Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Reducing Synchronization Overheads in CG-type Parallel Iterative Solvers by Embedding Point-to-point Communications into Reduction Operations

R. Oguz Selvitopi[a], Cevdet Aykanat[a*]

[a]*Bilkent University, Computer Engineering Department, 06800 Ankara, TURKEY*

Abstract

Parallel iterative solvers are widely used in solving large sparse linear systems of equations on large-scale parallel architectures. These solvers generally contain two different types of communication operations: point-to-point (P2P) and global collective communications. In this work, we present a computational reorganization method to exploit a property that is commonly found in Krylov subspace methods. This reorganization allows P2P and collective communications to be performed simultaneously. We realize this opportunity to embed the content of the messages of P2P communications into the messages exchanged in the collective communications in order to reduce the latency overhead of the solver. Experiments on two different supercomputers up to 2048 processors show that the proposed latency-avoiding method exhibits superior scalability, especially with increasing number of processors.

1. Introduction

Iterative solvers [1] are widely used and adopted to solve sparse linear systems of equations on modern large-scale parallel systems. In these systems, the communication requirements of the solver generally become the main bottleneck for obtaining a good scalable performance. For this reason, the coefficient matrix is usually processed in a pre-processing phase, which involves partitioning of this matrix to reduce the communication requirements. In the literature, the most often used and optimized communication matrix is the communication volume [2] [3] [4] [5].

In iterative solvers, there are two types of communication that are repeated through all iterations:

- *Collective communication operations*: This type of communication is used to gather the results of the inner product computations at all processors and requires all processors to join the communication. The MPI equivalent of this operation is the *MPI_Allreduce* (hereafter referred to as *ALL-REDUCE*) with the summation being the reduction operator. There are various algorithms that can be used to implement *ALL-REDUCE* [6]. In this work, we focus on the bidirectional exchange algorithm as it incurs the least communication overhead. This algorithm completes the reduction operation in $\log_2 P$ steps where P is the number of processors in the system, and it is a power of 2.
- *Irregular point-to-point (P2P) communication operations:* This type of operation is used to communicate the entries of the input and/or output vector of the sparse-matrix vector multiplication

* Corresponding author. *E-mail address*: aykanat@cs.bilkent.edu.tr

(SpMV). The irregular sparsity pattern of the coefficient matrix causes irregular task-to-task interactions between parallel processes. They are generally performed by simple MPI primitives, e.g., *MPI_Send* , *MPI_Recv*, and their variants. The P2P communications often constitute the most time-consuming communication phase as they are dependent on the sparsity pattern of the coefficient matrix and are not very suitable for optimization, as opposed to collective operators which generally involve all processors. Due to the irregular sparsity pattern, P2P operators usually do not require all processors to involve in communication.

These two types of communication operations introduce separate synchronization points to the solver, which hinders the solver's scalability. The existing pre-processing matrix decomposition tools generally aim at reducing the total communication volume. Our analysis at current supercomputers indicates that message latency is another important parameter that affects the communication performance of the solver, especially with increasing numbers of processors. For example, our experiments on IBM BlueGene/Q and Cray XE6 machines showed that a single message latency (transmission of a single message, startup time) can be as high as transmitting 2-4 KB of data between two processors.

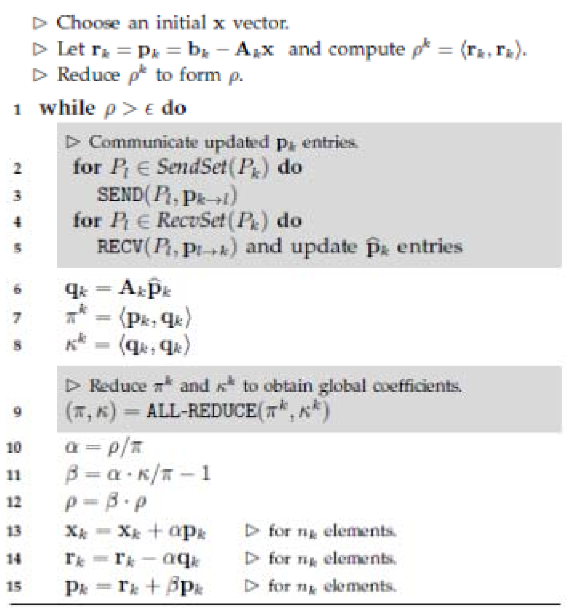
In this work, we devise a computational reorganization method to perform P2P and collective communication operations simultaneously. This allows the number of synchronization points in a single iteration of the solver to drop from two to one for a single pair of SpMV and its follow-up inner product(s). Our observation relies on the fact that SpMV operations are generally followed by the inner product computations that involve the output vector of the SpMV computations. This observation is valid for almost all Krylov subspace methods. We realize this opportunity by embedding P2P communication operations into the collective communication operations. This enables us to provide an upper bound (and an exact value) on the number of messages being communicated, hence providing an upper bound on the latency cost of the solver which should have been dependent on the coefficient matrix. More specifically, the proposed approach completely eliminates the message latency costs due to the SpMV operations and reduces the average and maximum number of messages handled by a single processor (send/recv) to $\log_2 P$ for a system with P processors. On the other hand, it increases the total volume of communication since the embedding method requires a store-and-forward scheme. To address the increase in the communication volume, we propose two heuristics whose main motivation is to keep the processors that communicate high volumes of data close to each other.

We use a modified Conjugate Gradient (CG) iterative solver to show the validity of the proposed methods. We use 1D partitioning of the matrix and test the solver on a BlueGene/Q and an XE6 up to 2048 cores with the matrices selected from UFL sparse-matrix collection [7].

2. Computational Reorganization

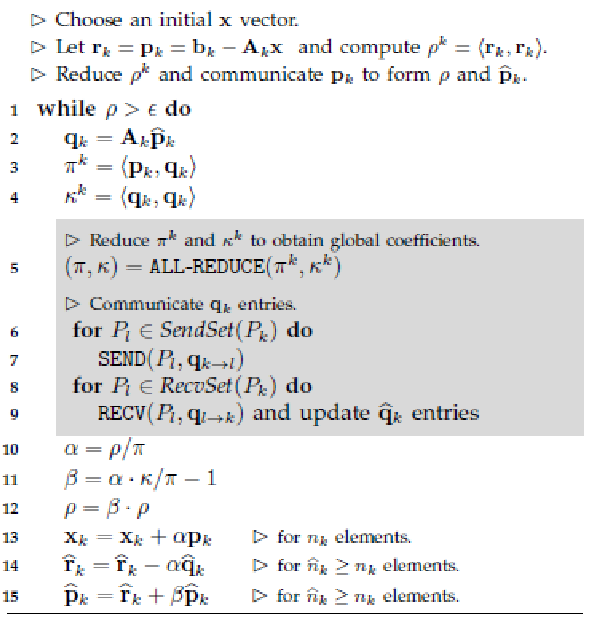
The variant of the CG we use is introduced in [8]. This version is more amenable to parallelization since, compared to basic textbook CG, the results of two inner products can be performed in a single reduction operation, allowing to avoid the overhead of a single reduction operation.

In common parallelizations of CG, the coefficient matrix is row-wise decomposed and distributed to P processors. Figure 1 illustrates this version without computational reorganization, for processor P_k. As seen, there are two separate communication phases highlighted as the shaded regions: (i) one of them is the P2P communications prior to SpMV computations (lines 2-5) and (ii) the other one is the collective communication after the local inner products (line 9). This common parallel algorithm has two synchronization points due to these P2P and collective communication phases.



```
▷ Choose an initial $\mathbf{x}$ vector.
▷ Let $\mathbf{r}_k = \mathbf{p}_k = \mathbf{b}_k - \mathbf{A}_k\mathbf{x}$ and compute $\rho^k = \langle \mathbf{r}_k, \mathbf{r}_k \rangle$.
▷ Reduce $\rho^k$ to form $\rho$.
while $\rho > \epsilon$ do
    ▷ Communicate updated $\mathbf{p}_k$ entries.
    for $P_l \in SendSet(P_k)$ do
        SEND($P_l, \mathbf{p}_{k \to l}$)
    for $P_l \in RecvSet(P_k)$ do
        RECV($P_l, \mathbf{p}_{l \to k}$) and update $\widehat{\mathbf{p}}_k$ entries
    $\mathbf{q}_k = \mathbf{A}_k\widehat{\mathbf{p}}_k$
    $\pi^k = \langle \mathbf{p}_k, \mathbf{q}_k \rangle$
    $\kappa^k = \langle \mathbf{q}_k, \mathbf{q}_k \rangle$
    ▷ Reduce $\pi^k$ and $\kappa^k$ to obtain global coefficients.
    $(\pi, \kappa) =$ ALL-REDUCE($\pi^k, \kappa^k$)
    $\alpha = \rho/\pi$
    $\beta = \alpha \cdot \kappa/\pi - 1$
    $\rho = \beta \cdot \rho$
    $\mathbf{x}_k = \mathbf{x}_k + \alpha\mathbf{p}_k$        ▷ for $n_k$ elements.
    $\mathbf{r}_k = \mathbf{r}_k - \alpha\mathbf{q}_k$        ▷ for $n_k$ elements.
    $\mathbf{p}_k = \mathbf{r}_k + \beta\mathbf{p}_k$        ▷ for $n_k$ elements.
```

Figure 1: Common parallelization



```
▷ Choose an initial $\mathbf{x}$ vector.
▷ Let $\mathbf{r}_k = \mathbf{p}_k = \mathbf{b}_k - \mathbf{A}_k\mathbf{x}$ and compute $\rho^k = \langle \mathbf{r}_k, \mathbf{r}_k \rangle$.
▷ Reduce $\rho^k$ and communicate $\mathbf{p}_k$ to form $\rho$ and $\widehat{\mathbf{p}}_k$.
while $\rho > \epsilon$ do
    $\mathbf{q}_k = \mathbf{A}_k\widehat{\mathbf{p}}_k$
    $\pi^k = \langle \mathbf{p}_k, \mathbf{q}_k \rangle$
    $\kappa^k = \langle \mathbf{q}_k, \mathbf{q}_k \rangle$
    ▷ Reduce $\pi^k$ and $\kappa^k$ to obtain global coefficients.
    $(\pi, \kappa) =$ ALL-REDUCE($\pi^k, \kappa^k$)
    ▷ Communicate $\mathbf{q}_k$ entries.
    for $P_l \in SendSet(P_k)$ do
        SEND($P_l, \mathbf{q}_{k \to l}$)
    for $P_l \in RecvSet(P_k)$ do
        RECV($P_l, \mathbf{q}_{l \to k}$) and update $\widehat{\mathbf{q}}_k$ entries
    $\alpha = \rho/\pi$
    $\beta = \alpha \cdot \kappa/\pi - 1$
    $\rho = \beta \cdot \rho$
    $\mathbf{x}_k = \mathbf{x}_k + \alpha\mathbf{p}_k$        ▷ for $n_k$ elements.
    $\widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_k - \alpha\widehat{\mathbf{q}}_k$        ▷ for $\widehat{n}_k \geq n_k$ elements.
    $\widehat{\mathbf{p}}_k = \widehat{\mathbf{r}}_k + \beta\widehat{\mathbf{p}}_k$        ▷ for $\widehat{n}_k \geq n_k$ elements.
```

Figure 2: Alternative parallelization

Figure 2 presents the alternative parallelization with computational reorganization. In this parallelization, the input vector of the SpMV computations is not formed with the P2P communications but it is formed with the help of the other vectors. Instead of communicating this input vector, the output vector is communicated and it is augmented with the entries that are received from other processors. This augmented output vector is then subjected to the same linear vector operations as the augmented input vector of the SpMV, requiring no further communication. This reorganization enables P2P communications to be performed right after collective communication operations, reducing two separate communication phases into one. The single and trivial drawback of this reorganization is the redundant computation performed by each processor in linear vector operations on augmented vectors instead of local vectors. This is not of prime concern since the main computational burden of the solver lies in SpMV operations.

3. Embedding of Communication Operators

We realize the opportunity provided by the computational reorganization by performing these two types of communication phases in a single one. The P2P and collective communications are performed simultaneously by embedding messages of P2P communications into the communication pattern of the algorithm used for the *ALL-REDUCE* algorithm. In other words, the latency overhead due to the P2P communications is completely eliminated by using the messages that are already transmitted for *ALL-REDUCE*.

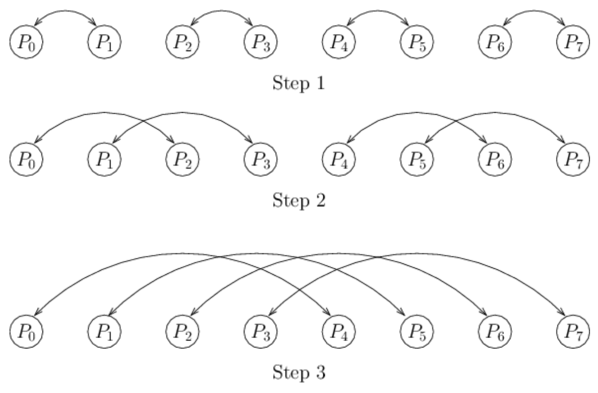


Figure 3: Steps of bidirectional exchange algorithm for 8 processors.

For *ALL-REDUCE*, we use the bidirectional exchange algorithm presented in [6]. In a P-processor parallel system, the collective communication for *ALL-REDUCE* can be completed in only $\log_2 P$ steps. This algorithm works in successive steps by simultaneous exchange of data between processors where in step d, each processor exchanges a message with the processor in its 2^{d-1} distance and updates its local buffer with the received elements by using an associative operator (MPI_SUM operator in the solver). The steps of this algorithm are depicted in Figure 3.

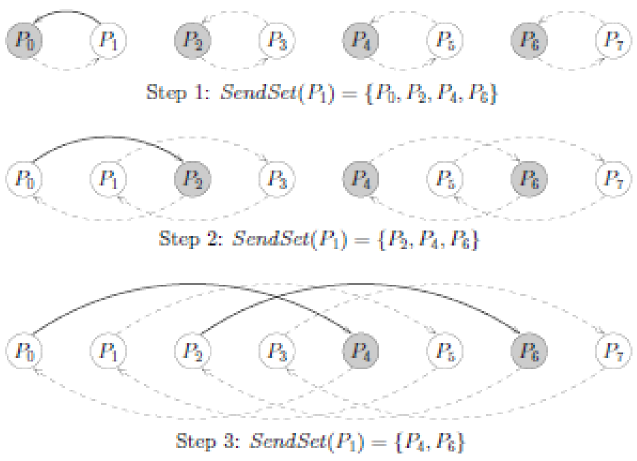

P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7
Step 1: $SendSet(P_1) = \{P_0, P_2, P_4, P_6\}$
Step 2: $SendSet(P_1) = \{P_2, P_4, P_6\}$
Step 3: $SendSet(P_1) = \{P_4, P_6\}$

Figure 4: Embedding example for processor P_1 that needs to send messages to P_0, P_2, P_4 and P_6.

Since processor P_k does not directly communicate with all processors in the *ALL-REDUCE* algorithm, a store-and-foward scheme is needed to work out the path of the messages for P2P communication operations. The critical observation here is that if P_k were to communicate with processor P_l, it is actually certain that a message from P_k will reach P_l, though it may be direct or indirect (thus requiring store). If these two processors do not communicate directly, then P_k can use other processors it directly communicates with to send the messages that are meant for P_l by embedding necessary vector elements into these messages. These other processors then simply forward them to P_l through a pre-determined path. An example of the store-and-forward scheme is illustrated in Figure 4, where in an 8 processor system, P_l is to send P2P messages to P_0, P_2, P_4, and P_6. Observe that since P_l directly communicates with P_0, there is no need for store-and-forward for the messages that will be communicated between these two processors. However, for P_1 to send messages to P_6, P_1 needs to embed the contents of the messages it will send to P_6 in Step 1 to send them P_0, which will in turn forward them to P_2 in Step 2, and then in the final Step 3, P_2will forward them to P_6.

Embedding messages of P2P communications into collective communications have the following implications:

- Startup costs of all messages due to P2P communications are completely avoided.
- An exact bound on the maximum and average number of messages is provided, which is $\log_2 P$ for a parallel system with P processors. This is a significant advantage and is actually the key factor in obtaining a good scalable performance at large processor counts.
- Communication volume increases due to the store-and-forward scheme required by the embedding.
- Embedding scheme requires buffering due to the store-and-forward scheme.
- There is a trade-off between avoiding latency costs and increasing communication volume. Here, the former is favoured, because, as will be shown with the experiments, message latency becomes the dominating factor in determining the communication costs with increasing number of processors.

4. Mapping Algorithms

The store-and-forward scheme used in embedding contents of P2P messages into the messages of collective communication operations may increase communication volume. If the total number of P2P messages is low, this can be a bottleneck in obtaining a good scalable performance. We present two heuristics to further reduce this increased communication volume. The objective of both mapping heuristics is to keep the pairs of processors that communicate a large volume of data close to each other. The closeness notion here refers to the communication pattern used for the *ALL-REDUCE* algorithm. Both of the heuristics are Kernighan-Lin (KL) [9] type of algorithms which try to find a good mapping by a number of successive swap operations:

- *KLF*: Use full neighbourhood information with $P(P-1)$ possible swaps.
- *KLR*: Restrict the search space to the processors that directly communicate, thus reducing the number of possible swaps to $P\log P/2$.

For more detail on these heuristics, refer to [10].

5. Experiments

We compare four schemes in our experiments:

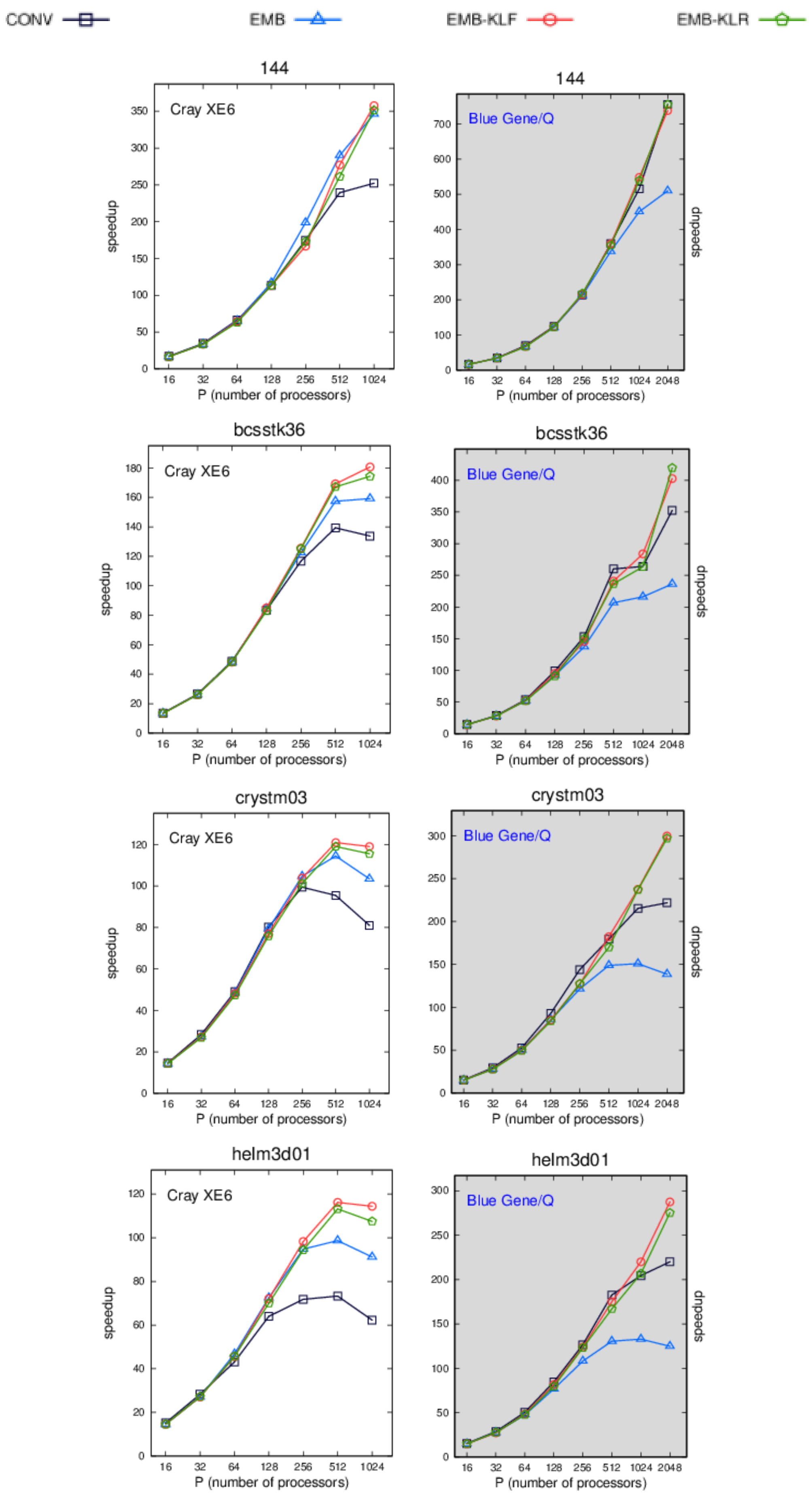
- *CONV*: Common parallelization of conjugate gradient solver.
- *EMB*: Alternative parallelization with computational reorganization.
- *EMB-KLF*: Alternative parallelization with computational reorganization and mapping algorithm KLF.
- *EMB-KLR*: Alternative parallelization with computational reorganization and mapping algorithm KLR.

We used PaToH [11] to partition all matrices prior to execution. Two parallel systems are used in the experiments: Cray XE6 (XE6) and IBM Blue Gene/Q (BG/Q). A node on XE6 consists of 32 cores (two 16-core AMD processors) with 2.3 GHz clock frequency and 32 GB memory. The nodes are connected with a high speed 3D torus network called CRAY Gemini. A node on BG/Q consists of 16 cores (single PowerPC A2 processor) with 1.6 GHz clock frequency and 16 GB memory. The nodes are connected with 5D torus chip-to-chip network.

We have tested these four schemes with 8 matrices selected from UFL [7]. The properties of these matrices are given in Table 1.

	Number of		**Nonzeros per row/col**		
Matrix	**rows/cols**	**nonzeros**	**avg**	**min**	**max**
144	144649	2148786	14.86	4	26
bcsstk36	23052	1143140	49.59	8	178
crystm03	24696	583770	23.64	8	27
helm3d01	32226	428444	13.29	3	37
olesnik0	88263	744216	8.43	1	11
onera_dual	85567	419201	4.90	3	5
pcrystk02	13965	968583	69.36	24	81
t3dl	20360	509866	25.04	8	27

Table 1: Test matrices and their properties.


CONV
EMB
EMB-KLF
EMB-KLR
144
Cray XE6
Blue Gene/Q
bcsstk36
crystm03
helm3d01
speedup
P (number of processors)
16
32
64
128
256
512
1024
2048

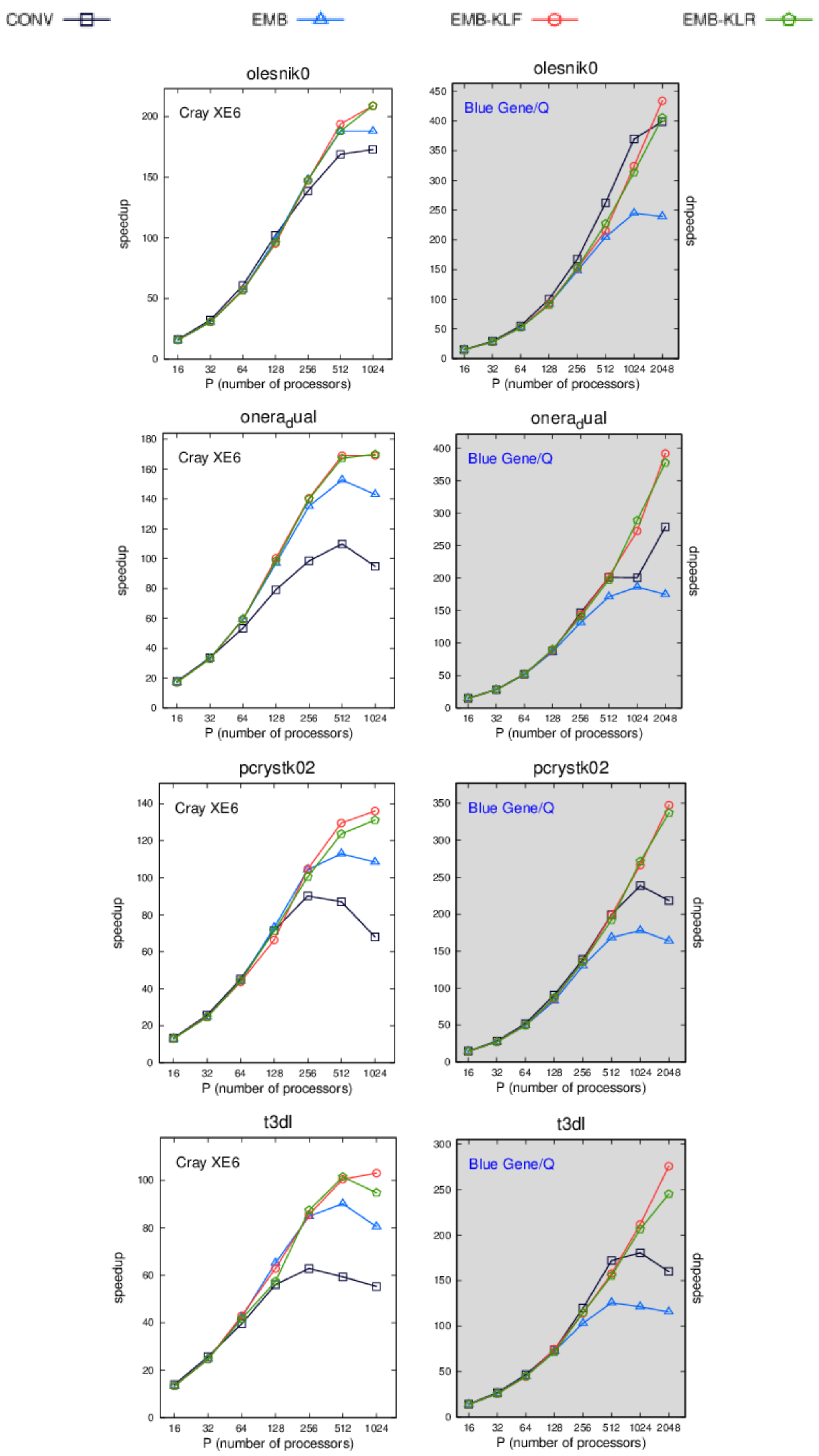


Figure 5: Speedup results for 8 matrices on Cray XE6 and BlueGene/Q

The obtained speedup values are illustrated in Figure 5. As seen from these figures, with increasing number of processors, the proposed three schemes usually scale better compared to the common parallelization. The effect of message latency is more prominent on the XE6. The poor performance of *CONV* schemes is due to its high latency overhead, which becomes the determining factor in communication costs and affects the overall performance of the solver negatively with increasing numbers of processors. The embedded schemes have better scalability characteristics due to their modest latency overheads.

Embedded schemes increase both total and maximum communication volume values compared to *CONV*. However, this increase remains relatively low compared to the increase in latency overhead of *CONV*. This is especially true for the embedded schemes *EMB-KLF* and *EMB-KLR* that further utilize mapping heuristics to further reduce the volume.

These speedup values validate that startup costs become more important with increasing numbers of processors and to obtain a good scalability, it is paramount that latency should be considered as a separate optimization objective.

6. Summary and Conclusions

We have presented a novel computational reorganization method that enables point-to-point communications to be performed simultaneously with the collective communications. We realized this by embedding the contents of the point-to-point messages into the messages exchanged in collective communications. This reduces the synchronization overhead of the iterative solver as well as providing an exact value on the number of messages being communicated in the solver. Using Conjugate Gradient to realize our methods, we obtained superior scalability on Cray XE6 and BlueGene/Q up to 1024 and 2048 processors, respectively. The obtained results show that latency is an important metric that determines the parallel running time and scalability of the solver with increasing number of processors.

References

[1] Y. Saad. Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[2] Cevdet Aykanat, Ali Pinar, and Umit V. Catalyurek. Permuting sparse rectangular matrices into block-diagonal form. SIAM J. Sci. Comput., 25:1860–1879, June 2004.

[3] Bruce Hendrickson and Tamara G. Kolda. Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. SIAM J. Sci. Comput., 21(6):2048–2072, December 1999.

[4] Bora Ucar and Cevdet Aykanat. Partitioning sparse matrices for parallel preconditioned iterative methods. SIAM J. Sci. Comput., 29(4):1683–1709, June 2007.

[5] Brendan Vastenhouw and Rob H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Rev., 47:67–95, January 2005.

[6] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience: Research articles. Concurr. Comput. : Pract. Exper., 19(13):1749–1783, September 2007.

[7] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. ACM Trans. Math. Softw., 38(1):1:1–1:25, December 2011.

[8] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. SIAM Journal on Scientific and Statistical Computing, 6(4):865–881, 1985.

[9] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Bell System Tech. J., 49:291–307, 1970.

[10] O. Selvitopi, M. Ozdal, C. Aykanat, A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies, Parallel and Distributed Systems, IEEE Transactions on PP (99) (2014) 1-1. doi:http://dx.doi.org/10.1109/TPDS.2014.2311804.

[11] Umit Catalyurek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst., 10:673–693, July 1999

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. PRACE-2IP: 283 493.