Springer

KSME

Journal of Mechanical Science and Technology 32 (11) (2018) 5315~5323
www.springerlink.com/content/1738-494x(Print)/1976-3824(Online)
DOI 10.1007/s12206-018-1030-y

Performance comparison of various parallel incomplete LU factorization preconditioners for domain decomposition method †

Sungwoo Kang[1], Hyounggwon Choi[2,*], Wanjin Chung[3], Yo-Han Yoo[4] and Jung Yul Yoo[5]

[1]*Powertrain NVH Development Team 1, Hyundai Motor Group, Hwaseong-si, Gyeonggi-do, Korea*
[2]*Department of Mechanical & Automotive Engg, Seoul National University of Science and Technology, Seoul, Korea*
[3]*Department of Mechanical System Design Engg, Seoul National University of Science and Technology, Seoul, Korea*
[4]*Agency for Defence Development, Yuseong P.O. Box 35-4, Daejeon 305-600, Korea*
[5]*School of Mechanical and Aerospace Engg, Seoul National University, Seoul, Korea*

(Manuscript Received May 24, 2018; Revised July 16, 2018; Accepted July 27, 2018)

Abstract

A finite element code is parallelized by vertex-oriented domain decomposition method which utilizes one- or multi-dimensional partitioning in structured mesh and METIS Library in unstructured mesh. For obtaining the domain-decomposed solution, iterative solvers like conjugate gradient method are used. To accelerate the convergence of iterative solvers, parallel incomplete LU factorization preconditioners are employed, and their performances are compared. For the communication between processors, Message Passing Interface Library is used. The speedups of parallel preconditioned iterative solvers are estimated through computing 2- and 3-dimensional Laplace equations. The effects of mesh and partitioning method on the speedup of parallel preconditioners are also examined.

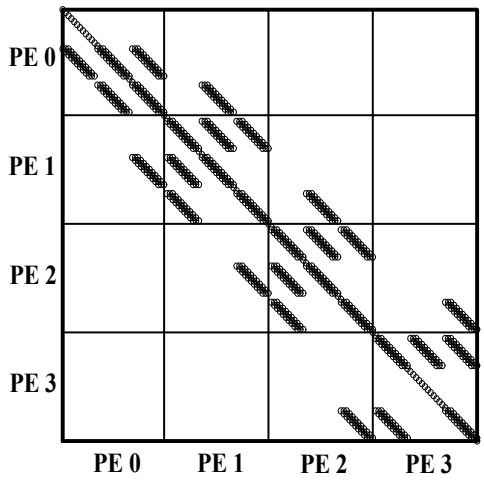
Keywords: Finite element method; Domain decomposition method; Preconditioned conjugate gradient; Parallel ILU preconditioner

1. Introduction

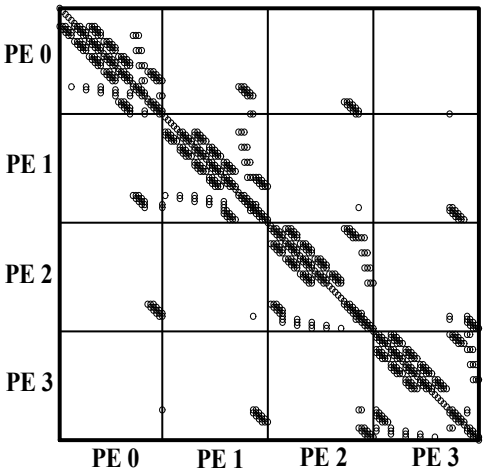
Domain decomposition method has been efficiently used in parallel machines with distributed memory as an analysis tool for governing equations including elliptic-type partial differential equations. In this method, a calculation domain is decomposed into a number of subdomains equal to the number of processors, so that each processor is assigned a subdomain. Each subdomain generates a local matrix which is implicitly coupled with adjacent local matrices as shown for example in Fig. 1. When a domain is decomposed, local matrices are generated from respective processors so that communications between adjacent processors are required to obtain the solution for the entire domain. This is called communication overhead. Besides, in the procedure for solving local matrices with iterative methods like conjugate gradient method (CG), the iteration number increases as the number of subdomains increases. The speedup (for definition, see Eq. (11)) decrease is mainly due to the increase in the iteration number, together with the increase in the communication overhead. Consequently, many efforts have been made to develop an algorithm that minimizes the communication overhead and the increase in the iteration number. In particular, in developing a scalable parallel preconditioned CG algorithm, it is important to design an optimal parallel preconditioner that minimizes these factors.

Basermann et al. [1] used block incomplete LU(0) factorization (block ILU(0)) preconditioner to solve various types of problems. The local matrix assigned to each processor is divided into several block matrices, and ILU(0) preconditioner is applied to each block matrix independently by ignoring elements that are implicitly coupled with other block matrices. To improve the performance of block ILU(0), Magolu monga Made and van der Vorst [2, 3] overlapped local matrices with those of neighboring subdomains, and preconditioned local matrices after increasing fill-in entries in the overlapped parts. They obtained more scalable results for elliptic partial differential equations, as compared with previous block ILU(0). They showed that increasing fill-in entries by extending the overlapped parts of a subdomain was more efficient in improving the performance of block ILU(0) than increasing fill-in entries without extending the overlapped parts. Parallel preconditioner utilizing the Schur complement was proposed by Saad and Sosonkina [4], where only boundary variables of subdomains were used to compose a Schur system so that their exact values could be obtained by solving this reduced system. Using these values as boundary conditions, each processor can deal with the corresponding subdomain completely in parallel. However, solving a Schur system accurately is a

*Corresponding author. Tel.: +82 1066550422
E-mail address: hgchoi@snut.ac.kr
† Recommended by Associate Editor Simon Song



(a) 1-dimensional decomposition



(b) 2-dimensional decomposition

Fig. 1. Non-zero pattern of a domain-decomposed matrix obtained by a finite element discretization for the meshes to be shown in Fig. 3.

time-consuming procedure resulting in more inefficient parallel algorithm than block ILU(0). Hence, they proposed a parallel iterative algorithm that uses GMRES to solve the Schur complement approximately before interior parts of the subdomains are solved. A hybrid parallel solver of sparse linear system was proposed by Manguoglu [5]. This technique contained both direct and iterative solver based on a domain decomposition method for parallel computing and was shown to be both robust and scalable. Lemmer and Hilfer [6] solved a very large scale problem of the flow through porous media consisting of 2048^3 voxels by using a parallel domain decomposition method with non-blocking communication. SIMPLE algorithm was used to solve the Stokes equation. Recently, an additive Schwarz preconditioner for adaptive finite element method was proposed for parallel computation based on domain decomposition method by Loisel and Nguyen [7] They employed a parallel conjugate gradient (CG) method to efficiently solve the global linear system and showed that the convergence rate of the CG method was dependent on the

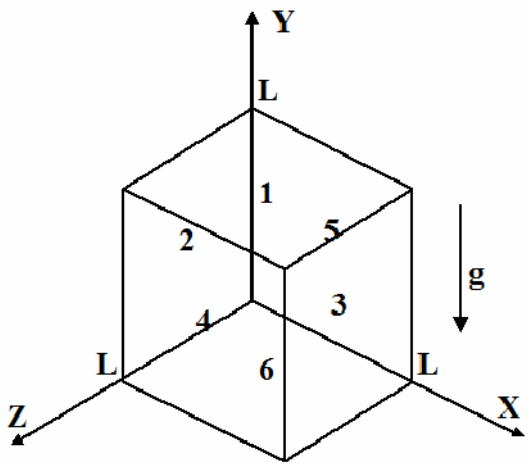


Fig. 2. Computational domain for 3-dimensional problems.

effective conditioner number.

The objective of the present study is to compare the performances of the well-known representative preconditioners such as diagonal preconditioning (DIAG), block ILU(0) without overlapping (BIWO) [8], iterative block ILU(0) (ITBI) and distributed ILU(0) (DILU) [9] by applying them for the solution of an elliptic partial differential equation. In addition, we propose modified distributed ILU(0) (MDLU) to compare its performance with aforementioned parallel preconditioners. For iterative solvers, CG and Bi-CGSTAB proposed by Van der Vorst [10] are used.

2. Test problems

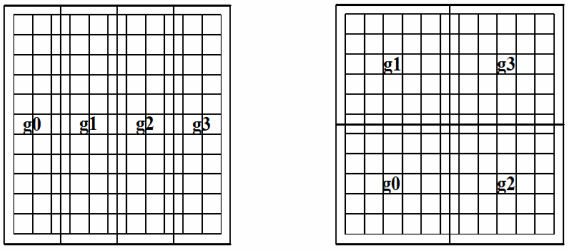
A square is chosen as the computational domain for 2-dimensional problems and a cube is chosen for 3-dimensional problems as shown in Fig. 2. Except for the case of estimating speedup in unstructured mesh, we use structured mesh that has equal number of uniform or non-uniform nodes in each direction. The relative precision of the final residual in each linear system is 10^{-6}. The test problems are run on Cray T3E of Korea Institute of Science and Technology Information (KISTI) that has 128 processors and 16 GB memory, although we utilize up to only 64 of processors. We consider the Laplace equation:

$$\frac{\partial^2 u}{\partial x_j \partial x_j} = 0 \tag{1}$$

where for 2-dimensional problem the boundary conditions are u = 1 at x = 0 and u = 0 at other sides, and for 3-dimensional problem they are u = 1 on face 1 and u = 0 on faces 2 - 6 in Fig. 2. Structured linear finite element mesh and CG are employed for solving the Laplace equation.

3. Parallelization

After a mesh for the whole domain is generated, domain



(a) 1-dimensional decomposition (b) 2-dimensional decomposition

Fig. 3. An illustration of decompositions of a 2-dimensional computational domain into 4 subdomains.

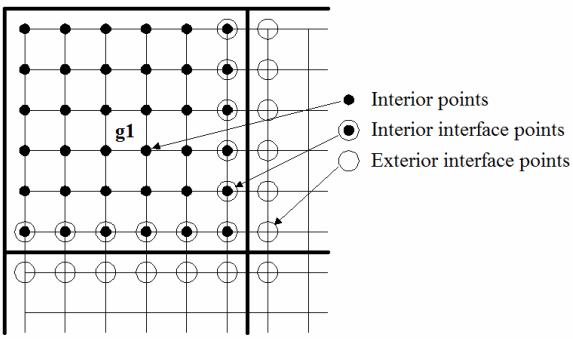


Fig. 4. Classification of cells in a subdomain.

decomposition method is implemented to solve the problem by a parallel machine. Communication between adjacent processors is achieved by Message Passing Interface (MPI) Library [11] and parallel preconditioned iterative solvers are used to obtain the solution for the whole domain.

3.1 Domain decomposition method

For efficient parallelization, we partition the computational domain in such a way that the calculation load of each domain is balanced and the communication load between processors is minimized. When structured finite elements are adopted, 1-dimensional or multi-dimensional domain decomposition is applied. For the case of an unstructured mesh, METIS Library [12] is to be used for mesh partitioning. Fig. 3 shows examples of 1-dimensional and 2-dimensional decomposed computational domains for solving the 2-dimensional Laplace equation (see Subsec. 4.1). A computational domain is decomposed by vertex-oriented domain decomposition method on the basis that each subdomain has the same number of vertices to satisfy the load balance for all processors. Each decomposed subdomain consists of internal nodes, interior and exterior boundary nodes as illustrated in Fig. 4 [13]. After domain decomposition, the nodes are locally renumbered in the order of internal nodes, interior boundary nodes and exterior boundary nodes. Each processor calculates the values of flow variables at internal nodes independently. However, for those at interior and exterior boundary nodes, communications with adjacent processors are needed before the calculation. In each processor, only local values belonging to each subdomain are computed and stored. Therefore, rows for exterior boundary nodes are neglected in constructing the local matrix for each processor. The local matrix $[A_n]$ for solving values at internal and interior boundary nodes in decomposed n^{th} subdomain can be typically constructed as follows:

$$[A_n]\{x_n\} = \{f_n\}$$

$$[A_n] = \begin{pmatrix} B_n & E_n & 0 \\ F_n & C_n & I_n^m \end{pmatrix}, \quad \{x_n\} = \begin{pmatrix} u_n \\ v_n \\ w_n^m \end{pmatrix}, \quad \{f_n\} = \begin{pmatrix} s_n \\ t_n \end{pmatrix}. \tag{2}$$

An example of non-zero patterns of the local matrices $[A_n]$ is illustrated in Fig. 5. It is noted that $\{u_n\}$ and $\{s_n\}$ are components from internal node values, and $\{v_n\}$ and $\{t_n\}$ are components from interior boundary node values. These are from a local subdomain and do not need any communication. But $\{w_n^m\}$ is the part from exterior boundary nodes. Thus, $[B_n]$ is a part of the local matrix generated from internal nodes, $[E_n]$ and $[F_n]$ are linked parts generated from internal and interior boundary nodes, $[C_n]$ is a part generated from interior boundary nodes, and $[I_n^m]$ is a linked part generated from interior and exterior boundary nodes that are communicated with m^{th} subdomain. Since $\{w_n^m\}$ and $[I_n^m]$ in Eq. (2) are linked components with adjacent subdomains, communication with adjacent processors is required to solve this part.

3.2 Preconditioned CG (PCG) algorithm

Serial PCG algorithm is represented by the following sequence of operations with subscript "j" denoting j^{th} step of PCG iteration only in this subsection:

$$\alpha_j = \frac{\{z_j\}^T\{r_j\}}{\{d_j\}^T[A]\{d_j\}} \tag{3}$$

$$\{x_{j+1}\} = \{x_j\} + \alpha_j\{d_j\} \tag{4}$$

$$\{r_{j+1}\} = \{r_j\} - \alpha_j[A]\{d_j\} \tag{5}$$

$$\{z_{j+1}\} = [M]^{-1}\{r_{j+1}\} \tag{6}$$

$$\beta_j = \frac{\{z_{j+1}\}^T\{r_{j+1}\}}{\{z_j\}^T\{r_j\}} \tag{7}$$

$$\{d_{j+1}\} = \{z_{j+1}\} + \beta_j\{d_j\} \tag{8}$$

where $\{r_j\}$ is the residual vector of which the initial value is defined as $\{r_0\}=\{f\}-[A]\{x_0\}$, $\{d_j\}$ is the search direction, $\{z_j\}$ is the preconditioned residual vector, and [M] is the preconditioner matrix. In this work, a parallel conjugate gradient code for P1P1 finite element formulation has been developed and tested with various parallel preconditioned matrices in Eq. (6).

3.3 Parallel preconditioning method

Convergence rate of an iterative solver depends on the con-

dition number of a matrix, and preconditioners accelerate the convergence rate by reducing the condition number. Generally, ILU [14] preconditioners show good convergence rate in serial computations. However, in parallel computations, the communication overhead originates from the intrinsic sequential characteristics of ILU preconditioners. Then, in order to acquire scalable results, one should modify the algorithms of ILU preconditioners so that they reduce the communication overhead at the cost of small increase in the iteration number.

3.3.1 Diagonal preconditioning (DIAG)

The preconditioner matrix for n^{th} subdomain is now defined as $[M_n] = \text{Diag}\begin{pmatrix} B_n & E_n \\ F_n & C_n \end{pmatrix}$ (see Eq. (2)) in diagonal preconditioning. Since they are independent of adjacent subdomains, the computation process for Eq. (6) can be executed independently in each processor. Therefore, for this preconditioner, there are neither additional modifications nor communication overhead due to parallelization.

3.3.2 Block ILU(0) without overlapping (BIWO) [10]

Instead of applying ILU(0) preconditioning to the whole domain, BIWO independently applies it to local subdomains. ILU precondtioner is applied to each local matrix $[A_n]$ excluding $[I_n^m]$ elements that are linked to adjacent processors, i.e. to each $\begin{pmatrix} B_n & E_n \\ F_n & C_n \end{pmatrix}$. By neglecting $[I_n^m]$ elements, there is an advantage that each processor can compute the preconditioner matrix ($[M_n] = [L_n][U_n]$) and execute forward and backward substitution procedures ($[L_n][U_n]\{z\} = \{r\}$) independently. L_n and U_n are lower and upper triangular matrices after incomplete LU(ILU(0)) decomposition of matrix M_n. But the number of ignored elements of the global matrix increases along with the increase of the number of subdomains, which results in the increase of the iteration number of PCG solver.

3.3.3 Iterative block ILU(0) (ITBI)

When computing the preconditioner matrix ($[M_n] = [L_n][U_n]$), ITBI neglects $[I_n^m]$ elements as BIWO does. But, in computing the preconditioned residual vector $\{z\}$, ITBI uses $[I_n^m]$ elements in Eq. (2). For example, processor 1 solves the following equation to obtain $\{z_1\}$ (preconditioned residual vector in subdomain 1) in the case as shown in Fig. 5:

$$[M_1]\{z_1\} = \{r_1\} - [I_1^m]\{z_m\}, \quad (m \neq 1) \tag{9}$$

where $\{z_0\}$ and $\{z_2\}$, which are computed locally in processors 0 and 2, are needed for calculating $[I_1^m]\{z_m\}$ in processor 1. Therefore, a communication procedure is necessary to transfer these values to processor 1. To solve Eq. (9) simultaneously in all the processors, inclusive of coupled components, the following equation instead of Eq. (9) is solved iteratively:

$$[M_1]\{z_1^{p+1}\} = \{r_1\} - [I_1^m]\{z_m^p\}, \quad (m \neq 1) \tag{10}$$

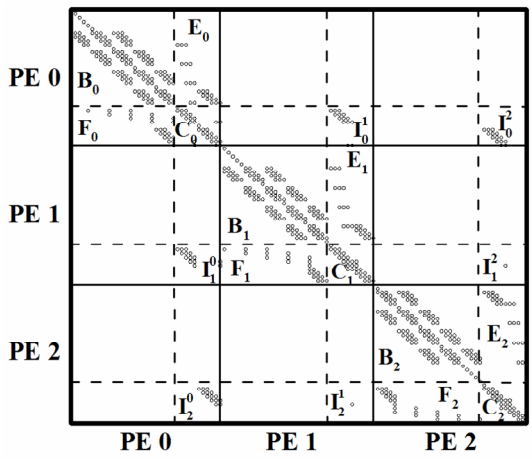


Fig. 5. An illustration of non-zero pattern of the global matrix when 3 processors are used.

where the superscript p denotes the iteration number. In the present ITBI, one more additional iteration is executed for solving Eq. (10), so that the iteration number of CG methods does not increase as much as BIWO, regardless of the increase of the number of subdomains.

3.3.4 Distributed ILU(0) (DILU)

DILU applies ILU(0) to the entire global matrix, which is reconstructed with local matrices of respective local subdomains, without dropping any nonzero entries. It differs from BIWO in that it does not neglect $[I_n^m]$ elements that are linked with exterior boundary nodes, while applying ILU(0). Therefore, while ILU(0) for $[B_n]$ and $[E_n]$ elements are executed independently in each processor "n" (see Eq. (2)), values of [C] and [I] elements that were previously preconditioned by ILU(0) in preceding processors "k" ($k < n$) must be obtained before applying ILU(0) to rows of matrices that include elements from $[C_n]$, $[F_n]$ and $[I_n^m]$. For the case of DILU, communications between processors and waiting times are needed to form a preconditioner matrix ($[M] = [L][U]$). Thus, these are some of the reasons that prevent DILU from being a scalable algorithm.

Once a parallel preconditioner matrix is obtained, Eq. (6) is used to obtain a preconditioned residual vector $\{z\}$. This vector consists of $\{z^{ext}\}$ components that are linked to other subdomains and $\{z^{int}\}$ components that are not linked to other subdomains. The forward substitution procedure in each processor would be summarized as follows [14]:

① $[U_n]\{z_n^{int}\}$ is computed in each subdomain(processor) "n" independently.

② Get $[U_k]\{z_k^{ext}\}$ from linked subdomain "k" ($k < n$).

③ Compute linked interface vector components $[U_n]\{z_n^{ext}\}$.

④ Send $[U_n]\{z_n^{ext}\}$ to linked subdomain "l" ($l > n$).

The backward substitution procedure will be in the order opposite to the forward substitution procedure. This preconditioning method, closest to the serial ILU(0) in that it does not neglect any nonzero elements, does not increase much the

iteration number of CG methods, but has a disadvantage that it is difficult to obtain scalable algorithm due to many non-parallelizable parts in the algorithm and long waiting times.

3.3.5 Modified distributed ILU(0) (MDLU)

In DILU, the efficiency of parallel algorithm is degraded due to the communications with adjacent processors and waiting times during the computation of the preconditioned residual vector at the interior boundary nodes. We now test three algorithms to increase the efficiency of parallel algorithm by reducing the waiting times in DILU.

First, MDLU-I changes the order of computations to reduce the waiting times. In the case of computing forward-substituted values ([U]{z}) at interior boundary nodes, the procedure can be divided into two parts. One consists of computations linked with the preceding processors and the other consists of computations linked with the following processors. Not like DILU, MDLU-I carries out computations linked with the following processors first. As a result, the forward substitution procedure for MDLU-I in processor "n" would be as follows:

(1) Computations linked with processor "l" (l > n) are carried out independently in each processor along with computations at internal nodes and ahead of computations linked with processor "k" (k < n). The non-zero terms related to subdomain "k" are ignored when calculating values at interior boundary nodes linked with processor "l".

(2) Before each processor "n" calculates the rest of forward-substituted values that are linked with processor "k", each processor sends $[U_n]\{z^{ext}_{n-l}\}$ components to processor "l" and receives $[U_k]\{z^{ext}_{n-k}\}$ from processor "k".

Next, to reduce the waiting times, MDLU-II ignores communications between processors linked by a small number of interface nodes. When a structured mesh is used and multi-dimensional partitioning is applied, the number of subdomains to communicate in the preconditioning procedure reduces from 8 to 4 in 2-dimensional case and from 26 to 6 in 3-dimensional case. As MDLU-II neglects some elements of $[I_n^m]$ for reducing the waiting times, the iteration number increases a little.

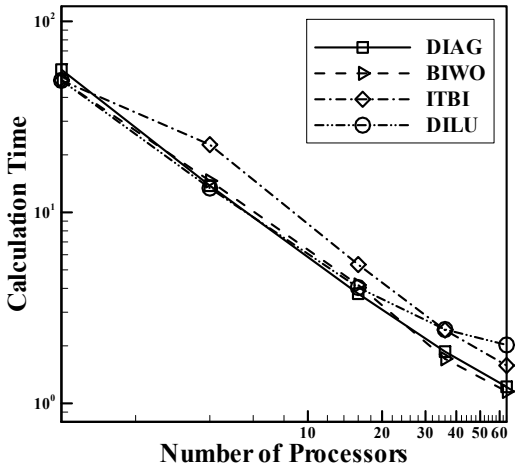
Lastly, MDLU-III is an algorithm that is a combination of MDLU-I and MDLU-II.

4. Results and discussions

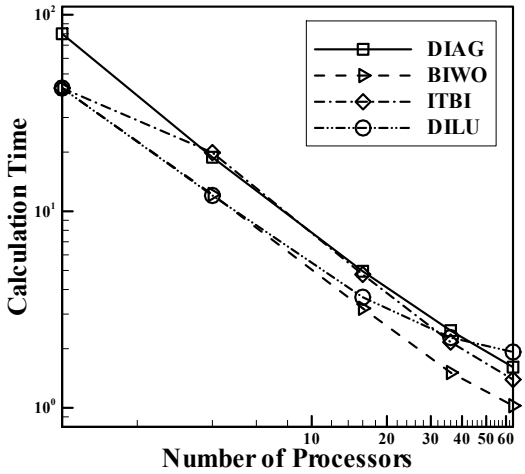
As a measure of parallelization efficiency, we use speedup which is defined as follows:

$$Speedup = \frac{T_1}{T_N} \tag{11}$$

where T_1 is the calculation time using one processor and T_N is the calculation time using N processors. The ideal speedup, which has no additional parallelization overheads, is just the number of processors used. In reality, the speedup decreases due to communications between processors, waiting times for synchronization and additional calculations caused by parallelization.



(a) Uniform mesh



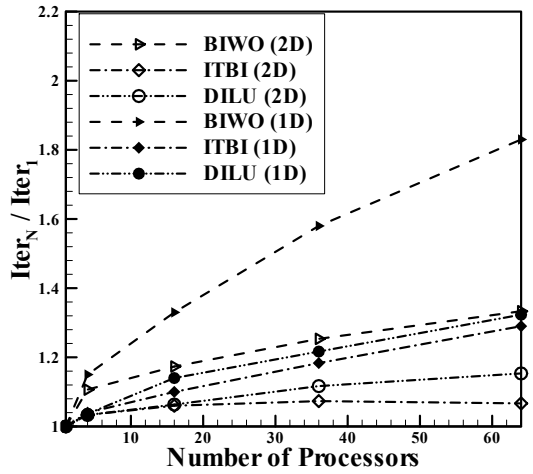
(b) Non-uniform mesh

Fig. 6. Comparison of calculation time for 512 × 512 meshes.

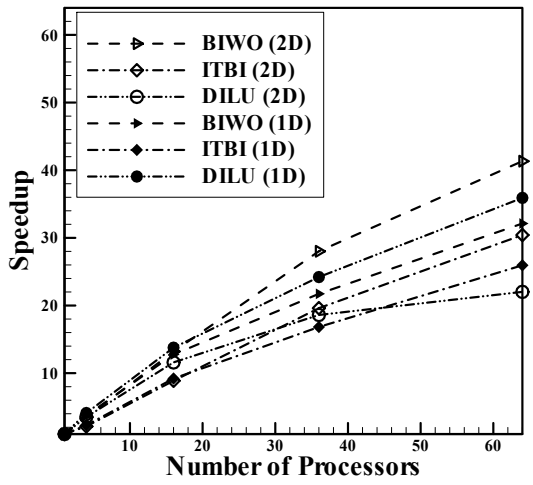
4.1 Efficiency of parallel preconditioners in conjunction with 2-dimensional Laplace equation

4.1.1 Performances in structured mesh

In Fig. 6, calculation times of parallel preconditioners are compared when 512 × 512 structured uniform and non-uniform meshes divided by 2-dimensional partitioning method are used. Non-uniform meshes are generated by the algebraic grid generation techniques suggested by Hoffmann and Chiang [15]. All preconditioners show almost the same performance in the two meshes except that the calculation time of DIAG is larger for non-uniform mesh. The calculation time for non-uniform mesh increases since the matrix from non-uniform mesh is more difficult to solve (due to the increase of matrix condition number with diagonal preconditioning). Due to the increase of waiting time, the decrease rate of calculation

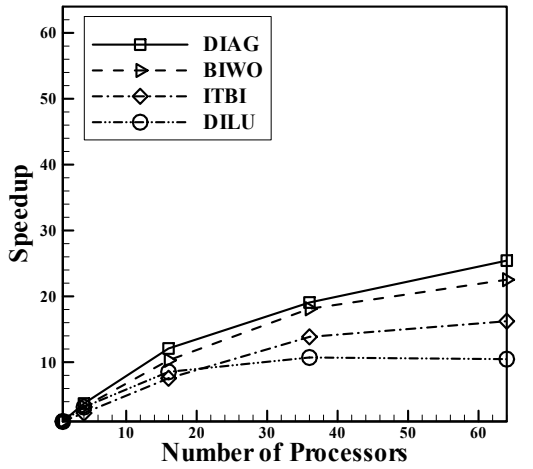


(a) Iteration number ratio

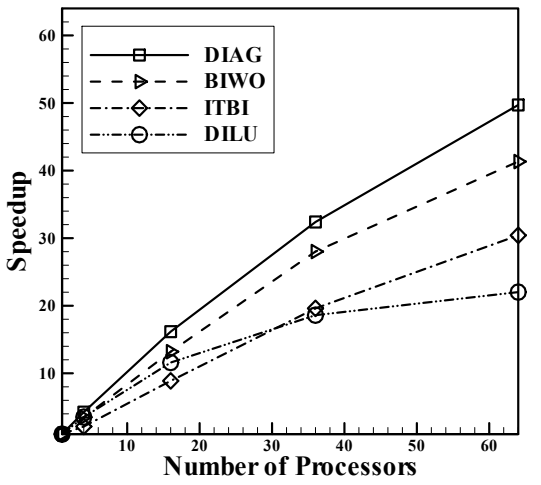


(b) Speedup

Fig. 7. Performance in conjunction with 1-dimensional and 2-dimensional domain decompositions of a 512 × 512 non-uniform mesh.



(a) 256 × 256 mesh



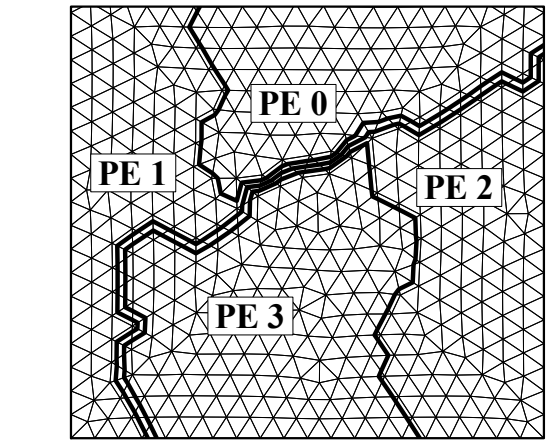
(b) 512 × 512 mesh

Fig. 8. Speedup in non-uniform meshes.

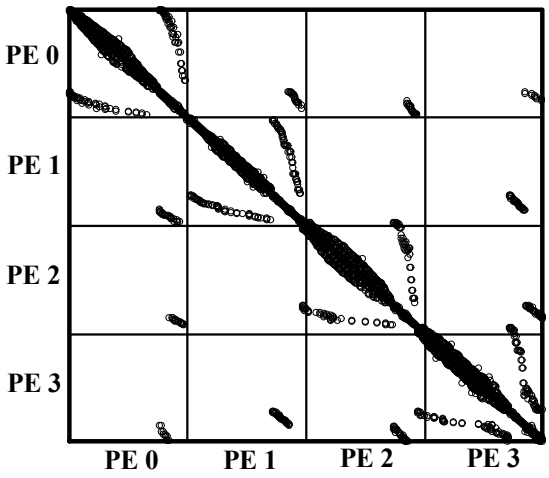
time for DILU is reduced in comparison with other preconditioners, as the number of processors increases.

4.1.2 Effects of domain partitioning methods

The speedup of each preconditioning method differs depending on how a domain is partitioned. This is due to changes in communication overhead, non-zero pattern of the global matrix and neglected terms in the preconditioning. Speedup and increase of the iteration number for 1-dimensional and 2-dimensional partitioning methods in a 512 × 512 non-uniform mesh are shown in Fig. 7. Since DIAG does not have any neglected terms and communication overhead in the preconditioning procedure, the efficiencies of the two partitioning methods are almost similar, so that the results for DIAG are not shown in Fig. 7. For the cases of other preconditioners, the iteration number varies with the virtually reconstructed global matrix as shown in Fig. 1 and with the neglected terms when preconditioners are applied. As shown in Fig. 7(a), 2-dimensional partitioning method shows less increase in the iteration number than 1-dimensional partitioning for all the preconditioning methods since 2-dimensional partitioning has less interface nodes than 1-dimensional partitioning for the same number of processors. Since BIWO neglects more non-zero terms in the preconditioning as the number of interface nodes increases, the iteration number increases greatly when 1-dimensional partitioning is applied. In spite of the same preconditioner matrix as BIWO, ITBI shows less increase in the iteration number because it is suppressed by taking account of the neglected terms in the additional iteration for computing the preconditioned residual vector. Due to larger increase in the iteration numbers in 1-dimensional partitioning, the speedup of 2-dimensional partitioning is larger than 1-dimensional case for BIWO and ITBI preconditionings, as shown in Fig. 7(b). In the case of DILU, the increase in the iteration number is larger for 1-dimensional partitioning, which is similar to other preconditioning methods, but the 1-dimensional speedup shows better results than 2-dimensional case due to larger communication overhead for 2-dimensional

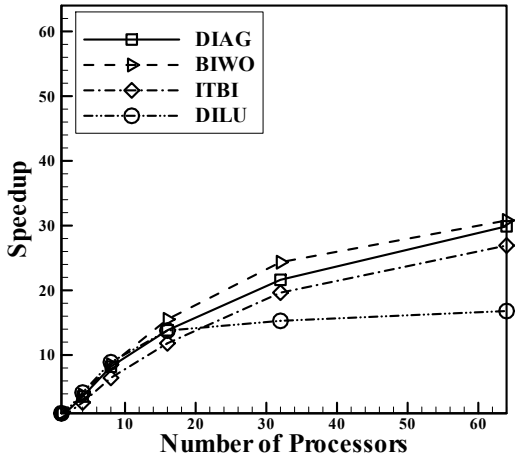


(a) Partitioned mesh

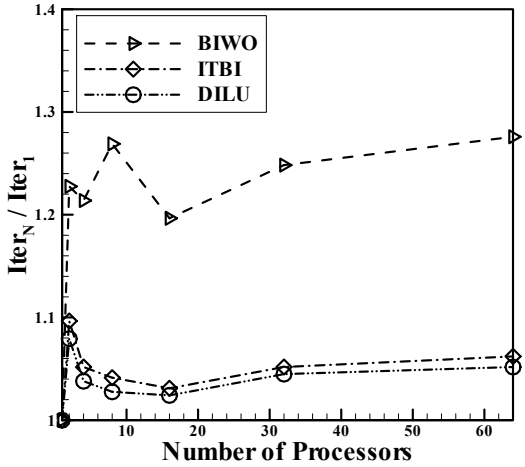


(b) Non-zero pattern of the global matrix

Fig. 9. An illustration of domain decomposition of unstructured mesh.



(a) Speedup



(b) Iteration number ratio

Fig. 10. Performance of unstructured meshes.

case. Since, in the case of DILU, the preconditioner matrices between adjacent processors are linked, a processor has to wait for its turn to calculate the linked parts until the respective parts are sent from the preceding processors. But in 1-dimensional case, as the two interfaces are not linked to each other, a processor can calculate the parts linked with the following processor and send the calculated values before it receives the linked parts from the preceding processor.

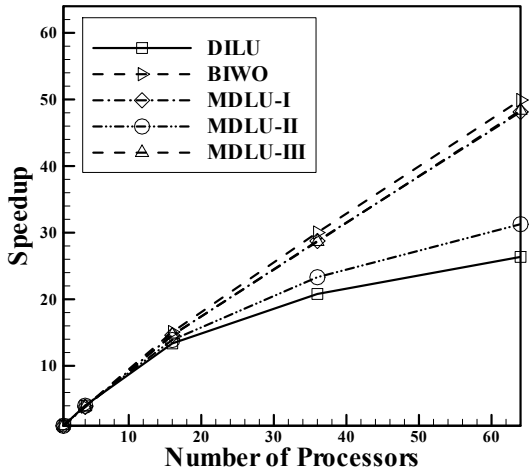
On the other hand, in 2-dimensional case, a processor has to postpone sending the linked parts to the following processors until it receives and calculates the values from all adjacent preceding processors. As a result, the communication overhead for 2-dimensional case increases considerably as the number of processors increases.

4.1.3 Effects of problem size

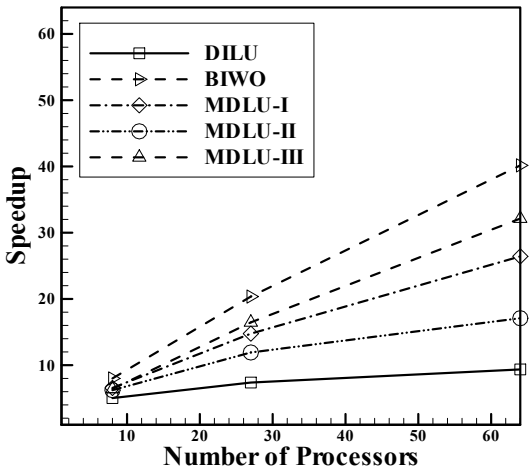
Speedup varies with the size of the problem that is to be solved. Fig. 8 compares the speedup of different problem sizes when non-uniform meshes partitioned by 2-dimensional method are adopted. All preconditioners show better speedup in Fig. 8(b) than in Fig. 8(a). Since the ratio of interface nodes to total nodes in a subdomain decreases as the number of nodes increases, the ratio of communication overhead to calculation load allocated to each processor relatively decreases. Due to this decrease, better speedup is obtained for larger problems.

4.1.4 Performance of unstructured meshes

In unstructured mesh, METIS [12] Library is used to decompose the domain, which pursues computational load balance as well as communication overhead minimization. Fig. 9(a) shows an unstructured mesh in the square calculation domain decomposed by METIS Library. The non-zero pattern of the matrix generated from this decomposed calculation domain is shown in Fig. 9(b), and is similar to that of multi-dimensionally decomposed structured mesh in Fig. 1(b). After the domain decomposition, the nodes are renumbered in the order of internal nodes, interior and exterior boundary nodes in each subdomain. Due to this renumbering, the arrangements of nodes become similar, regardless of using unstructured or structured mesh. As a result, non-zero patterns of the matrices resemble each other.



(a) 2-dimensional Laplace equation with a 512 × 512 mesh



(b) 3-dimensional Laplace equation with a 64 × 64 × 64 mesh

Fig. 11. Comparison of MDLU with DILU and BIWO.

The parallel efficiencies are evaluated in Fig. 10 for the 2-dimensional Laplace equation in an unstructured mesh. The speedup in unstructured mesh resembles that of the structured mesh in Fig. 8(a). However, the increase of the iteration number in Fig. 10(b) has a different trend compared to that of Fig. 7(a). It needs to be noted that the subdomains in an unstructured mesh, which are partitioned by METIS Library, have different shapes depending on the number of the subdomains and on the mesh geometry while in a structured mesh the shapes of the subdomains are similar to each other regardless of the number of processors.

4.2 Performance of MDLU preconditioning method

MDLU is compared with BIWO, which has shown the best speedup for most of the cases considered, in the case of 2-dimensional and 3-dimensional Laplace equations in Fig. 11. A 512 × 512 mesh is used for the 2-dimensional problem and a 64 × 64 × 64 mesh for the 3-dimensional problem. Serial calculation of the 3-dimensional case is excluded due to insufficient memory. BIWO which shows best performance when 8 processors are used, is taken as a reference for evaluating the speedup. From the fact that the speedup of MDLU-I is larger than that of MDLU-II, it can be inferred that the main cause for the degradation of DILU is the waiting time. MDLU-III, the combination of the two algorithms, shows almost the same performance as BIWO for the 2-dimensional cases.

But for the 3-dimensional cases it shows a slight performance degradation compared to BIWO. In the problems we have considered, the iteration numbers are smaller for the 3-dimensional cases than for the 2-dimensional cases, so that the increase of the iteration number was not so large for BIWO. From this result, we can predict that if the MDLU-III is applied to problems that have large iteration number (matrix of large condition number), MDLU-III will at least exhibit the performance of BIWO.

5. Conclusions

In this work, performances of several parallel incomplete LU preconditioners have been compared and examined for the convergence acceleration of parallel iterative solvers, so that the following conclusions are obtained:

(1) The iteration numbers of uniform and non-uniform structured meshes are not much different for most of the preconditioners except DIAG.

(2) The iteration numbers are smaller for multi-dimensional partitioning methods than for 1-dimensional partitioning method and the speedup increases as the problem size becomes larger. However, in the case of DILU, the 1-dimensional speedup shows better results than 2-dimensional case due to larger communication overhead for 2-dimensional case although the increase in the iteration number is larger for 1-dimensional partitioning.

(3) BIWO shows the best performance in the test problems, and MDLU-III seems to be a good alternative for the problems that have large condition numbers.

Acknowledgments

This study was supported by the Research Program funded by the SeoulTech (Seoul National University of Science and Technology).

References

[1] A. Basermann, B. Reichel and C. Schelthoff, Preconditioned CG methods for sparse matrices on massively parallel machines, *Parallel Computing,* 23 (3) (1997) 381-398.

[2] M. Magolu monga Made and H. A. van der Vorst, A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings, *Future Generation Computer Systems,* 17 (8) (2001) 925-932.

[3] M. Magolu monga Made and H. A. van der Vorst, Parallel incomplete factorizations with pseudo-overlapped subdo-

mains, *Parallel Computing,* 27 (8) (2001) 989-1008.
[4] Y. Saad and M. Sosonkina, Distributed Schur complement techniques for general sparse linear systems, *SIAM Journal on Scientific Computing,* 21 (4) (1999) 1337-1356.
[5] M. Manguoglu, A domain-decomposing parallel sparse linear system solver, *Journal of Computational and Applied Mathematics,* 236 (3) (2011) 319-325.
[6] A. Lemmer and R. Hilfer, Parallel domain decomposition method with non-blocking communication for flow through porous media, *Journal of Computational Physics,* 281 (2015) 970-981.
[7] S. Loisel and H. Nguyen, An optimal Schwarz preconditioner for a class of parallel adaptive finite elements, *Journal of Computational and Applied Mathematics,* 321 (2017) 90-107.
[8] G. Radicati di Brozolo and Y. Robert, Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor, *Parallel Computing,* 11 (2) (1989) 223-239.
[9] Y. Saad, *Iterative methods for sparse linear systems,* PWS Publishing Company: Boston (1996).
[10] 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 (2) (1992) 631-644.
[11] M. Snir, S. W. Otto, S. Huss-Lederman, D. Walker and J. Dongarra, *MPI: The complete reference,* The MIT Press: London, England (1996).
[12] *http://www-users.cs.umn.edu/~karypis/metis*.
[13] G. F. Carey, Y. Shen and R. T. McLay, Parallel conjugate gradient performance for least-squares finite elements and transport problems, *International Journal for Numerical Methods in Fluids,* 28 (10) (1998) 1421-1440.
[14] D. S. Kershaw, The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations, *Journal of Computational Physics,* 26 (1) (1978) 43-65.
[15] K. A. Hoffmann and S. T. Chiang, *Computational fluid dynamics for engineers,* A Publication of Engineering Education System: Wichita, Kansas, USA, 1 (1993).

Hyoung Gwon Choi obtained a Ph.D., major in the development of CFD algorithms of finite element method, from Seoul National University, Korea. He is currently a Professor in the Department of Mechanical/Automotive Engineering, Seoul National University of Science and Technology.