병렬 컴퓨터를 이용한 형상 압연공정 유한요소 해석의 분산병렬처리에 관한 연구

권 기 찬*· 윤 성 기**

(1999년 6월 23일 접수)

Finite Element Analysis of Shape Rolling Process using Distributive Parallel Algorithms on Cray T3E

Kie-Chan Kwon and Sung-Kie Youn

Key Words: Parallel Processing(병렬처리), Distributive Processing(분산처리), Finite Element Method(유한요소법), Shape Rolling Process(형상 압연공정)

Abstract

Parallel Approaches using Cray T3E which is MPP (Massively Parallel Processors) machine are presented for the efficient computation of the finite element analysis of 3-D shape rolling processes. Domain decomposition method coupled with parallel linear equation solver is used. Domain decomposition is applied for obtaining element tangent stiffness matrices and residual vectors. Direct and iterative parallel algorithms are used for solving the linear equations. Direct algorithm is parallel version of direct banded matrix solver. For iterative algorithms, the well-known preconditioned conjugate gradient solver with Jacobi preconditioner is also employed. Moreover a new effective iterative scheme with block inverse matrix preconditioner, which is named by present authors, is presented and its results are compared with the one using Jacobi preconditioner. PVM and MPI are used for message passing and synchronization between processors. The performance and efficiency of each algorithm is discussed and comparisons are made among different algorithms.

1. 서 론

유한요소법을 이용하여 형상 압연공정 해석 및 설계를 수행하려는 초기의 시도는 변분법을 이용하여 압연 재료의 변형에너지율을 나타내는 범함수를 설정하여 압연중 재료의 폭퍼짐과 늘어남을 해석하였으며,[1~4] 최근에 이르러 3차원 강소성 유한요소법을 사용하여 형강압연중 발생하는 소재의 변형을 세밀하게 해석하였다.[5~11] H형강 같은 복잡한 단면을 갖는 압연 해석에서는 계산 효율을 증대시키기 위하여 3차원 유한요소법과 슬래브법을 결합한 해석 방법이 제안되었다. 또한 압연 공정중에 압연 방향에 수직한 소재의 단면이 평면으로 유지된다는 가정하에 계산량이 대폭 감소된 3차원 해석기법이 개발되어 다단 I-형강 압연에 적용되었다.[12,13] 이러한 3차원 유한요소법을 이용한 형강압연의 해석은 다단 공정으로 이루어지는 형상 압연의 공정설계에의 적용시 막대한 계산시간이 소요된다.

소성 변형과 같은 비선형 공학 문제의 3차원 해석시 야기되는 계산시간을 줄이기 위한 해결 방안으로 병렬처리 컴퓨터에 대한 관심이 최근 증가되고 있다. 프로세서 자체의 속도증가는 거의 한계에 도달했으며 더 이상의 속도증가를 위한 노력보다는 느린 프로세서들을 연결해서 사용하는 방법이 오히려 경쟁력이 있을 수 있다. 이러한 컴퓨터 분야의 환경변화는 각 분야에서 "하

* 회원, 한국과학기술원 대학원 기계공학과
E-mail : kkc@hpsys03.kaist.ac.kr
TEL : (042)869-3074 FAX : (042)869-3095
** 회원, 한국과학기술원 기계공학과

나의 큰 작업을 여러 개의 작은 작업으로 나누어 여러 개의 프로세서에서 동시에 처리하게 하는 방법" 인 병렬처리법(Parallel Processing)에 대한 관심을 증가시키고 있으나, 병렬 컴퓨터를 이용한 금속가공 문제 해결의 적용[14~17]은 미미한 실정이다.

대부분의 병렬처리에 관한 연구는 MPP (Massively Parallel Processors)를 사용하여 이루어진다. MPP는 현재까지 개발된 컴퓨터 중에서 가장 강력한 성능을 발휘하며, 병렬처리 환경의 한 축을 형성한다. 그러나 일반적으로 MPP는 매우 고가의 장비로 이를 모두가 사용하기는 어려운 단점을 가지고 있다. Cray사의 T3E와 같이 각 프로세서가 메모리를 공유하지 않고 독립적인 메모리 영역을 소유하고 있을 경우, 각 프로세서간의 자료 교환은 자체 소프트웨어나 PVM,[18] MPI[19,20] 등을 사용한다. 이는 컴퓨터들을 망(network)으로 엮어서 구축되는 하나의 가상적인 병렬 컴퓨터 환경과 대응되며, 이러한 환경에서 하나의 작업을 각 프로세서에 나누어 계산하도록 하는 것으로 분산처리(distributed computing)라고 한다. 분산처리는 컴퓨터간의 자료교환을 위해 많은 시간이 소요되기 때문에 이를 이용한 병렬처리는 상대적으로 비효율적이라 많은 연구가 이루어지지 않았다. 그러나 분산처리가 가지는 큰 장점은 비용으로, 사용자는 비교적 저가의 장비들을 묶어 대형의 문제를 풀 수 있어 적절한 알고리듬을 사용하면 큰 효과를 얻을 수 있다고 기대된다.

병렬 유한요소 알고리듬으로는 문제영역을 여러 작은 부영역들로 분할하여 각 프로세서가 할당된 부영역을 계산하여 종합하는 영역분할법 (domain decomposition method)이 선호되고 있다.[21~24] 영역분할법에서는 행렬방정식의 해를 구하기 위한 솔버(solver)로 직접솔버(direct solver)와 축차솔버(iterative solver) 등이 가능하나, 각 방법의 성능에 대한 비교 연구는 거의 이루어지지 않고 있는 실정이다. 직접솔버는 유한요소의 순차적 계산에는 장점이 있지만, 병렬처리의 경우 많은 프로세서를 사용할 때 계산시간의 감소에 한계가 있기 때문에 영역분할법에서는 CG(Conjugate Gradient)와 GS(Gauss-Siedel) 방법 등과 같은 축차솔버를 ILU(Incomplete LU decomposition), Jacobi, SOR 법 등과 같은 선결조건행렬 (preconditioner matrix)과 결합해 병렬화하는 연구가 많이 이루어지고 있다.[25] 그러나 축차솔버의 사용은 계산량을 예측하기가 어렵고 경우에 따라 해의 수렴속도가 매우 느리게 되는 단점을 가지고 있다.

본 연구에서는 3차원 형상 압연공정의 유한요소해석시 야기되는 계산시간을 효과적으로 줄이기 위해 Cray사의 T3E 컴퓨터를 이용해 분산 병렬처리한다. 영역분할법에 위해 병렬처리를 구현하며 행렬방정식의 해를 구하는 과정을 병렬처리하기 위해 직접솔버와 축차솔버를 병렬화하는 알고리듬을 적용한다. 축차솔버에서 병렬처리에 효과적인 새로운 선결조건행렬을 제안하고 이를 이용한 알고리듬의 효율성을 논의하였다. 각 알고리듬을 이용한 분산처리의 효율성을 제시하고, 각 알고리듬에 대해 해석에 소요된 계산시간을 비교 검토하여 유한요소법의 병렬처리시 알고리듬 선택에 대한 정보를 제공하려 한다.

2. 이 론

유한요소해석시 구조물의 강성행렬과 하중벡터의 계산 그리고 이것에 의해 구성된 선형 방정식의 해를 구하는 단계에서 계산시간의 대부분이 소모되므로, 효과적인 병렬처리를 위해서는 이 부분의 적절한 병렬화가 중요하다. 강성행렬과 하중벡터의 계산은 요소 강성행렬과 요소 하중벡터의 결합(assembly)으로 이루어지며, 이 계산은 각 요소별로 독립적으로 수행된다. 따라서 영역분할법의 개념을 이용하면 강성행렬과 하중벡터

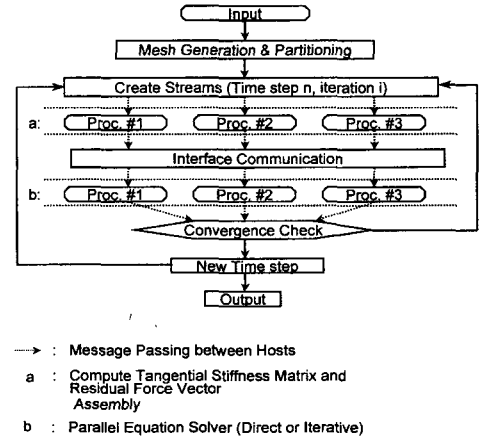


Fig. 1 Flow diagram of parallel algorithm

계산의 병렬화는 용이하게 이루어진다. 선형 방정식의 해를 구하는 부분의 병렬화로는 강성행렬과 하중벡터를 분할해서 병렬화된 직접솔버 또는 축차솔버를 사용하는 것이 가능하다. Fig. 1은 이에 대한 간단한 순서도를 보여준다.

2.1 형상 압연공정 유한요소해석에서 영역분할법의 적용

강점소성 모델를 가정하여 형상 압연공정에서의 소재의 변형을 유한요소해석할 때, 평형 방정식과 유동법칙을 사용한 구성방정식 그리고 적합 방정식을 전영역에 적용하고 경계조건을 부가하기 위해 다음과 같은 가상일의 원리(principle of virtual work)를 이용한다.

$$\delta\pi = \int_V \bar{\sigma}\,\delta\bar{\varepsilon}\,dV - \int_{S_F} F_i\,\delta v_i\,dS = 0 \qquad (1)$$

위 식에서 $\bar{\sigma}$ 는 유효응력으로 $\bar{\varepsilon}$ 와 $\dot{\bar{\varepsilon}}$ 의 함수이며, $\bar{\varepsilon}$ 는 유효변형률을 나타낸다. F_i 와 v_i 는 각각 응력이 부가된 경계에서의 응력벡터의 성분과 속도벡터의 성분의 증분을 의미한다. 형상 압연공정에서는 롤과 소재의 접촉면에서의 마찰력에 의한 영향을 나타낸다. 그리고 소성변형에서의 비압축성 조건을 부가하기 위해 벌칙함수(penalty function)를 도입하면, 식 (1)은 아래와 같이 표현된다.

$$\delta\pi = \int_V \bar{\sigma}\,\delta\bar{\varepsilon}\,dV + K\int_V \dot{\varepsilon}_V\,\delta\varepsilon_V\,dV - \int_{S_F} F_i\,\delta v_i\,dS = 0 \qquad (2)$$

여기서 K 는 벌칙상수이며, ε_V 는 체적 변형률이다.

$$F_s = mk\boldsymbol{l} = mk\left(\frac{2}{\pi}\tan^{-1}\left(\frac{|u_s|}{u_o}\right)\right)\boldsymbol{l} \qquad (3)$$

롤과 소재의 접촉면에서 위와 같이 표현되는 쿨롱(Coulomb) 마찰력을 적용하면, 식 (2)는 비선형 문제가 된다. 위 식에서 m 은 마찰상수, k 는 항복 전단응력 그리고 u_s 는 롤과 소재의 상대속도를 의미한다.

이 비선형 문제의 해를 구하기 위해 뉴톤-랩슨(Newton-Raphson) 축차를 이용하고, 유한요소 수식화를 하면 다음과 같이 각 축차에서 속도벡터 증분에 대한 선형 방정식을 얻게 된다.

$$[K]\{\triangle v\} = \{f\} \qquad (4)$$

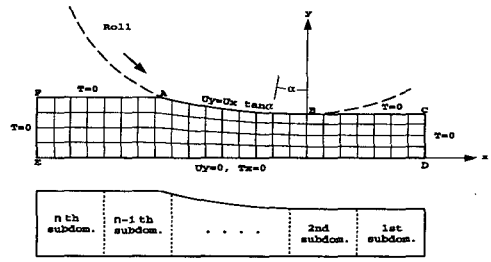


Fig. 2 Domain Decomposition of rolling process (side view of 3-D rolling process)

위 식에서 $[K]$ 는 접선 강성행렬이며, $\{f\}$ 는 잔여력(residual force) 벡터이다. $[K]$ 와 $\{f\}$ 의 계산은 요소 강성행렬과 요소 잔여력벡터의 결합으로 이루어지며, 이 계산은 각 요소에서 독립적이므로 병렬처리를 위해 영역분할법을 적용한다.

먼저 하나의 전체 영역을 n 개의 작은 부영역으로 나누었을 때(Fig. 2), 부영역 $S^{(j)}$ 에서의 접선 강성행렬과 잔여력벡터를 내부자유도(internal degrees of freedom)와 공유 경계상의 자유도로 분리된 형태를 표현하면 다음과 같다.

$$\begin{bmatrix} K_{ss}^{(j)} & K_{sb}^{(j)} \\ K_{bs}^{(j)} & K_{bb}^{(j)} \end{bmatrix}\begin{Bmatrix} \triangle v_s^{(j)} \\ \triangle v_b^{(j)} \end{Bmatrix} = \begin{Bmatrix} f_s^{(j)} \\ f_b^{(j)} \end{Bmatrix} \qquad j=1,2,\cdots,n \qquad (5)$$

여기서 첨자 s 와 b 는 각각 부영역의 내부와 공유경계를 의미한다.

$f_b^{(j)}$ 는 $S^{(j)}$ 에서 독립적으로 계산된 공유경계에 대한 잔여력벡터로 다음과 같이 외부에서 이 경계에 작용하는 외력과 내력으로 이루어져있다.

$$f_b^{(j)} = f_{b_e}^{(j)} + f_{b_i}^{(j)} \qquad (6)$$

따라서 식 (5)의 선형방정식을 다음과 같이 풀어쓸 수 있다.

$$f_s^{(j)} = K_{ss}^{(j)}\triangle v_s^{(j)} + K_{sb}^{(j)}\triangle v_b^{(j)} \qquad (7)$$

$$f_{b_e}^{(j)} + f_{b_i}^{(j)} = K_{bs}^{(j)}\triangle v_s^{(j)} + K_{bb}^{(j)}\triangle v_b^{(j)} \qquad (8)$$

공유경계에서의 변위의 적합조건은 아래와 같다.

$$\triangle v_b^{(j)} = \triangle v_b \qquad j=1,2,\cdots,n \qquad (9)$$

식 (7)은 $\triangle v_s^{(j)}$ 에 대해서 다음처럼 정리될 수 있다.

$$\triangle v_s^{(j)} = -K_{ss}^{(j)-1}(K_{sb}^{(j)}\triangle v_b^{(j)} - f_s^{(j)}) \qquad (10)$$

이 식을 식 (8)에 대입하여 정리하면 다음과 같이 $S^{(j)}$ 의 공유경계의 자유도로 강성행렬과 하중벡

터를 다음과 같이 축약할수 있다.

$$\overline{K}_{bb}^{(j)} \triangle v_b^{(j)} = \overline{f}_b^{(j)} \tag{11a}$$

여기서 $\overline{K}_{bb}^{(j)}$ 와 $\overline{f}_b^{(j)}$ 는 공유경계에서의 축약된 형태의 접선 강성행렬과 잔여력벡터로서 다음과 같은 식으로 구해진다.

$$\overline{K}_{bb}^{(j)} = K_{bb}^{(j)} - K_{bs}^{(j)} K_{ss}^{(j)-1} K_{sb}^{(j)} \tag{11b}$$

$$\overline{f}_b^{(j)} = f_{be}^{(j)} + F_{bi}^{(j)} - K_{bs}^{(j)} K_{ss}^{(j)-1} f_s^{(j)} \tag{11c}$$

각 부영역의 축약된 형태의 강성행렬과 하중벡터를 결합하면 아래의 식처럼 전체 영역의 공유경계 자유도 만의 선형방정식, 즉 축약된 선형대수방정식(reduced system or Schur complement)을 구할 수 있다.

$$[K_B]\{\triangle v_B\} = \{f_B\} \tag{12a}$$

$$K_B = \sum_{j=1}^{n} \overline{K}_{bb}^{(j)} \tag{12b}$$

$$F_B = \sum_{j=1}^{n} \overline{F}_b^{(j)} \tag{12c}$$

Fig. 4(a)는 형상 압연공정 유한요소해석에서 영역분할법을 적용해 접선 강성행렬과 잔여력벡터를 구할 때, 각 행렬과 벡터의 형태를 보여준다.

2.2 병렬 직접솔버(Parallel Direct Solver)의 적용

유한요소해석에서의 선형 방정식의 해를 구하기 위해 직접솔버를 병렬화하는 알고리듬을 생각할 수 있다. 이런 알고리듬은 여러가지가 제안되어 있으며 이중 분산 병렬처리에 적합한 ScaLapack 라이브러리[26]를 이용하였다. ScaLapack 라이브러리는 여러 종류의 행렬에 대해 행렬방정식, 고유치 문제 그리고 최소 제곱화(least square method)의 계산을 분산 병렬처리할 수 있는 라이브러리 루틴(routine)을 제공한다. 유한요소법에서 구해지는 강성행렬은 일반적으로 대칭(symmetry)이고 양정(positive definite)이며, 중심에서 밴드폭(bandwidth)을 가지게 된다. 특히 형상 압연의 경우는 재료의 형상이 길쭉한 모양이므로 유한요소 모델링을 하면 접선 강성행렬이 전체 자유도에 비해 매우 작은 밴드폭을 갖게 된다. ScaLapack 라이브러리에서 이와 같은 선형 방정식 시스템에 적합한 루틴은 pspbsv로 알고리듬[27]은 다음과 같다.

$$A x = b \tag{13}$$

여기서 A 는 매 축차과정에서의 접선 강성행렬로, 일정한 밴드폭은 β 를 갖는다. 그리고 b 와 x 는 각 축차에서의 잔여력 벡터와, 속도증분 벡터에 해당한다.

주어진 행렬 방정식의 해를 촐레스키 분해(Cholesky decomposition) 알고리듬을 이용해 구하는 과정을 병렬화하기 위해, 식 (13)을 다음과 같이 변형한다.

$$(P A P^T) P x = P b \tag{14a}$$

$$A' = P A P^T \tag{14b}$$

$$x' = P x \tag{14c}$$

$$b' = P b \tag{14d}$$

위 식에서 P 는 순열행렬(permutation matrix)로서, A 의 자유도 순서를 바꾸어 주는 역할을 하므로 A' 도 대칭이며 양정 행렬이다. 여기에 촐레스키 분해하면 아래와 같이 표현된다.

$$A' = P A P^T = L L^T \tag{15}$$

따라서 식 (14)와 식 (15)에 의해 다음과 같은 행렬 방정식이 얻어진다.

$$L L^T x' = b' \tag{16}$$

위 식에서 L 이 삼각행렬이므로 다음과 같은 방법으로 해가 구해진다.

$$L z = b' \tag{17a}$$

$$U x' = z \tag{17b}$$

$$x = P^T x' \tag{17c}$$

이 알고리듬은 촐레스키 분해 과정을 병렬처리할 때 프로세서 사이의 자료전달 양과 횟수를 줄이기 위해, 행렬의 자유도 순서를 바꾼 후, 행렬을 분해하여 각 프로세서에 할당하고 촐레스키 분할을 수행한다. 그러나 순차적인 촐레스키 분해 과정을 그대로 병렬화하는 것이 아니므로 계산량은 순차적인 것보다 많아진다.

Fig. 3(a)는 작은 밴드 폭을 갖는 행렬 A 의 형태를 보여준다. 여기서 A 는 굵은 실선으로 표시된 만큼 분할되어, 각 프로세서에 할당된다. 그림에서 소첨자 i 는 i 번째 프로세서를 의미한다. 각 부 행렬은 다시 네 개의 블록행렬 형태로 분해한다. A_i 의 크기는 $\beta \times O_i$ 이며, B_i , C_i 그리고 D_i 의 크기는 모두 $\beta \times \beta$ 이다.

병렬화를 위해 행렬 A 의 자유도 순서를 다음과 같이 바꾼다.

① A_i 의 순서대로 자유도 순서를 바꾼다.

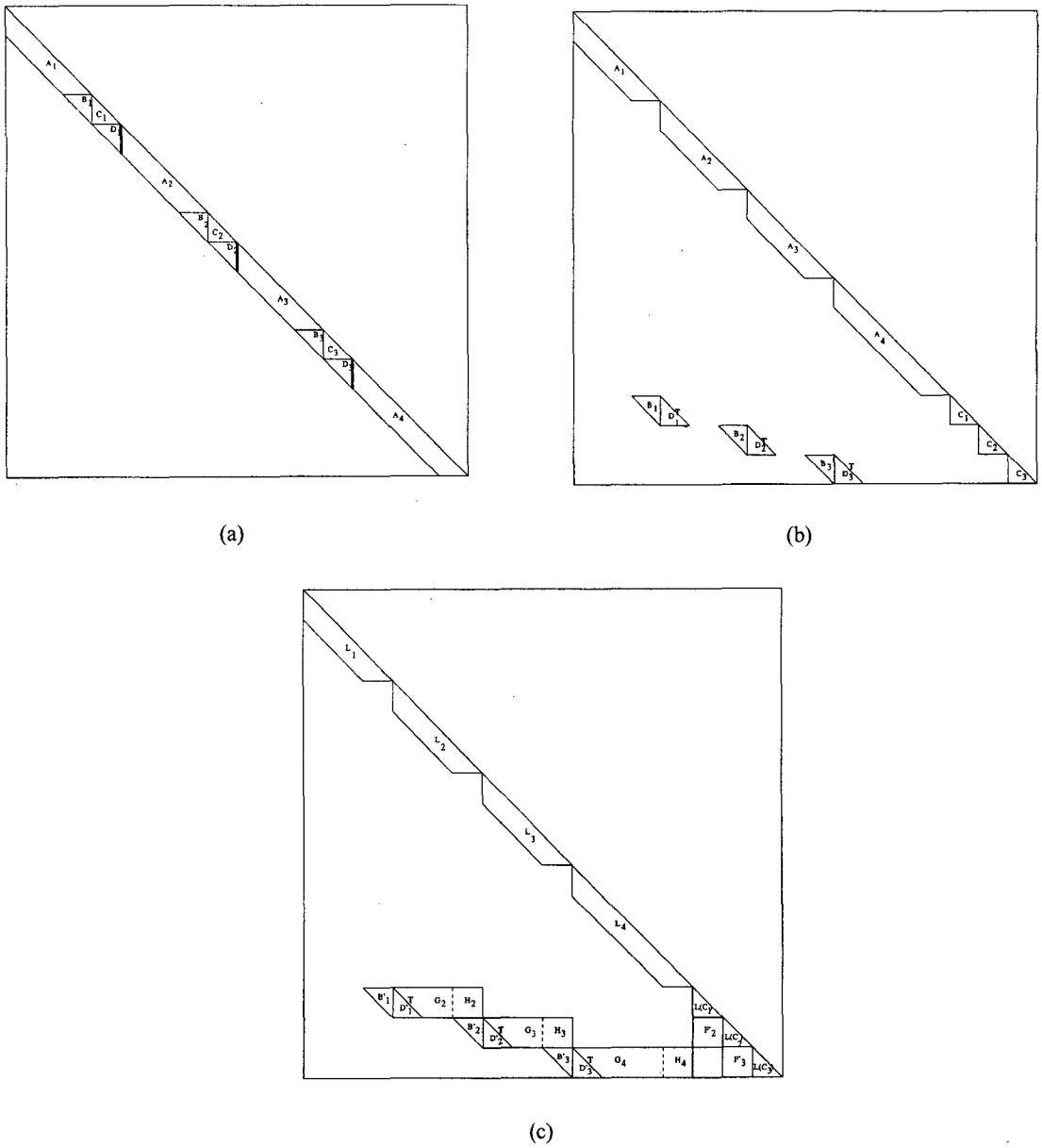


Fig. 3 The shape change of stiffness matrix in parallel direct solver.

② A_{nprocs} 다음에 C_i 의 순서대로 자유도 순서를 정한다.

Fig. 3(b)는 이 결과를 보여준다. 이 과정은 순열행렬 P 를 행렬 A 에 적용해 A' 을 구하는 것에 해당한다.

이제 행렬 A' 의 촐레스키 분해는 각 프로세서에서의 블록행렬들의 계산에 의해 이루어지며, 이 과정 중 프로세서 사이의 자료 교환은 아주 작은 양만을 필요로 한다. Fig. 3은 이 결과로 구성된 행렬을 보여준다.

이 알고리듬의 전 계산과정은 다음과 같이 세 부분으로 요약된다.

① 행렬의 경계 자유도에 대한 축약된 행렬 방정식(Schur complement 또는 reduced

system)의 구성

② 축약된 행렬 방정식의 해 계산

③ 각 부 행렬의 소거된 내부 자유도에 대한 해 계산

과정 ①에서 각 프로세서들은 다른 프로세서와 독립적으로 내부 블록 행렬들의 계산을 수행하며, 이 결과를 결합하여 축약된 행렬 방정식을 구성한다. 우선 각 프로세서 i 는 D_i 를 $i+1$ 번째 프로세서에 전달하고, $i-1$ 번째 프로세서로부터 D_{i-1} 을 전달받는다. 그리고 내부 블록 행렬들을 이용해 다음의 계산을 순차적으로 수행한다.

$$(\mathrm{i})\ A_i = L_i L_i^T \qquad (\mathrm{ii})\ L_i B_i'^T = B_i^T$$

$$(\mathrm{iii})\ C_i' = C_i - B_i' B_i^T \quad (\mathrm{iv})\ L_i G_i^T = D_i$$

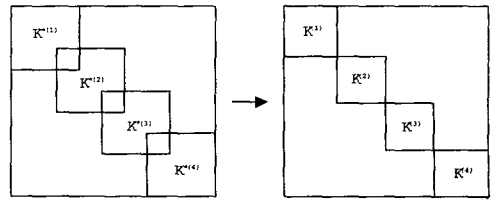
$$(\mathrm{v})\ E_i = G_i G_i^T \qquad (\mathrm{vi})\ F_i^T = H_i B_i'^T$$

Fig. 3(c)는 이 과정에서 구성된 블록 행렬들의 형태를 보여준다.

과정 ②에서는 각 프로세서가 소유하고 있는 블록 행렬들의 결합하여, 행렬의 내부 자유도는 소거되고 분해된 행렬 사이의 경계 자유도만으로 구성된 축약된 형태의 행렬 방정식을 구성하고, 이 경계의 해를 구한다. C_i 를 E_{i+1} 에 더하면, 축약 행렬 방정식의 대각 블록(diagonal block)이 되며, F_i 는 대각외 블록(off-diagonal block)을 구성한다.

과정 ③에서는 이 경계 해를 이용해 소거되었던 행렬의 내부 자유도에 대한 해를 복구한다. 그리고 이 과정은 각 프로세서에서 독립적으로 수행된다.

자유도 수가 N 이고 밴드 폭이 β 인 행렬에 순차적 촐레스키 분해를 수행할 때의 계산량은 약 $N \times \beta^2$ 이다. 그러나 알고리듬에서의 계산량은 이보다 상당히 증가하게 된다. 과정 ①에서의 모든 프로세서의 계산량을 살펴보면 다음과 같다. A_i 를 촐레스키 분해할 때 $N\beta^2 + O(N\beta)$ 의 계산량이 소모되며, G_i 를 구성하기 위해서 $2N\beta^2 + O(N\beta)$ 의 계산량이 필요하다. 그리고 G_i 를 이용해 C_i 를 수정할 때 수행되는 계산량은 $N\beta^2 + O(N\beta)$ 이다. 따라서 전 과정에서 $4N\beta^2 + O(N\beta)$ 의 계산량이 필요하게 되며, 이것은 순차적 촐레스키 분해의 경우보다 4



(a) (b)

Fig. 4 The Shape of stiffness matrix in domain decomposition method :
(a) after subdomain calculation
(b) after interface communication

배가 많은 양이다. 따라서 4개 이상의 프로세서를 사용해야 병렬처리 효과를 얻을 수 있게 된다. 일반적으로 병렬처리 알고리듬은 사용하는 프로세서의 수가 증가할수록, 필요한 프로세서 사이의 자료교환의 양도 증가하며, 이것은 많은 수의 프로세서를 사용할 때의 병렬처리 효율을 상당히 감소시킨다. 그러나 이 알고리듬은 사용하는 프로세서가 증가해도 프로세서 사이의 자료교환의 양이나 횟수는 거의 일정하다. 따라서 많은 프로세서를 사용하면 오히려 큰 효율을 얻을 수도 있다.

이 알고리듬은 ScaLapack 라이브러리에서 pspbsv라는 서브루틴(subroutine)에 구현되어 있으며, 형상 압연의 유한요소해석 프로그램 중 Newton-Raphson 법에 의해 선형화된 방정식의 행렬 방정식의 해를 구할 때 사용한다. 형상 압연 공정에 사용되는 소재는 일반적으로 압연 방향으로 긴 형상을 하기 때문에 영역분할법 개념을 이용한 유한요소해석 병렬처리를 할 때, 각 프로세서에 할당된 부영역에 대한 강성행렬과 잔여력 벡터를 구하고 나면, 각 프로세서는 Fig. 3(a)와 비슷한 형태로 분해된 접선 강성행렬 그리고 잔여력 벡터의 성분을 가지고 있게 된다. 그러나 부영역 사이의 공유 경계 자유도에 대해서는 아직 결합이 되어 있지 않은 상태이다. 각 프로세서에 일정한 계산량을 할당하기 위해 전체 영역을 같은 크기의 부영역들로 분할하면, 이 공유경계 자유도 안에서 Fig. 3(a)의 행렬 분할이 이루어진다. 따라서 이 부분의 접선 강성행렬과 잔여력 벡터를 관련된 프로세서들과 교환하고 결합하면(Fig. 4(a)), 각 프로세서는 Fig. 3(a)와 같이 분할된 행렬과 잔여력 벡터 성분을 갖게 된다. 여

기서 각 프로세서 i 는 $i+1$ 번째 프로세서와 $i-1$ 번째 프로세서하고만 자료교환을 하게 된다. 따라서 자료교환 양이 적고, 자료교환이 각 부분에서 독립적으로 이루어진다고 볼 수 있으므로, 병렬처리 효율을 감소시키지 않는다.

2.3 병렬 CG(Conjugate Gradient) 알고리듬

구조물의 유한요소해석시 일반적으로 축차솔버보다 직접솔버의 사용이 선호된다. 축차솔버는 계산시간의 예측이 불가능하고, 선형 방정식에서 강성행렬의 성질에 따라 해의 수렴이 좌우되므로, 경우에 따라 해의 수렴이 불안정할 수 있다. 그러나 큰 구조물의 정밀한 해석에서는 축차솔버가 오히려 경쟁력을 가질 수 있으며, 병렬처리에 이용될 경우 큰 효과를 가져올 수 있다. 여러 가지 축차솔버의 사용이 가능하지만, 유한요소해석에서는 주로 CG 알고리듬[28,29]이 가장 효율적이라고 알려져 있으며, 병렬화에도 많은 장점을 가지고 있다.

형상 압연공정의 유한요소해석에 영역분할법을 적용해 병렬처리할 때, 각 부영역은 다음과 같은 식을 만족시킨다.

$$\{f^{(j)}\} = [K^{(j)}]\{\triangle v^{(j)}\} \tag{18}$$

여기서 $[K^{(j)}]$, $\{f^{(j)}\}$ 그리고 $\{\triangle v^{(j)}\}$ 는 각각 부영역 $S^{(j)}$ 에서의 접선 강성행렬, 잔여력벡터 그리고 속도 증분 벡터를 나타낸다. 전체 속도 증분 벡터 $\{\triangle v\}$ 와 $\{\triangle v^{(j)}\}$ 의 관계 그리고 전체 잔여력벡터 $\{f\}$ 와 $\{f^{(j)}\}$ 의 관계는 다음과 같이 표현된다.

$$\{\triangle v^{(j)}\} = [A^{(j)}]\{\triangle v\} \tag{19a}$$

$$\{f\} = [A^{(j)}]^T\{f^{(j)}\} \tag{19b}$$

여기서 $[A^{(j)}]$ 는 전체 자유도에서 부영역 $S^{(j)}$ 의 자유도만으로 제한하는 행렬이며, $[A^{(j)}]^T$ 는 반대의 역할을 한다.

식 (18)과 식 (19)를 결합하면 다음과 같이 전영역의 선형 방정식을 구할수 있다.

$$\begin{aligned}\{f\} &= \sum_{j=1}^{n}[A^{(j)}]^T[K^{(j)}][A^{(j)}]\{\triangle v\} \\ &= [K]\{\triangle v\}\end{aligned} \tag{20}$$

그리고 결합을 한 후의 각 부영역의 벡터는 $\{s^{(j)}\} = [A^{(j)}]^T[A^{(j)}]\{q^{(j)}\}$ 는 다음과 같이 구해진다.

$$\{s^{(j)}\} = \sum_{i\in adj(S^{(j)})}\{q^{(i)}\} + \{q^{(j)}\} \tag{21}$$

여기서 $adj(S^{(j)})$ 는 부영역 $S^{(j)}$ 와 공유경계를 갖고 있는 다른 부영역들을 의미한다. 이부분을 분산 병렬처리 환경에서 구현하려면 다음과 같이 자료교환이 필요하다.

① $\{q^{(j)}\}$ 를 $adj(S^{(j)})$ 로 전달

② $\{q^{(j)}\}$ 를 $adj(S^{(j)})$ 로부터 전달받음

③ $\{s^{(j)}\} = \sum\{q^{(i)}\} + \{q^{(j)}\}$

이 부분의 자료교환을 효율적으로 구현하면, 벡터들의 내적과 행렬과 벡터의 곱만으로 구성되는 CG 알고리듬의 병렬화가 용이하게 이루어질 수 있다. CG 알고리듬의 중요 부분을 간단히 기술하면 아래와 같다.

(i) $\{u\} = [K]\{p\}$

(ii) $\alpha = \{r\}^T\{r\}/\{p\}^T\{p\}$

(iii) $\{\triangle v_{new}\} = \{\triangle v\} + \alpha\{p\}$

(iv) $\{r_{new}\} = \{r\} - \alpha\{u\}$

(v) $\lambda = \{r_{new}\}^T\{r_{new}\}/\{r\}^T\{r\}$

(vi) $\{p_{new}\} = \{r_{new}\} + \lambda\{p\}$

이 과정의 병렬화를 살펴보면 다음과 같다. 먼저 행렬과 벡터의 곱의 계산인 과정 (i)은 다음과 같이 표현할 수 있다.

$$\begin{aligned}\{u\} &= \sum_{j=1}^{n}[A^{(j)}]^T[K^{(j)}][A^{(j)}]\{p\} \\ &= \sum_{j=1}^{n}[A^{(j)}]^T[K^{(j)}]\{p^{(j)}\}\end{aligned}$$

$$\{u\} = \sum_{j=1}^{n}[A^{(j)}]^T\{u^{(j)}\}$$

따라서 이 과정은 각 부영역을 담당하는 프로세서에서 독립적으로 수행될 수 있다.

벡터와 벡터의 내적들로 이루어진 과정 (ii)는 다음과 같은 수식화를 통해 병렬처리가 가능해진다.

$$\{p^{(j)}\} = [A^{(j)}]\{p\}\ ,\quad \{\triangle v^{(j)}\} = [A^{(j)}]\{\triangle v\}$$

$$[A^{(j)}]^T\{f^{(j)}\} = \{f\}\ ,\quad [A^{(j)}]^T\{r^{(j)}\} = \{r\}$$

$$\begin{aligned}\gamma &= \{r\}^T\{r\} = \sum_{j=1}^{n}\left\{r^{(j)}\right\}^T[A^{(j)}][A^{(j)}]^Tr^{(j)}\Big\} \\ &= \sum_{j=1}^{n}\{r^{(j)}\}^T\{s^{(j)}\} = \sum_{j=1}^{n}\rho^{(j)}\end{aligned}$$

그리고 $\{p\}^T\{u\}$ 의 이와 비슷하게 병렬처리될 수 있다.

$$\beta = \{p\}^T[K]\{p\} = \sum_{j=1}^{n}\{r^{(j)}\}^T[A^{(j)}][K^{(j)}][A^{(j)}]^T\{r^{(j)}\} = \sum_{j=1}^{n}\{p^{(j)}\}^T\{u^{(j)}\} = \sum_{j=1}^{n}\beta^{(j)}$$

따라서 α 는 다음과 같이 계산된다.

$$\frac{1}{\alpha} = \frac{\sum_{j=1}^{n}\beta^{(j)}}{\gamma} = \sum_{j=1}^{n}\sigma^{(j)}$$

벡터와 실수의 곱의 계산인 과정 (iii)과 (iv)는 간단히 각 부영역이 할당된 프로세서에서 독립적으로 이루어진다.

$$\{\triangle v_{new}^{(j)}\} = \{\triangle v^{(j)}\} + \alpha\{p^{(j)}\}$$

$$\{r_{new}^{(j)}\} = \{r^{(j)}\} - \alpha\{u^{(j)}\}$$

그리고 과정 (v)는 과정 (ii)와 같은 방법으로 계산되며, 과정 (vi)은 아래와 같이 병렬화 된다.

$$\{p_{new}^{(j)}\} = \{r_{new}^{(j)}\} + \lambda\{p^{(j)}\}$$

이상의 병렬 CG 알고리듬을 요약하면 다음과 같다.

(i) $\{\triangle v^{(j)}\} = \{0\}$, $\{r^{(j)}\} = \{f^{(j)}\}$
과정 (v)로 감

(ii) $\{u^{(j)}\} = [K^{(j)}]\{p^{(j)}\}$, $\beta^{(j)} = \{p^{(j)}\}\{u^{(j)}\}$
$\sigma^{(j)} = \beta^{(j)} / \gamma$

(iii) $1 / \alpha = \sum_{j=1}^{n}\sigma^{(j)}$

(iv) $\{\triangle v^{(j)}\} = \{\triangle v^{(j)}\} + \alpha\{p^{(j)}\}$
$\{r^{(j)}\} = \{r^{(j)}\} - \alpha\{u^{(j)}\}$

(v) $\{r^{(j)}\}$ 를 $adj(S^j)$ 에 전달,
$\{r^{(j)}\}$ 를 $adj(S^j)$ 로부터 전달받음

(vi) $\{s^{(j)}\} = \sum\{r^{(j)}\} + \{r^{(j)}\}$
$\rho^{(j)} = \{r^{(j)}\}^T\{s^{(j)}\}$

(vii) $\lambda_{new} = \sum_{j=1}^{n}\rho^{(j)}$

(viii) $\gamma_{new} / \gamma_o < tolerance$ 이면 수렴
$\{p^{(j)}\} = \{s^{(j)}\} + (\gamma_{new}/\gamma)\{p^{(j)}\}$
$\gamma = \gamma_{new}$

(ix) 과정 (ii)로 감

2.4 병렬 PCG(Preconditioned Conjugate Gradient) 알고리듬

일반적으로 CG 알고리듬은 수렴속도가 느리기 때문에, 수렴속도를 가속화시키기 위해 선결조건(preconditioner) 행렬을 이용해 강성행렬의 조건수(condition number)를 줄이는 방법을 사용한다. 이를 위해 ILU(Incomplete LU decomposition), Jacobi, SOR 등 여러 가지 방법이 제안되었다. 대체로 ILU를 이용한 방법이 가장 효과적이지만, 효율적인 병렬처리가 용이하지 못한 단점을 가지고 있다. 본 연구에서는 Jacobi 방법과 이를 변형한 블록 역행렬 방법의 병렬처리 알고리듬을 병렬 CG 알고리듬에 적용한다.

PCG 알고리듬을 간단히 기술하면 다음과 같다. 아래에서 $[K_p]$ 는 선결조건 행렬을 나타낸다.

(i) $\{z\} = [K_p]^{-1}\{r\}$, $\{p\} = \{z\}$

(ii) $\alpha = \{z\}^T\{r\} / \{p\}^T[K]\{p\}$

(iii) $\{\triangle v_{new}\} = \{\triangle v\} + \alpha\{p\}$

(iv) $\{r_{new}\} = \{r\} - \alpha[K]\{u\}$

(v) $\{z_{new}\} = [K_p]^{-1}\{r_{new}\}$

(vi) $\lambda = \{z_{new}\}^T\{r_{new}\} / \{z\}^T\{r\}$

(vii) $\{p_{new}\} = \{z_{new}\} + \lambda\{p\}$

위 식에서 알수 있듯이 대부분이 벡터들의 내적과 행렬과 벡터의 곱으로 구성되는 점은 CG 알고리듬과 같으므로 이부분은 병렬 CG 알고리듬과 같은 방법으로 병렬처리할 수 있다. 그러나 PCG 알고리듬에서는 $[K_p]^{-1}\{r\}$ 의 계산이 추가되므로 이 부분의 병렬처리가 필요하다.

2.4.1 Jacobi 방법

PCG 알고리듬 중에서 가장 쉽게 구현할 수 있는 방법은, 접선 강성행렬의 대각성분(diagonal component)만으로 $[K_p]$ 를 대각행렬로 구성하는 것이다. 이 알고리듬은 일반적으로 다른 PCG 알고리듬에 비해 수렴속도가 떨어지지만, 접선 강성행렬의 대각성분이 다른 성분에 비해 아주 큰 값을 나타내면 상당히 효과적이다.

Jacobi 방법을 순차적 알고리듬에서 구현하기 위해서는, 접선 강성행렬에서 대각성분을 뽑아내어, 각 성분의 역수를 구하여 하나의 1차원 벡터 배열 $\{k_p^{-1}\}$ 에 저장한다. 이 배열은 대각행렬 $[K_p]^{-1}$ 의 대각성분을 보관하게 된다. 따라서 $[K_p]^{-1}\{r\}$ 의 계산은 $\{k_p^{-1}\}^T\{r\}$ 이 되기 때문에, 결국 두 벡터의 내적을 계산하는 것과 동일하다. 두 벡터의 내적의 계산은 병렬 CG 알고리듬에서

소개한 바와 같이 병렬처리가 용이하다.

이 부분의 병렬처리를 위해서는 먼저 각 부영역의 성분으로 제한된 $\{k_p^{(j)-1}\}$ 를 구해야 한다.

$$\{k_p^{(j)-1}\} = [A^{(j)}]\{k_p^{-1}\}$$

영역분할법을 적용해 각 부영역 $S^{(j)}$ 의 접선 강성행렬을 계산하고, 이 행렬의 대각성분을 1차원 벡터 배열 $\{q^{(j)}\}$ 에 저장한다. 여기서 $\{q^{(j)}\}$ 의 성분중 다른 부영역들과의 공유자유도 성분은 결합이 되어있지 않은 상태이므로, $\{k_p^{(j)-1}\}$ 를 구하기 위해서는 다음과 같이 공유경계를 가진 다른 부영역들과의 자료교환이 필요하다.

$$1 / \{k_p^{(j)-1}\} = [A^{(j)}][A^{(j)}]^T\{q^{(j)}\} = \sum_{i \in adj(S^{(j)})} \{q^{(i)}\} + \{q^{(j)}\}$$

이와 같이 $\{k_p^{(j)-1}\}$ 가 구성되면 $[K_p]^{-1}\{r\}$ 의 계산의 수행은 아래처럼 병렬처리될 수 있다.

$$[K_p]^{-1}\{r\} = \{k_p^{-1}\}^T\{r\} = \sum_{j=1}^{n} \{k_p^{(j)-1}\}^T\{r^{(j)}\}$$

2.4.2 블록 역행렬(block inverse matrix) 방법

이 방법은 Jacobi 방법의 수렴속도를 향상시키고 영역분할법을 이용한 병렬처리 알고리듬에 적합하도록 Jacobi 방법을 수정한 것으로 저자들이 제안한 것이다. Jacobi 방법에서는 $[K_p]$ 를 접선 강성행렬의 대각성분만으로 구성하였으나, 블록 역행렬 방법에서는 접선 강성행렬을 블록 대각행렬 (block diagonal matrix) $[K^{(j)}]$ 로 분할한 후, 이들의 역행렬의 결합으로 $[K_p]^{-1}$ 를 구성한다.

영역분할법을 적용해 접선 강성행렬을 구하면, 이 행렬은 Fig. 4와 같이 공유자유도 성분에서만 중복되는 블록 대각행렬 $[K^{*(j)}]$ 의 결합의 형태를 가지게 되며, $[K^{*(j)}]$ 는 부영역 $S^{(j)}$ 를 담당하는 프로세서에 저장되어 있게 된다. 여기서 경계 자유도를 공유하는 다른 부영역을 담당하는 프로세서와의 자료교환을 통해, 접선 강성행렬을 각 프로세서에서 서로 완전히 분리된 블록 대각행렬 $[K^{(j)}]$ 의 결합으로 변형할 수 있으며 각 프로세서는 이 행렬을 보관한다. $[K_p]^{-1}$ 를 이 행렬들의 역행렬의 결합으로 구성하면 다음과 같이 표현할 수 있다.

$$[K_p]^{-1} = \sum_{j=1}^{n} [K_p^{(j)}]^{-1}$$

따라서 $[K_p]^{-1}$ 를 구하는 과정은 각 프로세서에서 독립적으로 수행된다.

이 알고리듬을 구현하는 방법을 살펴보면 다음과 같다. PCG 알고리듬에서 $[K_p]^{-1}$ 를 직접 구하는 것이 필요하지 않고, 단지 $[K_p]^{-1}\{r\}$ 을 계산할 수 있으면 된다. 이 과정은 아래의 식에서 $\{z\}$ 를 구하는 것으로 생각할 수 있다.

$$\{z\} = [K_p]^{-1}\{r\} \quad \Leftrightarrow \quad [K_p]\{z\} = \{r\}$$

$[K_p]^{-1}\{r\}$ 의 계산은 매 축차과정에서 사용되므로, 처음에 $[K_p]$ 를 LU 분할(LU decomposition)하여 저장한다. 그리고 각 축차에서 $[K_p]^{-1}\{r\}$ 의 계산은 이 분할된 행렬을 이용하여 전대입(forward substitution)과 후대입(backsubstitution) 과정으로 수행된다. 보통 LU 분할에 사용되는 계산시간이 대입과정에 비해 월등히 많기 때문에, 선형 방정식 해의 수렴까지의 축차수가 적으면 병렬처리 효과를 얻을 수 있다.

3. 결과 및 검토

본 연구에서는 앞서 기술한 병렬처리 알고리듬을 강점소성 모델을 이용해 형상 압연공정을 유한요소해석할 때 계산시간의 효율성을 확인하기 위해 여러 경우의 예제를 선택하여 해석하였다. 계산에 사용된 장비는 시스템공학연구소 내에 설치되어 있는 Cray사의 T3E(Cray T3E-900 LC 128A-128)로 분산처리가 가능한 MPP(Massively Parallel Processors) 병렬 컴퓨터이다. OS 환경으로 UNICOS/mk 2.0.2.30을 사용하고 있으며 3차원 토러스(3-D Torus Interconnection) 구조로 여러 프로세서를 연결하고 있다. 이 장비는 총 130개의 프로세서로 이루어져 있으며, 이중 24개의 프로세서는 OS 및 시스템 환경 제어 등에 사용되고 있으며 응용분야의 계산에는 112개의 프로세서가 사용 가능하다. 총 메모리는 2GB로 각 프로세서당 128MB가 할당되어 있으며, 최대 계산능력은 115GFLOPS 이다. 본 알고리듬의 구현 중 프로세서간의 자료교환과 동기화를 위해 PVM과 MPI 라이브러리를 이용하였다.

병렬처리의 효율성을 판단하기 위한 척도로 아래와 같이 정의되는 속도증가(Speedup)와 효율(Efficiency)을 사용하였다.

$$Speedup = \frac{T_1}{T_n}$$

$$Efficiency = \frac{Speedup}{n} = \frac{T_1}{(n * T_n)}$$

여기서

T_1 : 1개의 프로세서로 계산했을 때 걸린 시간

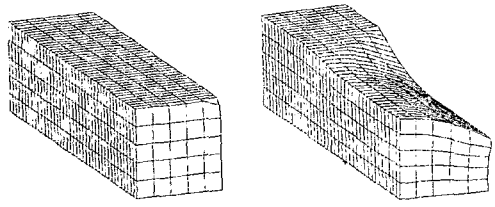
T_n : n개의 프로세서로 계산했을 때 걸린 시간

일반적으로 $0 \le E_f \le 1$ 이 되며 이상적인 경우에 1이 되지만, 실제에 있어서는 프로세서간의 정보교환, 동기화를 위한 대기시간 그리고 알고리듬의 순차적 종속성(sequential dependency) 등으로 인해 그 값이 감소한다.

3차원 형상 압연공정 유한요소해석은 개발된 순차적 프로그램에 병렬 알고리듬을 적용하여 병렬처리가 가능한 프로그램을 개발하였다.

앞에서 기술한 병렬처리 알고리듬을 구현했을 때의 계산시간을 제시하고, 각 알고리듬들을 비교, 검토 하기 위해 형상 압연공정의 한 예제를 선정하여 요소와 절점수를 세 경우로 변화시키면서 유한요소해석을 하였다. 선택된 예제는 정사각형에서 타원형태로 변형되는 경우로 Fig. 5는 해석전과 해석후의 소재의 변형 모습을 보여준다.

이 압연공정의 유한요소해석의 병렬처리를 위해 전체 영역을 2개부터 32개의 부영역으로 분할하고, 한 프로세서에 하나의 부영역만을 할당하여 처리하였다. 앞으로 제시할 병렬처리 계산시간은, 병렬처리 효과는 각 하중단계(loading step or time step)에서 일정하므로, 초기의 몇 하중단계만을 수행한 결과이다.



(a) (b)

Fig. 5 Deformation shape of shape rolling process

선택된 예제들에 대해, 각 알고리듬을 적용했을 때의 계산시간(CPU time)은 Table 1, Table 2, Table 3에 표기하였다. 계산시간은 영역분할법을 적용해 얻은 접선 강성행렬과 잔여력벡터의 계산시간 (Element Time), 선형 대수 방정식의 해를 구하는 시간 (Solution Time) 그리고 총 계산시간 (Total Time)을 수록하였다. 또한 각 경우에 대해 자유도 수와 밴드폭의 크기를 표기하였다. 이 Table의 결과들은 아래에서 도표를 통해 상세히 논의하려고 한다.

Fig. 6은 영역분할법을 이용했을 때의 프로세서 수의 증가에 따른 접선 강성행렬과 잔여력벡터의 계산시간을 보여준다. 이 부분은 프로세서 수의 증가에 그대로 비례하여 속도증가가 이루어짐을 볼 수 있으며 모든 경우에 있어서 같은 결과를 보인다.

3.1 병렬 직접솔버(Parallel Direct Solver)의 적용

Table 1은 각 경우에 대해 접선 강성행렬과 잔여력벡터의 계산은 영역분할법을 이용해 병렬처리하고, 선형 방정식은 병렬 직접솔버인 ScaLapack 라이브러리를 적용했을 때의 계산시간을 보여준다. 이 알고리듬에서는 각 부영역의 자유도 수가 밴드폭의 크기보다 2배 이상이어야 병렬처리가 가능하므로, 부분적으로 계산시간이 표기되지 않은 것은 전 영역을 너무 많은 부영역으로 나누면 병렬처리가 불가능하기 때문이다.

Fig. 7은 이에 대한 속도증가를 보여준다. 예제 3의 경우 자유도 수가 너무 많아 메모리가 부족

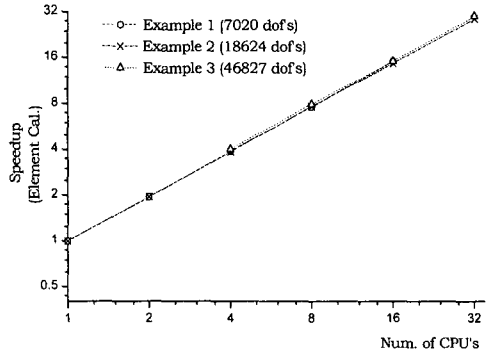


Fig. 6 Relationship of speedup and number of processors in element calculation

Table 1 Computation times of shape rolling analysis with domain decomposition and parallel direct solver

Examples	CPU Time (sec.)	1 CPU	2 CPU	4 CPU	8 CPU	16 CPU	32 CPU
Example 1	Element Time	104.3	53.6	26.9	13.7	6.9	-
	Solution Time	74.5	123.0	63.4	36.5	25.0	-
	Total Time	207.0	186.8	94.7	52.5	33.7	-
number of dof's = 7020				band width = 130			
Example 2	Element Time	184.5	94.8	47.9	24.1	12.6	6.4
	Solution Time	560.9	531.2	270.5	146.9	90.5	67.2
	Total Time	966.3	690.2	338.4	185.6	107.6	77.5
number of dof's = 18624				band width = 220			
Example 3	Element Time	-	-	216.6	109.3	57.2	29.0
	Solution Time	-	-	3636.9	1911.7	1105.8	751.4
	Total Time	-	-	3970.0	2057.9	1179.3	794.0
number of dof's = 46827				band width = 400			

Table 2 Computation times of shape rolling analysis with domain decomposition and parallel preconditioned conjugate gradient solver using Jacobi preconditioner

Examples	CPU Time (sec.)	1 CPU	2 CPU	4 CPU	8 CPU	16 CPU	32 CPU
Example 1	Element Time	104.3	51.4	25.4	12.7	6.5	3.3
	Solution Time	1666.0	806.1	413.1	224.0	132.0	86.7
	Total Time	1799.0	865.9	441.8	238.4	134.0	92.2
Example 2	Element Time	185.9	90.3	45.3	22.7	11.7	5.8
	Solution Time	9785.9	5115.8	2601.0	1325.2	699.4	390.8
	Total Time	10192.9	5265.1	2663.2	1354.0	714.6	399.8
Example 3	Element Time	-	-	204.9	102.8	52.6	26.1
	Solution Time	-	-	16083.8	7553.9	4144.2	2034.7
	Total Time	-	-	16395.5	7685.5	4211.1	2066.1

Table 3 Computation times of shape rolling analysis with domain decomposition and parallel preconditioned conjugate gradient solver using block inverse matrix preconditioner

Examples	CPU Time (sec.)	2 CPU	4 CPU	8 CPU	16 CPU	32 CPU
Example 1	Element Time	52.3	26.5	13.6	6.8	3.6
	Solution Time	120.5	84.8	59.4	46.2	39.9
	Total Time	181.3	116.2	74.5	54.5	45.7
Example 2	Element Time	96.8	48.5	24.1	12.7	6.5
	Solution Time	367.1	258.6	175.7	129.2	109.4
	Total Time	530.4	324.5	205.7	149.2	120.0
Example 3	Element Time	-	225.2	111.6	57.2	29.6
	Solution Time	-	2480.5	1577.5	1177.4	813.0
	Total Time	-	2815.5	1726.6	1244.4	847.8

하여 적은 수의 프로세서로는 계산을 수행할 수 없었다. Fig. 7(a)는 프로세서 수의 증가에 따른 선형방정식의 해를 구하는 부분의 속도증가를 보여준다. 예제 1의 경우는 2개의 프로세서를 사용할 때 속도증가가 반으로 떨어진 후, 다시 프로세서의 수가 증가함에 따라 속도증가도 선형적으로 비례함을 알 수 있다. 이것은 병렬 알고리듬이 순차적인 경우보다 계산량이 4배정도 많기 때문이다. 예제 2와 예제 3은 2개의 프로세서를 사용했을 때 속도증가의 감소를 거의 보여주지 않는다. 이는 전체 자유도 수와 밴드폭의 비가 앞의 경우보다 작기 때문이며, 이런 경우 이 알고리듬의 효율은 상당히 크다. Fig. 7(b)는 총 계산시간의 속도증가를 보여준다. 대체로 프로세서가 증가할수록 속도증가도 커짐을 알 수 있다. 적은 수의 프로세서를 사용할 때 병렬처리의 효과가 현저하지 않은 것은, 선형 방정식의 해를 구하는 부분에서 순차적 알고리듬보다 많은 계산량을 필요로 하기 때문이다. 프로세서의 수가 증가하면서 효율은 점차 감소함을 볼 수 있다.

직접솔버의 방법을 병렬화한 경우 많은 프로세서를 사용할 때 많은 계산시간의 감소를 얻기 위해 순차적인 알고리듬보다 계산량이 많은 방법을 택하였다. 따라서 적은 수의 프로세서의 사용에서는 병렬처리 효과가 별로 관찰되지 않으나, 프로세서의 수가 많아질수록 큰 효과를 볼 수 있다. 그리고 이 알고리듬은 문제의 자유도 수에 비해 밴드폭이 작을수록 효율이 커진다. 예제 2와 예제 3의 경우에 큰 속도증가를 얻을 수 있었던 것은 이 조건을 만족시키기 때문이다. 3차원 형상 압연 문제의 경우 소재의 형상이 압연 방향으로 길기 때문에, 각 차원에 대한 요소의 길이를 반으로 줄이면 자유도는 8배가 되지만 밴드폭은 4배만 증가한다. 따라서 정밀한 해석을 위해 요소 수를 증가시키게 되는데, 이 경우 병렬처리 효과를 크게 볼 수 있다. 예제 3의 경우 32개의 프로세서를 사용할 때 12.5의 속도증가를 보인다.

3.2 Jacobi 방법을 이용한 병렬 PCG 알고리듬의 적용

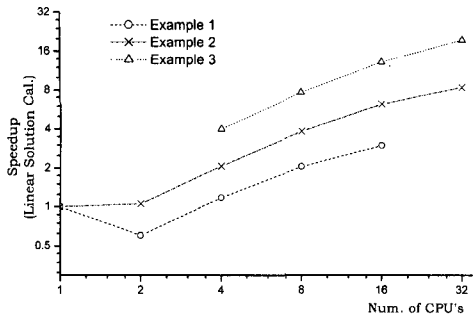
Table 2는 각 경우에 대해 접선 강성행렬과 잔여력벡터의 계산은 영역분할법을 이용해 병렬처리하고, 선형 대수 방정식은 병렬 PCG 알고리듬에 Jacobi 방법을 적용했을 때의 계산시간을 보여준다. 접선 강성행렬과 잔여력벡터를 계산하는 부분은 사용한 프로세서의 수가 증가할수록 그에 비례하여 계산시간이 감소함을 관찰할 수 있다. Fig. 8은 각 계산시간에 대한 속도증가와 효율을 보여준다. Fig. 8(a)는 선형 대수 방정식을 구하는 과정에서 프로세서 수와 속도증가의 관계를 보여준다. 프로세서의 수가 증가함에 따라 그대로 비례하여 속도증가가 이루어짐을 볼수 있다. Fig. 8(b)는 총 계산시간에 대한 속도증가를 보여준다. 접선 강성행렬과 잔여력벡터를 구하는 과정과 선형 방정식의 해를 구하는 과정이 모두 병렬화가 효과적으로 이루어졌기 때문에, 총 계산시간에 대한 속도증가도 사용된 프로세서 수에 그대로 비례함을 볼수 있다. 효율은 프로세서의 수가 증가하면서 대체로 감소하는 추세이지만, 큰 효율의 감소는 보이지 않으며 1 근처의 높은 값을 유지하게 된다.

이와 같이 Jacobi 방법을 이용한 PCG 알고리듬의 구현은 매우 효과적인 병렬처리 결과를 보여준다. 그러나 선형 방정식의 해를 구하기 위한 계산시간이, 해의 수렴속도가 느리므로, 다른 알고리듬에 비해 너무 많이 든다. 따라서 적은 수의 프로세서만을 사용할 때는 다른 알고리듬에 비해 경쟁력이 상당히 떨어진다. 하지만 병렬화의 효과가 크기 때문에 많은 수의 프로세서를 사용할 때는 경쟁력을 가질 수 있다. 이 알고리듬은 문제에 따라 수렴속도가 좌우되므로 적당한 문제에 적용하면 다른 알고리듬보다 효과적일 수 있다.

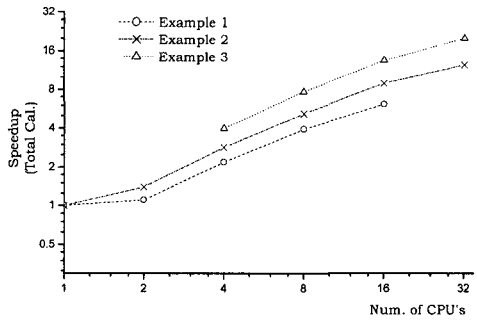
3.3 블록 역행렬 방법을 이용한 병렬 PCG 알고리듬의 적용

Table 3은 각 경우에 대해 접선 강성행렬과 잔여력벡터의 계산은 영역분할법을 이용해 병렬처리하고, 선형 방정식의 해를 구하기 위해 PCG 알고리듬에 블록 역행렬 방법을 적용했을 때의 계산시간을 보여준다. 모든 경우에 대해 접선 강성행렬과 잔여력벡터를 계산하는 부분은, 프로세서의 수가 증가함에 따라 그에 비례하여 계산시간이 감소함을 볼수 있다.

Table 3은 정사각형에서 타원형으로의 압연공정 해석 결과이며, Fig. 9은 이에 대한 속도증가와 효율을 보여준다. Fig. 9(a)는 프로세서 증가에 따른 선형 대수 방정식의 해를 구하는 부분의 속

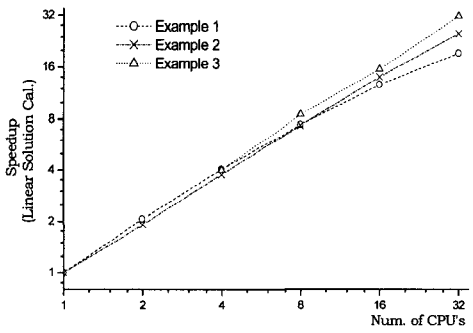


(a) For solving linear equation

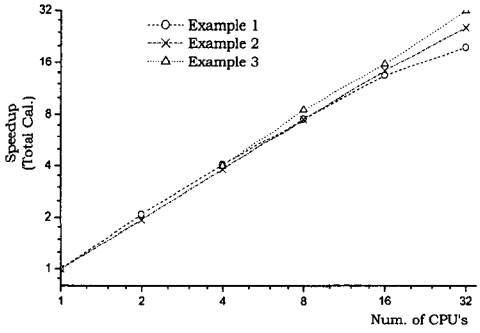


(b) For total calculation

Fig. 7 Relationship of speedup and number of processors for parallel direct solver

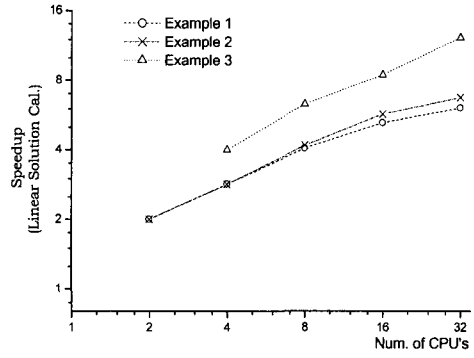


(a) For solving linear equation

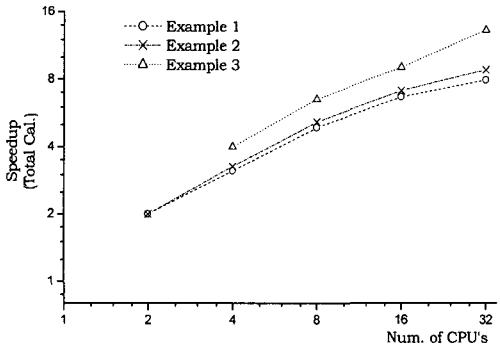


(b) For total calculation

Fig. 8 Relationship of speedup and number of processors for PCG with Jacobi preconditioner



(a) For solving linear equation



(b) For total calculation

Fig. 9 Relationship of speedup and number of processors for PCG with block inverse matrix preconditioner

도증가를 보여준다. 프로세서의 수가 증가할수록 속도증가가 선형적으로 커짐을 볼 수 있으나, 비례하여 증가하지는 않는다. 각 부영역에서 부행렬의 LU 분할 계산시간은 부영역의 수에 비례하여 감소하지만, 각 부영역에 해당하는 블록행렬의 크기가 작을수록 수렴속도는 떨어지므로 더 많은 축차과정을 필요로 하게 된다. 따라서 이 수렴속도의 감소 때문에 프로세서의 수에 비례하는 병렬처리 효과는 얻을 수 없게 된다. Fig. 9(b)는 총 계산시간에 대한 속도증가를 보여준다. 프로세서의 수가 증가함에 따라 속도증가가 선형적으로 이루어지며, 효율은 점차로 감소하고 있다.

4. 결 론

본 논문에서는 3차원 형상 압연공정 유한요소해석의 계산시간을 효과적으로 줄이기 위해 영역분할법과 선형 대수 방정식의 해를 구하는 과정을 분산 병렬처리하는 알고리듬을 적용하여 Cray사의 T3E 컴퓨터 상에서 분산 병렬처리를 수행하고 각 알고리듬의 성능을 비교 검토하였다. 전 영역을 여러 개의 부영역으로 분할해 각 프로세서에 할당하고, 각 프로세서는 독립적으로 담당 부영역에 대한 접선 강성행렬과 잔여력벡터를 구하는 과정을 수행하는 영역분할법을 적용하였다. 이렇게 구해진 선형 방정식의 해는 직접솔버를 병렬화한 알고리듬과 축차솔버를 병렬화한 알고리듬을 구현해 분산 병렬처리가 가능하도록 하였다.

직접솔버의 방법은 촐레스키 분해 알고리듬을 병렬화한 것으로 많은 프로세서를 사용할 때 많은 계산시간의 감소를 얻기 위해 순차적인 알고리듬보다 계산량이 많은 병렬화 방법을 택하였다. 따라서 적은 수의 프로세서의 사용에서는 병렬처리 효과가 별로 관찰되지 않으나, 프로세서의 수가 많아질수록 큰 효과를 보이며, 문제의 자유도 수에 비해 작은 밴드폭을 갖는 문제에서 큰 효율이 얻을 수 있다. 32개의 프로세서를 사용할 때 12.5배의 속도증가를 얻었다.

축차솔버의 병렬화로는 PCG 알고리듬을 병렬처리하였다. 선결조건 행렬로는 Jacobi 행렬과 블록 역행렬을 사용하는 방법을 병렬처리하였다. 일반적으로 문제의 자유도가 클수록 자료교환 양에 비해 계산량의 증가가 크기 때문에 병렬처리시 큰 효과를 얻게 된다. Jacobi 방법의 경우 병렬화의 효율은 1 근처에서 유지될 정도로 크다. 그러나 형상 압연 문제에서는 해의 수렴을 위해 너무 많은 축차를 필요로 하므로 다른 알고리듬에 비해 경쟁력이 떨어진다. 하지만 아주 많은 수의 프로세서를 사용하거나 접선 강성행렬의 조건수가 작게 되는 문제에 적용하면 오히려 경쟁력을 가질수 있다. 블록 역행렬 방법의 경우 적은 프로세서의 사용에서는 가장 병렬처리 효과가 뛰어났다. 하지만 프로세서의 수를 증가시킬수록 각 부행렬의 크기가 작아지므로 더 많은 축차수를 필요로 하게 되어 병렬처리의 효율을 감소시키게 된다. 이론적으로 자유도 수와 거의 비슷한 수의 프로세서를 사용하게 되면 Jacobi 방법과 일치하게 되어 있다. 그러나 문제의 크기가 아주 큰 경우에는 상당한 병렬처리 효과를 기대할 수 있다.

본 논문에서의 알고리듬은 병렬컴퓨터를 이용하여 구현하였지만, 기본적으로 분산 병렬처리를 위한 것이고 자료교환과 프로세서 사이의 동기화를 위해 PVM과 MPI 라이브러리를 사용하였기 때문에, 망(network)으로 연결된 여러 대의 EWS (Engineering WorkStation)으로 이루어진 가상 병렬환경에서도 그대로 이용될 수 있다. 그리고 분산 병렬처리를 할 때 얻게 되는 장점은 계산시간의 단축뿐만 아니라, 문제의 크기가 매우 큰 경우 한 프로세서만으론 메모리가 부족하여 계산할 수 없게 되는데, 분산 병렬처리시 여러 프로세서를 이용하여 이를 해결할 수 있다. 분산처리에서는 각 프로세서가 독립적인 메모리를 가지고 있기 때문이다. 본 알고리듬은 3차원 형상 압연문제뿐만 아니라 정밀한 유한요소해석을 하기 위해서 많은 계산시간을 요구하는 다른 문제에도 적용이 가능할 것으로 기대된다.

후 기

본 연구는 한국과학재단 (과제번호 95-0200-14-04-3)의 지원을 받아 이루어졌으며, 이에 관계자 여러분께 감사드립니다.

형상 압연공정 유한요소해석 프로그램을 제공해 주신 한국과학기술원 기계공학과 임용택 교수님과 관계자 여러분께 감사드립니다.

참고문헌

(1) Chitkara F. R. and Hardy G. M., 1977, "Rolling of I-Section Beams Using Lead as a Model Material: Some Experimental Results," *Int. J. Mech. Sci.*, pp. 575~594.

(2) Altan T., Oh S. I. and Gegel H., 1983, "Metal Forming Fundamentals and Applications," *American Society of Metals*, Metals Park.

(3) Altan T. and Fiorentino R. J., 1971, "Prediction of Loads and Stresses in Closed-Die Forging," *Trans. ASME.*, pp. 477~484.

(4) Oh S. I., 1982, "Finite Element Analysis of Metal Forming Processes with Arbitrary Shape Dies," *Int. J. Mech. Sci.*, Vol. 24, pp. 479~493.

(5) Kiuchi M. and Yanagimoto J., 1988, "Computer Aided Simulation of Shape Rolling Processes," *Proc. of 16th NAMRC*, pp. 34~40.

(6) Komori K. and Kato K., 1989, "Analysis of Temperature Distribution in Caliber Rolling of a Bar," *JSME International Jounal, Series.1*, Vol. 32, No. 2, pp. 208~216.

(7) Mori K. and Osakada K., 1989, "Finite Element Simulation of the Three-Dimensional Deformation in Shape Rolling," *Numiform 89*, Balkhema Press, Rotterdam, pp. 337~342.

(8) Park J.J. and Oh S. I., 1990, "Application of Three Dimesional Finite Element Analysis to Shape Rolling Processes," *Trans. ASME.*, Vol.112, pp. 36~46.

(9) Mori K., 1990, "General Purpose FEM Simulator for 3-D Rolling," *Advanced Technology of Plasticity*, Vol. 2, pp .619~624.

(10) Shin H. W., Kim D. W. and Kim N. S., 1991, "Finite Element Analysis of I-Section Beam Rolling," Seoul National University J. Engineering Research, Vol. 23, No. 2.

(11) Chen B. K., Thomson P. F. and Choi S. K., 1992, "Temperature Distribution in the Roll-Gap During Hot Flat Rolling," *J. Mater. Proc. Tech.*, Vol. 30, pp. 115~130.

(12) Shin H. W., Kim N. S. and Park J.J., 1993, "The Analysis of H-Shape Rolling by the Finite Element Method," *J. KSME.*, Vol. 17, No. 5, pp. 1095~1105.

(13) Shin H. W., Kim D. W. and Kim N. S., 1994, "A Study on the Rolling of I-Section Beams," *Int. J. Mech. Tools Manufact.*, Vol. 34, No. 2, pp. 147~160.

(14) Badea L. and Gilormini P., 1994, "Application of a Domain Decomposition Method to Elastoplastic Problem," *Int. J. Solids Structures*, Vol. 31, No. 5, pp. 643~656.

(15) Hu N., 1994, "A Parallel Algorithm for Analyzing Elasto-plastic Problems," *Comput. Struct.*, Vol. 52, pp. 1127~1133.

(16) Kacou S. and Parsons I.D., 1993, "A Parallel Multigrid Method for History-dependent Elastoplasticity Computations," *Comput. Methods Appl. Mech. Engrg.*, Vol. 108, pp. 1~21.

(17) Ferian A., Franchi A. and Genna F., 1996, "An Incremental Elastic-plastic Finite Element Solver in a Workstation Cluster Environment Part II. Performance of a first Implementation," *Comput. Methods Appl. Mech. Engrg.*, Vol. 130, pp. 299~318.

(18) Giest A., Begulin A., et al., 1994, Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing, MIT Press.

(19) Message Interface Forum, 1995, MPI : A Message-Passing Interface Standard, University of Tennessee.

(20) GDB/RBD, 1996, MPI Primer/Developing with LAM, Ohio State University.

(21) Horie T. and Kuramae H., 1994, "Evaluation of Parallel Performace of Large Scale Computing using Workstaion Network," *WCCM Ⅲ*, Vol. Ⅱ, pp. 1569~1570.

(22) Shivakumar K.N., Bigelow C.A. and Newman J.C., 1992, "Parallel Computation in a Three-Dimensional Elastic-Plastic Finite-Element Analysis," *Computers & Structures*, Vol. 43, No. 2, pp. 237~245.

(23) Doltsinis I. S. and Nolting S., 1991, "Studies on Parallel Processing for Coupled field Problems," *Comp. Meth. in Appl. Mech. and Engrg.*, Vol. 89, No. 1-3, pp. 497~521.

(24) El-Sayed M.E.M. and Hsiung C.K., 1994, "Parallel Finite Element Computation with Separate Substructures," *Computers & Structures*, Vol. 36, No. 2, pp. 261~265.

(25) Saad Y., 1996, Iterative Methods for Sparse Linear Systems, PWS Publishing Company.

(26) Blackford L.S., Choi J., et al., 1997, ScaLAPACK User's Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

(27) Cleary A., Dongarra J., 1996, "Implementation of ScaLapack of Divide-and-Conquer Algorithms for Banded and Tridiagonal Linear Systems," Lapack Working Note.

(28) Law K.H., 1986, "A Parallel Finite Element Solution Method," *Computers & Structures*, Vol. 20, No. 6, pp. 845~858.

(29) Carter W.T., Sham T.L. and Law K.H., 1989, "A Parallel Finite Element Method and Its Prototype Implementation on a Hypercube," *Computers & Structures*, Vol. 31, No. 6, pp. 921~934.