한국전산구조공학회
2004년도 봄 학술발표회 논문집

전-후 처리 과정을 포함한 거대 구조물의 유한요소 해석을 위한 효율적 데이터 구조

Efficient Data Management for Finite Element Analysis with Pre-Post Processing of Large Structures

박 시 형* 박 진 우* 윤 태 호** 김 승 조***
Park, Si Hyoung Park, Jin Woo Yoon Tae Ho Kim Seung Jo

ABSTRACT

We consider the interface between the parallel distributed memory multifrontal solver and the finite element method. We give in detail the requirement and the data structure of parallel FEM interface which includes the element data and the node array. The full procedures of solving a large scale structural problem are assumed to have pre-post processors, of which algorithm is not considered in this paper. The main advantage of implementing the parallel FEM interface is shown up in the case that we use a distributed memory system with a large number of processors to solve a very large scale problem. The memory efficiency and the performance effect are examined by analyzing some examples on the Pegasus cluster system.

1. 서 론

물리 현상을 수치적으로 해석하기 위하여 이용되는 방법으로 유한요소법(finite element method)이 있으며 그 해의 정확도는 사용한 요소의 개수와 밀접한 관련이 있다. 또한 유한요소법의 효율적 수행을 위하여 행렬 형태의 선형 방정식을 다룰 수 있는 해법(solver)이 필요하며 이는 유한요소법의 계산 속도에 결정적인 영향을 주게 된다. 최근 들어 공학에서 다루고자 하는 물리 현상의 복잡성이 증가하고 더욱 정확한 결과를 얻기 위하여 거대한 자유도 문제에 대한 유한요소법의 적용이 많은 관심을 모으고 있으며 이는 거대 행렬 방정식의 해법을 필요로 하게 되며, 또한 고성능 시스템의 발달과도 밀접한 관련이 있다. 최근에 대두되고 있는 High Performance Computing과 최소의 비용으로 최대의 효율을 얻을 수 있는 병렬 클러스터의 발달은 이러한 고성능 시스템과 병렬 해법에 대한 관심을 반영한 것이다.

클러스터 시스템을 기반으로 수행되는 수치 해석의 성능은 하드웨어적 최적화에 의해 많은 영향을 받으며, 클러스터 시스템의 하드웨어 성능 최적화는 크게 단위 노드 시스템의 성능 최적화, 네트워크 시스템의 성능 최적화로 나누어질 수 있다. 해석 알고리즘 측면에서의 최적화는 클러스터 시스템에 적합한 수치 연산 라이브러리의 최적화와 개발 프로그램의 병렬성으로 구분할 수 있으며, 프로그램 개발자의 입장에서는 병렬성 최

* 서울대학교 기계항공공학부 박사후과정
** 서울대학교 기계항공공학부 박사과정
*** 서울대학교 기계항공공학부 교수

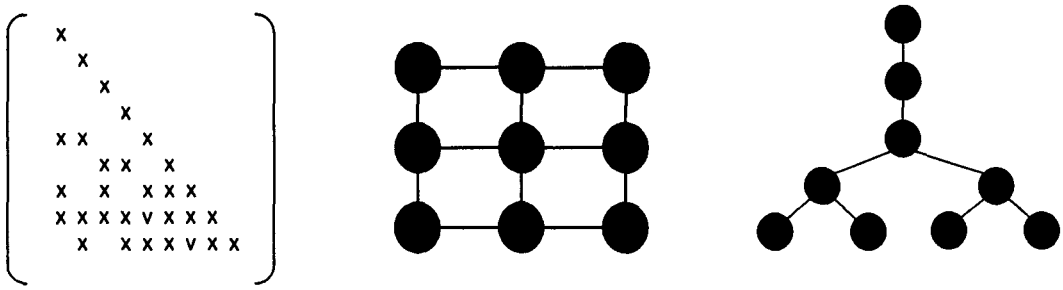
적화 측면으로 접근하는 것이 일반적이다. 앞서 언급한 바와 같이 유한요소법을 이용한 구조 해석의 핵심 기술에는 행렬 방정식의 해법이 자리 잡고 있으며 거대 행렬 방정식의 병렬 해법(parallel solver)은 성긴(sparse) 행렬에 대하여 집중적으로 연구되어 왔다[1][2][3]. 또한 병렬 해법의 발전과 더불어 최적화된 수치 연산 라이브러리의 개발이 동반되어 거대 문제 의 병렬 해석에 소요되는 시간은 핵심(core) 계산 시간 이외에 자료 입출력 또는 전-후처리 시간이 큰 비율을 차지하게 되었으며 속도 문제를 넘어서 메모리(memory)의 한계 문제에 도달하게 되는 경우가 많다. 본 연구에서는 직접(direct) 해법으로써 각광받고 있는 다중프론트(multifrontal) 해법을 이용하여 거대 구조물의 유한요소 해석에 적용하고, 적용 과정에서 발생하는 메모리 문제에 초점을 맞춘다. 수치해법과 유한요소법 사이의 인터페이스를 적절한 자료 구조로 저장함으로써 메모리 운용을 극도로 하고 이를 통하여 거대 자유도 구조 문제의 해법을 도출한다.

2. 병렬 유한요소 해석

구조 문제에 대하여 유한요소법을 적용하였을 경우에 생성되는 강성 행렬은 띠(band) 행렬 또는 성긴 행렬의 형태를 가지고 있다. 성긴 행렬로 이루어진 방정식의 병렬 해법은 직접 해법과 반복 해법으로 나눌 수 있으며 직접 해법 가운데에서 병렬 효율이 가장 높은 것으로 알려져 있는 것이 다중프론트 해법이다.

2.1. 다중 프론트 해법

다중 프론트 해법은 1983년을 시작으로 하여 현재까지 알려져 있는 직접 해법 가운데 그 효율을 가장 높게 평가받고 있으며 ABAQUS 등의 상용 소프트웨어에서 채택하고 있는 병렬 해법이다. 또한 현재 개발되고 있는 인터넷 상의 병렬 프로그램으로는 Super_LU[4], MUMPS[5] 등이 있다. 다중 프론트 해법의 기본적인 흐름은 그림 1과 같이 성긴 행렬의 데이터 배치 구조를 연결되어 있는 형태의 요소로 표현하고 이를 통하여 소거 순서를 정한 후, 정해진 소거 순서에 따라 분해(factorization)를 수행하게 된다. 그림 1-(c)에서 알 수 있듯이 소거 순서에 따라 서로 독립적인 계산이 가능하게 되므로 이를 통하여 병렬화 알고리즘이 구현될 수 있다. 다중 프론트 해법에서 소거 순서를 정하는 방법은 fill-in의 개수에 의하여 메모리 관리에 매우 중요한 요소로 작용하게 되는데, 이러한 방법으로는 recursive nested dissection 또는 multiple, local minimum degree 등의 기법이 있다. 본 연구에서 이용한 다중 프론트 해법는 일반적인 방법과는 조금 다르게 유한요소 격자의 그래프 변환 및 영역 분할을 통하여 소거 순서가 자동으로 정해지게 되며 이러한 특성은 다중프론트 해법을 유한요소 해석이라는 특정 분야에 적용할 때 가능한 것이다.

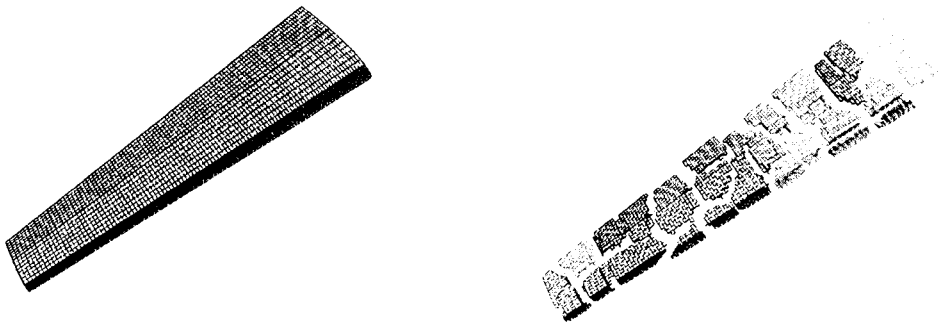


(a) sparse matrix (b) general element presentation (c) elimination tree

그림 1. Multifrontal solver 의 기본 과정

2.2. 유한요소 인터페이스 구조

유한요소법은 기본적으로 전처리(pre-process) 과정을 거치게 되며 전처리 과정은 기본적으로 격자 또는 요소 생성 과정을 의미한다. 요소 생성은 해석 공간상에 노드(node)를 배치하고 노드들간의 적절한 연결을 통하여 요소망을 만들어내는 과정이다. 이러한 과정은 다중프론트 해법의 그림 1-(b)와 유사하며 이러한 특성으로 인하여 다중프론트 해법과 유한요소 해석의 결합은 매우 용이하다. 또한 생성된 유한요소들 간의 상대적 인접(adjacency) 상태를 그래프 구조로 변환하고 그래프 구조의 분할 기법을 통하여 영역 분할을 수행할 수 있으며 분할된 영역과 영역의 색인(index)을 이용하면 소거 순서를 결정할 수 있다. 본 연구에서는 유한요소 생성 및 영역 분할 과정을 모두 포함하여 전처리 과정으로 정의하였으며 이러한 과정이 정상적으로 수행된 상태에서의 유한요소 인터페이스 구조를 고려하였다. 아래의 그림 2는 항공기 날개 구조물을 6면체 고체 요소로 요소생성을 수행하고 이를 영역 분할한 예제이다.



(a) finite element mesh of wing structure (b) partitioned domain

그림 2. 항공기 날개 구조의 요소 생성 및 영역 분할

유한요소법에서의 후처리(post-process)과정은 일반적으로 응력복원(stress recovery), 오차예측(error estimation) 등을 의미하며 이러한 과정은 국소적인 연산을 바탕으로 하기 때문에 비교적 병렬화가 쉽게 이루어질 수 있다[6]. 본 연구에서 다루고 있는 유한요소 인터페이스는 이러한 후처리 과정 또한 고려하여 개선된 것이다.

일반적으로 다중 프론트 해법은 아래의 그림 3과 같이 유한요소 데이터와 연결될 수 있다. 병렬화된 각각의 프로세서는 영역 정보를 가지고 있으며 각 영역에 해당하는 유한요소의 강성 행렬을 계산하여 분해하게 되며 분해 기법으로는 유한요소 강성 행렬의 대칭(symmetry) 특성 때문에 Cholesky factorization이 유용하다. 이와 같은 과정에서 각각의 프로세서는 전역(global) 색인을 가지고 요소를 호출하게 되는데, 이로 인하여 각각의 프로세서는 전체 영역에 걸친 요소 정보와 노드 정보를 순차적인 객체의 배열로 저장하게 된다. 실제로 분산된(distributed) 메모리를 가지는 병렬 시스템에서 각 프로세서의 연산에 필요한 정보는 그림 2-(b)의 분할된 영역 중에서 프로세서에 할당된 영역의 요소 데이터뿐이며 후처리 과정을 고려하더라도 그림 4와 같이 인접한 영역의 요소 정보만 호출할 수 있으면 모든 해석 과정은 가능하게 된다. 이와 같은 사실로 인하여 전역 정보를 메모리에 할당하고 있는 것은 메모리의 손실이 됨을 알 수 있지만 전역 색인에 의하여 적절한 요소와 노드 정보를 호출하는 과정 때문에 불필요한 정보의 저장이 불가피하게 된다.

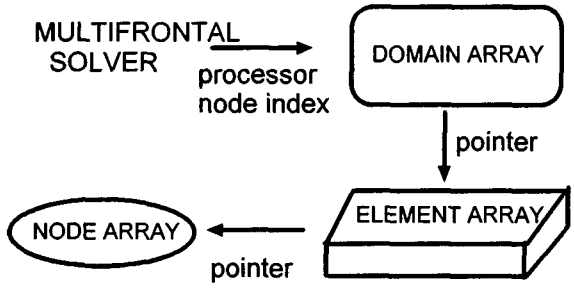


그림 3. Multifrontal solver와 FEM data strucure

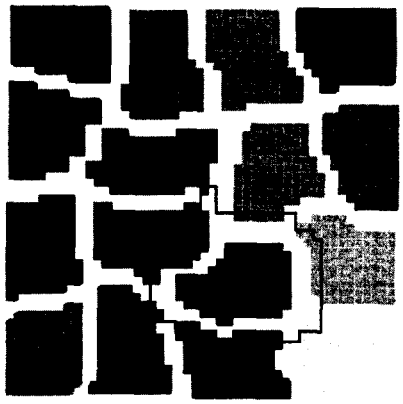
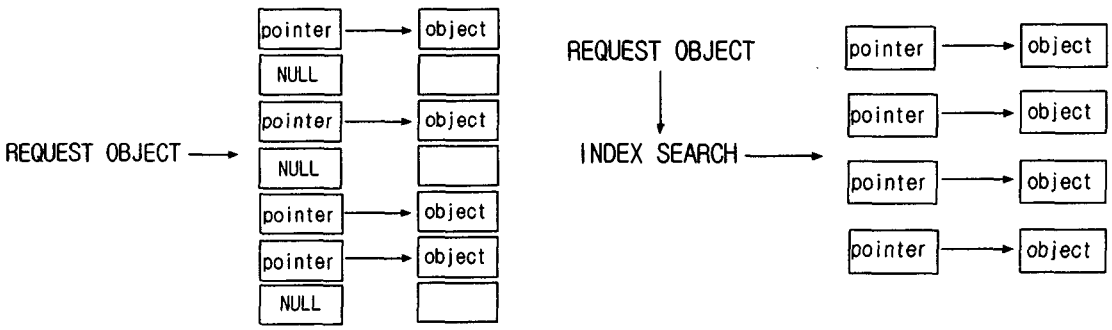


그림 4. 인접 영역을 포함한 local data

본 연구에서는 이와 같이 불필요한 유한요소 데이터를 효과적으로 삭제할 수 있는 2가지 데이터 구조를 제안하여 메모리 효율을 높이도록 하였으며 이는 아래의 그림 5에 표현되어 있다. 그림 5-(a)의 방법은 각각의 프로세서가 필요로 하는 객체만을 남겨두고 색인 역할을 하는 포인터 배열만을 남겨두는 데이터 구조이다. 이와 같은 과정을 통하여 지워진 객체들 만큼의 메모리를 활용할 수 있게 되며 이는 많은 수의 프로세서를 이용할수록 그 효과가 증가하게 됨을 의미한다. 그림 5-(b)는 메모리 할당을 받지 않은 객체들의 포인터 역시 존재하지 않도록 하는 데이터 구조를 나타내고 있다. 이러한 방법을 이용할 경우에는 전역 색인으로부터 지역에 존재하는 객체의 포인터를 찾아내는 알고리즘이 필요하게 되는데 쉽게는 간단한 검색 알고리즘으로부터 해쉬(hash) 구조를 이용하는 방법 등 다양한 기법이 있다. 특히 이와 같이 압축되어 있는 객체 포인터 배열을 이용하게 되면 색인 검색을 위한 자료가 필요하므로 지워진 포인터만큼의 메모리 절약 효과가 있는 반면에 색인 검색 자료의 저장을 위한 손실을 동반한다.



(a) null pointer를 이용한 구조 (b) packed pointer와 index search를 이용한 구조

그림 5. 유한요소 데이터 관리 구조

3. 메모리 분석 및 효율

앞에서 제안한 유한요소 데이터의 병렬 구조를 이용할 경우에 메모리 절감의 효과를 간단한 연산을 통하여 살펴보고 응용 예제를 통하여 실제 연산 속도에 미치는 영향을 살펴본다. 본 논문에서 사용된 클러스터

시스템은 Pegasus(그림 6. 200 노드, 400 cpu)의 64개 노드이며 기본 사양은 표 1에 나타내었다. 프로그램의 병렬화 middleware로는 message passing interface(MPI)[7]를 이용하였으며 라이브러리는 lam-mpi[8] (ver. 6.5.9), 그리고 최적화된 수치 연산 라이브러리인 Goto's BLAS[9]를 사용하여 개발된 다중프론트 해법 프로그램을 이용하였다.

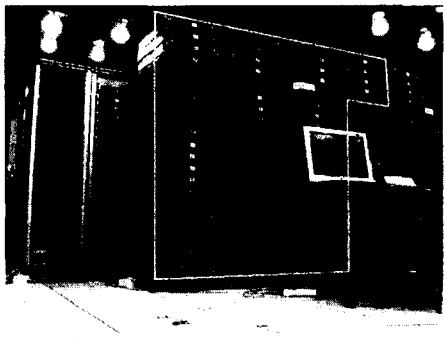
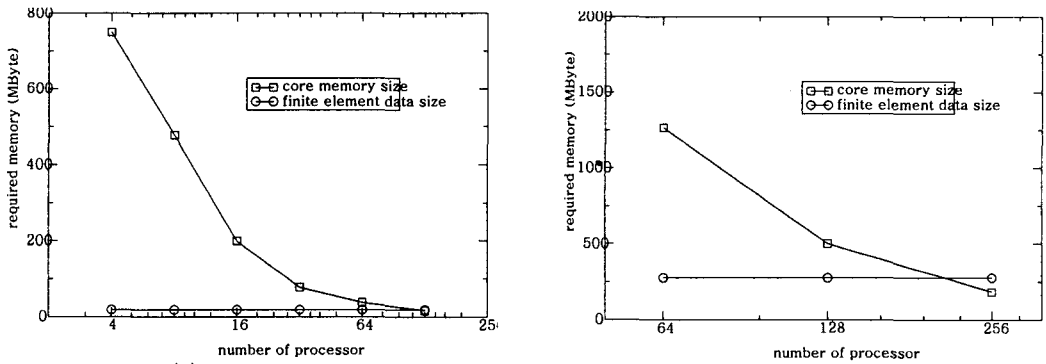


그림 6. Pegasus cluster system
(흰 선으로 표시된 부분이 64개 노드이다)

64 nodes of Pegasus	Xeon 2.2G dual
Memory / node	3.0 G
OS	Linux 2.4.24
Compiler	gcc 2.95.3 absoft pro fortran

표 1. 본 연구에 이용된 클러스터 환경

실제 계산에 적용하기 전에 먼저 프로그램 언어에서 사용되는 기본적인 데이터 크기를 이용하여 메모리 효율을 살펴보자. 일반적으로 사용되는 이중 정확도 부동 소수점 숫자 타입의 데이터 크기는 8 byte이며 정수형 숫자 타입의 크기는 4 byte라고 가정할 때 1개의 8-노드 고체 요소와 3차원 공간상의 노드가 가지는 데이터 크기는 각각 40 byte와 24byte 이다. 이에 덧붙여서 각 요소들과 노드들은 여러 가지 종류의 물성치 또는 요소 특성을 데이터 자체 또는 포인터로 보유하게 된다. 아래 그림 7은 각각 50*50*50 개와 128*128*128개의 8-노드 고체 요소로 분할된 육면체 구조의 해석에 소요되는 기본적인 메모리 크기를 나타내고 있다. 프로세서 개수가 증가함에 따라 core memory size는 감소하는 반면 finite element data size는 일정하게 유지되며 이에 따라 128 cpu's 정도에서 두 크기가 비슷해지며 그림 7-(b)에 의하면 256 cpu's 에서는 오히려 유한요소 데이터가 행렬의 크기를 능가함을 알 수 있다. 이와 같은 양상에 비추어 볼때 유한요소 데이터의 병렬화가 필요함을 알 수 있으나 이러한 양상을 보이는 구조 해석 문제 또는 대량의 프로세서를 보유하고 있는 클러스터 시스템의 경우임에 주목해야 한다.



(a) 50*50*50 elements (b) 128*128*128 elements

그림 7. 병렬 다중프론트 해법에 소요되는 메모리 크기와 유한요소 데이터 크기의 비교

유한요소 데이터 크기가 다중프론트 해법의 지역 행렬 크기보다 커지게 되는 이러한 경우에는 실제로 전처리 과정에서도 상당한 시간이 소요되고 예상치 못한 문제가 발생하게 된다. 예를 들어 그래프 분할의 문제에 있어서도 메모리 문제가 발생하거나 순차적 계산에 의한 분할 기법으로는 오히려 유한요소 해석의 행렬 방정식 연산 시간보다 전처리 시간이 더 길어지는 경우도 있게 된다. 이러한 전처리 과정의 문제점은 본 연구에서 다루고 있지 않으나 그래프 분할 등의 문제에 있어서는 이미 병렬화 등의 기법을 통하여 해결책이 제시되고 있다[10][11].

앞서 설명한 두가지 데이터 구조 중에서 객체에 대한 포인터를 압축하여 색인 검색 기능을 사용한 경우에는 검색에 소요되는 시간에 의하여 다중프론트 해법의 병렬 효율이 감소할 수 있다. 아래의 표 2는 앞서 제시한 50*50*50 요소 모델에 대하여 유한요소 해석에 소요되는 시간을 나타낸 것이다. 색인 검색에는 데이터를 4개의 테이블로 등분하여 순차적 검색 기법을 이용하였으며. 유한요소 전체 데이터를 저장한 상태에서 계산한 것과 데이터 병렬화를 통하여 색인 검색 기능을 사용한 경우를 비교하였을 때 element computation 시간에 있어서 약간의 차이를 보이고 있으나 이러한 경향은 프로세서 개수가 증가함에 따라서 그 효과가 사라짐을 알 수 있다. 그림 8은 전체 유한요소 해석 시간을 기준으로 하여 병렬화에 의한 speed-up을 나타낸 것으로써 병렬 압축된 유한요소 데이터 구조와 색인 검색 기법의 성능이 일반적인 데이터 구조의 경우와 비교하여 큰 차이가 없음을 보여준다.

	element computation	factorization	substitution
4 cpu	a: 42.48	a: 308.78	a: 13.24
	b: 148.62	b: 309.85	b: 13.19
8 cpu	a: 21.24	a: 189.68	a: 13.30
	b: 47.71	b: 189.81	b: 13.30
16 cpu	a: 11.12	a: 135.50	a: 14.47
	b: 18.19	b: 134.75	b: 14.93
32 cpu	a: 5.78	a: 95.49	a: 15.73
	b: 7.92	b: 95.14	b: 15.74

표 2. 데이터 구조에 따른 계산 시간 비교
(a: 전역 데이터 저장, b: 지역 데이터 저장 및 검색)

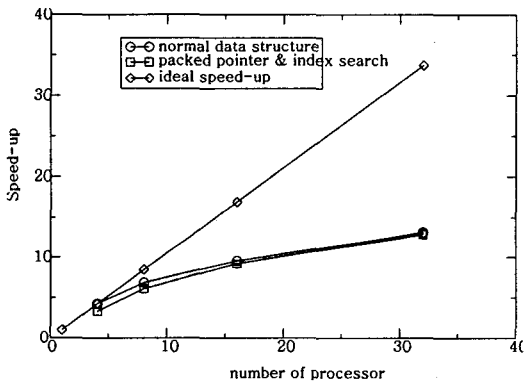


그림 8. 데이터 구조와 speed-up

4. 결 론

다중 프론트 해법과 유한요소 데이터 사이의 인터페이스 구조를 제안하고 이를 실제 계산에 적용하여 그 성능을 평가하였다. 본 연구에서 제안된 유한요소 데이터 구조는 거대 구조물의 해석과 대형 클러스터 시스템을 기반으로 하였을 경우에 적용될 수 있는 기법으로써 메모리 절약의 효과에 비추어 성능 저하가 없는 null pointer 기법과 약간의 성능 저하와 함께 메모리 절약 효과를 극대화 할 수 있는 색인 검색을 이용한 데이터 구조이다. 병렬 압축된 데이터 구조와 색인 검색 기법을 이용하였을 경우, 메모리 효율은 최대가 되며 성능 저하는 프로세서 숫자가 늘어남에 따라 사라짐을 확인하였다. 이러한 데이터 저장 기법은 대형 클러스터 또는 그리드 컴퓨팅 등을 이용하여 초대형 구조물의 해석을 할 경우에 매우 필요한 기본 기술이 될 것이다.

후 기

본 연구는 과학기술부의 핵심 우주 기술 개발 사업의 지원을 받아 수행되었습니다. 이에 관계자 여러분께 감사의 말씀을 드립니다.

참고 문헌

1. Gupta A., Karypis G. and Kumar V., "Highly Scalable Parallel Algorithms for Sparse Matrix Factorization," IEEE Trans. Parallel Distrib. Syst. Vol. 8, pp.502-520

2. Amestoy P.R., Duff I.S. and L'Excellent J.Y., "Multifrontal Parallel Distributed Symmetric and Unsymmetric Solvers," Comput. Methods Appl. Mech. Engrg., Vol. 184, 2000, pp.501-520

3. Xiaoye S.L. and James W.D., "SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems," ACM Transactions on Mathematical Software, Vol. 29, No. 2, 2003, pp.110-140

4. SuperLU, http://crd.lbl.gov/~xiaoye/SuperLU

5. MUMPS, http://graal.ens-lyon.fr/~jylexcel/MUMPS

6. Park S.H. and Kim J.H.. "A posteriori Error Estimation based on Variation of Mapping Function," Finite Elements in Analysis and Design, Vol. 40, 2004, pp.711-735

7. Message Passing Interface, http://www-unix.mcs.anl.gov/mpi

8. LAM/MPI, http://www.lam-mpi.org

9. High-Performance BLAS by Kazushige Goto, http://www.cs.utexas.edu/users/flame/goto

10. Karypis G. and Kumar V., "Parallel Multilevel k-way Partitioning scheme for Irregular Graphs." Siam Review, Vol. 41, No. 2, 1999, pp.278-300

11. Fjallstrom P.O. "Algorithms for Graph Partitioning: A Survey," Linkoping Electronic Articles in Computer and Information Science, Vol. 3, No. 10, 1998, pp.1-33