Journal of The Korea Society of Computer and Information
Vol. 20 No. 12, pp. 67-74, December 2015

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.12.067

Implementation and Experiments of Sparse Matrix Data Structure for Heat Conduction Equations

Jae-Gu Kim*, Ju-Hee Lee**, Geun-Duk Park***

Abstract

The heat conduction equation, a type of a Poisson equation which can be applied in various areas of engineering is calculating its value with the iteration method in general. The equation which had difference discretization of the heat conduction equation is the simultaneous equation, and each line has the characteristic of expressing in sparse matrix of the equivalent number of none-zero elements with neighboring grids.

In this paper, we propose a data structure for sparse matrix that can calculate the value faster with less memory use calculate the heat conduction equation. To verify whether the proposed data structure efficiently calculates the value compared to the other sparse matrix representations, we apply the representative iteration method, CG (Conjugate Gradient), and presents experiment results of time consumed to get values, calculation time of each step and relevant time consumption ratio, and memory usage amount. The results of this experiment could be used to estimate main elements of calculating the value of the general heat conduction equation, such as time consumed, the memory usage amount.

▸ Keyword : Sparse Matrix, Data Structure, Heat Conduction Equation, Conjugate Gradient Methods

I. Introduction

열전달현상은 대표적인 포아송방정식(Poisson's equation)으로 중첩에 의한 해석해(analytic solution)를 구할 수 있다. 그러나 대상으로 하는 계의 형상이 복잡해짐에 따라 해석적인 방법보다는 수치적인 방법에 의존해 해를 구하게 된다. 수치적 방법으로 해를 구하기 위해서는 전체 계산 영역을 여러 개의 작은 제어체적(control volume) 혹은 격자(cell)로 나누고 작은 제어체적에 대하여 지배방정식(governing equation)인 포아송 방정식을 적용한다. 격자를 구성하는 형식에 따라 정렬격자(structured grid)와 비정렬격자(unstructured grid)계로 나눌 수 있다. 정렬격자는 격자선(grid line)을 따라 인덱스(일반적으로 i, j, k)를 이용하여 격자의 형상정보나 해에 쉽게 접근이 가능하다. 가장 대표적인 카트시안 격자(Cartesian grid)의 경우 각 격자선은 좌표축을 따라 형성되어 있어 일차미분(대류항)과 이차미분(확산항)의 차분화가 매우 직관적이다. 복잡한 형상에 대하여 물리적공간과 계산공간을 서로 맵핑하는 방법 혹은 다중블록(multi-block)방법이 있다.[1][2] 그러나 이러한 정렬격자는 해석하고자 하는 영역이나 형상이 복잡해지는 경우 격자형성이 어렵거나 불가능해지는 치명적인 단점을 가지고 있다. 이에 반해 비정렬격자는 복잡한 형상에 적용이 가능하며 필요한 부분에 쉽게 격자를 밀집시킬 수 있다는 장점을 가지고 있다.[3] 그러므로 대부분의 상용해석프로그램은 비정렬격자를 사용하고 있으나 이산화된 대수방정식은 희소 행렬식(sparse matrix)이기 때문에 메모리의 낭비와 희소 행렬을 다루기 위한 기술을 필요로 한다. 열전도 방정식의 또 다른 해석 방법으로 유한요소법(FEM)이 있다.[4] 유한요소법에 차분화한 방정식도

• First Author: Jae-Gu Kim, Corresponding Author: Geun-Duk Park
*Jae-Gu Kim(kim5257@naver.com), Dept. of Computer Engineering, Hoseo University
**Ju-Hee Lee(juheelee@hoseo.edu), Dept. of Computer Engineering, Hoseo University
***Geun-Duk Park(gdpark@hoseo.edu), Dept. of Computer Engineering, Hoseo University
• This research was supported by a grant(15RERP-B082204-02) from Residential Environment Research Program funded by Ministry of Land, Infrastructure and Transport of Korean government.
• Received: 2015. 10. 29, Revised: 2015. 11. 10, Accepted: 2015. 12. 02.

최종적으로 얻어진 방정식의 형태는 유한체적법과 같은 연립방정식이 얻어지며 이를 다루는 방법으로 대상행렬법(banded matrix method)과 스카이라인법(skyline method)이 있으며 [4]에서는 희소 행렬의 저장 공간을 줄이기 위한 방법으로 2차원 가변길이 벡터 저장구조를 제안하였다. 또한 제안한 저장구조를 적용한 야코비반복법(Jacobin iteration method)과 그래픽용 계산기(GPU)를 이용한 병렬처리 코드도 제시하였다.

본 논문은 유한체적법을 이용한 열전도 방정식에 적용할 수 있는 적은 메모리와 효율적인 연산을 수행 할 수 있는 희소행렬 자료구조를 제안한다. 또한 제안한 행렬 자료구조와 다른 자료구조를 사용하여 방정식 해를 구하는 연산을 수행하여 연산시간과 메모리 사용량 등의 실험 결과를 제시한다. 행렬 연산을 하는 다른 수치 해석에 어떤 행렬 자료구조가 효율적인지 본 실험 결과를 활용할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 열전도 방정식을 푸는 방법과 이 때 사용되는 행렬 자료구조에 대해 설명하고, 3장에서는 논문에서 제안하는 행렬 자료구조의 구성과 연산 방법에 대해 설명한다. 4장에서는 본 논문에서 제안하는 행렬 자료구조에 대해 실험 한 결과를 제시하고, 5장에서는 실험결론과 실험 결과의 활용 방안을 기술한다.

II. Related Works

1. Heat conduction equation

열이나 유체의 유동의 경우, 지배방정식을 유도하기 위하여 개개의 입자를 고려한 접근보다는 공간내의 임의의 영역에서의 검사체적을 이용하여 에너지(온도), 운동량, 질량의 연속된 흐름을 관찰하게 된다.

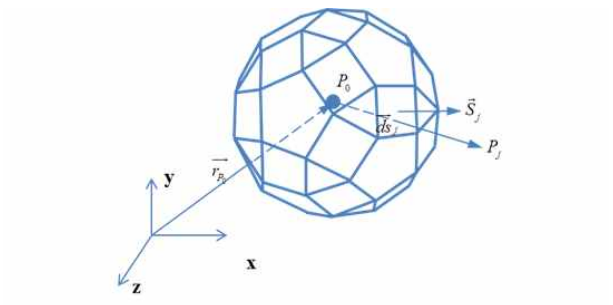


Fig. 1. Control volume of unstructured cell with surfaces.

이러한 관찰은 최종적으로 다음과 같은 일반화된 적분형의 Reynolds 수송 방정식(Reynolds transport equation)을 얻게 된다.[5] 수식 1은 왼쪽부터 첫 번째 항은 시간에 따라 변화하는 양인 비선형 항(unsteady), 두 번째 항의 첫 번째 항은 대류(convection), 두 번째 항은 확산 항(diffusion)이며 마지막 항은 생성 항에 해당하며, 오른쪽은 생성 항(source term)을 나타낸다. Γ는 종속변수(ϕ)에 따라 달라진다. 전열해석의 경우 Γ는 열전도 계수(k)가 된다. 전열해석의 경우 대류 항이 없는 수송방정식이 되며 확산 항과 생성 항이 차분화의 대상이 되며 비 정렬격자를 사용하기 때문에 확산 항과 격자의 표면에서의 종속변수의 값에 대한 적절히 모델링하거나 근사화가 필요하다.

$$\frac{\partial}{\partial t}\int_V \rho\phi dV + \int_S [\rho\phi \boldsymbol{v} - \Gamma\nabla\phi]\cdot d\boldsymbol{S} = \int_V Q dV \qquad (1)$$

2. Difference discretization

수식 1의 셀 중심-비 정렬격자를 이용하여 이산화한 대수방정식은 격자수와 같은 차원의 1차방정식의 행렬을 구성하게 된다. 전체 해석 시간 중 선형방정식의 해를 얻는데 가장 많은 시간이 소모된다. 그러므로 수렴성이 좋은 방법을 사용하는 것은 수치해석의 성능에 많은 영향을 미친다.

$$A\phi = b \qquad (2)$$

여기서 A는 $n\times n$의 행렬이 되며, ϕ와 b는 의 n개수를 갖는 벡터가 된다. n은 미지수의 개수로 격자수와 같아진다. 수식 1의 확산방정식을 수식 2의 비 정렬격자에 이산화한 경우 인접한 두 셀이 공유하는 면을 통해서 수수되는 열 유속은 서로 일치해야 함으로 A는 대칭행렬이 된다. 또 해 행렬의 고유 값이 모두 양수라면 정부호(positive definite)하다고 하며 이러한 행렬의 해를 구하는 데에는 CGS (Conjugate Gradient Solver)라는 매우 효과적인 방법이 존재한다.[6]

3. Conjugate gradient (CG)

Conjugate Gradient란 대칭인 양의 준정부호행렬을 갖는 선형계의 해를 구하는 수치 알고리즘이다. 이는 반복알고리즘으로 해를 구할 수 있으며, 촐스키 분해와 같은 방법이나 직접 풀기에 너무 큰 차수(order)를 갖는 희소행렬 등에 사용하기 적합하다.[7] Table 1은 CG를 사용하여 행렬의 해를 구하는 의사코드이다.

Table 3. Pseudocode of Conjugate Gradient

$r_0 = b - A_{x0}$
$p_0 = r_0$
$k = 0$
loop
  $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  if r_{k+1} *value is less than enough then end loop*
  $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} = r_{k+1} + \beta_k p_k$
  $k = k+1$
loop end
solution is x_{k+1}

4. Dictionary of keys (DOK)

DOK는 사전으로 구성된 행렬을 말한다. 사전은 행/열 쌍을 값으로 가지는 맵으로 구성된다. 만약 요소 값이 사전에 없으면 그 값은 0을 가진다. 이 형식은 임의 순서로 증가하는 희소행렬에는 좋으나, 행렬 값이 사전식 순서로 0이 아닌 값이 반복되는 경우에는 효과적이지 않다.[8]

5. Compressed Sparse Row (CSR)

CSR은 행렬을 row_ptr, col_ind, val 세 개의 1차원 배열로 표현하는 방식이다. Fig. 2와 같이 val 배열은 $n \times n$ 행렬 값을 (0,0)부터 (n,n)까지 0이 아닌 요소 값을 저장하고, col_ind는 val의 각 요소 값의 열 위치 값을 저장한다. 그리고 row_ptr은 col_ind와 val 배열에서 각 행의 시작 위치 값을 저장한다.[9]

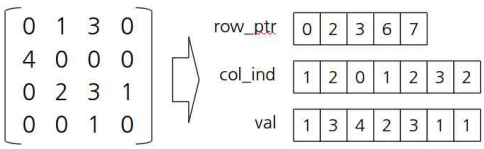


Fig. 2. The structure of CSR

III. Data structure of Vector-Array

1. Data structure

본 논문에서 제안하는 벡터배열(Vector-Array) 자료구조는 Fig. 3과 같이 구성된다. STL(Standard Template Library)에서 제공하는 가변길이 배열 자료구조인 Vector를 사용하여 행렬의 한 열의 데이터를 열 위치 값과 요소 값을 저장하고 행렬의 행 크기만큼의 Vector 배열을 가진다.

벡터배열 자료구조는 LIL과 비교하면, LIL은 Linked list를 구성하기위해 인접 노드를 연결하는 포인터가 필요한데 반해 벡터배열은 Vector 자료구조를 사용하므로 행렬의 각 요소 값을 저장하는데 필요로 하는 메모리가 더 적다. 또 요소 값을 추가, 삭제할 때 CSR은 모든 행렬의 요소 값이 한 개 배열에 저장되어 있어 최악의 경우 전체 배열의 값을 변경해야 하지만, 벡터배열은 한 개 행의 Vector의 값을 변경하면 된다.

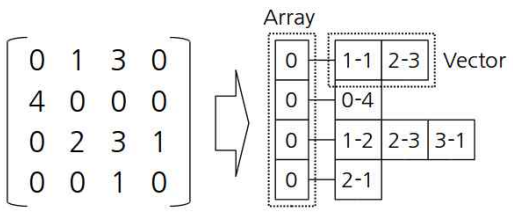


Fig. 3. The structure of Vector-Array

C++에서 Fig. 3과 같은 자료구조를 구성하기 위해 Table 2와 같이 클래스를 구성한다. node_t 클래스는 행렬의 한 개 요소를 저장하기 위해 열값과 행렬 요소 값이 선언되어 있으며, node_t 형식으로 사용하는 STL vector 배열을 선언하여 벡터 배열 구조를 정의한다.

Table 5. Definition of Vector-Array class

```
typedef double elem_t;

class node_t{
public:
    size_t mCol;
    elem_t mElem;

    node_t ( size_t col, elem_t data ){
        mCol = col;
        mElem = data;
    }

    inline bool operator== ( size_t col ) const
    {
        return (mCol == col);
    }
}
```

2. Operation

2.1 Addition

행렬 덧셈 및 뺄셈은 크게 두 가지 작업을 통해 수행한다. 예를 들어 Fig. 4와 같이 행렬 A와 행렬 B를 더하여 새로운 행렬 C를 얻고자 한다면, 우선 Fig. 5의 (1)과 같이 덧셈/뺄셈 하고자 하는 행렬 A를 새로 선언한 행렬 C에 복사를 한다. 그 다음 Fig. 5의 (2)와 같이 행렬 B의 행렬 요소 값을 순차 참조하면서 요소 값과 동일한 위치에 해당하는 행렬 C의 요소 값과 덧셈 또는 뺄셈한다. 만약 행렬 C에 요소 값의 위치에 해당하는 값이 없으면 새로운 요소 값을 행렬 C에 추가한다.

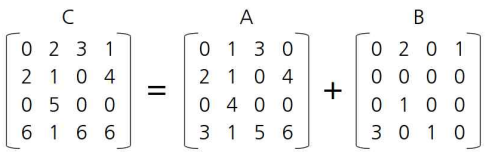
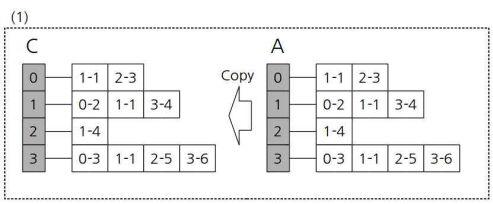


Fig. 4. Matrix addition



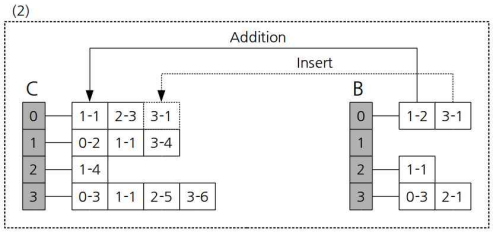


Fig. 5. Matrix addition process

2.2 Multiplication

행렬 곱셈은 크게 두 가지 작업을 통해 수행한다. 예를 들어 Fig. 6과 같이 행렬 곱셈을 한다고 가정하면, 먼저 Fig. 7의 (1)과 같이 행렬 A의 각 행렬 요소 값들을 순차 참조한다. 그리고 Fig. 7의 (2)와 같이 각각의 순차 참조한 요소 값과 행렬 B에서 요소 값의 열 위치에 해당하는 행 전체의 각각의 요소 값과 곱셈을 한 후 행렬 C에서 행렬 A에서 참조한 요소 값의 행 위치 값과 같은 행 위치와 행렬 B에서 참조한 요소 값의 열 위치 값과 같은 열 위치의 요소와 합산한다.

$$\begin{matrix} C \\ \begin{bmatrix} 0 & 3 & 0 & 3 \\ 12 & 4 & 4 & 2 \\ 0 & 0 & 0 & 0 \\ 18 & 11 & 6 & 8 \end{bmatrix} \end{matrix} = \begin{matrix} A \\ \begin{bmatrix} 0 & 1 & 3 & 0 \\ 2 & 1 & 0 & 4 \\ 0 & 4 & 0 & 0 \\ 3 & 1 & 5 & 6 \end{bmatrix} \end{matrix} \times \begin{matrix} B \\ \begin{bmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Fig. 6. Matrix multiplication

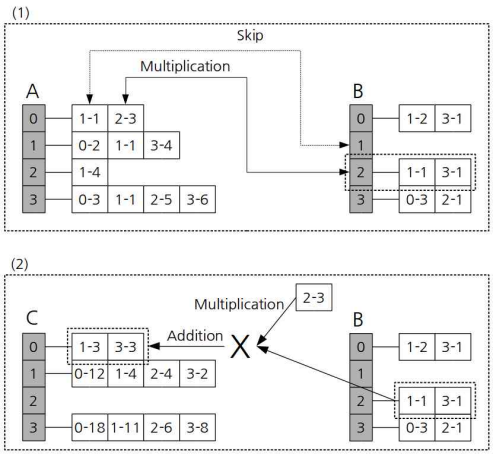


Fig. 7. Matrix multiplication process

2.3 Transposed Matrix multiplication

전치 행렬 곱셈은 Fig. 8와 같이 연산을 한다고 가정하면, Fig. 9의 (1)과 같이 행렬 A의 각 행렬 요소 값들을 순차 참조한다. 그리고 Fig. 9의 (2)와 같이 각각의 순차 참조한 요소 값과 행렬 B에서 요소 값의 행 위치에 해당하는 행 전체의 각각의 요소 값과 곱셈을 한 후 행렬 C에서 행렬 A에서 참조한 요소 값의 열 위치 값과 같은 행 위치와 행렬 B에서 참조한 요소 값의 행 위치 값과 같은 열 위치의 요소와 합산한다.

$$\begin{matrix} C \\ \begin{bmatrix} 9 & 0 & 3 & 0 \\ 3 & 6 & 1 & 5 \\ 15 & 6 & 5 & 3 \\ 18 & 0 & 6 & 0 \end{bmatrix} \end{matrix} = \begin{matrix} A^T \\ \begin{bmatrix} 0 & 1 & 3 & 0 \\ 2 & 1 & 0 & 4 \\ 0 & 4 & 0 & 0 \\ 3 & 1 & 5 & 6 \end{bmatrix} \end{matrix} \times \begin{matrix} B \\ \begin{bmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Fig. 8. Transposed matrix multiplication

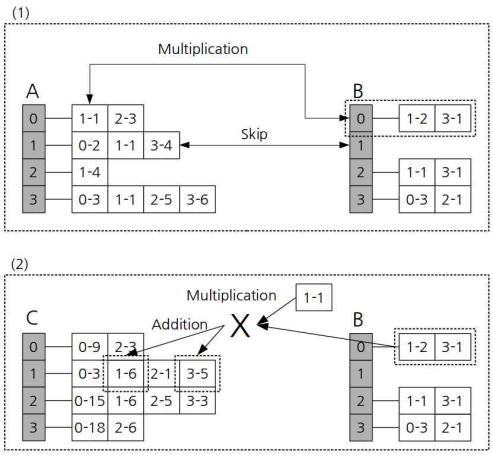


Fig. 9. Transposed matrix multiplication process

IV. Experiments

본 절에서는 구현한 벡터배열 자료구조와 Conjugate Gradient(CG) 알고리즘을 사용하여 $Ax = B$ 행렬 방정식의 해를 구하는 연산을 수행하여 연산에 걸린 시간 및 메모리 사용량을 측정하고 그 결과를 제시한다. 이 때 실험에 사용한 PC는 Table 3과 같이 구성하였다.

Table 6. Test Environment

CPU	Intel i7-4500U 1.8GHz
RAM	8GB
OS	ubuntu 14.04 LTS

1. Execution time for CG operations

본 연구에서 제안한 벡터배열 자료구조가 갖는 연산 및 각 단계에서 갖는 특성을 파악하기 위하여 다른 자료구조와 수치비교실험을 수행하였다. 또 서로 다른 크기의 행렬에 대해 연산을 수행하여 수행시간을 측정했다. 실험 조건은 행렬크기와 조건수(condition number), 그리고 자료구조 형식에 따라 나뉘며, 각 조건마다 10번 동일한 조건에서 계산을 수행하고 각 수행에 걸린 시간을 리눅스에서 제공하는 gettimeofday함수를 사용하여 측정하여 평균을 취했다. 이 때 조건수는 행렬 방정식 $Ax = B$에서 근사해에 의한 x가 얼마나 부정확할지 범위를 알려준다.

Table 7. Execution Time in CG

Matrix Size	100	100	1000	1000	5000	5000
Condition number	1.1942	318.5047	1.1975	5.2746E04	1.1978	2.311E04
Array	0.0004	0.0038	0.0159	5.5016	0.4154	734.217
CSR	0.0004	0.0095	0.0200	7.4739	0.4863	943.270
Vector-Array	**0.0017**	**0.0344**	**0.0102**	**3.7474**	**0.0615**	**112.592**
Map-Array	0.0351	0.2999	0.0721	26.170	0.3830	737.813

Table 8. Execution Time per Iteration in CG.

Matrix Size	100	100	1000	1000	5000	5000
Condition number	1.1942	318.5047	1.1975	5.2746E04	1.1978	2.311E04
CG Iteration	12	279	12	4878	13	26984
Array	3.69E-5	1.31E-5	0.00113	0.00093	0.00913	0.01172
CSR	3.82E-5	3.43E-5	0.00166	0.00153	0.03741	0.03495
Vector-Array	**0.00014**	**0.00012**	**0.00085**	**0.00076**	**0.00473**	**0.00417**
Map-Array	0.00292	0.00107	0.00601	0.00536	0.02946	0.02734

본 논문에서 제안하는 벡터배열을 포함하여 총 네 가지 자료구조에 대하여 비교하였다. Array는 이 차원 배열을 사용하여 행렬을 구현한 것이며, CSR은 일반적으로 알려진 희소행렬 표현 자료구조이다. 맵배열은 벡터배열에서 각 열의 자료를 Vector에 저장하는 대신 DOK 방식과 유사하게 Map을 사용하여 저장하도록 구현한 자료구조이다.

Table 4는 행렬 크기와 자료구조를 달리하여 CG 연산을 수행하여 수행 시간을 측정한 시간을 정리한 표이다. Table 4에서 행렬 크기는 시험에 사용한 표본의 행렬 크기를 의미하며, 조건수는 해당 행렬 데이터의 조건수를 의미한다. 자료구조는 CG연산에 사용된 행렬 자료구조를 의미한다. 그리고 소요시간은 CG연산에 소요된 시간을 의미한다.

CG 연산 시간을 측정한 결과 같은 크기의 행렬이라 하더라도, 조건수의 값에 따라 CG 연산 안에서 반복 회수가 달라지므로 각각의 CG 반복 회수와 한 번 반복할 때 소요되는 시간을 Table 5에 정리했다.

Table 5는 Table 4의 측정 결과와 CG 반복 회수를 나누어 CG 연산의 반복을 한 번 수행 할 때마다 소요되는 시간을 정리한 표이다.

2. Analysis of execution time

또한 CG 연산 중 CG 전체 연산 시간에 많은 비중을 차지하는 연산을 찾기 위해 CG 연산을 보다 작은 단위로 나누어 각 연산에 걸린 시간을 측정했다. 측정하고자 하는 연산들은 CG 전체 연산 중 반복과정에 포함 된 연산들이며, Table 6은 해당 연산들을 목록으로 정리한 것이다. Table 6의 각 항목에 대해 자료구조와 행렬 크기를 다르게 하여 연산 시간을 측정한 결과를 정리했다. 측정할 때 사용한 샘플 행렬은 100x100 (cond: 1.1942), 1000x1000 (cond: 1.1975), 5000x5000 (cond: 1.1978), 10000x10000 (cond: 1.1978) 이며, 각 항목 연산 시간은 Table 6의 연산 목록을 한 번 반복 할 때마다 각 항목에 대해 수행시간을 측정 한 결과들의 평균을 내었다. 이 때 조건수에 따라 계산의 형태가 각 스텝별로 다른 특성을 보일 것으로 예상하였으나 조건수는 단지 반복수를 증가시켜 행렬의 전체 수렴 구간을 증가시킬 뿐 한번 반복하는 시간에는 영향을 미치지 않는다. 따라서 조건수가 큰 행렬에 대해서도 같은 수치실험을 수행하였으나 본 논문에서 그 결과를 제시하지는 않았다.

Table 9. Operation Steps in CG

Order	Formula	Number of operations
1	$Ap = A \times p$	NxN
2	$ptval = p^{-1} \times Ap$	N
3	$\alpha = \frac{rsold}{ptval}$	1
4.1	$temp = p \times \alpha$	N
4.2	$x = x + temp$	N
5.1	$temp = Ap \times \alpha$	N
5.2	$r = r - temp$	N
6	$rsNew = r^{-1} \times r$	N
7	$sqrtval = \sqrt{rsNew}$	1
8	$result = x$	1
9.1	$temp = p \times \frac{rsNew}{rSold}$	1
9.2	$p = r + temp$	N
10	$rSold = rsNew$	1

Table 10. Memory usage in CG

Matrix Size	100	100	1000	1000
Condition number	1.1942	318.5047	1.1975	5.2746E04
Array	274KB	7.9MB	173MB	429MB
CSR	243KB	563KB	2.1MB	4MB
Vector-Array	**350KB**	**1.34MB**	**4.9MB**	**9.52MB**
Map-Array	515KB	2.1MB	9.27MB	18.5MB

Fig. 10~13은 CG 연산 리스트의 각 항목을 수행한 시간을 자료구조별 비율 그래프로 표현한 것이다. Fig. 10의 Array는 1번 항목이 가장 높은 비율을 차지하고 있으며, 행렬 크기가 증가함에 따라 1번 항목의 급격히 증가하고 있음을 알 수 있다. 행렬의 크기가 5000×5000이상에서는 거의 대부분의 연산시간을 1번 항목이 차지하고 있다. Fig. 11의 CSR은 1, 4.1, 5.1, 9.1 항목이 비슷한 비율을 가지며, 자료구조의 크기가 증가해도 각 항목의 비율은 비슷하게 유지한다.

Fig. 12 벡터배열은 처음엔 4.1, 5.1, 9.1에 비해 1번 항목의 비율이 낮지만, 배열 크기가 증가할수록 비율이 증가하여 5000x5000부터 비율이 일정하게 유지된다. Fig. 13 맵배열도 벡터배열과 마찬가지로 처음엔 4.1, 5.1, 9.1 항목에 비해 1 항목의 비율이 낮지만, 배열크기가 증가하면 1 항목이 비율이 높아지고 5000x5000부터 비율이 일정하게 유지된다.

Fig. 10에서 1번 항목의 비율이 점차 증가하는 이유는 1 항목의 연산 횟수가 N^2으로 다른 항목에 비해 연산 횟수 증가량이 많기 때문으로 보인다. 반면에 Fig. 11~13에서 1 항목의 비율이 일정하게 유지하는 이유는 CSR, 벡터배열, 맵배열 자료구조는 0인 항목에 대해서는 연산을 생략하기 때문에, 한 행에 0이 아닌 요소가 10개 정도인 행렬에 대해 연산 횟수가 N^2이 아닌 $10\cdot N$이 되기 때문이다.

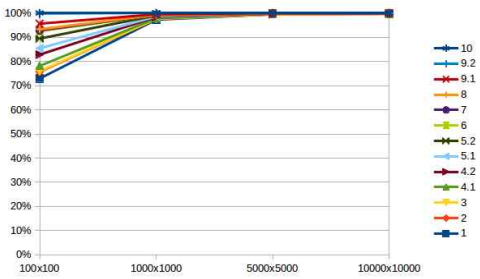


Fig. 10. Ratio Graph of Execution Time According to Operation Steps in CG using Array

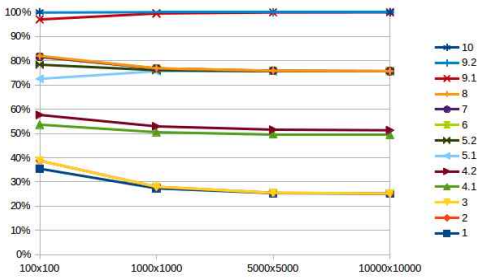


Fig. 11. Ratio Graph of Execution Time According to Operation Steps in CG using CSR

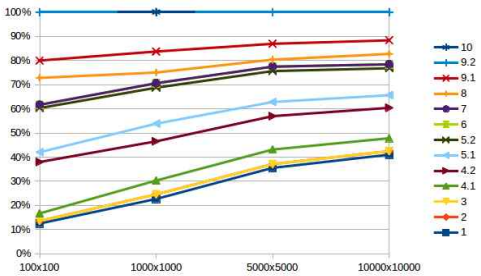


Fig. 12. Ratio Graph of Execution Time According to Operation Steps in CG using Vecotr-Array

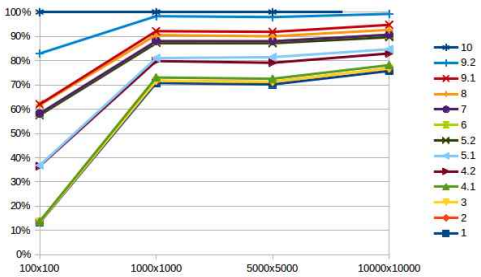


Fig. 13. Ratio Graph of Execution Time According to Operation Steps in CG using Map-Array

3. Memory usage for CG operations

CG 연산 할 때 실험에 사용된 행렬 자료구조들이 얼마나 메모리를 사용하는지 확인하기 위해 서로 다른 크기의 행렬을 CG 연산을 수행하여 프로그램이 사용 중인 메모리 크기를 측정했다.

실험 조건은 행렬크기와 자료구조에 따라 나뉘며, 각 조건마다 10번 동일하게 수행시간을 측정하여 측정 된 값의 평균을 취한다. 실험 방법은 우선 주어진 행렬 데이터의 CG 연산을 완료하고 사용자 입력을 받은 후 종료하도록 실험용 프로그램을 작성하여 실행한다. 그리고 CG 연산을 완료한 후 사용자 입력을 기다릴 때 프로세스 상태를 확인하여 메모리 사용량을 측정한다. Table 7은 행렬의 크기와 자료구조를 달리하여 CG 연산을 수행하여 실험 프로그램의 메모리 사용량을 측정한 값을 정리한 표이다. Table 7에서 행렬 크기는 시험에 사용한 표본의 행렬 크기 및 조건 변수를 의미하며, 자료구조는 CG연산에 사용된 행렬 자료구조를 의미한다. 그리고 메모리 사용량은 CG연산을 수행하는 프로그램의 메모리 사용량을 측정한 값이다.

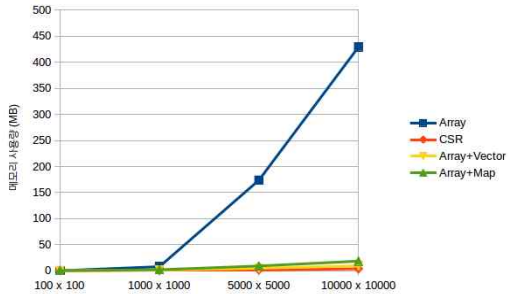


Fig. 14. The Graph of Memory Usage in CG

Fig. 14는 Table 7의 측정된 메모리 사용량을 그래프로 표현한 것이다. Array는 값이 0인 요소도 포함하므로 행렬 크기가 증가할수록 메모리 사용량이 급격히 늘어나는 반면 CSR, 벡터배열, 맵배열은 값이 0인 요소는 포함하지 않으므로 한 행 당 요소 값이 일정하게 고정되므로 행렬크기가 증가해도 메모리 사용량이 일정하게 증가한다.

또한 벡터배열은 CSR에 비해 메모리를 좀 더 많이 차지하는 이유는 CSR이 전체 행렬에 한 개 Vector 객체를 가지는 반면, 벡터배열은 한 행에 한 개씩 Vector 객체를 가지므로 더 많은 메모리를 사용하는 것으로 파악된다. 한편 맵배열은 Map 자료구조가 이진트리를 구성하기 위해 사용하는 포인터가 메모리를 사용하므로 세 자료구조 중 가장 많은 메모리를 사용한다.

V. Conclusions

본 논문에서는 반복법을 이용하여 열전도방정식의 해를 구하는 과정에서 적은 양의 메모리를 사용하면서 효율적으로 해를 구할 수 있는 희소행렬을 위한 자료구조를 제안하였다. 제안한 벡터배열 자료구조가 기존에 제안된 희소행렬 자료구조에 비해 열전도 방정식의 해를 효율적으로 구하는지 검증하기 위해 대표적인 반복 연산방법인 CG(conjugate gradient) 연산에 적용하여 실험하였으며 그 결과를 제시하였다.

실험 결과 제안한 방법은 CSR에 비해 행렬 크기에 상관없이 약 2.38배 많은 메모리를 사용하지만, 연산 시간은 행렬의 크기에 따라 격차의 편차가 있지만 평균적으로 약 8.37배 더 빨리 완료되었다. 특히, 실험을 통해 제안한 방법과 CSR의 연산 속도의 격차는 행렬의 크기에 비례하여 커짐을 확인하였다. 열전도방정식은 공간을 많은 격자로 나눌수록 얻는 해의 정확도가 높은 특성을 가지기 때문에 제안한 방법이 열전도방정식의 해를 구하는데 매우 효율적이라 할 수 있다.

또한 여러 자료구조에 대해 CG 연산의 단계별 연산 소요 시간을 실험하고 그 결과를 제시하였다. 본 논문에서 비교한 실험 결과를 활용하여 열전도 방정식의 해를 구하는데 있어, 주어진 조건이나 연산의 특성에 따라 가장 효율적인 자료 구조를 선택하는데 활용할 수 있다.

향후 CG연산 이외에도 현재 많이 활용되고 있는 BiCG (Biconjugate gradient), PCG (preconditioned conjugate gradient) 등 다른 반복 연산 방법에 대해서도 본 논문에서 제안한 자료구조의 효율성 검증과 그 결과에 따른 효율적인 자료구조 및 연산 방법을 설계하는 연구를 진행할 필요가 있다.

REFERENCES

[1] Thompson, Joe F.; Warsi, Zahir UA; Mastin, C. Wayne. Numerical grid generation: foundations and applications. Amsterdam: North-holland, 1985.

[2] Im, Eun-Jin. "An Efficient Computation of Matrix Triple Products." Journal of the Korea Society of Computer and Information 2006, Nov.: 141-149.

[3] Myong, H. K. Evaluation of Numerical Approximations of Convection Flux in Unstructured Cell-Centered Method. Journal of computational fluids engineering, 2006, Jan.: 36-42.

[4] Boo, Hee-Hyung; Kim, Sung-Ho. Two dimensional variable-length vector storage format for efficient storage of sparse matrix in the finite element method. Journal of the Korea Society of Computer and Information, 2012, Sep.: 9-16.

[5] White, Frank M. Fluid mechanics.(7thedn). 2011.

[6] Golub, Gene H.; Van Loan, Charles F. Matrix computations. JHU Press, 2012.

[7] Conjugate gradient method Wiki, https://en.wikipedia.org/wiki/Conjugate_gradient_method

[8] Sparse matrix Wiki, https://en.wikipedia.org/wiki/Sparse_matrix

[9] Liu, Chao, Junmin Ye, and Yining Ma. "Storage and Solving of Large Sparse Matrix Linear Equations." 2012 Fourth International Conference on Computational and Information Sciences. IEEE, 2012.

Authors

Jaegu Kim received the B.S. degree in Computer Engineering from Hoseo University, Korea, in 2011.

He is currently a M.S. student in the department of computer engineering at Hoseo university. He is interested in embedded computing, mobile computing, and network computing.

Juhee Lee received the B.S. degree in Mechanical Design from Kookmin University, M.S. and Ph.D. degrees in Mechanical Engineering from Hanyang University, Korea, in 1989, 1996 and 2010, respectively

Dr. Lee joined the faculty of the department of computer engineering at Hoseo university in 2007. He is currently a Professor in the department of computer engingeering, Hoseo University. He is interested in developing the code of computational fluid dynamics, heat and mass transfer, and optimization.

Geunduk Park received M.S. and Ph.D. degrees in Computer Engineering from Seoul National University in 1997 and 2005 respectively, and B.S. degree in Computer Science and Statistics from Seoul National University in 1993.

He is currently an associate professor of the department of computer engineering at Hoseo university. His current research interests include embedded software engineering, service oriented architecture, and semantic web technology.