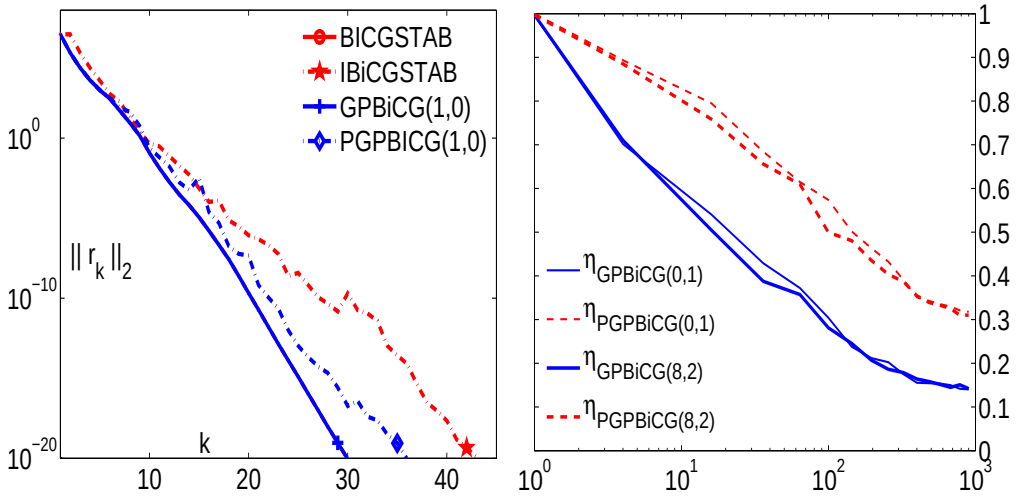
(c) GPBiCG(8,2) and GPBiCG(8,2)

(d) PGPBiCG(8,2) and PGPBiCG(8,2)

Figure 1: Comparison on the performance of GPBiCG(m, ℓ) method (blue curves) and PGPBiCG(m, ℓ) method (red curves). In (a) and (c), the solid lines represent the total solution time (T), the solid lines marked with circles represent the global communication time (G), and the solid solid lines marked with '×' represent the time excluding global communication time. (b) and (d) indicates that the global communication time sub-linearly depends on P, and $\omega(P) \sim P^{3/4}$, and the global communication time of GPBiCG(m, ℓ) is about 2.7 times of that of PGPBiCG(m, ℓ). The factor 2.7 reasonable, because there are 3 global synchronizations in the GPBiCG(m, ℓ) method, whereas there is only one synchronizationbut more inner products in PGPBiCG(m, ℓ) .

Figure 1 illustrates how the solution time for GPBiCG(m, ℓ) and PGPBiCG(m, ℓ) increases with the number of processors increasing. It is observed that the global communication complexity is sub-linear, $\omega(P) \sim P^{3/4}$, but not as good as $\log(P)$. Figure 2(a) compares the convergence behaviour of four mathematical equivalent algorithms, which indicated the recursive way (used in IBiCGSTAB) to compute the compute the inner product converges slow, while breaking down the recurrences (PGPBiCG(1,0)) can improve the performance. Figure 2(b) shows the relationship between the the computing efficiency, η, and the number of processors, where the computing efficiency is defined as the percentage of the time exclude global communications over the total time.

We also compare the global communication time and the matrix vector multiplication time (including necessary communication time). The way that we record these two kinds of time are listed in Appendix A and Appendix B. Figure 3 presents four group results which records the total solution time, the global communication times due to synchronizations, the times for matrix vector multiplications(including local communication.) The results indicates that that when the number of processor is large, the global communication time will outperform the matrix vector multiplication times plus the local communication time. In particular, for the improved algorithm, the matrix vector multiplication takes more times when using processor less than 100, as the number of process increasing, the global



(a) Convergence (b) Computing efficiency.

Figure 2: (a)The convergence performance of four mathematical equivalent algorithms: GPBiCG(1,0), PGPBiCG(1,0), BiCGSTAB and IBiCGSTAB. The result is computed with 256 processors for the test problem with 921,600 unknowns. The $x-$axis is the iteration step, the $y-$axis is 2 norm of the residual. (b) The computing efficiency with the number of processors increasing: $\eta = \frac{\text{Total time}-\text{Global communication time}}{\text{Total time}} \times 100\%$.

communication time will take a dominate share. We should point out, if there is no communications, for our case—keeping N/P fixed and n_z fixed, the computation time for matrix vector multiplication should be an constant. We observed that the matrix vector multiplication time including communication time increases in the rate $P^{1/2.5}$ as P increase. The global communication time increases in the order $P^{4/5}$.

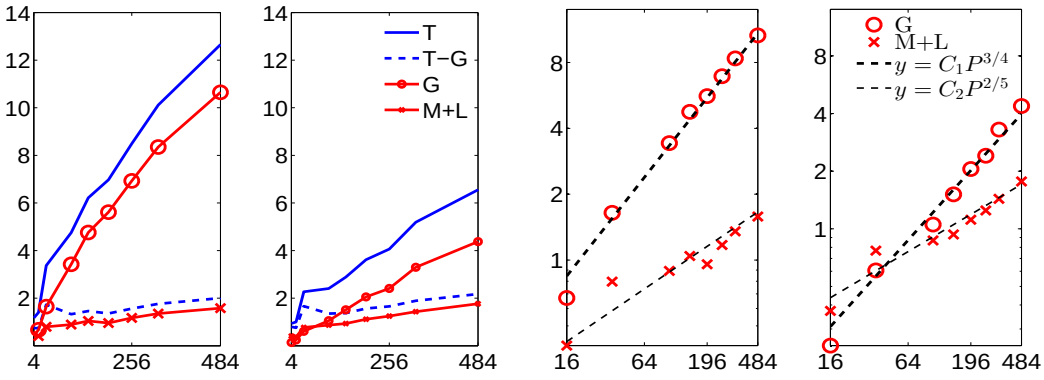
So far, the term $\omega(P)$ and $\mu(t_s, t_w, P)$ become clear. There is a very reasonable explanation for the local communication factor $P^{2/5}$. The latency t_s is believed have a positive correlation the furthest distance of two processors which need data exchange. And such distance depends on the architecture of the supercomputer, especially how the processors are arranged. If P processors are arranged in square lattices, then the furthest distance is $\sqrt{P} = P^{1/2}$, while if the P processors are arranged in a cube lattice, then the furthest distance is $\sqrt{3(P^{1/3})^2} = \sqrt{3}P^{1/3}$. The factor $P^{2/5} = P^{1/2.5}$ indicates that the processors are arranged in an cube-like lattice, but not regular ones, one can think the dimensions of the cube is 2.5.

7. Conclusions

The paper demonstrates that the way of inner product computation can significantly impact the performance of Krylov subspace methods on distribute supercomputers. According to communication complexity and the trends of current technology discussed above, the scalability of an underlying Krylov algorithm largely depends on how many the global communications there are. Consider the following formulae

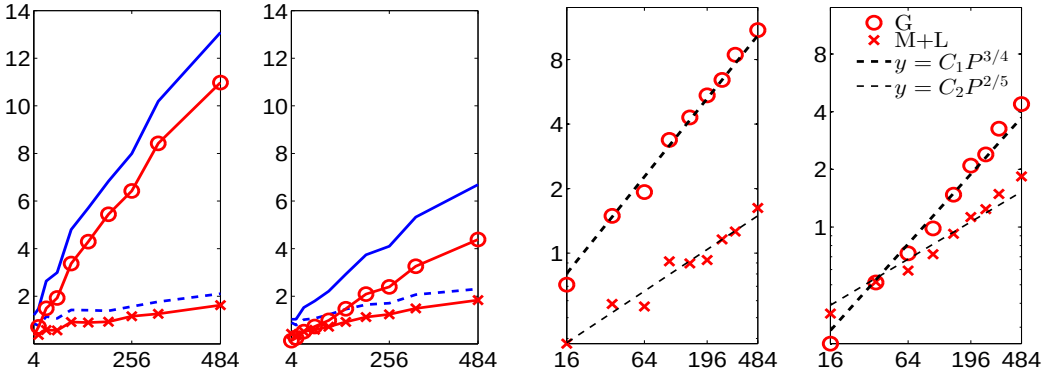
$$T = \varphi\left(\frac{N}{P}\right) t_{fl} + \phi\left(t_s, t_w\right) \log_2 P + \mu\left(t_s, t_w, P\right) \tag{21}$$

where the first item stands for the computation time, the second terms represent the global communication time, and the local communication time. It should be pointed out that, the term $\mu(t_s, t_w, P)$ is including the matrix vector multiplication time in this paper, and the term can be overlapped by other computations. It may not be strict to use the formulae 21. The paper also shows that the term for global communications can be fur large than the term $\mu(t_s, t_w, P)$. Rescheduling the algorithm can reduce global communication time by a factor of about 2.7. Overall, one can obtain a speed up around 2. One may argue this is not attractive. Instead, this is much promising than a 10 times speed up



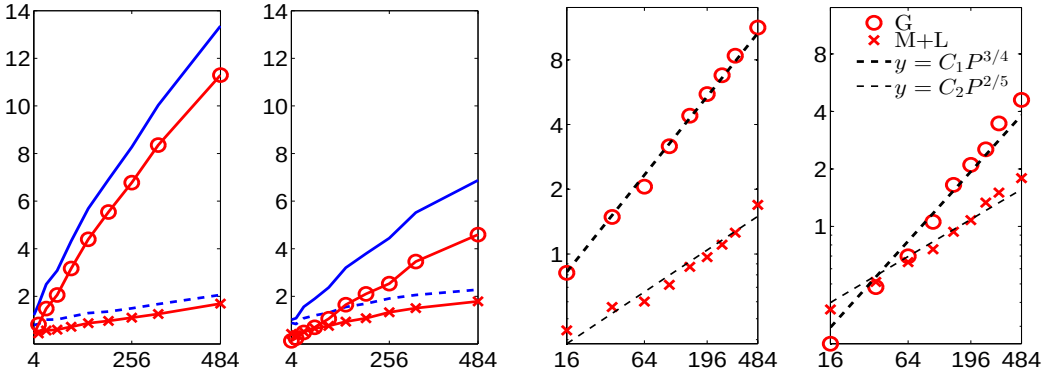
(a) GPBiCG(1,0) and PGPBiCG(1,0)

(b) GPBiCG(1,0) and PGPBiCG(1,0)



(c) GPBiCG(1,1) and PGPBiCG(1,1)

(d) GPBiCG(1,1) and PGPBiCG(1,1)



(e) GPBiCG(2,8) and PGPBiCG(2,8)

(f) GPBiCG(2,8) and PGPBiCG(2,8)

Figure 3: Global communication time and and local communication time. In each window, the left half side is the data of GPBiCG(m, ℓ) methods and the right half side is the data for PGPBiCG(m, ℓ). The left windows (a),(c) and (e) describe the total times (solid blue), the global communication times(solid red with circles), the time excluding global communication time (dash blue) and the matrix vector multiplication and local communication time (dash red with '$\times$'). The right windows (b),(d) and (f) indicate the global communication times increases in the order $P^{3/4}$ while the matrix vector multiplication and local communication time increase in the order $P^{2/5}$.

in matrix vector multiplications, because Figure 1 indicate that the matrix vector multiplications can only take a very small share of the total time when P is large, say 20%, a huge improvement of matrix vector multiplications, (say 100 speed up) at most reduce 20% simulation time; while reducing only 50% global communication time can reduce 40% total time(suppose the global communication consists 80% of the total time).

The term $\omega(P)$ depends on the MPI reduce algorithm and the latency of the the computer system which is very difficult to improve. The strategies proposed here insulate the algorithm from the underlying computer architecture, which is easy to put into practice and potable.

Appendix A. C Code to get global communication time

```
 \\ Global communication time.
dot_vec[0]  = ddot_(&N, y,     &one, y, &one); //yy
 ........
dot_vec[14] = ddot_(&N, r,     &one, r, &one); //rr
time1=AZ_second();     \\ timing
\\ MPI reduce operation, global communications
AZ_gdot_vec(15,dot_vec, dot_vec1,proc_config);
time2=AZ_second();      \\timing
status[AZ_global_comm]=status[AZ_global_comm]+time2-time1;
```

Appendix B. C code to get local communication time

```
\\ matrix vector mutiplicaion and local commonication time
time3=AZ_second();  \\ timing
 Amat->matvec(temppt, s, Amat, proc_config); //s=A*t or s=A*M^{-1}*t
time4=AZ_second();  \\ timing
status[AZ_local_comm]=status[AZ_local_comm]+time4-time3;
```

Appendix C

The indirect inner product computation can result in poor convergence; the stability can be improved by avoiding the recursive computation of inner products.

Consider the following estimate of the error due to round off when computing a inner product.

Lemma 2 ([28, p.75]). *If $x,\ y \in \mathbb{R}^n$, let $fl\left(x^T y\right)$ is the inner product computed by float points computations, $x^T y$ is the true value, $\delta = fl\left(x^T y\right) - x^T y$, then*

$$|\delta| \leq 1.01N\boldsymbol{u} \sum_{i=1}^{N} |x_i y_i|$$

where $\boldsymbol{u}$ is the machine precision.

Lemma 2 shows that $|\delta|$ can be far larger than the machine precision $\boldsymbol{u}$ when N is extremely large. With modern techniques, the accuracy of a inner product of two large vectors is usually much better than what the Lemma gives, but still it it is hard to get the exact value.

For simplicity, we only consider the GPBiCG$(1,0)$ method which equivalent to BiCGSTAB method. The inner product (r_0, r_{k+1}) are compute in PGPBiCG$(1,0)$ and IBiCGSTAB method in the following way respectively:

$$\text{PGPBiCG}(1,0): \quad \widetilde{rr}_{k+1} = (r_0^*, t_k) - \zeta_k(f, s_k); \tag{22}$$

$$\text{IBiCGSTAB}: \quad \widetilde{rr}_{k+1} = \widetilde{rr}_k - \alpha_k(r_0^*, q_k) - \zeta_k(r, s_k). \tag{23}$$

Denote the error of the inner product (r_0^*, r_{k+1}) according to (22)and (23)) as $\delta_{\widetilde{rr}_{k+1}}^{\text{PGPBiCG(1,0)}}$ and $\delta_{\widetilde{rr}_{k+1}}^{\text{IBiCGSTAB}}$ respectively, then according to Lemma 2

$$\delta_{\widetilde{rr}_{k+1}}^{\text{PGPBiCG(1,0)}} = fl(\widetilde{rr}_{k+1}) - r_0^{*T} r_{k+1} = fl\left(r_0^*, t_k\right) - \zeta_k fl\left(f^T s_k\right) - r_0^{*T} r_{k+1} = \delta_{r^T t_k} - \zeta_k \delta_{f^T s_k} \tag{24}$$

and thus we have

$$\left|\delta_{\widetilde{rr}_{k+1}}^{\text{PGPBiCG(1,0)}}\right| \leq \left|\delta_{r_0^* t_k}\right| + |\zeta_k| \left|\delta_{f^T s_k}\right|. \tag{25}$$

While according to (23) in IBiCGSTAB to compute $\widetilde{rr}_{k+1}$, then error

$$\begin{cases} \delta_{\widetilde{rr}_1}^{\text{IBiCGSTAB}} & = & \delta_{r_0^{*T} r_0} - \alpha_0 \delta_{r_0^{*T} q_0} - \zeta_0 \delta_{r_0^{*T} s_0}; \\ \delta_{\widetilde{rr}_{k+1}}^{\text{IBiCGSTAB}} & = & \delta_{\widetilde{rr}_k}^{\text{IBiCGSTAB}} - \alpha_k \delta_{r_0^{*T} q_k} - \zeta_0 \delta_{r_0^{*T} s_k}. \end{cases} \tag{26}$$

Combining the above recursive formulae, one can get

$$\left|\delta_{rr_{k+1}}^{\text{IBiCGSTAB}}\right| \leq \left|\delta_{r_0^T r_0} - \sum_{i=0}^{k} \left(\alpha_i \delta_{r_0^T q_i} + \zeta_i \delta_{r_0^T s_i}\right)\right| \leq \left|\delta_{r_0^T r_0}\right| + \sum_{i=0}^{k} \left(|\alpha_i| \left|\delta_{r_0^T q_i}\right| + |\zeta_i| \left|\delta_{r_0^T s_i}\right|\right) \tag{27}$$

Lemma 2 indicates that these errors $\delta_{r_0^T r_0}$, $\delta_{r_0^T, q_i}$, $\delta_{r_0^T s_i}$, $\delta_{r_0^T t_k}$, $\delta_{f^T s_k}$ can be much larger than machine precision. It is clear from (27) that the recursive way to compute the inner product can accumulate the round off error.

In most cases, the solution x and the residual r_k are dense. Accurately computing the inner product for large dense vectors becomes another challenging issue of large scale computing, which goes beyond our discussion. The reader is directed to [6, 7, 17] [28, Chap.3][39, 41] for details. It should be mentioned that accurate compute the inner product for large dense vectors itself is a challenging problem.

Acknowledgement

Thanks the referees for suggestions to improve the manuscript. Thanks technicians from Dawning Company for technique supports.

References

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation *J. Parallel and Distributed Computing* 44(1):71-79, 1997

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.

[3] D. Brown, P. Messina (Chairs). Scientific Grand Challenges: Crosscutting Technologies for Computing at the Exascale, 2010.

[4] H.M. Bücker and M. Sauren. A parallel version of the unsymmetric Lanczos algorithm and its application to QMR. 1996. www.fz-juelich.de

[5] E. Carson, N. Knight and J. Demmel, Avoiding communication in two-sided Krylov subspace methods, *Technical Report UCB/EECS-2011-93, University of California at Berkeley, USA*, 2011.

[6] A. M. Castaldo. Parallelism and error reduction in a high performance environment *Ph.D Thesis, The University of Texas at San Antonio*, 2011.

[7] A.Castaldo and R. Whaley and A. Chronopoulos. Reducing floating point error in DOT product using the superblock Family of Algorithms. *SIAM. J. Sci. Comput.* 31(2),1156-1174, 2008.

[8] A.T. Chronopoulos, A class of parallel iterative methods implemented on multiprocessors, *Ph.D Thesis, Techical Report UIUCDCS-R-86-1267, Department of Computer Science, University of Illinois, Urbana, Illinois* 1-116, 1986.

[9] A.T. Chronopoulos and C.W. Gear, s-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math*, 25(2):153-168,1989.

[10] A.T. Chronopoulos and C.W. Gear, On the efficient implementation of preconditioned s-step iterative methods on multiprocessors with memory hierachy. *Parallel Computing*, 11(1):37-53,1989.

[11] A. T. Chronopoulos and A.B. Kucherov, Block s-step Krylov iterative methods, *Numer. Linear Algebra Appl.*, 17(3):3-15,2010.

[12] A.T. Chronopoulos and C.D. Swanson, Parallel iterative s-step methods for unsymmetric linear systems. *Parallel Computing*, 22(5):623-641,1996.

[13] T.P. Collignon and M.B. van Gijzen, Minimizing synchronization in IDR(s). *Numer. Linear Algebra Appl.*, 18:805-825, 2011.

[14] E. de Sturler. Iterative methods on distributed memory computers. *Ph. D. Thesis, Delft University of Technology, Delft, Netherlands*, 1994.

[15] E. de Sturler and H. van der Vorst. Communication cost reduction for Krylov methods on parallel computers. In *High-Performance Computing and Networking*, pages 190–195. Springer, 1994.

[16] E. de Sturler and H.A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18(4): 441–459, 1995.

[17] J. Demmel and Y. Hida. Accurate and Efficient Floating Point Summation *SIAM J. Sci. Comput.* (25)4, 1212-1248, 2003.
[18] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE, 2008.
[19] S.L. Graham, and M. Snir, and C.A. Patterson, Getting up to speed: The future of supercomputing. *Committee on the Future of Supercomputing, National Research Council*, 2004
[20] A. Grama, and A. Gupta, and G. Karypis, and V. Kumar, Introduction to Parallel Computing Pearson Education Limited, 2003
[21] Seiji Fujino. GPBiCG(m, ℓ): a hybrid of BiCGSTAB and GPBiCG methods with efficiency and robustness. *Appl. Numer. Math.*, 41(1): 107–117, 2002. Developments and trends in iterative methods for large systems of equations—in memoriam Rüdiger Weiss (Lausanne, 2000).
[22] Tong-Xiang Gu, Xian-Yu Zuo, Xing-Ping Liu, and Pei-Lu Li. An improved parallel hybrid bi-conjugate gradient method suitable for distributed parallel computing. *J. Comput. Appl. Math.*, 226(1): 55–65, 2009.
[23] Tong-Xiang Gu, Xian-Yu Zuo, Li-Tao Zhang, Wan-Qin Zhang, and Zhi-Qiang Sheng. An improved bi-conjugate residual algorithm suitable for distributed parallel computing. *Appl. Math. Comput.*, 186(2): 1243–1253, 2007.
[24] Tong-Xiang Gu, Xing-Ping Liu, Ze-Yao Mo, and Xue-Bin Chi. Multiple search direction conjugate gradient method. I. Methods and their propositions. *Int. J. Comput. Math.*, 81(9): 1133–1143, 2004.
[25] Tong-Xiang Gu, Xing-Ping Liu, Ze-Yao Mo, and Xue-Bin Chi. Multiple search direction conjugate gradient method. II. Theory and numerical experiments. *Int. J. Comput. Math.*, 81(10): 1289–1307, 2004.
[26] M.H. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14(5): 1020–1033, 1993.
[27] M.H. Gutknecht and Z. Strakos, Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM. J. Matrx Anal. Appl.*, 22(1):213-229, 2000
[28] N. Higham. Accuracy and Stability of Numerical Algorithms. SIAM,1996
[29] , M. Hoemmen Communication-avoiding Krylov subspace methods, *Ph.D Thesis, University of California, Berkeley*, 2010.
[30] E.J. Im, K. Yelick, and R. Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications*, 18(1): 135–158, 2004.
[31] Xing-Ping Liu, Tong-Xiang Gu, Xu-Deng Hang, and Zhi-Qiang Sheng. A parallel version of QMRCGSTAB method for large linear systems in distributed parallel environments. *Appl. Math. Comput.*, 172(2): 744–752, 2006.
[32] Gangfeng Liu , Yunlan Wang, Tianhai Zhao, Jianhua Gu, and Dongyang Li mHLogGP:A Parallel Computation model for CPU/GPU Heterogeneous Computing Cluster *Network and Parallel Computing (Lecture Notes in Computer Science)* 217-224, 2012
[33] L. C. McInnes, B. Smith, H. Zhang and R. T. Mills, Hierarchical and Nested Krylov Methods for Extreme-Scale Computing, 2012. *Preprint ANL/MCS-P2097-0612* www.mcs.anl.gov/uploads/cels/papers/P2097-0612.pdf
[34] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24(4): 623–633, 1984.
[35] G. Meurant. Numerical experiments for the preconditioned conjugate gradient method on the Cray X-MP 2'. *Lawrence Berkeley Laboratory LBL-18023*, 1984.
[36] G. Meurant. The conjugate gradient method on vector and parallel supercomputers. Technical report, Technical Report CTAC-89, University of Brisbane, 1989.
[37] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 36. ACM, 2009.
[38] R. Nishtala and K.A. Yelick. Optimizing collective communication on multicores. In *Proceedings of the First USENIX conference on Hot Topics in Parallelism*, pages 18–18. USENIX Association, 2009.
[39] T. Ogita, and S. Rump, and S. Oishi Accurate Sum and Dot Product. *SIAM J. Sci. Comput.* 26(6), 1955-1988, 2005.
[40] D.A.Patterson. Latency lags bandwith. *Communications of the ACM*, 47(10): 71–75, 2004.
[41] S. Rump, and S. Ogita, and S. Oishi. Accurate Floating-Point Summation Part I: Faithful Rounding *SIAM J. Sci. Comput.* 31(1),189–224, 2008.
[42] G.L.G. Sleijpen and H.A. van der Vorst. Hybrid bi-conjugate gradient methods for CFD problems. *Computational fluid dynamics review*, pages 457–476, 1995.
[43] G.L.G. Sleijpen and H.A. van der Vorst. Reliable updated residuals in hybrid Bi-CG methods *Computing*, 56(2):141-163, 1996.
[44] Valiant, Leslie G. A bridging model for parallel computation *Commun. ACM*, 33(8):103-111, 1990.
[45] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13: 631, 1992.
[46] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick and J. Demmel, Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms, *SC07*, Nov. 10-16, 2007, Reno, Nevada, USA.
[47] L.T. Yang. The improved CGS method for large and sparse linear systems on bulk synchronous parallel architectures. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 232–237. IEEE, 2002.
[48] L.T. Yang and R.P. Brent. The improved BiCGSTAB method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 324–328. IEEE, 2002.
[49] T.R. Yang and H.X. Lin. The improved quasi-minimal residual method on massively distributed memory computers. In *High-Performance Computing and Networking*, pages 389–399. Springer, 1997.
[50] Lin-Bo Zhang, Xue-Bin Chi, Ze-Yao Mo, and Ruo Li. *Introduction to Parallel Computing*. Tsinghua University Press, Beijing, 2006, (in Chiness).
[51] S.L. Zhang. GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 18:537, 1997.
[52] S. Zhu Parellel GPBiCG(m, ℓ) methods and preconditioning techneques. *Master Thesis, Graduate School, China Academy of Engineering Physics*, 2010
[53] Xian-Yu Zuo, Tong-Xiang Gu, and Ze-Yao Mo. An improved GPBi-CG algorithm suitable for distributed parallel computing. *Applied*

Mathematics and Computation, 215(12): 4101–4109, 2010.

RECENT REPORTS

13/05	Simple computation of reaction-diffusion processes on point clouds	Macdonald Merriman Ruuth
13/06	A Volume-Based Method for Denoising on Curved Surfaces	Biddle von Glehn Macdonald März
13/07	Porous squeeze-film flow	Knox Wilson Duffy McKee
13/08	Diffusion of finite-size particles in confined geometries	Bruna Chapman
13/09	Mathematical analysis of a model for the growth of the bovine corpus luteum	Prokopiou Byrne Jeffrey Robinson Mann Owen
13/10	Capillary deformations of bendable films	Schroll Adda-Bedia Cerda Huang Menon Russell Toga Vella Davidovitch
13/11	Twist and stretch of helices: All you need is Love	Đuričković Goriely Maddocks
13/12	Switch on, switch off: stiction in nanoelectromechanical switches	Wagner Vella
13/13	Pinning, de-pinning and re-pinning of a slowly varying rivulet	Paterson Wilson Duffy
13/14	Travelling-wave similarity solutions for a steadily translating slender dry patch in a thin fluid film	Yatim Duffy Wilson
13/15	A stochastic model for early placental development	Cotter Klika Kimpton Collins Heazell
13/16	Experimentally-calibrated population of models predicts and explains inter-subject variability in cardiac cellular electrophysiology	Britton Bueno-Orovio Van Ammel

13/21	Glyph-based video visualization for semen analysis	Duffy Thiyagalingam Walton Smith Trefethen Kirkman-Brown Gaffney Chen
13/22	RBF multiscale collocation for second order elliptic boundary value problems	Farrell Wendland
13/23	Na/K pump regulation of cardiac repolarization: Insights from a systems biology approach	Bueno-Orovio Sánchez Pueyo Rodriguez
13/24	Cellular blebs - pressure-driven, axisymmetric, membrane protrusions	Woolley Gaffney Oliver Baker Waters Goriely
13/25	Growth-induced axial buckling of a slender elastic filament embedded in an isotropic elastic matrix	O'Keeffe Moulton Waters Goriely
13/26	The counterbend phenomenon: a generic property of the axoneme and cross-linked filament bundles	Gadêlha Gaffney Goriely
13/27	A well-posedness framework for inpainting based on coherence transport	März

Copies of these, and any other OCCAM reports can be obtained from:

Oxford Centre for Collaborative Applied Mathematics
Mathematical Institute
24 - 29 St Giles'
Oxford
OX1 3LB
England
www.maths.ox.ac.uk/occam