한국전산구조공학회
1990년 가을 학술발표회 논문집

大型 有限要素 解析을 위한 효과적인 移動 메모리 方式 스카이라인 方程式 解法의 개발

Development of Efficient Moving Memory Column Solver
for Large Finite Element Analysis

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

ABSTRACT

For the analysis of structures, specifically if it is large-scale, in which case it can not be solved within the core memory, the majority of computation time is consumed in the solution of simultaneous linear equation.

In this study an efficient in- and out-of-core column solver for sparse symmetric matrix utilizing memory moving scheme is developed. Compare with existing blocking methods the algorithm is simple, therefore the coding and computational efficiencies are greatly enhanced. Upon available memory size, the solver automatically performs solution within the core or outside core. Analysis example shows that the proposed method efficiently solve the large structural problem on the small-memory microcomputer.

1. 序論

컴퓨터를 이용한 構造解析에서 많은 計算 時間을 요하는 부분은 聯立方程式의 解를 구하는 부분이며, 특히 메모리內에서 解決할 수 없는 복잡한 大型 構造物에 대해서는 解를 구하는 時間이 構造解析의 대부분을 차지하게 된다.

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, 本 硏究에서는 直接解法을 이용한 효과적인 코아內·外 方程式 解法을 개발하였다. 가장 일반적으로 사용되는 直接解法은 가우스 消去法(Gauss elimination method)이며, 이 가우스 消去法을 근간으로 한 三角 分解法(LU decomposition), Crout 消去法, Cholesky 方法, Frontal 方法[1] 등의 여러가지 方法들이 사용되고 있다.

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(이후 疏散 行列 : sparsely populated matrix)라면 貯藏해야 할 容量이 더욱 더 줄어들 것이다. 本 硏究에서는 대부분의 構造解析時 발생하는 對稱 疏散 行列을 대상으로 하기로 한다. 方程式 解法은 消去될 行列(FEM에서는 剛性行列)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. 스카이라인 方式으로 貯藏하여 修正 Crout 消去法[2]에 의하여 解를 구하는 方法이 가장 효율적인 方法 중의 하나로 알려져 있다.

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 코아內에 貯藏할 수 없게 되므로 코아外 解法(out-of-core equation solver)이 필연적으로 사용되어야 한다.

本 硏究에서는 스카이라인 解法 중 疏散 行列에 대하여 매우 효과적인 參考 文獻 [2]에서 提案된 修正 Crout 消去法의 알고리즘을 이용하여, 移動 메모리 方式을 도입한 매우 효율적인 새로운 코아內 및 코아外 方程式 解法을 개발하였다.

2. 對稱 疏散 行列의 解法

가우스 消去法을 이용할 경우, 對稱 疏散 行列의 消去에 대해서는 상당히 비효율적이다. 가우스 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 修正되어야 하나, Crout 消去法에서는 하나의 係數에 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.

* 국민대학교 토목공학과 조교수
** 한국과학기술원 토목공학과 조교수
*** 한국과학기술원 토목공학과 박사과정

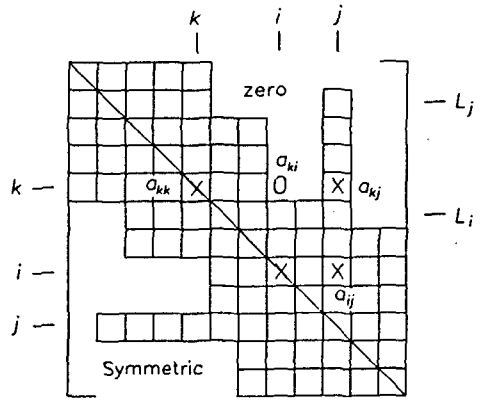
式 (1)과 같은 線形 方程式에서 行列 [A]가 對稱일 경우 對稱 行列 [A]와 벡터 {b}의 消去는 다음 式에 의하여 列 別로 구할 수 있다.

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

$$a_{ij}^{(i-1)} = a_{ij} - \sum_{k=1}^{i-1} \frac{a_{ki}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} \left.\begin{array}{l} j = 2,\ldots,N \\ i = 2,\ldots,j \end{array}\right\} \tag{2}$$

$$b_i^{(i-1)} = b_i - \sum_{k=1}^{i-1} \frac{a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)} \left.\begin{array}{l} \\ i = 2,\ldots,N \end{array}\right\} \tag{3}$$

式 (2)와 (3)은 行列의 對稱性만 고려했으므로, 그림 1과 같은 형태를 가진 零의 項을 많이 包含한 疏散 行列에 대해서는 불필요한 計算이 개입된다. 따라서 불필요한 計算을 除去하는 알고리즘이 필요하며, 이는 行列의 스카이라인을 고려함으로써 可能하다. 이를 修正 Crout 消去法[2] 이라 하며, 그 알고리즘을 附錄 A에 수록하였다.



x-nonzero value, 0-zero value

그림 1. 對稱 疏散 行列의 예

3. 大型 問題를 위한 코아外 解法

3.1 既存의 블럭 解法(Blocking Method)

現在까지 많이 사용되는 블럭 解法에는 밴드式 블럭 解法과 스카이라인式 블럭 解法[3]이 있다. 밴드式 블럭 解法은 밴드 형태(行列이 對稱이면 半밴드폭)의 行列을 몇 개의 블럭으로 나누어 貯藏하며, 스카이라인式 블럭 解法은 스카이라인 하부에 있는 요소들을 몇 개의 스카이라인 블럭으로 나누어 貯藏한다. 일반적으로 밴드式 블럭 解法보다는 스카이라인式 블럭 解法이 貯藏해야 할 行列의 容量이 작고, 解를 구하는 과정의 計算量도 작게 된다. 또한 밴드式 블럭 解法에서는 모든 블럭이 같은 數의 行(方程式)을 包含하고 있으나, 스카이라인式 블럭 解法은 밴드內의 불필요한 零을 貯藏하지 않으므로 블럭 別로 서로 다른 數의 列을 包含할 수 있어서 스카이라인 블럭 解法이 보다 효율적이다. 本 硏究에서 提案한 移動 메모리 方式 方程式 解法과의 비교를 위해서 既存의 스카이라인 블럭 解法에 대하여 간략히 설명하기로 한다.

스카이라인式 블럭 解法에서의 行列의 貯藏

코아의 可用 메모리가 60이라고 假定했을 때, 그림 2(a)에서 보는 行列을 블럭 別로 디스크에 貯藏한 예가 그림 2(b)에 나타나 있다. 이때 行列 [A]를 消去하는 과정에서 두개의 블럭을 디스크에서 코아內에 불러 와야 하므로, 블럭당 許容되는 메모리 容量은 30이 된다. 行列의 대각 成分의 주소도 블럭 내에 貯藏되어야 하므로 하나의 블럭에는 行列의 成分과 대각 成分의 住所가 包含되게 된다.

既存의 스카이라인式 블럭 解法에서는 消去하고자 하는 行列을 미리 블럭別로 디스크에 貯藏하여야 하며, 두개의 디스크 파일(消去되지 않은 行列이 貯藏된 디스크 파일과 消去된 行列이 貯藏된 디스크 파일)과 대각 成分만을 기억하는 디스크 파일이 필요하게 된다. 이러한 디스크 파일 오퍼레이션 때문에 코아外 解法의 알고리즘이 상당히 복잡하다는 단점을 안고 있다.

3.2 移動메모리 方式 解法(Moving Memory Method)

(1) 行列의 貯藏 形式

既存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, 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파일(direct access file)에 그 번호에 대응하도록 貯藏하기만 하면 된

다. 그림 2(a)와 같은 行列에 대하여, 本 硏究에서 사용한 貯藏 形式을 그림으로 나타내면 그림 2(c)와 같다. 여기서는 블럭의 개념이 없으며, 總 메모리가 60이라고 했으므로 92개의 行列 成分 중에서 1-60은 코아內에 貯藏이 되고 나머지 61-92는 直接接近파일의 기억 번호 61-92에 貯藏된다. 대각 成分의 주소는 데이타베이스 기법[4] 에 의하여 貯藏한다.

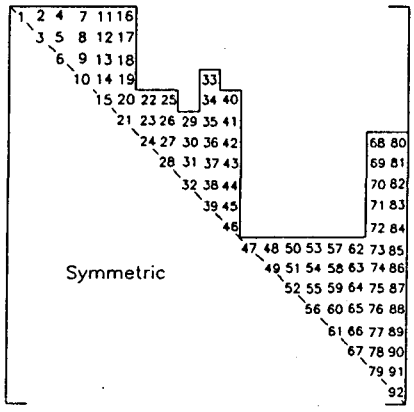
(2) 行列 [A]의 消去 과정

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

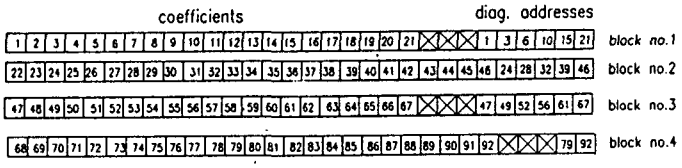
$$a_{ij}^{(i-1)} = a_{ij} - \sum_{k=k_0}^{i-1} \bar{a}_{ki}^{(k-1)} a_{kj}^{(k-1)} \qquad (4)$$

$$a_{jj}^{(j-1)} = a_{jj} - \sum_{k=l_j}^{j-1} \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \qquad (5)$$

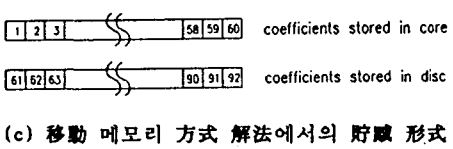
이 되고, 이 消去 과정은 列 別로 진행되어 式 (4)에서 $a_{kj}^{(k-1)}$는 현재 消去 과정이 진행 중인 j 번째 列에 해당하는 成分이므로 $a_{kj}^{(k-1)}$는 코아內에 있고, $\bar{a}_{ki}^{(k-1)}$는 코아內 또는 코아外에 존재하는 경우가 발생한다. 따라서 式 (4)에 대해서는 $\bar{a}_{ki}^{(k-1)}$項을 自動 I/O 데이타베이스 形式으로 치환한다면, 코아內와 코아外에 대하여 自動的으로 처리할 수 있을 것이다. 그림 3은 $\bar{a}_{ki}^{(k-1)}$項에 대한 自動 I/O 처리를 보여주고 있다. 또한 대각成分의 式 (5)에서는 $a_{kk}^{(k-1)}$만이 문제가 되므로 이 項에 대해서도 自動 I/O 처리를 한다.



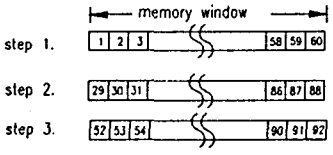
(a) 係數 行列



(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



(d) 메모리 윈도우의 단계적 移動

그림 2. 조합된 行列의 貯藏 形式과 메모리 윈도우의 移動 例

既存의 코아內 解法 알고리즘	自動 I/O 데이타베이스를 이용한 코아內·外 알고리즘
DO 30 K=KF,KL 30 AA=AA+A(K)*A(JJ+K) A(JIA)=A(JIA)-AA	DO 30 K=KF,KL 30 AA=AA+A(K)*GET(A,(JJ+K)) A(JIA)=A(JIA)-AA

그림 3. $\bar{a}_{ki}^{(k-1)}$項의 自動 I/O 데이타베이스 처리

本 硏究에서 提案한 方法에 의하여, 그림 2(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 2(d) 참조). 먼저, 메모리 윈도우에는 行列 [A]의 成分 중 1-60이 기억되어 있으므로, I/O 없이 1-56 까지의 係數를 修正한다.
2) 修正된 行列 A의 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 타목적으로 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의 成分을 기억시켜 벡터 {b}의 消去時 활용토록 한다.

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

(3) 벡터 {b}의 消去 과정

먼저 데이타베이스에 貯藏되어 있는 벡터 {b}를 코아內로 읽어 온다. 行列 [A]의 成分 중 일부가 코아에 담겨 있으므로, 벡터 {b}를 修正하는 式인 式 (A.4), (A.5)에서 $\bar{a}_{ki}^{(k-1)}$과 $a_{ii}^{(i-1)}$에 대하여 그림 3과 같은 方法으로 코아 內·外에서 自動 처리한다. 앞의 係數 行列의 消去 과정과는 달리 벡터 {b}의 消去 과정에서는 係數 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 코아內 解法의 알고리즘과 완전히 같다.

(4) 逆代入(Back Substitution) 과정

이 과정에서도 벡터 {b}는 여전히 코아內에 있고, 係數 行列의 각 成分들이 단 한번씩만 필요하므로 메모리를 移動시키는 기법이 필요없다. 따라서 式 (A.7)의 $\bar{a}_{ki}^{(k-1)}$을 코아內·外로 自動 처리하는 것 외에는 코아內 解法의 알고리즘과 같다.

(5) 提案된 方法의 잇점

本 硏究에서 提案한 移動메모리 方式 解法의 잇점은 다음과 같다.

1) 간편한 알고리즘 : 既存의 코아外 解法에서는 복잡한 I/O 오퍼레이션으로 인하여 그 알고리즘과 프로그램의 코딩이 상당히 복잡하지만, 本 硏究에서 提案한 方法은 I/O 오퍼레이션이 간단하기 때문에 알고리즘이 코아內 解法처럼 매우 간단하다.

2) 코아內와 코아外 解를 동시에 解決 : 既存의 解法은 코아內와 코아外의 解法을 별도로 취급하여야 하나, 本 硏究에서 提案한 方法을 사용하면 可用 메모리의 크기에 따라 코아內와 코아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 方法에 의한 解法에서는 行列의 조합과 貯藏 方式이 매우 간단하다.

4. 比較 考察

本 研究에서 개발한 移動 메모리 方式 解法의 효율성을 알아보기 위하여 그림 4에 보이는 SMC 물탱크 모델에 대하여 靜的 解析을 수행하였다. 이 모델은 方程式 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 研究에서 提案한 方法에 의하여 解를 구했을 때의 計算 時間을 표 1에 나타내었다. 本 論文에서는 수록하지 않았으나, 方程式 數가 4,000 정도 되는 보다 큰 大型 問題에 本 研究에서 提案한 기법을 적용했을 경우에도 대단히 효과적으로 解決할 수 있었다.

5. 結論

本 研究에서는 大型 有限要素 解析時 필연적으로 요구되는 코아外 方程式 解法에 대해 移動 메모리 方式을 적용한 효과적인 알고리즘을 개발하였다. 이 기법은 스카이라인 方式으로 貯藏된 對稱 疏散 行列에 대해 적용되며, 既存의 블럭 方式에 비해 알고리즘이 코아內 解法처럼 매우 간단하고, 코아內 및 코아外 解法이 自動的으로 구분되어 매우 효율적으로 解를 구할 수 있는 잇점이 있다.

예제의 解析으로부터 可用 메모리 容量이 매우 작은 PC와 같은 소형 컴퓨터에서도 大型 問題를 효과적으로 解決할 수 있음을 알 수 있었으며 제시된 기법의 효율성을 입증하였다. 또한 제시된 方程式 解法은 有限要素뿐만 아니라 數値解析을 이용한 각종 構造解析 프로그램에도 사용될 수 있으며, 大型 컴퓨터나 소형 컴퓨터에 관계없이 적용할 수 있는 기법으로 향후 널리 활용될 수 있을 것으로 기대된다.

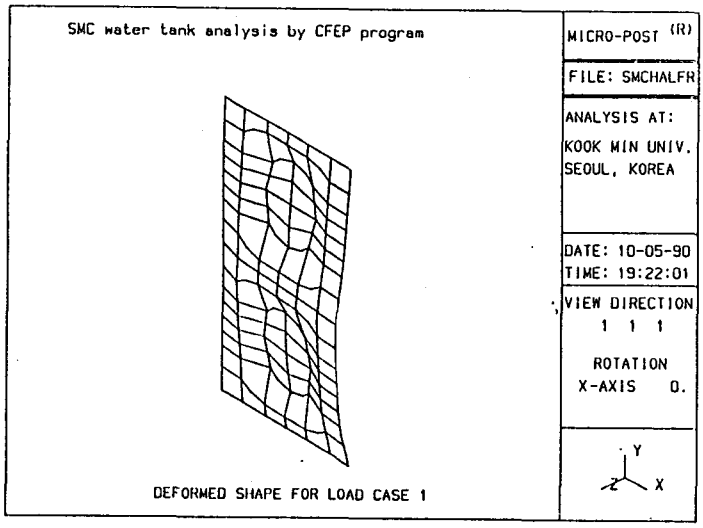


그림 4. SMC 물탱크 모델의 變形後 모델圖

표 1. SMC 물탱크 모델에 대한 方程式 解의 計算 時間

PC 기종	Compacq 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

6. 參考 文獻

[1] A.W.Al-Khafaji and J.R.Tooley *Numerical Methods in Engineering Practice*, Holt, Rinehart and Winston, Inc., New York, 1986

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

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

[4] 이 성우, "小型 컴퓨터에서 大規模 構造 解析을 위한 효율적인 데이타베이스 技法 개발," 대한토목학회 학술 발표회 개요집, 1990

부록 A. 修正 Crout 消去法 알고리즘

다음과 같은 對稱인 線形方程式을 생각해 보자.

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

여기서

$[\mathrm{A}] = [a_{ij}] = N \times N$ 對稱 疏散 行列
$\{\mathrm{x}\} = \{x_i\} = N \times 1$ 未知 벡터
$\{\mathrm{b}\} = \{b_i\} = N \times 1$ 既知 벡터

<u>行列 [A]를 消去하는 과정</u>

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

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

여기서

$j = 2, \ldots, N;\ i = L_j + 1, \ldots, j-1$
$k_0 = \max(L_i, L_j)$
$\overline{a}_{ki}^{(k-1)} = a_{ki}^{(k-1)} / a_{kk}^{(k-1)}$
L_j = 行列 [A]의 j번째 列에서 처음으로 零이 아닌 項이 나올 때의 번호

<u>벡터 {b}를 消去하는 과정</u>

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

여기서

$i = L_{N+1} + 1, \ldots, N$
$k_0 = \max(L_i, L_{N+1})$
L_{N+1} = 벡터 b에서 처음으로 零이 아닌 項이 나올 때의 行 번호

$$b_i^{(i-1)} = b_i^{(i-1)} / a_{ii}^{(i-1)} \Big\} \ i = L_{N+1}, \ldots, N \tag{A.5}$$

<u>逆代入 과정</u>

$$x_k = b_k^{(k-1)} \Big\} \ k = 1, \ldots, N \tag{A.6}$$

$$x_k = x_k - \overline{a}_{ki}^{(k-1)} x_i \Bigg\} \begin{array}{l} i = N, N-1, \ldots, 2 \\ k = L_i, L_i + 1, \ldots, i-1 \end{array} \tag{A.7}$$

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