J. KSIAM Vol.13, No.1, 13-20, 2009

PERFORMANCE ENHANCEMENT OF PARALLEL MULTIFRONTAL SOLVER ON BLOCK LANCZOS METHOD

Wanil BYUN[1] AND Seung Jo KIM[1,2†]

[1]SCHOOL OF MECHANICAL AND AEROSPACE ENG, SEOUL NATIONAL UNIV, SOUTH KOREA
E-MAIL address: wibyun@aeroguy.snu.ac.kr

[2]FLIGHT VEHICLE RESEARCH CENTER, SEOUL NATIONAL UNIV, SOUTH KOREA
E-MAIL address: sjkim@snu.ac.kr

ABSTRACT. The IPSAP which is a finite element analysis program has been developed for high parallel performance computing. This program consists of various analysis modules – stress, vibration and thermal analysis module, etc. The M orthogonal block Lanczos algorithm with shift-invert transformation is used for solving eigenvalue problems in the vibration module. And the multifrontal algorithm which is one of the most efficient direct linear equation solvers is applied to factorization and triangular system solving phases in this block Lanczos iteration routine.
In this study, the performance enhancement procedures of the IPSAP are composed of the following stages: 1) communication volume minimization of the factorization phase by modifying parallel matrix subroutines. 2) idling time minimization in triangular system solving phase by partial inverse of the frontal matrix and the LCM (least common multiple) concept.

1. INTRODUCTION

Structural vibration analysis considered in IPSAP is to obtain eigenvalues and eigenvectors of the eigen problem of symmetric positive semi-definite stiffness and mass matrix which are obtained by the finite element method. The eigenvalue problem of structural vibration is of the form:

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

Since the stiffness and the mass matrix are symmetric and semi-definite, the well-known Lanczos iteration is applicable to Equation (1). Exactly speaking, the shifted invert transformation as in Equation (2) should be used in presence of the mass matrix M.

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

The shifted eigenvalue α is associated with original one by the relation $\alpha = \lambda - \sigma$.

Received by the editors December 29 2008; Accepted February 20 2009.
2000 *Mathematics Subject Classification*. 68W10, 68W40.
Key words and phrase : Block Lanczos Method, Parallel Performance Enhancement, Eigenvalue Problem, Mutifrontal algorithm, IPSAP.
† Corresponding author.

The matrix L is the lower triangular matrix, where $M=LL^T$. The existence of the inverse of $K-\sigma M$ requires a linear equation solver during Lanczos iterations. The presence of $L^T\phi$ implies that the Lanczos basis should be M orthogonal. It is notable that the restarted Lanczos iteration [1] has been proposed as an alternative for the shift-invert transform.

2. ALGORITHM

In the shift-invert Lanczos method, a linear equation solver is needed to solve accurately a linear equation with the coefficient matrix $K-\sigma M$. One of the reliable means is a direct method [2]. Nowadays, due to the bottleneck in scalability and memory requirement of a parallel direct solver, iterative linear equation solvers or reduction methods are getting interest [3-4]. However, since the linear equation solver should be more accurate than the desired accuracy of eigenvalues, iterative solvers may be less powerful even with its great parallel scalability and memory efficiency.

In the IPSAP solver, we implemented a parallel block Lanczos eigensolver using M orthogonal iteration equipped with a direct linear equation solver (Algorithm 2.1).

Algorithm 2.1 *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)	
$U_j = MV_j$	step 1
$(K-\sigma M)W_j = U_j$, solve for W_j	step 2
$W_j^* = W_j - V_{j-1}B_{j-1}^T$	step 3
$C_j = V_j^T M W_j^*$	step 4
$W_j^{**} = W_j^* - V_j C_j$	step 5
$W_j^{**} = V_{j+1}B_j$, QR factorize for V_{j+1}	step 6
compute eigenvalue of T_j , $j = j+1$	step 7
re-orthogonalize V_{j+1} against V_i , $i = 0 \cdots j-1$	step 8
end	

Step 2 in Algorithm 2.1 is a linear equation solving procedure which takes most of the time consumed in structural vibration analysis using Lanczos iteration. The parallel multifrontal solver was chosen as a linear equation solver. The algorithm of the solver can be regarded as a FEM-oriented version of a conventional multifrontal solver. This solver does not require a globally assembled stiffness matrix while the element concept of the finite element mesh is utilized as graph information. Therefore, the assembly of element matrices takes place automatically during the factorization phase. The implemented parallel factorization in the multifrontal solver is the well-known Cholesky factorization

(Algorithm 2.2). It is noteworthy that the factorization needs to be conducted only once before the Lanczos loop if the shift is not varied.

Algorithm 2.2 *Parallel sparse Cholesky factorization*

```
for i = 0, ··· , N_d − 1
    K_f = [K_22  SYM; K_12  K_11]
    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
        factorize(K_11, K_12), update(K_22)
        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
```

In a matrix form, the Cholesky factorization $\mathrm{factorize}(\mathrm{K}_{11}, \mathrm{K}_{12})$ and the update of a dense matrix $\mathrm{update}(\mathrm{K}_{22})$ can be written in three steps.

$$\mathrm{K}_{11} = \mathrm{L}_{11}\mathrm{L}_{11}^{\mathrm{T}} \tag{3}$$

$$\mathrm{L}_{11}\mathrm{K}_{12}^{*} = \mathrm{K}_{12} \tag{4}$$

$$\mathrm{K}_{22}^{*} = \mathrm{K}_{22} - \mathrm{K}_{12}^{*\mathrm{T}}\mathrm{K}_{12}^{*} \tag{5}$$

Equation (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 according to the memory structure in Figure 1. The feature of the frontal matrix structure in Figure 1 is that the sub-matrix K_{22} is firstly assigned at the initial position of the allocated memory. Such a 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, the major communication overhead of the proposed algorithm is caused by parallel dense matrix operation such as $\text{factorize}(K_{11}, K_{12})$ and $\text{update}(K_{22})$.

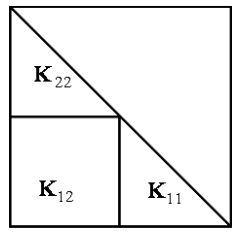

FIGURE 1. Memory structure of frontal matrix K

During the phase of solving triangular systems, a forward elimination followed by a backward substitution is a general procedure. They use the same processor mapping and frontal matrix distribution of the factorization phase. If the RHS (right-hand side) is indicated by W, the forward elimination is generally composed of two steps as follows.

$$L_{11}W_1^* = W_1 \tag{6}$$

$$W_2^* = W_2 - K_{12}^{*T}W_1^* \tag{7}$$

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

$$W_1^{**} = W_1^* - K_{12}^* W_2^* \tag{8}$$

$$L_{11}^T W_1^{***} = W_1^{**} \tag{9}$$

Here, W_1^{***} is the solution we are looking for. There are many communication overheads occurring from panel or block broadcast and summation in parallel operation of Equation (3)-(9). The main objectives of performance enhancement in these routines are to reduce the total communication volume and to minimize the idling time for transferring panel or block matrices.

3. PARALLEL PERFORMANCE ENHANCEMENT AND NUMERICAL TEST

Performance tuning of the multifrontal solver is conducted from three points of views.

The first one is to reduce communication occurring in the Cholesky factorization. It is notable that the naming convention of subroutines used in the eigensolver is based on those of BLAS (Basic Linear Algebra Subprograms)[5] and LAPACK (Linear Algebra Package) [6]. As shown in Figure 2, the symmetric frontal matrix is composed of three matrix entities. Therefore, three routines are required to factorize the frontal matrix. In parallel matrix operations based on panel communication which are adopted in the present research, each routine performs panel or block communications which are column(or row) broadcasting or reducing type. Considering the detailed communication pattern, it is apparent that some parts of broadcasting or reducing are duplicated. For example, column broadcasting of panels of K_{11} in POTRF is also present in TRSM (Figure 3). If combining

three routines into one is possible, duplicated communications will be avoidable. In Figure 3, they show a communication pattern of one condensation subroutine by combining duplicated communications among three subroutines.

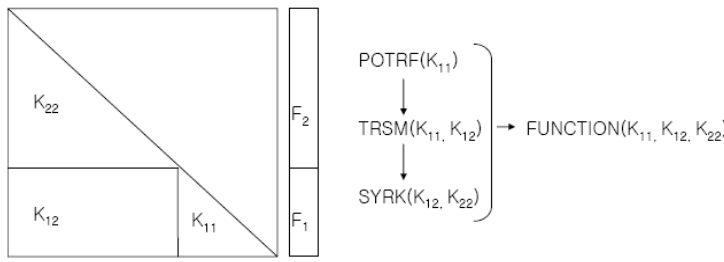

FIGURE 2. Frontal matrix and factorization routines

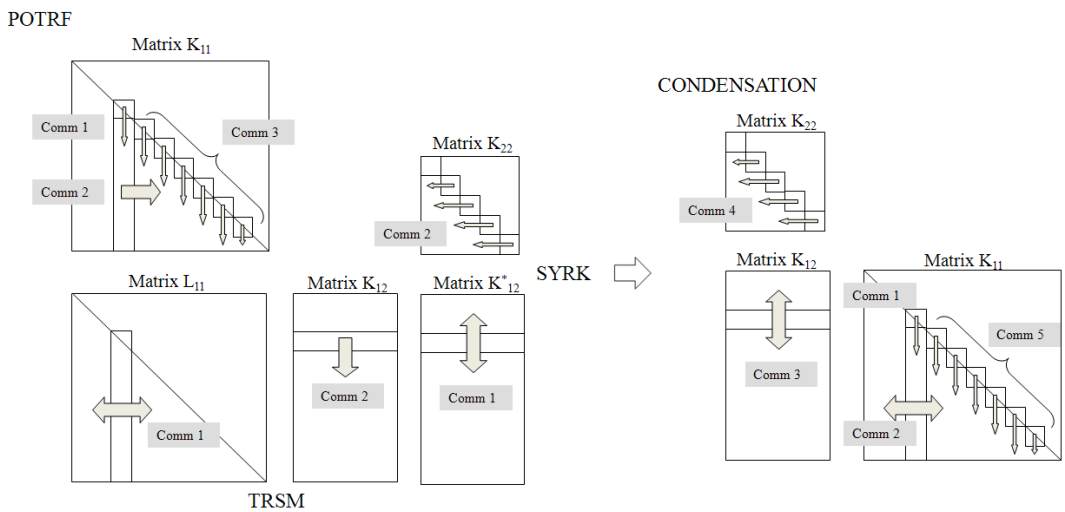

FIGURE 3. Communication pattern of each subroutine

Table 1 lists some numerical test comparison without condensation and with condensation. The condensation subroutine which combines duplicated communications between nodes and nodes takes less time compared with non-condensation one.

TABLE 1. Numerical test comparison without condensation and with condensation

process map=4x8, block size=100 with 32 nodes (32 CPUs)			
matrix dimension	without condensation	with condensation	difference
16000x16000	209.064 sec	201.258 sec	-7.806 sec
process map=8x8, block size=100 with 64 nodes (64 CPUs)			
matrix dimension	without condensation	with condensation	difference
16000x16000	129.261 sec	122.929 sec	-6.332 sec
32000x32000	1018.96 sec	816.922 sec	-202.038 sec

The second performance tuning can be conducted by topology control in panel or block communication. We implemented two kinds of topology which are ‘increasing ring’ and ‘split ring’. The condensation subroutine is composed of five topology options. Therefore, there is 32 topology sets. When the topology set is ‘issii’, the best performance in our parallel environment is achieved (Table 2). It is notable that the best topology set depends on own network environment.

TABLE 2. Topology comparison of the condensation subroutine

Topology Sets – 32 cases							
sssss	ssssi	sssis	sssii	ssiss	ssisi	ssiis	ssiii
sisss	sissi	sisis	sisii	siiss	siisi	siiis	siiii
issss	isssi	issis	issii	isiss	isisi	isiis	Isiii
iisss	iissi	iisis	iisii	iiiss	iiisi	iiiis	iiiii
Matrix dimension = 16000x16000, process map = 8x8, block size = 100 with 64 nodes (64 CPUs)							
115.432	111.501	116.717	111.242	120.286	113.605	120.819	112.83
120.961	111.241	119.621	113.135	124.059	114.537	122.407	114.573
115.825	110.642	117.704	109.211	120.259	112.433	120.02	111.096
120.185	111.083	121.129	111.52	123.041	113.296	124.083	113.459
Matrix dimension = 32000x32000, process map = 8x8, block size = 100 with 64 nodes (64 CPUs)							
607.318	589.898	600.976	586.587	627.982	612.927	624.301	609.839
611.379	593.751	608.472	594.00	638.387	616.156	635.957	614.28
600.697	588.547	598.778	585.828	624.406	608.557	622.545	606.83
611.518	591.975	610.131	590.734	637.907	615.17	633.893	611.836

The last one is to tune the triangular system solving routines. Operations involved in solving the triangular system are generally composed of TRSM and GEMM with F_1 and F_2. Since TRSM routine has data-dependent algorithmic flow, there may be idle processors between panel broadcasting. One idea to resolve this bottleneck is converting TRSM into TRMM by computing inverse of L_{11} by using TRTRI routine. Although TRMM requires the same number of floating point operations and communication volume as TRSM does, the LCM (least common multiple) concept can apparently reduce the idling time. The LCM concept which is originally proposed in the research of GEMM [7] can also be applied to any case of data-independent communication pattern. In order to apply such a concept to (6) and (9), the factorization phase should be modified so that the operation can be conducted only with matrix multiplication as follows.

$$L_{11}^{-1}W_1 = W_1^* \tag{10}$$

$$L_{11}^{-T}W_1^{**} = W_1^{***} \tag{11}$$

In this approach, performance negotiation between computing time for inverse of L_{11} and gains by using TRSM should be taken into account.

TABLE 3. Performance results of eigenvalue problem (32 CPUs)

	Case 1	Case 2	Case 3	Difference (Case 1 ③ - Case 3 ③)
Hex8 1260 x 1260 x 1 (10M DOF) 10 eigenvalues (9 loop)	① 239.119 sec ② 266.858 sec ③ 517.371 sec	① 227.354 sec ② 267.217 sec ③ 505.865 sec	① 239.676 sec ② 225.588 sec ③ 476.549 sec	- 40.822 sec (↓ 7.89%)
Hex8 1260 x 1260 x 1 (10M DOF) 50 eigenvalues (22 loop)	① 239.066 sec ② 648.929 sec ③ 899.302 sec	① 225.603 sec ② 642.754 sec ③ 879.649 sec	① 237.000 sec ② 521.791 sec ③ 770.086 sec	- 129.217 sec (↓ 14.4%)

TABLE 4. Performance results of eigenvalue problem (64 CPUs)

	Case 1	Case 2	Case 3	Difference (Case 1 ③ - Case 3 ③)
Hex8 2000 x 2000 x 1 (24M DOF) 10 eigenvalues (9 loop)	① 484.178 sec ② 797.078 sec ③ 1296.588 sec	① 424.165 sec ② 791.127 sec ③ 1230.586 sec	① 483.794 sec ② 694.068 sec ③ 1193.168 sec	- 103.420 sec (↓ 7.98%)
Hex8 1260 x 1260 x 1 (10M DOF) 100 eigenvalues (36 loop)	① 143.696 sec ② 1170.720 sec ③ 1320.015 sec	① 135.516 sec ② 1166.260 sec ③ 1307.391 sec	① 152.044 sec ② 909.117 sec ③ 1066.752 sec	- 253.263 sec (↓ 19.2%)
Hex8 1260 x 1260 x 1 (10M DOF) 500 eigenvalues (134 loop)	① 147.117 sec ② 4378.680 sec ③ 4531.387 sec	① 139.744 sec ② 4374.520 sec ③ 4519.864 sec	① 144.483 sec ② 3032.570 sec ③ 3182.611 sec	- 1348.776 sec (↓ 29.8%)
Hex8 1260 x 1260 x 1 (10M DOF) 1000 eigenvalues (251 loop)	① 143.202 sec ② 8414.730 sec ③ 8563.521 sec	① 136.644 sec ② 8408.190 sec ③ 8550.448 sec	① 143.147 sec ② 6215.300 sec ③ 6364.019 sec	- 2199.502 sec (↓ 25.7 %)

The eigenvalue analysis of simple finite element models was performed considering the three performance enhancement issues. Table 3 and 4 list the performance results in 32 and 64 CPUs parallel computing environment. Case 1 is for normal elimination-substitution algorithm with optimized network topology, Case 2 is for a condensation based on Case 1 and Case 3 is for partial inverting algorithm with LCM concept based on case 2. Then, ① lists the Cholesky factorization time, ② lists Triangular solution time and ③ lists MFS

elapsed time. The results of the Cholesky factorization time show performance enhancement compares with Case 1 and Case 2. And the Cholesky factorization time is similar between Case 1 and Case 3 because of condensation in spite of adding inverting routine TRTRI. In a view point of triangular system solving time, Case 3 has a good performance enhancement if the iteration loop is more and more.

4. CONCLUSION

Parallel performance tuning of the multifrontal algorithm in block Lanczos method was conducted from three points of views in this research. Using condensation to reduce communication volume results in better performance during Cholesky factorization phase. In a viewpoint of topology, a network topology optimization also needs to have a better performance. The best topology set is dependent on network structure and device type. In this research, about 8~12% enhancement can be obtained by topology control and condensation in case of parallel computing environment (64 CPUs). At last, the factorization time increases when applying the LCM concept because inverse subroutine (TRTRI) is added. However, if the number of Lanczos iteration is large, a partial inverting algorithm is more efficient and shows good performance. About 8~25% enhancement in this research can be obtained by using LCM concept due to the data-independent communication. Applying the LCM concept inside an iteration loop shows that it is significant for reducing the computing time in parallel matrix operations.

ACKNOWLEDGMENTS

This research has been partially supported by the ‘Rapid Design of Satellite Structures by Multi-disciplinary Design Optimization’ project from Aerospace Research Institute(KARI) and NRL program administered via the Institute of Advanced Aerospace Technology at Seoul National University.

REFERENCES

[1] D. Calvetti, L. Reichel and D. Sorensen, “An Implicit Restarted Lanczos Method for Large Symmetric Eigenvalue Problems,” *Electronic Transaction on Numerical Analysis*, Vol. 2, pp. 1-21, 1994

[2] K. Wu and H. Simon, “An Evaluation of the Parallel Shift-and-Invert Lanczos Method,” *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 2913-2919, Las Vegas USA, June 1999

[3] R. Morgan and D. Scott, “Preconditioning the Lanczos Algorithm for Sparse Symmetric Eigenvalue Problems,” *SIAM Journal on Scientific Computing*, Vol. 14, pp. 585-593, 1993

[4] Y.T. Feng and D.R.J. Owen, “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, 1996

[5] http://www.netlib.org/blas

[6] http://www.netlib.org/lapack

[7] Choi, J., “A Fast Scalable Universal Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers”, *Proceedings of the IPPS*, pp. 310-314, 1997