# 假想 메모리 데이타베이스를 이용한 大規模 構造解析用 코어 外 方程式 해석기법의 개발*

# Development of Out-of-Core Equation Solver with Virtual Memory Database for Large-Scale Structural Analysis

李 成 雨*
Lee, Sung Woo
宋 允 煥**
Song, Yoon Hwan
李 東 根***
Lee, Dong Guen

## 요 약

컴퓨터의 制限된 코어메모리로 大型문제를 해결하기 위하여 디스크를 마치 메모리처럼 使用할 수 있는 假想 메모리 데이타베이스 기법을 개발하였다. 이 기법과 아울러 최대 可用코어메모리를 移動시키는 方式을 사용하여 有限要素 解析시 흔히 발생하는 스카이라인 형태로 저장된 對稱疏散行列(Sparse Symmetric Matrix에 대한 매우 효과적인코어內 및 코어外 聯立方程式의 解法을 개발하였다. 제안된 방법은 다른 코어外 解法에 비해 알고리즘 및 코딩이 매우 간단하여 계산효율을 상당히 향상시켰다. 해석예에서는 제안된 방법을 사용하여 大規模 構造解析 문제를 메모리 용량이 작은 小型컴퓨터에서 대단히 효율적으로 해결하였음을 보여주었다.

## Abstract

To solve the large problems with limited core memory of computer, a disk management scheme called virtual memory database has been developed. Utilizaing this technique along with memory moving scheme, an efficient in-and out-of-core column solver for the sparse symmetric matrix commonly arising in the finite element analysis is developed. Compared with other methods the algorithm is simple, therefore the coding and computational efficiencies are greatly enhanced. Analysis example shows that the proposed method efficiently solve the large structural problem on the small-memory micro-computer.

* 정회원, 국민대학교 토목공학과 조교수
** 정회원, 한국과학기술원 토목공학과 박사과정
*** 정회원, 한국과학기술원 토목공학과 조교수
* 이 논문은 1990년도 문교부지원 학술진흥재단의 자유공모과제 학술연구 조성비에 의하여 연구 되었음.

이 논문에 대한 토론을 1991년 9월 30일까지 본 학회에 보내주시면 1992년 3월호에 그 결과를 게재하겠습니다.

## 1. 서 론

構造解析 분야에서 컴퓨터해법이 발달함에 따라 대단히 복잡하고 거대한 구조물도 어렵지 않게 그 해석이 가능하여 졌다. 大型 構造物의 모델을 해석하기 위해서는 프로그램內에서 Dimension이 큰 Array의 使用이 불가피하고, 따라서 컴퓨터의 큰 메모리 容量이 필요하게 된다. 이러한 문제는 小型 컴퓨터뿐만 아니라 大型 컴퓨터에서도 큰 制約條件이 된다. 특히 근래와서 低廉하면서도 機能이 우수한 Workstation이나 퍼스널 컴퓨터 등의 小型 컴퓨터가 널리 普及되면서 이러한 메모리 容量의 制限을 효과적으로 극복해야 할 必要性이 절실히 요청되고 있다.

이러한 문제를 解決하기 위하여 본 연구에서는 Binary 直接接近 화일(Direct Access File)을 사용하여 디스크를 마치 메모리처럼 이용할 수 있는 假想메모리 데이타베이스기법[1]을 제시하였다. 이 방법은 메모리 容量의 制限을 克服하는 기법이므로 構造分野뿐만 아니라 어떤 유형의 컴퓨터 이용분야에도 이용될 수 있을 것이다.

本 硏究에서는 이러한 데이타베이스 기법을 사용하여 有限要素法을 이용한 大規模 構造解析에서 많은 시간을 要하는 聯立方程式 解를 구하는 부분에 적용하여 효율적인 코어內 및 코어外 解法의 알고리즘을 개발하였다. 이 方程式 解法에서는 우선 假想 메모리 데이타베이스 기법을 이용하여 可用 메모리를 解法 부분에 最大로 이용할 수 있도록 한 후, 可用 메모리를 메모리 윈도우로 설정하여 이 메모리 윈도우가 移動하면서 方程式을 풀 수 있도록 하였다. 消去할 行列(有限要素 解析에서는 剛性行列)은 스카이라인 方式으로 저장하였고 解法은 對稱疏散行列(Sparse Symmetric Matrix)에 대해 가장 효율적인 方法중의 하나로 알려진 修正 Crout法[3]을 이용하였다. 또한 코어메모리와 디스크를 혼용한 假想메모리 개념을 사용하였기 때문에 기존의 블럭방식 코어外 해법[4]과는 달리 코어內와 코어外를 自動的으로 구분하여 해를 구할 수 있는 매우 효율적인 방법을 도입하였다.

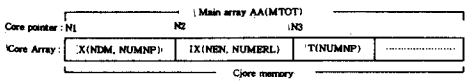
## 2. 假想메모리 데이타베이스

大型문제를 해결하고자 할 경우 일반적으로 프로그램內에서 Dimension이 큰 Array들을 使用하게 되고, 이 경우 情報의 貯藏과 回收를 코어메모리(RAM, Random Access Memory)內에서 처리하므로 신속한 연산이 가능하게 된다. 그러나 이러한 프로그램이 컴퓨터의 코어메모리 容量을 超過할 경우에는 이 방법을 더 이상 이용할 수 없게 된다. 이 경우 메모리 制限이 큰 小型 컴퓨터일수록 더욱 심각한 문제가 된다. 이러한 문제는 메모리에서는 演算機能만 해 주게 하고, 정보의 저장과 회수는 디스크를 利用하면 해결이 가능해진다. 그러나 기존의 디스크 I/O 方法으로 해 주어서는 효율적으로 처리할 수 없게 된다. 본 연구에서 제시하는 假想메모리 방식에서는 Dimension이 큰 Array를 단일 변수로 취급하여 이 Array가 필요한 곳에서는 메모리에서 처리하는 것과 유사하게 그 내용을 Binary 直接接近화일을 이용한 데이타베이스에 貯藏, 回收할 수 있도록 한다. 이 假想메모리 데이타베이스 Routine에는 2차원 및 1차원 Array를 자유롭게 취급할 수 있게 해 주고 Array의 Address에 상당하는 부분을 直接接近 화일의 기록 번호(Record number)로 대치시켜 준다. 情報의 저장은 Routine 內部에서 궁극적으로 'WRITE'로, 회수는 'READ'를 이용하지만 Routine 外部에는 다만 저장과 회수를 'PUT' 및 'GET' 등의 인지할 수 있는 문자를 사용하여 데이타 베이스 이용을 효율적으로 할 수 있게 만들어 준다. 그림1에는 假想메모리를 이용하여 Dimension이 큰 Array를 처리하는 FORTRAN 코딩 例를 보여 주고 있다. 이와같은 방법을 동원하면 대부분의 Array를 데이타베이스를 이용하여 디스크 I/O로 처리 가능하게 된다. 유한요소 해석 프로그램의 경우 요소망 데이타 등을 메모리 대신 데이타베이스에 저장하게 되면 可用 메모리 공간이 커지게 되고 이 메모리 공간을 모두 방정식 해법시 이용할 수 있게 된다. 따라서 構造解析에서 가장 많은 계산시간을 要하는 연립방정식 해석 부분에 많은 메모리를 할당할 수 있게 되어

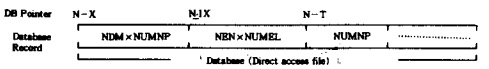
I/O 시간을 절대적으로 줄여 줄 수 있으므로 대형 문제도 효과적으로 해결이 가능해 진다. 그림2에는 기존의 코어메모리를 이용한 경우와 데이타베이스를 사용한 假想메모리를 이용한 경우의 有限要素 프로그램에 대한 動的 貯藏體系방식을 보여주고 있다.

| | 코어메모리 이용방식 | 가상메모리 데이타베이스 이용방식 |
|---|---|---|
| 정보의저장 | DIMENSION X(NDM,NUMNP)<br>DO 100 J=1, NUMNP<br>DO 100 I=1,NDM<br>READ(*,*) XX<br>100 X(I,J)=XX | DO 100 J=1, NUMNP<br>DO 100 I=1, NDM<br>READ(*,*) XX<br>100 PUT=DBASE(XX,I,J,<br>NDM,2,N-X) |
| 정보의회수 | DO 200 J=1, NUMNP<br>DO 200 I=1, NDM<br>200 YY=X(I, J) | DO 200 J=1, NUMNP<br>DO 200 I=1, NDM<br>200 YY=DBASE(GET, I,J,<br>NDM,1, N-X) |

그림1. 가상메모리를 이용한 Array처리의 FORTRAN 코딩예




(1)코어 메모리 이용방식




(2) 데이타 베이스를 이용한 假想메모리 이용방식

그림2. 動的 貯藏體系 비교

지금까지 서술한 방식은 코어 Array를 완전히 디스크상의 假想메모리 데이타베이스에서 처리한 방법이나 필요에 따라서는 假想메모리에서 일부는 코어메모리를 사용하고 나머지는 디스크를 사용하여 Array의 크기가 코어의 限界를 넘으면 자동적으로 디스크를 이용할 수 있게 함으로써 외견상으로는 Dimension의 크기에 제한없이 코어와 디스크를 복합하여 사용할 수 있게 해줄 수 있다.

한편 方程式解法 Routine에서는 1차원 Array만 사용되므로 이 방법을 더욱 간편히 수정하여 사용할 수 있게 되고, 다음 항에서 기술되는 바와 같이 실제 行列消去시에는 可用메모리를 하나의 윈도우로 설정해 놓고 移動메모리 방식으로 운용해 나가기 때문에 行列의 消去할 부분이 코어의 可用메모리에 포함되어 있지 않으면 자동적으로 디스크에서 I/O처리하게 된다.

## 3. 方程式 解析 기법

### 3-1. 對稱 疏散 行列의 解法

線形 聯立方程式의 解를 구하는 方法은 크게 直接解法(direct method)과 反復解法(iteration method)이 있으나[2], 本 硏究에서는 直接解法을 이용한 효과적인 코어內·外 方程式 解法을 개발하였다.

消去해야 할 行列이 對稱일 때에는 그 行列의 요소를 모두 貯藏할 필요가 없으며, 또한 行列에 零의 項이 많이 包含되어 있는 경우(疏散行列 : sparsely populated matrix)라면 貯藏해야 할 容量이 더욱 더 줄어들 것이다. 本 硏究에서는 대부분의 構造解析시 발생하는 對稱 疏散行列을 대상으로 하기로 한다. 方程式 解法은 消去될 行列이 어떤 方式에 의하여 貯藏되었는가에 따라 각각 다른 方法이 사용되며, 行列의 貯藏 方式은 밴드式 貯藏과 스카이라인式 貯藏方法이 있다. 현재까지는 스카이라인 方式으로 貯藏하여 修正 Crout 消去法에 의하여 解를 구하는 方法이 가장 효율적인 方法 중의 하나로 알려져 있다[3].

大型 構造物을 解析하는 경우에는 비록 스카이라인 貯藏 方式을 사용한다 하더라도 消去할 行列을 모두 코어內에 貯藏할 수 없게 되므로 코어外 解法(out-of-core equation solver)이 必然的으로 사용되어야 한다. 지금까지 많은 코어外 方程式解法[4)-7)]들이 개발되었으나 알고리즘이 복잡하고 I/O 시간이 많이 걸려서 계산 효율이 문제가 되어 왔다. 이러한 문제를 해결하기 위하여 本 硏究에서는 스카이라인 解法 중 疏散 行列에 대하여 매우 효과적인 參考文獻3에서 提案된 修正 Crout 消去法의 알고리즘을 이용하여, 假想 메모리를 이용한 메모리 方式을 도입한 매우 효율적인 새로운 코어內 및 코어外 方程式 解法을 제시하였다.

Gauss 消去法을 이용할 경우, 對稱 疏散行列의 消去에 대해서는 상당히 비효율적이다. Gauss消去法에 의한 消去 과정에서는 한 式에 대해 每減

算時마다 係數 行列의 각 項들이 수정되어야 하나 Crout 消去法에서는 하나의 계수에 대하여 全減算 과정을 한꺼번에 수행함으로써, 係數 行列의 각 項들이 단 한번만에 마지막 형태로 修正된다. 또한 對稱 行列의 경우 列別로 消去 과정을 진행할 수 있어서 스카이라인 貯藏 方式에 매우 효과적으로 이용할 수 있다.

式(1)과 같은 線形 方程式에서 行列 [A]가 對稱일 경우 행렬 [A]와 벡터 {b}의 消去는 다음 式에 의하여 列 別로 구할 수 있다.

$$[A]\{x\}=\{b\} \tag{1}$$

$$a_{ij}^{(i-1)}=a_{ij}-\sum_{k=1}^{i-1}\frac{a_{ki}^{(k-1)}a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} \tag{2}$$

$$j=2,\ \cdots,\ N$$

$$i=2,\ \cdots,\ j$$

$$b_i^{(i-1)}=b_i-\sum_{k=1}^{i-1}\frac{a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}}b_k^{(k-1)} \tag{3}$$

$$i=2,\ \cdots,\ N$$

식(2)와 (3)은 行列의 對稱性만 고려했으므로, 그림3과 같은 형태를 가진 零의 項을 많이 包含한 疏散行列에 대해서는 불필요한 計算이 개입된다. 따라서 불필요한 계산을 除去하는 알고리즘이 필요하며, 이는 行列의 스카이라인을 고려함으로써 可能하다. 이를 修正 Crout 消去法이라 하며, 그 알고리즘을 附錄 A에 수록하였다.

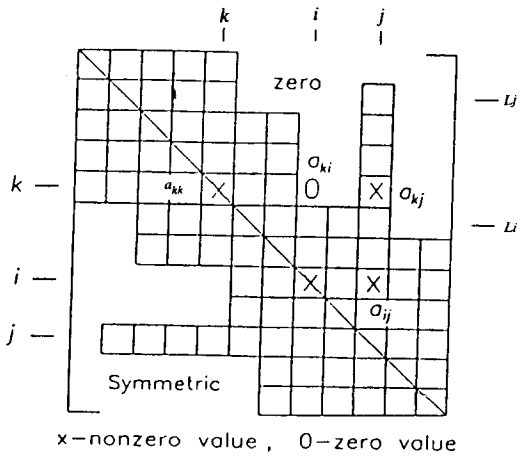



그림3. 對稱 疏散 행렬의 예

3-2 移動 메모리方式 解法(Moving Memory Method)

(1) 行列의 貯藏 形式

大型문제를 해결하기 위하여 現在까지는 주로 스카이라인式 블럭 해법[4]을 이용해 왔다. 旣存의 스카이라인 블럭 解法은 조합된 行列이 블럭別로 대각 成分의 주소와 함께 貯藏해야 하는 번거로움이 있으나 本 硏究에서 提案한 方法에 의한 解法에서는 行列의 貯藏 形式이 코아內 解法에서 하는 것과 거의 다를 바가 없다. 즉, 전항의 假想메모리 데이타베이스 項에서 기술한 바와 같이 行列 成分의 번호가 메모리 內에 들어오면 메모리內에 貯藏을 하고 그 번호가 메모리 바깥에 존재하는 것이라면, 그 行列 成分을 直接接近화일(direct access file)에 그 번호에 대응하도록 貯藏하기만 하면 된다. 그림 4(a)와 같은 行列에 대하여, 기존의 블럭 해법인 경우 디스크에 저장한 형태가 그림 4(b)와 같고 本 硏究에서 사용한 貯藏 形式은 그림 4(c)와 같다. 여기서는 블럭의 개념이 없으며, 總 可用메모리가 60이라고 가정 했을때 92개의 行列 成分중에서 1-60은 코아內에 貯藏이 되고 나머지 61-92는 直接接近화일의 기억번호 61-92에 貯藏된다. 대각 成分의 주소는 전항의 데이타베이스 기법에 의하여 디스크에 貯藏된다.

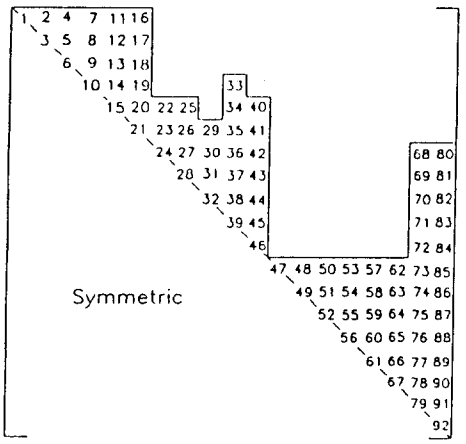
(2) 行列 [A]의 消去 과정

係數 行列[A]의 消去 과정을 나타내는 式(A.2)와 (A.3)을 다시 쓰면,

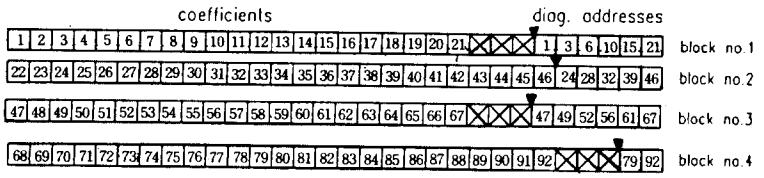
$$a_{ij}^{(i-1)}=a_{ij}-\sum_{k=k_0}^{i-1}\bar{a}_{ki}^{(k-1)}a_{kj}^{(k-1)} \tag{4}$$

$$a_{jj}^{(j-1)}=a_{jj}-\sum_{k=L_j}^{i-1}\frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)} \tag{5}$$

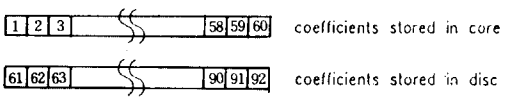
이 되고, 이 消去 과정은 列別로 진행되어 式(4)에서 $a_{kj}^{(k-1)}$는 현재 消去과정이 진행 중인 $j$번째 列에 해당하는 成分이므로 $a_{kj}^{(k-1)}$는 코어內에 있고; $\bar{a}_{kj}^{(k-1)}$는 코어內 또는 코어外에 존재하는 경우가 발생한다. 따라서 式(4)에 대해서는 $\bar{a}_{ki}^{(k-1)}$項을 전술한바와 같이 假想메모리방식을 이용한 自動 I/O 데이타베이스 形式으로 치환한다면, 코어內와 코어外에 대하여 自動的으로 처리할 수 있을 것이

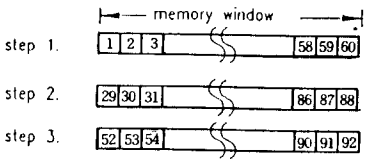



(a) 係數 行列




(b) 스카이라인式 블럭 解法에서의 디스크 貯藏 形式




(c) 移動 메모리 方式 解法에서의 貯藏 形式

**그림4. 조합된 행렬의 저장 형식과 메모리 윈도우의 이동 예**

다. 그림 5는 $\bar{a}_{ij}^{(k-1)}$項에 대한 이러한 방법을 사용한 코딩예를 보여주고 있다. 또한 대각 成分의 式(5)에서는 $a_{kk}^{(k-1)}$만이 문제가 되므로 이 項에 대해서도 같은 방법으로 처리를 한다.

本 硏究에서 提案한 方法에 의하여, 그림 4(a)의 行列[A]를 消去하는 과정을 알기 쉽게 단계적으로 설명하면 다음과 같다.

1) 可用 메모리가 60이므로 移動 메모리의 윈도우(window)가 60이 되고, 行列의 최대 成分이 92이므로 메모리 윈도우가 移動하면서 消去 과정이 진행된다(그림 4(d) 참조). 먼저, 메모리 윈도우에는 行列 [A]의 成分중 1−60이 기억되어 있으므로, I/O없이 1−56 까지의 係數를 修正한다.

2) 修正된 行列 [A]의 成分중 半(1−28)을 디스크에 貯藏한다.

3) 이제 메모리 윈도우를 29−88로 移動한다. (이때 61−88을 디스크로부터 읽어 옴)

4) 行列 成分 중, 다음 修正 차례인 57−79를 修正한 후, 다음 단계로 진행하기 위하여 메모리 윈도우 內의 半(29−54)을 디스크에 貯藏한다.

5) 마지막으로, 남은 係數를 修正하기 위하여 메모리 윈도우를 55−92까지로 移動하여 修正한 후,

이를 디스크에 貯藏한다. 이 단계를 거치면 모든 係數가 修正되어 디스크에 貯藏되게 된다.

6) 行列의 消去 과정이 완료되면 메모리 윈도우는 他목적으로 자유롭게 활용이 가능해 진다. 벡터 {b}를 消去하는 과정을 효율적으로 처리하기 위해 이 메모리에 벡터 {b}를 기억시킨다. 可用메모리 容量에서 벡터 {b} 만큼을 제외한 부분은 다시 修正된 行列을 成分을 기억시켜 벡터 {b}의 消去時 활용토록 한다.

既存의 스카이라인 블럭 解法[4]에서는 3개의 디스크 화일이 있어야 하나, 移動메모리 方式 解法에서는 直接接近화일을 사용함으로써, 하나의 디스크 화일만이 필요하다. 또한 消去하는 과정에서 필요로 하게 되는 行列의 成分이 현재 메모리 윈도우內 있는지, 없는지를 그림 5에서 보는 바와 같은 方法에 의해 처리함으로써, 그 알고리즘이 매우 간단하며, 코어內와 코어外에 대하여 自動으로 解決할 수 있다.

| 既存의 코어內 解法 알고리즘 | 假想 메모리데이타베이스를 이용한 코어內·外 알고리즘 |
|---|---|
| DO 30 K=KF, KL<br>30 AA=AA+A(K)*A(JJ+K)<br>A(JIA)=A(JIA)-AA | DO 30 K=KF, KL<br>30 AA=AA+A(K)*GET(A,<br>+ (JJ+K))<br>A(JIA)=A(JIA)-AA |

그림5. 가상메모리 기법을 이용한 $\bar{a}_{ki}^{(k-1)}$항의 계산

(3) 벡터 (b)의 消去 과정

먼저 假想메모리 데이타베이스에 貯藏되어 있는 벡터{b}를 코어內로 읽어 온다. 行列[A]의 成分 중 일부가 코어에 담겨 있으므로, 벡터 {b}를 修正하는 式에서 $\bar{a}_{ki}^{(k-1)}$과 $a_{ii}^{(i-1)}$에 대하여 그림 5와 같은 方法으로 코어 內·外에서 自動 처리한다. 앞의 行列의 각 成分들이 단 한번씩만 필요하므로 메모리를 移動시켜 消去과정을 진행할 필요가 없다. 따라서 벡터{b}의 消去 과정에 대한 알고리즘은 array의 自動 처리를 제외하고는 既存의 코어內 解法의 알고리즘과 완전히 같다.

(4) 逆代入(Back Substitution)과정

이 과정에서도 벡터{b}는 여전히 코어內에 있고, 係數行列의 각 成分들이 단 한번씩만 필요하므로 메모리를 移動시키는 기법이 필요없다. 따라서 $\bar{a}_{ki}^{(k-1)}$을 코어內·外로 自動 처리하는 것 외에는 코어內 解法의 알고리즘과 같다.

(5) 提案된 方法의 잇점

本 研究에서 提案한 假想메모리를 이용한 移動메모리 方式 解法의 잇점은 다음과 같다.

1) 간편한 알고리즘:既存의 코어外 解法에서는 복잡한 I/O 오퍼레이션으로 인하여 그 알고리즘과 프로그램의 코딩이 상당히 복잡하지만, 本 研究에서 提案한 方法은 I/O 오퍼레이션이 간단하기 때문에 알고리즘이 코어內 解法처럼 매우 간단하여 계산 효율을 크게 향상시킨다.

2) 코어內와 코어外 解를 동시에 解決:既存의 解法은 코어內와 코어外의 解法을 별도로 취급하여야 하나, 本 研究에서 提案한 方法을 사용하면 可用메모리의 크기에 따라 코어內와 코어外가 自動的으로 구분되어 解를 구할 수 있다.

3) 行列 貯藏 方式의 개선:既存의 블럭 解法에서는 行列의 貯藏 方式이 블럭 단위로 이루어져 복잡하여 剛性行列의 조합시 매우 번거롭고 비효율적이나, 本 研究에서 提案한 假想메모리 데이타베이스 방법을 이용한 解法에서는 行列의 조합과 貯藏 方式이 매우 간단하다.

## 4. 例題 解析

本 研究에서 개발한 解法의 효율성을 알아보기 위하여 그림 6에 보이는 SMC 물탱크 모델에 대하여 静的 解析을 수행하였다. 이 모델은 方程式

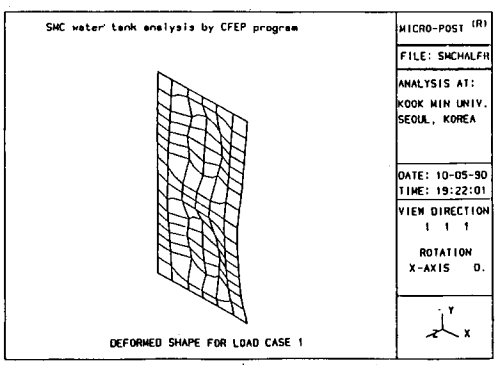



그림6. SMC 물탱크 모델의 變形後 모델도

표1. SMC 플탱크 모델에 대한 方程式 解의 계산 시간

| PC 기종 | Compaq 486/25 PC | Compaq 386/33 PC | AT 286/16 PC |
|---|---|---|---|
| 剛性行列의 減算 時間 | 3분 44초 | 4분 53초 | 21분 5초 |
| 荷重벡터의 減算 및 逆代入 時間 | 1분 9초 | 1분 22초 | 4분 15초 |

數가 1,338이며 剛性行列의 總 크기가 130,657(1,045KB)가 되는 문제이다. PC에서 배정도 실수(double precision)로 總 可用 array를 29,000(232KB)으로 하여 本 硏究에서 提案한 方法에 의하여 解를 구했을 때의 計算 時間을 표1에 나타내었다. 本 論文에서는 수록하지 않았으나, 方程式 數가 4,000 정도 되는 보다 큰 大型 問題에 本 硏究에서 提案한 기법을 적용했을 경우에도 대단히 효과적으로 解決할 수 있었다.

## 5. 結論

本 硏究에서는 制限된 코어메모리에서 大型문제를 해결하기 위하여 디스크를 마치 메모리처럼 이용할 수 있는 假想메모리 데이타베이스 기법을 개발하였다. 이 기법과 아울러 移動메모리 方式을 使用하여 大型 有限要素 解析時 필연적으로 요구되는 코어外 方程式 解法에 대해 효과적인 알고리즘을 개발하였다. 이 기법은 스카이라인 方式으로 貯藏된 對稱 疏散行列에 대해 가장 효율적인 解法 중의 하나로 알려진 修正 Crout法을 이용하였고, 旣存의 코어 外 解法들에 비해 알고리즘이 코어內 解法처럼 매우 간단하고 코어內 및 코어 外 解法이 自動的으로 구분 처리될뿐만 아니라 코딩이 간결하여 계산효율을 상당히 향상시켰다.

예제 解析으로부터 可用 메모리 容量이 매우 작은 PC와 같은 小型 컴퓨터에서도 大型 問題를 효과적으로 解決할 수 있음을 알 수 있었으며 제시된 기법의 효율성을 입증하였다. 제시된 假想메모리 데이타베이스 기법과 方程式 解法은 大型컴퓨터나 小型 컴퓨터에 관계없이 적용할 수 있는 기법으로 향후 널리 활용될 수 있을 것으로 기대된다.

## 參考文獻


(1) 이성우, "小型 컴퓨터에서 大規模 構造 解析을 위한 효율적인 데이타베이스 技法 개발", 대한토목학회 학술 발표회 논문집, 84-88(1990).

(2) A.W.Al-Khafaji and J.R.Tooley Numerical Methods in Engineering Practice. Holt, Rinehart and Winston. Inc., New York, (1986).

(3) D.P.Mondkar and G.H.Powell. "Towards Optimal In-Core Equation Solving," Computers & Structures 4. 531-548(1974).

(4) D.P.Mondkar and G.H.Powell, "Large Capacity Equation Solver for Structural Analysis," Computers & Structures 4, 699-728 (1974).

(5) G.Cantin, "An equation solver of very large capacity." Int. J. Numerical Meth. Engng 3, 379-388(1971).

(6) N.Ida and W.Load "Solution of Linear Equations for Small Computer Systems," Int. J. Numerical Meth. Engng 20, 625-641(1984).

(7) A. Recuero and J.P. Gutierrez, "A direct Linear System Solver with small core requirement." Int. J. Numerical Meth. Engng 14, 633-645(1979).


## 附錄 A 修正 Crout 消去法 알고리즘

다음과 같이 對稱인 線形方程式을 생각해 보자.

$$[A]\{x\}=\{b\} \tag{A.1}$$

여기서

$[A]=[a_{ij}]=N\times N$ 對稱 疏散 行列

$\{x\}=\{x_i\}=N\times 1$ 未知 벡터

$\{b\}=\{b_i\}=N\times 1$ 旣知 벡터

행렬 [A]를 消去하는 과정

$$a_{ij}^{(i-1)}=a_{ij}-\sum_{k=k_0}^{i-1}\bar{a}_{ki}^{(k-1)}a_{kj}^{(k-1)} \tag{A.2}$$

$$a_{jj}^{(j-1)}=a_{jj}-\sum_{k=L_1}^{j-1}\frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)} \tag{A.3}$$

여기서

$j=2, \cdots, N; i=L_j+1, \cdots, j-1$

$k_0=max(L_i, L_j)$

$\bar{a}_{ki}^{(k-1)}=a_{ki}^{(k-1)}/a_{kk}^{(k-1)}$

$L_j$=行列 [A]의 $j$번째 列에서 처음으로 零이 아닌 項이 나올 때의 번호

벡터 {b}를 消去하는 과정

$$b_i^{(i-1)}=b_i-\sum_{k=k_0}^{i-1}\bar{a}_{ki}^{(k-1)}b_k^{(k-1)} \qquad (A.4)$$

여기서

$i=L_{N+1}+1, \cdots, N$

$k_0=max(L_i, L_{N+1})$

$L_{N+1}$=벡터 {b}에서 처음으로 零이 아닌 項이 나올 때의 行 번호

$$b_i^{(i-1)}=b_i^{(i-1)}/a_{ii}^{(i-1)} \quad \} \quad i=L_{N+1}, \cdots, N \qquad (A.5)$$

逆代入 과정

$$x_k=b_k^{(k-1)} \quad \} \quad k=1, \cdots, N \qquad (A.6)$$

$$x_k=x_k-\bar{a}_{ki}^{(k-1)}x_i \quad \left.\begin{matrix} i=N, N-1, \cdots, 2 \\ k=L_i, L_i+1, \cdots, i-1 \end{matrix}\right\} \qquad (A.7)$$

식 (A.3)이 완료되면 $a_{kj}^{(k-1)}$項은 $\bar{a}_{kj}^{(k-1)}=a_{kj}^{(k-1)}/a_{kk}^{(k-1)}$로 대치된다. 式 (A.4)−(A.7)은 많은 하중에 대하여 독립적으로 반복하여 적용될 수 있다.

(접수일자 : 1991. 3. 15)