대규모 자유도 문제의 구조해석을 위한 병렬 알고리즘

A Parallel Algorithm for Large DOF Structural Analysis Problems

김 민 석* 이 지 호†
Kim, Min-Seok Lee, Jee-Ho

요 지

본 논문에서는 대규모 자유도 시스템의 병렬처리를 위하여 2단계로 이루어진 영역분할법(Domain Decomposition Method) 기반의 병렬 알고리즘을 제안하였다. 분할된 영역의 내부 및 외부 경계를 상위영역문제로 정의하고 국부영역문제는 변위 경계조건이 모두 주어지는 분할영역에서의 Dirichlet 문제로 구성한다. 상위영역에서는 전체 상위영역에 대한 강성행렬의 어셈블이 필요없는 반복법을 통하여 변위를 구하고, 이를 바탕으로 국부영역에서 Multi-Frontal Sparse Solver (MFSS)를 이용하여 변위를 계산한다. 상위영역문제의 연산에서 프로세서 간의 데이터 교환을 최소화하여 계산효율을 유지하며, 동시에 해석 가능한 자유도를 증대시키는 병렬 PCG(Preconditioned Conjugate Gradient)법 기반의 알고리즘을 개발하였다. 제안된 알고리즘을 적용하여 수치해석을 수행한 결과, 프로세서 수가 증가할수록 계산성능의 손실없이 해석 가능한 자유도가 비례하여 증가하는 선형 확장성을 관찰할 수 있었으며, 대규모 자유도 문제에 효과적으로 사용 가능함을 확인하였다.

핵심용어 : 병렬 알고리즘, 영역분할법, PCG법, 유한요소 구조해석, 대규모 자유도 문제

Abstract

In this paper, an efficient two-level parallel domain decomposition algorithm is suggested to solve large-DOF structural problems. Each subdomain is composed of the coarse problem and local problem. In the coarse problem, displacements at coarse nodes are computed by the iterative method that does not need to assemble a stiffness matrix for the whole coarse problem. Then displacements at local nodes are computed by Multi-Frontal Sparse Solver. A parallel version of PCG(Preconditioned Conjugate Gradient Method) is developed to solve the coarse problem iteratively, which minimizes the data communication amount between processors to increase the possible problem DOF size while maintaining the computational efficiency. The test results show that the suggested algorithm provides scalability on computing performance and an efficient approach to solve large-DOF structural problems.

Keywords : *parallel algorithm, domain decomposition method, preconditioned conjugate gradient method, finite element structural analysis, large DOF problem*

1. 서 론

최근 공학에서 다루는 문제들은 점점 더 복잡해지고 있다. 정교한 수치모형을 통하여 고체, 유체, 열유동 등이 복합적으로 연관되어 있는 문제를 다루거나 대규모 자유도를 가진 구조물의 동해석을 수행하기도 한다.

과거의 슈퍼컴퓨터는 대부분 SMP(Symmetric Multi Processor) 시스템과 MPP(Massively Parallel Processor) 시스템이 차지하고 있었으나, 현재에는 다수의 고성능 프로세서를 네트워크로 연결하여 병렬 처리하는 클러스터 시스템이 슈퍼컴퓨터들 가운데 가장 높은 비율을 차지하고 있다. 2000년 전세계 슈퍼컴퓨터중 약 2.2%이었던 클러스터 시스템의 비율은 2005년 60%로 증가하였고, 현재는 80%이상이다(www.top500.org). 이러한 클러스터 시스템을 중·소 규모로 구성하면 경제적으로 복잡한 공학 시스템의 계산을 위한 고성능 컴퓨팅 환경을 구축할 수 있다.

† 책임저자, 정회원 · 동국대학교-서울 토목환경공학과 교수
Tel: 02-2260-3352 ; Fax: 02-2266-8753
E-mail: jeeholee@dgu.edu
* 동국대학교-서울 토목환경공학과 박사과정

• 이 논문에 대한 토론을 2010년 12월 31일까지 본 학회에 보내주시면 2011년 2월호에 그 결과를 게재하겠습니다.

클러스터 시스템에서 선형 문제의 계산을 위하여 활용할 수 있는 수치 라이브러리 중 대표적인 것으로는 ScaLAPACK (Scalable Linear Algebra PACKage; Blackford 등, 1997)과 PETSc(Portable, Extensible Toolkit for Scientific Computation; Balay 등, 2008) 등이 있다. 그러나 이러한 수치 라이브러리를 사용한 병렬처리는 전체 강성행렬을 각 프로세서들이 담당하는 영역으로 분할한 후 분할된 영역을 프로세서에 분배하여 병렬 처리한다. 따라서 이와 같은 수치라이브러리를 유한요소해석의 병렬처리에 이용할 경우, 분할된 영역들에 대하여 전체 행렬의 어셈블 과정과 병렬처리를 위한 전체 행렬의 분할 및 재분배 과정이 필요하다. 또한, 유한요소해석모델의 영역분할(Domain Decomposition)과 강성행렬의 영역분할이 서로 다르기 때문에 네트워크 자원과 시스템 메모리가 중복적으로 사용되어 비효율적이다.

Brill 등(2002)은 병렬화된 Bi-CGSTAB법을 제안하였다. 이 알고리즘에서는 전체 행렬에 Red-Black 배열방법을 적용하여 준비행렬을 구성하고 전체 행렬을 논리적으로 분할하게 된다. 각 프로세서는 분할된 행렬에 대하여 행렬과 벡터의 연산을 병렬 처리하여 문제의 해를 구한다. 이 방법은 공유메모리를 사용하여 프로세서 사이의 데이터 사용이 자유로운 MPP 시스템을 대상으로 개발되었으며, 전체 시스템 행렬의 구성과 분할과정이 비교적 용이하게 구현되는 장점이 있다. 그러나 클러스터 시스템과 같은 분산메모리 환경에서는 한 프로세서가 사용한 메모리의 변수 값을 다른 프로세서에서 직접 이용할 수 없으므로 네트워크를 통한 데이터 교환 알고리즘이 추가되지 않으면 직접 적용이 곤란하다.

일반적으로 구조해석의 병렬처리에 이용되는 영역분할 알고리즘에서는 전체 영역을 소규모의 분할영역(subdomain)의 형태로 구성하며, 각 분할영역의 경계에서의 변위는 주위 분할영역과 관련된 상위영역문제(coarse problem)로 정의되고 내부변위는 경계와 외력이 주어진 국부영역문제(local problem)로 구성된다(Mandel, 1993; Toselli 등, 2004). 본 논문의 알고리즘이 구현될 대상인 클러스터 기반의 병렬컴퓨터에서는 프로세서간의 데이터 교환이 네트워크 통신을 이용하여 이루어진다. 따라서 네트워크 속도의 영향과 네트워크 스위치의 병목현상으로 인한 성능저하를 피하기 위해서 프로세서들간 데이터 교환을 최소화해야 한다. 전체 구조 시스템의 해결에 직접법으로 계산하는 것은 전체 계산성능에서 네트워크의 영향이 크기 때문에 영역분할을 기반으로 하는 병렬처리에서는 직접법 보다 반복법을 사용하고 있다(Smith 등, 1996; Toselli 등, 2004).

이 논문에서는 대규모 자유도의 구조해석 문제를 효율적으로 계산하기 위하여 국부영역문제는 Multi-Frontal Sparse Solver를 이용하여 직접법으로 계산하고, 상위영역문제에는 PCG(Preconditioned Conjugate Gradient)법 기반의 반복법을 이용하는 2단계 영역분할법 기반의 병렬 알고리즘을 제안한다. 일반적으로 상위영역문제에서는 전체 상위영역의 강성행렬 구성을 요구하지만, 반복법을 이용하여 상위영역의 어셈블 과정을 전체적으로 수행하지 않고 해를 구하는 알고리즘을 설명한다. 제안되는 알고리즘에서는 각 프로세서에 할당되는 물리적 모델의 영역분할과 전체 시스템 강성행렬의 논리적 영역분할이 해석 전 과정에서 일치되어 클러스터의 메모리 사용 효율이 높아지며, 결과적으로 해석 가능 전체 자유도를 효율적으로 증가시킬 수 있다.

개발된 알고리즘을 기반으로 유한요소해석 프로그램을 작성하고, 이를 구조해석 계산에 사용하여 프로세서의 증가에 따른 해석시간 변화 및 분할영역의 국부영역 자유도 크기를 비교하여 성능을 분석하였다. 또한 자유도 크기에 따른 메모리 할당량을 비교하여, 전체 자유도를 효율적으로 병렬 시스템에 분산시켜 계산하는 대규모 자유도 문제에 적합한지 평가하였다.

2. 2단계 영역분할 알고리즘

2.1 2단계 영역분할

일반적으로 영역분할법을 이용한 구조해석에서는 해석대상의 영역을 다수의 분할영역으로 나누어서 계산한다. 전체영역의 강성행렬 **K**는 분할된 영역들에서 각각 구성하여 어셈블하게 된다. 그림 1에서 검은색과 회색의 절점들은 전체자유도 번호가 필요한 상위영역 절점이고 흰색절점은 국부영역 절점이다.

분할영역에서의 자유도를 국부영역 절점(Subdomain DOF Nodes)의 자유도 I와 상위영역절점(Coarse DOF Nodes)의 자유도 Γ로 나누면(그림 1), 분할영역 i에서의 강성행렬

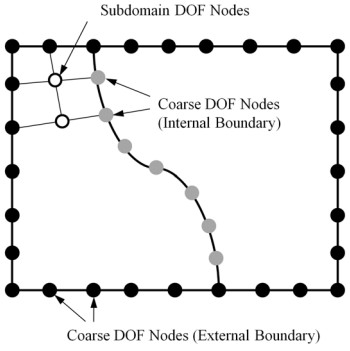

그림 1 영역분할과 상위영역 절점

$\mathbf{K}^{(i)}$는 다음과 같이 하위행렬의 결합형태로 나타낼 수 있다:

$$\mathbf{K}^{(i)} = \begin{bmatrix} \mathbf{K}_{II}^{(i)} & \mathbf{K}_{I\Gamma}^{(i)} \\ \mathbf{K}_{\Gamma I}^{(i)} & \mathbf{K}_{\Gamma\Gamma}^{(i)} \end{bmatrix} \tag{1}$$

전체영역이 n개의 분할영역으로 이루어지는 경우, 각 영역에서 계산된 $\mathbf{K}^{(i)}$를 어셈블한 전체 강성행렬 $\mathbf{K}$는 식 (2)와 같다.

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{II}^{(1)} & 0 & \cdots & \mathbf{K}_{I\Gamma}^{(1)} \\ 0 & \mathbf{K}_{II}^{(2)} & & \mathbf{K}_{I\Gamma}^{(2)} \\ \vdots & & \ddots & \vdots \\ \mathbf{K}_{\Gamma I}^{(1)} & \mathbf{K}_{\Gamma I}^{(2)} & \cdots & \mathbf{K}_{\Gamma\Gamma} \end{bmatrix} \tag{2}$$

식 (2)를 이용하면 외력 $\mathbf{F}$에 대하여 변위 $\mathbf{U}$를 구하는 선형방정식 $\mathbf{KU}=\mathbf{F}$는 식 (3a,b)로 표현된다.

$$\begin{bmatrix} \mathbf{K}_{II}^{(1)} & 0 & \cdots & \mathbf{K}_{I\Gamma}^{(1)} \\ 0 & \mathbf{K}_{II}^{(2)} & & \mathbf{K}_{I\Gamma}^{(2)} \\ \vdots & & \ddots & \vdots \\ \mathbf{K}_{\Gamma I}^{(1)} & \mathbf{K}_{\Gamma I}^{(2)} & \cdots & \mathbf{K}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \mathbf{u}_I^{(1)} \\ \mathbf{u}_I^{(2)} \\ \vdots \\ \mathbf{u}_\Gamma \end{bmatrix} = \begin{bmatrix} \mathbf{f}_I^{(1)} \\ \mathbf{f}_I^{(2)} \\ \vdots \\ \mathbf{f}_\Gamma \end{bmatrix} \tag{3a}$$

$$\mathbf{K}_{\Gamma\Gamma} = \mathbf{K}_{\Gamma\Gamma}^{(1)} + \mathbf{K}_{\Gamma\Gamma}^{(2)} + \cdots + \mathbf{K}_{\Gamma\Gamma}^{(n)} \tag{3b}$$

구성된 강성행렬 $\mathbf{K}$는 다음과 같이 $\mathbf{LR}$ 형태로 분해된다 (Toselli 등, 2004).

$$\mathbf{K} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{K}_{\Gamma I}^{(1)}\mathbf{K}_{II}^{(1)^{-1}} & \mathbf{K}_{\Gamma I}^{(2)}\mathbf{K}_{II}^{(2)^{-1}} & \cdots & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{K}_{II}^{(1)} & 0 & \cdots & \mathbf{K}_{I\Gamma}^{(1)} \\ 0 & \mathbf{K}_{II}^{(2)} & & \mathbf{K}_{I\Gamma}^{(2)} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S} \end{bmatrix} \tag{4a}$$

$$\mathbf{S} = \mathbf{K}_{\Gamma\Gamma} - \mathbf{K}_{\Gamma I}^{(1)}\mathbf{K}_{II}^{(1)^{-1}}\mathbf{K}_{I\Gamma}^{(1)} - \cdots - \mathbf{K}_{\Gamma I}^{(n)}\mathbf{K}_{II}^{(n)^{-1}}\mathbf{K}_{I\Gamma}^{(n)} \tag{4b}$$

식 (4b)에 식 (3b)를 대입하여 정리하면 $\mathbf{S}$는 식 (5)와 같이 분할영역들에서 계산된 $\mathbf{S}^{(i)}$의 합으로 정리된다:

$$\mathbf{S} = \sum_{i=1}^{n} \mathbf{S}^{(i)} \tag{5a}$$

$$\mathbf{S}^{(i)} = \mathbf{K}_{\Gamma\Gamma}^{(i)} - \mathbf{K}_{\Gamma I}^{(i)}\mathbf{K}_{II}^{(i)^{-1}}\mathbf{K}_{I\Gamma}^{(i)} \tag{5b}$$

식 (4a)를 식 (3a)에 대입하고 $\mathbf{L}$ 행렬을 제거하여 정리하면 전체문제는 분할된 영역에서 국부영역인 영역내부 자유도의 문제와 상위영역인 영역 경계에서 발생하는 문제로 구분하는 식 (6)으로 정리된다.

$$\begin{bmatrix} \mathbf{K}_{II}^{(1)} & 0 & \cdots & \mathbf{K}_{I\Gamma}^{(1)} \\ 0 & \mathbf{K}_{II}^{(2)} & & \mathbf{K}_{I\Gamma}^{(2)} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{S} \end{bmatrix} \mathbf{U} = \begin{bmatrix} \mathbf{f}_I^{(1)} \\ \mathbf{f}_I^{(2)} \\ \vdots \\ \mathbf{g}_\Gamma \end{bmatrix} \tag{6a}$$

$$\mathbf{g}_\Gamma = \mathbf{f}_\Gamma - \mathbf{K}_{\Gamma I}^{(1)}\mathbf{K}_{II}^{(1)^{-1}}\mathbf{f}_I^{(1)} - \cdots - \mathbf{K}_{\Gamma I}^{(n)}\mathbf{K}_{II}^{(n)^{-1}}\mathbf{f}_I^{(n)} \tag{6b}$$

식 (6a)의 마지막 행을 전개하면, 경계면에서의 변위 $\mathbf{u}_\Gamma$를 계산하기 위한 상위영역문제 식 (7)이 얻어진다.

$$\mathbf{S}\mathbf{u}_\Gamma = \mathbf{g}_\Gamma \tag{7}$$

상위영역문제의 계산에 필요한 $\mathbf{S}$ 행렬과 $\mathbf{g}_\Gamma$ 벡터를 구성하기 위하여는 각 분할영역들에 분산되어 있는 $\mathbf{S}^{(i)}$와 $\mathbf{g}_\Gamma^{(i)}$의 어셈블이 필요하다. 그러나 상위영역의 어셈블은 벡터 뿐만 아니라 행렬 정보의 교환을 필요하기 때문에 네트워크 자원의 소모가 커지게 된다. 따라서 상위영역문제의 해결을 위해서는 어셈블 과정을 필요하지 않는 반복법을 병렬화하여 이용하는 것이 효율적이다. 상위영역에서 계산된 $\mathbf{u}_\Gamma^{(i)}$를 이용하면 각 분할영역의 국부영역문제는 Dirichlet 문제로 해결할 수 있다. 국부영역에서의 변위는 식 (8)과 같다.

$$\mathbf{u}_I^{(i)} = \mathbf{K}_{II}^{(i)^{-1}}(\mathbf{f}_I^{(i)} - \mathbf{K}_{I\Gamma}^{(i)}\mathbf{u}_\Gamma^{(i)}) \tag{8}$$

2.2 반복법을 이용한 상위영역의 병렬 알고리즘

일반적인 영역분할법에서는 분할된 영역에서의 강성행렬 구성과 아울러 Schur 행렬 $\mathbf{S}^{(i)}$구성 과정을 병렬처리한다. 즉, 각 분할영역이 할당된 프로세서에서 식 (1)의 강성행렬 $\mathbf{K}^{(i)}$를 계산하고 상위영역과 국부영역으로 구분하여 식 (5b)와 같은 분할영역 i에서의 Schur 행렬 $\mathbf{S}^{(i)}$를 구성한다. 이 때, 상위영역의 변위는 분할영역에서 상위영역 문제를 어셈블하여 하나의 프로세서에서 계산하거나 이를 다시 각 프로세서로 할당하여 병렬 계산하는 방법이 사용된다.

그러나 이러한 방법은 MPP 시스템을 위하여 개발된 것으로 그림 2와 같이 상위영역 문제에서 Schur 행렬 $\mathbf{S}$의 어셈블이 필요하며, 병렬 반복법으로 $\mathbf{S}$를 계산하는 경우, 전체 어셈블된 행렬을 분할하여 각 프로세서에 재할당하기 때문에 프로세서가 담당하고 있는 해석모형의 상위영역과 할당된 $\mathbf{S}$ 행렬 부분이 일치하지 않을 수 있다. 이러한 기존 알고리즘을 클러스터 시스템에 적용할 경우, $\mathbf{S}$ 행렬의 어셈블, 어셈블된 행렬의 분할, 상위 영역 계산, 계산된 결과 값 재분배 등의 과정을 거치게 되므로 클러스터 노드 간 데이터 통신량

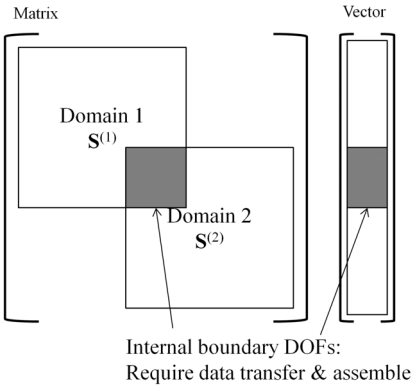

그림 2 Schur 행렬의 어셈블

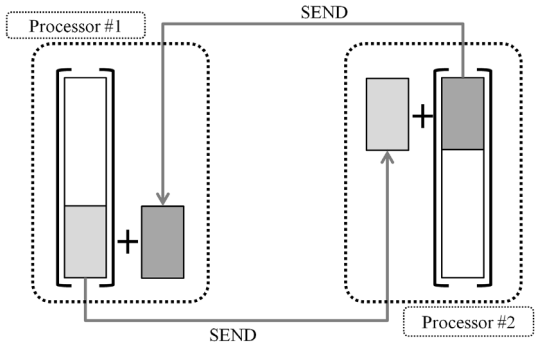

그림 3 분할영역의 벡터 어셈블

이 크게 증가하고, 이는 전체적인 계산 성능을 심각히 저하시킨다.

다수 영역으로 분할된 모형의 구조해석에서 실제 공유 데이터가 필요한 부분은 내부경계부분(그림 1의 회색절점)이다. 따라서 그림 3에서 나타내는 바와 같이 전체 데이터 중 내부경계부분의 필요 데이터만을 서로 경계면을 공유하는 프로세서로 전달하여 부분적인 어셈블을 수행하면 프로세서 간 통신량을 효과적으로 감소시킬 수 있다. 이때 상위영역문제의 계산에 반복법을 이용하면, 각 분할영역의 $\mathbf{S}^{(i)}$는 직접 전달하지 않고 필요한 벡터 데이터만을 MPI(Message Passing Interface; Gropp, W. 등, 1994)로 통신하여 클러스터 시스템에서 문제를 효율적으로 계산할 수 있다.

그림 4는 전체 어셈블된 벡터 $\hat{\mathbf{a}}$와 비교하여 분할영역에서 가지고 있는 벡터들의 차이점을 보여준다. 여기서, 분할영역에서 가지고 있는 벡터들 중 어셈블되지 않은 벡터는 $\mathbf{a}^{(i)}$이고 부분적인 어셈블을 통하여 분할영역에서 보유하게 되는 어셈블된 벡터는 $\hat{\mathbf{a}}^{(i)}$이다.

위에서 고찰한 내용을 기반으로 클러스터 시스템에서 식 (7)의 상위영역문제의 해를 효율적으로 구하기 위하여 기존의 병렬 PCG법을 그림 5와 같이 수정하였다. 그림 5의 과

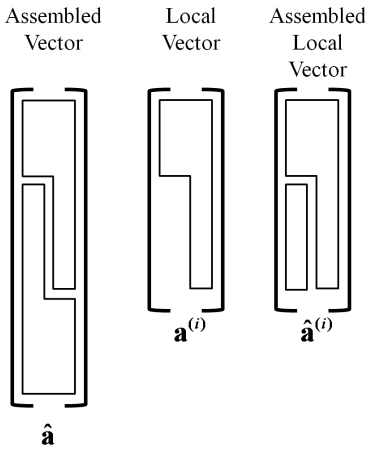

그림 4 전체벡터와 분할영역벡터

Initialize $\mathbf{u}^{(i)}$, $\mathbf{r}^{(i)}$, $\mathbf{d}^{(i)}$

do $\| \mathbf{r} \| <$ tol

$$\lambda_k = \frac{\sum \left(\hat{\mathbf{d}}_k^{(i)}\right)^T \mathbf{r}_k^{(i)}}{\sum \left(\hat{\mathbf{d}}_k^{(i)}\right)^T \mathbf{S}^{(i)} \hat{\mathbf{d}}_k^{(i)}} \qquad \text{Step 1}$$

$$\mathbf{u}_{k+1}^{(i)} = \mathbf{u}_k^{(i)} + \lambda_k \hat{\mathbf{d}}_k^{(i)} \qquad \text{Step 2}$$

$$\mathbf{r}_{k+1}^{(i)} = \mathbf{r}_k^{(i)} - \lambda_k \mathbf{S}^{(i)} \hat{\mathbf{d}}_k^{(i)} \qquad \text{Step 3}$$

$$\alpha_{k+1} = \frac{\sum \left(\hat{\mathbf{r}}_{k+1}^{(i)}\right)^T \mathbf{r}_{k+1}^{(i)}}{\sum \left(\hat{\mathbf{r}}_k^{(i)}\right)^T \mathbf{r}_k^{(i)}} \qquad \text{Step 4}$$

$$\hat{\mathbf{d}}_{k+1}^{(i)} = \hat{\mathbf{r}}_{k+1}^{(i)} + \alpha_{k+1} \hat{\mathbf{d}}_{k+1}^{(i)} \qquad \text{Step 5}$$

end

그림 5 수정된 병렬 PCG 알고리즘

정은 분할영역에서 PCG 기반의 반복법으로 해를 구하는 과정으로 각 프로세서에서 독립적으로 진행되며, 프로세서 간의 벡터 데이터 교환은 Step 1과 Step 4 과정에서 그림 3과 같은 방식으로 수행된다.

그림 5의 수정 병렬 PCG 알고리즘을 구체적으로 설명하면 다음과 같다. Step 1에서는 각 프로세서에서 식 (5b)로 구성되는 $\mathbf{S}^{(i)}$를 이용하여 방향벡터 $\mathbf{d}$를 어셈블하고, 이를 동기화한다. 아래 식 (9) 및 (10)과 같이 모든 분할영역에 대하여 상변과 하변을 각각 계산하고 이를 이용하여 이동길이 λ를 계산한다.

$$\hat{\mathbf{d}}^{\mathrm{T}} \hat{\mathbf{r}} = \sum_{i=1}^{ND} \hat{\mathbf{d}}^{(i)\mathrm{T}} \mathbf{r}^{(i)} \qquad (9)$$

$$\hat{\mathbf{d}}^{\mathrm{T}} \hat{\mathbf{S}} \hat{\mathbf{d}} = \sum_{i=1}^{ND} \hat{\mathbf{d}}^{(i)\mathrm{T}} \mathbf{S}^{(i)} \hat{\mathbf{d}}^{(i)} \qquad (10)$$

Step 2에서 **u**는, **d**가 Step 1 과정에서 어셈블 되어 있고 λ는 모든 프로세서에서 같은 값을 가지고 있으므로, 모든 경계영역에서 동일한 값을 가지게 된다. 따라서 별도의 어셈블 과정이 불필요하다. Step 3에서는 각 프로세서에서 해당 분할영역 각각의 잔차 벡터 **r**을 계산한다. Step 4에서는 데이터 교환을 통하여 **r**을 어셈블한다. 마지막으로 Step 5에서 이를 이용하여 $(k+1)$ 단계의 방향벡터를 설정한다.

Step 1과 Step 4의 벡터 어셈블 및 동기화 과정에서 각 분할영역 간의 데이터 교환이 필요하며, 제안하는 알고리즘에서는 벡터 전체가 아닌 필요한 부분만을 추출하여 MPI 통신을 통한 데이터 교환으로 어셈블 과정을 완료하여 데이터 통신량을 최소화한다. 아울러 벡터 데이터의 전달 과정이 해석모형의 영역분할을 동일하게 이용하게 되므로, 각 클러스터 노드에서 요구되는 메모리 크기가 자유도 증가에 따라 선형적으로 늘어나고, 결과적으로 계산 가능한 전체 자유도를 클러스터 노드의 추가(즉, 프로세서 및 메모리의 동시 증가)에 따라 계산성능의 저하없이 비례하여 증가시킬 수 있다.

2.3 알고리즘 구현

대규모 자유도 문제의 구조해석을 병렬 처리하기 위해 먼저 수치모형을 클러스터 시스템에 분배할 수 있도록 프로세서 수로 모형을 분할해야 한다. 그림 6은 전체 노드, 요소, 경계조건으로부터 영역을 분할하여 분할영역으로 전달하고 각 분할영역들이 수치모형을 구성하는 과정을 나타낸다. 전체 영역의 분할은 그래프 이론을 바탕으로 하는 영역 분할 라이브러리인 METIS를 사용하였다. 메인 프로세서에서는 전체 절점정보와 요소연결 정보를 이용하여 주어진 분할영역 수로 영역을 분할하고 분할영역에서 각각 요소들의 강성행렬을 구성할 수 있도록 각 분할영역에 속한 절점번호 및 요소번호를 분할영역으로 전달한다. 또한 모든 분할영역에서의 상위영역에 해당하는 절점을 검색하여 상위영역의 자유도번호를 설정하고, 이 정보는 해당 상위영역 절점를 사용하는 분할영역들로 전달된다.

분할영역에서는 해석을 위한 수치모형 준비 작업을 수행하게 된다. 주 프로세서에서 전달받은 내용을 바탕으로 각 분할영역에서 사용하는 절점좌표, 요소연결정보 및 경계조건 등을 데이터로 구성하고, 상위영역 절점들은 상위영역의 자유도번호와 분할영역의 내부 자유도번호 사이의 연결정보를 만든다.

구성된 데이터를 기반으로 분할영역에 대한 강성행렬을 어셈블하고(식 1), 이로부터 상위영역문제를 풀기 위해 Schur 행렬을 구성한다(식 5b). 2.2절에서 기술한 수정된 병렬

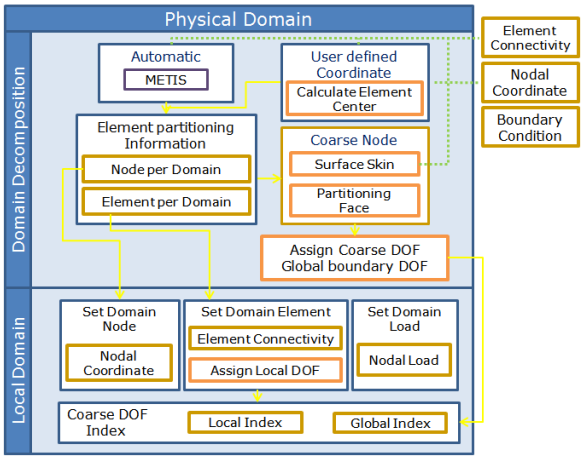

그림 6 영역분할과 분할데이터 전달 알고리즘

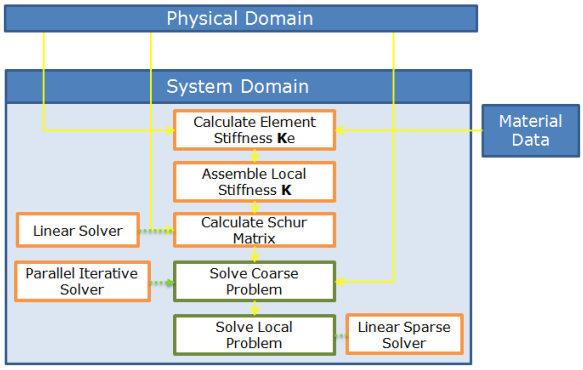

그림 7 상위영역과 국부영역변위계산 알고리즘

PCG법으로 상위영역의 변위를 계산하고, 계산된 상위영역에서의 변위를 이용하여 국부영역의 Dirichlet 경계조건 문제를 풀게 된다(식 8). 국부영역문제에서는 MFSS를 사용하여 내부노드들의 변위를 계산한다. 그림 7은 각 분할영역에서 강성행렬 구성부터 내부변위 계산까지 과정을 보여준다.

3. 수치해석을 통한 성능평가

3.1 시스템구성

제안된 병렬 구조해석 알고리즘을 Fotran 95 프로그램으로 구현하였으며, 사용 환경은 그림 8과 같이 Xserve G5 클러스터 시스템이다. 클러스터는 8개의 노드로 구성되며, 각 클러스터 노드는 2개의 64-bit PowerPC 970 2GHz 프로세서를 가지고 있어, 총 16개의 프로세서가 가용하다. 클러스터 노드 당 RAM 메모리는 2GB이며, 2개의 Gigabit Ethernet을 채널 결합하여 구성한 네트워크 환경을 사용하였다. 운영체제는 FreeBSD Unix를 기반으로 하는 Mac OS X 10.4 Unix Kernel을 사용하였다. MPI 환경은 MPICH2 1.0.7로 구축하였으며, 영역분할을 위한 METIS

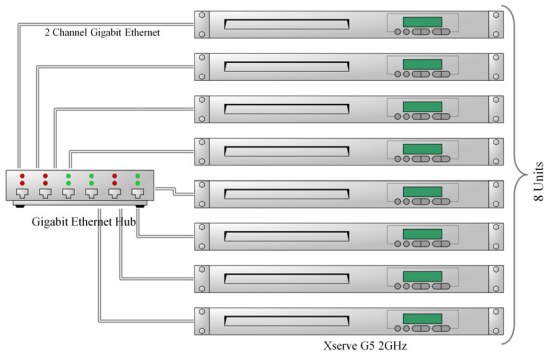

그림 8 프로그램 수행환경

(Karypis and Kumar, 1999), 행렬과 벡터 연산을 위한 Altivec BLAS, 그리고 비대칭 멀티프론탈 계산법을 기반으로 하는 UMFPACK(Davis, 2004)을 외부 라이브러리로 사용하였다.

3.2 해석모형

본 논문에서 제안된 알고리즘의 성능평가를 위하여 그림 9와 같은 평면 구조물의 유한요소해석을 수행하였다. 구조물의 좌측과 하단을 단순 구속하고 우측에 압력하중을 작용하였다. 재료물성은 선형탄성으로 가정하였으며, 탄성계수는 23GPa, 포아송비는 0.18을 사용하였다.

그림 9의 구조해석 문제를 1개부터 16개의 분할영역(sub-domain)으로 나누고 각 분할영역을 1개의 프로세서에 할당하여 제안된 알고리즘으로 계산을 수행하였다. 그림 10은 8개의 하위영역으로 분할된 예이다. 각 분할영역의 내부경계 및 외부경계의 절점은 모두 상위영역을 구성하게 되고, 이를 제외한 부분은 국부영역 절점이 된다.

제안된 알고리즘의 성능평가를 위한 관찰 항목은 다음과 같다:

1) 프로세서 수의 증가에 따른 해석시간 변화
2) 전체 계산시간 중 상위영역의 계산이 미치는 영향
3) 상위영역 크기와 메모리 사용량
4) 분할영역의 자유도 크기와 메모리 사용량

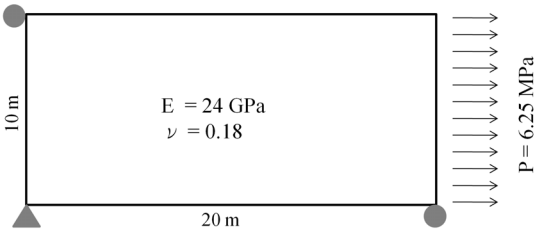

그림 9 해석 문제

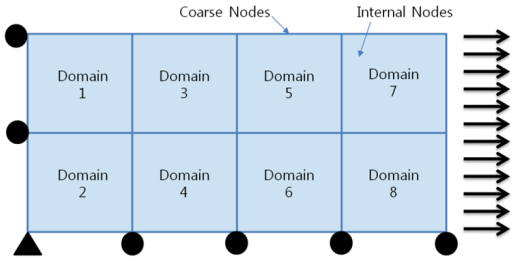

그림 10 해석모형의 영역분할

3.3 결과 분석

3.2절의 1)과 2)의 계산 성능평가를 위한 유한요소해석에는 총 20,301개의 절점과 20,000개의 4절점 평면요소로 이루어진, 총 40,300개의 자유도를 가진 모형을 사용하였다. 3.2절의 3)의 성능평가에는 총 40,000~326,000 자유도를 가지는 4 종류의 유한요소망을 사용하고, 4)에는 1개의 클러스터 노드의 최대 해석가능 크기인 총 60,000 자유도의 유한요소망을 사용하여 프로세서가 요구하는 메모리 크기의 변화를 분석하였다. 모든 성능평가에는 10회 반복 수행하여 얻은 결과의 평균값을 이용하였다.

3.3.1 프로세서 수의 증가에 따른 해석시간 변화

프로그램의 성능을 평가하기 위하여 프로세서 수의 증가에 따른 해석시간의 변화를 살펴보았다. 프로세서 수의 증가에 따른 유한요소해석 수행시간의 비교 결과는 표 1에 나타나 있다. 계산시간은 프로세서 수의 증가에 따라 감소하였으며 단일 프로세서 환경과 비교하여 16개의 프로세서를 이용한 경우 88%의 시간이 감소하였다. 2개의 프로세서를 이용한 경우 단일 프로세서에서 처리한 경우에 비하여 계산시간이 약간 증가한 것을 볼 수 있는데, 이는 단일 프로세서 처리의 경우 영역분할 과정이 필요없고 2개의 분할영역에서는 Schur 행렬의 구성이 필요하기 때문에 약간의 시간 증가(0.03%)가 있었다. 그러나 클러스터 노드에 데이터가 분할되어 각 클러스터 노드가 다루는 데이터의 양이 줄어들며, 이에 따른 모델구성시간의 감소로 전체적인 프로그램 수행시간은 약 32%

표 1 프로세서 수 증가에 따른 해석시간 변화

프로세서 수	상위영역 자유도	모델구성시간		시스템구성 및 해석		총 소요시간	
		시간 (sec)	비율	시간 (sec)	비율	시간 (sec)	비율
1	-	17.93	1.00	24.81	1.00	42.74	1.00
2	1096	3.57	0.20	25.58	1.03	29.15	0.68
4	1494	0.96	0.05	9.34	0.38	10.30	0.24
8	2048	0.99	0.06	3.84	0.15	4.84	0.11
16	2948	2.23	0.12	2.99	0.12	5.22	0.12

단축되었다.

모델구성 시간의 경우 프로세서 8개까지는 감소를 보이고 16개에서 증가하는 결과가 나타났는데, 이는 분할된 영역의 개수가 증가하면서 메인프로세서에서 처리하는 영역분할 시간이 증가하고 영역을 분할하는 동안 다른 프로세서들은 대기상태이기 때문이다. 프로세서의 수가 증가할수록 내부 자유도는 감소하고 분할영역에서 구성하는 강성행렬의 크기가 작아지게 되어 전체적인 계산시간이 감소하게 된다.

3.3.2 전체 계산시간 중 상위영역의 계산이 미치는 영향

병렬계산에 사용되는 프로세서 수가 늘어나 영역분할이 많아지면 국부영역의 자유도(분할영역의 자유도 중 상위영역에 속하는 자유도를 제외한 나머지)는 감소하고 상위영역의 자유도가 증가하게 된다. 표 2는 각 분할영역의 자유도 크기 변화에 따라 국부영역문제와 상위영역문제가 전체 해석시간에서 차지하는 비율을 보여준다.

병렬 PCG법을 이용한 상위영역의 계산은 자유도 크기에 따라서 약 100~200회의 반복 후 수렴하였고, 계산시간은 0.85초에서 2.08초까지 증가하였다. 상위영역의 계산이 전체 계산시간에서 차지하는 비율은 프로세서 개수가 증가하면서 약 3%에서 70%까지 증가하였다. 70%까지 증가한 경우 내부영역의 자유도 크기보다 상위영역의 자유도 크기가 크고, 이에 따라 반복 횟수와 통신량이 증가해서 전체 계산시간 중 차지하는 비율이 커지게 된다.

3.3.3 상위영역 크기에 따른 메모리 사용량

표 1과 2에서 나타난 바와 같이 영역 분할이 증가하게 되면 상위영역문제 크기가 증가한다. 이러한 상위영역 자유도의 크기가 각 클러스터 노드의 메모리 사용에 미치는 영향을 알아보기 위하여 각 분할영역의 자유도 크기를 일정하게 유지하고 상위영역의 자유도 크기를 증가시키며 메모리 사용량을 측정하였다.

표 3은 프로세서 수를 2개에서 16개까지 늘리면서 전체 문제의 자유도를 40,302에서 326,012까지 증가시켰을 때의 각 프로세서의 메모리 사용 결과이다. 하나의 분할영역이 담당하는 자유도의 크기를 약 20,000으로 유지하였을때, 상위영역문제의 자유도가 1,096에서 7,613까지 증가함에도 각 프로세서의 메모리 사용은 530~540 MB로 거의 일정하게 유지됨을 관찰할 수 있다.

3.3.4 분할영역의 자유도 크기에 따른 메모리 사용량

마지막으로 분할영역의 자유도 크기와 각 프로세서의 메모리 사용량의 상관관계를 알아보기 위하여 단일 클러스터 노드

표 2 전체 계산시간 중 상위영역문제 비율

프로세서 수	2	4	8	16
국부영역 자유도	19,602	9,702	4,782	2,335
상위영역 자유도	1,096	1,494	2,048	2,948
총 계산시간(sec)	25.58	9.34	3.84	2.99
상위영역 계산시간(sec)	0.85	1.26	1.49	2.08
상위영역비율(%)	3.31	13.49	38.90	69.70

표 3 프로세서 수 증가에 따른 메모리 사용량

프로세서 수	2	4	8	16
총 자유도	40,302	80,910	160,602	326,012
분할영역 자유도	20,151	20,228	20,075	20,376
상위영역 자유도	1,096	2,125	3,812	7,613
메모리사용(MB)	530	540	530	540

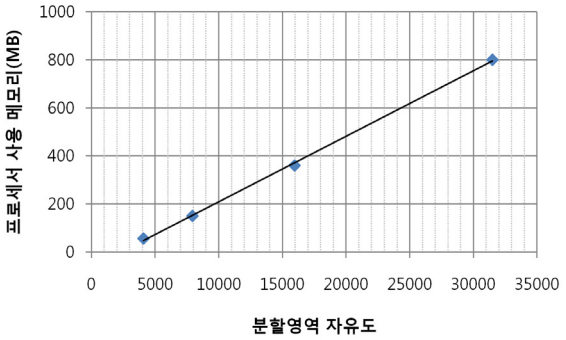

그림 11 자유도 크기에 따른 메모리 사용

(1개의 클러스터 노드는 2개의 프로세서를 가짐)에서 해석 가능한 최대 자유도인 총 62,875개의 자유도를 가진 유한요소망으로 그림 9의 구조해석을 수행하였다. 각 분할영역에 할당된 자유도 크기를 4,096(16개 분할영역)에서 31,511(2개 분할영역)까지 변화시켰다.

그림 11은 각 분할영역의 자유도 크기에 따른 소요 메모리 사용량들의 변화를 보여주며 선형회귀분석으로부터 다음과 같은 선형 관계식을 얻을 수 있다:

$$M = 0.0273n - 64.8 \qquad (11)$$

여기서, M은 각 프로세서의 소요 메모리 크기(단위 MB)이며, n은 분할영역의 자유도 크기이다. 일반적으로 RAM 메모리의 25%를 시스템 사용영역으로 보며, 이를 제외한 메모리 용량(프로세서 당 750MB)을 식 (11)에 대입하면 각 분할영역에 할당 가능한 자유도 크기의 이론치는 29,847개이다. 본 연구에서 사용한 클러스터 시스템은 16개의 프로세서로 이루어져 있으며, 따라서 제안된 2단계 영역분할 알고리즘이 프로세서 증가에 따른 자유도의 선형확장성(scalability)을 제공한다면 계산할 수 있는 최대 자유도는 476,800개가 될 것으로

예측할 수 있다. 실제 수치실험 결과 총 232,221개의 절점과 231,200개의 4절점 평면요소로 구성되어 462,941개 자유도를 갖는 구조해석 문제까지 계산을 수행할 수 있었으며, 이는 이론치에 매우 근접한 값으로 선형확장성을 만족한다고 볼 수 있다.

4. 결 론

본 논문에서는 대규모 자유도 구조문제의 병렬계산을 위하여 영역분할법 기반의 2단계 알고리즘을 제안하였다. 분할영역의 국부영역문제는 각 프로세서에서 MFSS를 이용하여 직접법으로 동시에 계산하고, 상위영역문제는 데이터 통신을 최소화하고 수정 PCG법 기반으로 병렬처리하는 알고리즘이다. 제안된 알고리즘을 구조해석에 적용하여 성능을 측정한 결과, 프로세서 수가 증가할수록 계산에 소요되는 시간이 효과적으로 감소하거나 해석 가능한 자유도 크기가 비례하여 증가함을 관찰할 수 있었다. 또한 각 프로세서의 메모리와 분할영역에 할당된 자유도 크기 간의 선형 관계식을 추출하여 계산 가능한 최대 자유도의 이론치를 산정하고, 실제 클러스터 시스템에서 구조해석을 수행하여 이론치에 부합하는 결과를 확인하였다. 이를 통하여 본 논문에서 제안된 2단계 영역분할 알고리즘이 프로세서 증가에 따른 자유도의 선형확장성(scalability)을 성공적으로 제공하는 것을 확인할 수 있었으며, 클러스터 노드의 추가에 따라 계산성능의 손실 없이 대규모 자유도의 구조해석을 수행할 수 있는 병렬 컴퓨터 시스템의 구축을 가능하게 할 것으로 기대한다.

참 고 문 헌

Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley M., McInnes, L.C., Smith, B., Zhang, H. (2008) *PETSc User''s Manual, Report ANL 95/11, Revision 3.0.0,* Argonne National Laboratory, USA.

Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C. (1997) *ScaLAPACK Users' Guide,* SIAM Publications, Philadelphia, USA.

Brill, S.H., Pinder, G.F. (2002) Parallel Implementation of the Bi-CGSTAB Method with Block Red-Black Gauss-Seidel Preconditioner Applied to the Hermite Collocation Discretization of Partial Differetial Equations, *Parallel Computing,* 28, pp.399~414.

Davis, T.A. (2004) A Column Pre-Ordering Strategy for the Unsymmetric-Pattern Multifrontal Method, *ACM Transactions on Mathematical Software,* 30(2), pp.165~195.

Gropp, W., Lusk, E., Skjellum, A. (1994) *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, USA.

Karypis, G., Kumar, V. (1999) A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM Journal on Scientific Computing,* 20, pp.359~392.

Mandel J. (1993) Balancing Domain Decomposition, *Communications on Numerical Methods in Engineering.*, pp.233~241.

Smith, B.F., Bjørstad, P., Gropp, W.D. (1996) *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations,* Cambridge University Press, UK.

Toselli, A., Widlund, O.B. (2004) *Domain Decomposition Methods-Algorithms and Theory,* Springer, Berlin-Heidelberg-New York.

● 논문접수일 2010년 1월 12일
● 논문심사일
1차 2010년 1월 22일
2차 2010년 9월 16일
● 게재확정일 2010년 9월 17일