ICMIEE18-324

Techniques to Improve the Parallel Finite Element Method to Solve Large Scale Problems

Abul Mukid Mohammad Mukaddes[1], *Masao Ogino*[2], *and Ryuji Shioya*[3]
[1]Shahjalal University of Science and Technology, Bangladesh
[2] Information Technology Centre, Nagoya University, JAPAN
[3]Faculty of Information Sciences and Arts, Toyo University, JAPAN
mukaddes1975@gmail.com, *masao.ogino@cc.nagoya-u.ac.jp*, *shioya@toyo.jp*

ABSTRACT
With advent of computer technology finite element method has achieved the popularity in the simulation of engineering problems. In order to get the accurate result, the finite element discretization of the problem should be as fine as possible. The finest discretization of three dimensional (3D) problems results in the large linear systems of equations. Consequently, the solution of equation having millions of co-efficient has become the biggest challenge of the researchers. Recent development of the computer technology necessitates the efficient ways to solve those linear systems of equations using less memory and time. In this paper, some efficient ways have been proposed to solve large scale heat transfer and solid mechanics problems using the parallel finite element methods. Firstly, an iterative method with different preconditioners is implemented in the both thermal and solid problems and significant results have been achieved. Secondly, it is shown that use of sparse matrix storage scheme reduces the computation time and the required memory. Different variations of sparse matrix storages schemes have been studied here for different class of engineering problems.

Keywords: Finite element, domain decomposition, compressed sparse row, heat conduction, preconditioner

1. Introduction

ADVENTURE System [1] has been developed to solve large scale engineering problems in the parallel computer. Adventure_Thermal is one of its module that can solve large scale steady and unsteady heat conduction problems. The basic algorithm is the Hierarchical Domain Decomposition Method (HDDM) which is based on the non-overlapping domain decomposition method. In this method, the whole domain is divided into parts and then each part is divided into subdomains. Each subdomain problem is solved using finite element technique. The Conjugate Gradient (CG) method is used to solve the interface problem that is constituted by share nodes of subdomains [2, 3]. The finite element discretization of the subdomain problems results in the sparse matrix. Sparse matrices, by definition are populated by many zeroes and thus special storage schemes are used to enable efficient storage and computational operations. These representations usually store the non-zero values of the matrix with additional indexing information about the position of these values [4]. The HDDM method used in the Adventure_Thermal is gradually improved with different ways. The research is still going on to make the module faster for the users. The techniques that are implemented to improve parallel finite element method used in Adventure system are discussed in this paper.

First: The subdomain problem is solved using the direct linear solution technique and iterative Conjugate Gradient (CG) method is used to solve the interface problem. The CG method is improved after implementing a suitable preconditioner. It is shown here that Balancing Domain Decomposition (BDD) can be an efficient preconditioner. This reduces the computation time and memory significantly when the problem is solved in the parallel computer.

Second: Both subdomain problem and interface problem is solved using the iterative CG methods which improves the computational time and memory significantly when the problem is solved in single computer.

Third: The CG method that is employed to solve the interface problem uses the Sparse Matrix Vector multiplication (SpMxV) as its basic operation. The non-zero sparse matrix storage formats used in this module are based on Compressed Sparse Row (CSR) [4]. The variations of this method are investigated in both thermal and solid problems. All CSR type storage formats are compared with the traditional skyline problems. They outperform the skyline method in terms of memory and computation time. Again in the construction part of the BDD type preconditioners, matrix vector multiplication is necessary thus sparse matrix storage schemes are used there too.

The subdomain discretization is discussed in the next section. BDD preconditioner and sparse matrix storage formats are introduced in section 3 and 4 respectively. The computational performances are given in section 5.

2. Subdomain Discretization

After the finite element discretization of the partial differential equation of heat transfer and structural problems yields a linear system of form,

$$Ku = f \qquad (1)$$

where K is the global stiffness matrix, u is unknown vector and f is a known vector. For a large scale problem, the linear system (1) cannot be solved using a single processor due to memory constraint. The solution is to use the parallel computer. The domain decomposition method is a popular technique to solve a large scale finite element problem in the parallel computing environment.

The method decomposes the domain Ω into N non-overlapping subdomains, $\{\Omega_i\}_{i=1,\dots,N}$. Thus the stiffness matrix K of equation (1) could be generated by subassembly:

$$K = \sum_{i=1}^{N} R^{(i)} K^{(i)} R^{(i)T} \qquad (2)$$

where $R^{(i)T}$ is the 0-1 matrix which translates the global indices of the nodes into local (subdomain) numbering. Denoting $u^{(i)}$ as the vector corresponding to the nodes in $\Omega^{(i)}$, 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 $\partial\Omega^{(i)}$, called interface degrees of freedom, $u_I^{(i)}$ for interior degrees of freedom and $u_T^{(i)}$ for essential boundary conditions (temperature for heat conduction problem). The subdomain matrix $K^{(i)}$, vector $u^{(i)}$ are then split accordingly:

$$K^{(i)} = \begin{pmatrix} K_{II}^{(i)} & K_{IB}^{(i)} & K_{IT}^{(i)} \\ K_{BI}^{(i)} & K_{BB}^{(i)} & K_{BT}^{(i)} \\ K_{TI}^{(i)} & K_{TB}^{(i)} & K_{TT}^{(i)} \end{pmatrix}, \quad \begin{Bmatrix} u_I^{(i)} \\ u_B^{(i)} \\ u_T^{(i)} \end{Bmatrix}. \qquad (3)$$

Similarly equation (1) can be written as

$$\begin{bmatrix} K_{II} & K_{IB} & K_{IT} \\ K_{BI} & K_{BB} & K_{BT} \\ K_{TI} & K_{TB} & K_{TT} \end{bmatrix} \begin{Bmatrix} u_I \\ u_B \\ u_T \end{Bmatrix} = \begin{Bmatrix} f_I \\ f_B \\ f_T \end{Bmatrix}. \qquad (4)$$

After eliminating the interior degrees of freedom, problem (4) is reduced to a problem on interface

$$S u_B = g \qquad (5)$$

where the Schur complement $S = \sum_{i=1}^{N} R_B^{(i)} S^{(i)} R_B^{(i)T}$ is assumed to be positive definite, u_B is the vector of the unknown variables on the interface, g is a known vector and $S^{(i)}$ are the local Schur complements of subdomain $i = 1,\dots,N$, assumed to be positive semi-definite. The problem (5) is solved by the Conjugate Gradient (CG) method which can use a preconditioner like Balancing Domain Decomposition (BDD) which is first proposed by Jan Mandel [5].

3. Balancing Domain Decomposition

The BDD preconditioner proposed by Jan Mandel is constructed by solutions of the local Neumann-Neumann problems on subdomains coupled with a coarse problem in a coarse space. The BDD preconditioner is of the form:

$$M_{BDD}^{-1} = Q_c + (I - Q_c S) Q_l (I - S Q_c) \qquad (6)$$

where Q_l is the local level part [5] and Q_c is the coarse level part of the preconditioner.

3.1 Coarse Level

The application of the coarse term $Q_c = R_0 (R_0^T S R_0)^{-1} R_0^T$ amounts to the solution of a coarse problem whose coefficient matrix is $S_0 = R_0^T S R_0$. The operator R_0 translates the coarse degrees of freedom to the corresponding global degrees of freedom and is defined by

$$R_0 = \left[R_B^{(1)} D^{(1)} Z^{(1)}, \dots, R_B^{(N)} D^{(N)} Z^{(N)} \right] \qquad (7).$$

For the scalar heat conductive problem, $Z^{(i)}$ is a column constant vector and can be defined by

$$Z^{(i)} = (1 \dots 1)^T \qquad (8)$$

where the number of element "1" is for each interface point in subdomain i. The operator R_0 is a n by N matrix, where n is the dimension of S. The implementation of the BDD preconditioner (6) goes as follows:

Step 1: Balance the original residual by solving the coarse problem for an unknown vector $\lambda \in \Re^N$ *:*

$$S_0 \lambda = R_0^T r. \qquad (9)$$

Step 2: Set

$$s = r - S R_0 \lambda. \qquad (10)$$

Step 3: Solve Neumann-Neumann problems and average these results

$$\bar{u} = \sum_{i=1}^{N} R_B^{(i)} D^{(i)} S^{(i)+} D^{(i)T} R_B^{(i)T} s. \qquad (11)$$

Step 4: Compute

$$\bar{s} = r - S\bar{u}. \qquad (12)$$

Step 5: Solve the coarse problem again for an unknown vector $\mu \in \Re^N$

$$S_0 \mu = R_0^T \bar{s}. \qquad (13)$$

Step 6: Find the preconditioned vector

$$z = \bar{u} + R_0 \mu. \qquad (14)$$

The equation (6) can also be expressed as:

$$M_{BDD}^{-1} = \left(P + (I - P) Q_l S (I - P) \right) S^{-1} \qquad (15)$$

where $P = Q_c S$ is the $S-$orthogonal projection onto the coarse space.

The construction of the BDD preconditioner uses the matrix vector multiplication in equation (10) and (12) where sparse matrix storages schemes are used.

4. Sparse Matrix Storage Formats

The matrix originates from the finite element discretization is naturally sparse pattern. The pattern of sparse matrix contains many zero elements within the non-zero elements. The most popular useful storage technique is the Skyline method which cannot avoid storing of zero elements within the matrix.
In order to take the advantage of avoiding the large number of zero elements, special formats are used to store sparse subdomain matrices. The main goal is to represent only the non-zero elements (nnz) considering the memory requirements and the computation time. Several sparse matrix storage techniques are described in this section and implemented in Adventure systems. Some of the techniques are described here.

Skyline or Variable Band (SKY)

The Skyline representation becomes popular for direct solvers especially when pivoting is not necessary. For symmetric matrices, this representation only stores the lower triangular matrix and hence requires half storage space. The matrix elements are stored using three arrays: *data*, *row_ptrn*, *col_ind*. The array *data* stores the elements of *A* row by row, *col_ind* contain column number of first element of each row and *row_ptr* array points to the start of every row.
This storage format stores some zero elements while other methods explained below do not. The skyline storage format of the examples matrix *A* is shown below.

$$A = \begin{bmatrix} 1.0 & 2.0 & 0.0 & 6.0 & 0.0 \\ 2.0 & 3.0 & 4.0 & 7.0 & 9.0 \\ 0.0 & 4.0 & 5.0 & 0.0 & 0.0 \\ 6.0 & 7.0 & 0.0 & 8.0 & 10.0 \\ 0.0 & 9.0 & 0.0 & 10.0 & 11.0 \end{bmatrix}$$

data (13):

1.0	2	3	4	5	6	7	0	8	9	0	10	11

row_ind(6):

1	2	4	6	10	13

col_ind(5):

1	1	2	1	2

Coordinate Storage (COO)

The simplest sparse matrix storage structure is COO. It uses three arrays of length *nnz(number of non-zeroes)* to store the sparse matrix: *data*, *row_ind* and *col_ind*. The array *data* stores the non-zero elements of the sparse matrix. The *row_ind* and *col_ind* stores the row and column indices of the corresponding element. The COO storage format of the *example matrix A* is shown below.
data

1.0	2	3	4	5	6	7	8	9	10	11

row_ind:

1	2	2	3	3	4	4	4	5	5	5

col_ind:

1	1	2	2	3	1	2	4	2	4	5

Compressed Sparse Row (CSR)

The CSR format is specified by *{data, row_ptr and col_ind}*. The 1D array *data* of length *nnz* contains the non-zero elements of *A* row-wise fashion, *col_ind* of length *nnz* contains the column indices which correspond to the non-zero elements in the array *data*. The integer vector *row_ptr* of length *nrow+1* contains the pointers to the beginning of each row in the array *data* and *col_ind*. With the *row_ptr* array we can easily compute the number of non-zero elements in the *i*th row as *row_ptr*[i+1] - *row_ptr*[i]. The last element of *row_ptr* is *nnz*. The CSR representation of an example symmetric matrix *A*:

data(11):

1.0	2	3	4	5	6	7	8	9	10	11

col_ind(11)	1	1	2	2	3	1	2	4	2	4	5

row_ptr(6)	1	2	4	6	9	11

Consider the symmetric part of a matrix (figure-1) which originates from structural problems. The CSR storage of the above matrix is shown below. The conventional CSR technique stores the elements of matrix using *data*, *col_ind* and *row_ptr* (figure-2). The pattern of the matrix has 3 off-diagonal block matrix and 4 diagonal tri-angular matrix. The diagonal block matrix and off-diagonal block matrix can be stored in separate vectors *(diag and data)*. In that case only location of first element of each off-diagonal block matrix is enough to locate the other elements. As a result the *brow_ptr* and *bcol_ind* will become like the figure-3 shown. The storing method is named as diagonal block compressed sparse row (DBCSR).

```
1
2  3
4  5  6
7  8  9  10
11 12 13 14 15
16 17 18 19 20 21
            22 23 24 25
            26 27 28 29 30
            31 32 33 34 35 36
37 38 39                40
41 42 43                44 45
46 47 48                49 50 51
```

Fig.1 Matrix format (B) for structural problem

data[51]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	18	19	12	21	22	23	24	25	26	27	28	29	30	31	32	33	34
	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51

col_ind [51]	0	0	1	0	1	2	0	1	2	3	0	1	2	3	4	0	1
	2	3	4	5	3	4	5	6	3	4	5	6	7	3	4	5	6
	7	8	0	1	2	9	0	1	2	9	10	0	1	2	9	10	11

row_ptr [12]	0	1	3	6	10	15	21	25	30	36	40	45	51

Fig. 2 CSR storage of block matrix B

diag[24]	1	2	3	4	5	6	10	14	15	19	20	21	25	29	30	34	35
	36	40	44	45	49	50	51										

data[27]	7	8	9	11	12	---	22	23	24	26	---	37	38	29	41	--	48
brow_ptr (4)	0						9					18					26
bcol_ind (3)	0						3					0					

Fig. 3 DBCSR storage of block matrix B

The advantage of DBCSR is that *diag* does not indexing thus indexing part reduces the required memory. This technique is implemented in Adventure_Solid. In this research, some others CSR type techniques are identified and implemented in the Adventure System. Detail results are shown in the next section

5. Performance Evaluation

5.1. Model Description

The model used to investigate the performance of ADVENTURE_Thermal in different aspects is High Temperature Test Reactor (HTTR) shown in Figure-2. The model parameters are given in Table-1.

5.2 Performance of preconditioner

The balancing domain decomposition method has been implemented in both Adventure_Thermal and Adventure_Solid. After implementing BDD, in the Adventure_Thermal, the CG converges rapidly with

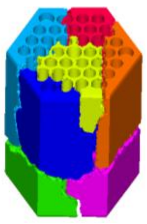

Fig. 4 High Temperature Test Reactor (HTTR)

Table-1 Parameters of HTTR

Model	meshing points	elements	#sub	#parts	#proc.
Mesh 1	1,893,340	1,167,268	4800	6	6
Mesh 2	13,853,784	9,338,144	76800	96	96

The HTTR model is analysed using the Adventure_Thermal-2.0. Figure 5 and Table 1 show the computational results of Mesh 1 of HTTR model. BDD takes less computational time but require more memory. In all types of preconditioning techniques, the CSR shows better results compare to Skyline storage schemes.

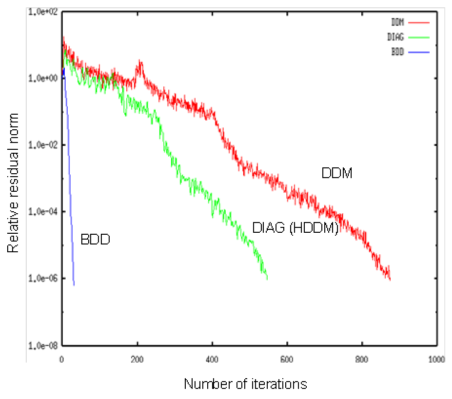

Fig. 5 Convergence of CG method

Table 1 Computational results (1.8 M mesh points)

Precon.	**Storage formats**	**#iter.**	**Total time(s)**	**Mem. (MB/pro)**
None	SKY	984	233	587
	CSR	956	130	350
Diagonal Scaling	SKY	613	150	587
	CSR	603	86	351
BDD	SKY	33	42	829
	CSR	33	31	659

5.3 Performance of Sparse Storage Formats

In order to find out the performance of different sparse storage formats, a structural problem of 5.4 M meshing points is analyzed using the Adventure_Solid. The memory required for the solution of a structural problem having 5.4 M meshing points using different matrix storage techniques are shown in figure 6. The figure gives the information that CSR types require less memory compare to the skyline storage schemes. Among the different CSR types, DBCSR shows best results in terms of memory requirement and computation time (figure 7).

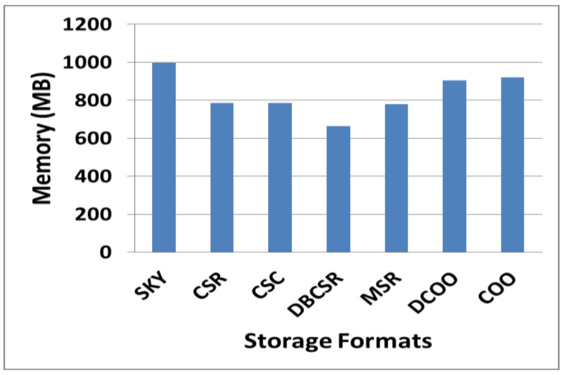

Fig. 6 Memory required for different storage formats

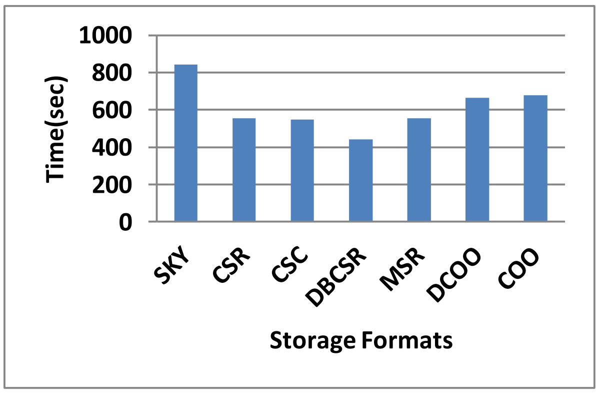

Fig. 7 Time required for different storage formats

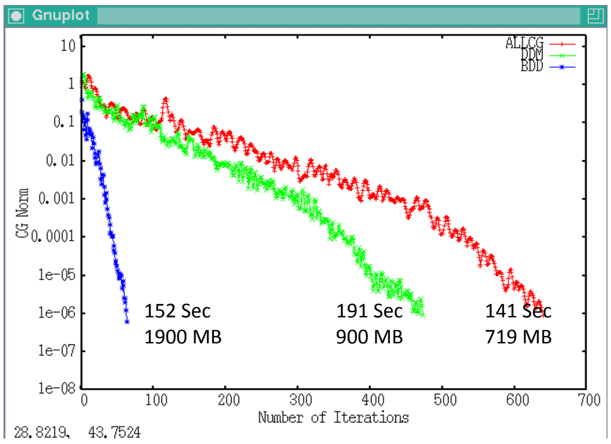

Fig. 8 Comparisons of BDD, DDM and AllCG

5.4 Performance of AllCG solvers

Both Adventure_Thermal and Adventure_Solid uses the CG solver to solve the interface problem and direct Gauss elimination technique is used to solve the interior of the subdomain problem. Recently a new solver is introduced to Adventure_Thermal called AllCG which solve both interface and interior problems using CG iterative method. According to figure-8 though it converges slowly but the computational time is reduced. It is advised to use this solver when Adventure_Theremal is used in a single computer.

6. Conclusion

Several techniques to solve the large scale finite element problems efficiently are studied. They are implemented in the open source CAE software Adventure System. Large scale problem of having 5.4 million meshing points is solved in the parallel computer to investigate the different preconditioning techniques and sparse matrix storage formats. Both thermal and structural problems CSR shows better performance. DBCSR shows best performance for the case of structural problems.

References

[1] Adventure project, http://adventure.sys.t.u-tokyo.ac.jp/

[2] Mukaddes, A.M.M., Ogino, M., Kanayama, H. and Shioya, R., A Scalable Balancing Domain Decomposition Based Preconditioner for Large Scale Heat Transfer Problems, *JSME International Journal*, *Series B*, Vol. 49, No. 2, pp. 533-540.

[3] Shioya, R., Ogino, M., Kanayama, H. and Tagami, D., Large Scale Finite Element Analysis with a Balancing Domain Decomposition Method, *Key Engineering Materials.*, 243-244 (2003), p.21-26.

[4] Mandel, J., Balancing Domain Decomposition, *Comm. on Numerical Methods in Engineering*, Vol. 9 (1993), p. 223-241.