論文

외연 Lagrangian 유한요소법 기반의 대규모 유한요소 모델 병렬처리

백승훈*, 김승조**, 이민형***

Parallel Computing of Large Scale FE Model based on Explicit Lagrangian FEM

Seung-Hoon Paik*, Seung-Jo Kim** and Minhyung Lee***

ABSTRACT

A parallel computing strategy for finite element(FE) processing is described and implemented in nonlinear explicit FE code and its parallel performances are evaluated. A self-made linux-cluster supercomputer with 520 CPUs is used as a bench mark test bed. It is observed that speed-up is increased almost idealy even up to 256 CPUs for a large scale model. A communication over head and its effect on the parallel performance is also examined. Parallel performance is compare with the commercial code and developed code shows superior performance as the number of CPUs used are increased.

초 록

비선형 외연 유한요소법에서 유한요소 병렬 처리 방안을 기술하고 코드에 구현하였다. 성능테스트 장비로 자체 구축한 520 개의 CPU를 갖는 리눅스 클러스터 슈퍼컴퓨터를 사용하였다. 대규모 모델 테스트 결과 256 개의 CPU 까지도 거의 이상적인 속도 증가를 보였다. 유한요소 계산시간 대비 통신시간 계산이 전체 성능에 미치는 영향도 검토하였다. 사용 프로세서가 증가할수록 상용코드의 병렬 성능 대비 더 좋은 성능을 보이는 것으로 나타났다.

Key Words : parallel computing(병렬계산), large scale problem(대규모 문제), linux-cluster (리눅스 클러스터), explicit time integration method(외연시간적분법)

Ⅰ. 서 론

컴퓨터의 가상공간상에서 구조물의 거동에 대한 수치모사는, 시험에 의한 검증 횟수의 단축, 그리고 우주환경과 같이 시험을 수행하기 어려운 극한 환경에서의 거동 모사 측면에서 널리 활용되고 있다. 항공 우주 분야에서 충돌 현상은 조류 충돌, 복합재료 충격, 헬기 추락, 랜딩기어의 충격 흡수 성능평가 등 여러 분야에서 중요하게 다루어지고 있다. 이러한 충돌/충격 현상 수치모사는 주로 외연 시간적분법 및 비선형해석을 기반으로 한 유한요소법을 사용하여 다루어지고 있으며, 시간에 따른 대 변형 운동 및 재료의 비선형 거동, 그리고 복잡한 접촉 문제를 동시에 다루어야 하기 때문에 많은 계산시간을 필요로 한다. 이러한 문제들을 단품을 대상으로 하기보다 전체 구조물을 대상으로 수치 모사를 하는 경우가 많고, 모델의 정밀화에 대한 필요성이 점차 증가 하고 있기 때문에, 통상적으로 많은 자유도의 유한요소 모델을 동반하게 된다. 따라서 주어진 기간 내에 신뢰성 있는 수치모사를 위해서는, 큰 계산능력이 요구되며, 단일 CPU로는 이러한

† 2006년 5월 26일 접수 ~ 2006년 7월 5일 심사완료
* 정회원, 서울대학교 기계항공공학부 대학원
** 정회원, 서울대학교 기계항공공학부/비행체특화센터
연락저자, E-mail : sjkim@snu.ac.kr
서울시 관악구 신림동 산 56-1
*** 세종대학교 기계항공우주공학부

요구사항을 충족시킬 수 없어 여러 컴퓨터의 능력을 동시에 사용하는 병렬계산 기술이 특히 필요하다. 충돌해석의 병렬화와 관련해서는 선진국은 이미 충돌해석 프로그램의 병렬화 기술을 개발하여 일부 응용하고 있으며, 여러 관련 연구가 보고 되어 있으나[1,2], 국내에서는 이 분야의 연구가 미미한 실정이다.

충돌해석의 병렬계산을 위해 해결해야 할 여러 기술에는 병렬 컴퓨터 구축 및 튜닝 기법, MPI와 같은 통신관련 프로그램을 이용한 기능 향상 연구, 영역분할 자동화에 대한 연구, 부하균등 향상 기술 연구, 병렬화 효율 향상 기법 연구, 전-후 처리 병렬화 등 여러 해결해야 할 분야가 있다. 그 중, 병렬화 효율 향상의 중요한 부분으로 유한요소의 내력벡터와 접촉력 계산의 병렬 효율향상이 있다. 접촉력의 경우, 구조물의 충격파 응답특성 분석과 같이 접촉력을 고려하지 않거나, 아니면 복합재료 충격문제와 같이 충격 면이 아주 국소 부위에 해당하는 경우처럼, 접촉력의 병렬화가 크게 문제가 되지 않는 경우도 있다. 그러나 유한요소 계산은 강체가 아닌 경우를 제외하고 항상 계산 되어야 하므로, 이 부분의 병렬 효율 극대화는 충돌해석 병렬효율향상의 선결조건이 된다. 본 연구에서는, 유한요소 계산의 병렬화 과정에 대해 자세히 기술하고, 접촉력 처리가 크게 문제되지 않는 예제인 테일러 임팩트 테스트(Taylor Impact Test)를 대상으로, 자체 제작한 리눅스 클러스터 환경에서 그 병렬 성능을 평가, 분석 하였다.

충돌 현상을 수치 모사할 수 있는 상용코드들이 많이 있으나, 새로운 수치알고리듬과 재료 모델링해석을 위해 자체 코드를 보유하고 있기도 하다. 본 연구팀에서도 이러한 목적과 더불어 효율적인 병렬 알고리들을 개발할 목적으로 IPSAP/Explicit (IPSAP: Internet Parallel Structural Analysis Program)을 자체 개발하고 있다[3]. 본 논문에서는 비선형외연유한요소법에 기반한 IPSAP/Explicit 에 구현한 병렬기법을 기술하고, 그 병렬성능을 평가하였다.

단일 프로세서 환경에서 해석코드의 신뢰성은 여러 참고문헌의 실험 및 다른 해석 코드와의 결과와 비교하여 확인하였다[4].

II. 유한요소 계산의 병렬화

Lagrangian 방식의 외연적 시간적분법을 기반으로 한 충돌 해석의 병렬처리는 크게 두 부분으로 나누어진다. 하나가 유한요소의 계산, 즉 내력벡터 계산의 병렬처리이고, 다른 하나는 접촉력 계산의 병렬처리이다.

유한요소 계산은 각 요소에 대해 절점의 속도 및 변위를 가지고, 응력을 구한 후, 응력값을 이용하여, 절점에 작용하는 내력을 벡터 형태로 구하는 과정이다. 각 요소에 대해 계산을 독립적으로 수행하므로, 절점을 공유하는 주변 유한요소와 데이터를 연성해서 풀 필요가 없고, 따라서 각 프로세서는 주변 영역과 데이터 교환 없이 독립적으로 내력벡터를 계산할 수 있다. 병렬화 관점에서 이러한 점이 외연적 유한 요소법의 큰 장점이라 할 수 있다.

다만 각 프로세서 별로 내력벡터를 구하는 과정이 끝나면, 영역간의 경계면에 위치한 절점들에 대해서는 경계면을 공유한 프로세서간의 통신을 통해서 서로 내력벡터 값을 교환 하고, 받은 내력 값은 내 영역의 경계면의 해당 절점에 합하면 된다. 이를 위해서, 각 프로세서들은 (1)자신과 영역을 공유한 프로세서(혹은 도메인) 개수 및 프로세서 리스트, (2) 자신과 영역을 공유한 각 도메인과의 공유 노드 개수 및 노드 리스트들에 대한 정보를 미리 작성해서 배열에 저장하고 있어야 한다. 메쉬 연결 상태는 처음부터 끝까지 변하지 않으므로, 각 프로세서는 처음에 한 번만 이러한 작업 수행하면 된다. 이렇게 되면, 매 시간스텝마다, 자신은 누구에게 어느 데이터를 주고, 누구로부터 어떤 데이터가 오는지 결정되게 된다. 한편 질량벡터는 처음에 한번 결정되면, 계산 끝까지 변하지 않기 때문에 처음에만 통신을 해주면 된다.

내력벡터의 통신과 동일하게 경계면의 절점의 질량을 서로 주고/받은 후 내 영역에서 계산된 질량벡터에 더하면 된다. 인접영역과 데이터를 주고받는 개념적인 그림을 Fig. 1 에 도시하였다.

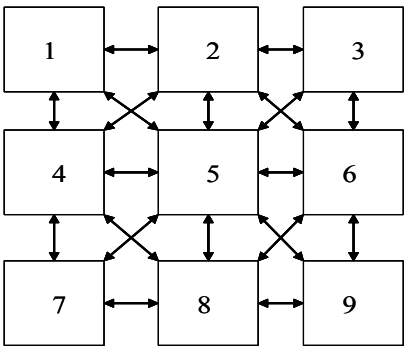

Fig. 1. Communication of Internal force between FE domains

한 편 영역 간 경계면의 값(내력벡터)을 교환하고, 내 영역에서 계산된 값에 합산해야 하는데, 이를 위한 단계별 통신 패턴은 다음과 같다.

Step 1.	송신할 데이터를 송신버퍼에 저장한다.
Step 2.	각 프로세서가 인접영역 개수만큼, Non-Blocking MPI_RECV를 호출하고, 각 영역에서 데이터를 받을 공간을 확보한다.
Step 3.	경계면 영역의 내력 벡터들을 해당 인접영역 (프로세서)들에게 MPI_SEND를 통해 Blocking 모드로 송신한다.
Step 4.	각 영역으로부터의 데이터 수신이 완료되었는지 확인한다.
Step 5.	수신한 데이터를 내력벡터에 갱신한다.

Step 1에서, 송신 버퍼의 개수는 인접영역과 개수가 같고, 각 송신 버퍼의 크기는 해당 인접영역과 공유하고 있는 총 절점들의 총 자유도 개수 만큼이다. 내력벡터 통신 시, Non-blocking 통신 모드의 경우에는 MPI_Wait 함수를 호출하게 된다. 그런데, 데이터 수신의 경우, 어느 프로세서에서 먼저 데이터가 오는지 알 수 없기 때문에, MPI_Wait 함수를 호출하기 이전에 자신에게 데이터를 보낸 프로세서 번호들을 받은 순서대로 MPI_Request 구조체의 정보를 이용해서 특정 배열에 저장하게 된다. MPI_Wait 함수까지 호출한 이후에는, 이 특정배열의 정보를 이용하여, 수신한 내력벡터를 내 영역에서 계산한 내력 벡터에 더해서 갱신을 완료한다. 이와 같이하며, 통신이 임의의 프로세서 p에 대해 도착하는 메시지의 순서가 일정하지 않는 경우에도 안전한 통신을 보장하게 된다.

한편, 외연적 유한요소기법에서는 내력벡터가 유한요소 단위로 계산되고, 계산된 내력벡터가 전체 내력벡터에 합치되며 이러한 과정을 유한요소 개수만큼 반복하면 된다. 따라서 전체 강성행렬을 만들 필요가 없고 질량도 대각행렬만 나오게 되기 때문에 내연적 방법에서와 같이 [K]{u}={F} 같은 선형연립방정식을 풀 필요가 없다. 이러한 이유로 외연적 적분방법은 선형연립방정식을 푸는 경우에 비해 메모리를 매우 작게 차지한다.

III. IO(Input/Output) 및 기타 부분 병렬처리

개발된 코드에서는 계산이 시작되면 모든 프로세서가 동일한 입력파일을 읽도록 되어 있으므로, 동일한 경로에 입력파일과 필요한 경우 영역분할에 대한 정보가 있는 파일이 있어야 한다. 실행파일도 동일한 경로에 있어야 한다. 물론, 실행파일의 위치와 입력파일의 위치는 달라도 상관없다. 따라서 NFS(Network File System)으로 묶여져 있으면 상관없지만, 그렇지 않은 경우는, 각 계산 노드별로 위의 파일들을 복사해 놓아야 한다. 결과를 출력할 때에는 각 계산 노드에서 계산된 결과를 0번 프로세서로 모두 모아서 0번 프로세서가 출력파일에 쓰는 방식을 취하였다. 이때, 영역 간 경계면의 절점의 값들을 그 절점이 공유된 개수만큼 데이터를 중복해서 받게 되므로, 모두 받아서 더한 후, 공유된 개수로 나누어 주었다. 이 밖에 병렬계산을 위해 통신을 해야 하는 변수로는 임계시간스텝이 있다. 임계시간스텝은 각 유한요소마다 계산되는 값이므로, 각 영역에서 계산된 임계 시간스텝 값을 서로 통신하여 그 중 최소값을 그 다음 스텝의 시간스텝 크기로 정하면 된다.

한편, 선형연립방정식을 푸는 경우, 패널티 방법을 사용하지 않는 경우라면, 경계조건에 해당하는 자유도를 강성행렬과 하중벡터에서 소거하고, 소거된 강성행렬 자유도와 연성된 하중 벡터를 갱신하여 풀게 된다. 이런 경우는 경계조건이 많이 정의되어 있을수록 풀어야할 자유도 수가 줄어들게 되므로, 사용해야 할 메모리나 계산시간이 함께 줄어들게 된다. 그러나 외연적 유한요소법에서는 경계조건처리 방식에서는, 변위 혹은 속도 경계조건은 특정 절점들에 대해 시간에 대한 함수로 입력 파일에서 미리 주어지게 되는데, 현재 시간에서 경계조건이 할당된 절점의 변위는 속도로 환산하고, 속도는 주어진 속도 값을 그대로 사용하여, 해당 절점에 지정하는 방식 즉, 속도 경계조건 부여 방식으로 처리 된다. 속도나 변위 경계조건들은 (x,y) 좌표값 형태로 시간에 따라 변하는 속도나 변위를 지정할 수 있는데, 좌표 값은 매 시간스텝마다 지정하지는 않으므로, 현재 시간에서의 속도는 주어진 경계조건 좌표 값을 선형 보간하여 사용한다. 속도 경계조건이 부여되면, 그에 상응하는 변위경계조건도 시간스텝 적분과정에 의해 자연히 결정되게 된다. 따라서 경계조건이 부여된 절점개수가 많을수록 그 만큼의 추가적인 계산이 더 필요하게 되며, 어느 특정 부 영역에 경계조건이 모두 지정되어 있으면, 부하 균등을 저하시키는 원인이 될 수 있다.

IV. 병렬효율 측정방법

외연적 시간적분방법에서 병렬효율 성능 측정

방법으로 유용하게 사용되는 기준으로 Grind time, 고정크기 모델의 Speed-Up (Fixed size Speed-Up), 확장성 테스트 Speed-Up (Scaled test Speed-Up), 병렬효율 (parallel efficiency) 등이 있다[1].

Grind time T_{grind}은 존-싸이클 타임(zone cycle time)이라고도 하며, 단일 CPU 컴퓨터와 병렬 컴퓨터의 성능을 측정할 때 모두 사용되며, 다음과 같이 정의된다.

$$T_{grind} = \frac{T_{execution}}{N_{elements}\, N_{cycles}} \tag{1}$$

$T_{execution}$은 실제 계산시간으로 Elapsed Time 을 의미하며, $N_{elements}$는 총 유한요소 개수, N_{cycles}는 총 싸이클 (전체 계산 스텝) 수 로, 한 싸이클 내에서 하나의 요소를 계산하는데, 소요되는 시간이다.

고정크기 모델 Speed-Up S_f는 다음과 같다.

$$S_f = \frac{T_{m,1}}{T_{m,p}} \tag{2}$$

여기서, $T_{m,1}$은 m개의 유한요소를 가진 문제를 단일 CPU에서 계산할 때 소요된 시간 (Elapsed Time)이고, $T_{m,p}$ 는 같은 모델을 p개의 CPU로 계산했을 때 소요된 계산시간이다.

확장성 테스트 Speed-Up은 사용 CPU 증가와 정비례해서 모델의 크기(유한요소 수)를 증가시키며, 테스트하는 방식을 다음과 같이 정의된다.

$$S_s = \frac{T_{m,1}p}{T_{m\times p,p}} \tag{3}$$

여기서, $T_{m\times p,p}$ 는 (m*p) 크기의 모델이 p 개의 CPU에서 계산될 때 소요된 계산시간(Elapsed Time)이다.

충돌 해석의 병렬 성능을 비교하는 경우, 해석의 종료시간 (termination time)은 입력변수로 주어져 있으므로, 객관적인 비교를 위해서는 종료시간까지의 총 시간스텝 수인 N_{cycle} 이 동일해야 하는 전제조건이 있어야 한다. 그러나, 시간스텝은 메쉬 사이즈에 의해 매 시간마다 새롭게 결정되므로, 모델의 자유도나 요소수가 동일하더라도, 메쉬 사이즈가 다르거나, 시간스텝을 결정하는 기준 등이 달라 시간스텝에 변화 즉, 총 스텝 수의 차이를 가져오는 요인이 있는 경우, 총 스텝 수에 비례해서 측정된 Elapsed time 을 단순 비교하면 객관성이 떨어질 수 있다. 또한 요소 수 증가에 따른 병렬효율을 측정할 때에도 요소 수 증가의 효과 이외에도 요소사이즈 감소에 의한 시간스텝 감소 효과 즉, 총 스텝 수 증가 효과가 있으므로 이런 경우에도 Elapsed time 만을 가지고 병렬 성능을 비교하면 객관성이 떨어질 수가 있다. 따라서 Elapsed time을 총 요소수와 총 싸이클 수로 나눈 값인 Grind time을 성능을 비교 지수로 사용한다. 물론, 동일 input file 을 동일한 병렬 코드로 동일한 컴퓨팅환경에서 프로세서 수만 변화시켜가며 비교할 때는 $N_{element}$ 와 N_{cycle} 이 같으므로 Elapsed time인 $T_{execution}$ 만 비교하면 된다.

한편, 모델의 크기를 따질 때 자유도 보다는 유한요소 개수를 가지고 따지는 경우가 많은데, 이는 선형연립방정식을 풀어야 하는 해석의 경우에는 대부분의 시간이 선형연립방정식의 해를 구하는데 소요되므로, 강성행렬의 크기 즉, 자유도 수가 문제의 크기를 비교하는 중요한 척도가 되지만, 내력벡터를 계산하는 외연적 시간적분방법에서는 접촉이 없는 경우, 대부분의 시간이 유한요소의 내력벡터를 계산하는 데 소요되고, 사용하는 유한요소 모델이 거의 유사하기 때문에 전체 계산시간은 유한요소의 개수에 비례하게 된다. 이러한 이유로 여러 관련 논문에서 충돌 해석 모델의 크기를 따질 때 자유도 수 보다도 주로 유한요소의 개수를 가지고 비교하는 경우가 많다.

Ⅴ. 병렬 성능 평가용 컴퓨터 시스템

병렬 알고리듬 성능 평가에서 어려운 점 중의 하나는 바로 대규모 문제에 대해 병렬 성능을 테스트 할 수 있는 대형 병렬 컴퓨터 시스템의 확보이다. 병렬 Speed-Up 을 측정하기 위해서는 다른 사용자의 방해 없이 수 백 개 이상의 프로세서를 단독으로 활용할 수 있는 환경이 되어야 한다. 본 연구실에서는 병렬 알고리듬 개발 및 테스트를 목적으로 PEGASUS 시스템을 개발하고 활용하고 있다. PEGASUS 시스템은 260개의 계산 노드로 구성되어 있으며, 각 노드마다 2개의 Xeon 프로세서를 장착하고 있다. Table 1 은 시스템의 계산 노드에 따른 CPU 정보이다. 병렬 성능 측정시 256개의 CPU 이내의 측정은 모두 2.2Ghz CPU 를 이용하였고, 256개 CPU 이상 사용한 경우 Speed-Up 측정 시 기준이 되는 분모는 2.2 Ghz 에서의 시간을 사용하였다. 각 계산 노드는 3~6 GBytes의 메모리와 80~200 GBytes의 하드디스크를 가지고 있으며, Gigabit 네트워크로 연결되어 있다. 한편, 많은 병렬 코드들이

MPI (Message Passing Interface) Library 를 사용하여 데이터를 송/수신하고 있다[5]. 대표적으로 많이 사용되는 LAM/MP와 MPICH의 통신성능을 비교한 결과, LAM/MPI의 성능이 좀 더 우수한 것으로 나타났기 때문에, LAM/MPI를 통신을 위한 라이브러리로 사용하고 있으며, 본 연구에서도 이를 사용하였다.

Table 1. CPU information of PEGASUS linux cluster system

Computational Node	NCPU	CPU
n001-n128	256	Intel Xeon 2.2 Ghz
n129-n184	112	Intel Xeon 2.4 Ghz
n185-n220	72	Intel Xeon 2.8 Ghz
n221-n260	80	Intel Xeon 3.0 Ghz

Ⅵ. 수치예제 : 테일러 임팩트 테스트

테일러 테스트는 실린더 형태의 금속 막대가 강체 벽면에 충돌되는 테스트로, 대변형과 중간 정도의 변형률 속도 및 압력 (moderate strain rate and pressure)을 동반하는 실험실 단위에서 수행할 수 있는 간단한 임팩트 테스트이다. 충돌 해석 및 알고리듬의 검증용으로 자주 인용되고 있다. 초기 형상과 물성치, 초기조건 등은 Table 2에 제시된 것과 같다. 재료는 선형 경화 탄-소성 모델을 사용하였다. 초기 충돌속도인 227m/sec는 재료의 소성파 전파속도와 거의 같은 속도이다. 계산은 금속 막대가 소성 압축 변형 후, 정지 상태에 이르는 80μs까지 계산을 하였다. 임팩트 되는 바닥면은 충격방향으로의 변위만 구속하였다. 변형 결과는 Fig. 2와 같다. Table 3에 ABAQUS/Explicit 과 LS-DYNA로 해석한 결과와 비교 하였다. 모든 경우에서 절점 개수는 1,369 개이고, 요소는 8 절점 육면체 요소로 개수는 972개이다.

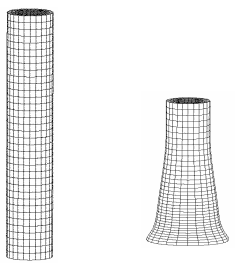

Fig. 2. Initial and deformed shape of Taylor impact model

이들 상용코드와 비교했을 때, 결과가 잘 일치하였다.

Table 2. Material properties and geometry of Taylor impact model

Properties	Value
Density(Kg/m^3)	8,930
Young's Modulus(GPa)	117
Poison Ratio	0.35
Initial Yield Stress(GPa)	0.4
Hardening Modulus(GPa)	0.1
Initial Velocity(m/sec)	227
Initial Length(mm)	32.4
Initial Radius(mm)	3.2

Table 3. Comparison of length and radius with commercial codes

Code	Length(mm)	Radius(mm)
ABAQUS/Explicit	21.48	7.08
LS-DYNA	21.23	6.18
IPSAP/Explicit	21.52	7.00

Ⅶ. 병렬성능 결과

유한요소 영역분할은 주로 METIS[6], Charco와 같은 그래프분할 알고리듬 기법을 사용하게 되며, 다른 유사 코드에서도 유한요소의 병렬화를 위한 부하 균등 시 이러한 그래프 분할 기법을 사용하고 있다[1,2,7].

본 연구에서도 METIS 에서 제공하는 그래프 영역분할 함수를 이용하였다. METIS의 경우, 모델의 사이즈가 어느 이상으로 커지게 되면, METIS 함수의 그래프 생성과정에 많은 메모리를 필요하게 되므로, 병렬 METIS를 이용하여 영역을 분할하여야 한다. 본 예제에서 사용된 모델 중 1000만 자유도 이상의 모델은 클러스터 슈퍼컴퓨터의 단일 노드의 최대 메모리인 6 GB이상을 요구하므로, 모두 병렬 METIS를 이용하여 분할하였다.

7.1 고정 크기 모델 Speed-Up

약 1000만 자유도(3,538,944 요소, 3,641,073 절점) 수준의 모델을 대상으로 고정크기 모델의 Speed-Up을 측정하였다. Fig. 3은 사용 CPU 증가에 따른 Grind time 및 Speed-Up이다. 노드당 1개의 CPU를 사용한 경우 128개의 CPU를

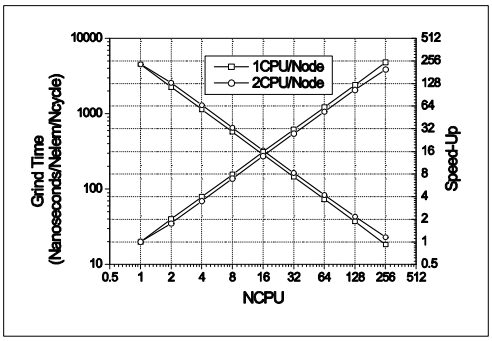

Fig. 3. Grind time and Speed-Up for fixed size model (1 CPU/Node vs. 2 CPU/Node)

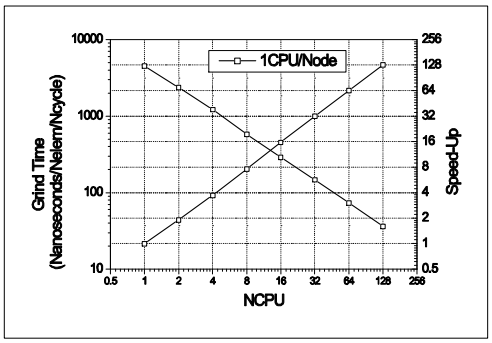

Fig. 4. Grind time and Speed-Up for scaled model (1CPU/Node)

사용한 경우보다 122배의 속도를 보이며, 거의 선형적으로 증가되고 있다. 노드 당 2개의 CPU를 사용하는 경우도 128개의 CPU에서 105배의 속도성능을 보이다가, 256개의 CPU에서 151배의 속도성능을 보여주고 있다. 3GB의 메인 메모리를 가진 단일 노드에서도 1000만 자유도 수준의 모델의 실행이 가능하였다. 실제로는 1000만 자유도 문제를 푸는데 단일 노드에서 약 2GB의 메모리를 사용하였는데, 이것은 코드에 적용한 외연시간적분법이 강성 매트릭스를 조립하지 않고, 하나의 요소에서의 [K]{u}에 해당하는 부분을 내력벡터로 계산하여, 전체 내력벡터 (global internal force vector)에 조립하는 방식이기 때문에 가능한 것으로, 일반 선형연립방정식을 풀어야 하는 해석기법에서는 심각한 메모리의 부족현상을 겪게 된다.

7.2 확장 크기 모델 Speed-Up

단일 CPU에서 55,296개의 요소를 가진 모델을 기준으로 CPU 증가에 따라 모델의 크기를 사용 CPU 개수와 비례하여 증가시키며 스피드 업을 측정하였다. 128개의 CPU를 사용한 경우 700만 개의 요소 (2000만 자유도) 수준까지 모델이 확장되었다. 확장성 테스트의 경우와 마찬가지로 Speed-Up이 128개의 CPU를 사용한 경우 까지도 선형적으로 증가하고 있음을 보여주고 있다(Fig. 4).

7.3 유한요소계산 대비 통신량의 영향

약 44만 유한요소 모델에 대해 CPU 개수를 증가 시켜가며 속도향상과 통신시간 비율을 측정하였다. 병렬효율에 미치는 통신량의 영향을 알아보기 위해, 모델의 길이 방향으로만 영역을 분할하여, 사용 CPU가 증가 하면서, 경계면 절점 개수 즉, 통신량은 그대로 유지한 채, 각 프로세서가 처리해야 할 요소 수는 점점 줄어들도록 하였다. 양쪽 경계면 절점 개수는 3,554 개로 128개 CPU부터는 하나의 프로세서가 통신하는 절점 개수가 계산해야하는 요소수보다 많아지게 된다. Fig. 5에서 보는 바와 같이 유한요소계산은 사용 CPU 개수 증가에 따라 이상적으로 증가를 하나, 통신시간은 조금씩 증가하다 512개의 CPU 에서는 유한요소계산시간 보다 더 많은 계산시간을 차지하며 속도향상이 더 이상 나타나지 않게 된다. 결국 접촉이 없는 경우, 병렬효율의 감소의

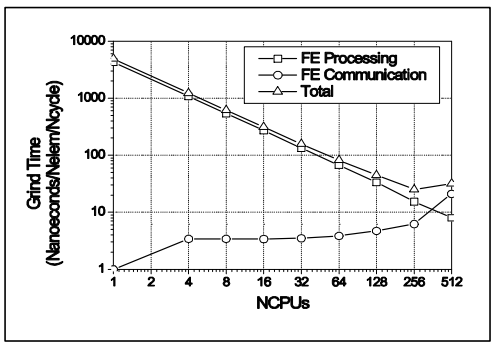

Fig. 5. Grind time of FE processing and FE communication

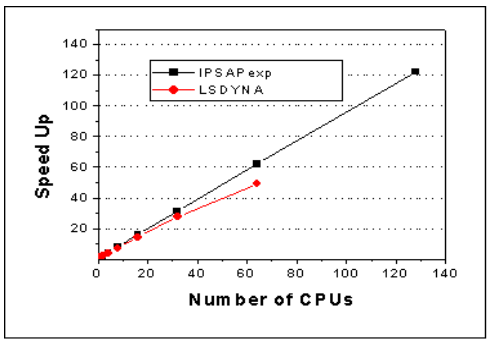

Fig. 6. Speed-Up of IPSAP/Explicit and LS-DYNA

원인은 통신시간 대비 유한요소 계산시간의 비율이라 할 수 있다.

7.4 LS-DYNA 와의 병렬 성능 비교

고정크기모델 (fixed size model : 350만 요소, 1000만 자유도)을 대상으로 LS-DYNA와의 계산속도를 비교하였다. 본 연구에서 개발된 코드의 실수 (real)변수는 모두 double precision 이므로, LS-DNA도 v.970 double precision version 과 비교하였다.

IPSAP/Explicit의 경우, 영역분할 정보가 있는 파일을 각 계산 노드가 가지고 있다가, 계산 초기화 과정에서 이 파일의 정보를 읽어 들여서 병렬작업에 들어가지만, LS-DYNA의 경우, 계산의 초기화과정에는 영역분할을 수행한다. 따라서 계산시간 측정 시, 영역분할과 입력파일을 읽어 들이고 변수를 초기하는 일련의 초기화과정은 제외하고 실제 시간스텝이 진행되는 부분만 측정하였다. 또한 계산과정 중 결과 출력은 하지 않도록 하였다. Table 4 에서 LS-DYNA의 Grind Time 은 이러한 효과가 고려된 "Clock time per zone cycle" 값이다. 이 값은 LS-DYNA 실행 종료 후 d3hsp file에서 제공되는 값이다. 계산은 1μs 까지 수행했고 이때의 총 싸이클(스텝)수 Ncycle 는 IPSAP/Explicit 의 경우 507, LS-DYNA의 경우 457이다. Elapsed Time 은 80 μs까지 계산했다고 가정했을 때의 추정치로 Elapsed time = Grind time × (N_{cycle} × 80) × $N_{element}$의 값이다. 실제로는 80 μs까지 계산이 진행되면, 충격부위에 메쉬가 길이방향으로 작아질 것이고, 그러면 시간스텝 사이즈 감소, 총 스텝 수 증가로 이어져 실제 Elapsed time 은 Table 4 의 추정치보다는 커질 것으로 예상된다.

본 예제의 경우, LS-DYNA double precision version이 1개의 CPU에서는 메모리 부족으로 실행이 되지 않아, 2개의 CPU에서 계산된 Grind time의 2배 값으로 추정하여 Speed-Up을 계산하였다(Single precision version은 1개의 CPU에서 실행됨).

앞에서도 언급했듯이 서로 다른 병렬코드의 경우 시간스텝 결정방법이나 사용자 지정 scale factor 등의 차이로 인해 총 싸이클 수가 달라질 수 있다. 이 예제의 경우에서도 IPSAP/Explicit 과 LS-DYNA의 총 싸이클 수는 다르므로, 싸이클 수가 고려된 Elapsed time을 상대 비교하는 것보다는 다른 관련 논문에서 비교하는 방법과 같이 Elapsed Time을 총 싸이클 수로 나누어 준 Grind Time 을 비교하는 것이 더 객관적인 비교라 할 수 있다. Fig. 6에서는 IPSAP/Explicit 과 LS-DYNA의 Speed-Up을 비교하였다. Grind Time 을 비교했을 경우, 1, 2개의 CPU 에서의 결과는 별 차이 없으나, 사용 CPU개수가 증가할수록, IPSAP/Explicit 의 병렬 효율이 더 좋게 나타나는 것을 확인할 수 있다. 64개의 CPU의 경우 LS-DYNA에 비해 약 25%정도의 Speed-Up 을 보이고 있다.

Table 4. Comparison of elapsed time and speed-up between IPSAP/Explicit and LS-DYNA

NCPU	IPSAP/Explicit (N_{cycle} = 507*)			LS-DYNA_double (N_{cycle}= 457*)		
	Grind Time (nsec)	Speed-Up	Elapsed Time**	Grind Time (nsec)	Speed-Up	Elapsed Time**
1	4540	1.00	181 hr	-	-	-
2	2260	2.00	90.5 hr	2270	2.00	81.6 hr
4	1140	3.98	45.5 hr	1295	3.51	46.4 hr
8	575	7.90	22.9 hr	623	7.28	22.4 hr
16	287	15.8	11.5 hr	316	14.4	11.3 hr
32	147	31.3	5.8 hr	163	27.9	5.8 hr
64	73.3	62.0	2.9 hr	92	49.4	3.3 hr
128	37.2	122.0	1.5 hr	-	-	-

* 1μs까지의 총 싸이클(스텝)수를 기준으로 계산된 시간

** 80μs까지 계산했다고 가정했을 때의 추정치
Elapsed time = Grind time × (N_{cycle} × 80) × $N_{element}$ (3,538,944)

Ⅷ. 결 론

비선형 외연 시간적분 유한요소 코드의 병렬 성능 향상을 위한 효율적 통신 기법을 제시하였고, 테일러 임팩트 문제에 대해 그 병렬효율을 살펴보았다. 그 결과를 요약하면 다음과 같다.

(1) 테일러 테스트 예제의 경우, 대규모 모델에 대해 256개의 CPU까지도 지속적인 속도 증가를 보였다. 더 많은 CPU에서도 속도증가가 가능할 것으로 기대된다. 1000만 자유도 수준의 모델이 단일 노드에서 계산되는데 약 2GB 정도의 메모리를 필요로 했다. 따라서 개발된 코드의 경우, 대규모 충돌문제를 수치 모사하는데 효과적으로 대응할 수 있을 것으로 판단된다.

(2) 노드 당 2개의 CPU를 사용하면, 노드 당 1개의 CPU을 사용할 때보다 약 10% 정도 느린 것으로 나타났다. 병렬 성능향상의 극대화를 위

한 계산인 경우 노드 당 1개의 CPU 사용을 권장한다.

(3) 동일 유한요소 모델에 대해 사용 CPU증가에 따른 통신 부하의 영향을 살펴보았다. 통신부하의 증가량과 비례하여 전체 효율이 감소되었으며, 약 44만 유한요소 모델의 경우, 512개의 CPU에서는 통신시간이 유한요소 계산시간 보다 크게 나오기 시작해 이 이후부터는 전체 계산시간을 오히려 증가될 것으로는 예상된다.

(4) 상용코드인 LS-DYNA와 병렬 성능을 비교하였다. CPU개수가 증가할수록 LS-DYNA 보다 더 나은 Speed-Up을 보였는데, 64 개의 CPU에서 약 25%정도 효율이 좋음을 알 수 있었다.

후 기

이 연구는 ADD 장기 기초 과제(UD040012AD)의 지원을 받아 수행되었습니다.

참고문헌

1) Attaway S.W., Hendrickson B.A, Plimpton S.J., Gardner D.R., Vaughan C.T., "A Parallel contact detection algorithm for transient solid dynamics simulation using PRONTO3D" *Computational Mechanics*. Vol. 22, 1998, pp. 143-59.

2) Kevin Brown, Steve Attaway, Steve Plimpton, Bruce Hendrickson, "Parallel Strategies for crash and impact simulations", *Comput. Methods Appl. Engng.*, Vol. 184, 2000, pp. 375-390

3) Paik, S.H. Moon, J.J, Kim, S.J and Lee, M., "Parallel performance of large-scale impact problem in linux cluster super computer", *Computers & Structures*, Vol.84, 2006, pp. 732-741.

4) Paik, S.H., Kim, S.J., "High speed impact and penetration analysis using explicit finite element method", *Journal of the Korea Institute of Military Science and Technology*, Vol. 8, No. 4, 2005, pp. 5-13.

5) Malard J. MPI: A message-passing interface standard. history, overview and current Status. Technology Watch Report, 1996, Edinburgh Parallel Computing Center.

6) Karypis G., Kumar V., Metis: Unstructured graph partitioning and sparse matrix ordering system. Technical Report, 1995, Department of Computer Science, University of Minnesota.

7) Malone J.G., Johnson N.L., "A parallel finite element contact/impact algorithm for non-linear explicit transient analysis : Part Ⅱ Parallel implementation", *Int. J. Num. Meth. Eng.*, Vol. 37, 1994, pp. 591-603.