초대형 구조해석을 위한 고성능 연산기법 연구

Study on High-Performance Computing Technique for Large-scale Structural Analysis

김 정 호* 우 성 운**

*한국과학기술정보연구원 슈퍼컴퓨팅센터 선임연구원
**한국과학기술정보연구원 슈퍼컴퓨팅센터 위촉연구원

1. 서 론

유한요소법을 활용한 전산구조해석 기법은 컴퓨터 기술의 눈부신 발전과 함께 그 적용 범위나 활용도에 있어 과거에 비하여 비교할 수 없을 만큼 그 중요도가 커졌다고 할 수 있다.

이에 따라 구조해석을 위한 유한요소 모델링도 점점 실제 대상물에 가깝게 상세해 지고, 그에 따른 계산량이나 계산 자원의 규모도 급격히 증가하고 있는 추세이다. 이러한 상황에서 단순히 단일 프로세서의 시스템 규모 및 성능 발전 속도에만 의존하고 있어서는 늘어나는 계산량과 계산 자원의 규모에 대한 요구를 더 이상 효과적으로 감당할 수가 없게 되었다. 따라서 이러한 고성능 계산에 대한 요구를 효과적으로 충족할 수 있도록 하기 위해서는 단일 프로세서 상에서 최적의 성능을 낼 수 있는 계산 기법뿐만 아니라, 병렬형 컴퓨팅 시스템이 점점 더 일반화 되어가고 있는 상황에서 이러한 병렬형 시스템을 제대로 활용해서 성능을 낼 수 있는 고성능 병렬 계산 기술이 필요하다. 즉, 유한요소해석 기법을 고성능 병렬 시스템에 효율적으로 적용할 수 있도록 하기 위한 고성능 계산 기술이 필수적인 상황이 되었으며, 이러한 기술이 없이는 아무리 고성능의 슈퍼컴퓨터라 하더라도 무용지물이 될 수밖에 없다.

한편, 내재적(implicit) 유한요소해석 기법을 이용한 구조해석 과정에 있어서 가장 많은 계산 시간과 계산 자원을 요구하는 부분이 편미분방정식의 이산화를 통해 구성된 연립방정식을 푸는 부분으로서 문제의 규모가 커지게 되면 대부분의 계산 시간이 이 부분에서 소요되게 된다. 따라서 초대형 구조해석 문제를 풀기 위해서는 유한요소해석 과정에서 구성된 연립방정식을 효율적으로 풀기 위한 고성능 선형 연립방정식 해석기(solver)가 필수적이다. 특히 초대형 문제의 해결을 위해서는 메모리를 효율적으로 활용할 수 있어야 할 뿐만 아니라, 단일 시스템의 물리적 한계를 뛰어넘어 보다 큰 문제를 풀 수 있도록 분산메모리 상에서 효율적으로 병렬화가 되어야 한다. 결국 초대형 구조해석을 위한 고성능 유한요소해석을 위해서는 해석 과정 전체의 성능을 결정하는 고성능 병렬 연립방정식 해석 기법에 대한 연구가 가장 핵심적이고 중요한 요소이므로 이에 대한 집중적인 연구가 필요하다.

따라서, 본 연구에서는 일반적인 단일 프로세서 시스템에서뿐만 아니라, 고성능 병렬형 컴퓨팅 시스템 상에서도 시스템을 최대한 활용해서 최대의 성능을 낼 수 있는 효율적인 유한요소해석을 위한 고성능 연립방정식 해석 기법에 대한 연구 현황 및 문제점을 살펴보고, 그에 대한 개선책으로써 새로운 기법을 제안하고자 한다.

2. 고성능 연립방정식 해석 기법

유한요소해석 과정에서 나타나는 행렬은 일반적으로 매우 0을 많이 가지고 있는 산재행렬(sparse matrix)이므로 효율적인 선형 연립방정식 해석을 위해서는 이러한 특성을 잘 살려서 계산량을 최소로 줄이고, 또한 하드웨어 구조에 최적화되어 최대의 성능을 낼 수 있어야 한다. 이러한 산재행렬을 위한 연립방정식 해법은 크게 반복적인 해법(iterative solver)과 직접적 해법(direct solver)으로 나눌 수 있다. 이 두 가지 해법의 특징 및 장·단점과 이들 기법을 활용한 초대형 구조해석 프로그램 개발 현황에 대하여 간략히 살펴보도록 하겠다.

2.1 반복적 해법

반복적 해법은 사실상 구조해석 문제의 해결에는 잘 사용되지 않아 왔으나, 근래에 초대형 유한요소해석에 대한 요구 및 병렬형 컴퓨터의 발달과 함께 병렬화 및 대형 문제 해결에서의 이점으로 인하여 많은 주목을 받고 있다. 이 방법이 유한요소해석 과정 전체를 쉽게 병렬화할 수 있고 병렬효율도 우수할 뿐만 아니라, 메모리를 상대적으로 많이 요구하지 않기 때문에 초대형 문제를 푸는 데에 보다 더 적합한 것으로 인식되었던 것이다. 이러한 반복적 해법들은 병렬화를 전제로 한 영역분할기법을 기반으로 한 접근이 많이 시도되어 왔다. 이러한 영역분할기법에 기반한 병렬 유한요소해석 기법으로는 C. Farhatp[1] 등이 제안한 FETI(Finite Element Tearing and Interconnecting) 기법이 가장 성공적인 것으로 평가되고 있으며, 이를 이용한 초대형 구조해석용 프로그램으로서 Salinas[2]가 있다. 그러나 근본적으로 영역분할기법에 기반한 방법들을 비롯한 반복적 해법의 경우 수치적인 안정성과 이로 인한 범용성에 문제가 있고 계산량의 예측이 불가능할 뿐만 아니라, 구조해석 문제에서 흔히 나타나는 다중우변항(Multiple Right-Hand Side) 문제를 효과적으로 처리할 수가 없다. 특히, 구조해석 문제에 있어서는 행렬의 수치적인 조건이 매우 나쁘게 나오는 경우도 많이 발생할 수 있어, 이러한 형태의 문제의 경우 원하는 수준의 정확도를 가진 해를 항상 안정적으로 구하기가 곤란한 경우가 많다. 이와 같은 이유로 인해 반복적 해법의 병렬화 및 대형 문제에 있어서의 많은 장점에도 불구하고 이러한 방법들이 초대형 구조해석 문제의 해결을 위한 최선의 해결책으로써 일반적으로 완전히 받아들여지지 못한 것이라고 볼 수 있다. 그 결과 ABAQUS, NASTRAN, ANSYS 등 가장 널리 사용되는 범용 구조해석을 위한 상용 프로그램들은 대부분 여전히 가우스 소거법을 기반으로 해서 선형 연립방정식을 푸는 직접적 해법을 주로 사용하고 있다. 그러나 범용성보다는 초대형 병렬 시스템에서 대규모의 구조해석 문제의 해결을 목적으로 하는 경우 어쩔 수 없이 반복적 해법을 사용하고 있으며, 대표적인 것으로 앞에서 언급한 Salinas 이외에 GeoFEM[3], ADVENTURE[4], Olympus[5] 등이 있다. 이들 중 GeoFEM은 병렬화된 공액구배법(Conjugate Gradient Method)을 문제에 따라 적절한 예조건화(preconditioning) 기법을 적용하여 연립방정식 해법으로 사용하고 있으며, ADVENTURE의 경우는 부구조화(substructuring)기법과 공액구배법이 결합된 전형적인 영역분할기법(Domain Decomposition Method)을 사용하고 있다. 가장 근래에 발표된 Olympus코드의 경우 AMG(Algebraic Multi-Grid) 기법을 사용하고 있다. 이들 프로그램들은 세계 최대 규모의 슈퍼컴퓨터였던 미국의 ASCI White, 또는 일본의 Earth Simulator와 같은 초대형 슈퍼컴퓨터에서 작동하도록 개발되었으며, 수천만 혹은 수억 개의 미지수를 가진 문제까지도 해결이 가능한 것으로 알려져 있다. 특히 슈퍼컴퓨팅 분야에서 최고의 성과를 이루어낸 고성능 계산 기법에 대하여 매년 수여되는 Gordon Bell상이 2002년도에는 Salinas팀에게, 2004년에는 Olympus팀에게 수여된 것만 봐도 고성능 계산 분야에서 초대형 유한요소해석 기법에 대한 관심 및 중요도가 얼마나 큰지 알 수 있다.

2.2 직접적 해법

반복적 해법의 경우 산재행렬의 0이 아닌 값들만 저장해서 계산에 이용하면 되는 반면, 계수행렬의 인수화(factorization)를 수행해야 하는 직접적 해법의 경우 행렬의 인수화 과정에서 원래 0이었던 행렬의 원소가 0이 아닌 값으로 채워지는 현상이 발생하게 되는데, 이를 '채워넣기'(fill-in)이라고 한다. 이 때문에 직접적 해법이 반복적 해법에 비해서 일반적으로 훨씬 더 많은 기억용량을 요구하게 되며, 그것을 병렬처리 시 프로세서 별로 적절히 나누는 데에도 많은 어려움이 따른다. 따라서 이로 인해 직접적 해법은 대형 문제의 해결에서 많은 한계를 지니게 되는 것이다. 뿐만 아니라 이 '채워넣기' 항의 개수에 따라 직접적 해법의 계산량도 크게 좌우되게 되므로, 이러한 '채워넣기' 항을 줄이는 것이 직접적 해법에 있어서 계산량 및 기억용량을 줄이기 위해 가장 중요한 문제가 된다. '채워넣기' 항을 줄이기 위한 산재행렬을 위한 직접적 해법의 가장 간단한 형태가 구조해석 시의 선형 연립방정식

의 계수행렬(강성행렬)을 대각원소를 중심으로 띠(band) 모양으로 0이 아닌 항이 존재하는 형태로 가정하고 푸는 밴드 해법 또는 스카이라인 해법이다. 이 경우 '채워넣기' 항은 띠 내부에서만 발생하게 되는데, 실제 구조해석 문제를 푸는 과정에서 나타나는 일반적인 산재행렬(sparse matrix)의 경우, 아무리 절점의 배열 순서를 최적으로 해서 띠의 크기를 줄인다고 하더라도 이런 띠 내부에 상당히 많은 0인 원소를 포함하므로 상당량이 '채워넣기' 항이 발생할 수밖에 없다. 그럼에도 불구하고 이보다 효율적인 다른 기법들은 적어도 구조해석 분야에서는 아직도 그렇게 잘 알려진 상태가 아닌 관계로 직접 개발한 구조해석 코드를 사용하는 많은 연구자나 또 일부의 상용 구조해석 프로그램들 조차도 이러한 비효율적인 기법에서 벗어나지 못하고 있는 경우가 많다. 그리고 이러한 비효율적인 기법들에 대한 정보를 근거로 해서 직접적 해법은 비효율적이고, 특히 분산메모리 병렬화를 통한 대형 문제의 해결에는 적합하지 않다는 결론을 내리기 쉽다. 그리고 실제로 이러한 직접적 해법을 이용한 초대형 유한요소해석의 시도는 병렬화의 어려움과 기억용량의 한계로 인하여 거의 이루어지지 않고 있는 실정이다. 그렇지만 직접적 해법이 가진 수치적 안정성 및 다중우변항 처리 능력 등 여러 가지 매우 중요한 장점들로 인해 초대형 구조해석을 구현하기 위한 직접적 해법의 효율적인 병렬화는 가능하기만 하다면 매우 중요하고 의미 있는 일이다. 직접적 해법을 이용하여 초대형 구조해석을 구현한 가장 성공적인 사례가 바로 2003년도 Gordon Bell 상의 3팀의 최종 후보에 포함되었던 IPSAP[6]프로그램이다. 여기에 사용되었던 기법이 영역기반 다중프론트 해법이며, 근래에는 해당 프로그램을 이용해서 32비트 PC 클러스터 상에서 1억 개의 미지수를 가진 구조해석 문제를 성공적으로 해결한 바 있다.[7] 본 연구에서는 이 영역기반 다중프론트 해법 및 이의 성능 개선 결과를 소개하고자 한다.

3. 다중프론트 해법

3.1 프론트 해법

프론트 해법(Frontal solution method)는 제한된 주기억 용량으로 유한요소해석 문제의 해를 효율적으로 구하기 위해 Irons[8]에 의해 처음으로 소개되었다. Irons가 제안한 프론트 해법은 기존의 해법(밴드 및 스카이라인 해법 등)들이 전역강성행렬을 완전히 조립한 후에 가우스 소거법 등을 이용하여 연립방정식을 푸는 것과는 달리 전역강성행렬을 조립하지 않고 이웃하는 요소들에 대해서 요소강성행렬을 하나씩 조립해나가는 과정에서 조립이 완료된 부분을 부구조화(substructuring)과정을 통하여 그 즉시 소거해 나가는 방법이다. 부구조화 과정은 요소강성행렬을 조립해 나가는 과정에서 식 (1)과 같이 연립방정식이 구성되었다면, 여기서 조립이 완료된 자유도 $\mathbf{u}_1$을 소거하고 조립이 완료되지 않은 자유도 $\mathbf{u}_2$만으로 구성된 식 (2)와 같은 새로운 연립방정식을 구성하는 과정이다.

$$\begin{bmatrix}\mathbf{K}_{11} & \mathbf{K}_{12}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{bmatrix}\begin{Bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{Bmatrix}=\begin{Bmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{Bmatrix},$$

where $\mathbf{u}_1$ has fully assembled DOFs (1)

$$\begin{aligned}\overline{\mathrm{K}}_{22}\mathrm{u}_2&=(\mathrm{K}_{22}-\mathrm{K}_{21}\mathrm{K}_{11}^{-1}\mathrm{K}_{12})\mathrm{u}_2\\&=\mathrm{f}_2-\mathrm{K}_{21}\mathrm{K}_{11}^{-1}\mathrm{f}_1=\overline{\mathrm{f}}_2\end{aligned} \quad (2)$$

이 과정에서 소거된 자유도 관련 데이터는 보조기억장치로 옮길 수 있기 때문에 계산에 요구되는 주기억 장치의 용량을 크게 줄일 수 있다. 그런데 프론트 해법은 적은 주기억장치를 사용하기 때문에 보다 큰 문제를 해석할 수 있다는 장점이 있기는 하지만, 계산량은 기존의 스카이라인 해법 등에 비하여 거의 차이가 없기 때문에 계산 시간에 대한 성능 향상은 기대하기 힘들다.

3.2 다중프론트 해법

다중프론트 해법은 유한요소해석을 위해 고안된 프론트 해법의 일반화 과정에서 그 개념의 확장을 통해 Duff 등[9]에 의해 제안되었으며 Liu[10]에 의해서 개념이 정립되었다. 다중프론트 해법은 산재행렬로 구성된 연립방정식을 풀기 위한 범용 연립방정식 해법으로써 기존의 프론트 해법이나 스카이라인 해법 등에 비하여 계산량 및 기억용량을 크게 줄일 수 있다.

다중프론트 해법은 계수행렬(유한요소해석에서 전역강성행렬)의 산재패턴(sparse pattern)을 이용한 방법이다. (그림 1)과 같은 행렬이 있다고 하자. 행 ④ ⑤ ⑥을 인수화 할 때 행 ① ② ③ 은 필요하지가 않으며, 그 역으로 행 ① ② ③ 을 인수화 할 때, 행 ④ ⑤ ⑥은 필요하지가 않다. 즉, 행 ① ② ③과 행 ④ ⑤ ⑥은 인수화 시에 서로 독립적인 연산이 가능하며 이 과정에서 '채워넣기' 항을 크게 줄일 수 있게 되며, 이런 개념을 (그림 2)와 같이 트리 구조 형태로 표현할 수 있다.[14] 이렇게 서로 독립적인 일반화된 프론트(generalized front)가 여러 개 형성되므

로 다중프론트 해법이라 하며, 이렇게 독립적으로 계산 가능한 여러 개의 프론트가 생성되므로 순차 계산에 있어서 기존의 방법에 비하여 훨씬 효율적일 뿐만 아니라 병렬처리에 있어서도 많은 유리한 점을 가진다.

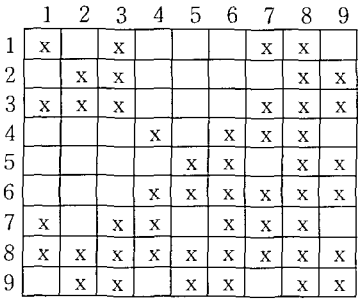

그림 1 Example matrix

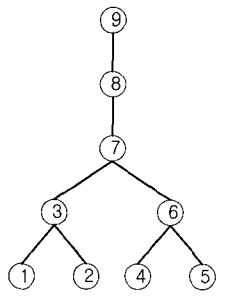

그림 2 Elimination tree

이 알고리즘을 기반으로 하는 여러 가지 고성능 연산 라이브러리가 개발되었으나,[11] 현재 가장 많이 사용되고 효율성이 우수한 것으로 널리 알려진 소프트웨어로는 MUMPS[12]와 WSMP[13]를 들 수 있다.

MUMPS는 P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, J. Koster 에 의해 개발되었으며, 다중프론트 해법을 이용한 범용 산재행렬 연립방정식 해석용 소프트웨어 패키지이다. Fortran90으로 쓰여졌으나 C에서도 호출이 가능하며 순차(serial) 버전과 MPI를 이용한 병렬 버전을 사용할 수 있다.[15]

WSMP는 A. Gupta에 의해 개발되었으며 MUMPS와 마찬가지로 다중프론트 해법을 이용한 범용 산재행렬 연립방정식 해석용 소프트웨어 패키지이다. Fortran으로 만들어 졌으나 C에서도 호출이 가능하며, 현재 순차 버전 및 공유메모리 병렬화, 분산메모리 병렬화 버전을 이용할 수 있다.[17]

3.3 영역기반 다중프론트 해법(Domain-wise Multifrontal Solver)

다중프론트 해법은 유한요소해석 과정에서 나타나는 산재행렬로 구성된 연립방정식을 매우 효율적으로 해결할 수 있는 방법이지만 산재행렬인 계수행렬이 우선 구성되어야만 적용할 수 있기 때문에 대형 문제의 해결을 위해 유한요소해석 과정을 병렬화하고자 하는 경우 문제에 부딪히게 된다. 즉, 연립방정식 해법 자체는 효율적으로 병렬화가 되어있다고 하더라도 행렬의 구성 과정이 병렬화가 되어서 적절히 분할되어 분포시킬 수 있어야만 대형 문제를 제대로 해결할 수 있는데, 일반적인 다중프론트 해법을 쓰는 경우 다중프론트 해법에서 병렬처리를 수행할 수 있도록 행렬을 적절히 분할해서 구성하기가 매우 곤란하다.

영역기반 다중프론트 해법은 이러한 문제점을 해결하기 위해 Kim등이 제안한 것으로 다중프론트 해법을 유한요소해석 과정에 보다 효율적으로 적용시키기 위해 고전적인 프론트 해법의 장점을 도입하여 모든 계산 과정을 요소 또는 영역 단위로 수행하도록 한 방법이다.[18] 이 방법은 유한요소해석 과정에서 필요한 기억용량 및 연산회수를 최소로 줄여줄 뿐만 아니라 모든 연산이 요소 및 영역 기반으로 진행되므로 유한요소해석 과정의 병렬화에 있어서도 매우 큰 이점을 가지고 있다.

영역기반 다중프론트 해법은 (그림 3)과 같이 문제의 영역을 반복적으로 반으로 분할한 다음, 분할한 역순으로 이웃하는 영역을 둘씩 합쳐나가면서 합쳐진 영역 내부의 자유도를 소거해 나가는 것으로 이 과정은 (그림 4)와 같이 재귀적 부구조화(Recursive substructuring)과정으로 설명할 수도 있다.

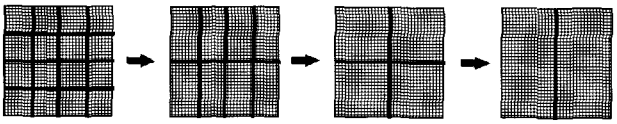

그림 3 Elimination (factorization and forward substitution)

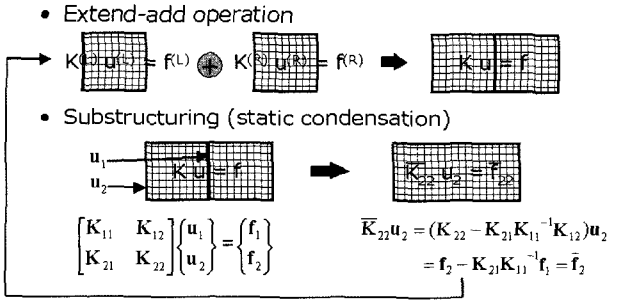

그림 4 Recursive substructuring

직접적 연립방정식 해법을 이용한 유한요소해석 문제에 있어서 또 하나 중요한 문제가 되는 것이 기억용량의 문제이다. 영역기반 다중프론트 해법은 소거가 된 자유도에 해당하는 행렬을 고전적인 프론트 해법과 마찬가지로 디스크에 임시로 저장(out-of-core)할 수 있도록 함으로써 성능에는 큰 영향을 미치지 않고 계산에 요구되는 주기억장치의 용량을 크게 줄일 수 있다. 뿐만 아니라 전역강성행렬을 조립하지 않고 메모리를 최대한 효율적으로 활용하도록 프로그래밍되어 있어 대형 문제를 매우 효율적으로 적은 메모리만으로 풀 수 있다.

초대형 구조해석을 위한 병렬 유한요소해석 기법을 구

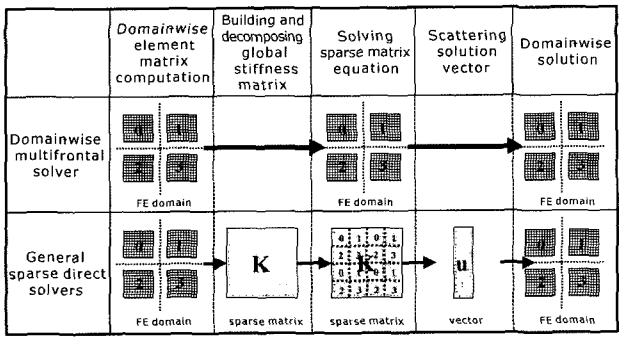

그림 5 Comparison of parallel finite element analysis procedure

현하는 경우 영역기반 다중프론트 해법의 장점을 다중프론트 해법과 같은 범용 직접적 산재행렬 해법의 경우와 (그림 5)에서 비교하였다. 영역기반 다중프론트 해법의 경우 전역강성행렬을 구성하지 않고 초기 유한요소망을 영역별로 분할한 상태에서 연립방정식 계산이 이루어지지만 범용 산재행렬 해법의 경우는 전역강성행렬을 구성해야 하므로 문제를 영역별로 나누었다하더라도 어떤 식으로든 산재행렬 해법에서 원하는 형태로 데이터를 재분산을 해야만 한다. 뿐만 아니라 연립방정식에 대한 계산이 끝난 후에도 영역기반 다중프론트 해법의 경우 계산 결과가 각 영역별로 저장이 되게 되지만, 범용 산재행렬 해법의 경우 이를 다시 영역별로 재분산을 해줘야만 한다. 따라서 유한요소해석의 병렬처리를 고성능 직접적 해법을 이용해서 구현하고자 하는 경우 본 연구에서 제안하는 영역기반 다중프론트 해법을 사용하는 경우 요소강성행렬만 계산해 주는 것으로 유한요소해석 과정 전체의 병렬화가 끝나는 반면, 다중프론트 해법과 같은 범용 산재행렬 해법을 사용하고자 하는 경우, 전역강성행렬의 구성을 행렬을 분할하여 병렬로 구현하는 작업 등의 복잡한 추가적인 작업이 필요하고 이로 인해 보다 더 많은 기억용량과 프로그래밍 노력이 필요하게 된다.

4. 영역기반 다중프론트 해법의 병렬화

초대형 유한요소해석 문제의 해결을 위해서는 물리적인 단일 메모리 시스템이 감당할 수 있는 규모보다도 훨씬 더 큰 규모의 문제를 풀 수 있어야 하는데 이를 위해서 분산메모리 병렬화는 필수적이다.

병렬화의 구현은 영역기반 다중프론트 기법의 내재적인 병렬성을 최대로 활용하여 (그림 6)과 같이 각각의 프로세서에 적절한 크기의 영역을 할당해서 프로세서에 할당된 영역 내부에서의 영역 결합은 완전히 독립적으로 수행하게 한 다음, 2개 이상의 프로세서에 할당된 영역에 대한 계산은 통신을 통해 해당 행렬식을 구성한 다음 병렬 행렬 연산을 통해서 수행하도록 하였다. 이 과정에서 데이터를 물리적으로 분리된 별개의 메모리 시스템에 나누어서 저장하는 것이 가장 크고 어려운 문제이다. (그림 6)에서 하나 이상의 프로세서가 가진 특정 영역에 대한 프론트 행렬을 다른 영역과 합치는 것은 어느 한 메모리 시스템에서 일어나는 것이 아니고 그 영역을 포함한 계산에 참여하고 있는 모든 프로세스가 각자가 가진 데이터를 재분배해야 한다. 여기서 적절한 방법을 사용하지 못하면 통신에 있어서 지나친 부하가 발생해서 프로그램의 확장성을 크게 해치게 된다.

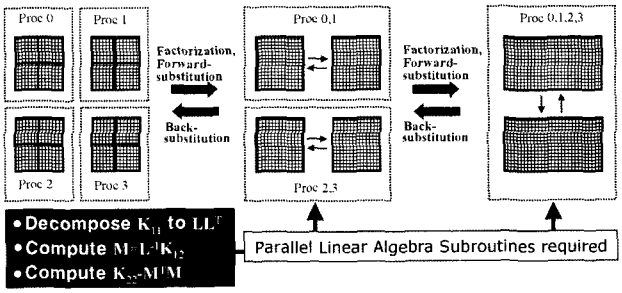

그림 6 Parallel implementation of parallel domain-wise multifrontal solver

여기서 두 프로세서 이상이 하나의 행렬 계산을 수행하기 위해서는 분산메모리 상에서 병렬로 행렬 계산을 할 수 있는 루틴이 필요하다. 이에 해당하는 기능을 가진 것이 PBLAS와 ScaLAPACK[19]인데 이것들을 사용하기 위해서는 블록 크기가 반드시 고정된 형태로 데이터가 분산되어야만 한다. 그러나 이들 라이브러리에서 요구하는 이러한 데이터 분산 패턴에 따라 데이터를 분산하려면 행렬 계산 구성을 위한 데이터의 재분배에 지나치게 많은 통신량이 요구되어 확장성을 크게 해칠 수 있다. 따라서 이러한 문제점을 감수하고 기존의 PBLAS 및 ScaLAPACK을 활용하여 병렬 행렬 계산을 구현하거나, 그렇지 않으면 효율적인 데이터 재분배 시에 발생하는 데이터 분산 패턴을 처리할 수 있는 병렬 행렬 계산 루틴을 따로 개발해야 한다. 그런데 MUMPS의 경우 전자의 방법을 선택하고 있고, WSMP 및 본 연구를 통해 개발된 영역기반 다중프론트 해법은 훨씬 더 많은 프로그래밍 상의 노력이 필요한 후자의 방법을 택하고 있다. 후자의 방법을 이용하여 행렬 계산을 구성할 때의 통신 형태를 16개의 영역으로 구성된 2차원 문제의 경우를 예로 들면 다음 (그림 7)와 같다. 여기서 점선으로 둘러싸인 부분은 각 단계에서 하나의 행렬 계

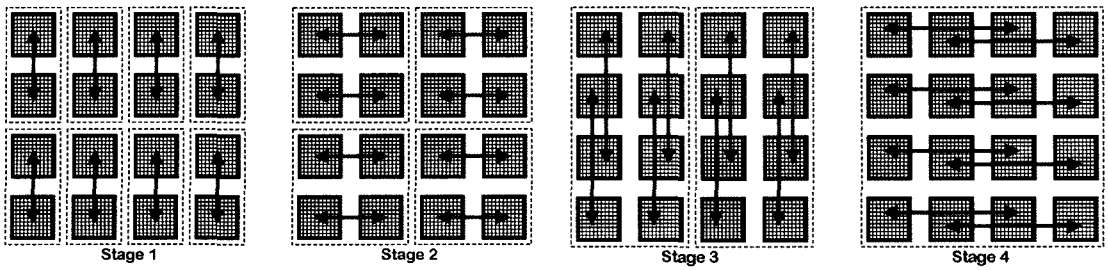

그림 7 Communication pattern for front matrix construction

산에 참여하고 있는 프로세스들을 나타낸다. 여기서 보는 바와 같이 모든 단계에서 행렬식 구성 시 프로세스 간의 통신은 1대1로만 발생되고 있고, 이를 통해서 최종 단계에서는 모든 프로세스가 참여하는 행렬 계산식이 자동적으로 구성되게 된다. 반면 MUMPS와 같이 PBLAS 및 ScaLAPACK을 이용할 수 있도록 이 부분을 구현하게 되면 각 단계별 통신은 점선 안의 각 영역 간에 All-to-all 통신 형태로 일어나야 하기 때문에 병렬효율이 매우 나빠지게 된다. 각 구현 방법에 따른 성능에의 영향은 다음 절에서 확인할 수 있다.

5. 영역기반 다중프론트 해법의 성능

본 연구에서 구현된 영역기반 다중프론트 해법의 성능을 검증하기 위해 몇 가지 모델의 정적 구조해석을 수행하였다. 영역기반 다중프론트 해법의 성능은 일반적으로 구조해석 프로그램에 많이 사용되는 직접적 해법인 스카이라인 해법 등에 비하면 비교할 수 없을 정도로 월등히 성능이 뛰어나며, 최적으로 구현된 경우와 비교하더라도 최소 3~4배에서 수십 배 이상의 성능 차이가 난다. 본 연구에서는 다중프론트 알고리즘을 사용하는 직접적 해법으로써 가장 잘 알려진 WSMP와 MUMPS를 이용하여 구조해석을 수행한 결과와 상용 구조해석 소프트웨어 중 연립방정식 해석 성능이 가장 우수한 것으로 알려진 ABAQUS와 순차 및 병렬 성능을 32개의 POWER4 1.7GHz 프로세서를 가진 IBM p690 시스템에서 비교하였다. 그리고 IPSAP[6]의 연구 결과 이후 불규칙하고 복잡한 형상을 가진 유한요소 모델에 대한 성능을 개선한 결과를 검증하기 위해 IPSAP[6]의 성능과 프론트 분할 기법을 개선해 성능 개선을 이룬 현재 버전의 성능을 비교하였다. 성능 비교에 사용된 유한요소 모델은 (그림 8)과 같으며, 미지수의 개수는 순서대로 각각 106만개, 56만개, 120만개가 된다.

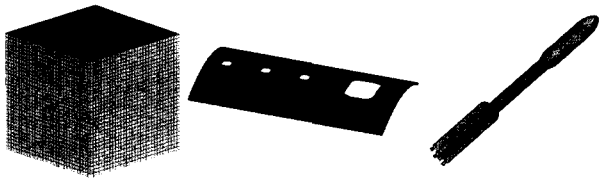

그림 8 Finite element models for test problems

(그림 9), (그림 10), (그림 11)에 이들 유한요소 모델에 대한 성능 비교 결과가 나타나있다. 여기서 'D-MFS'로 표시된 것이 현재 구현된 영역기반 다중프론트 해법이며 'D-MFS(org)'로 표시된 것은 IPSAP[6]에서 사용됐던 영역기반 다중프론트 해법에 해당한다. 여기서 보는 바와 같이 유한요소 해석 프로그램 전체의 병렬화에 영역기반 다중프론트 해법이 효율적으로 적용되어 다른 산재행렬 해법이나 상용 구조해석 프로그램에 비하여 순차 성능 뿐만

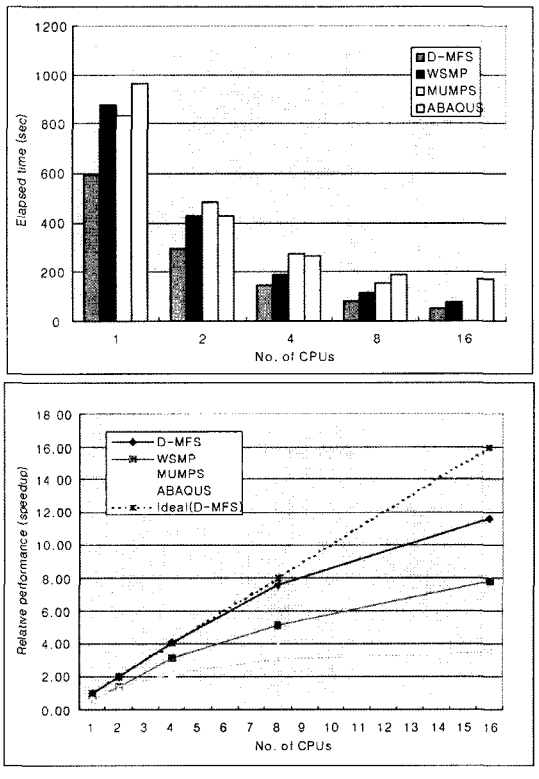

그림 9 Parallel performance comparison for 'S48' model

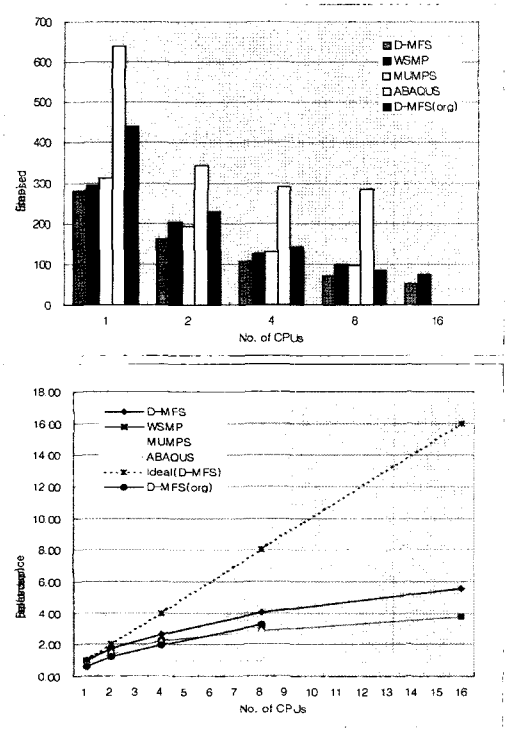

그림 10 Parallel performance comparison for 'Rocket' model

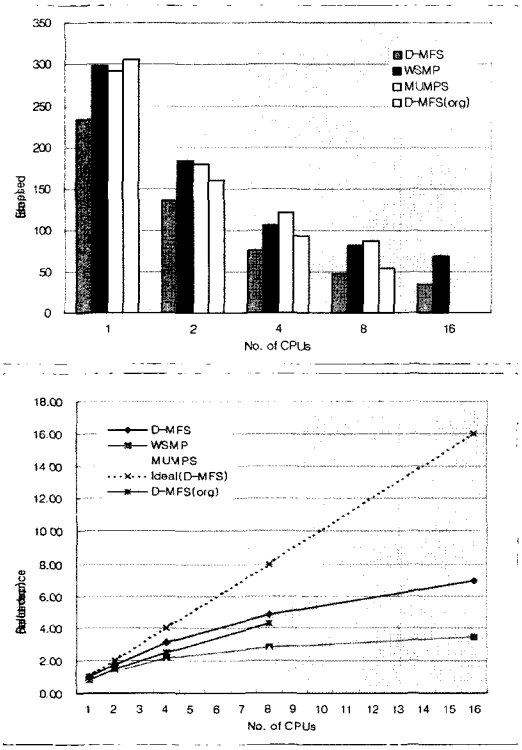

그림 11 Parallel performance comparison for 'Hole' model

아니라 병렬 성능에 있어서도 매우 우수함을 보이는 것을 알 수 있다.

앞의 결과에서는 16개의 프로세서를 사용한 경우까지 병렬 성능 비교를 했는데 보다 대규모의 문제를 보다 대규모의 시스템에서 처리 능력을 비교하기 위해 64개의 프로세서를 사용하여 80×80×80개의 고체요소로 구성된 문제를 푼 결과를 비교해 보았다. 이 경우 총 미지수의 개수는 1,574,640개이며, 그 비교 결과를 (표 1)에 나타내었다. 계산 시간은 인수화에 걸린 시간이며 요소행렬 계산에 걸린 시간은 제외한 것으로써 이 부분의 병렬화가 여의치 않은 WSMP나 MUMPS의 경우 이 부분을 포함하면 성능차가 더 커지게 된다.

표 1 Elapsed time for 80×80×80 solid element mesh on 64 CPUs

Solver	D-MFS	WSMP	MUMPS
Elapsed time	345 sec	418 sec	1073 sec

(표 1)의 결과에서 보는 바와 같이 MUMPS는 다른 두 경우에 비하여 성능이 크게 떨어지는 것을 알 수 있는데, 이는 앞에서 언급했던 행렬식 구성 방법과 병렬 행렬 계산 방법의 차이에 기인하는 것으로 본 연구 및 WSMP에서 사용한 방법이 문제의 규모가 커지고 프로세서 수가 많아짐에 따라 월등히 우수한 결과를 내는 것을 확인할 수 있다.

6. 결 론

본 연구에서는 초대형 구조해석에 있어서 범용성이나 수치적 안정성이 부족한 반복적 해법에 의존하지 않고 범용성을 가지면서도 보다 대규모의 문제를 해결할 수 있는 실용적인 기법으로써 영역기반 다중프론트 해법을 제안하고 그 성능의 우수성을 보였다. 제안된 기법은 현재 사용가능한 직접접 해법 중 최고 수준의 성능을 가질 뿐만 아니라, 다른 것들로는 도저히 해결이 불가능한 월등히 큰 규모의 문제를 풀 수 있다. 따라서 이를 활용함으로써 유한요소해석의 전 과정을 쉽게 높은 효율을 가지도록 병렬화하여 대규모의 구조해석을 보다 실용적으로 구현하여 연구에 활용하는데 많은 도움을 줄 수 있게 될 것으로 기대된다.

참 고 문 헌

1. Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., and Rixen, D., "FETI-DP: a dual-primal unified FETI method - part I: a faster alternative to the

two-level FETI method," *International Journal for Numerical Methods in Engineering.*, Vol. 50 2001, pp. 1523~1544

2. Bhardwaj, M., Pierson, K., Reese, G., Walsh, T., Day, D., Alvin, K., Peery, J., Farhat, C., and Lesoinne, M., "Salinas: A Scalable Software for High-Performance Structural and Solid Mechanics Simulations," *SC2002 Conference*, Baltimore, Maryland, November 16~22, 2002
3. "GeoFEM," URL: http://geofem.tokyo.rist.or.jp/w
4. "ADVENTURE Project," URL: http://adventure.q.t.u-tokyo.ac. jp/
5. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, "Ultrascale implicit finite element analysis in solid mechanics with over a half a billion degrees of freedom mark," *SC2004 Conference*, Pittsburgh, PA, November 6~12, 2004
6. S. J. Kim, C. S. Lee, J. H. Kim, M. Joh and S. Lee, "*IPSAP* : A High-performance Parallel Finite Element Code for Large-scale Structural Analysis Based on Domain-wise Multifrontal Technique," *SC2003 Conference*, Phoenix, AZ, November 17~21, 2003
7. 박시형, 윤영하, 김승조, 김정호, "분산 메모리 클러스터 시스템에서 직접 해법에 의한 1억 자유도 구조해석의 구현", 한국항공우주학회 추계학술발표회 논문집 (I), 2004. pp. 18~19
8. B. M. Irons. A frontal solution program for finite-element analysis. *Int J. Numerical Methods in Engineering*, 2:5~32, 1970
9. S. Duff and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, *ACM Trans. Math. Software,* 9, 302~325, 1983
10. J. W.-H. Liu, The multifrontal method for sparse matrix solution: Theory and practice, *SIAM Review, 23:82-109*, 1992
11. J. Dongarra, "FREELY AVAILABLE SOFTWARE FOR LINEAR ALGEBRA ON THE WEB," URL: http://www.netlib.org/utk/people/JackDongarra/la-sw.html
12. P.R. Amesoy, I.S. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng*, 2000.
13. A. Gupta, G. Karypis and V. Kumar, Highly Scalable Parallel Algorithms for Sparse Matrix Factorization, *IEEE Transactions on Parallel and Distributed systems*, 1995
14. V. Kumar, A. Grama, A, Gupta, G, Karypis. Introduction to parallel computing Design and analysis of algorithms, *The Benjamin/Cummings Publishing Company, Inc.* 1994
15. P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, J.Koster. MUltifrontal Massively Parallel Solver Users' guide(MUMPS Version 4.3), 2003
16. G. Karypis, V. Kumar, METIS, A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, Version4.0, 1998
17. A. Gupta, WSMP: Watson Sparse Matrix Package. Part I, II -direct solutions of symmetric sparse systems, *IBM Research Report*, 2000
18. J. H. Kim and S. J. Kim, A Multifrontal Solver Combined with Graph Partitioners, *AIAA Journal, Vol 38, No.8,* 1999
19. "The ScaLAPACK Project," URL: http://www.netlib.org/scalapack/index.html