Journal of The Korea Society of Computer and Information
Vol. 17 No. 9, September 2012
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci/2012.17.9.009

유한요소법에서 희소행렬의 효율적인 저장을 위한 2차원 가변길이 벡터 저장구조

부 희 형*, 김 승 호**

Two dimensional variable-length vector storage format for efficient storage of sparse matrix in the finite element method

Hee-Hyung Boo*, Sung-Ho Kim**

요 약

본 논문에서는 유한요소법에서 희소행렬의 효율적인 저장을 위한 2차원 가변길이 벡터 저장구조를 제안한다. 제안한 저장구조는 유한요소 전체 방정식의 거대희소행렬 $N \times N$ 대신, 전체 행의 개수 N의 상삼각행렬에서 0이 아닌 실제 필요한 값들만 2차원 가변길이 벡터를 이용하여 저장하는 방법이다. 이 방법을 이용하면, 해석대상의 2차원 격자구조에서는 각 절점당 최소 1개에서 최대 5개까지의 저장 공간이 필요하게 되고, 3차원 격자구조에서는 각 절점당 최소 1개에서 최대 14개까지의 저장 공간이 필요하게 된다. 인덱스를 포함해도 2배 이상을 넘지 않는다. 본 논문의 실험 결과에 의해, 제안한 저장구조는 총 절점 개수가 많아질수록 기존의 최대칼럼 높이를 저장하는 스카이라인 저장구조보다 메모리 공간을 효과적으로 줄일 수 있는 구조임을 알 수 있었다.

▸ Keywords : 2차원 가변길이 벡터 저장구조, 희소행렬 저장구조, 유한요소법, 야코비반복법

Abstract

In this paper, we propose the two dimensional variable-length vector storage format which can

---

• 제1저자 : 부희형 • 교신저자 : 김승호
• 투고일 : 2012. 02. 23, 심사일 : 2012. 07. 26, 게재확정일 : 2012. 09. 07.
* 경북대학교 전자전기컴퓨터학부(School of Electrical Engineering and Computer Science, Kyungpook National University)
** 경북대학교 컴퓨터학부(School of Computer Science and Engineering, Kyungpook National University) 교수
※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (과제번호 2012-0001829).
※ 이 논문은 2012학년도 경북대학교 학술연구비에 의하여 연구되었음.

www.kci.go.kr

be used for efficient storage of sparse matrix in the FEM (finite element method). The proposed storage format is the method storing only actual needed non-zero values of each row on upper triangular matrix with the total rows N, by using two dimensional variable-length vector instead of $N \times N$ large sparse matrix of entire equation of finite elements. This method only needs storage spaces of the number of minimum 1 to maximum 5 in $2D$ grid structure and the number of minimum 1 to maximum 14 in $3D$ grid structure of analysis target. The number doesn't excess two times although involving index number. From the experimental result, we can find out that the proposed storage format can reduce the memory space more effectively, as the total number of nodes increases, than the existing skyline storage format storing maximum column height.

▸ Keywords : Two Dimensional Variable-length Vector Storage Format, Sparse Matrix Storage Format, Finite Element Method, Jacobi Iterative Method

I. 서 론

유한요소법 (FEM: finite element method)은 유체역학, 구조물, 전자기학 등의 공학의 다양한 분야에서 이용되고 있는 수치해석법이다 [1,2]. 유한요소법은 전체 해석대상에 대해 대상물을 유한개의 요소들로 분할하고 각각의 유한요소들에 대하여 유한요소식을 만든 후, 유한요소들을 조립하여 전체 해석대상에 대한 연립방정식을 구성한다. 분할요소가 많아질수록 연립방정식은 커지게 되고 해를 구하는 과정에서 많은 계산이 필요하게 된다.

유한요소법의 전체 유한요소식은 $Ax=b$의 형태가 되고 A 행렬은 온도 값을 구하는 경우 총 절점 개수가 N이면, $N \times N$으로 구성되는 행렬이며 거대희소대칭 양정칙행렬 (large sparse symmetric positive definite matrix)의 특성을 갖는다. 기존 연구에 의하면 유한요소법의 거대희소대칭 양정칙행렬을 저장하기 위한 방법으로서 대칭성을 고려한 대상행렬법 (band matrix method)과 스카이라인법 (skyline method) 등이 잘 알려져 있다. 대상행렬법은 전체 행렬에 대하여 대각성분을 기준으로 전체 행의 최대 밴드폭만큼 각 행을 저장하는 방법이고 스카이라인법은 대각성분을 기준으로 0이 아닌 최고 높이에 해당되는 값까지 저장하는 방법이다 [3,4]. 이 방법은 각 행에서 최대 칼럼까지 저장하는 방법으로도 이용된다. 스카이라인법은 각 행마다 밴드폭을 따로 지정해주기 때문에 저장 공간을 많이 줄여주지만 유한요소들이 많아질수록 밴드폭이 넓어지게 되고 그에 따라 중간에 0이 많아지는 단점이 발생하게 된다. 그 외에 희소행렬을 저장하기 위한 CSR (compressed sparse row) 저장구조가 있지만 CSR 저장구조는 행렬의 정보를 미리 알고 있는 상태에서만 이용할 수 있다 [5].

본 논문에서는 유한요소법의 최종 연산인 $Ax=b$ 형태의 선형연립방정식에서 A 행렬의 저장 공간을 줄이기 위한 방법으로서 2차원 가변길이 벡터 저장구조를 제안한다. 또한 제안한 저장구조를 적용한 야코비반복법의 병렬처리 코드도 함께 제시한다. 실험 결과에 의해, 제안한 저장구조는 스카이라인 저장구조에 비하여 해석 구조의 크기가 커질수록 메모리 이용면에서 더욱 효율적이었고 병렬처리 기법에도 간단하게 적용할 수 있었다.

다음 제2장에서는 본 논문에서 이용된 유한요소방정식의 원 식인 열전도방정식을 소개하고 스카이라인 저장구조와 CSR 저장구조에 대하여 간단히 설명한다. 제3장에서는 본 논문에서 제안한 2차원 가변길이 벡터 저장구조에 대하여 상세히 설명하고, 제4장에서는 실험 및 결과로서 제안한 저장구조와 스카이라인 저장구조를 비교한다. 또한 제안한 저장구조를 적용한 야코비반복법의 병렬처리 코드도 함께 제시한다. 마지막으로 제5장에서는 실험 결과에 대한 결론을 맺는다.

II. 열전도방정식의 유한요소방정식

본 논문에서는 유한요소법의 희소행렬을 저장하기 위한 방법으로서 2차원 가변길이 벡터 저장구조를 제안한다. 본 논문에서 이용된 편미분방정식은 3차원 정상상태의 열전도방정식이고 지배방정식은 아래 식 (1)과 같다 [1,6].

$$\chi\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right)+Q=0 \tag{1}$$

www.kci.go.kr

위 식 (1)에서 T 는 온도를 나타내는 공간에 대한 함수이고 χ 는 열전도 계수이며 Q 는 단위체적당 공급되는 열량으로서 내부 발열률을 나타낸다. 식 (1)은 Galerkin 방법을 이용하여 공간에 대한 이산화 과정을 수행하면, 아래 식 (2)와 같은 형태의 전체 유한요소방정식을 얻을 수 있다.

$$[K]\{\Phi\} = \{F\} \tag{2}$$

위 식 (2)에서 $[K]$는 열전도 매트릭스 (heat conductivity matrix)이고 $\{\Phi\}$는 전체 절점온도 벡터이며 $\{F\}$는 열유속 벡터 (flux vector)이다.

본 논문에서는 전체 유한요소방정식에 해당되는 식 (2)에서 행렬 $[K]$에 이용될 수 있는 메모리 저장구조로서 2차원 가변길이 벡터 저장구조를 제안한다. 제안한 저장구조는 스카이라인 저장구조에 비하여 해석 구조의 크기가 커질수록 더욱 효율적인 저장구조임을 제4장의 표 2를 통해 알 수 있다.

다음은 스카이라인 저장구조와 CSR 저장구조에 대하여 프로그램에서 이용되는 저장 형식을 간단히 예를 들어 설명한다.

1) 스카이라인 저장구조

유한요소법에서 이용되는 스카이라인 저장구조의 예는 아래와 같다.

$$K = \begin{bmatrix} 11 & 0 & 13 & 0 & 0 & 16 \\ & 22 & 0 & 24 & 0 & 0 \\ & & 33 & 34 & 0 & 0 \\ & & & 44 & 0 & 46 \\ & & & & 55 & 56 \\ & & & & & 66 \end{bmatrix} \tag{3}$$

① 1차원 배열을 이용하여 K 대칭행렬의 상위 원소들을 칼럼 순으로 저장한다.

② $(N+1)$ 정수 배열 p에 대각 원소의 포인트들을 저장한다 (대각 원소의 개수: N). 즉, p의 $(i+1)$번째 원소는 i 번째 대각 원소의 포인트가 된다.

③ 심볼 S를 s와 p로 구성한다.

$s = \{11, 22, 13, 0, 33, 24, 34, 44, 55, 16, 0, 0, 46, 56, 66\}$,

$p = \{0, 1, 2, 5, 8, 9, 15\}$,

$S = \{p, s\}$.

2) CSR 저장구조

희소 행렬을 저장하는 방법에는 CSR 저장구조가 있고, 이 저장구조는 제안한 저장구조와 유사하다. CSR 저장구조의 예는 아래와 같다.

$$K = \begin{bmatrix} 11 & & & 14 & & & 17 \\ & & 23 & & & & \\ & & 33 & 34 & & 36 & \\ & & 43 & 44 & & 46 & \\ & & & 54 & & 56 & \\ & 62 & & & & & 67 \\ & 72 & & & 75 & & \end{bmatrix} \tag{4}$$

① 1차원 배열 A에 K 행렬의 원소들을 행 우선으로 저장한다.

② 1차원 배열 J에 K 행렬의 칼럼 인덱스를 저장한다.

③ 1차원 배열 LEN에 0이 아닌 각 행의 길이를 저장한다.

④ 1차원 배열 PTR에 배열 J 또는 배열 A에서 각 행의 첫 번째 요소를 가리키는 포인터를 저장한다.

표 1. CSR 저장 구조의 예
Table 1. Example of the CSR storage format

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	11	14	17	23	33	34	36	43	44	46	54	56	62	67	72	75
J	1	4	7	3	3	4	6	3	4	6	4	6	2	7	2	5
LEN	3	1	3	3	2	2	2									
PTR	1	4	5	8	11	13	15									

다음 제3장에서는 본 논문에서 제안한 2차원 가변길이 벡터 저장구조에 대하여 상세히 설명한다.

III. 격자구조에서의 2차원 가변길이 벡터 저장 구조

본 논문에서 제안한 2차원 가변길이 벡터 저장구조는 유한요소법에서 희소행렬을 저장하기 위한 방법으로서, 분할 요소가 많아질수록 메모리 공간을 효과적으로 줄일 수 있는 방법이다.

아래 그림 1은 2차원과 3차원 격자구조에서 절점번호에 대한 저장개수를 나타낸다. 유한요소법은 보통 요소들이 인접해 있는 경우 이웃하는 절점들의 값들을 모두 저장하여 전체 행렬을 구성하는 반면에, 제안한 저장구조는 자신의 번호부터 윗번호에 해당되는 이웃하는 절점들의 값들만으로 상삼각행렬을 구성한다. 그림 1의 격자구조의 예를 들면, 왼쪽의 2차원 격자구조의 경우 18번, 34번, 35번 등의 내부에 있는 절점들은 자신의 번호 (대각원소)부터 윗번호에 해당되는 이웃하는 절점들의 값들을 저장하기 위해 최대 5개의 저장 공간이 필요하게 되고, 오른쪽 3차원 격자구조의 경우는 18번, 274번 등의 절점에서 최대 14개의 저장 공간이 필요하게 된다. 즉 제

www.kci.go.kr

안한 저장구조는 거대희소대칭 양정치행렬 $N \times N$ 을 저장하기 위해 2차원 격자구조에서는 최대 $N \times 5$만 저장하면 되고 3차원 격자구조에서는 최대 $N \times 14$만 저장하면 된다.

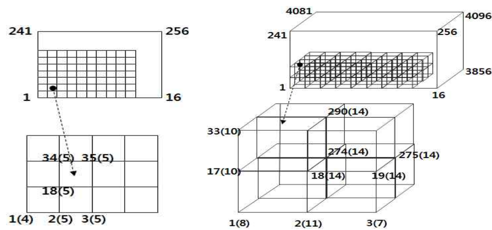

그림 1. 2차원(16×16)과 3차원($16 \times 16 \times 16$) 격자구조에서 절점번호당 필요한 저장개수 [형식: 절점번호(저장개수)]
Fig. 1. The number of storage spaces needed per node number at $2D(16 \times 16)$ and $3D(16 \times 16 \times 16)$ grid structure [Format: node number(The number of storage spaces)]

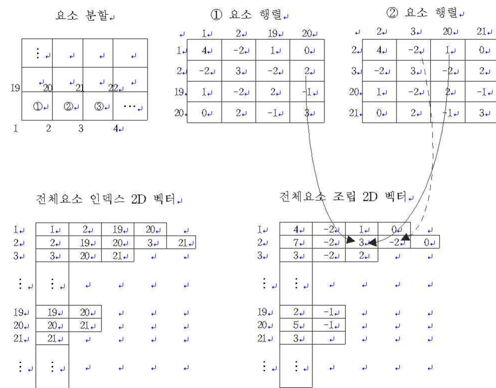

그림 2. 제안한 2차원 가변길이 벡터 저장구조의 예
Fig. 2. Example of the proposed two dimensional variable-length vector storage format

위 그림 2는 조립과정에서 ①번 요소행렬과 ②번 요소행렬이 저장될 때, 2차원 가변길이 벡터 저장구조의 저장 방식을 나타낸다. 처음 ①번 요소행렬이 들어오면 1, 2, 19, 20번의 각 절점에 대해 전체요소 인덱스 $2D$ 벡터의 해당 절점에서 같은 원소의 인덱스를 찾는다. 저장된 인덱스가 있는 경우 전체요소 조립 $2D$ 벡터의 해당 원소에 값을 누적하고, 저장된 인덱스가 없는 경우 인덱스와 값을 해당되는 전체요소 인덱스 2D 벡터와 전체요소 조립 $2D$ 벡터의 각 절점에 추가한다. 같은 방식으로 ②번 요소행렬과 ③번 요소행렬을 차례대로 저장한다.

본 논문에서 제안한 2차원 가변길이 벡터 저장구조는 메모리 공간을 효과적으로 줄일 수 있는 방법이다. 제안한 저장구조에 의해 얻을 수 있는 이점은 메모리 공간을 줄임으로써 실행 속도에 도움을 줄 수 있고, 기존의 저장구조로는 실행이 불가능 했던 것을 가능하게 만들 수도 있다. 예를 들어, 16,384×16,384 이상의 행렬의 경우 스카이라인 저장구조나 대상행렬 저장구조는 제안한 저장구조보다 최소 10배 이상의 메모리 공간을 요구하기 때문에, 제한된 메모리를 사용하는 경우 실행할 수 있는 경우와 없는 경우로 나누어질 수 있다. 또한 CSR 저장 구조와는 달리 행렬의 정보를 미리 알고 있지 않아도 필요한 원소만 각 행에 추가해주는 방식이기 때문에 값을 효율적으로 저장할 수 있으며 자동처리 계산에도 도움을 준다.

제안한 저장구조와 스카이라인 저장구조의 비교는 다음 4장에서 나타내었다.

IV. 실험 및 결과

본 논문에서는 유한요소법에서 희소행렬을 효율적으로 저장하기 위한 2차원 가변길이 벡터 저장구조를 제안한다. 본 논문의 실험 목표는 제안한 저장구조가 스카이라인 저장구조보다 메모리 이용 면에서 더 효율적인지 알아보기 위한 것으로, 표 2와 그림 3를 통해 두 가지 방법을 비교해 본다.

아래 표 2는 제안한 저장구조와 스카이라인 저장구조에 대하여 상삼각행렬만을 저장했을 때, 해석 구조의 크기에 따른 필요한 저장 공간의 개수를 나타낸다.

표 2. 2차원 격자구조에서, 구조 크기에 따른 세 가지 경우의 $[K]$ 행렬 저장 공간의 비교
Table 2. Comparison storage space of $[K]$ matrix of three cases according to structural sizes at $2D$ grid structure

정방행렬 저장구조	제안한 저장구조	스카이라인 저장구조
전체구조크기: 6×6 총절점수: 36 $[K]$ 행렬 저장공간: $36 \times 36 = 1,296$	146 인덱스 포함: 292	246
16×16 $256 \times 256 = 65,536$	402 인덱스 포함: 804	4,337
32×32 $1,024 \times 1,024$ $= 1,048,576$	4,930 인덱스 포함: 9,860	33,760
64×64 $4,096 \times 4,096$ $= 16,777,216$	20,098 인덱스 포함: 40,196	266,176
128×128 $16,384 \times 16,384$ $= 268,435,456$	81,154 인덱스 포함: 162,308	2,113,408

www.kci.go.kr

위 표 2의 결과에 의해, 6×6 구조에서는 스카이라인 저장구조가 제안한 저장구조보다 적은 메모리 공간을 이용하였지만, 16×16 구조 이상에서는 제안한 저장구조가 스카이라인 저장구조보다 더 적은 메모리 공간을 이용함을 알 수 있었다. 특히 아래 그림 3의 그래프에서 볼 수 있듯이 총 절점개수가 16,384인 구조에서는 제안한 저장구조가 스카이라인 저장구조에 비하여 약 1/13의 저장 공간만을 이용함으로써 그 비율의 차이가 더욱 크게 나타남을 알 수 있었다. 즉 총 절점 개수가 많아질수록 제안한 저장구조가 메모리 공간을 더욱 효과적으로 줄일 수 있는 구조임을 확인할 수 있었다.

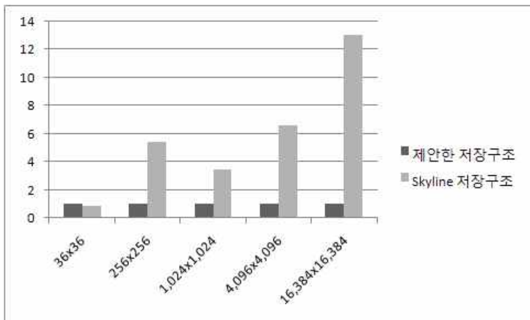

그림 3. 제안한 저장구조와 스카이라인 저장구조의 메모리 이용에 대한 비율
Fig. 3. Ratio of memory usage between the proposed storage format and the skyline storage format

아래 표 3은 선형연립방정식에서 여섯 가지 해법에 대한 순차 처리의 계산 복잡도를 나타낸다.

표 3. 순차처리에서 여섯 가지 해법에 대한 계산복잡도
Table 3. Computational complexity of six solvers at sequential processing

해법	계산복잡도 (순차처리)
가우스 소거법 (직접법)	$n^3/3+n^2/2$
촐레스키 분해법 (직접법)	$n^3/3+2n^2$
특이값 분해법 (직접법)	$2n^3+4n^3$
야코비반복법 (반복법)	n^2 (반복문 내에서)
가우스-자이델법 (반복법)	n^2 (반복문 내에서)
공액 구배법 (반복법)	$2n^3+13n^2$ (반복문 내에서)

직접법은 행마다 계산이 연관되어 있기 때문에 전체적인 병렬처리가 불가능하여 속도 향상에 제한이 있다. 그러나 야코비반복법과 같은 반복법에서는 전체적인 병렬처리 연산이 가능하여 수행 속도를 더욱 향상시킬 수 있다. MATLAB에서의 선형연립방정식의 해는 $x=A\setminus b$ 에 의해 구해질 수 있고, A 행렬이 대칭이면서 양정칙행렬이면 내부적으로 촐레스키 (Cholesky) 분해법이 이용된다. 이러한 직접 해법에서는 기존의 스카이라인 저장구조나 대상행렬 저장구조가 효율적으로 이용될 수 있고, 본 논문에서 제안한 2차원 가변길이 벡터 저장구조는 전체적인 병렬처리 연산이 가능한 반복법에 효율적으로 이용될 수 있다.

아래 표 4는 제안한 저장구조를 적용한 야코비반복법과 MATLAB의 $x=A\setminus b$ 에 대해, 행렬 크기에 따른 수행시간을 나타낸다. 결과에서 야코비반복법의 허용오차는 $1e-8$을 만족한다.

표 4. 행렬 크기에 따른 수행시간의 비교
Table 4. Comparison for computation time according to matrix sizes

행렬 크기	야코비반복법 (GPU)	MATLAB ($x=A\setminus b$)
$82,654\times 82,654$	0.010688 sec$onds$	0.315166 sec$onds$
$1,228,045\times$ $1,228,045$	0.067325 sec$onds$	8.190626 sec$onds$

위 표 4의 결과에 의해, GPU를 이용한 야코비반복법이 MATLAB의 해법보다 더 빠른 수행시간을 나타냄을 알 수 있었다. 그러나 실제 상황에서의 수행시간은 행렬의 크기, 제로가 아닌 값의 개수, 그리고 값의 분포형태 등에 따라 차이를 보일 수도 있다.

아래 그림 4는 제안한 저장구조를 GPU 병렬처리에 적용하기 위한 준비과정을 나타내고, 그림 5는 야코비반복법의 GPU 병렬처리 코드를 나타낸다. 병렬처리에 이용된 GPU는 Tesla 1060 모델이고 프로그램 언어는 CUDA를 이용하였다 [7,8,9]. 제안한 2차원 가변길이 벡터 저장구조는 병렬처리를 수행하기 위해 대각 원소 아래의 값들을 각 행의 끝에 더 추가해주었고 1차원으로 변환하여 수행하였다.

본 논문에서 제시한 병렬처리 야코비반복법에서는 총 절점 개수 N이 총 스레드 개수의 범위 내에 있다면, 정방행렬의 경우 반복문 내에서 N만큼 반복을 해줘야하는 반면에 제안한 저장구조를 이용하는 경우 kk 변수에 의해 2차원 격자구조에서는 최대 8회만 반복하면 되고, 3차원 격자구조에서는 최대 26회만 반복하면 된다. 조립 단계에서의 2차원 가변길이 벡터는 CUDA의 thrust::device_vector를 이용하였고 push _back() 함수에 의해 각 행에서 값을 추가해 주었다. 만약 2차원 가변길이 벡터가 클 경우 C++ STL의 vector를 전역변수로 선언하여 이용할 수 있다.

www.kci.go.kr

```
...

int* ptrIdx = thrust::raw_pointer_cast(&dL_idx1D[0]);
float* ptrNumEleNode =
            thrust::raw_pointer_cast(&dL_numEleNode1D[0]);
float* ptrX = thrust::raw_pointer_cast(&dL_X[0]);
float* ptrT1 = thrust::raw_pointer_cast(&dL_T1[0]);
int* ptrAccmSize =
            thrust::raw_pointer_cast(&dL_accmSize[0]);
float* ptrR = thrust::raw_pointer_cast(&dL_R[0]);

float* squareDiff =
            thrust::raw_pointer_cast(&dL_squareDiff[0]);
float* squareX = thrust::raw_pointer_cast(&dL_squareX[0]);

int memNumEleNode = dL_accmSize[N-1]*sizeof(float);
int memT1 = N*sizeof(float);
int memR = N*sizeof(float);
int nodeN = N;

cudaBindTexture(0,texIdx,ptrIdx);
cudaBindTexture(0,texNumEleNode, ptrNumEleNode,
                memNumEleNode);
cudaBindTexture(0,texT1, ptrT1, memT1);
cudaBindTexture(0,texAccmSize, ptrAccmSize);
cudaBindTexture(0,texR,ptrR, memR);

do{
    solver <<< n_blocks, block_size >>> (nodeN, ptrX, ptrT1,
                          ptrSquareDiff, ptrSquareX);

    norm1 =
        std::sqrt(  thrust::reduce(dL_squareDiff.begin(),
                     dL_squareDiff.end()) );
    norm2 = std::sqrt(  thrust::reduce(dL_squareX.begin(),
                     dL_squareX.end()) );
    err = norm1/norm2;
    iter++;

} while ((iter < iterMax) &&  (err > tol));

cudaUnbindTexture(texIdx);
cudaUnbindTexture(texNumEleNode);
cudaUnbindTexture(texT1);
cudaUnbindTexture(texAccmSize);
cudaUnbindTexture(texR);
...
```

그림 4. 제안한 저장구조를 적용한 야코비반복법의 GPU 병렬처리를 위한 준비과정과 반복문
Fig. 4. Preparation and iteration for GPU parallel processing of the jacobi iterative method which the proposed storage format is applied to

```
texture<int, 1, cudaReadModeElementType> texIdx;
texture<float, 1, cudaReadModeElementType>
                                     texNumEleNode;
texture<float, 1, cudaReadModeElementType> texT1;
texture<int, 1, cudaReadModeElementType> texAccmSize;
texture<float, 1, cudaReadModeElementType> texR;


__global__  void  solver(int  nodeN,  float*  x,  float*  T1,  float*
squareDiff, float* squareX)
{
  int d_idxNode=0, d_curr=0, d_diag=0;
  float sum=0.0;
  int tx = blockIdx.x * blockDim.x + threadIdx.x;
  int thread_n = blockDim.x * gridDim.x;

  while (tx < nodeN)
  {
    for(int kk = 1; kk < (tx == 0 ?
   tex1Dfetch(texAccmSize, tx):(tex1Dfetch(texAccmSize, tx)
                 - tex1Dfetch(texAccmSize, tx-1)) ); ++kk)
    {
        __syncthreads();
        d_curr = (tx == 0 ? kk :
              (tex1Dfetch(texAccmSize, tx-1) + kk) );
               d_idxNode = tex1Dfetch(texIdx, d_curr);
        sum = sum +  ( tex1Dfetch(texNumEleNode, d_curr)
                  * tex1Dfetch(texT1, d_idxNode-1) );
      }
      __syncthreads();
      d_diag = (tx == 0 ? 0 : tex1Dfetch(texAccmSize,
                 tx-1) );
      x[tx] = ( tex1Dfetch(texR, tx) - sum ) /
            tex1Dfetch( texNumEleNode , d_diag);

      squareDiff[tx] =  (x[tx]  - tex1Dfetch(texT1, tx))
               * (x[tx]  - tex1Dfetch(texT1, tx)); // for norm
      squareX[tx] = x[tx] * x[tx];

      __syncthreads();
      T1[tx] = x[tx];
      tx = tx + thread_n;
   }
}
```

그림 5. 제안한 저장구조를 적용한 야코비반복법의 GPU 병렬처리 코드
Fig. 5. GPU parallel code of the jacobi iterative method which the proposed storage format is applied to

위 그림 5에서 이용된 야코비반복법의 공식은 아래 식 (5)와 같다.

$$x^{(k+1)} = D^{-1}\left(b - Rx^{(k)}\right) \quad (5)$$

www.kci.go.kr

위 식 (5)에서 $x^{(k+1)}$은 $(k+1)$번째 반복 단계에서의 근사해의 벡터를 나타내고 D는 주어진 행렬에서 대각 원소들로 구성된 행렬이며, R은 주어진 행렬에서 대각원소들을 제외한 나머지 원소들로 구성된 행렬이다. b는 선형연립방정식에서 우변 항에 해당되는 벡터이다.

V. 결 론

본 논문에서는 유한요소법에서 희소행렬을 효율적으로 저장하기 위한 2차원 가변길이 벡터 저장구조를 제안하였고 제안한 저장구조를 적용한 야코비반복법의 GPU 병렬처리 코드를 함께 제시하였다. 제4장의 실험 결과로부터, 6×6 구조에서는 스카이라인 저장구조가 제안한 저장구조보다 적은 메모리 공간을 이용하였지만 16×16 구조 이상에서는 제안한 저장구조가 스카이라인 저장구조보다 더 적은 메모리 공간을 이용하였다. 특히 128×128 구조에서는 약 $1/13$의 저장 공간만을 이용함으로써 해석 구조의 크기가 커질수록 스카이라인 저장구조와의 차이는 더욱 크게 나타남을 알 수 있었다.

제안한 저장구조를 적용한 병렬처리 야코비반복법에서는 반복문 내에서 2차원 격자구조의 경우 최대 8회만 반복하면 되었고, 3차원 격자구조의 경우 최대 26회만 반복하면 되었다.

결과적으로, 총 절점개수가 많아질수록 제안한 저장구조가 메모리 공간을 효과적으로 줄일 수 있는 구조임을 알 수 있었고 야코비반복법의 병렬처리 기법에도 간단하게 적용됨을 알 수 있었다.

향후 연구에서는 분할된 요소가 많은 해석대상에서 처리속도를 더욱 높일 수 있는 효율적인 알고리즘들의 연구가 더 많이 이루어져야 할 것이다.

참고문헌

[1] G. R. LIU, S. S. QUEK, "THE FINITE ELEMENT METHOD: A practical course," Butterworth-Heinemann, 2003.

[2] R.E. Lewis, J.P. Ward, "The finite element method: principles and applications," Addison-Wesley, 1991.

[3] Takeo Taniguchi, Kohji Fujiwara, "Parallel Skyline Method using Two Dimensional Array," Memoirs of the Faculty of Engineering, Okayama University, Vol.24, No.2, pp.99-112, March 1990.

[4] Felippa, C.A., "Solution of Linear Equations with Skyline-stored Symmetric Matrix," Computers and Structures, Vol. 5, pp. 13-29, 1975.

[5] Eun-Jin Im, "An Efficient Computation of Matrix Triple Products," Journal of the Korea Society of Computer and Information, Vol.11, No.3, pp. 141-149, July 2006.

[6] Thomas J. Rudolphi, "Finite Element Method," McGraw-Hill Korea, 2010.

[7] NVIDIA CUDA (Compute Unified Device Architecture): Programming Guide, Version 2.1, December 2008.

[8] Junghwan Kim, Jinsoo Kim, "Implementation of Efficient Power Method on CUDA GPU ," Journal of the Korea Society of Computer and Information, Vol.16, No.2, pp. 9-16, Feb. 2011.

[9] Zhihui Zhang, Qinghai Miao, Ying Wang, "CUDA-Based Jacobi's Iterative Method," Computer Science-Technology and Applications, IFCSTA '09. International Forum, IEEE Computer Society, Vol. 1, pp. 259-262, 2009.

www.kci.go.kr

저 자 소 개

부 희 형

2004: 목포대학교 컴퓨터공학과
공학사
2006: 전남대학교 컴퓨터정보통신공학과
공학석사
2008년~현재: 경북대학교 전자전기
컴퓨터학부 박사과정
관심분야: 동영상 압축, 그래픽스,
유한요소법 등
Email : hhboo@knu.ac.kr

김 승 호

1981: 경북대학교 전자공학과 공학사
1983: 한국과학기술원 전산학과 공학석사
1993: 한국과학기술원
전산학과 공학박사
1985년 3월~현 재: 경북대학교
컴퓨터학부 교수
관심분야: 알고리즘, 멀티미디어,
다시점 동영상, 감시 시스템,
동기식 이더넷 등
Email : shkim@knu.ac.kr

www.kci.go.kr