1320 대한기계학회논문집 A권, 제30권 제10호, pp. 1320~1327, 2006

〈 논문 〉

Two-Step Eulerian 기법에 기반 한 충돌 해석의 병렬처리 및 병렬효율 평가

백승훈† · 김승조* · 이민형**

(2006년 5월 24일 접수, 2006년 7월 25일 심사완료)

Parallel Procedure and Evaluation of Parallel Performance of Impact Simulation Based on Two-Step Eulerian Scheme

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

Key Words: Parallel Computing(병렬 계산), Lagrangian Scheme(라그랑지안 기법), Two-Step Eulerian Scheme(2단계 오일러리안 기법), Remap(재배치)

Abstract

Parallel procedure and performance of two-step Eulerian code have not been reported sufficiently yet even though it was developed and utilized widely in the impact simulation. In this study, parallel strategy of two-step Eulerian code was proposed and described in detail. The performance was evaluated in the self-made linux cluster computer. Compared with commercial code, a relatively good performance is achieved. Through the performance evaluation of each computation stage, remap is turned out to be the most time consuming part among the other part such as FE processing, communication, time marching etc.

1. 서 론

충돌문제는 다른 해석 분야에 비해 계산 시간이 많이 필요로 하기 때문에, 병렬계산이 더욱 요구되는 분야이다. 고속 충돌 문제를 모사하는 기법에는 대표적으로 Lagrangian 방식과 Eulerian 방식이 있다. Lagrangian 기법의 병렬화 연구는 90년대 초반 이후로 본격적으로 시작되었는데, PRONTO3D 코드의 경우, 특별히 디자인된 OS를 갖춘 Intel Paragon, Intel Teraflops 장비에서 수천 CPU 까지도 성능향상을 보이는 결과를 보고하고 있다.[1,2] CTH와 같은 대표적인 Eulerian 고속충돌해석 코드의 경우, 병렬 환경에서 실제 적용문제를 풀 때, 계산이 보통 수일이 소요되는 등[3] 아직도 계산 시간에 대한 부담이 크기 때문에 병렬효율에 대한 연구는 지속적으로 필요하다. 고속 충돌 해석을 위한 Eulerian의 경우, 자세한 병렬화 과정이나 병렬성능 평가결과 등은 미국 국립연구소를 중심으로 한 특정 연구그룹에 의해 주로 이루어지고 있으며, 특히 국내의 경우 관련 연구는 거의 찾아 볼 수 없는 실정이다.

본 논문에서는 효율적인 병렬 알고리듬 개발 및 성능평가를 위해 개발된 코드를 이용하여 Eulerian 기법에 구현한 병렬화 방안을 자세히 기술하고, 자체 구축한 리눅스 클러스터 병렬 컴퓨터 환경에서 병렬성능 및 Eulerian 기법의 각 계산 단계에서의 부분별 병렬특성을 분석하였다.

상용코드를 이용하여도 충돌해석을 수행할 수도 있지만, 많은 연구 그룹들이 새로운 수치 기법 및 재료모델 테스트 등을 이유로, 특수 목적

† 책임저자, 회원, 서울대학교 대학원 기계항공공학부
* 회원, 서울대학교 기계항공공학부, 비행체 특화센터
E-mail : sjkim@snu.ac.kr
TEL : (02)880-7388
** 회원, 세종대학교 기계항공우주공학부

의 코드를 자체 개발하여 사용하고 있다. 본 연구팀도 이와 같은 목적과 더불어 병렬알고리듬을 테스트할 목적으로 병렬 충돌 해석 코드인 IPSAP/Explicit(IPSAP : Internet Parallel Structural Analysis Program)을 개발하고 있다. Lagrangian 방식에 적용된 수치기법, 해의 정확성, 병렬화 방안, 병렬 성능 등은 참고문헌[4,5]에 보고하였다.

Eulerian 코드는 Lagrangian 단계와 Remap 단계로 나누어 푸는 Two-step 기법이 가장 대표적이다.[6] 본 연구에서는 순수 Lagrangian 코드에 Remap 단계를 추가하여 개발한 Two-step Eulerian 코드에 대해 계산속도 향상을 위한 코드 최적화 과정과 병렬화 과정을 다루었다. 각 계산과정에서 시간 소요 비중 및 병렬 효율성을 평가하기 위해, 유한요소 계산 단계, Remap 단계, 통신시간 등의 계산 과정 병렬 성능을 분석하였고, 적용된 기법의 효율성을 평가하기 위해 상용코드와 비교하였다.

한편, 앞서 언급한 바와 같이, 대규모 충돌 문제의 병렬 성능 평가에 사용된 병렬컴퓨터는 특수하게 설계 제작된 OS를 기반으로 하는 경우가 많다.[1,2] 그러나 최근 각종 컴퓨터의 부품을 이전에 비해 저렴한 가격으로 구입할 수 있기 때문에, 자체 제작한 병렬 컴퓨터가 학교나 연구실 단위에서 구축하여 활용되고 있다. 그 중 가장 보편적인 형태는 리눅스 클러스터 형태이다. 이러한 자체개발 리눅스 클러스터에서의 병렬 성능 평가 결과는 특별히 제작된 OS 시스템에서의 성능 평가 결과에 비해 일반 리눅스 클러스터 사용자에게는 더 객관적인 비교 자료로 참고 될 수 있을 것으로 기대된다. 이러한 배경에서, 본 연구에서는 자체 개발 리눅스 클러스터에서의 병렬 성능 평가하고 그 결과를 제시하였다.

2. Two-step Eulerian 코드

2.1 코드 구조

Two-step Eulerian 코드는 Lagrangian 단계와 Remap 단계로 이루어져 있다. Lagrangian 단계에서는 시간 스텝 진전에 따른 메쉬(mesh)와 이에 따른 물질의 변형을 계산하고, Remap 단계에서는 시간진전 없이 변형된 메쉬에 있는 해(solution)들을 처음의 고정된 메쉬로 재배치하는 물질의 이류만을 계산한다.

Remap 단계는 (1) 체적유동(volume flux), (2) 재료유동(material flux), (3) 요소중심 값 이류(element-centered advection), (4) 절점중심 값 이류(vertex-centered advection) 의 4 단계이다.

재료 유동을 위한 재료 경계면 추적은 SLIC 기법,[7] 요소중심 값 이류에는 1차 정확도의 Donor cell 및 2차 정확도의 van Leer 방법,[8] 속도 이류에는 SALE 기법[9]을 사용하였다. 개발된 Two-step Eulerian 코드에 적용된 기법 및 검증 과정은 참고문헌[10]에 자세히 기술하였다.

2.2 계산속도 향상을 위한 코드 최적화

개발된 코드의 경우 Remap 부분이 전체 계산 시간의 80-90% 정도를 차지한다. 뒤에 설명되지만 LS-DYNA와 같은 상용코드에서도 동일한 양상을 보인다. 따라서 Remap 부분의 성능 최적화가 전체 코드성능을 좌우한다고 할 수 있다. 순수 Lagrangian 코드에서는(문제에 따라 크게 달라지는 접촉처리 부분을 제외하면) 유한요소의 내력벡터계산이 가장 많은 계산 비중을 차지한다. 유한요소 내력벡터는 각 유한요소에 대해 계산하여, 전체 내력벡터에 합하는 과정이다. 따라서 메모리 관리도 거의 모두 1차원 벡터로 관리되는 경우가 많다.

초기에 Remap 단계에서의 유한요소계산을 3차원 셀(cell) 구조의 I, J, K 루프에 .따라 계산되도록 하였다. 하지만 루프의 I, J, K 접근 순서와 3차원 메모리 구조의 접근 순서를 동일하게 해야 메모리에서 L2 cache로 한번 load/store 된 데이터를 가능한 많이 사용할 수 있다.[11] 즉 CPU가 L2 cache에서 데이터를 찾는 비율을 올릴 수 있게 되고, FLOPS가 증가하게 되어 전체 계산 시간을 대폭 줄일 수 있다.

본 연구에서 코드 최적화를 위해 3차원 배열의 메모리 할당(allocation)을 메모리 상에서는 1차원 벡터로 할당하여 데이터가 메모리 상에서 끊어지지 않고 연결되도록 하였다. 이와 동시에 I, J, K 루프계산 시 1차원 배열로 할당된 3차원배열의 데이터가 메모리 상에서 연결된 순서로 루프를 구성하였다.

3. 병렬 처리

3.1 유한요소 계산의 병렬처리

유한요소 계산은 각 요소에 대해 절점의 속도 및 변위를 가지고, 응력을 구한 후 그 값을 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 이것은 외연적 유한요소법의 큰 장점이다.

프로세서 별로 내력벡터를 구하면, 영역간의 경계면에 위치한 절점들에 대해서는 경계면을 공유한 프로세서간의 통신을 통해서 서로 내력벡터 값을 교환하고, 받은 내력 값은 내 영역의 경계면의 해당 절점에 합하면 된다. 즉 각 프로세서들은 자신과 영역을 공유한 프로세서 및 공유한 절점 리스트에 대한 정보를 가지고 있어야 한다.

메쉬 연결 상태는 처음부터 끝까지 변하지 않으므로, 각 프로세서는 처음에 한 번만 이러한 작업을 수행한다. 유한요소 부분 병렬화 및 병렬성능에 관련된 내용은 참고문헌[12]에서 다루었다.

3.2 Remap 단계 병렬화

요소중심 변수의 이류처리 시, 계산하고자 하는 요소의 각 I, J, K 방향으로 상류(upstream)와 하류(downstream)의 값이 필요하다. 특히 영역 경계면에 위치한 요소의 경우 요소의 면이 공유된 다른 영역의 요소의 값도 통신을 통해 주고 받아야 한다. Fig. 1(a)은 2차원에 대해, 각 경계면에서 2개 층의 요소의 값을 주고받는 경우에 해당한다. 이류 처리 시 대각선 방향의 요소값은 필요하지 않다. 받은 데이터는 이류 계산에 사용되므로, 가상의 buffer에 저장해 놓고 사용한다. Fig. 1(b)은 주변 영역으로 2개 층의 데이터를 받은 후 가상 buffer(shade 처리된 부분)에 저장한 상태를 표현한 그림이다. 이러한 영역 경계면에 가상 buffer는 가장자리에 있는 셀에서 체적유동, 물질유동, 경계면추적 등을 계산할 때 필요하므로 병렬이 아닌, 단일 프로세서 환경에서 계산하더라도 구성해야 하는 배열 구조이다.

한편 절점에서 운동량이나 질량을 계산할 경우, 영역 경계면의 절점에 연결된 모든 요소의 값을 주고받은 후 내 영역에서 계산된 값과 합산한다. Eulerian 메쉬구조는 처음 한번만 영역분할을 하면 통신 패턴이 끝까지 유지된다.

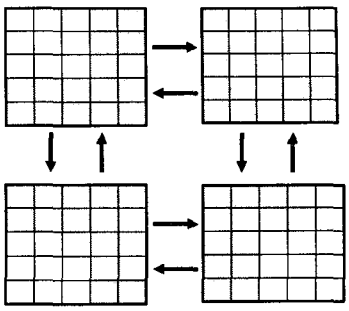

Fig. 1(a) Communication of cell centered value

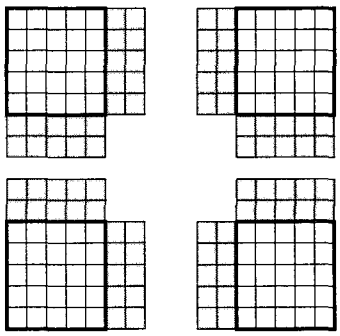

Fig. 1(b) Cell structure after communication of cell centered value

아래 Table 1은 Remap 단계의 병렬 계산 과정으로 밑줄 친 두 군데에서 통신이 이루어진다. 'Remap 계산에 필요한 경계면에서의 값 통신' 의 경우, 셀에서의 물질 경계면은 주변 요소의 VOF (Volume Of Fraction)을 이용하므로 영역 가장자리 셀의 경우 인접 영역의 셀과 VOF 값을 주고받아야 한다. SLIC 기법의 경우, 각 방향별로 전후 2개 요소의 VOF를 이용하여 셀의 물질경계면을 계산하므로, Fig. 1에서 보는 바와 같이 각각의 축 방향으로 2개씩 주고받는다.

제 3항 요소 중심 값의 이류의 경우, 1차 정확도는 상류와 하류의 셀 값이 각각 1개씩 필요하고, 2차의 정확도는 상류 셀 2개 하류 셀 1개의 값이 필요하다. 상류와 하류는 셀 면에서의 속도 방향에 따라 결정되므로, 영역 바깥쪽 면에서 셀 중심 값의 이류를 처리하기 위해, 인접영역과 1차 정확도인 경우 1개씩 주고받고, 2차 정확도인 경우 2개씩 주고받는다. 통신해야 할 셀 중심 값은 재료 모델에 따라 다르다. 탄소성 모델의 경우 밀도 1개, 응력텐서 6개, 등가소성변형률 1개, 인공점성 1개, VOF 1개 이다. 필요에 따라 내부

에너지 (1개)도 포함하면, 모두 11개 변수를 통신한다. 2개 씩 주고받는 경우, 영역 경계면에 있는 하나의 셀 당 22개 변수를 주고받는다.

제 4항의 절점 값 이류의 경우, 절점에서의 운동량, 질량을 계산해야 하는데, 절점의 운동량과 질량은 셀에서 계산된 값을 3차원인 경우 8로 나누어 셀의 절점에 할당하므로, 각 영역의 경계면에서 계산된 운동량과 질량은, 인접한 영역에서 계산된 값과 통신하여, 내 영역에서 계산된 값과 합치한다. 이 경우, 요소중심 값 계산 시, I, J, K 방향으로만 통신했던 것과는 달리 내력벡터 통신의 경우처럼 절점이 연결된 모든 영역과 통신해야 한다. 통신해야 할 항목은 각 방향 운동량 3개, 절점 질량 1개, 총 4개 이다. 통신이 끝나면 운동량을 질량으로 나누어 속도를 구한다.

Eulerian 메쉬는 통상 육면체를 이루므로 영역 분할방법도 I 방향으로 NI 등분, J 방향으로 NJ 등분, K 방향으로 NK 등분과 같이 각 방향으로 나누도록 하였다. 이 경우, 총 영역 수는 NI x NJ x NK 가 되며, 반드시 2^n 개로 나누지 않아도 되는 이점이 있다.

Table 1 Procedure of Parallel Remap Computation

1. Remap 계산에 필요한 경계면에서의 값 통신
   - VOF(1개), 밀도(1개), 응력텐서(6개),
   - 유효소성변형률(1개), 인공점성(1개)
   - 변형에너지(1개)

2. 재료 유동 계산
   - 체적유동, 경계면 추적,
   - 경계면에서 재료 유동량 계산

3. 요소중심 값 이류
   - VOF(1개), 밀도(1개), 응력텐서(6개),
   - 유효소성변형률(1개), 인공점성(1개),
   - 변형에너지(1개)

4. 절점 값 이류
   - 절점에서 운동량 계산, 질량 계산
   - 경계면의 절점 값 각 방향 통신
     절점 운동량(3 개), 절점질량(1개)
   - 절점속도 계산

4. 병렬 성능

4.1 병렬효율 측정 방법

외연적 시간적분방법에서 병렬효율 성능 측정 방법으로 유용하게 사용되는 기준으로 Grind time, Speed-Up 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.

$$T_{grind} = \frac{T_{execution}}{N_{elements} N_{cycles}} \quad (1)$$

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

한편, 병렬 성능을 측정하는 방식인 Speed-Up, S는 다음과 같이 측정한다.

$$S = \frac{T_1}{T_p} \quad (2)$$

여기서, T_1은 단일 CPU에서 계산할 때 소요된 시간 (Elapsed Time)이고, T_p 는 같은 모델을 p개의 CPU로 계산했을 때 소요된 계산시간이다.

4.2 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-Up을 측정하기 위해서는 다른 사용자의 방해 없이 수백 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. 장착된 프로세서는 2.2Ghz, 2.4Ghz, 2.8Ghz, 3.0Ghz 이나 본 연구에서는 2.2 Ghz CPU만 사용하여 병렬성능을 측정하였다. 또한, 각 계산 노드는 3~6 GBytes의 메모리와 80~200 GBytes의 하드디스크를 가지고 있으며, Gigabit 네트워크로 연결되어 있다.

대부분의 병렬 코드는 MPI(Message Passing Interface)를 사용하고 있다.[13] 대표적으로 많이

사용되는 LAM/MPI와 MPICH의 통신성능에 대한 테스트 결과, LAM/MPI의 성능이 좀 더 우수한 것으로 나타났기 때문에, LAM/MPI를 통신을 위한 표준 라이브러리로 사용하였다. PEGASUS 시스템의 자세한 성능 최적화 과정은 참고문헌[4]에 기술하였다.

4.3 병렬 성능 평가 예제

구현된 기법의 정확성 및 병렬성능 평가 위한 예제는 3차원 Bar Impact 예제이다. Fig. 2에서 보는 바와 같이 1/4 대칭 모델에 대해 크기는 32 x 10 x 10(mm x mm x mm) 이다. 검증을 위해 메쉬 크기를 1 mm로 하였고 전체 요소 수는 3,200개이다. 재료는 32 x 3 x 3 만큼 초기 충전되어 있다. 요소의 최 외곽면의 절점에서는 벽면에 수직방향으로는 변형이 없고, 재료가 미끄러지는 조건이다. 충돌 속도는 200 m/sec, 종료시간은 80 μs이다. 재료모델은 선형 등방 경화 모델을 사용하였고, 탄성계수 100 GPa, 포아송 비 0.3, 초기 항복응력 400MPa, 소성 구간 기울기 100 MPa, 밀도는 8,930 kg/m^3 이다.

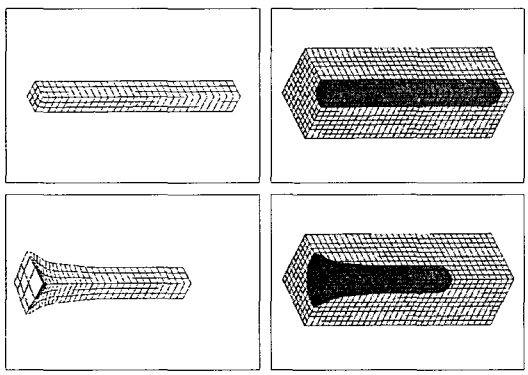

Fig. 2(a) Simulation results using IPSAP/Explicit at 0, 80μs : Lagrangian (Left), Eulerian (Right)

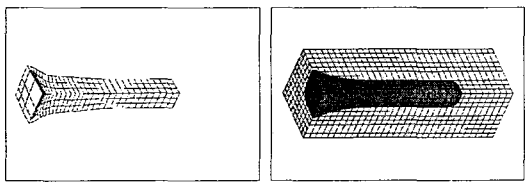

Fig. 2(b) Simulation result using LS-DYNA at 80 μs : Lagrangian (Left), Eulerian (Right)

순수 Lagrangian 코드 결과와, Eulerian 코드 결과를 Table 2와 Fig. 2에 LS-DYNA [14]의 결과와 함께 비교하여 나타내었다. 변형양상과 결과가 잘 일치하고 있다.

4.4 병렬 성능 결과

Eulerian 코드의 검증에 사용되었던 사각막대의 3차원 충돌 예제에 대해 병렬 성능을 측정하였다. 모델의 형상 및 크기, 물성치, 충돌속도 등은 동일하게 유지하되, 다만 대규모 모델에 대해 병렬효율을 평가하기 위해 요소개수를 I, J, K 방향으로 각각 1024개, 20개, 20개로 증가시켰다. 전체 요소 수는 409,600, 절점 수는 452,025, 자유도 수는 1,356,075 이다. 계산종료 시간은 10 μs까지 하였다. 영역분할은 통신량을 일정하게 유지하기 위해 충돌방향으로만 분할하였다. 64개 Xeon 2.2 Ghz CPU에서 2CPU/Node를 사용하였다.

Table 3에 CPU 증가에 따른 각 계산 부분의 시간 비중을 비교하였다. Remap 단계가 통신시간까지 포함하면 전체 계산시간의 90% 가까이 차지하고 있음을 알 수 있다. 이 부분의 계산 시간을 최대한 줄이는 것이 계산시간 감소에 가장 중요한 요인이 되며, 전체 병렬효율 또한 Remap 간계의 병렬효율을 따라가므로, 병렬 성능향상을 위해서는 Remap 단계의 병렬효율 및 통신시간을 최소화 하는 것이 관건이라 할 수 있다. 통신시간은 내력벡터 통신의 경우 사용 프로세서가 증가하면 점차 증가하는 양상을 보이나 Remap 변수 통신시간은 거의 변화가 없다. 또한 Remap 변수의 통신이 내력벡터의 통신시간에 비해 훨씬 많은 시간을 필요로 하고 있음을 알 수 있다.

내력벡터 계산하는 부분과 Remap 부분의 Speed-Up을 비교하기 위해 Table 3의 내용 중 두 부분의 Speed-Up을 Table 4에 비교하였다. Remap

Table 2 Comparison of deformation results between the pure Lagrangian and Two-step Eulerian codes for IPSAP/Explicit and LS/DYNA

Deformed value	IPSAP/Explicit		LS-DYNA	
	Lag	Eul	Lag	Eul
Length (mm)	21.9	21.5	21.8	21.0
Width (mm)	6.12	6.1	6.0	5.6

부분 보다는 내력벡터 계산의 병렬 효율이 좋지가 않은데, 이는 내력벡터 계산 시 요소가 void 인 경우는 계산을 하지 않고, 넘어가도록 했기 때문이다. 즉 Void 요소를 많이 가지고 있는 프로세서는 계산이 일찍 끝날 것이고, 반대인 경우는 늦게 끝난다. 반면 Remap 단계에서는 void 부분을 체적유동과 재료 유동을 계산한 후에 결과로 나오는 것이므로, 최소한 이 부분은 확인하도록 하였다.

Table 5에 병렬 환경에서 계산 시간 결과를 LS-DYNA와 비교하였다. LS-DYNA의 경우, 요소 중심 값 이류 시에는 IPSAP/Explicit과 동일하게 donor cell 기법을 사용했다. 그러나 절점 속도 이류 시에는, IPSAP/Explicit 에 적용된 SALE 기법을 제공하지 않으므로, 대신 1차 정확도의 HIS 기법[15]을 사용했다.

IPSAP/Explicit 에서는 총 스텝수가 1,500이 나왔고, LS-DYNA의 경우 1,505 가 나왔다. 두 코드에서 계산 스텝 수가 거의 비슷하나, 좀 더 객관적 평가를 위해 Elapsed Time을 총 요소수와 시간스텝수로 나눈 Grind time 으로 Speed-Up을 비교하였다. Grind time은 LS-DYNA의 경우 'clock time per cycle' 에 해당한다. Grind time과 elapsed time 은 'd3hsp' 파일에서 제공되는 'clock time per cycle' 과 'Total clock time'에서 'initialization time'을 제외하고 산출하였다. LS-DYNA의 경우 초기화시간은 10% 미만이고, IPSAP/Explicit는 이보다 훨씬 작다. IPSAP/Explicit이 double precision 이므로 LS-DYNA double precision과 비교하였다.

Speed-Up은 4 CPU 까지는 LS-DYNA가 조금 좋게 나타나나 그 보다 많은 CPU를 사용한 경우 IPSAP/Explicit이 더 나은 성능을 보이고 있다. 사용 CPU가 증가할수록 병렬 성능이 좋아지는 것은 Lagrangian 코드에서와 유사한 경향이다.

Elapsed time을 보면 IPSAP/Explicit이 2~3 배 정도 작게 나오고 있다. 알고리듬이 자세히 공개되어있더라도 개발자의 프로그램밍 기법에 따라 계산 성능의 차이가 발생할 수 있으며, LS-DYNA의 경우는 공개 되어있지도 않다. 다만 추측컨대, LS-DYNA의 경우 계산 cost가 상대적으로 비싼 HIS기법을 사용하고 있으며, 다중 물질(Multi-material)을 다루기 때문에, 계산 량 및 통신해야 할 변수가 그만큼 증가하게 된다. IPSAP/Explicit의 경우 현재까지는 단일물질과 void 만을 가정하기 때문에 그 만큼 처리해야 할 변수가 작다.

Table 3 Elapsed time and communication time of Lagrangian and Remap steps

NCPU	Lagrangian step		Remap step		Time Integ.	Total Elapsed Time
	Internal Force	Internal Force Comm.	Remap	Remap Comm.		
1	3.230e+02	0.0	4.062e+03	0.0	3.257e+02	4.666e+03
2	1.957e+02	6.773e-01	3.034e+03	4.408e+01	3.482e+02	3.589e+03
4	1.056e+02	1.191e+00	1.502e+03	5.873e+01	1.842e+02	1.804e+03
8	6.301e+01	1.381e+00	6.829e+02	6.038e+01	1.046e+02	8.591e+02
16	3.899e+01	1.505e+00	3.555e+02	5.639e+01	6.226e+01	4.630e+02
32	2.625e+01	1.642e+00	2.172e+02	5.762e+01	4.103e+01	2.902e+02
64	1.910e+01	1.820e+00	1.282e+02	5.736e+01	3.010e+01	1.840e+02

Table 4 Speed-Up of Internal force and Remap

NCPU	Lag. step	Remap step	Total
	Int. Force	Remap	
1	1.00	1.00	1.00
2	1.65	1.34	1.30
4	3.06	2.70	2.58
8	5.13	5.95	5.42
16	8.28	11.4	10.1
32	12.3	18.7	16.1
64	16.9	31.7	25.3

Table 5 Comparison of Elapsed time and speed-up between IPSAP/Explicit and LS-DYNA for Eulerian Scheme

NCPU	IPSAP/Explicit			LS-DYNA_double		
	Elapsed Time (sec)	Grind Time (nsec)*	Speed Up (grind time)	Elapsed Time (sec)	Grind Time (nsec)*	Speed Up (grind time)
1	4.666e+03	7.584e+03	1.00	7.698e+03	1.222e+04	1.00
2	3.589e+03	5.842e+03	1.30	5,240e+03	8.171e+03	1.50
4	1.804e+03	2.936e+03	2.58	2.533e+04	3.887e+03	3.14
8	8.596e+02	1.399e+03	5.42	1.722e+04	2.606e+03	4.69
16	4.630e+02	7.536e+02	10.1	1.343e+03	1.988e+03	6.15
32	2.902e+02	4.723e+02	16.1	6.956e+02	9.387e+02	13.0
64	1.840e+02	2.995e+02	25.3	5.610e+02	7.048e+02	17.3

* clock time per zone cycle

IPSAP/Explicit의 경우 Remap 단계가 전체 계산 시간의 약 90% 차지하는데, LS-DYNA의 경우 Lagrangian 코드는 IPSAP/Explicit와 크게 차이 없으나, Eulerian 코드에서는 LS-DYNA가 훨씬 많은 시간을 차지하는 것으로 나타났다. 따라서 LS-DYNA의 경우 Remap 단계 전체계산시간의 90% 이상의 많은 계산 비중을 차지한다.

5. 결 론

Two-step Eulerian 기반의 고속충돌해석 코드에 대한 병렬화 방안을 제시하고, 리눅스 클러스터 환경에서 병렬성능을 연구하였다. Lagrangian 단계 및 Remap 단계로 이루어진 코드를 각 단계에 대해 병렬화를 이루었고 각 계산 단계 별 계산시간을 분석하였다. Eulerian 기법 적용 시 메쉬구조는 주로 육면체 형상으로 구성된다는 점에 착안하여, 영역 분할 방식 및 통신 방식을 효율적으로 되도록 하였고, 각 부 영역의 경계면에 가상 요소를 두어 경계면에 있는 요소의 특성 값 송수신 및 이류 계산에 사용하도록 하였다. 적용된 병렬기법의 효율성을 평가하기 위해 상용 코드의 병렬 성능과 비교하였다. 적용 알고리듬과 코드 구조가 다를 것이므로 단순 비교에는 무리가 있지만, 속도향상 측면만 본다면, 본 연구를 통해 구현된 기법에서 사용 프로세서가 증가 할수록 더 좋은 병렬 효율을 보이는 것으로 나타났다. Eulerian 계산의 각 단계별 시간을 비교한 결과 Remap 단계의 계산 시간이 Lagrangian 단계의 계산 시간에 비해 매우 큰 비중을 차지하는 것으로 나타나, 이 부분의 효율 향상이 전체 효율 향상에 가장 큰 영향을 미치는 것으로 분석되었다.

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., 1998, "A Parallel Contact Detection Algorithm for Transient Solid Dynamics Simulation Using PRONTO3D," *Computational Mechanics* Vol. 22, pp. 143~159.

(2) Brown, K., Attaway, S. W, Plimpton, S., Hendrickson, B 2000, "Parallel Strategies for Crash and Impact Simulations," *Comput. Methods Appl. Engng* Vol. 184 pp. 375~390.

(3) McGlaun, J. M. and Thompson, S. L., 1990, "CTH: A Three-Dimensional Shock Wave Physics Code," *Int. J. Impact Engng,* Vol. 10, pp. 351~360.

(4) Paik, S. H. Moon, J. J, Kim, S. J and Lee, M., 2005, "Parallel Performance of Large-Scale Impact Problem in Linux Cluster Super Computer," *Computers & Structures,* Vol. 84, No. 10-11, pp. 732~741.

(5) Paik, S. H., Kim, S. J., 2005, "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, pp. 5~13.

(6) Benson, D. J. 1995, "A Multi-Material Eulerian Formulation for the Efficient Solution of Impact and Penetration Problems," *Computational Mechanics,* Vol. 15, No. 6, pp. 558~571.

(7) Noh, W. F. and Woodward, P., 1977, "SLIC (Simple Line Interface Calculation)," Lecture Notes in Physics 59, Springer Verlag.

(8) Van Leer, B., 1977, "Towards the Ultimate Conservative Difference Scheme, IV. A New Approach to Numerical Convection," *J. Comp. Phys.* Vol. 23, pp. 276~299.

(9) Amdsden, A. A, Ruppel, H. M and Hirt, C. W., 1980, "SALE: A Simplified ALE Computer Program for Fluid Flow at All Speeds," Los Alamos Scientific Laboratory.

(10) Paik, S. H., Kim, S. J., Lee, M. H., 2006, "Numerical Simulation of Impact and Dynamic Deformation Based on Two-Step Eulerian Method," *Journal of the Korean Society for Aeronautical and Space Sciences,* Vol. 34, No. 8. pp. 47~54.

(11) Goedecker S. and Hoisie A., 2001, Performance Optimization of Numerically Intensive Codes, SIAM.

(12) Paik, S. H., Kim, S. J., Lee, M. H., 2006,

"Parallel Computing of Large Scale FE Model Based on Explicit Lagrangian FEM," *Journal of the Korean Society for Aeronautical and Space Sciences,* Vol. 34, No. 8. pp. 33~40.

(13) Malard J. 1996, MPI: A Message-Passing Interface Standard. History, Overview and Current Status. Technology Watch Report, Edinburgh Parallel Computing Center.

(14) Hallquist J. O., 1991, ,LS-DYNA3D Theoretical Manual, Livermore Software Technology Corporation.

(15) Benson, D. J., 1992, "Momentum Advection on a Staggered Mesh," *J. of Computational Physics,* Vol. 100, pp. 143~162.