한국항행학회 논문지 제13권 제1호 2009년 2월

# 대형 스파스 행렬로 표현되는 선형시스템 방적식의 해를 구하기 위한 지능적 병렬 반복법

# Intelligent Parallel Iterative Methods for Solving Linear Systems of Equations with Large Sparse Matrices

채수환*, 김명규**

Soo-Hoan Chae*, Myung-Kyu Kim**

## 요 약

VLSI 설계를 위한 회로 시뮬레이션, 영상처리, 구조 공학, 항공역학 등 공학 분야에서 대형 선형시스템 방정식의 해를 구하기 위해 고성능 컴퓨터에 대한 요구가 증가되고 있다. 이런 요구를 충족하기 위해 많은 다양한 병렬처리시스템이 제안되고 제작되고 있다. 선형시스템의 특성에 따라 그 해를 구하기 위한 적절한 알고리즘이 필요하다. 선형시스템 방정식의 해를 구하기 위해 여러 가지 직접법, 반복법이 사용되고 있다. 본 연구에서는 대형 스파스 행렬 형태를 가진 선형시스템 방정식의 해를 구하기 위해 지능적인 병렬반복법을 제안하고 효율성을 시뮬레이션에 의해 증명하였다.

## Abstract

The demand for high performance computer grows to solve large linear systems of equations in such engineering fields - circuit simulation for VLSI design, image processing, structural engineering, aerodynamics, etc. Many various parallel processing systems have been proposed and manufactured to satisfy the demand. The properties of linear system determine what algorithm is proper to solve the problem. Direct methods or iterative methods can be used for solving the problem. In this paper, an intelligent parallel iterative method for solving linear systems of equations with large sparse matrices is proposed and its efficiency is proved through simulation.

Key words : Linear Systems of Equations, , Parallel Iterative

## I. 서 론

컴퓨터 하드웨어 기술의 향상에 따라 새로운 컴퓨터 적용분야가 증가하고 있으며 특히 고성능 컴퓨터에 대한 요구가 과학 및 공학 분야에서 크게 늘어나고 있다. 이런 추세에 따라 다양한 고성능 컴퓨터가 제작되고 있다[1]. 이런 고성능 컴퓨터 시스템에 적용하기 적절한 병렬알고리즘도 꾸준히 개발되어왔고 그에 대한 연구도 다양하고 활발하다[2]-[5].

특히 선형시스템 방정식의 해를 구하는 문제는 공

* 한국항공대학교 항공전자정보통신공학부
** 한국항공대학교 컴퓨터공학과
· 제1저자 (First Author) : 채수환
· 투고일자 : 2009년 1월 9일
· 심사(수정)일자 : 2009년 1월 14일 (수정일자 : 2009년 2월 23일)
· 게재일자 : 2009년 2월 28일

학, 과학 분야에서 가장 기초가 되고 핵심이 된다. 선형시스템의 성격에 따라 그 해를 구하는 방법이 달라진다. 그동안 밴드(banded) 구조, 대칭구조 등 잘 정형화된 행렬구조를 가진 선형시스템의 해를 구하기 위한 병렬알고리즘들이 많이 개발되어 왔다[6]-[8].

본 논문에서는 행렬의 구조에 관계없이 대형이고 스파스(sparse)한 행렬을 가지는 선형시스템에 대하여 다룬다. 여기에서 스파스하다는 말은 행렬의 요소 중에 값이 0 인 비율이 매우 높다는 것을 의미한다. VLSI 회로의 루프(loop) 방정식의 해를 구하는 문제, 열역학, 구조역학 등에서 사용되는 편미분방장식의 해를 구하는 문제의 경우, 시스템의 규모가 커짐에 따라 행렬의 크기는 기하급수적으로 증가하면서 더욱 스파스해진다. 이런 경우에 대용량의 메모리가 필요하다. 이런 문제를 해결하기 위해 직접법보다 반복법이 유용하다. 그런데 반복법을 적용하기 위해서는 행렬이 대각적으로 우세(strictly diagonally dominant) 해야 된다. 그런데 위의 문제를 풀기 위한 행렬들은 이런 조건을 만족한다[9].

여러 반복법들 중에서 Jacobi 반복법은 대량의 병렬성을 추출할 수 있고 병렬처리 중에 발생하는 통신시간을 줄일 수 있다[10].

따라서 본 논문에서는 Jacobi 반복법을 병렬처리 알고리즘으로 채택했고 스파스한 행렬의 성격을 효율적으로 이용하여 처리시간을 줄이도록 하였다. 이 알고리즘을 적용하기 위한 적절한 병렬컴퓨터시스템의 구조는 (그림 1)과 같다. 여기에서 프로세서의 수는 제한을 두지 않았다.

## II. 병렬 Jacobi 반복법

선형시스템 방정식은 식 (1)과 같다.

$$Ax = b, \qquad (1)$$

여기에서 A 는 n x n 계수 행렬이고, x 는 $n$ x $l$ 미지(unknown) 벡터이고, b는 $n$ x $l$ 상수 벡터이며 n은 벡터나 행렬의 크기이다. 이 행렬 A 는 $D-L-U$로 나눌 수 있는데, 여기에서 D 는 행렬의 주 대각부분 (principal diagonal part), L 은 하부 삼각부분 U 는 상부 삼각부분이다. 이 때 D, L, U 는 모두 $n$ x $n$ 행렬이다.

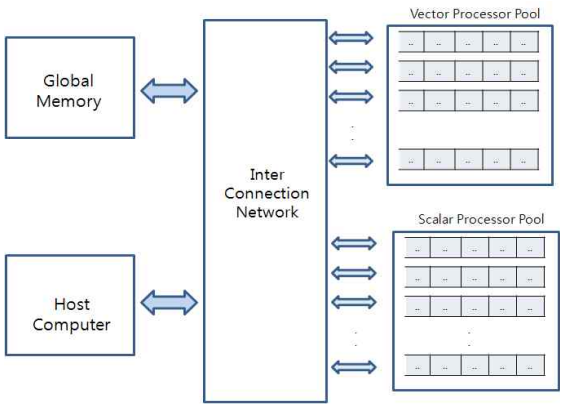


그림 1. 적용 병렬컴퓨터시스템
Fig. 1. Application of Parallel Computing System

식 (1)은 다음과 같이 변형되어 표시될 수 있다.

$$\begin{aligned}(D-L-U)x &= b \qquad (2)\\ x &= D^{-1}(L+U)+D^{-1}b\\ &= Tx+c\end{aligned}$$

여기에서 $T=D^{-1}(L+U)$이고 $c=D^{-1}b$ 이다.

Jacobi 반복법은 식 (2)로 표시된 행렬식을 바탕으로 한다. 이 문제를 해결하기 위해 먼저 초기 벡터 $x^{(0)}$를 시작으로 구하려는 해 벡터 x에 반복하여 오차를 줄이면서 접근하는 방법이다. 이 접근하는 과정을 식(3)에 나타냈다.

$$x^{(k)} = Tx^{(k-1)} + c \qquad (3)$$
여기에서 $k=1,2,3\ldots$

이 반복을 끝내기 위해 벡터의 놈(norm)을 이용한다. x 벡터의 놈은 식 (4)로 표시할 수 있다.

$$\|x\| \text{ for } x=\{x_1, x_2, \ldots x_n\}^t \qquad (4)$$
$$\max|x_i|,\ 1 \le i \le n$$

일련의 접근되는 벡터는 해에 접근하여 주어진 오차 범위에 달했을 때 수행을 마친다. 이를 식 (5)에

나타냈다.

$$\| x^{(k)} - x \| < \varepsilon \quad (5)$$

적용할 때, 실지의 정확한 해 벡터 x를 알지 못함으로 식 (6)이 사용된다.

$$\| x^{(k)} - x^{(k-1)} \| < \varepsilon \quad (6)$$

(그림 2)에 병렬 Jacobi 반복법을 나타냈고 $Nmax$을 최대 반복수로 정하였다. 만약에 너무 수렴 속도가 늦으면 너무 오랜 시간이 걸리므로 이 값을 넘으면 프로그램 수행을 중단하기 위해서이다.

```
Begin
    k=0;
    while k < Nmax do
      broadcast the new   ;
        for all 1≤ i ≤ n do
            xi(k) = Ti · x (k-1) + ci ;
        end for all
      collect x(k) ;
      if || x(k) - x(k-1) || < ε then terminate;
    end while
end
```

그림 2. 병렬 Jacobi 반복법
Fig. 2. Parallel Jacobi Iteration.

## III. 제안한 지능적 Jacobi 병렬반복법

(그림 2)에서 보는 바와 같이 병렬 Jacobi 반복법은 두 개의 과정으로 되어 있다. 하나는 계산하는 과정이고 다른 하나는 제어하는 과정이다. 제어하는 부분은 새로운 x(k)를 모든 벡터 프로세서에게 전달하고 연산 후 새로운 벡터를 모으고 식(6)에 의해 반복 여부를 결정짓는 부분이다. 이 부분은 (그림 1)에 나타낸 host 컴퓨터가 직렬적으로 수행한다. 따라서 이 논문에서는 대량의 병렬성을 추출할 수 있는 계산하는 부분에 중점을 두었다.

계산하는 부분은 다시 두 부분으로 나눌 수 있다.

- 행렬 $T_i$ 와 벡터 $x^{(k-1)}$ 내적을 구하는 부분
- 구해진 내적 벡터와 벡터 $c_i$을 더하는 부분이다.

본 논문에서는 대량의 연산을 요구하는 내적을 구하는 문제에 초점을 맞추었다. 식 (2)에 표시한 행렬 T는 대형이고 스파스하며 행렬 T의 각 열의 스파스한 정도가 불규칙(irregular)하다. 본 연구에서는 두 가지 사항에 관심을 두었다.

1) 지능적인 방법으로 스파스한 특성을 이용한다.
2) 내적 연산 파이프라인을 가진 대량의 벡터프로세서를 사용한다.

(그림 2)에서 보는 바와 같이 연산하는 과정에서 행렬 T 값과 벡터 c 값은 변하지 않고 오직 벡터 x 값만 변한다. 따라서 반복 수행 전에 (그림 3)에 보인 바와 같이 행렬 T 중에서 요소 값이 0인 것을 제거한 새로운 축약 행렬 T'를 만들고 각 행의 0이 아닌 (nonzero) 요소의 인덱스(index) 값을 인덱스 행렬에 저장한다. 축약 알고리즘을 (그림 4)에 나타냈다.

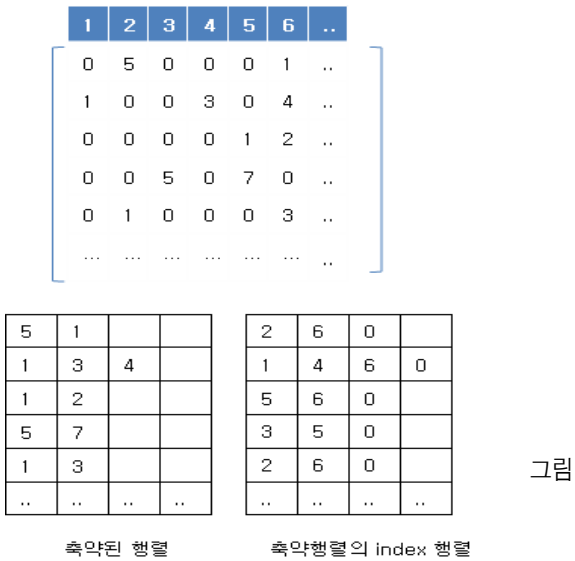


그림 3. 스파스 행렬 축약 예
Fig. 3. Example of reduction in Sparse Matrix.

제안한 지능적 병렬처리 방법은 (그림 5)와 같다.

```
procedure compact(matrix T, index)
 begin
  for i =1 to n do
    counter = 0;
    for j =1 to n do
```

```
        if (Tij = 0 ) then skip
        else begin
            counter++;
            T(i,counter) = T(i,j);
            index(i, counter)= j ;
        end
    end for
end
```

그림 4. 행렬 축약 알고리즘
Fig. 4. Algorithm for Matrix Reduction.

```
begin
 compact (matrix T, matrix index);
 initialize x(0);
 loop_counter =0;
 while counter < Nmax do
   broadcast the new ;
   for all 1≤ i ≤ n do
    do intelligent processing of x(k) = Ti · x (k-1) + ci
        using matrix index;
    end for all
   collect x(k) ;
  if || x(k) - x(k-1) || < ε then terminate;
  loop_counter++;
 end while
end
```

그림 5. 제안한 지능적 Jacobi 병렬 방법
Fig. 5. Proposed Intelligent Parallel Jacobi Iterative method.

## Ⅳ. 제안한 방법의 평가

제안한 지능적인 방법에 의해 벡터 처리 과정의 예를 (그림 6)에 나타냈다. (그림 6)은 (그림 3)에서 보인 행렬의 첫 번째 행벡터와 $(x)^t$ 벡터의 연산 과정이다. 내적 파이프라인은 인덱스 값이 0이 되었을 때, 연산의 끝으로 간주한다. 파이프라인의 하나의 입력 열(queue)은 행렬의 첫 번째 축약된 행벡터 값이 순서대로 들어가며 다른 입력 열은 인덱스 행렬에서 미지의 x 벡터의 인덱스 값을 제공하여 x 벡터 중에 해당되는 x[index] 값만 들어온다.

병렬 벡터의 내적을 수행하기 위해 (그림 1)에 보인 바와 같이 내적 파이프라인을 가진 다량의 벡터 프로세서가 사용된다.

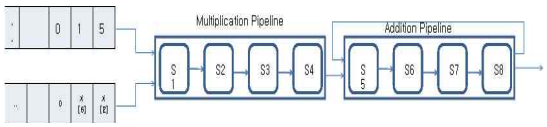


그림 6. 벡터의 내적 수행 방법
Fig. 6. Inner Product Operation on Vectors

벡터 파리프라인의 수행시간, $Tp$는 식 (7)로 나타낼 수 있다.

$$Tp = Ns * Tc + [Ns + (Nt - 1)] * Tc \quad (7)$$

여기에서 $Tc$는 파이프라인 사이클 시간, $Nt$는 벡터의 크기, $Ns$는 파이프라인 스테이지(stage) 수를 나타내며 여기에서는 (그림 6)에 보인 바와 같이 $Ns$ = 8로 정하여 시뮬레이션 한다. $Ns * Tc$는 이상적인 경우, 최초 출력시간(startup time : $Tf$)이 된다. 즉, $Tf$는 벡터 처리시 파이프라인에서 첫 번째 연산 결과가 나오는 데 까지 걸리는 시간을 말한다.

$Tc$는 식 (8)에 의해 정해진다.

$$Tc = \max_{i=1,\ldots,Ns}\{Tc + D_i\} \quad (8)$$

여기에서 $D_i$는 각 스테이지 레지스터 지연시간을 의미하며 시뮬레이션에서는 무시된다. $Tc$는 각 스테이지의 수행시간으로 시뮬레이션의 기본 단위가 된다.

(그림 5) 알고리즘에서 보인 바와 같이 필요로 하는 행렬 축약에 소요되는 시간은 처음 시작할 때 필요하고 한 번 축약된 값들과 인덱스 값들은 변하지 않기 때문에 반복과정에서 그대로 사용된다. 따라서 본 평가에서는 반복 중에 1회 수행 시간을 시뮬레이션하기 때문에 스파스 행렬의 축약시간, $Tc$는 반복 횟수로 나누어 계산된다.

(그림 4)에 나타난 바와 같이 스파스 행렬을 축약하는데 소요되는 명령의 수, Totalcomp를 다음과 같이 계산하였다.

- outer loop 시작점에서 n과 비교하는 명령: n개의

compare 명령어

- clear counter 명령 : n 개의 clear 명령
- inner loop 시작점에서 n과 비교하는 명령: $n^2$개의 compare 명령어
- 행렬 요소가 0 인지 비교하는 명령: $n^2$개의 compare 명령어
- 행렬 요소의 0 여부에 따라 branch 주소를 결정하는 $n^2$개의 conditional branch 명령
- 행렬 요소가 nonzero 인 경우, 4개의 명령이 필요하다. ( counter 증가를 위한 increase 명령, 해당 값을 compact 행렬에 저장하고 그 열(column) 인덱스를 index 행렬에 저장하기 위한 2개의 store 명령, 다시 loop를 시작하기 위한 branch 명령)

따라서 $Totalcomp$는 식 (9)로 나타낼 수 있다.

$$Totalcomp = (4*Sf+3)*n^2 + 2n \qquad (9)$$

여기에서 $Sf$는 T 행렬에서 0이 아닌 요소의 비율을 나타낸다.

$Tf$가 파이프라인에서 두 개의 명령을 수행하는 시간이기 때문에 하나의 명령을 수행하는 시간을 $0.5*Tf = 4*Tc$로 정하여 시뮬레이션 하므로 식 (9)는 식 (10)으로 표현할 수 있다.

$$Totalcomp = 4*\{(4*Sf+3)*n^2 + 2*n\}*Tc \qquad (10)$$

이 축약시간을 1회 반복하는 시뮬레이션에 반영하기 위해서는 Totalcomp 값을 (그림 5) 알고리즘에 나타난 loop_counter로 나누어야 한다. 본 시뮬레이션에서는 loop_counter를 100으로 가정하였다. 따라서 본 시뮬레이션에서는 식 (11)이 사용되었다.

$$Totalcomp/loopcounter = 0.04*\{(4*Sf+3)*n^2 + 2*n\}*Tc \qquad (11)$$

시뮬레이션에 사용되는 주요 파라메터들(parameters)을 (표 1)에 나타냈다.

(그림 7) 에 시뮬레이션 결과를 나타냈다.

표 1. 시뮬레이션 파라메터
Table 1. Parameters of Simulation.

| 기호 | 내용 |
|---|---|
| Sf | nonzero 요소의 비율 |
| Tc | 파이프라인 사이클 시간 |
| Tf | Start up 시간 ( = 8) |
| Ns | 파이프라인 스테이지 수 (= 8) |
| Totalcomp | 축약시간 |

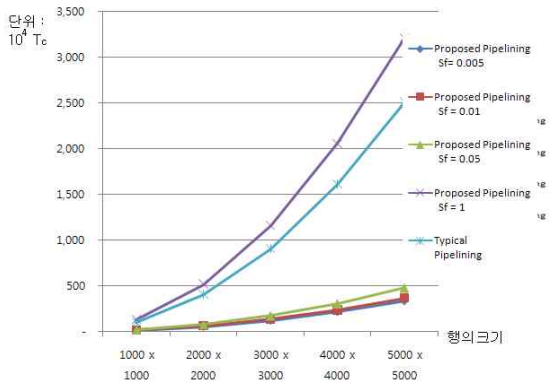


그림 7.시뮬레이션 결과
Fig. 7. Results of simulation.

(그림 7)에서 보는 바와 같이 시뮬레이션을 통해 Sf가 작을수록 제안한 지능적 방법이 효율적임을 알 수 있다. 이 방법을 밀집(dense)한 행렬을 가진 시스템에 적용할 경우, 오히려 성능이 나빠짐을 알 수 있다.

Sf의 값에 의한 효과를 (그림 8)에 나타냈다.

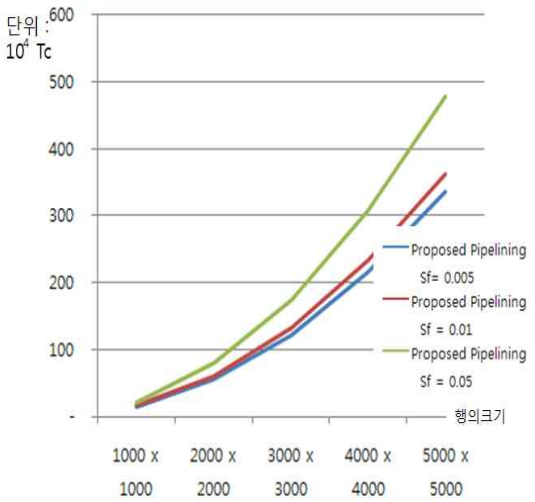


그림 7. Sf의 값에 의한 효과
Fig. 8. Effect of value of Sf

## V. 결론 및 장래 연구

병렬성의 추출은 병렬처리에서 가장 중요한 문제이다. 제안한 방법을 통해 최대한도로 병렬성을 추출하였고 스파스한 대형 행렬을 처리함에 있어 지능적인 방법을 통해 선형시스템 방정식의 해를 구하는데 소요되는 시간을 단축할 수 있었다. 앞으로 이 방법을 좀 더 많은 공학 및 과학 분야에 적용하기 위해 하드웨어적인 접근을 할 필요가 있다. 또한, 본 연구에서는 벡터 프로세서의 수에 대한 제한을 두지 않았지만, 실용적으로 적용하기 위해서는 벡터 프로세서 및 스칼라 프로세서가 제한된다. 이에 벡터 프로세서와 스칼라 프로세서를 가진 병렬처리 시스템에 적용하기에 효율적인 지능적 방법을 연구할 것이다.

## 감사의 글

"본 논문은 지식경제부 한국산업기술평가원 지정 한국항공대학교 부설 인터넷정보검색 연구센터의 지원에 의함"

## 참 고 문 헌

[1] D.E. Culler and J. P. Singh, *Parallel Computer Architecture,* Morgan Kofmann, 1999.

[2] E. Hagersten and G. Papadopoulus, "Parallel computing in the commercial marketplace: research and innovation at work," *Proceeding of the IEEE,* pp. 405-410, March, 1999.

[3] E. T. Chong, B. Lim, R. Bianchini, and J. A. Argarwal, "Application performance on the MIT Alewife machine," *Computers*, pp.57-64, Dec. 1996.

[4] K. Hwang, "Advanced parallel processing with supercomputer architecture," *Proceeding of the IEEE,* pp.1348-1378, Oct. 1987.

[5] P. Srini, "An Architectural comparison of dataflow systems," *Computers,* pp. 66-88, March, 1985.

[6] T. Opsahl and D. Parkinson, "An algorithm for solving sparse sets of linear equations with almost tridiagonal structure on SIMD computers," *Proceeding International Conf. Parallel Processing,* pp. 369-374, 1986.

[7] G. R. Rao, "A pipelined solution method of tridiagonal linear equation systems," *Proceeding International Conf. Parallel Processing,* pp. 84-91, 1986.

[8] T. Dehn, M. Eiermann, K. Giebermann, and V. Sperling, "Structured sparse matrix-vector multiplication on massively parallel SIMD architectures," *Parallel Computing,* 1987-1895, Dec. 1995.

[9] R. L. Burden and J. D. Faires, *Numerical Analysis, Brooks/Cole Thomson Learning Inc.*, 2001.

[10] L. H. Jamieson, D. B. Gannon, and R. J. Douglass, *The Characteristics of Parallel Algorithms, MIT Press,* 1987.

### 채 수 환 (蔡洙煥)

1973년 한국항공대학교 항공전자공학과 졸업(공학사)
1985년 미국 Univ. of Alabama 전자계산학과 졸업(공학석사)
1988년 미국 Univ. of Alabama 전기공학과 졸업(공학박사)
1989~ 현재 한국항공대학교 항공전자정보통신공학부 교수
관심분야 : 컴퓨터 구조, 병렬처리 시스템 등임

### 김 명 규 (金明奎)

2002년 한국항공대학교 컴퓨터공학과 졸업(공학사)
2004년 한국항공대학교 컴퓨터공학과 졸업(공학석사)
현재 한국항공대학교 컴퓨터공학과 박사과정
관심분야 : High Performance Computing, 자연어 처리 등임