Treatment of Block-Based Sparse Matrices in Domain Decomposition Method

Abul Mukid Mohammad Mukaddes
Department of Industrial and Production Engineering
Shahjalal University of Science and Technology
Sylhet, Bangladesh
mukaddes1975@gmail.com

Ryuji Shioya
Faculty of Information Science and Arts
Toyo University
Kawagoe, Japan
shioya@toyo.jp

Masao Ogino
Information Technology Center, Nagoya University,
Nagoya, Japan
masao.ogino@cc.nagoya–u.ac.jp

Hiroshi Kanayama
Faculty of Mathematical and Physical Sciences
Japan Women's University, Japan
kanayamah@fc.jwu.ac.jp

***Abstract*— The domain decomposition method involves the finite element solution of problems in the parallel computer. The finite element discretization leads to the solution of large systems of linear equation whose matrix is naturally sparse. The use of proper storing techniques for sparse matrix is fundamental especially when dealing with large scale problems typical of industrial applications. The aim of this research is to review the sparsity pattern of the matrices originating from the discretization of the elasto-plastic and thermal-convection problems. Some practical strategies dealing with sparsity pattern in the finite element code of adventure system are recalled. Several efficient storage schemes to store the matrix originating from elasto-plastic and thermal-convection problems have been proposed. In the proposed technique, inherent block pattern of the matrix is exploited to locate the matrix element. The computation in the high performance computer shows better performance compared to the conventional skyline storage method used by the most of the researchers.**

Keywords: sparse matrix, compressed sparse, block size, domain decompose, adventure system;

I. INTRODUCTION

With the advent of high performance computer, researchers are showing interest to analyze large scale three dimensional problems using the finite element method. To get the accurate simulation results, the finite element grids should also be finer. The degrees of freedom (dof) grow rapidly as the grid gets finer and nowadays it is customary to deal with trillions of dof. At the same time, domain decomposition method has become much popular to treat those large scale problems in the parallel cluster computer. This method decomposes the whole problem into subproblems and divides the dof into two groups. One group belongs to the dof within the subdomains and the other belongs to the shared dof. The shared dof is solved by the conjugate gradient (CG) method which involves finite element solution for each subproblem and matrix-vector multiplication in each CG iteration.

We have been developing the Adventure system [1] for the solution of large scale engineering problems. Among the modules of the adventure system, Adventure_Solid [2], Adventure_Thermal [3, 4] and Adventure_sFlow have been considered in this paper. The hierarchical domain decomposition method (HDDM) has been adopted for the parallel solver in those modules. Simplified diagonal scaling and balancing domain decomposition (BDD) are used to accelerate the CG iterations.

There are several ways to speed-up the domain decomposition method; *one*-reducing CG iterations using a suitable preconditioner [4]; *two*-storing the matrix efficiently [5]; *three*-constructing the program more parallel. The finite element discretization of the partial differential equations leads to sparse linear system with different sparsity patterns based on the selected problems. An efficient storing technique can reduce the computational time of matrix-vector multiplication and the used memory.

The sparsity pattern is described by the distribution of non-zero elements of the sparse matrix. The distribution of non-zero elements depends on the topology of the adopted computational grid, on the kind of finite element method chosen and on the kind of problem discretized. Thus the sparsity pattern is completely known before the construction of the matrix from its nodal connectivity. Therefore the matrix can be stored efficiently by excluding the unnecessary a priori known zero-elements. The use of efficient storing techniques for sparse matrices is computationally important especially when dealing with large scale problems in the high performance computer. Moreover, the matrix results in the discretization of different problems have its own sparsity pattern. The matrix originating from the elasto-plastic and thermal problems is symmetric while from the thermal-convection is asymmetric. Again structural and thermal-convection problem have some inherent block shape. Suppose, thermal problem has one dof per node, elasto-plastic problem has three dof per node and thermal-convection problem has five dof per node. This dof per node shape the matrices as block matrices. The built-in block shape is exploited in storing of the sparse matrices. In this paper, several sparse matrix storage schemes have been reviewed and some efficient techniques are proposed. The proposed techniques are problem sensitive. The proposed sparse matrix storing

IJSMS

techniques have been implemented in Adventure_Solid and Adventure_sFlow and good performances have been achieved.

II. DOMAIN DECOMPOSTIION

Discretization of partial differential equations in a domain Ω using the finite element method leads to large and sparse linear systems like

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

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

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

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 elements 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 (for the subdomains which has these boundary conditions). The subdomain matrix $K^{(i)}$, vector $u^{(i)}$ are then split accordingly:

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

Similarly equation (1) can be written as

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

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

$$Su_B = g \quad (5)$$

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 $i = 1,...,N$, assumed to be positive semi-definite. The problem (5) is solved by the conjugate gradient method. The flow of algorithm of the basic domain decomposition method is shown next.

Step 0: Initialize: $u_B^{(0)}$

(1a) Find: $u_I^{(0)}$, $\begin{bmatrix} K_{II} & K_{IB} & K_{IT} \end{bmatrix} \begin{Bmatrix} u_I^{(0)} \\ u_B^{(0)} \\ u_T^{(0)} \end{Bmatrix} = f_I$

(1b)Calculate: $g^{(0)} = w^{(0)} = \begin{bmatrix} K_{BI} & K_{BB} & K_{BT} \end{bmatrix} \begin{Bmatrix} u_I^{(0)} \\ u_B^{(0)} \\ u_T \end{Bmatrix} - f_B$

Step 1: (iteration)

(2a) Find $v^{(k)}$: $\begin{bmatrix} K_{II} & K_{IB} & K_{IT} \end{bmatrix} \begin{Bmatrix} v^{(k)} \\ w^{(k)} \\ 0 \end{Bmatrix} = 0$

(2b) Find: $Sw^{(k)} = \begin{bmatrix} K_{BI} & K_{BB} & K_{BT} \end{bmatrix} \begin{Bmatrix} v^{(k)} \\ w^{(k)} \\ 0 \end{Bmatrix}$

2(c) Update for each k iteration: $u_B^{(k)}, g^{(k)}$ *and* $w^{(k)}$

$$\rho^{(k)} = \frac{\left(g^{(k)} \cdot g^{(k)}\right)}{\left(w^{(k)} \cdot Sw^{(k)}\right)}$$

$$u_b^{(k+1)} = u_b^{(k)} - \rho^{(k)} w^{(k)}$$

$$g^{(k+1)} = g^{(k)} - \rho^{(k)} Sw^{(k)}$$

$$If \left(\frac{\left(g^{(k+1)} \cdot g^{(k+1)}\right)}{\left(g^{(0)} \cdot g^{(0)}\right)} \right) \prec \varepsilon \ /* \ convergence \ */$$

$$\gamma^{(k)} = \frac{\left(g^{(k+1)} \cdot g^{(k+1)}\right)}{\left(g^{(k)} \cdot g^{(k)}\right)}$$

$$w^{(k+1)} = \gamma^{(k)} w^{(k)} + g^{(k+1)}$$

A. Scope of Sparse Storage Schemes

We remark that the Schur complement matrix S is not constructed explicitly, though the multiplication of a vector with the Schur complement is required in the CG method. The action of the matrix S on a typical vector x can be implemented by using the block elements of the matrix $K^{(i)}$. Again the

$$\begin{aligned} y &= S \cdot x \\ &= \left\{ \sum_{i=1}^{N} R_B^{(i)} S^{(i)} R_B^{(i)T} \right\} \cdot x \\ &= \sum_{i=1}^{N} R_B^{(i)} S^{(i)} x^{(i)} \\ &= \sum_{i=1}^{N} R_B^{(i)} y^{(i)} \end{aligned} \quad (6)$$

where,

$$y^{(i)} = S^{(i)} x^{(i)} = \left(K_{BB}^{(i)} - K_{IB}^{(i)T} \left(K_{II}^{(i)} \right)^{-1} K_{IB}^{(i)} \right) x^{(i)}. \quad (7)$$

Action of $S^{(i)}$ on a vector can be implemented by solving a *Dirichlet* problem on $\Omega^{(i)}$ and it needs subdomain wise matrix-vector multiplication twice in each iteration as shown in equation (7). The contribution of the matrix in equation (7)

IJSMS

comes from subdomains matrix which is stored using non-zero-only sparse matrix storage schemes.

III. ADVENTURE SYSTEM

We have been developing the adventure system [1] which is a group module to solve the large scale engineering problems. Initially, its target was to solve 100 million degrees of freedom. But some of its module is now able to solve more than 1000 million degrees of freedom. Hierarchical Domain Decomposition Method (HDDM), a parallel finite element technique is used to solve the engineering problems in the parallel computer. The HDDM system uses the conjugate gradient method to solve the interface problem. Matrix-vector multiplication is the most time consuming part in the conjugate gradient method. The HDDM system can be accelerated either by using a preconditioner or by reducing time for matrix-vector multiplication. An efficient technique for storing sparse matrix can reduce the computational time as well as required memory of the HDDM system.

Most of the adventure modules use the basic finite element approach. In fact, in finite element matrix (i, j) belongs to the pattern if node i and j share a common element. Note that by using this definition it is possible that we allocate storage for elements which may eventually be zero: we exclude only elements which are a-priori zero. As example, consider a matrix that might have arisen using linear finite elements on the grid of Fig. 1, left. The pattern of this matrix is shown on the right. In particular, the matrix A (Fig. 2) is in full format (array) where the values of the matrix elements are not relevant for this discussion, and indeed they have been just made up to allow to identify them easily.

In this paper we try to study the sparsity pattern of Adventure_Solid and Adventure_sFlow. Adventure_Solid uses the tetrahedral and quadratic tetrahedral elements while Adventure_sFlow uses the tetrahedral element. From information of the nodal connectivity, a binary tree has been developed. This binary tree is used to locate the non-zero elements in the subdomain matrix.

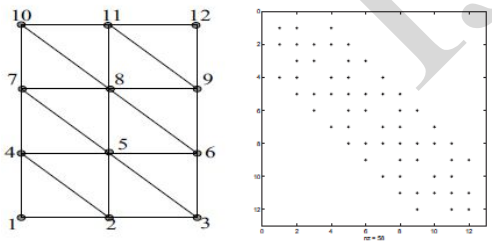


Fig 1. Example of Grid with Linear Finite Element and Associated Pattern of the Matrix

$$A = \begin{bmatrix} 101. & 102. & 0. & 103. & 0. & 0. & 0. & 0. & 0 & 0. & 0. & 0. \\ 104. & 105. & 106. & 107. & 108. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 109. & 110. & 0. & 111. & 112. & 0. & 0. & 0. & 0. & 0. & 0. \\ 113. & 114. & 0. & 115. & 116. & 0. & 117. & 0. & 0. & 0. & 0. & 0. \\ 0. & 118. & 119. & 120. & 121. & 122. & 123. & 124. & 0. & 0. & 0. & 0. \\ 0. & 0. & 125. & 0. & 126. & 127. & 0. & 128. & 129. & 0. & 0. & 0. \\ 0. & 0. & 0. & 130. & 131. & 0. & 132. & 133. & 0. & 134. & 0. & 0. \\ 0. & 0. & 0. & 0. & 135. & 136. & 137. & 138. & 139. & 140. & 141. & 0. \\ 0. & 0. & 0. & 0. & 0. & 142. & 0 & 143. & 144. & 0 & 145. & 146. \\ 0. & 0. & 0. & 0. & 0. & 0. & 147. & 148. & 0. & 149. & 150. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 151. & 152. & 153. & 154. & 155. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 156. & 0 & 157. & 158. \end{bmatrix},$$

Fig 2. Numerical Values of the Matrix

IV. SPARSE MATRICES STORING TECHNIQUES

Efficient storage of sparse matrix effects on both computational time and memory especially in the matrix-vector multiplication part. Most useful technique is the Skyline method. But its main disadvantage is that, it needs to store a countable amount of unnecessary zero elements. To avoid this problem, the compressed sparse row (CSR) is proposed by the researchers [6,7]. This is known as non-zero only sparse storage technique. In this technique, one dimensional double array is used to store the element of the matrix and two vectors are used for indexing; locating the matrix element in the one dimensional array. In this section, different sparse matrix storage schemes differentiated on the indexing have been proposed and discussed. Some of them are described below.

Skyline or Variable Band (SKY)

The Skyline representation becomes popular for direct solvers especially when pivoting is not necessary. The matrix elements are stored using three arrays: *data, row_ptr, col_ind.* The array *data* stores the elements of A in a row-wise fashion; *col_ind* contains column number of the first element of each row and *row_ptr* array points to the start of every row. The SKY representation of a typical symmetric matrix A:

$$A = \begin{bmatrix} 11 & 12 & 0.0 & 16 & 0.0 \\ 12` & 13 & 14 & 17 & 19 \\ 0.0 & 14 & 15 & 0.0 & 0.0 \\ 16 & 17 & 0.0 & 18 & 20 \\ 0.0 & 19 & 0.0 & 20 & 21 \end{bmatrix}$$

Data	11	12	13	14	15	16	0.0	17	18	19	0.0	20	21

col_ind	1	1	2	1	2

row_ptr	1	2	4	6	10	13	5

This storage format stores some zero elements while other methods explained above does not.

Compressed Sparse Row (CSR)

The CSR format for sparse matrices [6, 7] is perhaps most widely used format when no sparsity structure of the matrix is

available. The CSR format is specified by *{data, row_ptr and col_ind}*. The 1D array *data* of length number of non-zero (*nnz)* contains the non-zero elements of *A* in 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 the symmetric matrix *A*:

data	11	12	13	14	15	16	17	18	19	20	21

col_ind	1	2	2	3	1	2	4	2	4	5

row_ptr	1	2	4	6	9	11

Diagonal Block Compressed Sparse Row (DBCSR)

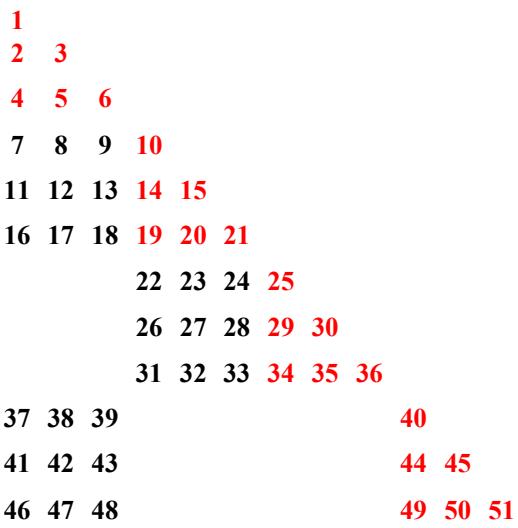
In the structural problems, the finite element discretization leads to an inherent block sparse matrix. The fixed block size 3x3 can be exploited in indexing of the sparse matrix. Knowing the location of first element of each block, remaining elements of the block can be located. The column index of the first element of each block gives the information of remaining elements. Again, in most cases, diagonal elements are treated separately, especially for preconditioning techniques. For this particular case, diagonal block matrix (symmetric part) is stored first, followed by the off-diagonal elements. Indexing for the diagonal block is not required.

In this scheme, two vectors are used to store matrix elements: *diag* for diagonal matrix as they are traversed row by row, *data* for the non-zero values of the off-diagonal elements as they are also traversed row by row. For indexing two vectors are used: *index_brow* for pointing the first element of the first block of each row, *index_bcol* for the column number of the first element of each block. The DBCSR representation of the following matrix:

Diagonal Compressed Sparse Row and Column (DCSRC)

This format is proposed here to the store asymmetric matrix (symmetric structure). Finite element discretization of thermal-convection problems leads to these types of matrices.

The block size of these matrices is 5x5 as shown in Fig. 3. The diagonal elements are stored in a separate double array: *diag.* Indexing of this part is not necessary as before. The lower triangular and upper triangular off-diagonal parts are stored in two separate double arrays: *l_data and u_data.* For indexing two vectors are used: *index_brow* for pointing the first element of each row, *index_bcol* for the column number of all off-diagonal elements of the lower triangular part. Since the matrix is symmetric structured, *index_brow and index_bcol* of lower triangular part can be used to index the elements of the upper triangular part.



diag[25]	1	2	3	4	5	6	10	14	15	19
	20	21	25	29	22	30	34	35	36	40
	44	45	49	50	51					

data[27]	7	8	9	11	12	13	16	17	18	22
	23	24	26	27	28	31	32	33	37	38
	39	41	42	43	46	47	48			

index_bcol[3]	1	3	1

index_brow[4]	1	10	19	27

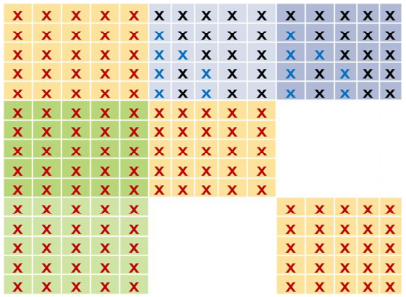


Fig 3. 5x5 Block Shaped Matrix

Diagonal Block Compress Sparse Row and Column (DBCSRC)

In this proposed format, the fixed block size shown in Fig. 1 can be exploited in indexing the sparse matrix. Memory can be saved in the indexing part. The block diagonal part is stored in two separate double arrays: *l_diag* and *u_diag*. Indexing of this part is not necessary. The lower triangular and upper triangular off-diagonal parts are stored in two separate double arrays: *l_data and u_data.* For indexing two vectors are used: *index_brow* for pointing the first element of the first block of each block row, *index_bcol* for the column number of the first element of each block. To store the matrix shown in figure, for locating all off-diagonal elements, location of the circled element is enough to know. Since the matrix is symmetric

IJSMS

structured, *index_brow and index_bcol* of lower triangular part can be used to locate the upper triangular part.

Storage Requirements (working set) for Sparse Storage Schemes

The proposed storage formats have been compared with other formats that were discussed in [5]. The storage requirement of each storage scheme describedin the previous section is given in Table-1.

Table I. Memory Requirements (Symmetric Matrix)

Storage scheme	Storage requirement (bytes)
DBCSR	*8nnz + 4(nblock_off + nbrow)*
CSR	8*nnz* + 4(*nnz + nrow + 1*)
CSC	8*nnz* + 4(*nnz + ncol + 1*)
COO	8*nnz* + 4(*nnz + nnz + 1*)
DCOO	8*nnz* + 4(*nnz-ncol)+4(nnz- nrow)*
ICSR	8*nnz* + 4*nnz*
MSR	8*nnz* + 4(*nnz-ncol) +4nrow*
VBCSR	8*nnz +4(2nblock+nrow*)
SKY	*8(nnz+*nzero) + 4(nrow) + 4(nrow)*

*nblock: number of blocks; *nzero: number of zeroes in between nonzero

Table II. Memory Requirements (Asymmetric Matrix)

Storage scheme	Storage requirement (bytes)
DBCSRC	*8x2nnz + 4(nblock_off + nbrow)*
DCSRC	*8x2nnz + 4(nnz - nrow) +4 nrow*

V. NUMERICAL RESULTS

It is remarkable that matrix generated in the finite element discretization of the elasticity and thermal-convection problems have certain fixed features by construction. For example, the finite element discretization of the elasticity problem generates the 3x3 block shaped symmetric matrix while thermal-convection problem generates 5x5 block shaped asymmetric matrix. This built-in block size is exploited here to test the computational performance of the techniques discussed above.

A. Elasticity Problem

The Adventure_Solid is a member of the Adventure system that can solve elastic and elasto-plastic problems. The present version of this module is able to solve problems of 1000 million degrees of freedom in the K-computer []. The sparse matrix storage techniques have been incorporated in the Adventure_Solid and considerable results have been found.

A.1 Model Description

The HTTR model [4] shown in Fig. 4 is used to evaluate the techniques. The model is decomposed using the open source ADVENTURE_Metis (adventure project). The number of elements and nodes are 1167263 and 5.4 million respectively.

A.2 Comparative Results

The matrices originating from these problems are symmetric. Only lower part is enough to store in the memory.

Different storing techniques have been tested. The conventional skyline method is first compared with the CSR method for different preconditioning techniques. For both diagonal scaling and BDD preconditioners CSR method shows better performance in terms of computation time and required memory. The computational performance for different storing techniques is shown in Fig. 3. The result in the Fig. 3 depicts that DBCSR is the most efficient one compare to other storing tecniques. It reduces the computation time to 60% of SKY and 80% of CSR. The requred memory per processor using different storing techniques are shown in Fig. 5. The results shown in Fig. 6 depicts that DBCSR takes the less memory compare to the other storing techniques.

A.3 Thermal-Convection Problems

ADVENTURE_sFlow is another module of Adventure System that can solve the non-stationary thermal convection problem. Non-stationary thermal-convection problems have five dof per nodal point. They are values of velocity, pressure and temperature. The FEM discretization of these types of problems results in the matrix with fixed size of block shape.

The size of the block can be exploited to store the matrix during the matrix-vector multiplication. The matrix originating from these problems are asymmetric but its structure is symmetric. So both upper and lower triangular part of the matrix are to be stored.

Table III. Comparision between Sky and Csr for Different Preconditioner

Preconditioner	Storing Techniques	#CG iter.	Time (sec)	Memory (MB)
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

B.1Analysis Example

As an analysis example, a thermal cavity problem (Fig. 7) is considered. Velocity and temperature boundary conditions are used accordingly. The total number of degrees of freedom is around one million. Computational platform is Intel Corei7-960 (3.20GHz/L2 256KB/L3 8MB/QuadCore).

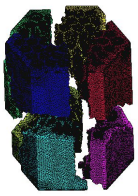


Fig 4. High Temperature Test Reactor (Httr)

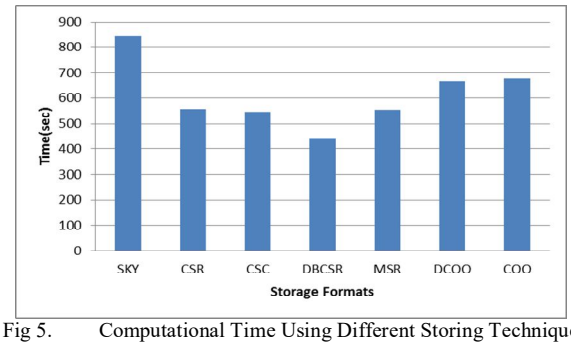


Fig 5. Computational Time Using Different Storing Technique (Symmetric)

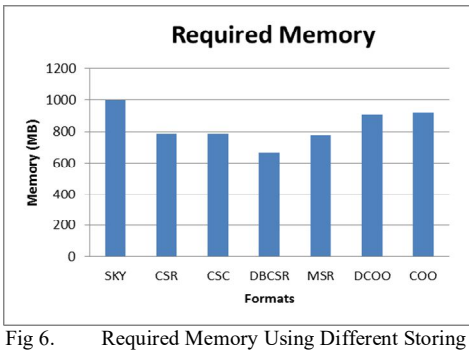


Fig 6. Required Memory Using Different Storing Technique (Symmetric)

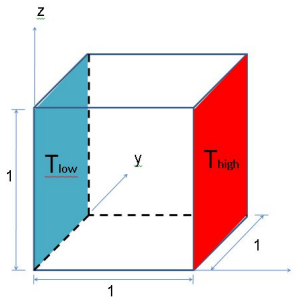


Fig 7. Cavity Problem

B.2 Comparative Performance

The DCSRC and DBCSRC are compared with the conventional skyline matrix. Results are shown in Table.

Table IV. Computational Time and Required Memory For Sky, Dcsrc And Dbcsrc

Format	Time (sec)	Memory (MB)
SKY	2877	980
DCSRC	922	495
DBCSRC	812	440

VI. CONCLUSION

Sparsity pattern of elasto-plastic and thermal-convection problem have been studied. Several modifications on compressed sparse row techniques have been proposed for both problems. Use of built-in block shape of the matrix reduces the computational time and required memory.

Treatment of the diagonal elements or diagonal block separately accelerates the techniques. The proposed method can be used for storing any block-based sparse matrices.

ACKNOWLEDGEMENT

This research is financially supported by a Japan Science and Technology (JST), CREST projects named “Development of Numerical Library Based on Hierarchical Domain Decomposition for Postpeta Scale Simulation”.

REFERENCES

[1] http://adventure.sys.t.u-tokyo.ac.jp
[2] H. Kawai, M. Ogino, R. Shioya, and S. Yoshimura, Large scale elasto-plastic analysis using domain decomposition method optimized for multi-core CPU architecture. Key. Eng. Mater. 462-463: 605-610, 2011.
[3] R. Shioya, H. Kanayama, A. M. M. Mukaddes and M. Ogino, Heat conductivity analyses with balancing domain decomposition. *Theor. Appl. Mech. Japan.* **52**: 43-53, 2003.
[4] A. M. M. Mukaddes, M. Ogino, R. Shioya, and H. Kanayama, A scalable balancing domain decomposition based preconditioner for large scale heat conduction problems, *JSME Int. J., Ser. B.* Vol. 49, pp. 533-540, 2006.
[5] A. M. M. Mukaddes, M. Ogino, and R. Shioya, Performance study of domain decomposition method with sparse matrix storage schemes in modern supercomputer, International Journal of Computational Method, 11, 1344007, DOI: 10.1142/S0219876213440076, 2014.
[6] E. Montagne, and A. Ekambaram, An optimal storage format for sparse matrices. Inf. Process. Let. **90**: 87-92, 2004.
[7] Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout, V., Pozo. R., Romine, C. and Dervorst, H.V. Templates for the solution of linear systems: building block for iterative methods. Society for Industrial and Applied Mathematics, *1994*.

IJSMS