Journal of Computational Science and Technology

Vol.6, No.3, 2012

CG-Based Subdomain Local Solver with ICT Factorization Preconditioner for Domain Decomposition Method*

Yasunori YUSA**, Satsuki MINAMI**, Hiroshi KAWAI***
and Shinobu YOSHIMURA**

** Department of Systems Innovation, School of Engineering, The University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113–8656, Japan

*** Department of Mechanical Systems Engineering, Faculty of Systems Engineering,
Tokyo University of Science, Suwa
5000–1 Toyohira, Chino-shi, Nagano 391–0292, Japan
E-mail: yusa@save.sys.t.u-tokyo.ac.jp

Abstract

To analyze large-scale problems by a domain decomposition method (DDM), it is important to accelerate the subdomain local solver. For utilizing cache memory effectively and for saving main memory usage, we employ the preconditioner of incomplete Cholesky factorization with threshold (ICT) optimized for the subdomain local solver of the DDM. Though the ICT preconditioner was originally proposed for ill-conditioned problems, we employ it in this study because it can freely control the number of nonzeros of the preconditioning matrix. By controlling the number of nonzeros, both the coefficient and the preconditioning matrices can fit on the cache memory. By using the cache memory effectively, the computation time of the ICT-based subdomain local solver becomes comparable to that of the direct LDL-based solver. In addition, when the number of degrees of freedom (DOFs) of an analysis model becomes very large, the LDL-based DDM solver suffers from overflow of the main memory whereas the ICT-based solver can complete the analysis. Using this solver, we succeeded in analyzing a structural problem of 64 million DOFs in 8 minutes on a parallel computing cluster of 8 nodes.

Key words : Finite Element Structural Analysis, Parallel Processing, Domain Decomposition Method, Subdomain Local Solver, Conjugate Gradient Method, Preconditioners, Incomplete Cholesky Factorization with Threshold

1. Introduction

In the field of large-scale finite element analysis, parallel computation using a domain decomposition method (DDM)[1][2] has been studied. In a DDM, a whole analysis domain is first decomposed into multiple non-overlapping subdomains and is then solved by repeatedly analyzing a number of subdomain local problems. Each subdomain is solved on one processor core, and the subdomain local solver is one of hot-spots in DDM-based software. In recent scalar computers with deep hierarchical memory systems, since the memory byte per flop has been smaller and smaller every year, the cache memory utilization becomes essential for high-performance computing. The aim of this paper is to propose a memory-saving iterative solution technique to accelerate the subdomain local solver of the DDM. In addition, since main memory usage becomes a key factor in solving very large-scale problems. In a real situation, main memory overflow often occurs. Therefore, we save main memory usage and analyze large-scale problems on a computer which has a limited amount of main memory.

In the cases of a DDM with preconditioners such as balancing domain decomposition (BDD)[3][4], the cost of coarse grid correction (CGC) cannot be ignored, whose operation

*Received 21 Feb., 2012 (No. 12-0121)
[DOI: 10.1299/jcst.6.157]

count is directly related to the number of subdomains. The total number of subdomains must be, thus, restricted under a certain number. For example, when the total number of DOFs of a whole analysis domain is 200 millions and the total number of subdomains is restricted to 100 thousands, the number of DOFs of each subdomain would become approximately 2,000. In the BDD-based structural solver ADVENTURE_Solid[5] which was optimized for the Earth Simulator by Ogino et al.[6], the DOFs of the subdomains are set as approximately 3,000 for a problem of 100 million DOFs. Yamada et al.[7] proposed 2,700 DOFs to be the optimum for a problem of 48 million DOFs. Considering those studies, we set the target of our study to be 1,000–6,000 DOFs for the subdomains of problems of 100 million–1 billion DOFs.

Direct solution methods have been usually employed as the subdomain local solvers of a DDM. In such cases, the DOFs of each subdomain are typically set as a few hundreds. Iterative solution methods have seldom been utilized for the subdomain local solver of a DDM because of their large operation count. In addition, a matrix factorization in the direct solver is conducted once on each subdomain, and then, only forward and backward substitutions are conducted for solving linear equations. However, in DDM-based electromagnetic analyses, the incomplete Cholesky conjugate orthogonal conjugate gradient (ICCOCG) method has been used. Kanayama and Sugimoto[8] used an ICCOCG-based subdomain local solver and analyzed eddy currents of 5.55 million complex DOFs. Takei et al.[9] analyzed a full-wave electromagnetic field of 12.8 million complex DOFs. Kawai et al.[10] used conjugate gradient (CG) solvers with the preconditioners of diagonal scaling and symmetric successive over-relaxation (SSOR) for structural analyses.

In this study, the target is a three-dimensional solid finite element linear elastic mechanics problem. Due to the symmetric positive definite coefficient matrix of the problem, we chose a preconditioned conjugate gradient (PCG) method for the subdomain local solver. Although memory-saving PCG solvers are usually slower than the skyline-based direct LDL solver, they are able to become comparable by utilizing the cache memory. To accelerate CG convergence, we also chose incomplete Cholesky (IC) preconditioners which are usually used in the field of structural analysis. Various versions of IC and incomplete LU (ILU) which is advantageous for asymmetric matrices are available. ILU with threshold (ILUT) by Saad[11] ignores small nonzeros by tolerance parameters. ILUTP[11] uses a pivoting technique to avoid zero diagonal entries and to stabilize convergence. Robust IC (RIC) by Ajiz and Jennings[12] modifies diagonal entries to avoid the failure of IC factorization. Quasi RIC by Kakihara and Fujino[13] uses a relaxation parameter to modify diagonal entries in order to accelerate convergence. In addition to the IC preconditioners, approximate inverse (AINV) and stabilized AINV (SAINV) by Benzi et al.[14], and robust incomplete factorization (RIF) by Benzi and Tuma[15] are being actively studied.

In this study, since the local solver is activated on one processor core, it is not required to parallelize preconditioning. Though robust preconditioners and approximate inverse preconditioners are effective for ill-conditioned problems such as a shell or a beam, the condition number of our coefficient matrices is not very large, because the subdomains after domain decomposition are bulky-shaped. Therefore, we adopt the relatively simple ILUT (incomplete Cholesky factorization with threshold (ICT) for a symmetric system) preconditioner optimized for the subdomain local solver of a DDM. Although the ILUT preconditioner was originally proposed for ill-conditioned problems, we adopted the incomplete Cholesky factorization with threshold (ICT) preconditioned CG method in this study for two reasons. First, the PCG solver is generally more efficient in main memory usage than is the direct solver. Using the memory-saving solvers, one can analyze large-scale problems on a computer having a limited amount of main memory. Second, the number of nonzeros (memory usage) of the ICT preconditioning matrix can be freely controlled by a tolerance parameter. Though the tolerance-based nonzero dropping is conducted during the incomplete Cholesky factorization in the original algorithm, the dropping is conducted after the complete factorization in our modified algorithm. It is possible to make an ICT preconditioning matrix which fits on the

cache memory. Cache memory utilization is an important factor to accelerate the iterative solvers in multi-core/many core environments.

2. Domain Decomposition Method

2.1. Overview

In DDMs, the analysis domain is first decomposed into multiple non-overlapping subdomains. The global simultaneous linear equation for the interface DOFs of the subdomains is solved by iterative solution methods. The local simultaneous linear equations of the subdomains are solved in every DDM iteration step. Since our target is a linear elastic problem, the coefficient matrices of the local simultaneous linear equations become symmetric and positive definite. Therefore, we employ the PCG method for the subdomain local solution. We adopt the CG method with the preconditioner of ICT for solving the local equations, because it is possible to save memory by dropping small nonzeros of the preconditioning matrix.

We next explain the basic algorithm of the DDM and state the roles and features of the subdomain local solver of the DDM.

2.2. DDM Algorithm

The global simultaneous linear equation

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \tag{1}$$

is derived from the finite element discretization of a linear elastic body of infinitesimal deformation. $\boldsymbol{K}$ is a stiffness matrix (a coefficient matrix), $\boldsymbol{u}$ is a nodal displacement vector (an unknown vector) and $\boldsymbol{f}$ is a nodal load vector (a right-hand side; RHS). In DDMs, an analysis domain Ω is decomposed into non-overlapping subdomains $\Omega_1, \ldots, \Omega_N$ as shown in Fig. 1. Then, the coefficient matrix $\boldsymbol{K}_i$, the unknown vector $\boldsymbol{u}_i$ and the RHS vector $\boldsymbol{f}_i$ of subdomain i are partitioned into the internal DOFs (I) of Ω_i and the interface DOFs (B) of Γ_i, as written in the following:

$$\boldsymbol{K}_i = \begin{bmatrix} \boldsymbol{K}_{IIi} & \boldsymbol{K}_{IBi} \\ \boldsymbol{K}_{IBi}^{\mathrm{T}} & \boldsymbol{K}_{BBi} \end{bmatrix}, \tag{2}$$

$$\boldsymbol{u}_i = \begin{Bmatrix} \boldsymbol{u}_{Ii} \\ \boldsymbol{u}_{Bi} \end{Bmatrix}, \tag{3}$$

$$\boldsymbol{f}_i = \begin{Bmatrix} \boldsymbol{f}_{Ii} \\ \boldsymbol{f}_{Bi} \end{Bmatrix}. \tag{4}$$

After eliminating the internal DOFs of Ω_i by static condensation, Eq. (2) becomes

$$\boldsymbol{S}_i\boldsymbol{u}_{Bi} = \boldsymbol{g}_i \tag{5}$$

with respect to the interface DOFs of Γ_i. The coefficient matrix $\boldsymbol{S}_i$ and the unknown vector $\boldsymbol{g}_i$ are represented by

$$\boldsymbol{S}_i = \boldsymbol{K}_{BBi} - \boldsymbol{K}_{BBi}^{\mathrm{T}}\boldsymbol{K}_{IIi}^{-1}\boldsymbol{K}_{IBi}, \tag{6}$$

$$\boldsymbol{g}_i = \boldsymbol{f}_{Bi} - \boldsymbol{K}_{BBi}^{\mathrm{T}}\boldsymbol{K}_{IIi}^{-1}\boldsymbol{f}_{Ii}. \tag{7}$$

By superposing the equations from $i = 1$ to $i = N$, the following global simultaneous linear equation can be obtained.

$$\boldsymbol{S}\boldsymbol{u}_B = \boldsymbol{g} \tag{8}$$

where, $\boldsymbol{S}$ and $\boldsymbol{g}$ are represented by

$$\boldsymbol{S} = \sum_{i=1}^{N} \boldsymbol{R}_{Bi}^{\mathrm{T}}\boldsymbol{S}_i\boldsymbol{R}_{Bi}, \tag{9}$$

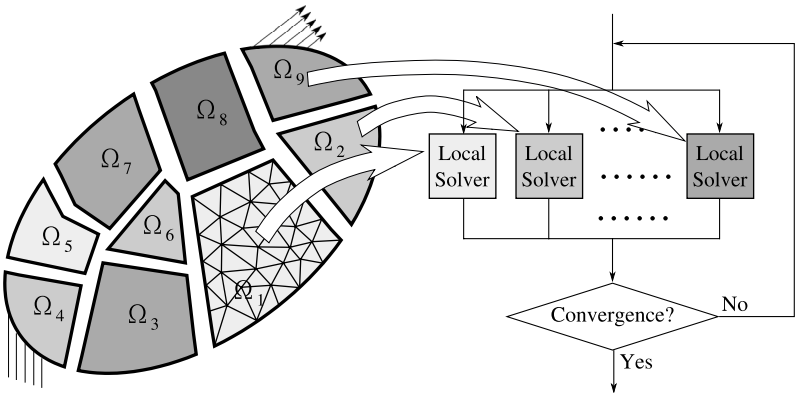

Fig. 1 Decomposed domain and flowchart of the domain decomposition method (DDM) solver.

$$\boldsymbol{g} = \sum_{i=1}^{N} \boldsymbol{R}_{Bi}^{\mathrm{T}} \boldsymbol{g}_i. \tag{10}$$

$\boldsymbol{R}_{Bi}$ is a restriction operator matrix which has entries of 0 or 1 and which restricts the DOFs. $\boldsymbol{S}$ is called the Schur complement matrix and it is symmetric positive definite in linear elastic problems. In the DDM, the global problem of Eq. (8) is solved by iterative solution methods.

When solving Eq. (8) with PCG methods, it is difficult to generate $\boldsymbol{S}$ directly by Eq. (8) and to store it in memory. Thus, the matrix-vector multiplication of $\boldsymbol{q} = \boldsymbol{S}\boldsymbol{p}$ in the PCG iterations is generally calculated by the following steps. First, $\boldsymbol{q}_{Bi}$ is calculated subdomain-wise by

$$\begin{Bmatrix} \boldsymbol{h}_{Ii} \\ \boldsymbol{h}_{Bi} \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}_{IIi} & \boldsymbol{K}_{IBi} \\ \boldsymbol{K}_{IBi}^{\mathrm{T}} & \boldsymbol{K}_{BBi} \end{bmatrix} \begin{Bmatrix} \boldsymbol{0} \\ -\boldsymbol{R}_{Bi}\boldsymbol{p} \end{Bmatrix}, \tag{11}$$

$$\begin{Bmatrix} \boldsymbol{v}_{Ii} \\ \boldsymbol{v}_{Bi} \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}_{IIi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}^{-1} \begin{Bmatrix} \boldsymbol{h}_{Ii} \\ \boldsymbol{h}_{Bi} \end{Bmatrix}, \tag{12}$$

$$\begin{Bmatrix} \boldsymbol{q}_{Ii} \\ \boldsymbol{q}_{Bi} \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}_{IIi} & \boldsymbol{K}_{IBi} \\ \boldsymbol{K}_{IBi}^{\mathrm{T}} & \boldsymbol{K}_{BBi} \end{bmatrix} \begin{Bmatrix} \boldsymbol{v}_{Ii} \\ \boldsymbol{R}_{Bi}\boldsymbol{p} \end{Bmatrix} \tag{13}$$

in parallel, as in the flowchart of Fig. 1. These three equations represent the processing of the enforced displacement boundary conditions on the interface Γ_i, the local finite element analysis concerning the internal DOFs of Ω_i, and the calculation of the nodal reaction force on the interface Γ_i. It is one of the hot-spots to solve these equations in the DDM and this solver is called the subdomain local solver. Finally, $\boldsymbol{q}$ is calculated by superposing $\boldsymbol{q}_{Bi}$ as follows:

$$\boldsymbol{q} = \sum_{i=1}^{N} \boldsymbol{R}_{Bi}^{\mathrm{T}} \boldsymbol{q}_{Bi}. \tag{14}$$

The unbalance is calculated by superposing the nodal reaction forces.

2.3. Subdomain Local Solver

In the DDM, an analysis domain is decomposed into subdomains and every subdomain problem of Eq. (12) has to be solved repeatedly. Subdomain local analyses can be performed in parallel and each subdomain problem can be solved on one processor core. When the unbalance of interface converges, the DDM loop stops. It is noticeable that the local coefficient matrices $\boldsymbol{K}_{IIi}$ are constant through the DDM loop. Then, if the direct methods-based subdomain local solvers are selected, the factorized skyline matrices are stored in the main memory in general. If iterative methods such as PCG methods are selected, more powerful but larger-sized preconditioning matrices containing many nonzeros can be stored in the main memory.

3. Incomplete Cholesky Factorization with Threshold

3.1. Overview

To accelerate the subdomain local solver of the DDM, it is essential to make both the coefficient and the preconditioning matrices fit on the cache memory. In this study, we employ the CG method with the preconditioner of ICT. In this section, the ICT implementation optimized for the subdomain local solver of the DDM is explained. In the context of the DDM, we generate an ICT preconditioning matrix that has a higher potential than the original ICT to accelerate the CG convergence.

3.2. ICT Algorithm Modification

For high performance computing on recent hierarchical memory systems, it is important to utilize the cache memory effectively. In detail, it is possible to store both the coefficient matrix and the preconditioning matrix in the last-level cache memory. However, in general, many preconditioners such as IC (0) do not consider the memory usage of the preconditioning matrix, thus their memory usage changes depending on the number of DOFs or the number of nonzeros of the coefficient matrix. In the ICT, small nonzero entries in the matrix are detected by a tolerance parameter and then dropped to zero. In the original implementation of the ICT, the nonzero dropping is conducted on each row in the process of factorizing the matrix. In addition, the original ICT has two tolerance parameters τ and p. In the case of the subdomain local solver of the DDM, the coefficient matrices K_{IIi} of simultaneous linear Eq. (12) are constant through the DDM iterations. Thus, the cost of generating the preconditioning matrices can be ignored. To make the number of PCG iterations smaller, we employ the strategy of dropping nonzeros *after* complete Cholesky factorization. The detailed algorithm of generating the ICT preconditioning matrix is shown in the following.

for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $i - 1$ **do**
    $l_{ij} \leftarrow \left(a_{ij} - \Sigma_{k=0}^{j-1} l_{ik} l_{jk}\right) \Big/ l_{jj}$
  end for
  $l_{ii} \leftarrow \sqrt{a_{ii} - \Sigma_{j=0}^{i-1} l_{ij}^2}$
end for
for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $i - 1$ **do**
    if $|l_{ij}| < \tau$ **then**
      $l_{ij} \leftarrow 0$
    end if
  end for
end for

In this algorithm, a_{ij} is an entry of the coefficient matrix, l_{ij} is an entry of the lower triangular preconditioning matrix, and τ is a tolerance of dropping small nonzero entries of the preconditioning matrix. It should be noted that this implementation has only one tolerance parameter τ.

3.3. Nodal Block ICT

In the previous subsection, nonzero dropping was conducted by point entries, thus the ICT preconditioner in the previous subsection is called a *point* ICT. In three-dimensional solid mechanics problems, the number of DOFs per node is three and the generated coefficient matrix has a 3×3 block structure. A faster well-known register or cache blocking technique stores the nonzero entries of the matrix in memory with the ordering by nodal blocks. In the case of the nodal block ICT, it is also effective to use the register blocking approach. Then, a key factor is how to drop nonzero 3×3 blocks. In this study, the following five strategies are investigated in the next section.

- Maximum entry (maximum norm): $\max |a_{ij}|$
- Minimum entry: $\min |a_{ij}|$
- 1-norm: $\Sigma_{i,j} |a_{ij}|$
- Square of 2-norm (Frobenius norm): $\Sigma_{i,j} a_{ij}^2$
- Determinant: $a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$

4. Numerical Experiments

4.1. Overview

Using a personal computer (PC) with the specification shown in Table 1 and a PC cluster with the specification shown in Table 2, we evaluate the performances of the implemented subdomain local solver. First, benchmarks of the subdomain local solver are measured. Second, benchmarks of the DDM solver that contains the subdomain local solver are measured.

4.2. Number of PCG Iterations of the Subdomain Local Solver

Figure 2 represents a subdomain model for measuring the numbers of iterations of the CG solver with the ICT preconditioner. The shape of the model is set to be similar to a cubic shape and the mesh is generated with natural ordering by quadratic tetrahedral elements.

In general, the boundary condition of the subdomain is constrained with displacement on four of the six faces. This is because the real structures are generally thin-walled and thus the boundary condition on two faces often becomes free. However, since the deformation mode is generally bending, the boundary condition of the subdomain can be modeled by a constraint on one face and shear loading on the opposite face. The material constants are Young's modulus of 210 GPa and Poisson's ratio of 0.3.

Figure 3 represents the measured number of iterations of the point ICT preconditioned CG-based subdomain local solver. The diagonal scaling technique is performed before the PCG loop is called. The tolerance of the relative residual 2-norm in the PCG method is set as 10^{-10}. In the figure, the horizontal axis is the numbers of nonzeros of the preconditioning matrix. The vertical axis is the measured number of PCG iterations. The figure obviously shows that the number of iterations becomes small when the ICT preconditioning matrix contains many nonzero entries. Figure 4 shows the comparison among the ICT, the IC (0) and the diagonal scaling. The DOFs of the model are 4,719. It is noticeable that the IC (0) and the diagonal scaling are represented by points whereas the ICT is represented by a curve. This is because the IC (0) and the diagonal scaling do not consider the memory usage of the preconditioning matrix. As shown in the figure, although the diagonal scaling is a little better than the ICT in the case of a very small number of nonzeros, the ICT is generally more efficient than either the IC (0) or the diagonal scaling. Especially, the number of iterations of the ICT is almost half number of the IC (0) when their numbers of nonzeros are the same.

Figure 5 shows the comparison of the dropping strategies of the nodal block ICT. The DOFs of the model are 4,719. All the 3×3 block dropping strategies require a larger number of iterations than the point ICT. However, the speedup of the 3×3 register blocking technique would be superior in some hardware specifications of the memory hierarchy system. Thus, we adopt the nodal block ICT instead of the point ICT. The figure shows that the numbers of iterations of the two norm cases and the one determinant case are approximately the same. That of the maximum norm case is the smallest in the range of the relatively small number of nonzeros. When the number of nonzeros becomes large, the numbers of iterations of the 1-norm, the 2-norm and the determinant becomes almost as small as that of the maximum norm.

4.3. Computation Time of the Subdomain Local Solver

Using the 1-node PC whose specification is shown in Table 1, numerical experiments of the subdomain local solver are conducted. In the previous subsection, the number of PCG iterations was discussed for comparing the PCG solvers, but in this subsection, the compu-

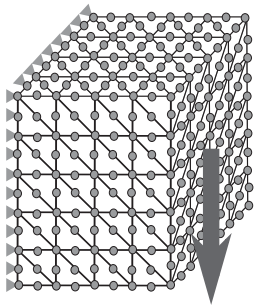

Fig. 2 Subdomain model for measuring the numbers of PCG iterations.

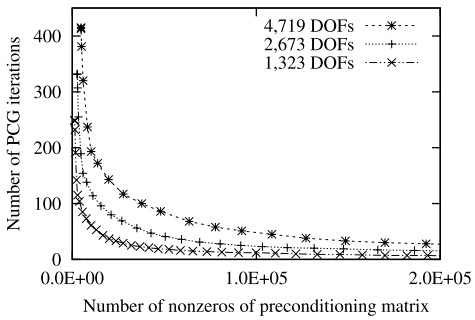

Fig. 3 Number of iterations of the point ICT preconditioned CG solver for various DOFs.

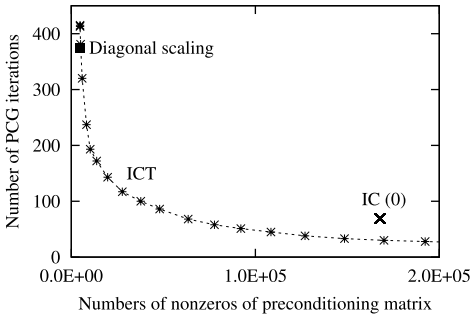

Fig. 4 Number of iterations of the point ICT, the IC (0) and the diagonal-scaling preconditioned CG solvers.

tation times are presented to compare the PCG and the direct LDL solvers. The measured computation times are shown in Table 3. The measured parameters of the coefficient matrix and the memory usage of the solvers are shown in Table 4. LDL is the skyline-based direct LDL solver, and CG is the preconditioned CG solver. These benchmarks are conducted on the DDM solver in which the subdomain local solvers are called. It is noticeable that the measured times include the times of generating preconditioning matrices. The tolerance of the PCG is set as 10^{-10}. The tolerance of the ICT is set as 0.01, so that the number of nonzeros of the ICT is almost the same as that of the IC (0). The dropping strategy of the 3×3 block ICT is the maximum entry. The compiler is Intel C/C++ Compiler (`icc`) version 12.0 and its compiler option is `-O3 -xAVX` so that one can use compiler optimization with the single instruction, multiple data (SIMD) interaction set of the advanced vector extensions (AVX).

For 2,187 DOFs, the LDL solver is 1.39 times faster than the CG solver with the ICT. For 3,993 DOFs, 1.39 becomes 1.26. This is because the CG solver utilizes the cache memory effectively whereas the LDL solver (forward and backward substitutions) always meets the memory wall. However, when the number of DOFs is 6,591, the coefficient matrix and the preconditioning matrix of the IC (0) and the ICT suffer from overflow of the last-level cache memory (2 MB/core). As discussed above, the computation time of the ICT preconditioned CG solver is comparable to that of the LDL solver in that the coefficient matrix and the preconditioning matrix of the ICT fit on the cache memory. In addition, Table 4 shows that the CG solver with the ICT is much more efficient in memory usage than is the LDL solver. One can analyze larger-scale problems with limited main memory using the ICT-based subdomain local solver than using the LDL-based local solver.

We discuss an estimation model for the subdomain local solvers. First, the matrix parameters of the DOFs N, the average half bandwidth w, the number of nonzeros of the coefficient matrix in upper triangular nnz_K and that of the preconditioning matrix $\mathrm{nnz}_{\tilde{L}}$ in the ICCG solvers are given. It can be noted that these parameters are not of $\boldsymbol{K}_i$ but of $\boldsymbol{K}_{Ii}$ in Eq. (2). When the mesh is structured and its elements are quadratic tetrahedrals, as shown in Fig. 2,

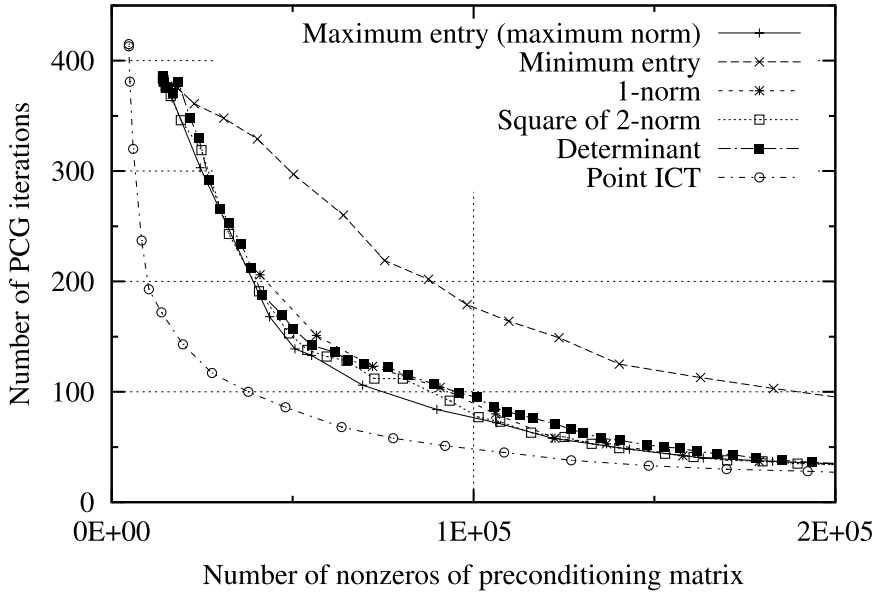

Fig. 5 Number of iterations of the point and the nodal block ICT preconditioned CG solvers.

the matrix parameters are represented by the following mesh parameters:

$$N = 3n_x n_y n_z, \tag{15}$$

$$w \simeq 6n_x n_y, \tag{16}$$

$$\mathrm{nnz}_K = cN, \tag{17}$$

$$\mathrm{nnz}_{\tilde{L}} = \begin{cases} N & \text{(CG with diag)} \\ \mathrm{nnz}_K & \text{(CG with IC (0))} \\ \text{variable} & \text{(CG with ICT)} \end{cases} \tag{18}$$

where n_x, n_y and n_z are the numbers of nodes on each edge and c is a constant factor of 30–40. By using N, w, nnz_K and $\mathrm{nnz}_{\tilde{L}}$, the computation time of the subdomain local solver is estimated in the following. When the coefficient and preconditioning matrices fit on the cache memory, a flops-based estimation model is optimal whereas a memory-based model is optimal in memory-limited solvers such as the LDL solver. The computation time of the CG solvers with the IC preconditioners is estimated as

$$t_{\mathrm{ICCG}} = \left(\frac{O_{\mathrm{SpMV}}}{F_{\mathrm{SpMV}}} + \frac{O_{\mathrm{FBS}}}{F_{\mathrm{FBS}}} \right) \times S = \left(\frac{4\mathrm{nnz}_K}{F_{\mathrm{SpMV}}} + \frac{4\mathrm{nnz}_{\tilde{L}}}{F_{\mathrm{FBS}}} \right) \times S \tag{19}$$

and that of the LDL solver is estimated as

$$t_{\mathrm{LDL}} = \frac{M}{W} = \frac{2Nw \times 8}{W} \tag{20}$$

where O_{SpMV} is the operation count of the sparse matrix-vector multiplication (SpMV), O_{FBS} is that of the forward and backward substitutions (FBS). F_{SpMV} is the measured performance (flops) of the SpMV. and F_{FBS} is the measured performance of the FBS. S is the measured number of ICCG iterations. M is the amount of memory reading and W is the measured memory bandwidth. Since S is known to be almost $O(N^{1/3})$, one can also use the estimated S in the case that the tolerance of the ICT is constant. The measured performances of components F_{SpMV}, F_{FBS} and memory bandwidth W are shown in Table 5. These benchmarks are performed in OpenMP parallel to consider the conflicting effect of the shared L3 (last-level) cache memory. Using Table 4, Table 5, Eq. (19) and Eq. (20), we can compare the measured

Table 1 Specification of the 1-node PC.

CPU	Intel Core i7-2600 (Sandy Bridge) 108.8 Gflops (27.2 Gflops/core) 3.4 GHz 4 cores
Cache	L2: 256 KB/core L3: 8 MB (2 MB/core)
DRAM	DDR3-1333 16 GB (4 GB × 4) 21 GB/s (5.25 GB/s/core)
OS	Debian GNU/Linux 6.0 (Squeeze)
Compiler	Intel C/C++ Compiler ver. 12

Table 2 Specification of the PC cluster.

Cluster	8 nodes
Network	Gigabit Ethernet
CPU	Intel Core i7-920 (Nehalem) 42.56 Gflops (10.64 Gflops/core) 2.66 GHz 4 cores
Cache	L2: 256 KB/core L3: 8 MB (2 MB/core)
DRAM	DDR3-1600 12 GB/node (2 GB × 6) 96 GB (*total*) 25.6 GB/s (6.4 GB/s/core)
OS	OpenSUSE 11.1
Compiler	Intel C/C++ Compiler ver. 12

Table 3 Measured computation time [ms] of the subdomain local solver. The numbers in parenthesis are the average numbers of the PCG iterations and those in brackets are the tolerances of the ICT. A hyphen means the cache memory was insufficient.

DOFs	2,187	3,993	6,591
Internal DOFs	1,323	2,673	4,719
LDL	1.09	3.92	10.3
CG with diag	4.05	11.3	31.0
	(115.)	(147.)	(183.)
CG with IC (0)	2.39	7.30	-
	(26.7)	(34.2)	
CG with ICT	1.51	4.95	-
	(10.8)	(12.4)	
	[0.01]	[0.01]	

Table 4 Measured parameters of the coefficient matrix and the memory usage [MB] of the subdomain local solver. The numbers in brackets are the tolerances of the ICT. A hyphen means the cache memory was insufficient.

DOFs	2,187	3,993	6,591
Internal DOFs	1,323	2,673	4,719
Ave. Half Bandwidth	250	407	604
# of Nonzeros	44,600	98,000	183,000
LDL	3.05	9.44	24.1
CG with diag	0.703	1.33	2.25
CG with IC (0)	1.36	2.57	-
CG with ICT	1.31	2.66	-
	[0.01]	[0.01]	

and the estimated performances of the subdomain local solvers, as shown in Fig. 6. The estimated computation time of the LDL and the CG with the diag well represents the measured computation time. However, for 6,591 DOFs, the estimation of the CG with the diag is a little smaller than the measured estimation. This is because, as shown in Table 4, the memory usage of the CG with the diag is a little larger than the system cache memory (2 MB/core). In the CG with the IC (0) and the ICT, the estimated time remains a little smaller than the measured time. This is because the computation time of generating the preconditioning matrix is not small. However, this estimation model is accurate enough to represent an approximate computation time.

4.4. Computation Time of the DDM Solver

4.4.1. OpenMP Benchmark on a Single Node On the PC of Table 1, the DDM solver is executed to evaluate the performance of the DDM with various subdomain local solvers. The number of OpenMP threads is 4. The parameters of the subdomain local solver are the same as those in the previous subsection. As parameters of the DDM solver, the preconditioner of the DDM is the BDD-diag(16) and the tolerance of the relative residual 2-norm in the DDM is 10^{-6}. The shape of the analysis model is a large bending plate shown in Fig. 7 and decomposed into cubic-shaped subdomains in which four surfaces of the six surfaces are constrained.

The computing result is shown in Table 6. The DOFs of each subdomain are 2,187. For the total DOFs of 7.1×10^6, the LDL solver is the fastest among the four subdomain local solvers, and the CG solver with the ICT is the second fastest. The speedup from the LDL to the CG with the ICT is only 0.732. When the total number of DOFs becomes 16×10^6, the LDL requires more memory than the system main memory (16 GB). When it becomes 28×10^6, the CG with the IC (0) also suffers from overflow of the main memory. However, the

Table 5 Measured performances and memory bandwidth per processor core on the PC of Table 1.

F_{SpMV} [Gflops/core]	F_{FBS} [Gflops/core]	W [GB/s/core]
5.26 (19.3 %)	4.06 (14.9 %)	4.50 (85.7 %)

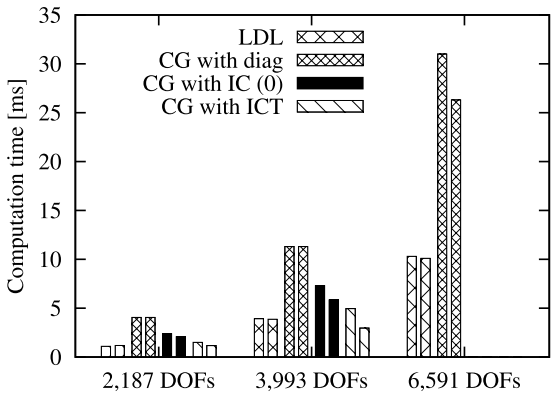

Fig. 6 Measured (left bar of each pair) and estimated (right bar) computation time of the subdomain local solver.

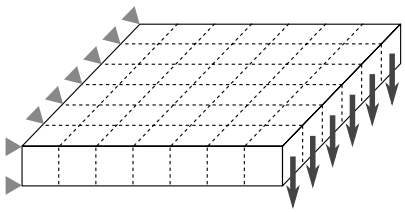

Fig. 7 Decomposed analysis model for measuring the computation time of the DDM solver.

Table 6 Measured computation time [s] and memory usage [GB] of the DDM solver with various subdomain local solvers on the 1-node PC. The numbers in brackets are the tolerances of the ICT. N/A means the main memory (16 GB) was insufficient.

Total DOFs	7.1×10^6	16×10^6	28×10^6	44×10^6
Subdomain DOFs	2,187	2,187	2,187	2,187
# of Subdomains	4,096	9,216	16,384	25,600
Measured Computation Time [s]				
LDL	180	N/A	N/A	N/A
CG with diag	573	1,300	2,270	3,530
CG with IC (0)	383	858	N/A	N/A
CG with ICT	246	530	969	2,360
	[0.01]	[0.01]	[0.02]	[0.2]
Measured Memory Usage [GB]				
LDL	11.0	N/A	N/A	N/A
CG with diag	1.8	4.1	7.4	11.0
CG with IC (0)	5.6	12.0	N/A	N/A
CG with ICT	4.1	9.2	14.0	14.0

CG with the diag and the CG with the ICT can be used for such large DOFs. Since the number of nonzeros of the preconditioning matrix can be freely controlled by a tolerance parameter, the CG with the ICT remains faster than the CG with the diag. For the DOFs of 44×10^6, the CG with the ICT remains faster than the CG with the diag even if the tolerance parameter becomes larger. As shown above, using the ICT preconditioner and waiting a little longer than using the LDL solver with an infinite amount of memory, one can analyze large-scale problems on a computer having a limited amount of main memory.

4.4.2. Flat MPI Benchmark on the PC Cluster Table 7 shows the computation time of the DDM solver conducted on the PC cluster of 8 nodes. The compiler option is `-fast`,

Table 7 Measured computation time [s] of the DDM solver with various subdomain local solvers on the PC cluster. The numbers in brackets are the tolerances of the ICT. N/A means the main memory (96 GB) was insufficient.

Total DOFs	44×10^6	64×10^6
Subdomain DOFs	2,187	2,187
# of Subdomains	25,600	36,864
Measured Computation Time [s]		
LDL	199	N/A
CG with diag	772	1,130
CG with IC (0)	529	755
CG with ICT	341	489
	[0.01]	[0.01]
Measured Memory Usage [GB]		
LDL	83.2	N/A
CG with diag	20.8	29.6
CG with IC (0)	44.0	64.8
CG with ICT	34.4	48.8

so that one can use compiler optimization using the SIMD interaction set of the streaming SIMD extensions (SSE). Then, as shown in the table, the computation time is similar to the benchmark of the 1-node PC, the LDL is also faster than the CG with the ICT for 44×10^6 DOFs. In larger-scale problems than 64×10^6 DOFs, the LDL solver is undesirable due to the overflow of memory, and the CG with the ICT becomes the fastest of the three PCG solvers. We analyzed a problem of 64 million DOFs in 8 minutes using the ICT-based subdomain local solver.

5. Conclusions and Future Work

In this paper, we proposed an optimized implementation of the CG method with an ICT factorization preconditioner for solving subdomain local problems in the DDM. In the modified ICT, threshold is conducted after the complete Cholesky factorization, while the threshold is originally conducted on each matrix row during the incomplete factorization. The modified ICT preconditioner has a higher potential to accelerate CG convergence than the original ICT and also than other IC preconditioners. In our implementation for the subdomain local solver of the DDM, the number of nonzeros is controlled to make both coefficient and preconditioning matrices fit on the cache memory. By using the cache memory effectively, the local solver of the DDM can be accelerated on multi-core/many core environments. In another point of view, using the main memory efficiently, one can analyze very large-scale problems on a computer with limited main memory.

Through the numerical experiments, the number of PCG iterations of the ICT-based subdomain local solver was observed to be the smallest among those of the ICT, the IC (0) and the diagonal-scaling preconditioned CG solvers. By comparing computation times, the ICT-based CG solver was comparable to the skyline-based direct LDL solver. However, using the memory-friendly ICT, we analyzed large-scale problems which cannot be solved by the LDL-based solver, because the ICT-based solver is more efficient in main memory usage than the direct LDL solver.

The future work is to solve problems of over 1 billion DOFs. In the problems, the number of DOFs of each subdomain becomes over 10,000. Such a large number of DOFs requires a parallel subdomain local solver, that can perform with several threads on a shared memory environment.

Acknowledgments

This work was performed as a part of the JST-CREST program on *Simulation for Predicting Quake-Proof Capability of Nuclear Power Plants*. The authors also wish to thank the members of the ADVENTURE project for having discussions.

References

(1) Smith, B., Bjørstad, P. and Gropp, W. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, (1996).

(2) Toselli, A. and Widlund, O. *Domain Decomposition Methods—Algorithms and Theory (Springer Series in Computational Mathematics)*. Springer, (2005).

(3) Mandel, J. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, Vol. 9, No. 3 (1993), pp. 233–241.

(4) Shioya, R., Ogino, M., Kanayama, H. and Tagami, D. Large scale finite element analysis with a balancing domain decomposition method. *Key Engineering Materials*, Vols. 243–244 (2003), pp. 21–26.

(5) `http://adventure.sys.t.u-tokyo.ac.jp/`

(6) Ogino, M., Shioya, R., Kawai, H. and Yoshimura, S. Seismic response analysis of nuclear pressure vessel model with ADVENTURE system on the Earth Simulator. *Journal of the Earth Simulator*, Vol. 2 (2005), pp. 41–54.

(7) Yamada, T., Ogino, M. and Yoshimura, S. Prediction and numerical validation of optional number of subdomains on balancing domain decomposition method (*in Japanese*). *Transactions of the Japan Society for Computational Engineering and Science*, Paper No. 20090014 (2009).

(8) Kanayama, H. and Sugimoto, S. Effectiveness of A-ϕ method in a parallel computing with an iterative domain decomposition method. *IEEE Transactions on Magnetics*, Vol. 42, No. 4 (2006), pp. 539–542.

(9) Takei, A., Yoshimura, S. and Kanayama, H. Large-scale full wave analysis of electromagnetic field by hierarchical domain decomposition method. *Computer Modeling in Engineering and Sciences*, Vol. 40, No. 1 (2009), pp. 63–81.

(10) Kawai, H., Ogino, M., Shioya, R. and Yoshimura, S. Large scale elasto-plastic analysis using domain decomposition method optimized for multi-core CPU architecture. *Key Engineering Materials*, Vols. 462–463 (2011), pp. 605–610.

(11) Saad, Y. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, (2003).

(12) Ajiz, M. A. and Jennings, A. A robust incomplete Choleski-conjugate gradient algorithm. *International Journal for Numerical Methods in Engineering*, Vol. 20, No. 5 (1984), pp. 949–966.

(13) Kakihara, M. and Fujino, S. Convergence of quasi-robust incomplete Cholesky CG method by means of uniformed modification for diagonal entries. *INFORMATION*, Vol. 8, No. 1 (2005), pp. 53–60.

(14) Benzi, M. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, Vol. 182, No. 2 (2002), pp. 418–477.

(15) Benzi, M. and Tuma, M. A robust incomplete factorization preconditioner for positive definite matrices. *Numerical Linear Algebra with Applications*, Vol. 10 (2003), pp. 385–400.

(16) Ogino, M., Shioya, R. and Kanayama, H. An inexact balancing preconditioner for large-scale structural analysis. *Journal of Computational Science and Technology*, Vol. 2, No. 1 (2008), pp. 150–161.