Copyright © 2012 Tech Science Press CMES, vol.86, no.6, pp.551-576, 2012

Efficient Parallel Computing of Multifrontal Linear Solver in Block Lanczos Algorithm for Large-Scale Structural Eigenproblems

Wanil Byun[1] **and Seung Jo Kim**[2]

Abstract: A structural eigensolver for large-scale finite element analysis is developed. The algorithms and data structures implemented in this paper are well suited for a distributed memory environment. As an eigenvalue extracting algorithm, the well-known **M** orthogonal block Lanczos iteration incorporated with a parallel multifrontal solver (PMFS) was chosen. Basically, for the better performance of this algorithm in parallel computation, Lanczos vector allocation, mass matrix multiplication, and **M** inner product procedures were efficiently implemented. And the PMFS for a linear equation which is the most time-consuming part during Lanczos iterations was improved. The idea was to optimize network topologies of parallel matrix subroutines which are working in a 2-dimensional block-cyclic processor map, as well as to reduce both communication volume and idling time of parallel matrix subroutines. To reduce the communication volume, we condensed the parallel matrix multiplication subroutine from which duplicated communications are observed in the Cholesky factorization phase. To reduce the idling time, we adopted the least common multiple (LCM) concept by inverting a frontal matrix in the triangular system.

Keywords: Parallel performance, finite element method, natural frequency, block Lanczos algorithm, parallel multifrontal solver.

1 Introduction

Structural eigenvalue analysis considered in this paper is to obtain eigenvalues and eigenvectors of an eigenvalue problem composed of large, sparse, and symmetric positive definite (SPD) stiffness and mass matrix, and the eigenvalue problem can be computed by well-known finite element method (FEM). An eigenvalue problem

[1] School of Aerospace and Mechanical Engineering, Seoul National University, Seoul, Republic of Korea
[2] Korea Aerospace Research Institute, Daejeon, Republic of Korea

for structural vibrations is of the form:

$$\mathrm{K}x = \lambda \mathrm{M}x \tag{1}$$

In general, stiffness matrix **K** and mass matrix **M** are large and sparse, and eigenvalue λ and eigenvector x are directly related to the natural frequencies and the mode shapes in structural eigenvalue problems. Lanczos method has been preferred for extremal eigenvalues extraction and is widely implemented in commercial softwares such as MSC.NASTRAN[1], ABAQUS[2], and ANSYS[3]. In many engineering fields, a block Lanczos incorporated with a shift-invert transformation [Arbenz, Hetmaniuk, Lehoucq and Tuminaro (2005)] is known to be the most sophisticated eigenvalue analysis method. The benefits of blocking Lanczos vectors are that memory hierarchy of modern computer architecture can be fully utilized by using algebraic libraries such as BLAS[4] and LAPACK[5] and that it shows better convergence than single vector iteration methods [Baglama, Calvetti and Reichel (2003)]. Additionally, PBLAS and ScaLAPACK can be used in a distributed memory environment. The shift-invert transform is one of the most effective methods from the aspect of convergence rate as long as an efficient linear equation solver is incorporated with it. The shifted and inverted eigenvalue problem is as follow:

$$\frac{1}{\alpha}\mathrm{L}^T x = \mathrm{L}^T \left(\mathrm{K} - \sigma \mathrm{M}\right)^{-1} \mathrm{L}\mathrm{L}^T x \tag{2}$$

The shifted eigenvalue α is associated with original one by the relation of $\alpha = \lambda - \sigma$, and the matrix **L** is the lower triangular part of the factorized mass matrix. The presence of an inverse of $\mathrm{K} - \sigma\mathrm{M}$ indicates that a linear equation solver is required during Lanczos iterations, and the $\mathrm{L}^T x$ implies that the Lanczos basis should be **M** orthogonal. It is notable that the restarted Lanczos iteration [Calvetti, Reichel and Sorensen (1994)] has been proposed as an alternative for the shift-invert transform. However, since a generalized eigenvalue problem always requires a linear solver, the shift-invert transformation is particularly useful for solving Eq.1.

In the present research, we implemented a parallel block Lanczos eigensolver using **M** orthogonal iteration equipped with a direct linear equation solver, which is the shift-invert transformation. To accurately solve the linear equation with the coefficient matrix, $\mathrm{K} - \sigma\mathrm{M}$, a direct method [Wu and Simon (1999)] is found to be

[1] http://www.mscsoftware.com/
[2] http://www.3ds.com/products/simulia/overview/
[3] http://www.ansys.com/
[4] Basic Linear Algebra Subprograms, http://www.netlib.org/blas/
[5] Linear Algebra Package, http://www.netlib.org/lapack/

one of the reliable means. Nowadays, due to bottlenecks in scalability and memory requirement of a parallel direct solver, using iterative linear equation solvers or reduction methods are gaining attentions [Morgan and Scott (1993); Feng and Owen (1996); Benninghof and Lehoucq (2004)]. However, since the linear equation solver should be more accurate than the desired accuracy of eigenvalues, an iterative solver is less powerful even with its great parallel scalability and memory efficiency.

Parallel Lanczos libraries are available at a few libraries such as PARPACK [Maschhoff and Sorensen (1996)], BLZPACK [Marques (1995)], SLEPc [Hernández, Román, Tomás and Vidal (2006)] and TRLAN [Wu and Simon (2000)]. However, some libraries are generalized Lanczos iterators which mean they require user implementation of mass matrix multiplication and a linear equation solver. In that case, in order to build a complete eigensolver based on Lanczos iteration, one should combine it with a mass matrix multiplication routine and a linear equation solver. In general, implementation of the mass matrix multiplication depends on sparse matrix packages and the parallel linear equation solver is available in public libraries such as MUMPS [Amestoy, Duff and L'Excellent (2000)], PARDISO [Schenk and Gärtner (2004)] and SuperLU [Demmel, Gilbert and Li (1999)]. Although the libraries are black-box ones, it is apparent that each of them has been optimized in the aspect of performance [Amestoy, Guermouche, L'Excellent and Pralet (2006); Gupta and Karypis and Kumar (1997); Li and Demmel (2003); Schenk and Gärtner (2006); Wu and Simon (1999)]. However, since the Lanczos iteration procedure is deeply coupled with the mass matrix multiplication and the linear equation solver, there are many factors affecting overall performance that cannot be optimized by combining libraries which are developed and optimized independently.

The objective of this paper is to develop an optimized FEM-oriented parallel eigensolver based on the Lanczos iteration suited for a distributed memory environment in the in-house FE software, IPSAP[6]. The direct linear equation solver combined with the $\mathbf{M}$ orthogonal block Lanczos algorithm is the PMFS [Kim, Lee and Kim (2002); Kim, Lee and Kim (2005); Kim and Kim (2012)], which is one of the most suitable solvers for the FEM. Parallel computing efficiency of the developed code is enhanced with the following procedures.

- Effective implementation of the mass matrix multiplication using the Lanczos vector distribution technique
- Reducing idling time in solving the triangular system by partial inversion of the frontal matrix and adopting the LCM concept

[6] http://ipsap.snu.ac.kr/

- Reducing communication volume in the factorization phase by condensing subroutines of the parallel matrix multiplication kernel
- Searching the best topology set by tuning communication topology parameters of the subroutines

For the first two stages, we partially referred Mackay and Law (1996), Choi (1997) and Raghavan (1998) which use different linear equation solvers. What we propose is how much a structural eigensolver can be optimized and how feasible a direct solver is as a linear equation solver when it is incorporated with a Lanczos iterator in parallel environments. The four stages proposed above are associated with the following three major costs in eigenvalue extraction.

- Maintenance of **M** orthogonality in the Lanczos iterator
- Factorization of $\mathrm{K} - \sigma\mathrm{M}$
- Solving triangular systems in the Lanczos iterator

Reducing these prohibitive costs is the proposition of the present research. The accelerating technique [Sorensen (1997)] of Lanczos iterator or graph partitioning by METIS [Karypis and Kumar (1998)] for the parallel multifrontal solver is not the scope of this paper. The paper is organized as follows. In Section 2, we describe the characteristics and the costs of the **M** orthogonal block Lanczos method and the PMFS incorporated with the Lanczos iterator in the distributed memory environment parallel computing system. Section 3 presents bottlenecks in the structural eigenvalue analysis and their amelioration. Finally, we present numerical experiments and practical examples in Section 4, followed by conclusions.

2 Overview of the computational costs

In this section, the algorithms of the **M** orthogonal Lanczos iteration with the shift-invert transformation and the associated direct solver are presented, and the computational costs are estimated. For the structural eigenvalue problem in Eq. 1, the **M** orthogonal Lanczos method [Meerbergen and Scott (2000)] is given by the following form:

V_j is a matrix composed of Lanczos vectors. B_j and C_j are off-diagonal and diagonal entities of the block triangular matrix T_j in each loop. All procedures except steps 2 and 7 can be referred to as a pure Lanczos iteration. Then, it is apparent that parallelization of the pure Lanczos iteration is straightforward as long as QR factorization (step 6), mass multiplication (step 1), and inner product (step 4) are

Table 1: **M** orthogonal block Lanczos algorithm with shift-invert transformation

```
        Let $V_0$ be a set of initial vectors with $V_0^T M V_0 = I$
        $j = 0$
        while(required eigenvalue > converged eigenvalue)
step 1      $U_j = MV_j$
step 2      $(K - \sigma M) W_j = U_j$,solve for $W_j$
step 3      $W_j^* = W_j - V_{j-1} B_{j-1}^T$
step 4      $C_j = V_j^T M W_j^*$
step 5      $W_j^{**} = W_j^* - V_j C_j$
step 6      $W_j^{**} = V_{j+1} B_j$,QR factorize for $V_{j+1}$
step 7      compute eigenvalue of $T_j$, $j = j + 1$
step 8      reorthogonalize $V_{j+1}$ against $V_i$, $i = 0 \ldots j - 1$
        end
```

implemented appropriately. Reorthogonalization (step 8) of V_{j+1} is a similar procedure with step 4 and step 5. Providing B_j and C_j are shared by all processors, the operations of step 3 and step 5 can be performed in a single processor without affecting distributed manner of V_j, W_j and W_j^*. Thus, parallel effectiveness depends upon the efficiency of the QR factorization and the mass matrix multiplication. In this study, the QR factorization subroutine is parallelized using the classical Gram-Schmidt technique with reorthogonalization [Giraud, Langou and Rozloznik (2005)], which needs a modification into a block version in order to use the level 2 BLAS library as follows:

Table 2: Block version of CGS2 (classical Gram-Schmidt with reorthogonalization)

```
for $i = 0, \ldots, n-1$
    $w = {}_iW_j^{**}$
    for $k = 0, 1$
        ${}_i^{1:i-1}B_j^{(k)} = {}_{1:i-1}V_{j+1}^T M w$
        $w = w - {}_{1:i-1}V_{j+1}\,{}_i^{1:i-1}B_j^{(k)}$
    end
    ${}_i^iB_j^{(1)} = w^T M w$
    $V_{j+1}^{=} w / {}_i^iB_j^{(1)}$
end
$B_j = B_j^{(0)} + B_j^{(1)}$
```

n is the block size of the Lanczos vector V_j, and superscript and subscript in the left side denote the row and column respectively. In the QR factorization process, inter-processor communication occurs only at the mass matrix multiplication in the inner product. As a consequence, parallel performance can be enhanced only through an efficient mass matrix multiplication in the pure Lanczos iteration. The efficiency is also related to how V_j is distributed between the processors.

Step 2 in Tab. 1 is the linear equation solving procedure which takes most of the time consumed in a structural eigenvalue analysis using Lanczos iteration. As mentioned in Section 1, the PMFS was chosen as the linear equation solving subroutine. The algorithm of the solver can be regarded as an FEM-oriented version of the conventional multifrontal solver. The computational profile of the PMFS is similar with that of the conventional multifrontal algorithm in that the recursive nested dissection ordering scheme [Karypis and Kumar (1996)] is used. However, the difference is that the PMFS does not require a globally assembled stiffness matrix while the element concept of the FE mesh is utilized as graph information. Therefore, element matrices are assembled automatically during the factorization phase. The PMFS is parallelized using the well-known Cholesky factorization implemented in the subtree-subcube mapping because the element matrices are SPD in the general FE eigenvalue problem for the natural frequencies. Also, the 2-dimensional block-cyclic matrix distribution is utilized during the extend-add step. It is noteworthy that the factorization needs to be conducted only once before the Lanczos loop if the shifted value (σ) does not change. For a given FE mesh and a decomposed domain shown in Fig. 1, the parallel sparse Cholesky factorization can be understood as a post-order traverse through an elimination tree in Fig. 2. The algorithm can be simplified as follows:

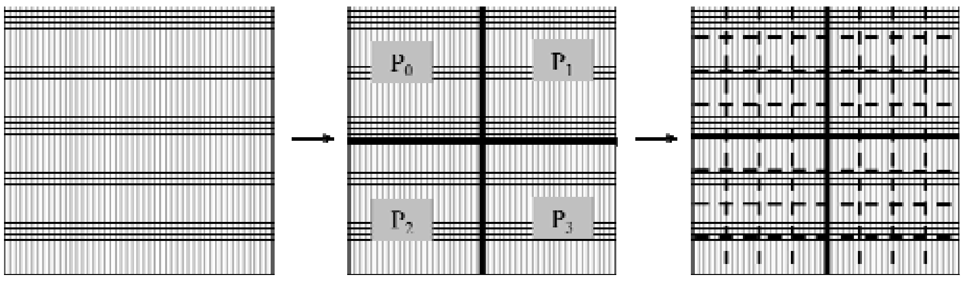


Figure 1: Partitioning domain from FE mesh and its ownership

N_p is the number of processors participating the computation and N_d is the number of domains assigned to each processor. The function *rem(a,b)* computes integer remain after dividing an integer "*a*" by another integer "*b*". It is remarkable that in

Table 3: Parallel sparse Cholesky factorization

```
   $K_f = \begin{bmatrix} K_{22} & SYM \\ K_{12} & K_{11} \end{bmatrix}$
      factorize ($K_{11}$, $K_{12}$), update ($K_{22}$)
      $j = i$
      while (rem(j,2)=1)
         extend_add ($K_f$) with another domain branch in tree
         j=(j-1)/2
      end
   end
parallel procedure
   $n = 1$
   while ($n < N_P$)
      extend_add ($K_f$) with another processor branch in tree
      factorize ($K_{11}$, $K_{12}$), update ($K_{22}$)
      $n = 2n$
   end
```

the second line, the frontal matrix K_f is formed from K - σM. Further information of the function *extend_add*(K_f) is available in the reference [Gupta, Karypis and Kumar (1997) and Kim, Lee, Kim, Joh and Lee (2003)].

In a matrix form, the Cholesky factorization, *factorize* (K_{11}, K_{12}), and the update process of a dense matrix, *update* (K_{22}), can be written by the following three steps.

$$K_{11} = L_{11}L_{11}^T \tag{3}$$

$$L_{11}K_{12}^* = K_{12} \tag{4}$$

$$K_{22}^* = K_{22} - K_{12}^{*T}K_{12}^* \tag{5}$$

Eq.3-5 are based on the assumption that the lower part of the symmetric frontal matrix is active and that the sub-matrix K_{ij} is allocated following the memory structure in Fig. 3. The feature of the frontal matrix structure in Fig. 3 is that the sub-matrix K_{22} is firstly assigned at the initial position of the allocated memory. Such structure of the frontal matrix makes the *extend_add*(K_f) operation be easily implemented because K_{22}^* remains always at the head of the allocated memory. If the *extend_add* operation is implemented appropriately, major communication overhead of the proposed algorithm is caused by parallel dense matrix operations such as *factorize* (K_{11}, K_{12}) and *update* (K_{22}).

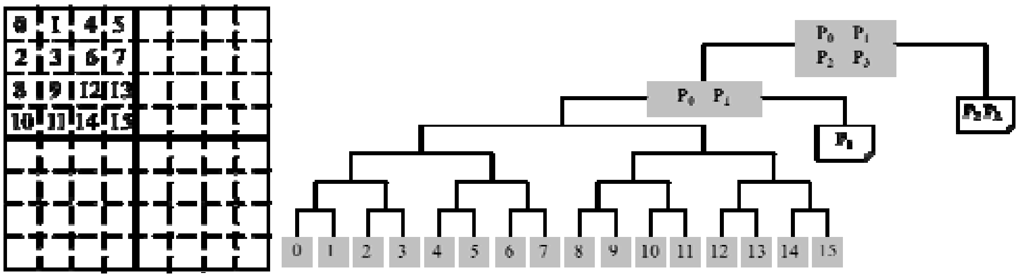


Figure 2: Domains assigned to Processor 0 and its domain-wise elimination tree

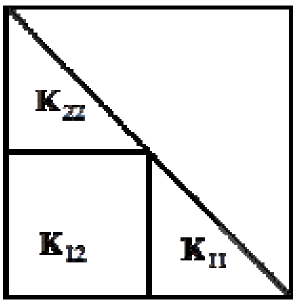


Figure 3: Memory structure of frontal matrix **K**

For solving a triangular system, a forward elimination followed by a backward substitution is a general procedure. Typically, processor mapping and frontal matrix distribution techniques used for the tridiagonal system are identical to those in the factorization phase. When the right-hand side is given as **W**, the forward elimination consists of the following two steps.

$$\mathrm{L}_{11}\mathrm{W}_1^* = \mathrm{W}_1 \tag{6}$$

$$\mathrm{W}_2^* = \mathrm{W}_2 - \mathrm{K}_{12}^{*T}\mathrm{W}_1^* \tag{7}$$

The backward substitution is also completed by the next two steps.

$$\mathrm{W}_1^{**} = \mathrm{W}_1^* - \mathrm{K}_{12}^*\mathrm{W}_2^* \tag{8}$$

$$\mathrm{L}_{11}^T\mathrm{W}_1^{***} = \mathrm{W}_1^{**} \tag{9}$$

Here, W_1^{***} is the solution of the given problem. Since the matrix is distributed throughout the processors by a 2-dimensional block-cyclic manner in Eq. 3-9, an expensive communication overhead is expected from parallel subroutines such as panel or block broadcast and summation, which is described with details in the next section. It should be noted that the concept of a block in this paragraph is different from that used for the Lanczos vectors. The main objectives for an efficient parallel computing in these routines are to reduce total communication volume and to minimize idling time during transferring panel or block matrices.

3 Overcome bottlenecks in parallel implementation

In implementing parallel computing routines, a factor that determines the overall efficiency is the communication overhead, which is defined as the difference between the parallel processor-time product and the serial run time [Gupta, Karypis and Kumar (1997)]. The communication overhead is proportional to the communication volume and is directly related to the idling time of the processors. Among the four stages in Section 1 that were proposed to enhance efficiency in parallel computing, the first and the third components are related to the communication volume while the second part is the strategy to minimize idling time. Reducing idling time is often considered as pipelining parallel operations.

In the pure Lanczos iteration defined in Section 2, the bottleneck is the mass matrix multiplication with the blocked vector V_j, W_j^* in Tab. 1 and the multiplication with the single vector ${}_i\mathrm{W}_j^{**}$ in Tab. 2. When implementing parallel routines, distributing vectors and mass matrices in the row-wise block is relatively simple although it requires a sparse or dense matrix multiplication kernel for parallel computation environment. While PBLAS and ScaLAPACK also support parallel multiplication routine, in this research, a dense matrix multiplication kernel called PLASC (Parallel Linear Algebra Subroutines in C) is developed and applied which is also used for the linear equation solver. Since the block size of V_j is small, parallel performance cannot be enhanced substantially. However in the domain level, the block size is large and efficiency can be improved significantly when the developed multiplication kernel is used for the linear equation solver. As mentioned in Section 2, the PMFS firstly divides the whole FE mesh into the number of domains assigned to each processor. Our distribution approach for the Lanczos vector also bases on the partitioned domain. In Fig. 4, the domain partitioned for four processors is presented. The symbol P_i represents the unknowns belonging to *i-th* processor that excludes unknowns at the interface (I_{ij} or I_{ijkl}).

On the interfaces such as I_{01}, I_{02}, I_{13}, I_{23} and I_{0123}, the processors share the Lanczos vectors while the mass matrix is assembled independently in a sparse matrix format within each processor. Mass matrix assembled in *i-th* domain, $\mathrm{M}^{(i)}$, is composed

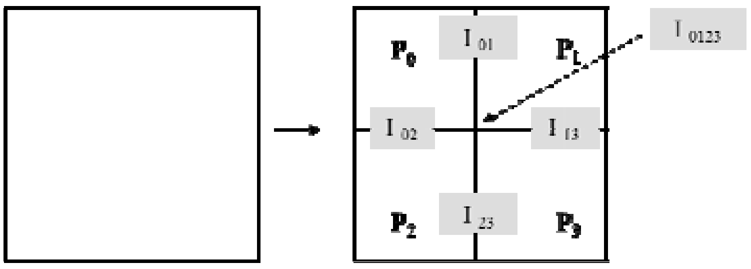


Figure 4: Domain partitioned for four processors

of three components as

$$\mathrm{M}^{(i)} = \begin{bmatrix} \mathrm{M}_D^{(i)} & \mathrm{M}_C^{(i)T} \\ \mathrm{M}_C^{(i)} & \mathrm{M}_I^{(i)} \end{bmatrix} \tag{10}$$

The subscripts D and I denote the entities belonging to the inner portion of the domain and the interface (I_{ij} or I_{ijkl}), respectively. In a single processor, the mass matrix multiplication with a blocked vector q is represented as $\mathrm{M}^{(i)}\mathrm{q}^{(i)}$, where $\mathrm{q}^{(i)}$ is a blocked vector assigned to *i-th* processor which is shared with the other domains at the interfaces. The frequently appearing mass inner products in Tab. 1 and Tab. 2 can be expressed using blocked vectors **p** and **q** for a single processor.

$$\mathrm{p}^{(i)T}\mathrm{M}^{(i)}\mathrm{q}^{(i)} = \begin{bmatrix} \mathrm{p}_D^{(i)T}, & \mathrm{p}_I^{(i)T} \end{bmatrix} \begin{bmatrix} \mathrm{M}_D^{(i)} & \mathrm{M}_C^{(i)T} \\ \mathrm{M}_C^{(i)} & \mathrm{M}_I^{(i)} \end{bmatrix} \begin{bmatrix} \mathrm{q}_D^{(i)} \\ \mathrm{q}_I^{(i)} \end{bmatrix} \tag{11}$$

Summing over the four domains in Fig. 4 yields:

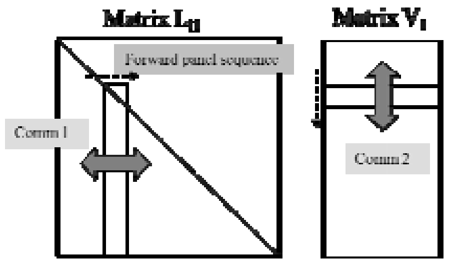
$$\sum_{i=0}^{3} \mathrm{p}^{(i)T}\mathrm{M}^{(i)}\mathrm{q}^{(i)} = \\ \sum_{i=0}^{3} \mathrm{p}_D^{(i)T}\mathrm{M}_D^{(i)}\mathrm{q}_D^{(i)} + \sum_{i=0}^{3} \mathrm{p}_I^{(i)T}\mathrm{M}_I^{(i)}\mathrm{q}_I^{(i)} + \sum_{i=0}^{3} \mathrm{p}_D^{(i)T}\mathrm{M}_C^{(i)T}\mathrm{q}_I^{(i)} + \sum_{i=0}^{3} \mathrm{p}_I^{(i)T}\mathrm{M}_C^{(i)}\mathrm{q}_D^{(i)} \tag{12}$$

The right-hand side of Eq.12 is identical to the mass inner product of vectors **p** and **q** since both $\mathrm{p}^{(i)}$ and $\mathrm{q}^{(i)}$ share the interface values across the processors. As a consequence, the summation of $(N_b \times N_b)$ matrix by calling a parallel communication (i.e. MPI_ALLREDUCE) is sufficient for a block Lanczos iteration with a block size N_b. The second case considered is the multiplication of a blocked vector

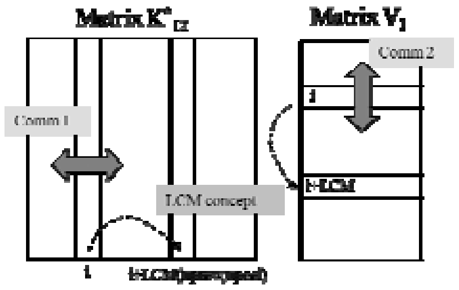
q and the mass matrix which is used at step 2 in Tab. 1. For a serial calculation, multiplication is conducted to form vectors $q_D^{*(i)}$ and $q_I^{*(i)}$ as follows:

$$\begin{bmatrix} q_D^{*(i)} \\ q_I^{*(i)} \end{bmatrix} = \begin{bmatrix} M_D^{(i)} & M_C^{(i)T} \\ M_C^{(i)} & M_I^{(i)} \end{bmatrix} \begin{bmatrix} q_D^{(i)} \\ q_I^{(i)} \end{bmatrix} \tag{13}$$

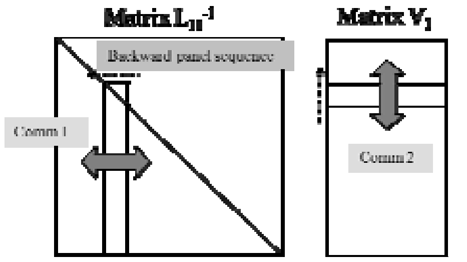
In this case, the blocked vector $q_D^{*(i)}$ is composed of fully summed values while $q_I^{*(i)}$ on the interface comprises partial values. In other words, the fully summed values of $q_I^{*(i)}$ can be obtained by summing over the processors sharing the unknowns. However, since the multifrontal solver already sums the unknowns at the forward elimination process, additional communication is not required for step 1 of Tab. 1.



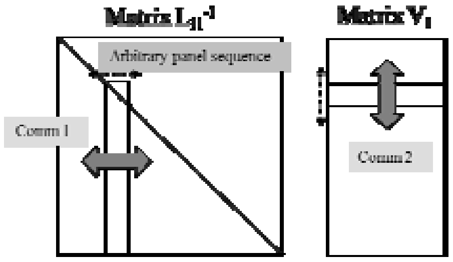
(a) Forward panel sequence of TRSM subroutine such as Eq.6



(b) Broadcasting sequence with the LCM concept of GEMM subroutine such as Eq.7 and Eq.8



(c) Backward panel sequence of TRMM subroutine such as Eq.14



(d) Arbitrary panel sequence of modified TRMM subroutine such as Eq.14

Figure 5: Broadcasting sequences for a triangular system

As for the linear equation solver, the Cholesky factorization in Tab. 3 should be conducted at least once, while the triangular system (Eq.6-9) should be solved repetitively at each iteration. In parallel computing environments, Eq.3-9 can be computed by parallel dense matrix kernels such as PBLAS, ScaLAPACK or PLASC. The first aspect to be considered is that the communication pattern in Eq.6 and Eq.9 is inefficient compared to that of Eq.7 and Eq.8. In fact, since data dependency of Eq.6 in Fig. 5-(a) enforces the sequential panel broadcast, pipelining is unavoidably disturbed. On the other hands, Eq.8 is more scalable if data-independency is utilized by appropriately profiling the broadcasting sequence with the LCM concept in Fig. 5-(b), where the LCM is the least common multiple of the processor row and column sizes in the processor map. Fig. 6 shows the panel sequences for the cases with and without LCM concept for a matrix distributed in a 2-dimensional processor map. One advantage of the LCM concept is that since panel broadcast or summation operation does not stop until the last block, idling time is minimized. In order to apply this concept to Eq.6 and Eq.9, the factorization phase should be modified such that the operation can be conducted with matrix multiplication only:

$$\mathrm{L}_{11}^{-1}\mathrm{W}_1 = \mathrm{W}_1^{*} \tag{14}$$

$$\mathrm{L}_{11}^{-T}\mathrm{W}_1^{**} = \mathrm{W}_1^{***} \tag{15}$$

Therefore, L_{11}^{-1} should be computed in the factorization phase. Once L_{11}^{-1} is computed, the triangular system can be solved using the multiplication routine. In general, W_1 and W_1^{*} share the same memory space, and the panel broadcasting pattern for computing Eq.14 becomes backward sequence as presented in Fig. 5-(c) which makes computing L_{11}^{-1} meaningless. Moreover, Eq.15 has the same problem. However, if W_1 and W_1^{**} can be stored at separate buffered arrays, the panel broadcasting pattern for computing Eq.14 and Eq.15 has arbitrary sequence as depicted in Fig. 5-(d), then the LCM concept can be finally applied in the same manner of Fig. 5-(b). Since the Lanczos block size is generally not so large, it is not difficult to allocate temporary memory space for W_1 and W_1^{**}. It is noteworthy that when the number of processor is small, benefit for the LCM concept may diminish since computing L_{11}^{-1} increases the number of floating point operations.

Four subroutines (Eq.3, Eq.4, Eq.5 and inverting L_{11}) have various block or panel communication patterns as presented in Fig. 5. However, it is apparent that some of the communications are duplicated among the subroutines. For example, Comm.2 of Fig. 7-(a) is similar to Comm.1 of Fig. 7-(b) and (d), and Comm.2 of Fig. 7-(b) is the same communication as Comm.1 of Fig. 7-(c). It means that one can develop a condensed subroutine which includes the functions of all four subroutines. If one uses small number of processors, parallel performance of the condensed subroutine will certainly be better than using the four subroutines separately because the total amount of communication is smaller. However, data-independency of Eq.5

is lost by condensing subroutines, and therefore, the LCM cannot be used. In addition, if the number of processors is large and the degree of freedom is small, merging several routines may disturb the pipeline between the processors because each processor would end up having too small frontal matrix. One should determine a criterion of frontal matrix size or a maximum number of processor that the condensed subroutine would result in better performance than the sequential execution of the four subroutines. It is notable that the criterion is dependent on the performance of each processor and the network speed. Furthermore, in case of applying the LCM concept with a small number of eigenvalues, it has a possibility to be deteriorated due to the increased floating point operation by adding a partial inverting subroutine. Therefore, for flexible application of the LCM concept and to prevent pipeline disturbance, the condensed subroutine in this research is composed of three subroutines while the inverse matrix subroutine (L_{11}^{-1}, Fig 7-(d)) is excluded.

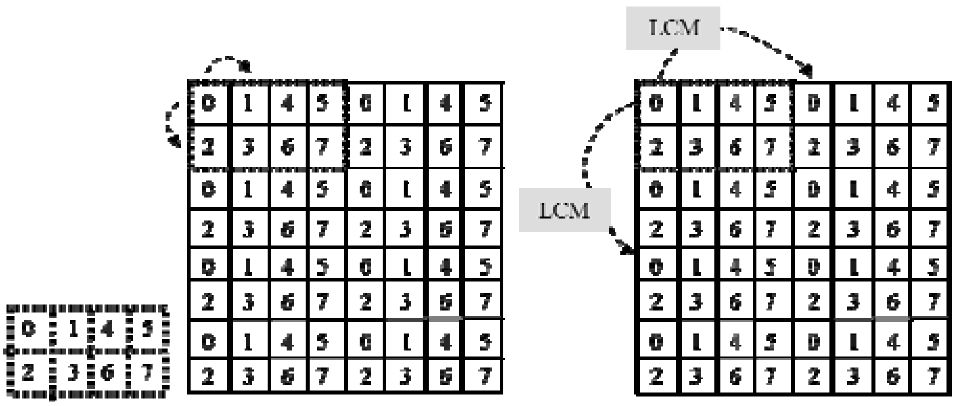


Figure 6: Panel broadcasting sequences without (left) and with (right) the LCM concept

4 Numerical test and application to structural eigenproblems

In this section, a series of numerical tests is performed to verify how much the modified methods improve parallel computing performance. The tests are also to confirm that the performance of the structural eigensolver of IPSAP enhances once the proposed approaches are used. Most time consuming part in the block Lanczos algorithm (Tab. 1) is step 2, which is the linear algebra operation part. The step 2 is composed of the Cholesky factorization routine (Tab. 3 and Eq.3-5) and the triangular system solving routine (Eq.6-9) of which Eq.6 and Eq.9 are replaced by

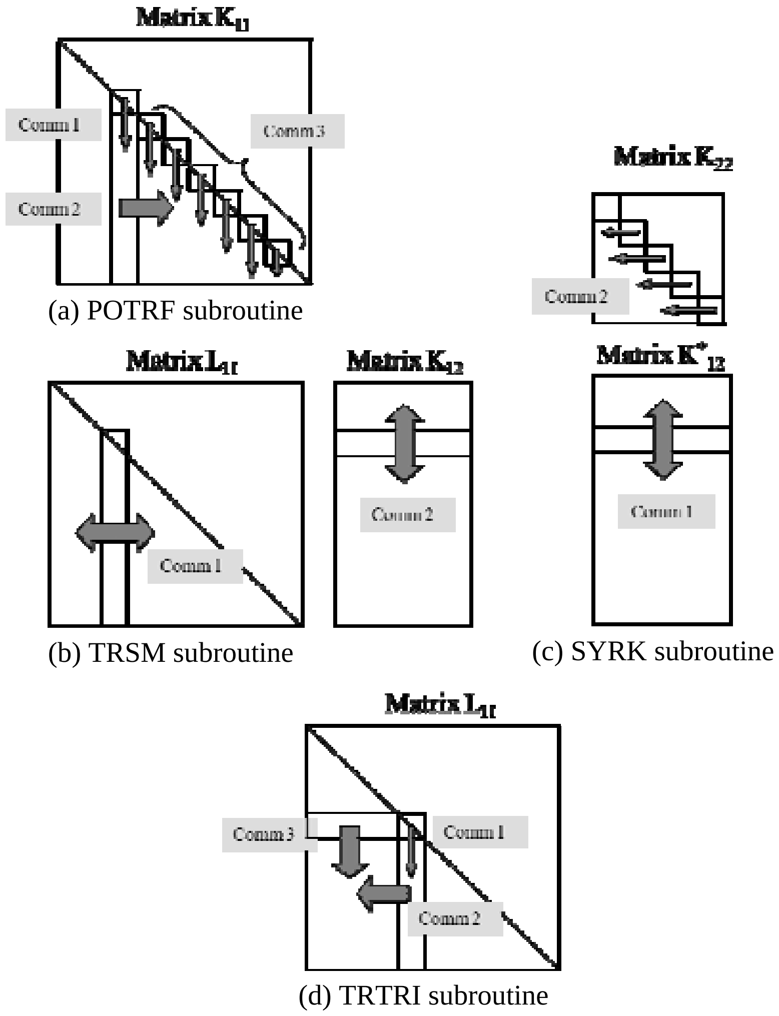


Figure 7: Communication patterns of various subroutines during the Cholesky factorization phase

Eq.14 and Eq.15 to utilize the LCM concept. For this linear equation, the eigenvalue analysis code of IPSAP applies the PMFS. For a single processor case, these matrices are generally calculated using high performance libraries such as 'TRSM', 'SYRK', 'GEMM' subroutines of BLAS and 'POTRF', 'TRTRI' subroutines of LAPACK. On the other hand, for a distributed parallel computing architecture, the solver adopts 'TRSM', 'TRMM', 'SYRK', 'GEMM', 'POTRF', 'TRTRI' subroutines of PLASC to maximize parallel computation performance. Different from other libraries, subroutines of PLASC can be modified to fit developer's intention and can be combined with an optimal network topology set suited for a distributed memory environment. Therefore, numerical tests to find the best combination of network topologies of each subroutine were involved. There are two kinds of network topology controls in the PLASC as shown in Fig. 8. One is an 'increasing ring' option that has one-way communication flow, and the other is a 'split ring' option that has two-way communication flows.

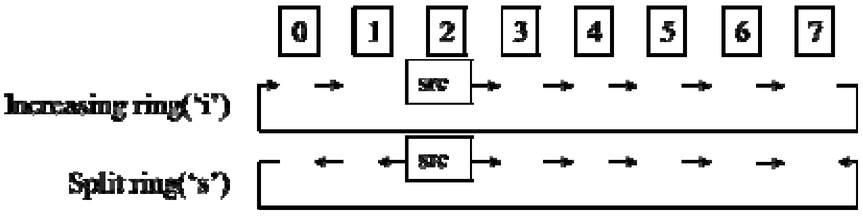


Figure 8: Topology control in panel of block communication

Table 4: Elapsed time of subroutines used in the Cholesky factorization phase

Subroutine	Network topology set (unit: sec)	
	ss	ii
POTRF	119.6	106.2
TRSM_NN	241.7	221.8
SYRK	321.0	320.6
Total elapsed time	682.3	648.6

Numerical tests and performance measurements are performed in the PEGASUS[7][7] system – a parallel computing system with the distributed memory environment. Various sets of the number of computing nodes and block sizes were tested. However, because the results show similar trends, we representatively investigated a test

[7] http://astl.snu.ac.kr/ENG/Research/pegasus01.asp

case of 32,000x32,000 matrix size calculated using 64 computing nodes (64 processors) in this study. The process map is 8x8 and the communication block size is 100. At first, the results of the 'POTRF', 'TRSM_NN[8]' and 'SYRK' subroutines that correspond to the Cholesky factorization procedure (Eq.3 to Eq.5) are shown in Tab. 4.

The network topology set of each subroutine consists of two topology options, from which 4 cases are possible. Among them, when the same topology is repeated ('ii' or 'ss'), the performance is better than mixed cases ('is' or 'si'). Also, 'ii' combination shows better computing performance than 'ss' combination in all three subroutines.

As mentioned in section 3 and Fig. 7, there are duplicated communications among 'POTRF', 'TRSM' and 'SYRK' subroutines. Therefore, a condensed subroutine 'CONDENSATION' is developed in order to minimize communication volume, and numerical test is also performed for an optimal network topology set. Since three subroutines are combined, the condensed routine now has five network topology controls. In other words, numerical tests for 32 cases should be considered (Fig. 9). Performance is usually better when all topologies are in the same option. In contrast, 'CONDENSATION' subroutine has the best performance when the combined network topology is 'issii'. Total elapsed time of the condensed subroutine (585.8sec) is found to be better than the uncondensed case in Tab. 4 (648.6sec). Performance is improved by at least 9.7%. It should be noted that the optimal topology set is sensitive to the network environment.

To apply the LCM concept during the triangular system solving phase which have the forward elimination (Eq.14 and Eq.7) and the backward substitution (Eq.8 and Eq.15), inverse of L_{11} should be calculated using 'TRTRI' subroutine after the Cholesky factorization. Therefore, numerical test of the matrix inverse routine is also performed to find an optimal network topology set (Fig. 10). According to Fig. 7-(d), 'TRTRI' is composed of three topology options (Comm.1 to Comm.3), resulting in total 8 cases. As listed in Fig. 10, 'iss' set shows the best performance as 123.3sec, while 'sss' set is also fine (123.4sec).

Results from the tests of the triangular system solving phase are presented. The left part of Tab. 5 lists elapsed time of each subroutine corresponds to Eq.6 to Eq.9, and the right part shows elapsed time of the subroutines correspond to Eq.14, Eq.7, Eq.8 and Eq.15. Each subroutine is composed of two topology options, and 'i' combination set results in the best performance except for 'TRTRI' (Fig. 10) and 'TRSM_TN' (Tab. 5) subroutines. According to Tab. 5, the shortest total elapsed time through Eq.6-7-8-9 procedure is 1,554.3sec and that through Eq.14-7-8-15

[8] 'N' means a general matrix. cf) 'T' means a transposed matrix.

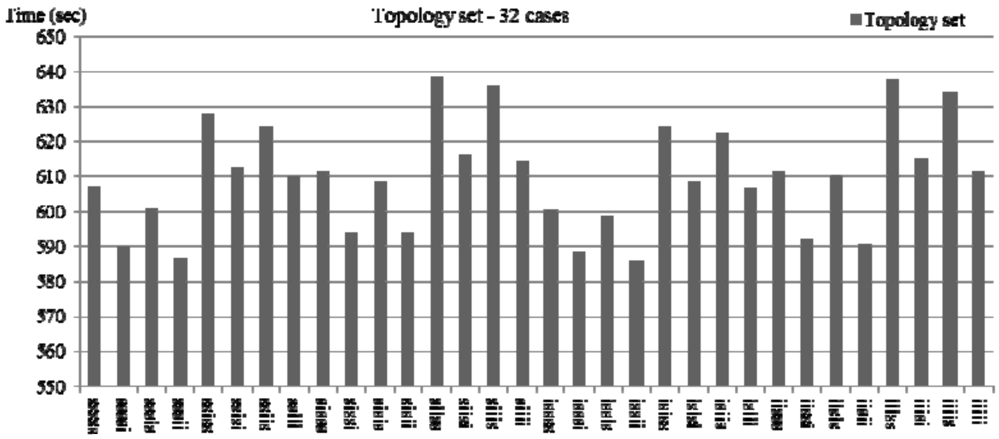


Figure 9: Elapsed time of each network topology set of the 'CONDENSATION' subroutine for the Cholesky factorization phase

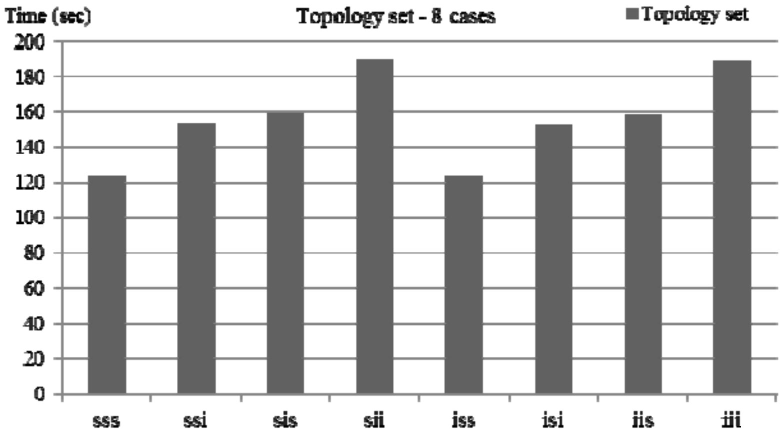


Figure 10: Elapsed time of each network topology set of the 'TRTRI' subroutine

procedure is 1,555.1sec. In other words, parallel performances for the two procedures are almost same since 'TRMM' requires the same number of floating point operations and the communication volume as 'TRSM' does. However, the LCM concept can be applied to the latter case owing to the data-independent communication pattern. The elapsed time applied the LCM concept is 1,340.6sec, and the total elapsed time (1,463.9sec) is reduced compared to the normal case (1,554.3sec) in spite of an addition of inverting 'TRTRI' process which takes 123.3sec. Since 'TRTRI' subroutine is called only once before the Lanczos iteration for a fixed σ, parallel performance will be vastly improved especially when the number of Lanczos iteration is large. On the other hand, at a small number of Lanczos iteration, performance negotiation between the computational cost for calculating L_{11}^{-1} and the efficiency upgrading from using 'TRSM' needs to be considered.

Table 5: Elapsed time of subroutines used in the triangular system solving phase

Eq.6-7-8-9	Network topology set (unit: sec)		Eq.14-7-8-15	Network topology set (unit: sec)		
	ss	ii		ss	ii	ii & LCM
TRSM_NN	241.7	221.8	TRMM_NN	246.8	230.0	202.0
GEMM_TN	592.5	577.3	GEMM_TN	592.5	577.3	498.6
GEMM_NN	384.3	369.6	GEMM_NN	384.3	369.6	340.4
TRSM_TN	385.6	412.5	TRMM_TN	388.5	378.2	299.6
Total elapsed time	1554.3		Total elapsed time	-	1555.1	1340.6

Lastly, the structural eigenvalue analyses for two different FE models are performed applying the optimized topology sets on the parallel subroutines and using the PMFS considered the results of numerical test. The first eigenvalue analysis is performed for a square plate with two different mesh sizes. The FE models composed of 8-node hexahedral (Hex8) solid elements are investigated using 64 processors, and DOFs (degree of freedom) for the two models are 10 million (10M) and 24 million (24M). Tab. 6 lists the performance results, where symbols ① and ② represent elapsed times of the Cholesky factorization phase as well as the inverting process (①) and the triangular system solver (②). Additionally, ③ indicates total time taken for the PMFS which corresponds to the total elapsed time of step 2 in Tab. 1. In Tab. 6, Case 1 is when conventional elimination-substitution with the optimized network topology set is used. From this baseline case, Case 2 adopts the 'CONDENSATION' subroutine to the Cholesky factorization phase and Case

Table 6: PMFS parallel performance of eigenvalue problems with a simple shape – a square plate model

Problem description	PMFS elapsed time (unit: sec)		
	Case 1	Case 2	Case 3
Hex8 1260 x 1260 x 1 (10M DOF) 100 eigenvalues (36 loop)	① 143.7 ② 1170.7 ③ 1320.0	① 135.5 ② 1166.3 ③ 1307.4	① 152.0 ② 909.1 ③ 1066.8
Hex8 1260 x 1260 x 1 (10M DOF) 500 eigenvalues (134 loop)	① 147.1 ② 4378.7 ③ 4531.4	① 139.7 ② 4374.5 ③ 4519.9	① 144.5 ② 3032.6 ③ 3182.6
Hex8 1260 x 1260 x 1 (10M DOF) 1,000 eigenvalues (251 loop)	① 143.2 ② 8414.7 ③ 8563.5	① 136.6 ② 8408.2 ③ 8550.4	① 143.1 ② 6215.3 ③ 6364.0
Hex8 2000 x 2000 x 1 (24M DOF) 10 eigenvalues (9 loop)	① 484.2 ② 797.1 ③ 1296.6	① 424.2 ② 791.1 ③ 1230.6	① 483.8 ② 694.1 ③ 1193.2

3 further applies the LCM concept to the triangular system.

Fig. 11 shows times consumed for the Cholesky factorization phase relative to the case when the condensed subroutine is used. Performance has been enhanced due to condensing subroutines and reducing communication volume (compare Case 1-① and Case 2-①). Also, efficiency gain is larger since 24M model has larger communication amount for each processor. Nevertheless, elapsed time of the factorization phase in Case 3 is increased because Case 3 includes inevitable matrix inverse routine to apply LCM. However, Case 3 has a room for further enhancement as the number of iteration loop increases as shown in Fig. 12. Fig. 12 shows relative time consumed for the triangular system scaled to the LCM cases. In fact, total elapsed time for solving the triangular system is reduced significantly in Case 3 (compare Case 2-② and Case 3-②). Moreover, as the number of eigenvalue increases, the performance gain becomes significant because the number of Lanczos iteration increases. For example, performance is increased by 30% for the case of calculating 500 eigenvalues for the 10M model.

FE analysis is also performed for a launch vehicle. Fig. 13 shows a FE model composed of 0.25M Hex8 solid elements of which DOF is 1.2M [and Kim, Lee, Kim, Joh and Lee (2003)]. FE analysis is performed for 100 and 1,000 eigenvalue cases using 32 and 64 processors, respectively. Tab. 7 lists the performance of the cases and Fig. 14 shows a couple of bending modes among the mode shapes. Tab. 7 shows that there is a similar trend in the performance enhancement compared to the

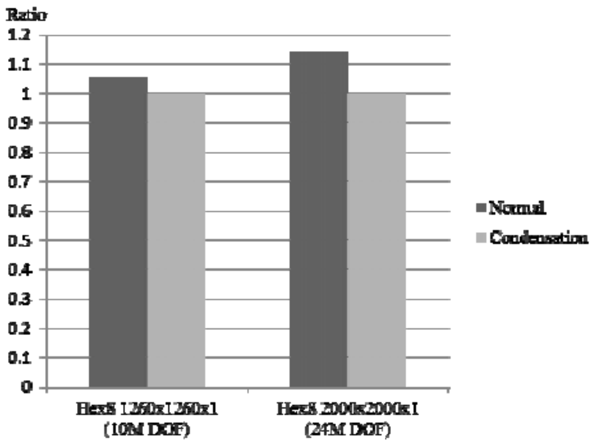


Figure 11: Elapsed time comparison of the Cholesky factorization phase of square plate models for the norml case and the case using a condensed subroutine

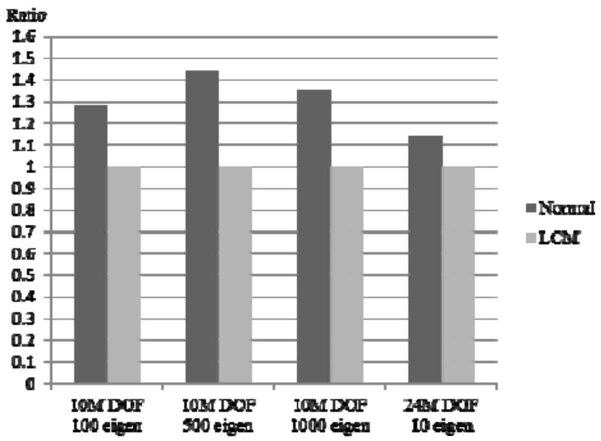


Figure 12: Elapsed time comparison of solving triangular system of square plate models

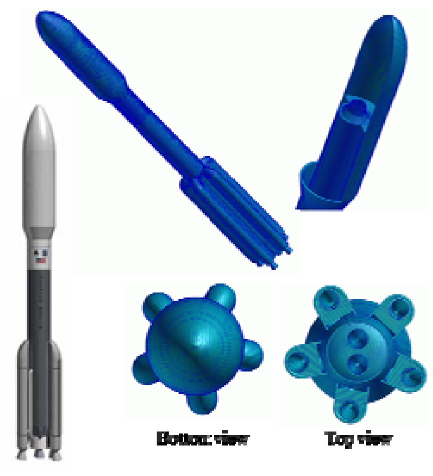


Figure 13: FE model of a launch vehicle

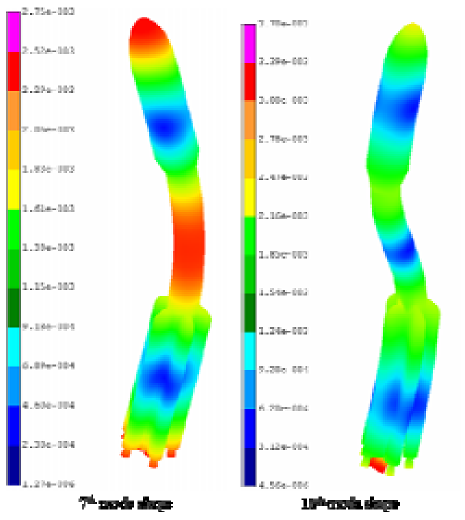


Figure 14: Some mode shapes of FE launch vehicle model

square plate model in which elapsed time of the PMFS is reduced by approximately 17%. In this particular problem, parallel efficiency of the Cholesky factorization phase is low, because the frontal matrix size per each node is small. Fig. 15 and Fig. 16 show relative time consumed for the Cholesky factorization phase and the triangular system, respectively.

Table 7: PMFS parallel performance of an eigenvalue problem with a complicated shape – a launch vehicle model

Problem description	Elapsed time (unit: sec)		
	Case 1	Case 2	Case 3
100 eigenvalues (39 loop) with 32 nodes (32 processors)	① 60.4 ② 374.9 ③ 437.1	① 55.5 ② 368.4 ③ 425.8	① 59.0 ② 311.3 ③ 372.2
1000 eigenvalues (291 loop) with 64 nodes (64 processors)	① 40.0 ② 3061.2 ③ 3102.1	① 37.1 ② 3060.4 ③ 3098.4	① 40.8 ② 2528.6 ③ 2570.3

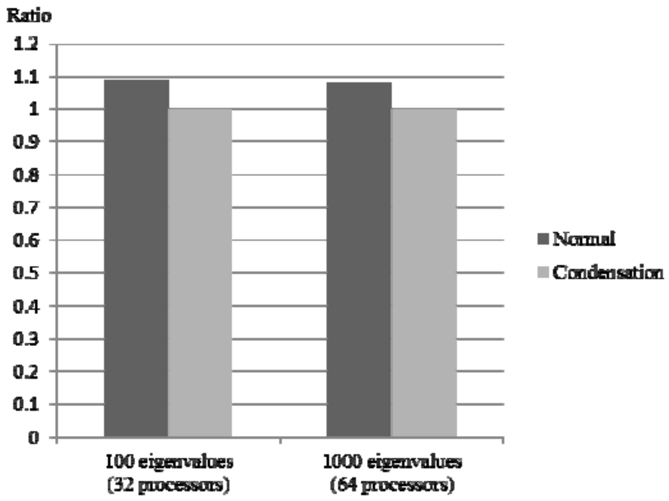


Figure 15: Elapsed time comparison of the Cholesky factorization phase of a launch vehicle model

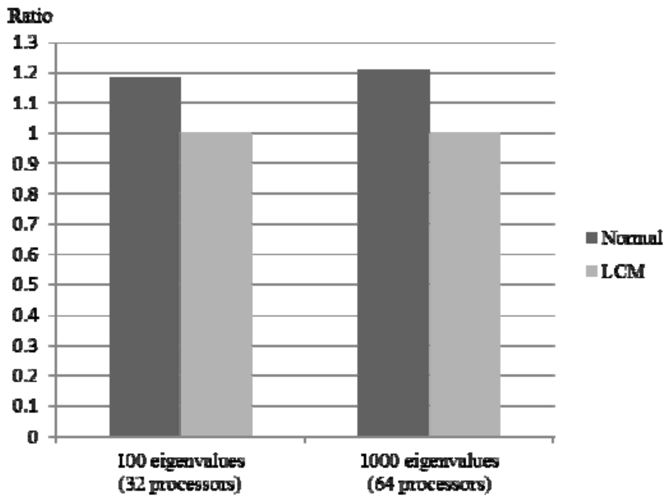


Figure 16: Elapsed time comparison of solving triangular system of a launch vehicle model

5 Conclusions

In this study, parallel performance for a structural eigensolver is enhanced in the distributed memory environment. The **M** orthogonal block Lanczos algorithm with the shift-invert transformation as the eigensolver is adopted for extracting large amount of eigenvalues. This algorithm is also incorporated with the PMFS as a linear equation solving subroutine. In general, the procedures require substantial computational costs during extracting eigenvalues are reorthogonalization, factorization ofK - σM, and solving triangular systems. For an efficient parallel computing performance, two factors are mainly concerned: reducing communication volume and reducing idling time. To reduce communication volume, Lanczos vector distribution technique is used for mass matrix multiplication and repetitive similar communication routines are condensed at the Cholesky factorization phase. Furthermore, to reduce idling time, the LCM concept based on inverting the frontal matrix is introduced in triangular system solving phase. While the proposed approaches show limited performance gain for smaller problems which uses fewer processors, small number of Lanczos iterations, or small frontal matrix size, the performance enhances significantly as the problem size increases. In fact, a set of numerical experiments and eigenvalue analyses for structural problems suggest that the parallel computing performance is improved by as much as 30% when the proposed methods are used.

Acknowledgement: This work was supported by the National Space Lab program through the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (Contract No. 2009-0092052).

References

Amestoy, P. R.; Duff, I. S.; L'Excellent, J. -Y. (2000): Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, vol. 184, no. 2-4, pp. 501-520.

Amestoy, P. R.; Guermouche, A.; L'Excellent, J. -Y.; Pralet, S. (2006): Hybrid Scheduling for the Parallel Solution of Linear Systems. *Parallel Computing*, Vol. 32, no.2, pp. 136-156.

Arbenz, P.; Hetmaniuk, U. L.; Lehoucq, R. B.; Tuminaro, R. S. (2005): A Comparison of Eigensolvers for Large-scale 3D Modal Analysis using AMG-Preconditioned Iterative Methods. *International Journal for Numerical Methods in Engineering*, vol. 64, no. 2, pp. 204-236.

Baglama, J.; Calvetti, D.; Reichel, L. (2003): IRBL: An Implicitly Restarted Block Lanczos Method for Large-Scale Hermitian Eigenproblems. *SIAM Journal* *on Scientific Computing*, vol. 24, no. 5, pp. 1650-1677.

Benninghof, J. K.; Lehoucq, R. B. (2004): An Automated Multilevel Substructuring Method for Eigenspace Computation in Linear Elastodynamics. *SIAM Journal on Scientific Computing*, vol. 25, no. 6, pp. 2084-2106.

Calvetti, D.; Reichel, L.; Sorensen, D. C. (1994): An Implicitly Restarted Lanczos Method for Large Symmetric Eigenvalue Problems. *Electronic Transaction on Numerical Analysis*, vol. 2, pp. 1-21.

Choi, J. (1997): A Fast Scalable Universal Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers. *Proceedings of the 11th International Symposium on Parallel Processing*, Switzerland, pp. 310-314.

Demmel, J. W.; Gilbert, J. R.; Li, X. S. (1999): An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 4, pp. 915-952.

Feng, Y. T.; Owen, D. R. J. (1996): Conjugate Gradient Methods for Solving the Smallest Eigenpair of Large Symmetric Eigenvalue Problems. *International Journal for Numerical Methods in Engineering*, vol. 39, pp. 2209-2229.

Giraud, L.; Langou, J.; Rozloznik, M. (2005): The Loss of Orthogonality in the Gram-Schmidt Orthogonalization Process. *Computer and Mathematics with Applications*, vol. 50, no. 7, pp. 1069-1075.

Gupta, A.; Karypis, G.; Kumar, V. (1997): A Highly Scalable Parallel Algorithm for Sparse Matrix Factorization. *IEEE Transaction on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 502-520.

Hernández, V.; Román, J. E.; Tomás, A.; Vidal, V. (2006): *Lanczos Methods in SLEPc*. SLEPc Technical Report STR-5, UPV, Spain.

Karypis, G.; Kumar, V. (1996): Parallel Multilevel Graph Partitioning. *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, HI, USA, pp. 314-319.

Karypis, G.; Kumar, V. (1998): A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359-392.

Kim, S. J.; Lee, C. S.; Kim, J. H. (2002): Large-Scale Structural Analysis by Parallel Multifrontal Solver through Internet-Based Personal Computers. *AIAA journal*, vol. 40, no. 2, pp. 359-367.

Kim, S. J.; Lee, C. S.; Kim, J. H.; Joh, M. S.; Lee, S. S. (2003): IPSAP : A High-Performance Parallel Finite Element Code for Large-Scale Structural Analysis Based on Domain-Wise Multifrontal Technique. *SC2003*, Phoenix, AX, USA, pp. 15-21.

Kim, J. H.; Lee, C. S.; Kim, S. J. (2005): High-Performance Domainwise Parallel Direct Solver for Large-Scale Structural Analysis. *AIAA journal.* vol. 43, no. 3, pp. 662-670.

Kim, M. K.; Kim, S. J. (2012): Hybrid Parallelism of Multifrontal Linear Solution Algorithm. *CMES: Computer Modeling in Engineering & Sciences*, vol. 84, no. 4, pp. 297-331.

Li, X. S.; Demmel, J. W. (2003): SuperLU DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 2, pp. 110-140.

Marques, O. A. (1995): *BLZPACK: Description and User's Guide.* Technical Report TR/PA/95/30, CERFACS, Toulouse, France.

Mackay, D. R.; Law, K. H. (1996): A Parallel Implementation of a Generalized Lanczos Procedure for Structural Dynamic Analysis. *International Journal of High Speed Computing*, vol. 8, no. 2, pp. 171-204.

Maschhoff, K. J.; Sorensen, D. C. (1996): A Portable Implementation of ARPACK for Distributed Memory Parallel Architectures, Preliminary proceedings, *Copper Mountain Conference on Iterative Methods.*

Meerbergen, K.; Scott, J. (2000): *The Design of a Block Rational Lanczos Code with Partial Reorthogonalization and Implicit Restarting.* Technical Report RAL-TR-2000-11.

Morgan, R. B.; Scott, D. S. (1993): Preconditioning the Lanczos Algorithm for Sparse Symmetric Eigenvalue Problems, *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 585-593.

Raghavan, P.; (1998): Efficient Parallel Triangular Solution Using Selective Inversion. *Parallel Processing Letters*, vol. 9, no. 1, pp. 29-40.

Schenk, O.; Gärtner, K. (2004): Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO. *Journal of Future Generation Computer Systems*, vol. 20, no. 3, pp. 475-487.

Schenk, O.; Gärtner, K. (2006): On Fast Factorization Pivoting Methods for Symmetric Indefinite Systems. *Electronic Transactions on Numerical Analysis*, vol. 23, no. 1, pp. 158-179.

Sorensen, D. C. (1992): Implicit Application of Polynomial Filters in a k-Step Arnoldi Method. SIAM Journal on Matrix Analysis and Application, vol. 13, no. 1, pp. 357-385.

Wu, K.; Simon, H. (1999): An Evaluation of the Parallel Shift-and-Invert Lanczos Method, *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, USA, pp. 2913-2919.

Wu, K.; Simon, H. (1999): A Parallel Lanczos Method for Symmetric Generalized Eigenvalue Problems. *Computing and Visualization in Science*, vol. 2, no. 1, pp. 37-46.

Wu, K.; Simon, H. (2000): Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems, *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 2, pp. 602-616