論文 J. of The Korean Society for Aeronautical and Space Sciences 42(6), 445-452(2014)
DOI:http://dx.doi.org/10.5139/JKSAS.2014.42.6.445

대규모 구조해석을 위한 보조기억장치 활용 선형 직접해법

김민기*, 김승조

An Out of Core Linear Direct Solution Method for Large Scale Structural Analysis

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

ABSTRACT

This paper discusses the multifrontal direct solution method with out of core storage for large scale structural analysis in a limited computing resource. Large scale structural analysis requires huge amount of memory space and computation, so out of core solution method is needed in limited computing resource. In this research, out of core multifrontal solution algorithm which utilize the small size of physical memory and minimize the amount of access of low speed out of core storage is introduced. Three ideas, which are stack space in lower trianglar part of square factorization matrix, inverse stack data structure and selective data caching and recovery by data block size, are proposed.

초 록

본 논문은 제한된 계산 자원을 가진 환경에서 대규모 구조해석을 위해 고안된 보조기억장치를 활용하는 선형 직접해법에 대해 논의한다. 대용량 구조해석은 많은 기억공간과 계산량을 요구하기에 계산 자원이 부족할 경우 보조기억장치를 활용하는 해법을 개발할 필요가 있다. 본 연구는 한정된 주기억장치의 활용성을 극대화하고 상대적으로 느린 보조기억장치 저장량을 최소화하는 다중프론트 해법의 알고리즘을 소개한다. 구조해석 문제의 대칭성을 활용한 스택 공간 사용 기법과 역순 스택 자료 구조, 데이터 블록 크기에 따른 선택적 저장 기법과 데이터 복원 기법을 제시하였다. 본문에서 논의된 방법들을 적용한 다중프론트 해법이 여러 성능비교 문제에서 더 나은 계산 성능을 보임을 확인할 수 있다.

Key Words : Structural Analysis(구조해석), Out of Core Storage(보조기억장치), Multifrotnal Method(다중프론트 해법), Stack space(스택 공간), Invesse Stack(역순 스택), Selective caching and recovery(선택적 저장 및 복원)

Ⅰ. 서 론

유한요소 구조해석은 컴퓨터의 발전에 따라서 점점 그 활용도가 높아지고 있다. 항공우주 분야를 포함한 기계, 조선, 토목 등의 산업에 이어서 반도체 등의 전기전자까지 그 필요성이 증가하고 있다. 컴퓨터 성능이 1.5년마다 2배로 증가한다는 무어의 법칙에 따라서, 전산구조해석의 활용빈도

† Received: Novembet 22, 2013 Accepted: May 12, 2014
* Corresponding author, E-mail : mkkim12@kari.re.kr
http://journal.ksas.or.kr/
pISSN 1225-1348 / eISSN 2287-6871

와 더불어 각 문제의 평균 크기도 그에 걸맞게 증가하였다. 더군다나 현재의 항공우주 구조물을 포함한 대다수의 기계, 조선 등의 제품 설계 시 상세 설계 단계에서 최적 설계 기법을 적극적으로 도입하면서 많은 수의 구조해석을 필요로 하는 추세를 감안하면 고정밀의 신뢰성 높은 해석 결과를 얻기 위해서 고성능의 해석 이론이 필요하다는 것을 알 수 있다. 전산 구조해석에서 많이 풀게 되는 선형 응력해석, 고유치 해석, 비선형 해석 등에서 가장 높은 계산 비율을 차지하는 것이 바로 선형해법이므로 이의 효과적인 해법이 필요하다.

과거부터 현재까지 해석하고자 하는 문제의 크기에 비해 설계자의 컴퓨터 자원은 한정되어 있기에 보조기억장치를 활용한 선형해법이 유한요소 구조해석에 적용되고 있다. 다수의 선형해법 알고리즘[1,2]이 고안되어 구조해석을 포함한 많은 분야에 적용되고 있지만, 보조기억장치 활용 가능한 해법은 그 수가 많지도 않을 뿐만 아니라 그 성능도 뛰어나다고 보기 힘들다. 유한요소구조해석의 선형해법으로 널리 사용되는 다중프론트 해법(multifrontal method)[3,4,5,6]은 조립이 완료된 자유도는 그 즉시 소거(elimination)가 가능하며, 소거된 자유도에 해당하는 정보는 이후의 행렬 분해(factorization) 단계에서는 활용되지 않기에 이를 임시로 보조기억장치 등에 저장하여 이후에 방정식 삼각 해법(Triangular solve) 단계에서 다시 불러와서 활용할 수 있다. 이 경우 제한된 주기억장치(physical memory)의 활용성을 극대화하고 상대적으로 느린 속도의 보조기억장치 접근량을 최소화하는 노력이 요구된다. 특히 주기억장치의 용량이 적을 경우 보조기억장치의 비율이 그만큼 증가하기에 한정된 주기억장치의 효과적인 활용 방안이 성능에 매우 큰 영향을 끼친다. 따라서 이상을 종합하면 주기억장치만을 활용하는 기존 이론과는 별개로 주기억장치와 보조기억장치를 효과적으로 이용하기 위한 새로운 방식의 알고리즘이 필요하다고 할 수 있다.

본 연구에서는 유한요소 구조해석에 가장 많이 활용되는 선형해석 기법인 Cholesky 행렬 분해 기법을 다중프론트 해법으로 구현한 것을 논하기로 한다. Cholesky 행렬 분해 기법은 양정치(positive definite)이고 우량조건인(well-posed) 유한요소 문제의 해법에 흔히 적용되는 기법이다. 다만 이 경우 경계조건이 없는 부유구조물 등에는 직접 적용하기가 어렵기에 별도의 알고리즘 개조를 통한 해석방법의 개선이 필요하다.

II. 본 론

2.1 다중프론트 해법 개요

다중프론트 해법은 Iron[3]이 제안한 방법을 Duff[4]가 유한요소법에 적용하여 개량한 선형해법으로서 단일프론트 방법보다 기억장치 요구량이 적고 병렬화에 용이하다. 주 개념은 조립이 완료된 유한요소 강성행렬의 성분들을 그렇지 않은 성분들로 치환하는 과정을 반복하는 것이다. 이 과정을 정적 축약(static condensation)이라고 부르며 이 과정은 다수의 각 프론트의 소거 과정에서 동일하게 적용되는 수치 연산 과정이다.

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{1}$$

위 식 (1)에서 조립이 완료된 자유도를 u_1, 여기에 연결된 자유도를 u_2라고 하면, u_1은 조립이 끝나서 더 이상 행렬 계수 K_{11}, K_{12}, F_1이 변하지 않기에 (1)의 강성행렬은 바로 아래 식 (2)와 같이 분해가 가능하다.

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{12} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \overline{K_{22}} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{12}^T \\ 0 & I \end{bmatrix} \tag{2}$$

$$\begin{aligned} K_{11} &= L_{11} L_{11}^T \\ \overline{L_{12}} &= K_{12} L_{11}^{-1} \\ \overline{K_{22}} &= K_{22} - L_{12} L_{12}^T \end{aligned} \tag{3}$$

이상의 식 (2)와 (3)처럼 분해가 완료된 성분 L_{11}과 L_{12}는 이후의 분해 과정에서는 더 이상 활용되지 않기에 이들을 주기억장치 외에도 보조기억장치에도 저장 가능하다. 그리고 조립이 끝나

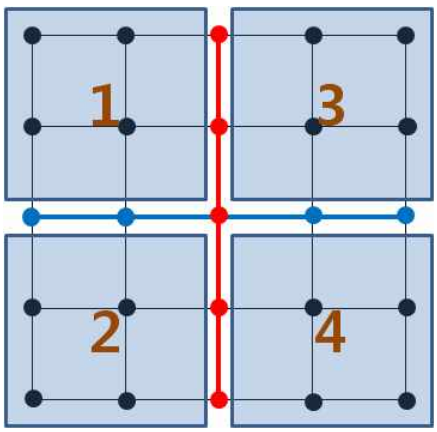

Fig. 1. A Finite Element Problem Divided by Four Subdomain

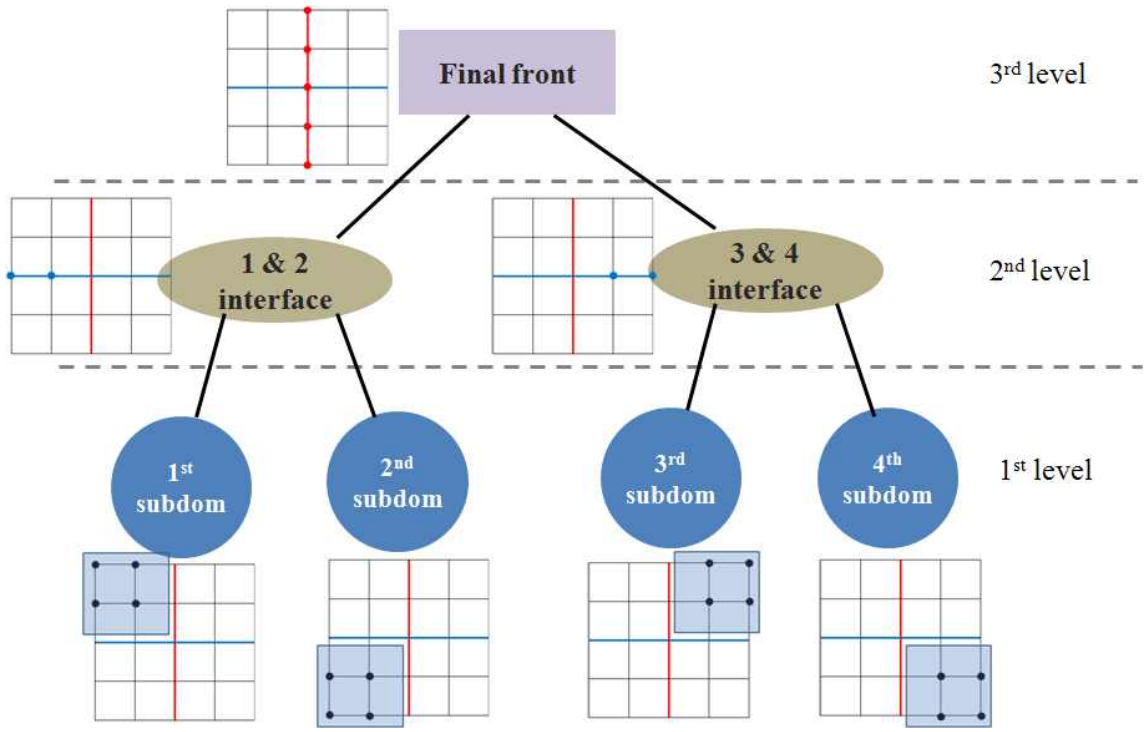

Fig. 2. Elimination Tree of the Numerical Factorization of Four Subdomain Problem

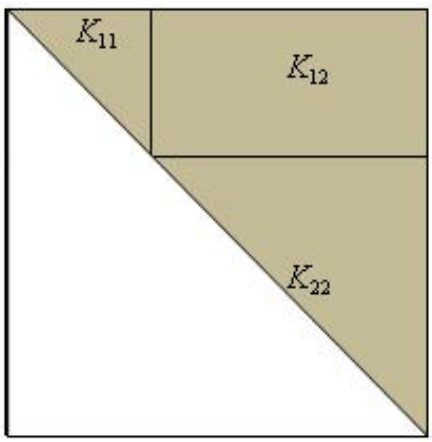

Fig. 3. Memory Structure of Square Form of Factorization Matrix

지 않은 u_2와 관련된 행렬 성분 K_{22}는 임시 공간인 프론트 스택(front stack)에 저장된다. 본 연구 방식의 다중프론트 해법은 부영역 별로 조립이 완료되면 소거하므로 여타 다른 선형해법들과 다르게 전역강성행렬을 희소 행렬 형식으로 조립할 필요가 없다.

계산량과 저장 공간을 최소화고 병렬화의 용이성을 높이기 위해 완전이진트리로 계산 단계가 구성된 방식을 채택하였다. 수치연산에 도입하기 전에 METIS[7]라이브러리를 활용하여 영역 분할을 수행[8]한다.

Figure 1은 4x4개의 균일한 격자계에서 2x2로 영역을 분할했을 경우 분할된 영역과 경계를 나타낸 그림이고 Fig. 2는 그에 맞는 계산 순서와 트리 구조를 나타낸 것이다.

2.2 프론트 스택 공간 재사용

2.2.1 하삼각행렬 스택 공간 활용

행렬분해가 각 단계마다 진행되는 동안 식 (2)의 K22는 추후 계산을 위해 임시 공간인 스택에 저장해야 한다. 중간 단계 행렬을 포함한 중간 데이터는 스택처럼 후입선출(Last Iput, First Output)이라는 점에서 스택 공간을 최소화하기 위한 알고리즘을 유추할 수 있다. 첫 번째는 정사각행렬로서 식 (2)의 정적 축약을 위해 사용되는 공간 중 하부의 삼각행렬을 스택으로 사용하는 것이다. 구조해석을 포함한 대다수의 유한요소 행렬은 대칭 양행렬이므로 상부 혹은 하부 중의 한 부분의 계산 공간만 있으면 된다. 이 때 계산 효율성 때문에 연속적으로 이어진 삼각행렬을 사용하지 않고, 정사각행렬 공간 중 상부의 반을 연산에 이용한다. 압축된 삼각행렬과 정사각행렬 사용 시 실측된 계산 효율성은 Kim 등이 제시한 논문[6]에 잘 나와 있으며, 따라서 정사각행렬의 반을 스택으로 사용하는 것이 올바르다고 할 수 있다.

Figure 3은 정사각 프론트 행렬 적용 시 메모리 구조와 공간 할당을 나타낸 그림이다. 강성행

렬의 대칭성을 적극 활용하여 하부의 빈 공간을 임시 스택으로 활용하면 성능상의 이점과 동시에 기억공간의 활용성이 높아짐을 알 수 있다.

2.2.2 역순 스택 자료 구조

스택 공간을 최소화하기 위한 다른 방법은 역순 스택 자료 구조(inverse stack data structure)이다. 일반적으로 필요한 스택의 크기는 Fig. 3에 나온 하삼각행렬의 크기로는 부족한 경우가 많으며 추가로 여분의 공간이 필요한 경우가 많다. 이 여분의 공간은 임시 공간이므로 그 활용 시점이 끝나면 분해된 데이터(factored data)를 저장하기 위한 공간으로 활용할 수 있다. 이를 위해 고안된 것이 역순 스택 자료 구조이다.

본 자료구조는 긴 배열의 한쪽 끝은 배열처럼 분해된 데이터가 순차적으로 쌓이고, 다른 한 쪽 끝은 스택으로 활용된다. 스택의 특성 상 그 크기가 가변적으로 변하고 이는 일반적인 스택의 성장 방향과 반대로 보이기에 역순 스택이라고 명명하였다.

Figure 4에서 보듯이 배열의 시작 주소부터 행렬 분해가 진행될수록 분해된 데이터가 축적된다. 반대편 끝은 스택의 시작점으로서 스택의 성장과 축소가 배열의 시작방향으로 이루어지게 된다. 행렬 분해가 진행될수록 최정점까지 올랐던 스택은 점차적으로 축소하고 그 자리를 분해된 데이터를 저장하기 위한 공간으로 할당한다. 계산이 최종적으로 완료되면 스택으로 사용되었던 공간까지 모두 분해된 데이터를 저장하기 위한 공간으로 활용된다.

Figure 5는 4개의 부영역을 가진 문제의 행렬 분해 진행 시 계산 순서(elimination order)에 따른 역순 스택 내부의 데이터 저장과 인출에 대한 흐름도이다. 이 경우는 물리메모리가 모든 프론트를 저장할 만큼 충분하다고 가정하였다. 하지만 본 논문의 연구 목적과 같이 물리메모리가 부족할 경우에 본 방식이 효력을 발휘한다.

이 방식은 수치 연산 전에 분해된 데이터의 크기와 스택의 최대 크기가 계산의 진행에 따라 어떻게 될지 미리 결정하여 두 데이터 공간이 충돌하지 않게 그 크기를 결정해야 한다. 일단 해당 크기만큼 물리메모리에 저장공간을 할당하면, 계산 단계마다 각 메모리 공간을 할당할 필요가 없기에 프로그램의 간접적인 부하(overhead)를 줄일 수 있다. 보통 작은 조각의 빈번한 메모리 할당과 커다란 크기의 메모리 할당 모두 시스템

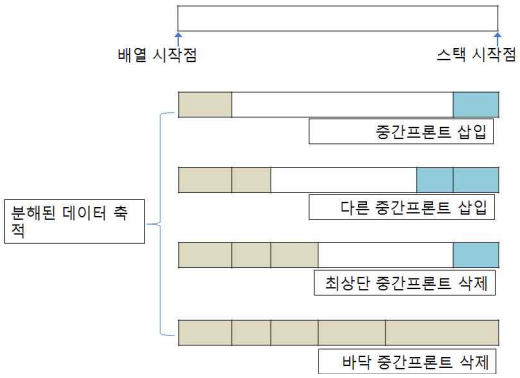

Fig. 5. Inverse Stack Data Structure

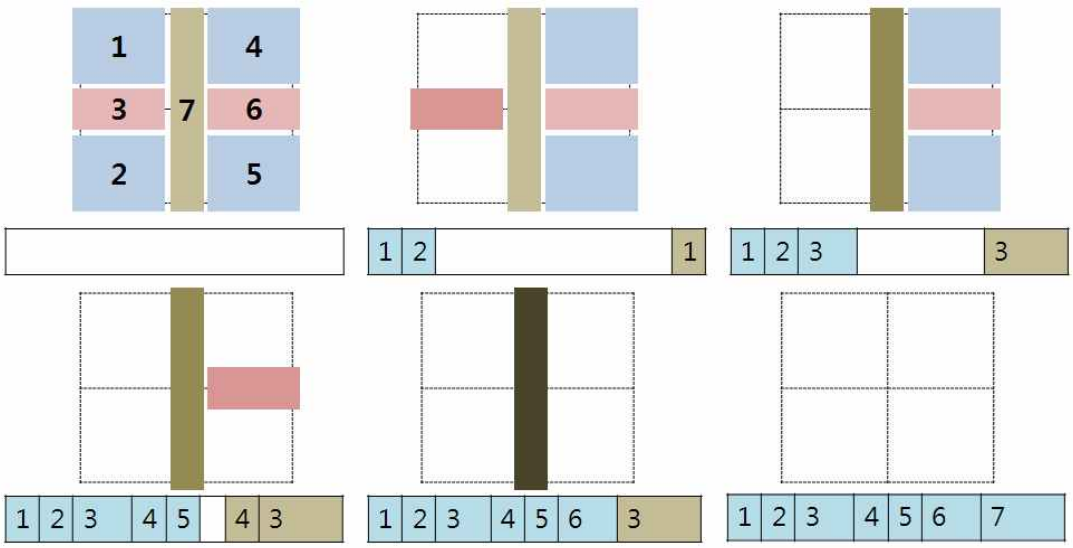

Fig. 4. Data Flow Diagram of Inverse Stack Data Structur of Four Subdomain Problem during Numerical Factorization

에 큰 부하를 일으킬 여지가 있기에 위 방법은 물리메모리 공간을 절약함과 동시에 불필요한 부하를 줄일 수 있다.

2.3 선택적 데이터 저장과 복원

선택적 데이터 저장 기법은 데이터의 크기에 따라서 주기억장치에 저장하는 우선 순위를 결정하는 기법이다. 다중프론트 해법은 Fig. 2의 계산 순서(elimination order)에서 후위(postorder)로 계산을 진행하는 것이 가장 저장공간 필요 측면에서 효율적이라고 알려져 있다. 이 때 계산 순서대로 데이터를 저장하는 것을 순차적 저장(sequential caching), 데이터 크기에 따라서 저장하는 것을 선택적 저장(selective caching)이라고

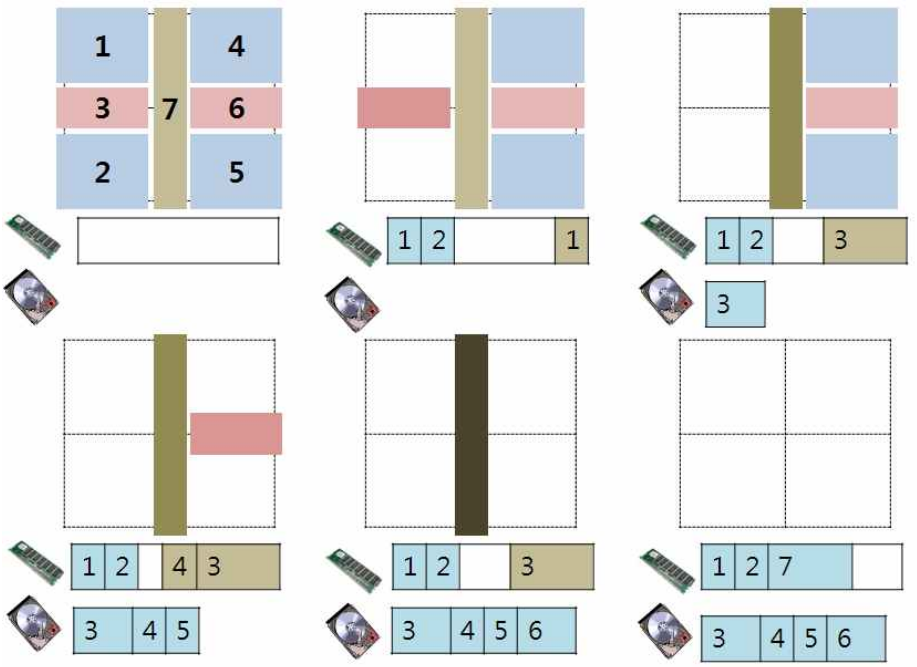

Fig. 6. Sequential Caching on Liminted Memory

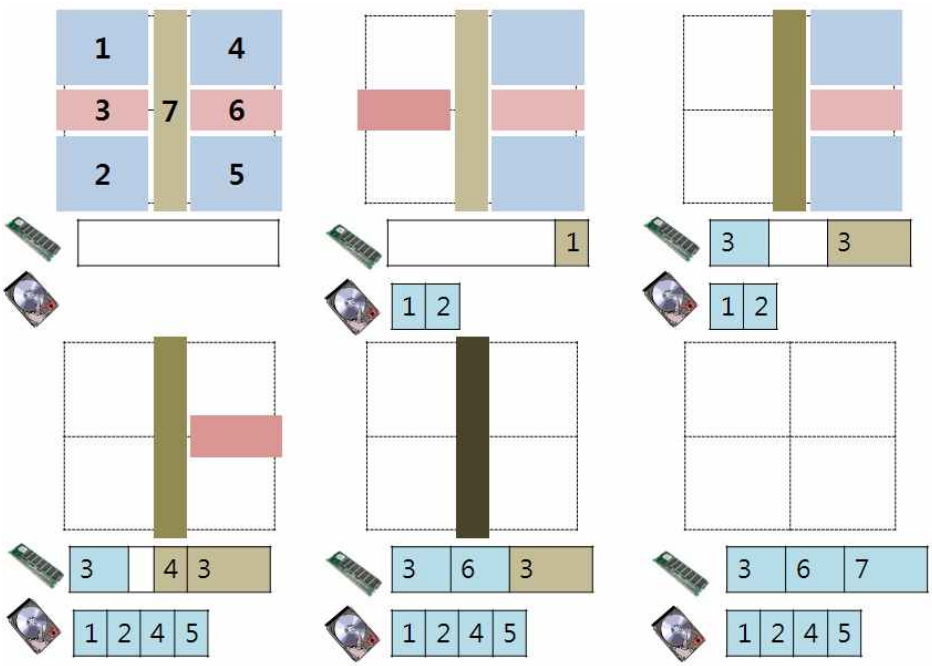

Fig. 7. Selective Data Caching on Limited Memory

한다. Fig. 2에서 상위 단계의 계산으로 진행하면 프론트의 크기가 커져서 데이터 크기도 그에 맞게 커지게 된다.

순차적 저장 기법을 채택 시 하위 단계들의 크기가 작은 데이터들이 가용한 주기억장치를 차지하게 되고 계산 후반의 큰 데이터 블록들은 그 크기에 맞는 여분의 공간 부재로 인해 보조기억장치에 저장되게 된다. 반면 선택적 저장 기법은 데이터 크기에 따라서 주기억장치 할당 순위를 정하기 때문에 계산 순서에 상관없이 큰 데이터부터 메모리 할당 순위에서 우선을 차지하게 된다. Fig. 6과 7은 선택적 데이터 저장과 순차적 데이터 저장을 역순 프론트 스택을 적용했을 때 계산 순서에 따라 나타낸 그림이다. Fig. 6은 가용한 주기억장치를 모두 분해된 데이터 저장에 사용할 수 있는 반면, Fig. 7은 활용하지 못하는 공간이 남는다는 것을 알 수 있다.

이와 같은 과정은 행렬 분해 후 삼각해법(triangular solve)과정에도 적용할 수 있다. 행렬 분해 시 사용되던 식 (2)의 프론트 행렬을 위한 정사각행렬은 분해 후에도 재활용될 수 있다. 보조기억장치의 데이터를 불러 들여서 정사각행렬이 차지하던 공간에 저장할 수 있고 이는 삼각해법 과정에서 큰 효과를 발휘한다. 이를 데이터 복원(data recovery)라고 하며 이 과정에서도 데이터 크기가 큰 순서대로 복원 순위를 결정하면 데이터 크기와 가용한 저장공간 크기가 맞지 않아 생기는 빈 공간을 최소화할 수 있다.

2.4 성능 비교 및 결과

2.4.1 메모리 및 데이터 용량 비교

제안된 세 가지 방식을 적용한 보조기억장치 활용 다중프론트 방법을 기존 방법과 비교하였다. 첫 번째로 프론트 스택 자료구조에 따라서 두 가지 비교 문제를 선정하여 다음의 세 가지 경우의 메모리 사용량을 비교하였다.

Algorithm 1. 하삼각행렬 사용하지 않는 일반적 스택

Algorithm 2. 하삼각행렬 사용하는 일반적 스택

Algorithm 3. 하삼각행렬 사용하는 역순 스택

비교로 사용한 문제는 3차원 균열 모델과 2차원 날개 모델로서, 각각 4절점 고체요소와 4절점 사각형 요소로 구성되어 있고, 174,510, 511,110개의 자유도를 가지는 문제들이다. Fig. 8에서 보듯이 하삼각행렬을 스택공간으로 사용하는 역순 스택이 총 기억공간 요구량이 가장 작음을 알 수 있다. 스택 공간을 최소화함으로서 얻게 되는 저장공간 확보가 그렇지 않은 경우에 비해 약 10% 가량 향상됨을 알 수 있다.

두 번째는 분해된 데이터 저장과 복원 기법에 따른 데이터 복원 크기를 비교하였다.

Algorithm 1. 선택적 데이터 저장과 선택적 복원

Algorithm 2. 선택적 데이터 저장과 순차적 복원

Algorithm 3. 순차적 데이터 저장과 순차적 복원

시험에 사용된 문제는 8절점 고체요소 50x50x50으로 이루어져 있고 3000MByte의 메모리 용량을 가지고 성능을 측정하였다.

Figure 9는 세 가지 방법으로 데이터 저장과 복원을 수행했을 시 삼각해법 과정에서 복원된 데이터 크기를 비교한 것이다. 그림에서 파란색 막대는 복원된 데이터, 빨간색 막대는 삼각해법에서 필요로 하는 계산 공간을 나타내었다. 선택적 저장과 선택적 복원이 가장 많은 데이터를 보조기억장치로부터 주기억장치로 복구할 수 있으며, 삼각해법 중에 필요한 공간이 가장 작다는 것도 알 수 있다. 요약하면 선택적 저장과 선택

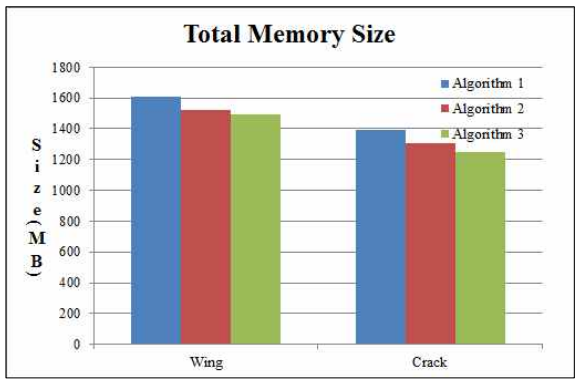

Fig. 8. Memory Requirement of Three Kinds of Front Stack

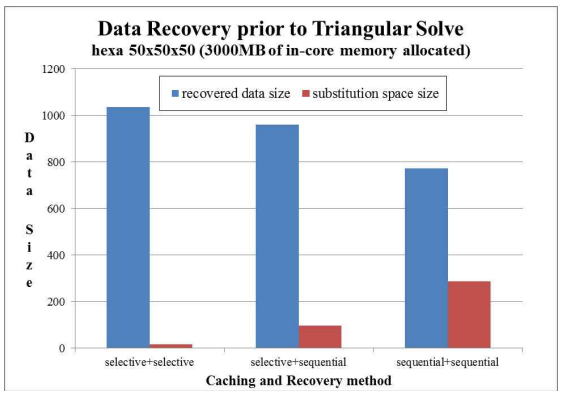

Fig. 9. Recovered Data Sizes Comparison between Selective and Sequential Data management

적 복원이 허용된 메모리 공간에 가장 많은 데이터 블록을 저장 가능하며, 가장 적은 계산 공간을 요구하기에 실제 가용한 저장 공간이 더 확장되는 효과까지 있다고 할 수 있다.

2.4.2 실제 계산 속도 비교

기존 방법과 본 논문에 제시된 방법들과 실제 계산 속도를 측정하여 비교하였다. 시험에 사용된 문제는 이미 언급된 3차원 정육면체 50x50x50 문제이며 컴퓨터는 인텔 i7 X980 3.3GHz, 8GByte 메모리에 윈도우 7 64비트 운영체제를 갖추었으며, 인텔 MKL [9] 10.3.2와 인텔 C/C++ 컴파일러로 코드를 개발하였다.

시험한 문제는 약 1GB의 프론트 행렬, 800MB의 프론트 스택을 요구하며, 5.3GB의 분해된 행렬의 데이터가 생성된다. 이를 3000MB, 4000MB, 5000MB의 메모리 용량을 허용하고 아래의 제시된 네 가지 옵션으로 해석하였다.

Algorithm 1. 선택적 데이터 저장과 복원, 하삼각행렬 공간이 더해진 역순 스택

Algorithm 2. 순차적 데이터 저장과 복원, 하삼각행렬 공간이 더해진 역순 스택

Algorithm 3. 선택적 데이터 저장과 복원, 하삼각행렬 공간을 제외한 일반적 스택

Algorithm 4. 순차적 데이터 저장과 복원, 하삼각행렬 공간을 제외한 일반적 스택

Table 1과 Fig. 10에 위의 4가지 옵션으로 해석 결과를 MFlops(10^6 floating point operation /sec)단위의 계산 속도로 나타내었다. Fig. 10은 해석기법 중 수치적 행렬 분해(numerical factorization)와 삼각 해법 두 가지를 고려하였으며 기호 분해(symbolic)를 포함한 전처리 과정은 본 결과에 포함하지 않았다.

2.4.1절의 데이터 용량 비교에서 예상할 수 있듯이 실제 해석 속도 결과도 본문에서 제안된 방법들이 모두 적용된 1번이 가장 높은 속도를 보임을 알 수 있다. 그리고 1,2와 3,4를 비교하면 프론트 스택의 필요 공간 최소화가 데이터 저장/복원 기법에 비해 속도 향상에 기여하는 바가 더욱 크다는 점도 알 수 있다. 프론트 스택은 행렬 분해시에만 소요되는 공간이므로 삼각해법 과정에는 데이터 저장과 복원 기법의 선택 여부가 속도에 영향을 미치게 된다. Fig. 11은 삼각해법의 속도를 나타낸 그래프로서, 선택적 데이터 저장과 복원이 삼각해법의 속도 향상에 기여함을 알 수 있다. 이는 특히 선형 내연적 해석 기법

Table 1. Computational Speed of Various Kinds of Available Memory

옵션	3000	4000	5000
1	25278.1	30362.2	37028.4
2	23798.5	28669.8	34779.4
3	21265.9	25231.7	31072.9
4	20961.3	24862.1	29815.7

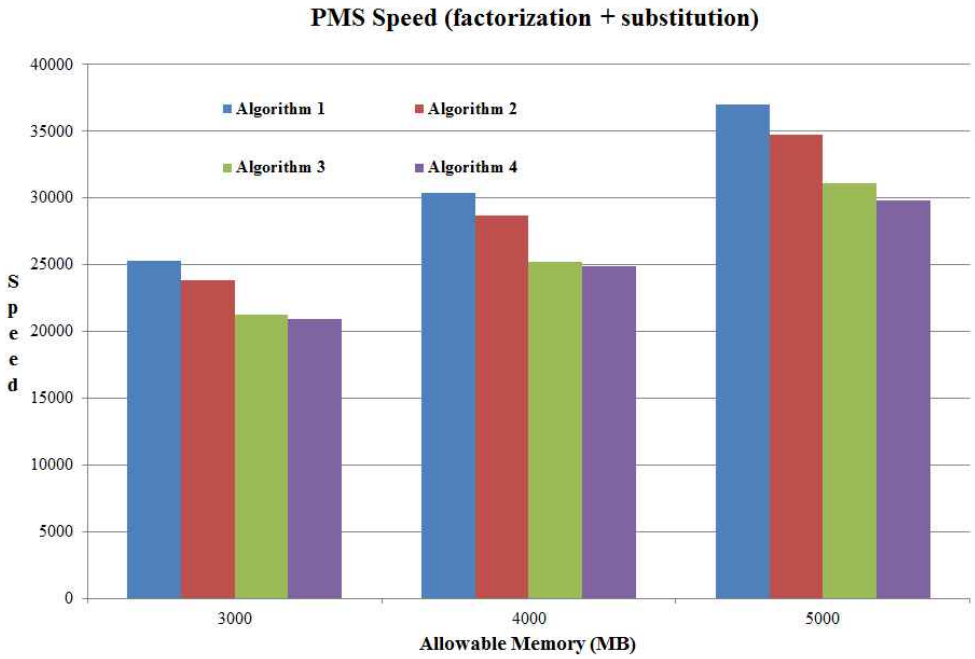

Fig. 10. Compuiting Performance according to Available Physical Memory

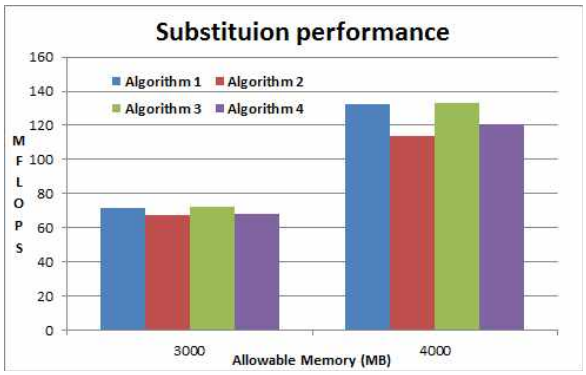

Fig. 11. Computing Speed of Triangular Solution of Various Kinds of Available Memory

(linear implicit dynamic analysis)나 고유치 해석(natural frequency analysis)과 같이 반복적인 삼각해법이 계산의 큰 비중을 차지하는 곳에서 큰 강점을 발휘할 것이다.

III. 결 론

본 논문에서 다중프론트 해법의 보조기억장치 활용 성능을 극대화하기 위한 다양한 기법들을 제시하였고 그 성능을 비교하였다. 제시된 방법들은 임시 공간의 크기를 줄임으로서 초기 가용 물리메모리 공간을 더 크게 확보할 수 있고, 데이터 저장 순서를 최적화하여 보조기억장치 데이터 입출력량을 최소화함으로서 보조기억장치로부터 나오는 유휴 대기 시간을 크게 줄일 수 있다. 다양한 분야의 구조해석 현업에서 제한된 컴퓨터 자원으로 대규모 문제를 해석하는 경우가 빈번함을 고려하면, 본 연구는 여기서 크게 강점을 발휘할 것으로 예상할 수 있다.

후 기

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

References

1) 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.

2) O. Schenk, and K. Gärtner, "Solving Unsymmetric Sparse System of Linear Equation with PARDISO,", *Journal of Future Generation Computer Systems*, Vol. 20, No. 3, 2004, pp. 475-487.

3) 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.

4) 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.

5) Kim, J. H., Lee, C. S., and Kim, S. J. "High Performance Domainwise Parallel Direct Solver for Large-Scale Structural Analysis," *AIAA journal*, Vol. 43, No. 3, 2005, pp. 662-670.

6) M.K.Kim, J.H.Kim, C.Y.Park and S.J.Kim, “Parallelization of Multifrontal Solution Method for Shared Memory Architecture," *Journal of The Korea Society for Aeronautical and Space Science*, Vol. 40, No. 11, 2012, pp. 972-978

7) Karypis, G., and Kumar, V. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, 1998, pp. 359-392.

8) Kim, M. K., and Kim, S. J. "Parallelization of Bisection Mesh Partitioning Routine for Parallel Multifrontal Solver," *ICCES Special Symposium of Meshless and Other Novel Computational Methods*. 2010, pp. 58-58.

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