Minimizing synchronizations in sparse iterative solvers for distributed supercomputers

by

Sheng-Xin Zhu
Tong-Xiang Gu
Xing-Ping Liu
Minimizing synchronizations in sparse iterative solvers for distributed supercomputers

Sheng-Xin Zhu\textsuperscript{a,1}, Tong-Xiang Gu\textsuperscript{b,2}, Xing-Ping Liu\textsuperscript{b,2}

\textsuperscript{a}Oxford Centre for Collaborative and Applied Mathematics, Mathematical Institute, 24-29 St Giles’, Oxford, OX1 3LB, U.K.
\textsuperscript{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 demonstrate 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 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...
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, $t_{fli}$ 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 an iteration in a Krylov subspace methods as follow

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

Where $\phi(N/P)t_{fli}$ 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>
<thead>
<tr>
<th>Name</th>
<th>Operations</th>
<th>Computation time</th>
<th>Communication time</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix-vector multiplication</td>
<td>$y = Ax$</td>
<td>$n_i N_{fli}/P$</td>
<td>$\mu(t_s, t_w)$</td>
</tr>
<tr>
<td>one inner product</td>
<td>$\text{dot} \leftarrow x^T y$</td>
<td>$2N_{fli}/P$</td>
<td>$2(t_s + t_w)\omega(P)$</td>
</tr>
<tr>
<td>k inner products</td>
<td>—</td>
<td>$2kN_{fli}/P$</td>
<td>$2(t_s + kt_w)\omega(P)$</td>
</tr>
<tr>
<td>vector update</td>
<td>$y \leftarrow ax + y$</td>
<td>$2N_{fli}/P$</td>
<td>—</td>
</tr>
</tbody>
</table>

$N$: the number of unknowns; $P$: the number of processors; $n_i$: the average number of non-zeros per row of $A$; $t_{fli}$: 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 one, is to remove inner production computations which need global communications, developing inner product free 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 \( r_s \) 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, \( \zeta_k \) depends on \( s_k \) and \( t_k \) which themselves depends on \( a_k, b_k \) in line 18 depends on the residual \( r_{k+1} \) which itself depends \( \eta_k \) and \( \zeta_k \). 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 \( \mu(t_s, t_u, P) \) 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 \( (r_o^s, r_{k+1}) \) can be computed indirectly by

\[
\tilde{r}_{r_{k+1}} := (r_o^s, r_{k+1}) = (r_o^s, t_k) - \eta_k (r_o^s, y_k) - \zeta_k (r_o^s, s_k).
\] (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($m, \ell$)

1: $r_0 = b - A x_0$, $t_{-1} = w_{-1} = 0$, choose arbitrary $r_0^*$, s.t. $(r_0^*, r_0) \neq 0$. 
2: for $k = 0, 1, \ldots$, until $\|r_k\| < \eta_0$ do 
3: $p_k = r_k + \beta_{k-1} (p_{k-1} - u_{k-1})$; $q_k = Ap_k$
4: $\alpha_k = \frac{(r_k, r_k)}{\eta_k}$
5: $t_k = r_k - \alpha_k q_k$; $s_k = At_k$
6: $y_k = t_{k-1} - t_k - \alpha_k w_{k-1}$
7: if $\mod(k, m + \ell) < m$ or $k = 0$ then 
8: $\zeta_k = \frac{(s_k, t_k)}{(s_k, s_k)}$
9: $u_k = \zeta_k q_k$
10: $z_k = \zeta_k t_k - \alpha_k u_k$
11: $r_{k+1} = t_k - \zeta_k s_k$
12: else 
13: $h_k = tem - r_k + \beta_{k-1} u_{k-1}$
14: compute all the inner products:
15: 
16: end if 
17: end for 

Algorithm 2 Parallel GPBiCG($m, \ell$)

1: $r_0 = b - A x_0$, $t_{-1} = w_{-1} = 0$, $p_0 = r_0$, $q_0 = Ap_0$; choose arbitrary $r_0^*$, s.t. $(r_0^*, r_0) \neq 0$, $f = A^T r_0^*$, $\alpha = (r_0^*, r_0)/(f, p_0)$.
2: for $k = 0, 1, \ldots$, until convergence do 
3: $t_k = r_k - \alpha q_k$;
4: $s_k = At_k$
5: $y_k = tem - t_k - \alpha w_{k-1}$
6: if $\mod(k, m + \ell) < m$ or $k = 0$ then 
7: compute all the inner products:
8: 
10: $\zeta_k = \frac{(s_k, t_k)}{(s_k, s_k)}$; $\eta_k = \frac{(s_k, y_k)(s_k, t_k) - (y_k, s_k)(s_k, t_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
11: $\zeta_k = \frac{(s_k, y_k)(s_k, t_k) - (y_k, s_k)(s_k, t_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
12: $\zeta_k = \zeta_k t_k - \alpha_k u_k$
13: $r_{k+1} = t_k - \zeta_k s_k$
14: else 
15: $h_k = tem - r_k + \beta_{k-1} u_{k-1}$
16: compute all the inner products:
17: 
19: end if 
20: $\beta_k = \frac{\eta_k\eta_k}{\eta_k\eta_k}$
21: $\eta_k = \frac{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
22: $\zeta_k = \frac{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
23: $\zeta_k = \zeta_k t_k - \alpha_k u_k$
24: $r_{k+1} = t_k - \zeta_k s_k$
25: end for 
27: $\beta_k = \frac{\eta_k\eta_k}{\eta_k\eta_k}$
28: $\alpha_k = \frac{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
29: $\alpha_k = \frac{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}{(s_k, s_k)(y_k, y_k) - (y_k, s_k)(s_k, y_k)}$
Table 2: The main computations of GPBiCG($m, \ell$) and PGPBiCG($m, \ell$)

<table>
<thead>
<tr>
<th>Methods</th>
<th>vector update</th>
<th>inner product</th>
<th>matrix-vector</th>
<th>synchronization points</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPBiCG($m, \ell$)</td>
<td>$H$ if else $T$</td>
<td>1 2 5 2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>PGPBiCG($m, \ell$)</td>
<td>$H$ if else $T$</td>
<td>0 9 15 0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

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, $\alpha_{k+1}$ is considered instead of $\alpha_k$ for each loop. And the synchronization point to compute $\alpha_{k+1}$ is also reduced. See Algorithm 2.

It is noted that computing $\alpha_{k+1}$ needs $(r_0^*, q_{k+1})$; the update of $q_{k+1}$ depends on a matrix-vector multiplication, $q_{k+1} = A p_{k+1}$. 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

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

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

$$\tilde f p_{k+1} := (f, p_{k+1}) = (f, r_{k+1}) + \beta_k ((f, p_k) - (f, u_k)).$$

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

$$\tilde f r_{k+1} := (f, r_{k+1}) = (f, t_k) - \eta_k (f, y_k) - \zeta_k (f, s_k).$$

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

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

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 $\zeta_k$ and $\eta_k$ 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.
Algorithm 1, and substituting the time for each kernels into equation 1, we get a the expected time for a "unit loop" integer and regard \( m \) inner products, therefore we can not use the average number of branch and else

\[ t_{n+1} = 2t_{fl}/N + (t_1 + t_w) \omega(P) \]

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

\[ t_{inn(k)} = \frac{2kNt_{fl}}{P} + (t_s + kt_w) \omega(P) \]  

(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 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 \( \ell \) 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”

\[ T_{\text{GPBiCG}(m, \ell)} = (8(m + \ell) + 3m + 7\ell)t_{vec} + 2(m + \ell)t_{mv} + (m + \ell)(t_{inn(1)} + t_{inn(2)}) + mt_{inn(2)} + \ell t_{inn(5)}, \]

(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

\[ T_{\text{GPBiCG}(m, \ell)} = \frac{\lambda_1}{P} + \lambda_2 \omega(P) + 2(m + \ell)\mu(t_s, t_w, P), \]

(9)

where \( \mu \) is the local communication time defined in Table 1 and

\[ \lambda_1 = \{32m + 46\ell + 2(m + \ell) n_z\} Nt_{fl}, \]

(10)

\[ \lambda_2 = 3(m + \ell)t_s + (5m + 8\ell)t_w. \]

(11)

Similarly, the time for one “unit loop” in PGPBiCG(m, \ell) is

\[ T_{\text{PGPBiCG}(m, \ell)} = \frac{\sigma_1}{P} + \sigma_2 \omega(P) + 2(m + \ell)\mu(t_s, t_w, P), \]

(12)

where

\[ \sigma_1 = \{40m + 60\ell + 2(m + \ell) n_z\} Nt_{fl}; \]

(13)

\[ \sigma_2 = (m + \ell)t_s + (9m + 15\ell)t_w. \]

(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

\[
\eta = \frac{T_{\text{PGPBiCG}(m, \ell)} - T_{\text{PGPBiCG}(m, \ell)}}{T_{\text{PGPBiCG}(m, \ell)}} = \frac{(\lambda_2 - \sigma_2) P^2(P) + \lambda_1 - \sigma_1}{\lambda_2 P^2(P) + 2(m + \ell) \mu P + \lambda_1},
\]

\[
= \frac{[2(m + \ell) t_s - (4m + 6\ell) t_w] P^2(P) - (8m + 14\ell) N t_f}{[3(m + \ell) t_s + (5m + 8\ell) t_w] P^2(P) + 2(m + \ell) \mu P + \lambda_1}.
\]

Because \( t_s \gg t_w \), when \( P \) is large enough, \( \eta \rightarrow \frac{2}{3} \).

**Conclusion 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 \( t_s = r_k - \alpha_k q_k \) in line 4, then the inner product \( \bar{f}_{r_{k+1}} \) in line 12 and line 22 and inner product \( \bar{f}_{p_{k+1}} \) in line 32 can be computed in a recursive way:

\[
\bar{f}_{r_{k+1}} = \bar{f}_{r_k} - \alpha_k (f, q_k) - \eta_k (f, y_k) - \zeta_k (f, s_k),
\]

\[
\bar{f}_{p_{k+1}} = \bar{f}_{p_{k+1}} + \beta_k (\bar{f}_{p_k} - (f, u_k)).
\]

In such a recursive way, if the inner product \((f, r^*_0) = (f, p_0)\) is provided, then the inner products \(\bar{f}_{r_{k+1}}\) and \(\bar{f}_{p_{k+1}}\) 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

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + c \frac{\partial u}{\partial x} + d \frac{\partial u}{\partial y} + eu = 0, \quad x, y \in (0, 1),
\]

with boundary conditions

\[
\begin{cases}
\frac{\partial u}{\partial x}\big|_{x=0} = 0, & \frac{\partial u}{\partial x}\big|_{x=1} = 0 \\
\frac{\partial u}{\partial y}\big|_{y=0} = 0, & u_{y=1} = 10,
\end{cases}
\]

where \( a = b = 1512.0, \quad c = d = 1.0, \quad e = 0.0 \). 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.
Figure 1: Comparison on the performance of GPBiCG($m$, $\ell$) method (blue curves) and PGPBiCG($m$, $\ell$) 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($m$, $\ell$) is about 2.7 times of that of PGPBiCG($m$, $\ell$). The factor 2.7 reasonable, because there are 3 global synchronizations in the GPBiCG($m$, $\ell$) method, whereas there is only one synchronization but more inner products in PGPBiCG($m$, $\ell$).

Figure 1 illustrates how the solution time for GPBiCG($m$, $\ell$) and PGPBiCG($m$, $\ell$) 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, $\eta$, 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
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 $P^{1/5}$ 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 $\sqrt[3]{(P^{1/3})^2} = \sqrt[15]{P^{1/3}}$. 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

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

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.
Figure 3: Global communication time and local communication time. In each window, the left half side is the data of GPBiCG($m, \ell$) methods and the right half side is the data for PGPBiCG($m, \ell$). 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 ‘×’). 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 multiplication and local communication time
time3=AZ_second(); \# timing
Amat->matvec(temptt, 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 \( fl(\langle x^T y \rangle) \) is the inner product computed by float points computations, \( x^T y \) is the true value, \( \delta = fl(\langle x^T y \rangle) - x^T y \), then

\[
|\delta| \leq 1.01Nu \sum_{i=1}^{N} |x_i y_i|
\]

where \( u \) is the machine precision.

Lemma 2 shows that \( |\delta| \) can be far larger than the machine precision \( u \) 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 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:

\[
\text{PGPBICG(1, 0)}: \quad \tilde{r}_{k+1} = (r_0, t_k) - \zeta_k(f, s_k); \quad (22)
\]

\[
\text{IBiCGSTAB}: \quad \tilde{r}_{k+1} = \tilde{r}_k - \alpha_k(r_0, q_k) - \zeta_k(r, s_k). \quad (23)
\]
Denote the error of the inner product \((r_0^T, r_{k+1})\) according to (22) and (23) as \(\delta^\text{PGPBiCG(1,0)}_{r_{k+1}}\) and \(\delta^\text{IBiCGSTAB}_{r_{k+1}}\) respectively, then according to Lemma 2

\[
\delta^\text{PGPBiCG(1,0)}_{r_{k+1}} = f_l(\tilde{r}_{k+1}) - r_0^T r_{k+1} = f_l\left(r_0^T t_k\right) - \zeta_k f_l\left(t_k^T s_k\right) - r_0^T r_{k+1} = \delta r^T t_k - \zeta_k \delta^T s_k \tag{24}
\]

and thus we have

\[
|\delta^\text{PGPBiCG(1,0)}_{r_{k+1}}| \leq \left|\delta r^T t_k\right| + |\zeta_k| \left|\delta^T s_k\right|. \tag{25}
\]

While according to (23) in IBiCGSTAB to compute \(\tilde{r}_{k+1}\), then error

\[
\begin{align*}
\delta^\text{IBiCGSTAB}_{r_{k+1}} &= \delta r^T r_0 - \alpha_0 \delta r^T q_0 - \zeta_0 \delta^T s_0, \\
\delta^\text{IBiCGSTAB}_{r_k} &= \delta r^T r_k - \alpha_k \delta r^T q_k - \zeta_k \delta^T s_k. \tag{26}
\end{align*}
\]

Combining the above recursive formulae, one can get

\[
|\delta^\text{IBiCGSTAB}_{r_{k+1}}| \leq \left|\delta r^T r_0\right| - \sum_{i=0}^{k} \left(\alpha_i \left|\delta r^T q_i\right| + \zeta_i \left|\delta^T s_i\right|\right) \leq \left|\delta r^T r_0\right| + \sum_{i=0}^{k} \left(|\alpha_i| \left|\delta r^T q_i\right| + |\zeta_i| \left|\delta^T s_i\right|\right) \tag{27}
\]

Lemma 2 indicates that these errors \(\delta r^T r_0, \delta r^T q_k, \delta^T s_k, \delta^T s_k\) 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 \(r_0\) 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

## RECENT REPORTS

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

RBF multiscale collocation for second order elliptic boundary value problems
Farrell
Wendland

Na/K pump regulation of cardiac repolarization: Insights from a systems biology approach
Bueno-Orovio
Sánchez
Pueyo
Rodríguez

Cellular blebs - pressure-driven, axisymmetric, membrane protrusions
Woolley
Gaffney
Oliver
Baker
Waters
Goriely

Growth-induced axial buckling of a slender elastic filament embedded in an isotropic elastic matrix
O’Keeffe
Moulton
Waters
Goriely

The counterbend phenomenon: a generic property of the axoneme and cross-linked filament bundles
Gadêlha
Gaffney
Goriely

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
England
www.maths.ox.ac.uk/occam