

特輯論文

DOI: <http://dx.doi.org/10.5139/JKSAS.2012.40.11.972>

다중프론트 해법의 공유메모리 병렬화

김민기*, 김정호**, 박찬익***, 김승조****

Parallelization of Multifrontal Solution Method for Shared Memory Architecture

Min Ki Kim*, Jeong Ho Kim**, Chan Yik Park*** and Seung Jo Kim****

ABSTRACT

This paper discusses the parallelization of multifrontal solution method, widely used for finite element structural analyses, for a shared memory architecture. Multifrontal method is easier than other linear solution methods because the solution procedure implies that unknowns can be eliminated simultaneously. Two innovative ideas are introduced to achieve optimal solver performance on a shared memory computer. Those are pairing two frontal matrices and splitting the frontal matrix in order to reduce the temporal memory space required by independent computing tasks. Performance comparisons between original algorithm and proposed one prove that proposed method is more computationally efficient on current multicore machines.

초 록

본 논문은 유한요소 구조해석의 선형해법으로 널리 사용되는 다중프론트 해법의 공유메모리 환경하의 병렬화 방법을 논의한다. 다중프론트 해법은 병렬성이 내재되어 있어서 여타 해법보다 상대적으로 병렬화가 용이한 방법이다. 다중프론트 해법의 공유메모리 컴퓨터에서 최적의 성능을 내도록 병렬 계산을 수행하기 위한 기법들이 제시되었다. 주로 독립적인 계산 작업 시에 필요한 주 메모리 용량을 줄이는 데 초점을 맞춘 방법들로서 프론트 행렬 연성화와 행렬 분리로 명명된 두 기법에 대해 자세히 설명한다. 개발된 방법으로 기존의 알고리즘과의 성능 비교를 수행하여 본지에 제안한 방법이 현대의 다중코어 컴퓨터에서 훨씬 더 효율적인 기법임을 입증하였다.

Key Words : Multifrontal Method(다중프론트 해법), Parallelization for SMP(공유메모리 병렬화), Parallel Performance(병렬성능), Structural Analysis(유한요소 구조해석)

I. 서 론

다중프론트 해법은 유한요소 구조해석에 널리 사용되는 기법 중 하나이다. 선형 정적해석, 고유치 해석, 시간영역 해석, 비선형 해석 등에 발생하는 선형 방정식은 주로 대칭 양행렬이므로 이의 해법에 다중프론트 Cholesky 분해가 널리 사용된다. 컴퓨터의 발전과 함께 고정밀 해석의 요

† 2012년 8월 30일 접수 ~ 2012년 10월 4일 심사완료

* 정회원, 서울대학교 기계항공공학부 대학원
교신저자, E-mail : kimmk2004@gmail.com

** 정회원, 인하대학교 항공우주공학과

*** 정회원, 국방과학연구소

**** 정회원, 한국항공우주연구원
대전광역시 유성구 어은동

구가 커짐에 따라 구조해석 문제의 크기 또한 커지게 되고 이를 효과적으로 해석하기 위한 병렬화 기법들이 다양하게 등장하고 있다.

미국과 일본은 대규모 유한요소 구조해석을 위해 반복해법에 기반을 둔 SALINAS[1,2](미국)와 GeoFEM[3,4], Adventure[5,6](일본)을 연구한 바 있다. 반복해법은 보통 직접해법보다 병렬성이 뛰어나기에 대규모 문제의 병렬계산에 널리 활용되는 선형해석 기법이다. 그러나 반복해법은 해의 수렴성 문제 등으로 아직까지도 전산구조해석 분야에서는 직접해법이 더 선호되고 있다. 국내에서는 서울대에서 다중프론트 해법의 대규모 분산병렬화[7-9]에 대한 연구와 영역분할 기법과의 융합해법[10] 연구가 있다.

현대에 사용하는 많은 컴퓨터들은 여럿의 계산유닛을 장착한 다중코어 프로세서가 다수를 차지하고 있다. 따라서 이에 걸맞는 새로운 방식의 공유메모리 환경에 적합한 병렬 선형해법 알고리즘이 필요하다. 다중프론트 해법은 이미 다중 분산컴퓨팅을 전제하고 개발한 적이 있다[7,8]. 하지만 이 방법은 한 대의 컴퓨터에는 하나의 CPU를 가정하고 고안했기에 현대의 다중코어 컴퓨터에는 그 성능상의 한계가 분명하다. 한 대의 다중코어 컴퓨터를 여러 대의 단일 CPU 계산노드의 모임으로 생각하고 병렬계산을 할 수도 있으나 이는 주 메모리 사용량 및 계산효율 측면에서 권장할 방법은 아니다. 다만 다중프론트 해법을 포함한 다수의 희소행렬 해법에 사용되는 BLAS나 LAPACK등의 수치연산 루틴들 내부에 병렬성이 내장되어 있어서 실제 계산을 수행하면 이들로부터 파생되는 병렬성이 나타나기도 한다. 물론 이렇게 보이는 성능은 당연히 선형해법 자체 알고리즘을 병렬화했을 때보다 뒤쳐진다.

본 논문에는 현대의 다중코어 컴퓨터에 맞는 공유메모리 병렬화 기법을 다중프론트 선형해법에 적용하여 그 성능을 기존 알고리즘과 비교하여 그 성능과 유용성을 검증하였다.

II. 본 론

2.1 다중프론트 해법 소개

다중프론트 해법은 Duff 등[11]에 의해 소개된 방법으로서 Iron[12]이 소개한 단일 프론트 해법을 다수의 프론트 단위로 나눠 수행한다는 개념으로 계산량과 메모리 소요량 측면에서 보다 더 유리한 방법이다. 이를 유한요소법에 적용 시 조립이 완료된 미지수를 즉시 소거가 가능하며 소

거된 미지수는 보조기억 장치에 저장할 수 있기에 주 메모리가 제한된 환경에서 특히 장점을 발휘한다. Kim 등[7-9]은 유한요소법의 이러한 특성을 살려 다중프론트 해법 계산 과정 중에 조립을 수행하면서 동시에 소거를 진행하는 방식으로 구현하였다. 따라서 전역강성행렬을 명시적으로 조립할 필요가 없고 여기에 필요한 추가적인 계산과 저장공간 역시 불필요하기에 부수적인 장점을 얻을 수 있는 방법이다.

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \quad (1)$$

여기서 하첨자 1은 현재 단계에서 조립이 완료된 자유도이고 2는 자유도 그룹 u_1 과 연결되어서 아직 조립이 끝나지 않은 자유도이다. u_1 은 조립이 완료되었으므로 소거하여 u_2 항으로 표현할 수 있으며 이는 아래 식과 같다.

$$K = \begin{bmatrix} L_1 & 0 \\ \bar{K}_{12}^T & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \bar{K}_{22} \end{bmatrix} \begin{bmatrix} L_1^T & \bar{K}_{12} \\ 0 & I \end{bmatrix} \quad (2)$$

위 식은 유한요소 구조해석에서 주로 등장하는 대칭 양행렬에 적용되는 Cholesky 분해를 의미한다. 조립이 끝난 자유도 u_1 에 연관되는 L_1 과 \bar{K}_{12} 는 주 메모리 혹은 하드디스크에 저장되고 \bar{K}_{22} 는 임시로 주 메모리에 저장된다. 이 과정은 밀집한 행렬의 계산이므로 BLAS와 LAPACK으로 처리할 수 있다. 보통 BLAS와 LAPACK은 해당 컴퓨터의 아키텍처에 맞게 최적화되어 있으므로 위의 연산을 통해 최적의 계산성능을 이끌어 낼 수 있다.

다중프론트 해법은 위의 연산을 다음과 같이 처리한다. 전체 격자계 영역을 몇 개의 요소로 이루어진 작은 크기의 다수의 부영역(subdomain)들로 METIS[13]로 영역 분할 후, 그 결과로 전체 계산량과 메모리 소요량, 자유도 소거 순서를 결정하는 기호 분석 과정(Symbolic factorization)을 거친다. 다음에 기호 분석 결과를 바탕으로 실제 부동소수점 연산을 포함하는 수치적 분해(numerical factorization)을 수행한다. 마지막으로 분해된 행렬 데이터로 우변항(right hand side) 방정식을 푸는 전방 소거/후방 대입 과정을 통해 방정식의 해를 얻게 된다.

다중프론트 해법의 전체 계산량이 언급한 영역분할 과정에서 결정되는데, 이를 최적화하기 위해 여러 가지 방법이 제안되었다. 초창기에 Kim[9]이 유한요소의 연결 그래프와 가중치를

부과하는 방법을 제안했고, Kim[14]은 그것을 분할하는 과정을 병렬화한 알고리즘을 제안했다. 본 논문에는 Kim[14]의 요소 연결도 그래프의 영역분할 과정을 병렬화한 결과를 바탕으로 모든 논의를 진행한다.

2.2 다중프론트 해법의 공유메모리 병렬화

다중프론트 해법의 공유메모리 환경하의 병렬화는 여러 가지 방법으로 수행될 수 있다. 일반적으로 공유메모리 환경에서 병렬계산은 스레드(thread)라는 단위로 이루어진다. 이를 위해 가장 널리 사용되는 기법으로 OPENMP[15]를 들 수 있다. OPENMP는 공유메모리 머신에서 다중스레드 병렬화를 달성하기 위해 가장 흔하고 가장 쉽게 구현할 수 있는 방법이다. 이것은 컴파일러가 구현하는 특정 키워드(keyword)들을 병렬화하고자 하는 반복루프에 적용하여 손쉽게 다중스레드 병렬화를 구현할 수 있다. 이 외에 TBB[16]와 같은 외부 병렬 라이브러리를 이용하는 방법과 POSIX/Win32 Thread와 같은 운영체제가 제공하는 고유의 다중스레드 제어 함수를 활용하는 방법이 있다. 후자의 방법은 다수의 스레드들을 운영체제가 제어하는 다양한 방법들을 직접 활용할 수 있어 복잡한 스레드 스케줄링(scheduling) 제어에 적합한 방법이지만 그 사용법이 어렵다는 단점이 있다. 본 논문은 다중프론트 해법의 다중스레드 병렬화의 기본에 초점을 맞추고 있으므로 OPENMP로 병렬화한 결과를 언급하겠다.

다중프론트 해법은 각각의 부영역 혹은 그 영역 사이의 경계를 독립적인 단위로 분해할 수 있어서 병렬화에 굉장히 용이한 방법이다. 따라서 단일 스레드 알고리즘의 미지수 분해는 Fig. 1

처럼 해의 분해 절차에 서로의 어떠한 방해 없이도 동시에 수행 가능하다. Fig. 1은 2x2로 규칙적으로 분할된 4개의 부영역들로 구성된 문제를 4개의 프로세서로 계산할 때의 예제를 나타낸 그림이다. 좌측은 병렬화된 다중프론트 해법의 소거 순서의 트리 구조와 그에 해당하는 부영역과 미지수 소거 순서를 우측에 나타내었다.

Fig. 1의 우측에 보이는 부영역 1,2,3,4의 내부 자유도는 외부의 어떤 자유도와 무관하게 오직 그들 사이의 경계의 자유도와 연결되므로 이들 내부 자유도는 경계의 자유도로 식 (1), (2)의 과정으로 분해할 수 있다. 여기서 이 수치연산 과정은 외부의 자유도와는 어떠한 관련도 없으므로 각각 독자적으로 동시에 4개의 프로세서로 수행할 수 있다. 그리고 상위 단계의 자유도 그룹 5,6 역시 오직 자유도 그룹 7과 연결된 채로 서로와는 무관하므로 각각 2개의 프로세서로 분해가 가능하다. 마지막 단계에서 자유도 그룹7은 모든 프로세서가 계산에 참여하여 분해를 완료한다.

서론에서 언급한 것처럼 분해식 (1), (2)는 하드웨어 제조사에서 제공하는 최적화된 성능의 BLAS와 LAPACK에 의해 수행되므로 식 (1), (2)를 위해 별도의 병렬 알고리즘을 고안할 필요는 없다. 이는 분산컴퓨팅 병렬화 알고리즘[7]과 큰 차이를 보이는 요인 중 하나이며 실제로 이런 점 때문에 분산환경 병렬화보다 쉽게 공유메모리 병렬화를 구현할 수 있다.

분산컴퓨팅 환경의 병렬 알고리즘은 식 (1), (2)의 구현을 위해 분산된 행렬 데이터를 취급하는 독자적인 알고리즘과 PLASC(Parallel Linear Algebra Subroutines written in C)라고 명명한 자료구조를 고안해야 했지만 공유메모리 하에서는 단순히 병렬화된 수치연산 라이브러리를 링크

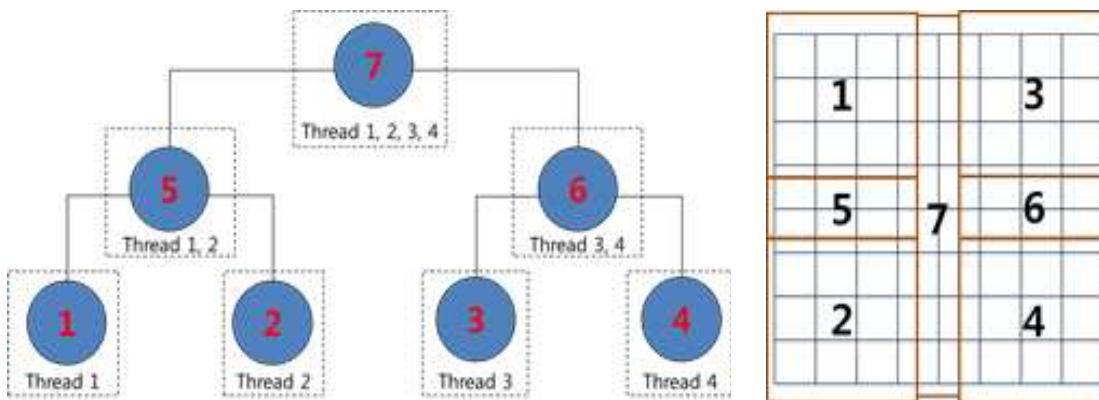


Fig. 1. 다중스레드 병렬 다중프론트 해법의 소거 단계 트리(왼쪽)와 그 부영역(오른쪽)

(link)해서 구현할 수 있다. 이외에도 분산컴퓨팅과 공유메모리 병렬화의 차이점은 공유메모리 환경에서는 정보를 주고받기 위한 통신이 불필요하며 각 쓰레드 간 필요한 정보는 전역적으로 접근 가능한 주 메모리를 통해 손쉽게 얻을 수 있다.

다중쓰레드 병렬화의 큰 단점 중 하나는 단일 쓰레드 알고리즘보다 많은 메모리가 필요하다는 점에 있다. 이는 각각의 분해 과정에서 각자의 임시 데이터 공간이 필요하다는 점에서 어느 정도는 불가피한 현상이다. 본 논문에는 이러한 저장공간의 크기를 줄일 수 있는 두 가지 기법을 소개한다.

2.3 프론트 행렬 연성화

첫째 개념은 프론트 행렬 연성화이다. 이 개념은 두 개의 독립적인 계산 단위(task)가 하나의 정사각형 꼴의 프론트 행렬의 위, 아래 부분을 각각 점유해서 사용하여 저장공간을 약 1/2 수준으로 줄인다는 것이다. 이를 위해 우선 대칭행렬의 연산 효율에 대해 살펴보아야 한다. 대칭행렬의 연산은 크게 두 가지 방법이 있는데, 첫째는 정사각형 꼴 행렬의 윗부분 혹은 아랫부분만을 계산에 적용하는 방법이 있고 다른 하나는 대칭행렬의 절반 부분을 삼각형으로 취급하여 하나의 긴 배열로 저장된 것을 계산하는 방법이 있다. 두 가지 모두 대칭행렬의 연산이지만 실제 계산 성능은 상이하다. 전자인 정사각형 행렬의 절반을 다루는 것이 비록 메모리 사용량 측면에서는 불리하지만 계산효율은 후자에 비해 월등하게 높다. 이는 아래와 같은 실제 성능 시험에서도 확인할 수 있다.

Table 1은 삼각형 꼴 행렬과 정사각형 행렬의 분해 루틴에 따른 실측 계산 시간을 초 단위로 나타낸 것이다. DPPTRF는 LAPACK의 삼각형 행렬의 Cholesky 분해 루틴, DPOTRF는 정사각형 행렬의 분해 루틴이다. 위 결과는 인텔 Xeon X5482 두 개로 이루어진 총 8개의 다중코어 프로세서로 수행되었다. 표 1에서 보듯 크기 100부터 10000까지의 행렬에 대해서 정사각형 행렬의

Table 1. 삼각형 꼴 행렬 분해와 정사각형 꼴 행렬 분해의 계산 소요 시간

루틴 \ 행렬크기	100	500	1000	5000	10000
DPPTRF [1]	1.25 E-4	3.79 E-3	1.83 E-2	1.37	8.88
DPOTRF [2]	1.09 E-4	2.43 E-3	9.36 E-3	0.55	3.90
Ratio [1]/[2]	1.15	1.56	1.95	2.51	2.28

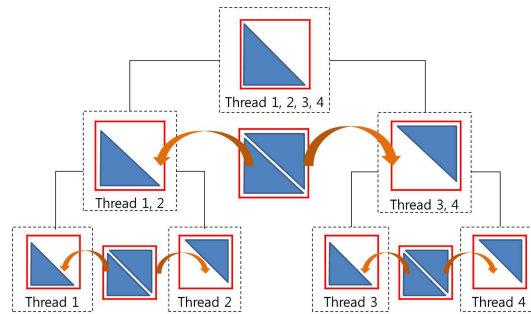


Fig. 2. 프론트 행렬 연성화

분해가 계산 효율성 측면에서 월등하며 이를 통해 일반적으로 정사각형 행렬을 계산에 사용하는 것이 권장된다고 볼 수 있다.

그렇다면 정사각형 행렬의 나머지 절반을 어떻게 활용하는 것이 좋을지 명확하다. 바로 두 개의 계산 단위가 사용하는 프론트 행렬의 위, 아래 부분을 하나의 정사각형 행렬의 위, 아래 부분에 할당해서 각각 독립적으로 활용하는 것이 뛰어난 계산 성능도 얻으며 동시에 메모리 소량도 삼각형 행렬과 비슷한 수준으로 유지할 수 있다. 두 계산 단위가 쓰는 프론트 행렬이 하나의 정사각형 행렬을 통해 연성(pairing)되었다는 의미로 프론트 행렬 연성화라고 부른다.

Fig. 2는 4개의 쓰레드 사용 시 연성된 프론트 행렬을 개념적으로 나타낸 그림이다. 최하단 단계에서 4개의 쓰레드가 필요로 하는 4개의 프론트 행렬들이 2개의 정사각형 행렬을 통해 2개씩 쌍을 구성하고 있다. 두 번째 소거 단계는 두 개의 계산 단위가 하나의 정사각형 행렬을 공유하여 각각 위, 아래 부분을 점유하고 있다.

2.4 프론트 행렬 분리

두 번째 방법은 프론트 행렬 분리라고 명칭하는 기법이다. 식 (2)의 K 행렬은 각각 K_{11} , K_{12} , K_{22} 의 세 부분으로 분리해 생각할 수 있으며 이들 부행렬(submatrix)들이 합쳐진 K 행렬을 강성행렬과 구분하여 프론트 행렬이라고 칭한다. 프론트 행렬 분리(splitting frontal matrix)는 프론트 행렬 K 의 세 부행렬들을 실제 메모리에서 분리된 공간에 배치하고 이를 논리적으로는 하나의 행렬로 취급한다는 점이 핵심이다.

기존 알고리즘은 프론트 행렬을 정사각형의 연속적인 메모리 공간에 배치하였다. 그러므로 K_{11} 다음에 바로 K_{12} 가 배치되고 이어서 K_{21} , K_{22} 가 순서대로 저장공간을 차지하는 형태이다. 실제로는 행렬이 대칭이므로 K_{21} 에 해당하는 공간

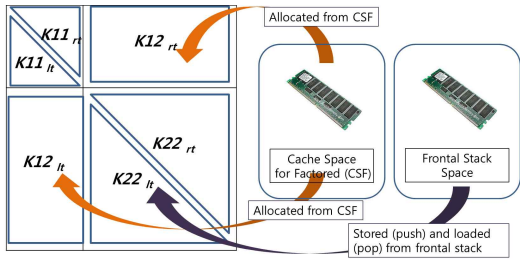


Fig. 3. 연성 분리 프론트 행렬 개념도

은 임시 데이터를 저장하는 선입선출의 스택으로 활용된다.

이에 비해 프론트 행렬 분리 기법은 K_{11} , K_{12} , K_{22} 에 해당하는 부행렬이 실제 메모리 공간에서 정사각형 행렬 내부에 위치하는 것이 아니라 서로 독립적으로 떨어진 공간에 할당된다. 정사각형 내부에 배치하는 것에 비해 이렇게 불연속적인 공간에 할당하는 것이 구현상의 어려움과 계산상의 불리함에도 가지는 장점은 중복되는 저장 공간의 크기를 줄일 수 있다는 점이다. K_{11} , K_{12} 는 Cholesky 분해가 끝나고 별도의 주기억장치 혹은 보조기억장치에 저장되고 K_{22} 는 조립이 완료되지 않았으므로 스택에 저장된다. 여기서 프론트 행렬을 분리하여 K_{11} , K_{12} 를 분해완료정보 캐쉬 공간(Cache Space for Factored)에 할당하고, K_{22} 는 스택 공간에 할당하여 이들을 하나의 논리적 묶음으로 다루면 별도의 프론트 행렬이 차지하는 공간이 불필요하다.

이 방법의 또 다른 이점은 이렇게 함으로서 메모리 복사 연산(memory to memory copy operation)을 줄일 수 있다는 점이다. 이 방법을 채택하지 않을 경우 프론트 행렬 공간으로부터 분해완료 정보 캐쉬 공간과 스택 공간으로의 부행렬 이동이 필요하다. 반면에 프론트 행렬 분리 기법은 처음부터 해당 공간에 부행렬들을 배치한 후 Cholesky 분해를 수행하므로 메모리 복사 연산이 필요가 없다.

Fig. 3은 연성화된 분리 프론트 행렬의 개념도를 나타낸다. 이 경우 2개의 계산 단위가 각각 3개씩의 부행렬들이 필요하므로 총 6개의 부행렬들이 모여 하나의 정사각형 행렬처럼 다루어진다. 실제로는 서로 다른 불연속적인 메모리 영역에 위치하지만 논리적으로 6개의 부행렬들이 하나의 정사각행렬을 구성하게 된다.

2.5 메모리 사용량 측정

이상에서 논의한 프론트 행렬 배치 방식에 따른 메모리 사용량을 측정하여 그 효율성을 비교

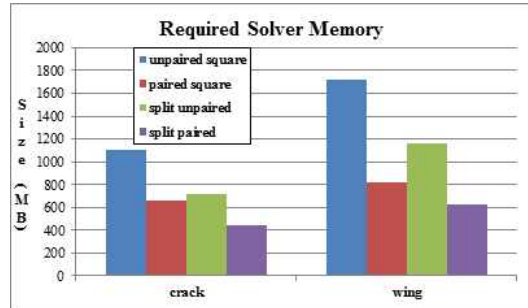


Fig. 4. 프론트 행렬 취급 방식에 따른 메모리 요구량

하였다. 프론트 행렬을 아래의 4가지 경우에 대한 프론트 행렬 및 스택 공간 크기의 합을 측정하였다. 이는 다중프론트 해법으로 문제를 해석하기 위한 최소의 메모리 요구량이기도 하다.

1. 연성되지 않은 정사각형 프론트 행렬 (unpaired square)
2. 연성된 정사각형 프론트 행렬 (paired square)
3. 연성되지 않은 분리 프론트 행렬 (split unpaired)
4. 연성된 분리 프론트 행렬 (split paired)





시험에 사용된 문제는 다음 장의 표 2에서 등장하는 4절점 사면체 요소로 이루어진 균열 해석과 4절점 사각형 요소로 구성된 비행체 날개 문제이다. 비정렬 격자계(unstructured mesh)와 정렬 격자계(structured mesh)에서 메모리 요구량의 양상을 자세히 보기 위해 두 가지 문제를 선택하였다.

Fig. 4에 두 개의 시험 문제의 메모리 요구량의 전술한 4개의 경우에 따라 나타내었다. Fig. 4에서도 보이듯이 두 가지 문제 모두 연성된 분리 프론트 행렬을 채택할 시 메모리 요구량이 가장 적다. 첫 번째와 두 번째, 세 번째와 네 번째 선택지를 각각 비교해 보면 프론트 행렬 연성화가 메모리 요구량을 절반 수준 가까이 줄여준다는 사실을 알 수 있다. 그리고 첫 번째와 세 번째, 두 번째와 네 번째 옵션을 각각 비교하면 프론트 행렬 분리 역시 메모리 요구량을 줄이는 데 큰 역할을 한다는 점을 알 수 있다.

2.6 단일쓰레드 및 다중쓰레드 알고리즘 계산성능 비교

이상에서 논의된 단일쓰레드 다중프론트 알고리즘과 다중쓰레드 병렬 알고리즘을 4개의 문제

Table 2. 단일쓰레드/다중쓰레드 알고리즘 성능 비교용 문제 정보

문제	특성
 Cubic	50x50x50 Hexa8 397953 DOFs
 Square	1000x1000 Quad4 6012006 DOFs
 Crack ¹⁾	393216 Tetra4 241533 DOFs
 Wing ²⁾	170356 Quad4 1023342 DOFs

에 대해 실제 계산 성능을 비교하였다. Table 2에 문제들의 요소, 자유도 개수가 요약되어 있다. 앞 장에서 소개된 균열 해석과 비행체 날개 문제와 함께 성능 검증용 문제인 3차원 50x50x50 육면체 요소로 구성된 정육면체 문제와 2차원 1000x1000 사각형 요소 평판 문제를 선택하였다.

그리고 각각의 문제에 대해 성능비교는 다음의 4가지 경우에 대해 비교하였다.

1. 다중쓰레드 알고리즘의 연성 정사각형 프론트 행렬 (PMFS paired)
2. 다중쓰레드 알고리즘의 연성 분리 프론트 행렬 (PMFS split paired)
3. 다중쓰레드 알고리즘의 비연성 분리 프론트 행렬 (PMFS split)

1) http://ipsap.snu.ac.kr/Download/IPSAP_FILE/input/stress6.in

2) http://ipsap.snu.ac.kr/Download/IPSAP_FILE/input/stress5_1M dof.in

4. 단일쓰레드 알고리즘 (MFSP)

여기서 다중쓰레드 알고리즘은 PMFS(Parallel Multi-Frontal Solver)라고 칭하였고, 단일쓰레드는 MFSP(Multi-Frontal Solver with Parallel numerical routines)라고 명명하였다. 2.4절에서 언급된 것처럼 프론트 행렬 분리 기법은 메모리 복사 연산을 줄여줌으로서 성능 향상을 꾀할 수 있다고 예상한 바, 실제 계산 성능으로 이를 확인하고자 하였다.

Fig. 5는 행렬 분해(5(a)) 및 전방소거/후방대입(5(b))에 걸리는 계산 소요 시간을 초 단위로 나타낸 그래프이다. 계산은 2.3절에 언급된 8개의 프로세서로 수행되었다.

Fig. 5(a), 5(b)에 보이는 것처럼 단일쓰레드 알고리즘보다 다중쓰레드 알고리즘이 최대 2배 가까이 빠르게 나옴을 알 수 있다. 특히 5(b)에서 보이듯이 전방소거/후방대입 시에 그 성능상의 이점이 두드러지게 나타난다. 그리고 5(a)에서 첫 번째와 두 번째 옵션의 비교를 통해 프론트 행렬

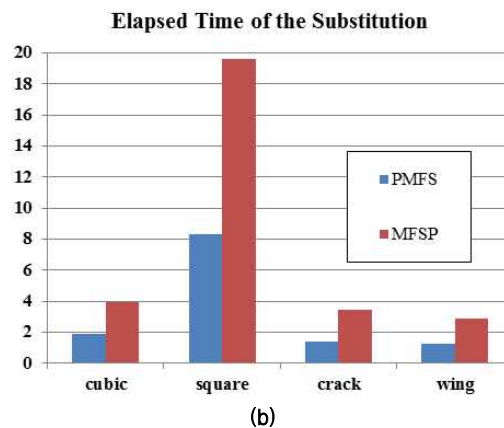
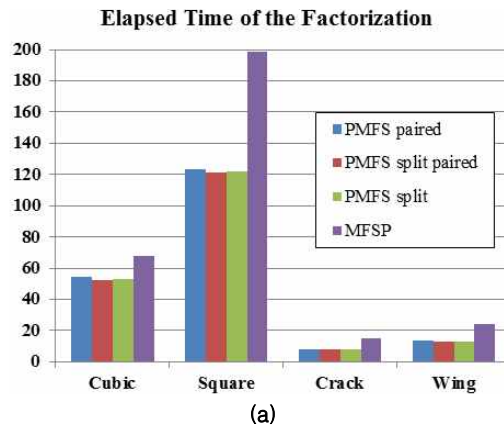


Fig. 5. 다중쓰레드/단일쓰레드 알고리즘의 분해(a) 및 전방소거/후방대입(b) 계산 시간

분리 유무에 따라 성능이 다소 차이가 남을 확인할 수 있다. 프론트 행렬 분리에 따른 메모리 복사 연산이 줄어들면서 4가지 문제 모두 계산 시간이 약간 짧아졌음을 확인할 수 있다.

III. 결 론

본 논문에는 기존의 분산병렬 다중프론트 알고리즘을 공유메모리 환경에 맞게 추가적인 병렬화를 수행하여 그 성능을 비교했다. 제시된 알고리즘은 다수의 계산 단위가 사용하는 주기억장치의 크기를 줄이며 동시에 성능향상도 꾀하는 두 가지 방법론을 제시하였다. 제안된 방법들을 적용한 다중쓰레드 병렬 알고리즘을 종전의 단일쓰레드 알고리즘과 여러 가지 문제에 대해 그 성능을 비교하여 본문에서 제안한 방법이 계산 성능이 훨씬 뛰어난 것을 보였다. 현대에 생산되는 많은 컴퓨터가 기본적으로 다중 프로세서를 장착하기 때문에 본문의 방법은 유한요소 구조해석에 큰 도움이 될 것이라고 기대할 수 있다.

후 기

본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었으며, 이에 대해 깊이 감사드립니다. (계약번호 UD100048JD)

참고문헌

- 1) Bhardwaj, M., et al. "Salinas: a scalable software for high-performance structural and solid mechanics simulations," *IEEE Computer Society Press*, 2002.
- 2) Farhat, C., M. Lesoinne, and K. Pierson, "A scalable dual-primal domain decomposition method," *Numerical linear algebra with applications*, Vol. 7, 2000, pp. 687~714.
- 3) Nakajima, K. and H. Okuda, "Parallel iterative solvers for unstructured grids using a directive/MPI hybrid programming model for the GeoFEM platform on SMP cluster architectures," *Concurrency and Computation-Practice & Experience*, Vol. 14, 2002, pp. 411~429.
- 4) Nakajima, K., "Parallel iterative solvers for finite-element methods using an OpenMP/MPI hybrid programming model on the Earth

Simulator," *Parallel Computing*, Vol. 31, 2005, pp. 1048~1065.

- 5) Yoshimura, S., et al., "Advanced general-purpose computational mechanics system for large-scale analysis and design," *Journal of computational and applied mathematics*, Vol. 149, 2002, pp. 279~296.

- 6) Miyamura, T., et al., "Elastic-plastic analysis of nuclear structures with millions of DOFs using the hierarchical domain decomposition method," *Nuclear engineering and design*, Vol. 212, 2002, pp. 335~355.

- 7) 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.

- 8) 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.

- 9) Kim, J.H. and S.J. Kim, "Multifrontal solver combined with graph partitioners," *AIAA journal*, Vol. 37, 1999, pp. 964~970.

- 10) Kim, M.K. and S.J. Kim, "High performance hybrid direct-iterative solution method for large scale structural analysis problems," *International Journal for Aeronautical and Space Science*, Vol. 9, 2008, pp. 79~86

- 11) Duff, I.S. and J.K. Reid, "The Multifrontal Solution of Indefinite Sparse Symmetric Linear-Equations," *ACM Transactions on Mathematical Software*, Vol. 9, 1983, pp. 302~325.

- 12) Irons, B.M., "A frontal solution program for finite element analysis," *International Journal for Numerical Methods in Engineering*, Vol. 2, 1970, pp. 5~32.

- 13) Karypis, G. and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol. 20, 1998, pp. 359~392.

- 14) Kim, M.K. and S.J. Kim, "Parallelization of bisection mesh partitioning routine for parallel multifrontal solver," *ICCES Special Symposium of Meshless and Other Novel Computational Methods*, 2010. pp. 58-58.

- 15) <http://www.openmp.org/>

- 16) <http://threadingbuildingblocks.org/>