論文 J. of The Korean Society for Aeronautical and Space Sciences 42(5), 361-367(2014)
DOI:http://dx.doi.org/10.5139/JKSAS.2014.42.5.361

유한요소 구조해석 다중쓰레드 병렬 선형해법의 스케쥴링 및 부하 조절 기법 연구

김민기*, 김승조

Scheduling and Load Balancing Methods of Multithread Parallel Linear Solver of Finite Element Structural Analysis

Min Ki Kim* and Seung Jo Kim
Korea Aerospace Research Institute

ABSTRACT

In this paper, task scheduling and load balancing methods of multifrontal solution methods of finite element structural analysis in a modern multicore machine are introduced. Many structural analysis problems have generally irregular grid and many kinds of properties and materials. These irregularities and heterogeneities lead to bottleneck of parallelization and cause idle time to analysis. Therefore, task scheduling and load balancing are desired to reduce inefficiency. Several kinds of multithreaded parallelization methods are presented and comparison between static and dynamic task scheduling are shown. To reduce the idle time caused by irregular partitioned subdomains, computational load balancing methods, Balancing all tasks and minmax task pairing balancing, are invented. Theoretical and actual elapsed time are shown and the reason of their performance gap are discussed.

초 록

본 논문은 최근에 널리 사용되는 다중코어 컴퓨팅 환경에서 병렬 다중프론트 해법의 스케쥴링 및 부하조절 기법에 대해 논의한다. 통상적으로 구조해석 문제들은 불규칙한 격자계와 혼재된 물성 때문에 병렬화 알고리즘 구현 시 병목현상을 일으키고 불필요한 유휴시간을 초래한다. 따라서 이를 극복하며 효율성을 향상시키기 위해 다중쓰레드 기반 환경에 걸맞는 작업 스케쥴링 및 부하 분산 기법의 적용이 필수적이다. 본 논문에서 제시된 정적, 동적 스케쥴링 기법과 정적 전 임무 분산, 최소최대 임무 결합 등의 부하 분산 기법들에 대한 이론적, 실제 결과를 제시함으로서 그 유용성을 논의하고자 한다.

Key Words : Multicore Computer(다중코어 컴퓨터), Multithread Paralllelization(다중쓰레드 병렬화), Multifrontal Method(다중프론트 방법), Task Scheduling(작업 스케쥴링), Load Balancing(부하 조절)

† Received: November 25, 2013 Accepted: March 31, 2014 http://journal.ksas.or.kr/
* Corresponding author, E-mail : mkkim12@kari.re.kr
pISSN 1225-1348 / eISSN 2287-6871

Ⅰ. 서 론

다중코어 환경의 병렬 컴퓨팅은 최근의 전산 추세의 변화와 함께 다양한 분야에서 진지하게 논의되는 부분이다. 기존에는 단일 컴퓨터의 효율성 향상과 분산 컴퓨팅의 병렬성 향상에 주로 초점이 맞추어져 있다면 현재는 다중코어 컴퓨터의 성능을 최대로 이끌어 낼 수 있는 새로운 형태의 알고리즘이 필요하다. 이를 위해서 다양한 분야의 응용분야마다 그에 맞는 새로운 형태의 이론들이 제시되고 있으며 유한요소 구조해석 분야도 이에 맞추어 성능 향상과 정확도 향상을 위해 노력하는 추세이다.

유한요소 구조해석에서 가장 큰 계산을 차지하는 선형 연립방정식 해법의 병렬화를 위해 여러 이론들이 제시되었다. 이 중에서 다중프론트 해법(multifrontal method)[1,2]은 제한된 자원에서도 문제를 해결할 수 있고 자체적으로 병렬성을 내장하고 있기에 현재 상당수의 상용해석코드를 포함한 유한요소 해석 소프트웨어에서 기본적으로 선택한 해법이다. 다중프론트 해법은 전체 문제 영역을 다수의 작은 부영역(Subdomain)으로 분할한 후에 각 부영역의 내부 자유도를 인접한 경계 자유도로 치환하는 과정을 단계에 따라 거듭하며 최종적인 미지수를 소거하는 방식이다. 여기서 각 부영역의 내부 자유도 치환 과정은 다른 부영역의 계산과는 무관하게 독립적으로 이루어 질 수 있으므로 여타 직접 연립방정식 해법에 비해 상대적으로 병렬화에 유리하다. 그리고 강성행렬 분해(factorization)가 완료된 자유도의 행렬 성분들은 이후의 분해 과정에는 전혀 영향을 미치지 않으므로 이들을 보조기억장치 등에 저장하였다가 삼각해법(Triangular solve)과정에서 다시 적재하여 쓸 수 있으므로 제한된 컴퓨팅 자원 하에서도 비교적 큰 문제를 해석할 수 있다는 장점이 있다.

다중프론트 해법의 병렬화를 위해 여러 종류의 알고리즘이 고안되었고 이 중 상당수가 소스 공개[3,4]로 배포되고 있다. 유한요소 구조해석에 적용한 것으로는 NASTRAN, ABAQUS, ANSYS 등의 상용해석코드들에서 다중프론트 해법을 채택하였고, 국내에서 개발한 유한요소 해석 소프트웨어 IPSAP에서 이 해법에 기반하여 좋은 병렬성을 보여준 바 있다.

Ⅱ. 본 론

2.1 병렬 다중프론트 해법 개요

김정호 등[5], 김승조 등[6]의 연구에서 분산된 컴퓨팅 환경의 다중프론트 해법의 병렬화에 대해서 자세히 논의한 바 있다. 해당 연구들에서 최대 256대의 계산 컴퓨터를 활용하여 직접해법임을 감안한 상당히 높은 수준의 병렬성을 달성할 수 있다는 점을 보여준 바 있다. 하지만 해당 연구는 분산 병렬(distributed parallelism) 환경의 다중프론트 해법 병렬화를 다루고 있기에 다중코어 환경의 최근 컴퓨팅 환경에 그대로 적용하기에는 필요한 계산자원 측면에서 최선의 방법이라고 할 수 없다.

본 논문에는 다중코어 환경에 적합한 다중쓰레드 기반 병렬 해법에 대한 내용을 주로 소개한다. 다중쓰레드 병렬 알고리즘은 김민기 등[7,8,9]의 연구에서 소개된 바 있다. 요지는 분산 병렬 해법과 유사하게 프로세서 개수에 맞게 분할된 부영역들의 계산을 각각의 프로세서에 맞기고, 인접 경계 간 영역은 해당 프로세서 그룹에서 BLAS와 LAPACK 등의 수치연산 라이브러리를 이용하여 처리하는 방법이다. 현재 대부분의 하드웨어 제조사에서 그들의 CPU에 최적화되고 병렬화된 BLAS와 LAPACK을 제공하므로 이것을 그대로 활용할 수 있다. 분산 병렬 해법과 다르게 프로세서들이 공통된 메모리 주소공간을 공유할 수 있으므로 별도의 통신이 필요하지 않다는 장점이 있다. 본 연구에서는 슈퍼컴퓨터를 포함한 대다수의 계산용 컴퓨터의 CPU인 인텔 x86계열에 최적화된 수치연산 라이브러리인 인텔 MKL[10]을 사용하여 프로그램을 개발하였다. Fig. 1과 2는 4개로 균일하게 분할된 부영역들을 가진 문제의 미지수 소거 순서(elimination order)에 따른 다중쓰레드 미지수 소거 트리(elimination tree)와 계산 흐름 각각 나타낸 Fig.들이다.

분산 병렬 환경에서 각 컴퓨터들이 처리하는 프로세스(process)를 기본 계산 단위라고 한다면 다중코어 컴퓨터의 공유메모리 환경에서 각 프로세서가 맡은 작업 처리 단위를 쓰레드(thread)라고 하고 이것이 공유메모리 기반 병렬화의 기본적인 계산 처리 단위가 된다. 다수의 쓰레드들을 생성하여 병렬계산을 처리하는 기법을 다중쓰레드 병렬화(multithread parallelism)이라고 하며 쓰레드들이 같은 메모리 주소공간을 가지기에 분산 병렬화에 비해 개발이 쉽고 통신이 필요하지 않다는 장점이 있지만 각 쓰레드들의 작업 시간 조절과 공유 자원 분배에 많은 주의가 필요하다. 유한요소 구조해석을 포함한 수치연산 분야에서 가장 쉽고 널리 사용되는 다중쓰레드 병렬 개발에는 OPENMP[11]가 있다.

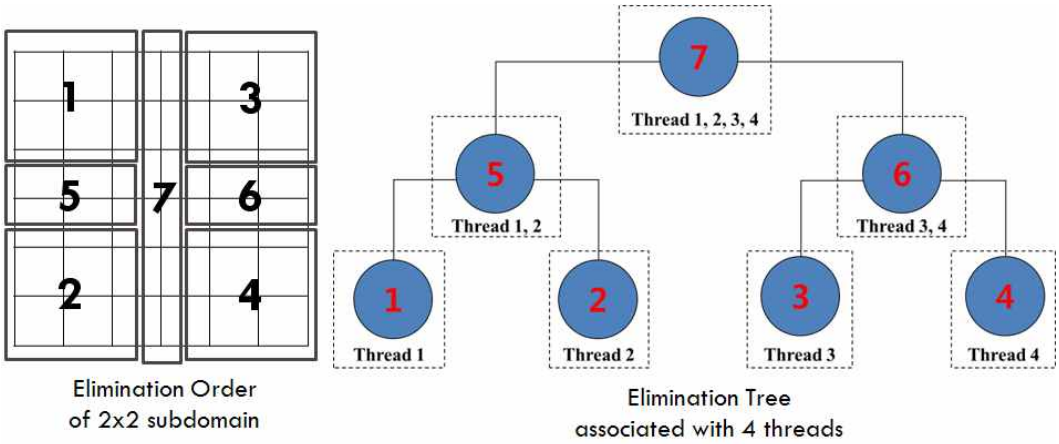

Fig. 1. Elimination Tree of multithreaded parallel factorization in four subdomain problem

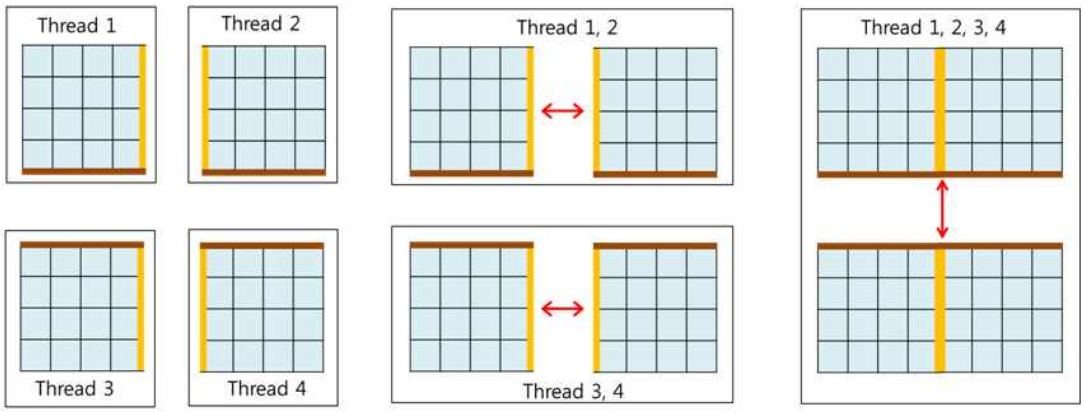

Fig. 2. Flow of the multithreaded parallel factorization of four subdomain

OPENMP는 반복 루프의 상단에 지정 키워드를 추가하는 방식으로 OPENMP 지원 컴파일러가 자동으로 병렬화 루프를 생성하므로 적은 노력으로도 병렬화 코드를 개발할 수 있기에 가장 널리 사용되는 개발 방법이다. 그리고 상용화된 쓰레드 병렬 라이브러리로 TBB[12], Cilk[13] 등이 있다. 그렇지만 위와 같은 방법들은 빠르고 쉬운 병렬 개발이라는 장점이 있으나 각 쓰레드들의 작업 순위 조절 등이 쉽지 않아 본 연구에서는 운영체제가 제공하는 Win32, POSIX의 기본 쓰레드 라이브러리를 사용하여 윈도우와 리눅스, 유닉스 환경에서 소스코드 호환성을 갖도록 개발하였다.

2.2 다중쓰레드 환경 작업 스케줄링

다중 프론트 해법에서 각 부영역의 계산 실행 시간을 조절하는 방법을 작업 스케줄링이라고 한다. Fig. 1의 미지수 소거 트리 구조에서 상위 순위(level)의 계산으로 진행하는 데에는 크게 두 가지 방법이 있다.

가장 쉽고 현재도 널리 일반적으로 사용되는 방법은 정적 스케줄링(static scheduling)이다. 통상적으로 다중쓰레드 병렬화에 많이 사용되는 기법인 OPENMP 및 TBB의 경우 각 병렬 루프의 실행이 모든 프로세서에서 종료된 후에 다음 단계로 진행한다. 이 방법은 구현이 쉽고 개발비용이 낮다는 장점이 있으나 각 루프의 계산량이 동일하지 않을 경우 발생하는 유휴시간(idle time)을 줄이기 어렵다.

반면 각 계산을 수행하는 쓰레드들을 계산 종료 후에 바로 다음 단계의 계산을 수행하게 하면 유휴시간을 크게 줄일 수 있다. 상위 영역 경계 자유도 소거도 내부 자유도 소거와 비슷하게 인접 영역 경계 자유도와 무관하게 독립적으로 수행할 수 있다. 이와 같이 각 루프들의 계산 종속성에 따라서 선행 루프의 계산 결과가 끝나면 그 다음 작업을 바로 수행하는 방식을 동적 스케줄링(dynamic scheduling)이라고 한다.

이러한 동적 스케줄링은 각 계산 작업 간 의

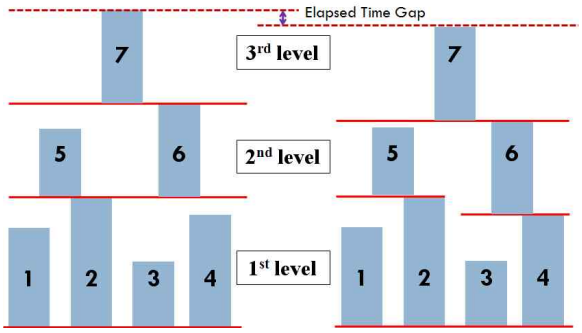

Fig. 3. The diagram of elapsed time of static(left) and dynamic(right) scheduling

존성을 철저히 파악하여 해당 작업 수행 전에 필요한 작업들이 끝나는 시점에 바로 작업을 시작하므로 복잡한 쓰레드 제어가 필요하기에 OPENMP 등으로는 구현이 쉽지 않고 오히려 운영체제 제공 쓰레드로 구현을 해야 한다. Fig. 3은 정적, 동적 스케쥴링을 나타낸 그림으로서 동적 스케쥴링이 유휴시간을 줄이는 데 큰 역할을 함을 알 수 있다. Fig. 3의 왼쪽에 있는 정적 스케쥴링에 따른 계산은 첫 번째 순위의 1~4번의 4개의 작업이 모두 종료된 후에 상위 순위의 계산인 5, 6으로 진행된다. 그러나 오른쪽의 동적 스케쥴링의 경우 1, 2번 작업 종료 직후 5번 작업이 수행되고, 3, 4번 작업 종료 후 곧바로 6번 작업이 수행되며 이러한 일은 각 소거 트리마다 독립적으로 이루어진다. 특히 동적 스케쥴링은 정적 스케쥴링에 비해 각 쓰레드 별로 할당된 작업량의 불균일성이 심할 때 강점을 발휘할 수 있다. 대다수의 현업에서 사용되는 구조해석 문제는 복잡한 형상으로 인한 작업량의 불균일성을 감안할 때, 동적 스케쥴링 방식이 큰 효력을 발휘한다는 점을 알 수 있다.

정적 및 동적 스케쥴링 시 계산 시간은 아래 식 (1)과 (2)처럼 표현할 수 있다. 여기서 이 식들의 상첨자는 계산에 참여하는 프로세서 개수, 하첨자 i, j는 i번 레벨의 j번 작업을 의미한다. 정적 스케쥴링 기법은 각 레벨의 모든 작업이 종료된 후에 계산이 진행되므로 식 (1)처럼 각 레벨의 최대 시간의 총합이 전체 계산 시간이 된다. 동적 스케쥴링은 각 트리의 작업 노드에서 서브레벨 단계에서 재귀적으로 식 (2)처럼 정의할 수 있다.

$$T_s = \sum_{i=1}^{L} \max_{1 \le j \le 2^{L-i}} \left[T_{i,j}^{(2^{i-1})} \right] \qquad (1)$$

$$T_d = T_{L,1}^{d(2^{L-1})} \qquad (2)$$

$$T_{i,j}^{d(p)} = T_{i,j}^{(p)} + \max_{2 \le i \le L} \left[T_{i-1,2j-1}^{d(p/2)}, T_{i-1,2j}^{d(p/2)} \right]$$

$$T_{1,k}^{d(1)} = T_{1,k}^{(1)}$$

2.3 다중쓰레드 환경 계산 부하 조절

다음으로 논의할 사항은 계산량 부하조절이다. 일반적으로 대다수의 구조해석 문제는 그 격자계의 불균일성으로 인해 분할된 각 부영역 역시 서로 다른 양의 계산량을 갖게 된다. 이러한 불균일성은 유휴시간을 야기시키며 이는 전체적인 계산 효율성을 저하시킨다. 따라서 작업 스케쥴링만으로는 이를 극복하는 것은 한계가 있으며 각 프로세서별로 할당된 계산량을 최대한 균등하게 맞추어 줄 필요가 있다.

첫 번째 방법은 각 계산단위에 할당된 부영역들의 내부 자유도 소거를 독립적으로 수행한 후에, 상위의 영역 경계 자유도 계산은 모든 프로세서들이 병렬로 수행하는 방법이 있다. 언급한 것처럼 BLAS, LAPACK 라이브러리는 병렬계산 수행이 가능하고, 상위 영역 경계 자유도 행렬은 비교적 큰 크기를 가지기에 이로서도 충분한 계산 효율을 기대할 수 있다. 이 방법을 사용하게 되면 상위의 경계 자유도 소거는 모든 프로세서마다 균등하게 나뉘게 된다. 이 방법을 정적 전 임무 분산(Static All Balancing)로 칭하기로 한다.

두 번째 방법은 최대최소 임무 결합(Min-Max Task Pairing)이다. 이 방법은 각 임무의 계산량을 미리 추정한 다음, 이를 그 크기순으로 정렬하여 최소와 최대를 일대일로 짝지음으로서 수행할 수 있다. 결합된 임무 쌍은 그 크기가 다른 쌍들과 비슷할 것으로 여길 수 있고 맺어진 각 쌍들은 다른 쌍들과 독립적으로 계산을 수행한다. 정적 전 임무 분산과 다른 점은 Fig. 1의 소거 트리 각 단계마다 임무 쌍이 짝지어지며 이들은 그 쌍에 할당된 프로세서들이 독립적으로 수행한다는 점이다. 이로서 정적 전 임무 분산에서 나올 수 있는 실제 병렬 계산 효율 저하를 줄일 수 있다.

Figure 4의 상부에 정적 전 임무 분산, 하부에 최대최소 임무 결합에 의한 작업 부하 조절이 나타나 있다. 정적 전 임무 분산은 4개의 프로세서가 병렬 연산을 1부터 4번 작업까지 순차적으로 수행한다. 최대최소 임무 결합은 각 작업의 크기 순으로 최대인 2와 최소인 3이 우선적으로 맺어지고, 그 다음 최대 1번과 최소 4번이 쌍으로 맺

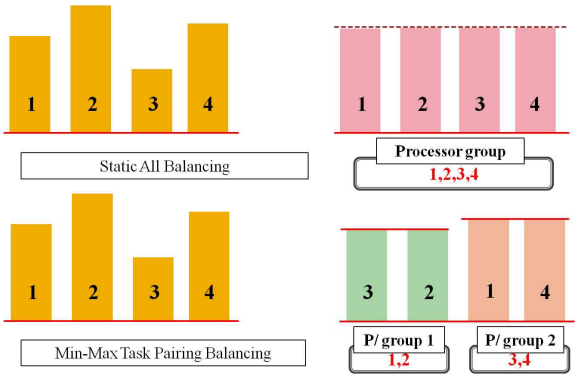

Fig. 4. Static all balancing (upper) and min-max task pairing

어진다. Fig. 4의 하단 우측에 보듯이 두 쌍들의 작업량 총합은 유사하므로 작업쌍들의 총 개수만큼의 프로세서 그룹을 조성하여 각 그룹별로 작업쌍 처리를 할당시킬 수 있다.

4개의 부영역을 가진 문제를 4개의 프로세서로 해석 시에 두 개의 쌍들은 두 프로세서가 두 개의 그룹을 이루어서 각 그룹은 한 쌍의 두 개의 작업들을 순차적으로 처리한다. 즉 전 임무 분산 대신 한 프로세서 그룹이 한 쌍으로 묶인 두 개의 작업을 처리한다고 볼 수 있다.

두 번째 방법을 확장하여 최하단의 부영역 계산 역시 이를 적용할 수 있으며 이를 확장된 최대최소 임무 결합(Extended Min-Max Pairing)이라고 한다. 이를 본문의 예시에 적용하면, 4개 대신 8개의 최하단 부영역 개수를 사용하여 이들 부영역들도 최대최소 쌍으로 맺어서 각 프로세서에 작업을 할당하여 계산을 진행시킬 수 있다.

2.2절의 작업 스케쥴링과 본 절의 작업량 부하 조절은 서로 독립적인 개념이기에 결합이 가능하다. 정적 전 임무 분산과 최대최소 임무 결합과 동적 스케쥴링을 적용 시 계산 시간은 각각 아래 식 (3)과 (4)와 같다. 식에 사용되는 상, 하첨자의 의미는 식 (1), (2)에 사용된 것과 그 의미가 같다. 정적 전 임무 분산은 (3)과 같이 각 레벨의 모든 작업들을 전 프로세서가 동시에 처리하여 순차적으로 처리하고, 맨 하단은 각 프로세서가 자신의 부영역을 독자적으로 처리한다. 최대최소 임무 결합은 두 쌍의 프로세서 그룹이 최대최소로 묶인 두 작업 그룹을 독립적으로 처리하므로 식 (4)처럼 재귀적으로 풀이할 수 있다.

$$T_{sb} = \sum_{i=2}^{L}\sum_{j=1}^{2^{i-1}} T_{i,j}^{(2^{L-1})} + \max_{1 \le j \le 2^{L-1}}\left[T_{1,j}^{(1)}\right] \qquad (3)$$

$$\begin{aligned} T_{tpd} &= T_{L,1}^{tpd(2^{L-1})} \\ T_{i,j}^{tpd(p)} &= T_{i,j}^{tpl(p)} + \max_k\left[T_{i-1,k}^{tpd(p/2)} | G_k \ni t_{2j1-1}, t_{2j1}, t_{2j2-1}, t_{2j2}\right] \\ T_{i,j}^{tpl(p)} &= \begin{cases} T_{L,1}^{(p)} & i = L \\ T_{i,j1}^{(2p)} + T_{i,j2}^{(2p)} & 2 \le i \le L-1 \end{cases} \\ T_{1,k}^{tpd(1)} &= T_{1,k}^{(1)} \qquad 1 \le k \le 2^{L-1} \end{aligned} \qquad (4)$$

2.4 작업 스케쥴링 및 부하 조절이 적용된 해석시간 이론적, 실측 결과

본문에서 제안한 스케쥴링 기법과 부하 분산 기법들을 적용하여 다섯 가지 구조해석 문제에 대해 이론적인 계산시간 추정치와 실제 계산 결과를 비교하였다. 계산에 사용된 문제는 Table 1에, 계산시간 추정치와 실제 결과를 각각 Fig. 5, 6에 정리하였다. 계산에는 인텔 Xeon E7-2830 2.13GHz 2개가 설치된 총 16개의 코어를 탑재한 컴퓨터를 활용하였으며 운영체제는 윈도우 서버 2008 R2 이다. Fig. 5는 식 (1)~(4)를 사용하여 각 계산 방법 별 계산시간 추정치이며, Fig. 6은 실측된 계산 시간을 비율로 나타낸 그림이다.

Figure 5에서 계산 시간 추정을 위해 MKL의 행렬 루틴의 병렬성능 추정을 아래와 같은 식을 적용하였다. 이상적인 병렬 효율성은 프로세서의 개수에 비례해야 하지만 실제로는 그렇지 않기에 적당한 근사식을 적용해야 한다. p개의 프로세서 사용 시 행렬연산 효율은 아래와 같다.

$$E_p = \alpha^{\log_2 n}, \qquad \alpha = 1.9 \qquad (5)$$

이는 p개의 프로세서와 $2p$개의 프로세서 사이에 아래와 같은 효율성 공식이 성립한다는 것을 의미한다.

$$E_{2p} = \alpha E_p \qquad (6)$$

식 (5)와 (6)은 2배의 프로세서가 계산에 참여할 때 성능 향상은 불과 1.9배 정도 이루어짐을

Table 1. Brief information of benchmark problems

Problem	No. of DOF	No. of Elements
Hexa 60x60x60	680,943	216,009
Quad 600x600	2,167,206	360,000
Wing Structure	3,214,406	767,302
Ship Hull	274,584	98,828
Blade 1.5M	1,522,146	253,440

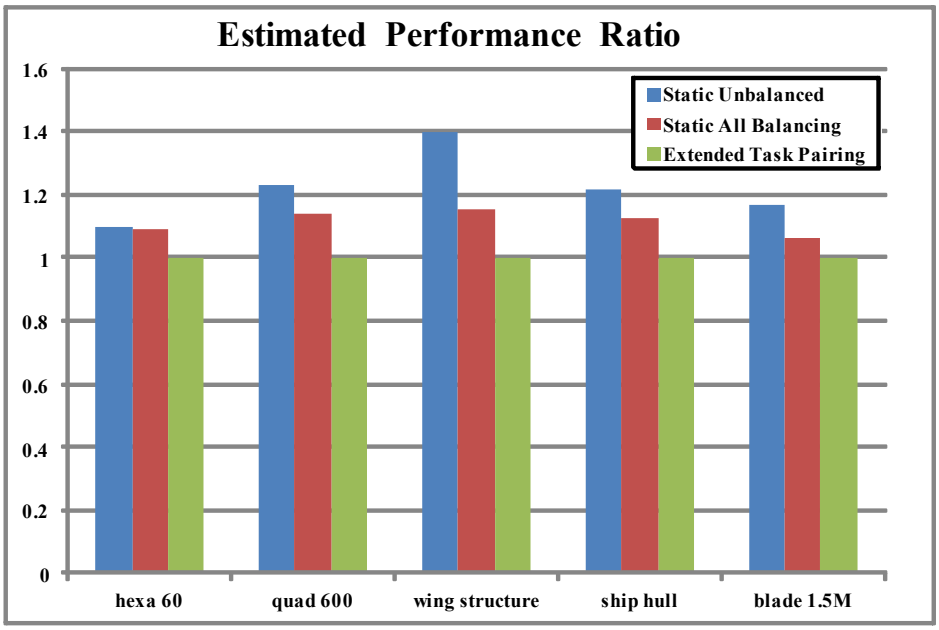

Fig. 5. Estimated computing time of load balancing and scheduling methods

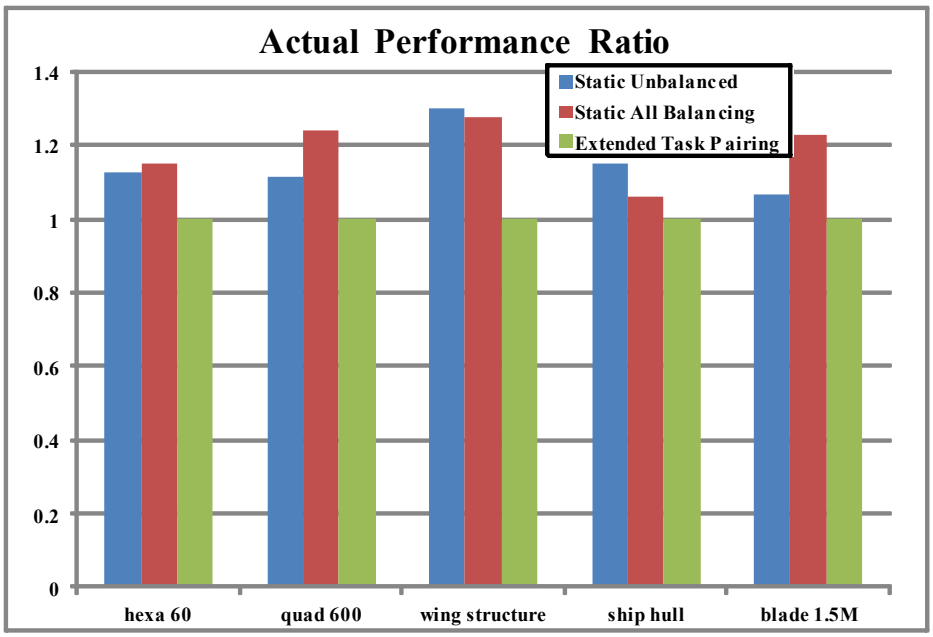

Fig. 6. Actual computing time of load balancing and scheduling methods

의미한다. 여기서 성능향상계수는 실제 하드웨어, 및 운영체제, 컴파일러, 그리고 다중쓰레드 라이브러리 등에 많은 영향을 받는다. 그리고 문제 크기와 종류에 따라서도 달라지기에 그 성능함수는 단순히 모델링할 수 없다. 본문에서는 식 (5), (6)의 로그스케일 단위의 성능함수를 활용하였다.

Figure 5와 6에서 파란 막대는 부하분산이 적용되지 않은 정적 스케쥴링, 빨간 막대는 정적 전 임무 분산, 녹색 막대는 확장된 최대최소 임무 결합 부하분산 동적 스케쥴링이다.

Figure 5와 6에서 보듯이 최대최소 임무 결합 부하분산 동적 스케쥴링이 추정치로도, 실제로도 가장 좋은 효율을 보임을 알 수 있다. 다만 이론적 계산 시간 추정치와 실측 계산시간에 다소 차이가 있음을 알 수 있다. 특히 계산시간 공식으로는 부하분산이 없는 정적 스케쥴링이 모든 문제에 대해 가장 느리지만, 실제로는 정적 전 임무 분산이 가장 효율이 좋지 않을 때가 많음을

알 수 있다. 정적 전 임무 분산은 각 단계의 작업들 하나하나를 모든 프로세서가 동시에 처리하고, 이들 다수의 작업들을 순차적으로 처리하기에 각 작업 크기가 충분히 크지 않으면 병렬 성능은 저하될 수밖에 없다. 다만 이를 사전에 정확히 추정하는 것은 극히 어려운데, 이는 각 부하 분산 방법의 계산 시간 추정치 공식에 따른 것으로 병렬 수치연산 시 성능 저하를 어느 정도로 감안하느냐에 크게 좌우된다. 본 논문에서는 식 (5)의 효율성 공식을 적용하였으나 실제로는 단순히 프로세서 개수뿐만 아니라 문제의 크기와 루틴의 종류에 따라서 달라질 수 있는 복잡한 관계에 있다. 행렬 크기와 연산의 종류, 쓰레드 수가 고려된 정확한 병렬효율성을 알아야 비교적 정확한 계산시간 추정이 가능하다.

그리고 하드웨어 제조사에서 제공하는 BLAS와 LAPACK의 병렬화 정도에도 그 결과가 달라진다. PLASMA[14]와 같은 다중쓰레드 전용 수치연산 라이브러리가 개발되어 있으나 실제 계산 결과 그 성능이 좋지 않아서 본지에는 수록하지 않았다. 그렇지만 제조사 제공 기본 라이브러리와 다중쓰레드 전용 라이브러리의 내부 구현을 고려한 심도 있는 추가 연구가 필요하다.

Ⅲ. 결 론

본 논문에서는 다중쓰레드 기반 다중프론트 해법 병렬화의 계산 성능 향상과 관련된 몇 가지 기법을 제시하였고 이론적 계산시간 추정치와 실측 계산 시간을 수록하였다. 이러한 작업 스케쥴링 및 부하 분산 방법들을 통해 계산 시간을 단축시켜 그 효율이 증가함을 확인할 수 있다.

후 기

본 연구는 한국항공우주연구원이 주관하는 “위성종합설계 S/W 기능고도화 사업”의 일환으로 수행되었음을 알려드립니다.

References

1) Duff, I. S., and Reid, J. K. "The Multifrontal Solution of Indefinite Sparse Symmetric Linear-Equations," *ACM Transactions on Mathematical Software,* Vol. 9, No. 3, 1983, pp. 302-325.

2) Irons, B. M. "A frontal solution program for finite element analysis," *International Journal for Numerical Methods in Engineering,* Vol. 2, No. 1, 1970, pp. 5-32.

3) P. R. Amestoy, I. S. Duff, J. Koster, and J. -Y. L'Excellent, "A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling," *SIAM Journal of Matrix Analysis and Applications*, Vol 23, No. 1, 2001, pp. 15-41.

4) T. A. Davis, "Algorithm 832 : UMFPACK, An Unsymmetric-Pattern Multifrontal Method," *ACM Transaction on Mathematical Software*, Vol 30, No. 2, 2004, pp. 196-199.

5) Kim, J.H., C.S. Lee, and S.J. Kim, “High-performance domainwise parallel direct solver for large-scale structural analysis,” *AIAA journal*, Vol. 43, 2005, pp. 662-670.

6) Kim, S.J., C.S. Lee, and J.H. Kim, “Large-scale structural analysis by parallel multifrontal solver through Internet-based personal computers,” *AIAA journal*, Vol. 40, 2002, pp. 359-367.

7) M.K. Kim, J.H. Kim, C.Y. Park and S.J. Kim, “Parallelization of Multifrontal Solution Method for Shared Memory Architecture", KSAS Journal, Vol. 40, No. 11, 2012, pp 972-978.

8) Kim, M.K, Kim, S.J, "Hybrid Parallelism of Multifrontal Linear Solution Algorithm with Out Of Core Capability for Finite Element Analysis", *CMES,* Vol. 84, 2012, pp. 297-331.

9) M.K. Kim, “Studies of Scheduling and Load Balancing Ways of the Parallel Direct Solution Method in Multicore Computers for Large Scale Structural Analysis", KSAS Autumn Conference, 2013, pp. 201-204.

10) http://software.intel.com/en-us/intel-mkl

11) http://openmp.org/wp/

12) https://www.threadingbuildingblocks.org/

13) http://software.intel.com/en-us/intel-cilk-plus

14) Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P., Tomov, S. "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *Journal of Physics: Conference Series*, Vol. 180, 2009.