PARALLEL PERFORMANCE OF PRECONDITIONED DOMAIN DECOMPOSITION METHOD FOR LARGE SCALE HEAT TRANSFER PROBLEM

*A.M.M. Mukaddes, **Masao Ogino and **Atsushi Uragami
*Department of Industrial and Production Engineering, Shahjalal University of Science and Technology, Japan
**Department of Intelligent Machinery and Systems, Kyushu University, Japan
Email: mukaddes-ipe@sust.edu

ABSTRACT
Domain decomposition is a well known parallel finite element method for distributed parallel computing environment. Incorporation of a suitable preconditioner like balancing domain decomposition (BDD) method makes the method more efficient due to its excellent convergence rate. Several studies have considered applications of the BDD method to various phenomena and improvement of its convergence rate. However, in applying the BDD method to large-scale problems, it is difficult to solve the coarse problem since the size of the coarse problem increases in proportion to the number of subdomains. In this work, domain decomposition technique with an incomplete balancing preconditioner has been adapted to the distributed parallel environment of networked workstations and the supercomputer. Using the developed code, several heat transfer model problems are solved and the parallel performances are analyzed on the network of eight Pentium IV PC cluster and GP7000F workstations. Finally, very large size of problem over 11 million DOFs has been solved using 1024 SR8000 supercomputer.

KEYWORDS: parallel computing, finite element, balancing domain decomposition, domain decomposition, finite element.

1. INTRODUCTION
Large scale problem, particularly those defined in three dimensional domain, often need substantial computation time and memory to run on ordinary sequential computers. Even if they can be solved, powerful computation capability is required to obtain the accurate and precise results within a reasonable time. Parallel computing can meet requirements of high performance computation [1]. Among the various parallel computing algorithms, domain decomposition method has become much popularity in the recent years [2]. In this method, the whole domain to be solved is first decomposed into a number of subdomains without overlapping. Finite Element Analysis (FEA) is defined and solved in each subdomain in parallel and then partial solutions are assembled together to get the global solution.

This method is exclusively implemented on structural analysis [3], heat conduction problem [4] and fluid analysis [5]. Most of the papers emphasizes only on the implementation of the method without analyzing the parallel performance as well as taking the necessary advantages of the parallel computing environment. Again incorporation of a suitable preconditioner like balancing domain decomposition method (BDD) [6] makes the method more efficient due to its excellent convergence rate. The BDD method includes the Neumann-Neumann preconditioner and a coarse grid correction. Several studies have considered applications of the BDD method to various phenomena and improvement of its convergence rate. However, in applying the BDD method to large-scale problems, it is difficult to solve the coarse problem since the size of the coarse problem increases in proportion to the number of subdomains.

In this research, the two parallel approaches (dynamic load distribution [7] and static load distribution) of domain decomposition method with an incomplete balancing preconditioner [8] have been adapted to the distributed parallel environment of networked workstations [9] and the supercomputer.

Using the developed code several heat transfer models are solved and the performances are measured in different computing environments. The important factors affecting the performance of parallel computing using the domain decomposition method are found and analyzed. This analysis will be useful as a guideline for the user of the domain decomposition based software in the parallel computing environment.

2. DOMAIN DECOMPOSITION METHOD
Main method:
Consider a linear system of algebraic equations,

\[ [K][u] = \{f\} \]  

arising from a finite element discretization of a linear elliptic boundary value problem in the domain \( \Omega \), where \( K \) is the global stiffness matrix, \( u \) is an unknown vector and \( f \) is a known vector. The domain \( \Omega \) to be solved is first decomposed into \( N \) number of subdomains, \( \{\Omega^{(i)}\}_{i=1,...,N} \), for which the union of all subdomains boundaries is

\[ \Gamma = \bigcup_{i=1}^{N} \Gamma^{(i)}. \]
Let $K^{(i)}$ be the local stiffness matrix corresponding to subdomain $\Omega^{(i)}$. Then as usual the global stiffness matrix $K$ can be generated by subassembling:

$$K = \sum_{i=1}^{N} R^{(i)T} K^{(i)} R^{(i)T}$$

(3)

where $R^{(i)T}$ is the 0-1 matrix which translates the global indices of the nodes into local numbering. Let $u^{(i)}$ be the vector corresponding to the variables in $\Omega^{(i)}$ and it can be expressed as $u^{(i)} = R^{(i)T} u$. Each $u^{(i)}$ is split into degrees of freedom $u_B^{(i)}$, which correspond to $\partial \Omega^{(i)} \setminus \Gamma_u$, called interface degrees of freedom and the remaining interior degrees of freedom $u_I^{(i)}$. The subdomain stiffness matrices $K^{(i)}$, the vector $u^{(i)}$ and the 0-1 matrices $R^{(i)T}$ are then split accordingly:

$$K^{(i)} = \begin{pmatrix} K_{II}^{(i)} & K_{IB}^{(i)} \\ K_{BI}^{(i)} & K_{BB}^{(i)} \end{pmatrix}$$

(4)

$$u^{(i)} = \begin{pmatrix} u_I^{(i)} \\ u_B^{(i)} \end{pmatrix}$$

(5)

and

$$R^{(i)T} = \begin{pmatrix} R_I^{(i)T} & 0 \\ 0 & R_B^{(i)T} \end{pmatrix}.$$  

(6)

$u_B^{(i)} = R_B^{(i)T} u_B,$ $i = 1,...,N$  

(7)

The unknown in the interior of the subdomain $\Omega^{(i)}$ is eliminated by Gaussian elimination according to the equation (8) using an initial value of $u_B^{(i)}$,

$$u_I^{(i)} = K_{II}^{(i)-1} \left( f_I^{(i)} - K_{IB}^{(i)} u_B^{(i)} \right).$$

(8)

Then, after elimination of the interior degrees of freedom, the problem (1) reduces to a problem on the interface,

$$S u_B = g$$

(9)

where $S$ is the Schur complement matrix:

$$S = \sum_{i=1}^{N} R_B^{(i)T} S^{(i)} R_B^{(i)T}.$$  

(10)

$$S^{(i)} = K_{BB}^{(i)} - K_{IB}^{(i)T} \left( K_{II}^{(i)} \right)^{-1} K_{IB}^{(i)}. $$

(11)

The problem (9) is solved using the preconditioned conjugate gradient method which requires the solution of the following auxiliary problem in each iteration.

$$M z = r$$

(12)

where $M$ is a preconditioning matrix, $z$ is a preconditioned vector and $r$ is the residual of the interface problem.

**Preconditioners [6, 8]:**

The present domain decomposition method includes the following preconditioning techniques [8]

**Simplified diagonal scaling:**

$$M_{}\text{DIAG} = \sum_{i=1}^{N} R_B^{(i)} \left( \text{diag}(K_{BB}^{(i)}) \right)^{-1} R_B^{(i)T}$$

(13)

**Balancing domain decomposition (BDD):**

This preconditioning matrix is defined as

$$M_{BDD} = Q_c + (I - Q_c S) Q_c (I - SQ_c)$$

(14)

where $Q_c$ is the local level part and $Q_c$ is the coarse level part of the preconditioner.

**Balancing domain decomposition with diagonal scaling:**

$$M_{BDD-DIAG}^{-1} = (I - Q_c S) Q_{\text{DIAG}}$$

(15)

**Incomplete balancing domain decomposition (IBDD):**

$$M_{IBDD}^{-1} = \tilde{Q}_c + (\tilde{I} - \tilde{Q}_c S) Q_c (I - S\tilde{Q}_c)$$

(16)

where $\tilde{Q}_c$ is constructed from the incomplete factorized coarse operator.

**Incomplete balancing domain decomposition with simplified diagonal scaling (IBDD-DIAG):**

$$M_{IBDD-DIAG}^{-1} = \tilde{Q}_c + (\tilde{I} - \tilde{Q}_c S) Q_{\text{DIAG}} (I - S\tilde{Q}_c)$$

(17)

3. PARALLELISM OF THE DOMAIN DECOMPOSITION METHOD

Constructing the DDM algorithms for parallel computers, a good principle is to divide the original domain into parts, which are further decomposed into smaller subdomains. In this research we adopt Hierarchical Domain Decomposition Method
HDDM) [6] which is a well known parallel DDM. This hierarchically structured DDM classifies processors in two different following ways.

**Dynamic load distribution (h-mode):**

The h-mode classifies processors into 3 groups, ‘Grand Parent’, ‘Parent’ and ‘Child’. Fig. 1 illustrates the practical implementation of hierarchical organized processors in the present method. The role of Grand Parent is to organize all processor communications (i.e. message passing) which occur between all processors. Parents prepare mesh data, manage FEA (Finite Element Analysis) results, and coordinate the CG iteration, including convergence decision for the CG iteration. Parents send data to Child processors, where FEA is performed in parallel. After the FEA, Child processors send the results to Parents. This computation will be repeated until the CG iteration is convergent.

**Static load distribution (p-mode):**

However, because almost all computation is performed in Child processors and the most communication time is taken between Parent processors and Child processors, so the communication speed becomes important. In recent years although the communication performance has also been improving by improvement in network technology, the high-speed network is still expensive. On the other hand, for the PC cluster generally used, the network speed becomes a bottleneck to the processing performance of CPU. Moreover, when a parallel processing performance is considered, it is important to reduce the amount of communication time as much as possible. Therefore, the Parent-Only type (static load distribution: p-mode) is useful than the conventional Grand-Parent-Child type (dynamic load distribution: h-mode).

Fig. 4 shows the parallel processor scheme of p-mode. In the p-mode, the Parent processors perform the FEA by themselves, which is computed by the Child processors in the h-mode. In the h-mode, although Parent processors store some of the subdomain analysis data and coordinate the CG iteration as the main work, the idling time of CPU increases because of less computation in Parent processors.

### 4. PARALLEL PERFORMANCE

**Speed up:**

The key issue in the parallel processing in a single application is the speed up achieved, specially its dependence on the number of processors used. For simplicity speed up with $n$ processors, $S_n$ is defined as follows:

$$S_n = \frac{t_1(A)}{t_n(A)}$$

where $t_1(A)$ and $t_n(A)$ are total time for solving the problem using one processor and $n$ processors, respectively. Also scaled speed up $S'_n$ is defined as follows:

$$S'_n = \frac{n \times t_1(A)}{t_n(nA)}$$

where $t_n(nA)$ is the total time for solving the problem of size $nA$.

**Sequential processing and parallel processing:**

The program might be sequential, parallel or combination of sequential and parallel. The execution time to solve a problem of size $n$ having the sequential processing is given by the equation (20)

$$T(n) = T_{\text{calc}}(n) + T_{i/o}(n)$$

where $T(n)$, $T_{\text{calc}}(n)$, and $T_{i/o}(n)$ are the total execution time, calculation time and input/output time. Again the execution time to solve a problem of size $n$, having the parallel processing also is given by the equation (21)

$$T(n, p) = T_{\text{calc}}(n, p) + T_{i/o}(n, p) + T_{\text{comm}}(n, p)$$

where $T(n, p)$, $T_{\text{calc}}(n, p)$, $T_{i/o}(n, p)$ and $T_{\text{comm}}(n, p)$ are the execution time, calculation time input/output time and time for communication respectively.

The ratio of the calculation, input/output and communication time over the total execution time is defined as follows

$$R_{\text{calc}} = \frac{T_{\text{calc}}(n, p)}{T(n, p)}, \quad R_{i/o} = \frac{T_{i/o}(n, p)}{T(n, p)}, \quad R_{\text{comm}} = \frac{T_{\text{comm}}(n, p)}{T(n, p)}$$

and

$$R_{\text{calc}} + R_{i/o} + R_{\text{comm}} = 1$$

### 5. NUMERICAL RESULTS AND DISCUSSIONS

**A. Physical model and computational environment:**

Using the developed code, two heat transfer models [4] are analyzed here. Models are High Temperature Test Reactor (HTTR) and Advanced Boiling Water Reactor (ABWR) which are shown in fig. 2. The calculations are performed in three different computational environments shown in Table 1.

**B. Scalability and speed-up:**

...
Fig. 1 Hierarchical domain decomposition method (left), dynamic load distribution (right)

Fig. 2 Left- Parallel processor scheme (p-mode), middle- HTGR and ABWR , right- computation time in different process

Fig. 3 Speed up (left) and scaled speed-up in different number of processors for HTGR model.

Computation time (sec)

Fig. 4 Computation time in two different computational environment
Fig. 5. Ratio of calculation, input/output and communication time (p-mode)

Computation time (sec)

Fig. 6. Execution time in two different computational environments (h-mode)

Computation time (sec)

Fig. 7. Computation time vs number of subdomains in different number of processors
We evaluate the parallel scalability of the BDD and IBDD-DIAG preconditioner, varying the number of processors employed. Such evaluation is performed using the HITACHI SR8000. The results are shown in Fig.2 (right). Fig. 2 (right) reports that BDD and IBDD-DIAG can compute faster solutions of a fixed mesh problem when the number of processors is increased. The speed-up and scaled speed-up are both compared with the number of processors, referring the value for using 8 processors in Fig. 3. The results shown in figures confirm the parallel scalability properties of the BDD and IBDD-DIAG preconditioners. Since the program has both parallelizable and nonparallelizable portion, results are too far from ideal one. For the parallelizable portion, parallel computing with \( n \) processors can shorten computation time to \( 1/n \) in ideal.

C. Results concerning the number of subdomains:

The number of subdomains is the major concern in the domain decomposition method. We compare the computation time (for IBDD-DIAG) in two different computational environments corresponding to the number of subdomains and results are shown in Fig. 4. CM shows better performance than GP7000F. The ratio of the calculation time, time for input/output data and communication time corresponding to the number of subdomains are shown in the Fig. 5. With the increase of number of subdomains, ratio of calculation time decreases while ratio of input/output time and communication time increases. Since the network speed of GP7000F is faster than the CM, the communication takes less time compared to the CM clusters. The Fig. 6 shows the computation time for h-mode in two different computation environments, which shows that when the number of subdomain is 3000 the computation time increases in CM cluster compared to the GP7000F. This is because, in the h-mode the communication takes place between the Parents and Childs as the number of subdomains increases thereby in CM it takes more time for the communication. So in the parallel computing environment having low performance it is better to use the static load distribution mode and having high performance it is better to use the dynamic load distribution mode.

ABWR model is analyzed in the vector type supercomputer SR8000 changing number of processors. Fig. 7. shows the variation of the computation time using the different number of processors in SR8000. As the number processor increases, the difference decreases. As a result, the difference between the time required using 512 processors and 1024 processors is too small. So it is not necessary to use more than 1024 processors for this model.

6. CONCLUSION

Two parallel approaches, static load distribution and dynamic load distribution of the domain decomposition method are adapted to the parallel computing environment and their parallel performances are analyzed. Static load distribution seems to be more efficient in the parallel computer having low the network speed while dynamic load distribution shows good performance in the high speed networked parallel computers. By the performance test, the effectiveness of the domain decomposition method in the parallel computer is verified. Important factors affecting the performance of the distributed parallel computing are found and analyzed. By using workstation cluster, a huge size of problem having over 11 million DOFs is solved successfully.

7. REFERENCES


