Journal of the Korea Institute for Structural Maintenance and Inspection
Vol. 18, No. 2, March 2014, pp.030-037
http://dx.doi.org/10.11112/jksmi.2014.18.2.030

pISSN 2234-6937
eISSN 2287-6979

블록화기법을 이용한 대형구조물의 해석방법

An Analysis Method of Large Structure Using Matrix Blocking

정 성 진[1)] 이 민 섭[2)*]
Sung-Jin Jung Min-Sup Lee

Abstract

In this study, we studied how to perform the structural analysis which need a large-capacity flash memory with the computer program when the flash memory storage of a personal computer has no enough room for the analysis of structure. As one of the solutions of this problem, the blocking method of stiffness matrix, which is a method that stiffness matrix is divided by a few blocks and each block is sequentially used for the calculation of matrix decomposition, is proposed and an algorithm available in computer program is derived on the method. Finally, A structural analysis program (sNs) based on this study is developed and the correctness and efficiency of the algorithm is founded through some examples which are fundamental in structural analysis.

Keywords : Structural analysis, Massive matrix decomposition, Blocking method, Stiffness matrix, Algorithm, Structural analysis program

1. 서 론

최근 상용 구조해석 프로그램에서 제공하고 있는 그래픽 인터페이스 (Graphic Interface)가 편리해짐에 따라 구조해석 모형을 작성하는 방식에서 많은 변화가 일어나고 있다. 특히, 과거에는 구조물의 거동을 잘 나타낼 수 있으면서 단순화 된 해석모형을 이용하여 구조해석을 수행하였으나, 최근에 들어서는 구조물을 최대한 세밀하게 모형화하여 해석을 수행하는 방향으로 구조해석 동향이 변화하고 있다.

이러한 경향은, 구조해석 시 고려해야 할 절점 수 및 부재수를 기하급수적으로 증가시키게 되어, 결과적으로 상용 구조해석 프로그램의 안정적인 연산 수행을 위해서는 일시에 많은 양의 메모리 (Memory)를 필요로 하는 문제점을 일으키게 되었다.

일부에서는 개인용 컴퓨터 (Personal Computer)의 메모리 용량이 비약적으로 증가하고 있으므로 이러한 문제는 해소될 수 있는 문제로 보고 있다. 이러한 판단에 따라 이들은 향후 구조실무자들이 더욱 상세한 구조해석 모형을 작성할 수 있을 것이며, 또한 이러한 구조물을 해석하는데 있어서 아무런 제약을 받지 않을 것으로 판단하고 있다.

그러나 사실적인 구조해석 모형화 경향의 심화는 가용할 수 있는 메모리의 더욱 많은 부분을 전·후처리를 위한 그래픽 인터페이스에 할당하도록 요구하게 될 것이다. 또한 다양한 프로그램들을 통하여 여러 가지 작업이 동시에 수행되어야하는 향후의 업무환경은 상용 구조해석 프로그램이 가용할 수 있는 메모리의 크기를 암묵적으로 제한할 수밖에 없을 것이다.

이러한 이유로 인해 개인용 컴퓨터를 이용한 구조해석은 제한적인 메모리를 사용하여 이루어질 수밖에 없으며, 부족한 메모리 공간상에서 대용량 행렬연산 수행을 위한 최적의 방법론을 찾는 것은 여전히 해결되어야 할 문제점으로 인식되고 있다.

전술된 문제점들을 해결하기 위한 방법들로는 부분구조법 (Substructuring method) (Przemieniecki, 1963), 병렬연산 (Parallel

1) 정회원, 한남대학교 건축공학과 교수
2) 정회원, 한남대학교 건축공학과 박사과정, 교신저자

* Corresponding author : alstjq1@hanmail.net
• 본 논문에 대한 토의를 2014년 4월 30일까지 학회로 보내주시면 2014년 5월호에 토론결과를 게재하겠습니다.

computing) (Patterson and John, 1998), 블록화기법 (Blocking method) (Bathe, 1996) 등 여러 가지 방법들이 있으며, 이들에 대해서는 여전이 여러 분야에서 활발히 연구되고 있다.

본 연구에서는, 전술된 제 방법들 중 강성행렬을 가용 가능한 메모리 범위 내에서 몇 개의 블록 (Block)으로 나누고, 개별 블록으로 나누어진 강성행렬을 순차적으로 메모리에 저장하여 연산을 수행함으로써 전체 구조물에 대한 구조해석을 원활하게 수행하는 방법에 대하여 연구하고 있다. 또한 제시된 방법론을 바탕으로 유한요소해석 (Finite Element Analysis) 프로그램을 구현하였으며, 예제 검증을 통하여 제안된 방법론의 효율성을 검증하고 있다.

2. 블록화기법 알고리즘

컴퓨터 프로그램을 이용한 구조해석에서 풀어야할 평형방정식의 가장 간단한 형태는 식 (1)과 같다.

$$KU = R \tag{1}$$

여기서, K = 강성행렬 (Stiffness Matrix)
U = 변위벡터 (Displacement Vector)
R = 하중벡터 (Load Vector)

식 (1)과 같은 방정식을 풀기 위한 컴퓨터프로그램을 작성할 때, 가장 먼저 수행하여야 할 작업은 컴퓨터 메모리에 K 행렬과 R 벡터를 올릴 수 있느냐의 문제를 판단하는 것이라 볼 수 있다. 이때 컴퓨터 메모리가 충분하지 못하여 이들 행렬과 벡터를 한 번에 올릴 수 없다면, 이를 해결하기 위한 방법을 찾아야 하는 문제가 발생한다. 이러한 문제를 해결하기 위한 방법들은 여러 가지가 있을 수 있는데, 본 연구에서는 블록화기법 (Blocking Method)을 이용하여 문제를 해결하기 위한 방법에 대해 연구하고 있다.

블록화기법을 이용하여 대용량의 메모리가 요구되는 문제를 풀기 위한 과정을 크게 나누어 보면 Table 1과 같이 정리

Table 1 Process of Blocking Method

Step	Contents
1	calculation of memory capacity
2	calculation of number of blocks and storage of information about coupling blocks
3	matrix decomposition per block and solution of simultaneous equations

될 수 있다. Table 1에서의 첫 번째 과정은 컴퓨터 프로그램 작성 시 사용된 프로그래밍 언어 (Programming Language)와 관련이 있다. 즉, 구조해석 프로그램이 실행 (Run)되는 시점에서 얼마나 큰 메모리를 사용할 수 있는지를 알아내기 위한 방법은 프로그래밍 컴파일러에 따라 조끔씩 다르므로 각자의 형편에 맞게 처리하면 된다. 본 연구에서 사용한 인텔포트란 (Intel Fortran)의 경우에는 포인터 (Pointer)와 메모리 할당 (Allocate) 문장을 이용하면, 컴퓨터 프로그램이 실행되는 시점에서 사용가능한 메모리의 크기를 정할 수 있다.

따라서, 본 연구에서는 Table 1에 나타난 과정 중 두 번째 및 세 번째 방법에 대해서 연구하고 있다.

2.1 행렬분해법의 분석

본 연구에서는, 행렬분해 (Matrix Decomposition)를 위한 제 기법 중 가우스 소거법 (Gauss Elimination Method) (Bathe, 1996)을 응용한 LDL^T법에 대하여 연구하고 있다.

LDL^T법을 이용하여 평형방정식의 해를 구하는 과정은 식 (2a)~식 (2d)와 같다.

$$KU = LDL^TU = R \tag{2a}$$

$$LV = R \tag{2b}$$

$$DL^TU = V \tag{2c}$$

$$L^TU = D^{-1}V \tag{2d}$$

여기서, L = 하부삼각행렬 (Lower Triangle Matrix)
L^T = 상부삼각행렬 (Upper Triangle Matrix)
D = 대각행렬 (Diagonal Matrix)

LDL^T법을 이용한 구조해석 프로그램들은 일반적으로 대각행렬 D와 상부삼각행렬 L^T를 구한 다음, 이를 이용하여 평형방정식의 해를 구하는 방식을 채택하고 있다.

열소거법 (Active Column Solver) (Farhat and Wilson, 1988)을 이용할 경우, 강성행렬은 Fig. 1과 같이 메모리에 저장된다. 이와 같이 저장된 강성행렬에 대하여 LDL^T법을 이용한 행렬분해를 수행하면, Fig. 2와 같아진다. Fig. 2에 나타난 l_{ij}, d_{jj}를 구하기 위한 알고리즘을 정리하면, Table 2와 같다.

여기서 주목해야 할 점은, 식 (4a)에서 볼 수 있는 바와 같

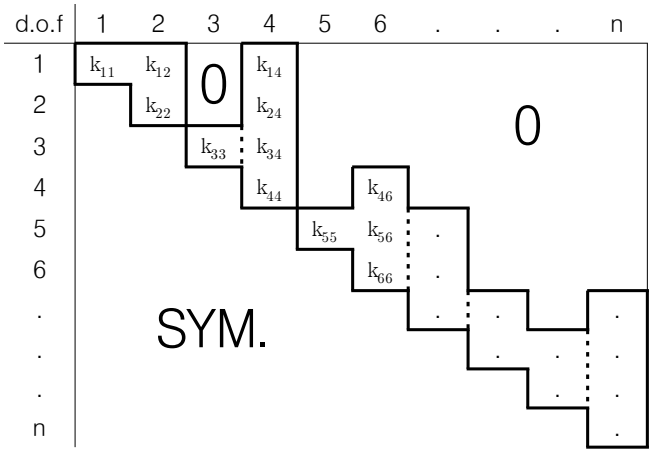


Fig. 1 Storage of Stiffness Matrix

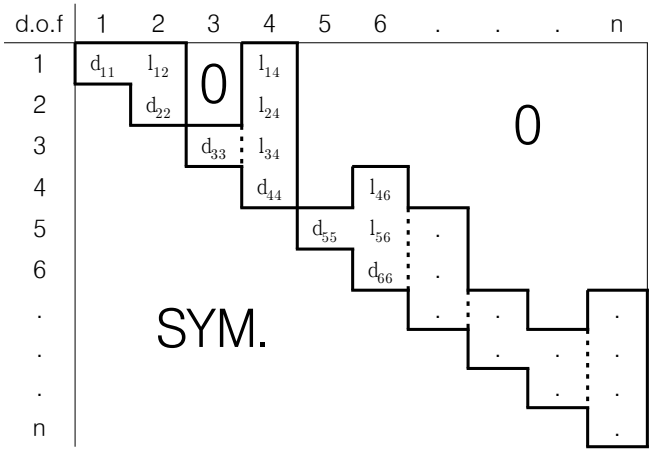


Fig. 2 Stiffness Matrix Decomposition by $\mathbf{LDL}^{\mathrm{T}}$ Method

Table 2 Algorithm for the Calculation of the Elements l_{ij}, d_{ij} in the j−th Column

Step	Calculation
1	• calculation of 1st column(j=1) $d_{11} = k_{11}$
2	• after 2nd column(j=2, 3, 4, ⋯, n) 1) calculation of intermediate quantity g_{ij} $g_{m_j,j} = k_{m_j,j}$ (3a)* $g_{ij} = k_{ij} - \sum_{r=m_m}^{i-1} l_{ri} g_{rj}$, i = mj+1,...,j−1 (3b)* 2) calculation of l_{ij} and d_{jj} $l_{ij} = \frac{g_{ij}}{d_{ii}}$, i = m_j, . . . , j − 1 (4a) $d_{jj} = k_{jj} - \sum_{r=m_j}^{j-1} l_{rj} g_{rj}$ (4b)

*) m_j = row number of the first nonzero element in column j
$m_m = \max\{m_i, m_j\}$
m_i = skyline of i−th d.o.f

이 j-자유도에서 l_{ij}를 구하기 위해서는 중간변수 g_{ij}가 필요

Table 3 Calculation of the Elements l_{14}, l_{24}, l_{34}

Step	Calculation
1	• calculation of l_{14} $m_1 = 1, m_4 = 1 \rightarrow m_m = \max(m_1, m_4) = 1$ $g_{14} = k_{14}$, $l_{14} = \frac{g_{14}}{d_{11}}$
2	• calculation of l_{24} $m_2 = 1$, $m_4 = 1 \rightarrow m_m = \max(m_2, m_4) = 1$ $g_{14} = k_{14}$, $g_{24} = k_{24} - l_{12} g_{14}$ $l_{24} = \frac{g_{24}}{d_{22}}$
3	• calculation of l_{34} $m_3 = 3$, $m_4 = 1 \rightarrow m_m = \max(m_3, m_4) = 3$ $g_{34} = k_{34}$ $l_{34} = \frac{g_{34}}{d_{33}}$
4	• calculation of d_{44} $d_{44} = k_{44} - \sum_{r=1}^{3} l_{r4} g_{r4} = k_{44} - (l_{14} g_{14} + l_{24} g_{24} + l_{34} g_{34})$

한데, 이 때 이전 단계에서 계산된 l_{ri}가 필요하다는 점이다. 이러한 사실의 의미를 명확히 파악해보기 위해서는, Fig. 1과 같이 저장된 강성행렬을 $\mathbf{LDL}^{\mathrm{T}}$ 법을 이용하여 분해하기 위한 과정을 분석해볼 필요가 있다.

먼저 Fig. 1과 같이 저장된 강성행렬에 대한 스카이라인(Skyline)을 구해보면, m_1=1, m_2=1, m_3=3, m_4=1, m_5=5, m_6=4, ⋯, m_n=n-4+1이 된다.

여기서 자유도 2 (2열)에 대한 l_{12}를 구하기 위해서는 식 (4a)에서와 같이 g_{12}가 필요한데, 이 값은 식 (3a) 또는 식 (3b)를 이용하여 구할 수 있다. 이 경우 m_1=1, m_2=1이므로 m_m = max (m_1, m_2)= 1가 된다. 따라서 g_{12}는 식 (3a)로부터 바로 구할 수 있으며, 식 (3b)는 계산할 필요가 없다. 이와 같이 g_{12}를 구한 후, 식 (4a)와 식 (4b)를 이용하여 l_{12}를 구하면 된다. 자유도 3 (3열)의 경우, l_{13}과 l_{23}에 해당하는 강성행렬의 항들이 스카이라인 밖에 존재하므로 이들을 계산할 필요가 없다. 따라서 식 (4b)로부터 $d_{33} = k_{33}$ 을 바로 구할 수 있다.

자유도 4 (4열)의 경우에는 m_4=1이므로 l_{14}~l_{34}를 구하여야 식 (4b)로부터 d_{44}를 구할 수 있다. $l_{14}$$l_{34}$를 구하기 위한 과정을 정리해보면 Table 3과 같다.

Table 3으로부터 알 수 있는 바와 같이 g_{24}를 계산하는 과정에서 l_{12}가 필요한데, 이 값은 자유도 2 (2열)를 소거하는 과정 중에 계산되는 값이다. 이와 같이 g_{24}를 계산할 때 l_{12}가 필요하게 된 이유는 자유도 2와 자유도 4는 서로 연관성(Coupled Simultaneous Equation)을 가지고 있기 때문이다.

컴퓨터 프로그램에서는 일반적으로 Fig. 1과 같은 행렬을

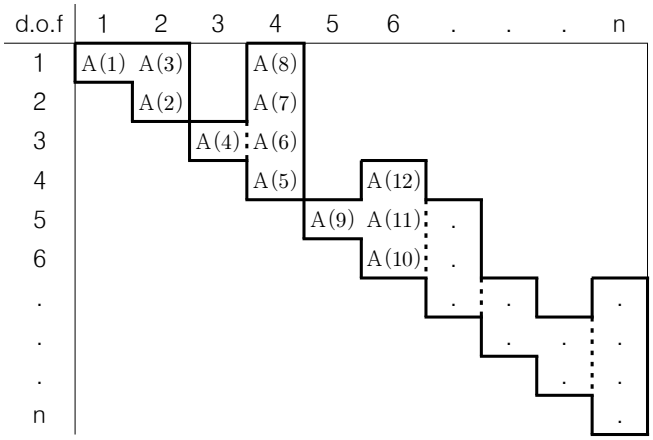


Fig. 3 Array Storing elements of K

일차원배열 (One Dimensional Array)에 저장한 후 소거를 한다. 이와 같이 저장할 경우, Fig. 1과 같은 강성행렬은 Fig. 3과 같이 일차원배열로 저장될 것이다.

구조해석 프로그램이 실행되는 시점에서, Table 1의 절차에 따라 사용가능한 메모리를 계산하여 보았더니 일차원배열을 4개 밖에 사용할 수 없는 상황을 가정해 보자. 이 경우에는 (k_{11}, k_{22}, k_{12}, k_{33}), (k_{44}, k_{34}, k_{24}, k_{14}), (k_{55}, k_{66}, k_{56}, k_{46}) 등과 같이 블록 (Block)으로 나누어 이들 값을 파일 (File) 등에 써놓은 다음, 필요한 시점에서 각 블록의 값을 파일로부터 읽은 후 이들 값을 A(1)~A(4)에 저장한 다음 열소거를 수행하여야 한다. 이와 같이 블록단위로 열소거를 진행할 경우, 두 번째 블록은 첫 번째 블록과 연관이 있으므로 첫 번째 블록에서 산정된 l_{ij}가 필요하며, 이에 대한 정보를 사전에 알고 있어야 한다. 즉, 이는 특정 블록에 대한 열소거를 시작하기 전에 이 블록이 어느 블록과 연관되어 있는지 사전에 알고 있어야 한다는 사실을 말해주고 있다.

전술된 사항들을 종합적으로 분석하여 정리하면, 행렬을 블록으로 나눈 다음 여기에 LDL^T법을 적용하여 행렬을 분해하기 위해서는 다음과 같은 사항들을 핵심적으로 처리해야 함을 알 수 있다. 먼저 사용가능한 메모리를 계산한 후, 이를 근거로 행렬을 블록으로 나눈다. 둘째, 특정 블록과 연관성을 가지고 있는 첫 번째 블록을 알아낸다. 마지막으로 각각의 블록에 대해 순차적으로 분해를 진행한다.

2.2 블록화 및 행렬의 분해

행렬분해 시 요구되는 항들은 대각항과 상부삼각행렬에 위치한 유효 값들이다. 컴퓨터 프로그램에서는 이 값들을 메모리에 저장한 후 원하는 계산을 수행하는 것이 일반적이다.

Table 4 Algorithm for the Calculation of the Number of Blocks

Step	Calculation
1	• execution of Step 2 ~ 3 from d.o.f 2 to d.o.f n_{dof}
2	• calculation of number of elements stored $n^1_{stor} = n_{arr} - ip_{dia} + n^0_{stor} + 1$
3	• if $n^1_{stor} > 0$, go to Step 1 • if $n^1_{stor} < 0$, calculate the number of columns and rows in present block

*) n_{dof} = number of d.o.f
n^1_{stor} = number of elements stored in random block
n_{arr} = number of useable array
ip_{dia} = array number of diagonal element in i-th d.o.f
n^0_{stor} = number of elements stored up to former block

만일 사용 가능한 메모리가 부족하여 필요한 데이터를 한 번에 메모리에 저장하지 못하는 경우가 발생한다면, 이를 해결할 수 있는 방안을 생각해보아야 한다.

열소거법 (Active Column Solver)을 이용하여 선형시스템의 해를 구하는 문제에서는, 특정 블록에 몇 개의 열을 배당하여야 사용가능한 배열을 이용할 수 있는가의 문제일 것이다. 이 문제는 행렬을 몇 개의 블록으로 분할하여야 하는가의 문제와 같다. 행렬을 일차원배열로 저장할 때 대각항이 저장되는 위치에 대한 정보가 있다면, 이 문제는 쉽게 해결될 수 있는데, 그 내용은 Table 4와 같다.

Table 4의 내용을 Fig. 1, Fig. 3을 중심으로 생각해 보면 다음과 같다. 먼저 사용가능한 배열의 개수를 $n_{arr} = 4$ 라 가정하면, 자유도 3에서는 $n^1_{stor} = 0$가 되므로, 자유도 1~자유도 3까지를 하나의 블록으로 지정하여 저장할 수 있다는 의미가 된다. 자유도 4에서는 $n^1_{stor} = -4$ 가 되므로, 이는 자유도 1~자유도 3까지를 하나의 블록으로 지정해 놓고 자유도 4부터 새로운 블록으로 지정하여야 함을 의미하게 된다. 전술한 바와 같이 계산을 수행하면 사용가능한 배열의 수를 고려하여 행렬을 블록으로 분할할 수 있다.

행렬을 블록으로 분할한 이후에는 특정 블록과 연결된 (Coupling) 첫 번째 블록을 찾아야 하는데, 이를 찾기 위한 방법을 논의하기에 앞서 Table 2에서 제시된 식 (3a)~식 (3b)를 분석해 보면 다음과 같다. 먼저 식 (3a)의 의미는, 자유도 j에서 처음으로 0이 아닌 값을 가져다가 $g_{m_j,j}$에 대입하면 되므로 이 값을 계산하는데 이전 자유도는 상관이 없다는 것을 뜻한다.

식 (3b)를 분석하여 보면, g_{ij}를 계산할 때 자유도 j와 연결된 첫 번째 자유도는 $j - m_j - 2$ 와 관련이 있음을 의미한다. 이로부터 자유도 j와 연결된 첫 번째 자유도는 식 (5)와 같이 계산될 수 있다.

Table 5 Algorithm for the Calculation of the Coupling Blocks

Step	Calculation
1	• execution of Step 2 ~ 3 from block 2 to block n_b
2	• calculation of maximum i_c in specific block $i_{c,\max} = \max(n_{i+1} - n_i - i - 1)$, i=1, 2, ···, n_c
3	• if $i_{c,\max} \leq 0$, go to Step 1 • if $i_{c,\max} > 0$, evaluate the first coupled block up to $i_{c,\max} \leq 0$ $i_c = n_{ib} - 1$; $i_{c,\max} = i_{c,\max} - n_{ic}$; $i_c = n_{ib} - 1$

*) n_b = number of blocks
n_i = array number of diagonal element in column I
n_{ic} = number of columns in present block
n_{ib} = present block number

$$i_c = j - m_j - 2 > 0 \tag{5}$$

여기서, j = j-자유도
m_j = j-자유도의 스카이라인 (Skyline)

전술한 바와 같은 개념을 사용하면 블록으로 분할된 행렬에서 블록 사이의 연결 관계를 계산할 수 있는데, 이를 수행하기 위한 알고리즘을 작성하면 Table 5와 같다.

앞에서 전술한 바와 같은 방법들을 이용하여 강성행렬을 필요한 만큼 블록으로 나누고, 각각의 블록과 연결된 첫 번째 블록에 대한 정보를 알아냈다면, 이러한 정보를 이용하여 강성행렬을 분해하기 위해서는 파일 (File)을 이용하여야 한다. 즉, 강성행렬을 블록으로 분할해야할 필요성은 사용가능한 메모리가 작아졌을 때 발생하는 것이므로, 강성행렬을 분해하는 과정에서 파일이 사용되어야 한다.

파일로부터 블록 단위로 강성행렬을 읽은 후, 이를 LDL^T 법을 이용하여 분해하기 위한 과정은 일반적인 행렬분해법과 같다.

3. 해석 예

2장에서 전술한 행렬의 블록화기법 및 행렬분해법의 정합성을 평가해보기 위하여, 본 연구에서는 컴퓨터 프로그램 (sNs)을 작성하고 이를 다양한 예제에 적용하여 구조해석을 수행하였다.

3.1 해석 정확도 평가

구조물 설계 시 가장 일반적으로 사용되는 트러스 (Truss), 골조 (Frame) 구조물에 대하여 제시된 블록화기법 알고리즘이 적용된 선형정적해석을 수행하고 이를 수계산 (Manual)

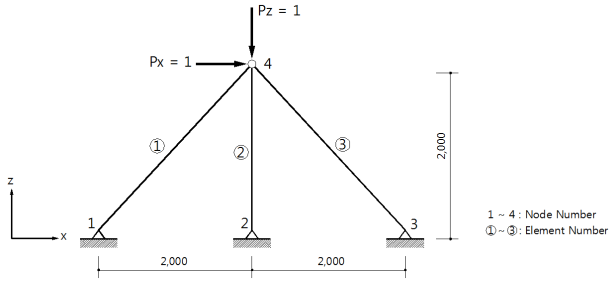


Fig. 4 Truss Structure

Table 6 Material and Section Properties

Modulus of Elasticity	Area	Support Condition
100.0	1.0	node 1, 2, 3 Hinged

Table 7 Analysis Results

	Displacement		Array Size	Number of Blocks
	δ_{X4}	δ_{Y4}		
Manual	28.284	−11.761	−	−
sNs	28.284	−11.761	3	1

과 비교하였다. 이 때, 가용 가능한 메모리의 크기를 각 해석 예제마다 임의로 설정하여 다양한 블록수 상에서 해석이 수행될 수 있도록 하였다.

3.1.1 트러스 구조물

사용가능한 배열의 수가 매우 작은 경우를 가정하여 Fig. 4와 같은 트러스구조에 대한 해석을 수행하였다. 이 경우에는 사용가능한 배열의 수를 3으로 가정하였으며, 블록의 수는 1이 되도록 조정하였다.

해석 시 가정된 재료 및 단면특성은 Table 6과 같으며, 수계산 및 개발프로그램을 이용한 해석결과는 Table 7과 같다.

3.1.2 2차원골조 구조물

Fig. 5에서 보는 바와 같이, 다이아프램 (Diaphragm) 구속조건을 갖는 2차원골조 (2D Frame)구조에 대한 해석을 수행하였다. 이 경우에는 사용가능한 배열의 수를 6으로 가정하였으며, 블록의 수는 3이 되도록 조정하였다.

해석 시 가정된 재료 및 단면특성은 Table 8과 같으며, 수계산 및 개발프로그램을 이용한 해석결과는 Table 9와 같다.

3.1.3 3차원 프레임 구조물

본 연구에서 제시된 블록화기법의 안정성 평가를 위하여,

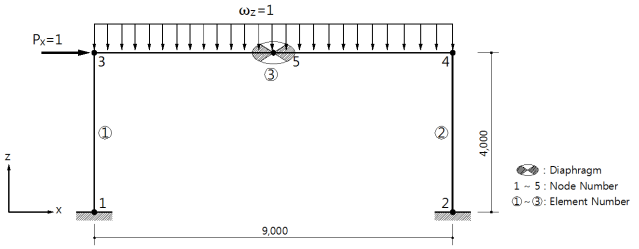


Fig. 5 2D Frame Structure

Table 8 Material and Section

Modulus of Elasticity	Area	Moment of Inertia	Supprot Condition
96,000.0	1,000.0	12,000.0	node 1, 2 Fixed

Table 9 Analysis Results

	Displacement				Array Size	Number of Blocks
	δ_{X3}	θ_{Y3}	δ_{X4}	θ_{Y4}		
Manual	4.209	4.795	4.209	−4.793	−	−
sNs	4.209	4.795	4.209	−4.793	6	3

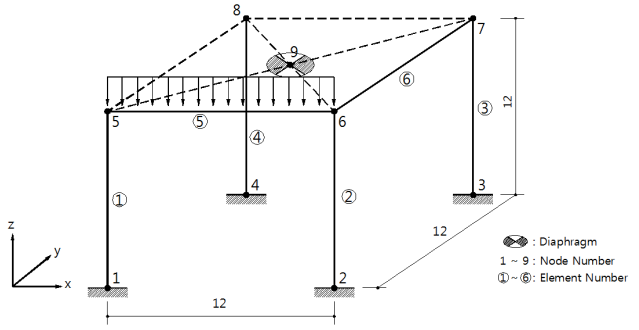


Fig. 6 3D Frame Structure

Table 10 Material and Section

Modulus of Elasticity	Area	Moment of Inertia	Support Condition
960.0	1.0	1.2	node 1~4 Fixed

Fig. 6과 같이 일정한 강성행렬 크기를 갖는 3차원 프레임구조를 대상으로 사용가능한 배열의 수를 변화시키면서 해석을 수행하였다.

해석 시 가정된 재료 및 단면특성은 Table 10과 같으며, 수계산 및 개발프로그램을 이용한 해석결과는 Table 11과 같다.

Table 11 Analysis Results

	Displacement			Array Size	Number of Blocks
	δ_{X6}	δ_{Y6}	δ_{Z6}		
Manual	0.06323	−0.02285	−0.00251	−	−
sNs	0.06323	−0.02285	−0.00251	93	1
	0.06323	−0.02285	−0.00251	35	3
	0.06323	−0.02285	−0.00251	19	6
	0.06323	−0.02285	−0.00251	17	7

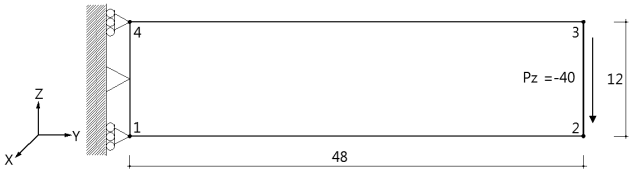
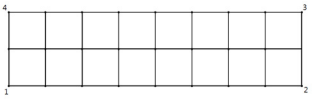
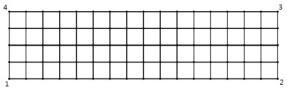


Fig. 7 Cantilever Beam

Table 12 Patterns of Analysis Modelling

Pattern 1 (Mesh 8×2)	
Pattern 2 (Mesh 16×4)	

3.2 해석 속도 평가

본 연구에서 제시한 블록화기법을 사용할 경우, 강성행렬을 분해하기 위해서는 강성행렬이 분할되는 블록 수에 따라 강성행렬을 파일로 저장하고 읽어야 하는 부분이 추가되어야 한다. 일반적으로 컴퓨터 프로그램의 실행시간 (Run Time)을 지배하는 주요인자 중 하나는 파일을 이용한 입출력 시간이라고 볼 수 있다. 이와 같은 이유로 인하여 대용량 메모리가 요구되는 강성행렬에 대한 블록화기법 적용 시 해석속도의 차이가 발생할 수 있으므로 이에 대한 효율성을 검토해 볼 필요가 있다.

전술된 바와 같은 필요성에 따라, 본 연구에서는 Fig. 7과 같은 캔틸레버 (Cantilever)보를 대상으로 요소분할을 통한 구조해석을 수행하였다. 요소분할은 Table 12에서 보는 바와 같이 분할되는 요소의 크기를 두 가지 형태로 설정하여 강성행렬 크기의 변화에 따른 해석속도 차이를 비교・분석해 보고자 하였다.

해석속도는 프로그램의 실행 후 하중에 의한 절점변위의 산정까지 소요되는 시간을 기준으로 측정하였으며, 사용가능

Table 13 Material and Section

Modulus of Elasticity	Poisson's Ratio	Thickness
30,000	0.25	2.0

Table 14 Analysis Speed Results

	Analysis Speed (sec)	Array Size	Number of Blocks	Ratio
Pattern 1	0.559	5,686	1	1.0
	0.405	4,258	2	0.7245
	0.234	1,758	4	0.4186
	0.203	1,005	6	0.3631
Pattern 2	6.549	17,422	3	1.0
	2.253	7,419	6	0.3440
	1.196	2,401	18	0.1826
	0.956	1,127	39	0.1460

한 배열수에 따라 각각 10회씩 해석을 수행한 결과에서 최대값과 최소값을 뺀 평균값을 적용하였다.

해석 시 가정된 재료 및 단면특성은 Table 13과 같으며, 개발프로그램을 이용한 해석속도 결과는 Table 14와 같다.

4. 결 론

효율적인 행렬분해를 위한 전산학적 방법론의 연구는 다양한 연구자들에 의해 수행되어 왔으나, 개인용 컴퓨터를 이용하여 행렬분해를 수행할 경우 발생하는 메모리 공간 부족 문제는 여전히 해결해야 할 문제점으로 인식되고 있다. 본 연구에서는 이를 효율적으로 해결 할 수 있는 방법론 중 하나인 블록화기법의 이론적 배경을 분석하였으며, 이를 실제 프로그램으로 구현할 수 있는 프로세스 (Process) 및 프로그래밍 과정에서 발생하는 데이터 처리 문제에 대한 해결 방안을 제시하였다.

또한, 다양한 예제에 대한 해석을 통하여 제시된 방법론 및 이를 바탕으로 작성된 프로그램 (sNs)의 정합성을 검증하였으며, 이에 대한 내용은 다음과 같다.

(1) 임의의 블록수로 나누어져 수행되어진 선형정적해석의 결과들은 모두 수계산에 의한 정해와 일치하고 있다. 이를 통하여 제시된 블록화기법은 전체적인 행렬분해 알고리즘의 정확도에 영향을 미치지 않으면서 구조물에 대한 평형방정식의 해를 효율적으로 구하고 있음을 알 수 있다.

(2) 3차원 프레임 구조물의 해석 시, 여러 경우의 블록수로 수행되어진 해석 결과가 동일한 값을 보이고 있으며, 수계산을 통한 정해와도 일치하고 있음을 알 수 있다. 이는 개인 컴퓨터를 통한 업무 수행 시 빈번히 발생할 수 있는 가용 가능 메모리 크기의 변화에도 불구하고 제시된 블록화기법 알고리즘이 안정적으로 해를 구할 수 있음을 보여주고 있다. 또한 작은 양의 메모리를 가지는 개인 컴퓨터에서도 신뢰할 수 있는 결과를 얻을 수 있음을 알 수 있다.

(3) Fig. 7의 캔틸레버 보 예제의 경우, 유형별 블록수에 따른 해석 속도는 블록수가 늘어날수록 빨라지는 경향을 보이고 있다. 다시 말해, 행렬분해 연산의 반복수행 및 파일 입출력 횟수의 증가에도 불구하고 실제적인 해석 시간은 줄어드는 경향을 보이고 있다. 이것은 유한한 메모리 크기에서 한 번에 많은 양의 데이터를 연산하고 파일로 입출력할 때 발생하는 컴퓨터의 부하증가가 블록화 알고리즘에 의한 입출력과정의 증가보다 해석속도에 크게 영향을 미칠 수 있음을 보여주는 결과로 판단된다. 따라서 적절한 블록수로 나누어 행렬분해를 수행하는 것이 전체 구조물에 대한 해석 속도를 향상시킬 수 있는 하나의 방법임을 알 수 있다.

(4) 다만, 텍스트 입출력 방식을 취하고 있는 개발 프로그램의 한계로 인하여 보다 큰 규모의 구조물에 대한 해석이 수행되지 못하였다. 이로 인하여 블록 수에 따른 해석속도 비교・분석과정에 미비한 점이 존재할 것으로 분석되며, 이에 대한 보완이 필요할 것으로 판단된다.

(5) 본 연구에서 개발된 블록화기법 알고리즘은 대표적인 행렬분해 연산기법인 스카이라인 솔버 (Skyline Solver)에 최적화된 알고리즘으로서, 다른 상용프로그램에서 제시되고 있는 멀티-프론탈 솔버 (Multi-Frontal Solver)에 대한 적용을 위해서는 추가적인 연구를 통한 알고리즘의 수정・보완이 필요할 것으로 판단된다.

본 연구를 통하여 제시된 블록화기법은 개인 컴퓨터의 사용가능한 메모리 크기가 충분하지 않은 경우에도 정확하고 안정적인 구조해석을 수행할 수 있는 것으로 분석된다. 향후, 자유도 수 만개의 구조물 해석을 위한 그래픽 인터페이스 (Graphic Interface) 전・후처리기 개발과 이를 바탕으로 한 블록화 알고리즘 최적화에 대한 지속적인 연구를 통하여 전체적인 해석속도 향상을 도모할 수 있을 것으로 판단된다.

감사의 글

본 연구는 정부 (교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임. [NRF-2010-0024 517]

References

1. ADINA ENGINEERING (1983), ADINA - System verification manual, REPORT AE 83-5.
2. Almasi, G. S. and Gottlieb, A. (1989), Highly Parallel Computing, Benjamin-Cummings publishers.
3. Bathe, K. J. (1996), Finite Element Procedures, Prentice Hall.
4. Computer and Structures, Inc. (2003), ETABS-Software Verification Exmaples, Ver. 8.
5. Computers and Structures, Inc. (1998), SAP2000-ANALYSIS REFERENCE, Ver. 7.0.
6. Farhat, C., Wilson, E. (1988), A parallel active column equation solver, Computers & Structures, 28(2), 289-304.
7. Han, J. I., Lee, K. D. (1998), Dynamic Analysis of Vehicle-Bridge System by the Dynamic Condensation Method, *The Journal of the Korea Institute for Structural Maintenance and Inspection*, 2(2), 177-184.
8. Mario Paz and William Leigh (2004), Structural Dynamics, Kluwer Academic Publishers.
9. Patterson, D. A., John, L. H. (1998), Computer Organization and Design, Second Edition, Morgan Kaufmann Publishers.
10. Przemieniecki, J. S. (1963), Matrix Structrual Analysis of Substructures, *AIAA Journal*, 1(1), 138-147.
11. Zienkiewicz, O. C., Taylor, R. L. and Zhu, J. Z. (2005), The Finite Element Method - Its Basis & Fundamentals, Elsevier Butterworth-Heinemann.

Received : 06/20/2013
Revised : 09/13/2013
Accepted : 10/04/2013

요 지

본 연구에서는 개인 컴퓨터의 플래시 메모리가 충분하지 않을 경우 대용량의 플래시 메모리를 필요로 하는 구조해석을 컴퓨터 프로그램으로 수행하는 방법론을 연구하였다. 이러한 문제점의 해결방안으로 강성행렬의 블록화기법 -강성행렬이 몇 개의 블록으로 나뉘고 각각의 블록에 대하여 행렬분해가 수행되는 방법- 을 제안하였으며 제안된 방법론을 바탕으로 컴퓨터 프로그래밍이 가능한 알고리즘을 제시하였다. 끝으로, 본 연구를 바탕으로 구조해석 프로그램을 개발하였으며 몇 가지 기초적인 구조해석 예제를 통하여 개발 알고리즘의 정합성 및 효율성을 확인할 수 있었다.

핵심 용어 : 구조해석, 대용량 행렬분해, 블록화기법, 강성행렬, 알고리즘, 구조해석 프로그램