반복기법을 이용한 대규모, 소(疏)선형 시스템의 병렬처리에 관한 연구

(An Experimental Study on Parallel Implementation of an Iterative Method for Large Scale, Sparse Linear System)

김 상 원 · 장 수 영

포항공과대학 산업공학과

Abstract

This thesis presents a parallel implementation of an iterative method for large scale, sparse linear system and gives result of computational experiments performed on both single transputer and multi transputer parallel computers. To solve linear system, we use conjugate gradient method and develope data storage techinique, data communication scheme.

In addition to the explanation of conjugate gradient method, the result of computational experiment is summarized.

1. 서 론

많은 이, 공학관련 계산문제 (Scientific problem) 을 해결하는데 있어서 선형시스

템의 해를 구하는 과정이 중요한 역할을 하며 이의 일반적인 형태는 다음과 같다.

$$Ax = b$$

(단, A : 계수행렬 (Coefficient Matrix)
b : 상수벡터 (Constant Vector)
x : 미지수 벡터 (Unknown Value vector))

윗 시스템에서 방정식의 갯수가 미지수보다 많으면 Overdetermined System, 미지수가 방정식의 갯수보다 많으면 Underdetermined System, 미지수와 방정식의 갯수가 같으면 Square System이라고 부른다.

이와같은 선형시스템 을 해결하는 방법으로는 크게 직접법과 반복법이 있으며, 직접법에는 Gaussian Elimination, Gauss-Jordan Elimination, LU-Decomposition, Cholesky's Method 등이 있고, 반복법으로는 Jacobi Method, Gauss-Seidel Method, SOR Method (Successive Overrelaxation Method) , Steepest Descent Method, Conjugate Direction Method, Conjugate Gradient Method, Preconditioned Conjugate Gradient Method등이 있다. 실제문제에서 발생하는 선형시스템, Ax = b에 나타나는 A 행렬은 대규모이며 행렬내의 원소의 대부분이 0을 원소값으로 가지는 소(疏)한 경우가 많다 (이를 Large Scale, Sparse Matrix라 한다). 이때 직접법을 사용하게 되면 원래 행렬이 소(疏)하더라도 해를 구해나가는 과정에서 행렬의 모양이 변형되어 0을 값으로 가지던 행렬의 원소가 0이 아닌 값을 가지게 된다 (이를 Fill-in 이라 한다). 그러나 반복법의 경우는 원래행렬의 형태가 그대로 유지되므로 Fill-in이 발생하지 않는다. 따라서 직접법을 사용하면보다 반복법에 비해 많은 기억장소가 필요하며, Fill-in의 발생으로 불필요한 계산시간이 추가되므로 소

(疏)한 행렬을 다루는 경우 반복법이 보다 효과적인 방법이라 할 수 있다.

반면에 특수한 하드웨어 (Hardware)를 사용하는 것도 선형시스템을 효율적으로 해결하는데 매우 유용한 도구가 된다고 하겠다. 이러한 하드웨어로는 병렬처리를 가능케 하는 병렬처리컴퓨터를 들 수 있는데 병렬처리는 많은 계산을 여러개의 컴퓨터가 나누어 처리하는 것과 같은 효과를 가지므로 적은비용으로 높은 계산능력을 얻을 수 있다.

따라서 본 연구에서는 이러한 취지에서 선형시스템의 형태가 대규모이며 소(疏)한 경우에 대하여 반복법의 하나인 Conjugate Gradient Method에 준한 병렬기법을 개발하여 여러개의 Transputer를 병렬로 조합한 특수 하드웨어에 구현함을 목적으로 한다. 또한 이러한 병렬기법을 계산실험을 통하여 행렬의 밀도(Density)와 크기에 따른 그 효율성의 변화를 규명해 보기로 한다.

2. Conjugate Gradient Method 의 병렬처리

2-1. Conjugate Gradient Method

본 연구에서는 선형시스템, $Ax = b$를 푸는데 여러가지 반복법중 Conjugate Gradient method를 사용하였다. Conjugate Gradient Method는 다음의 Conjugate Direction Theorem을 기초로한 반복법이다.

일반적인 선형시스템, $Ax = b$ 를 해결하는데 있어서 행렬 A 가 $n \times n$, Positive Definite 행렬이며 $d^0, d^1, ..., d^{n-1}$이 Conjugate Direction 벡터이면, 다음 반 복식, $x^{k+1} = x^k + \alpha^k d^k$는 정확히 적절히 선택된 α값 (그림 1참조) 을 취하면 정확히 n 스텝 이전에 해에 도달한다.

(윗 정리에서 Positive Definite 행렬의 정의는 x가 0이 아닌 모든 벡터에 대하여 $xQx^T > 0$ 을 만족시키는 행렬 A를 말하며 Conjugate Direction 벡터는 다음과 같은 성질,

$$d^i A d^j = 0, \quad i \neq j$$

을 만족하는 벡터 d^i 들을 말한다.)

Conjugate Gradient Method는 Conjugate Direction 벡터를 매 스텝마다 하나씩 결정하며, 현재스텝에서 얻은 Gradient와 전스텝에서 사용했던 Conjugate Direction벡터의 선형 조합으로 다음 스텝에서의 Conjugate Direction 벡터를 얻으며 n차원 벡터공간에서 n개 Conjugate Direction 벡터를 이용하면 정확히 n 스텝이내에 해에 수렴한다는 것이 수학적으로 보장되어 있는 방법이다. 그림 1 은 Conjugate Gradient Method를 정리한 것이다.

choose x^0, set $r^0 = A x^0 - b$, $d^0 = -r^0$

for k=0,1,2,....

$$\alpha^\kappa = -(r^\kappa)^T (r^\kappa) / (d^\kappa)^T A (d^\kappa)$$

$$x^{\kappa+1} = x^\kappa + \alpha^\kappa d^\kappa$$

$$r^{\kappa+1} = r^\kappa + \alpha^k A (d^\kappa)$$

test for convergence

$$\beta^\kappa = (r^{\kappa+1})^T (r^{\kappa+1}) / (r^\kappa)^T (r^\kappa)$$

$$d^{\kappa+1} = -r^{\kappa+1} + \beta^\kappa d^\kappa$$

그림 1. Conjugate Gradient Method

그림 1에 나타난 바와 같이 Conjugate Gradient Method의 주요계산은 밑줄친 행렬과 벡터의 곱을 계산하는 과정이다. 따라서 이 부분을 병렬처리하는 것이 가장 큰 효과를 내리라고 생각된다.

2-2. 병렬처리 (Parallel Processing)

2-2-1. Parallelization & Vectorization

병렬처리(Parallel Processing) 에는 크게 두가지 종류가 있다. 그 중 하나는 Parallelization으로 이는 하나의 일(Job) 을 수행하는데 있어서 여러개의 프로세서 (Processor)를 사용하여 일을 분할, 이를 각각의 프로세서로 보내어 처리하는 것을 말한다. 나머지 하나는 Vectorization으로 여러개의 계산장소를 보유한 하나의 프로세서내에 서 동일한 수리계산 (+, -, x 등) 을 여러곳에서 할 수 있게 하여 하나의 일을 분할 처리할 수 있게 만든 것이다. 본 연구에서는 선형시스템 을 풀기 위해서 Parallelization을 사용하였다.

소(疏)한 행렬을 병렬처리하는 것은 서론에서 말한바와 같은 장점도 있으나 반면 이로 인해서 기계가동율이 낮아지고 데이타접근(Data Access)이 복잡해진다. 따라서 이를 적용시키는데 다음 사항을 고려해야만 한다.

1. 데이타 저장 (Special Storage Techniques)
2. 데이타 커뮤니케이션 (Data Communication Scheme)
3. 병렬처리가 가능한 부분 선정 (Parallel Processing Part)

이러한 사항을 고려하는 것은 다음과 같은 이유 때문이다.

1. 제한된 기억용량으로 인하여 데이타 저장장소를 최소화해야한다.
2. 많은 양의 데이타를 여러개 트랜스퓨터 (Transputer)를 이용해 처리하므로 교환하여야 할 데이타가 가능한 빠른 시간내에 정확히 전달되어야만 한다.
3. 가장 계산부담이 큰 부분을 찾아 병렬처리를 하여야 최고의 효율을 얻을 수 있다.

2-2-2. 병렬처리 계획

병렬처리를 위하여 4개의 Transputer로 구성된 병렬처리보드를 퍼어스널컴퓨터와 연결 시켜 사용하였다. (그림 2 참조) 각 트랜스퓨터에는 4개의 포트(Port)가 존재하며 이중 3개가 나머지 트랜스퓨터 와 연결되어 있다.

Transputer 1 은 Root Transputer라 부르며 유일하게 퍼어스널 컴퓨터와 연결되어 있다. 따라서 이를 통해서만이 나머지 트랜스퓨터를 사용할 수 있다.

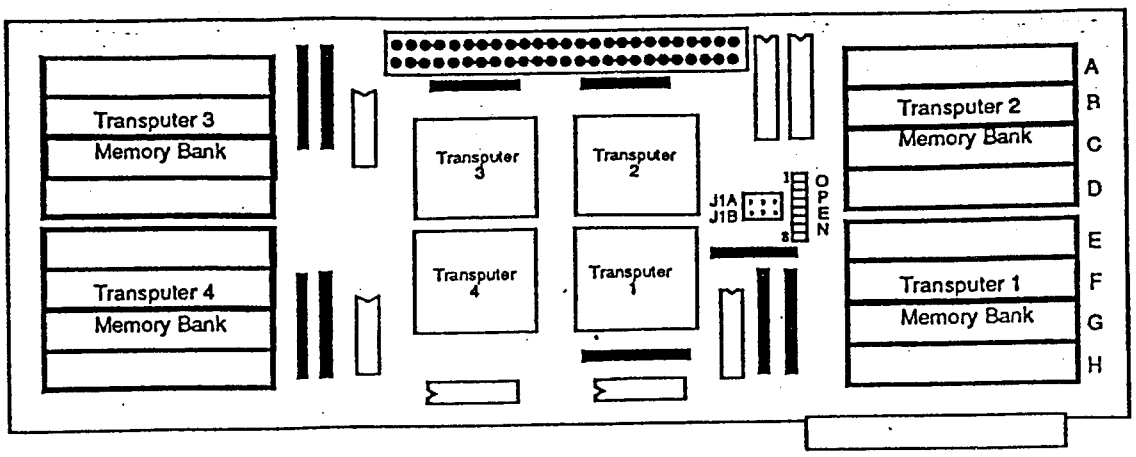

그림 2. 본 연구에 사용된 병렬컴퓨터의 구조

1) 데이타 저장계획

소(疏)한 행렬을 처리하기 위해서는 특별한 데이타 저장계획(Data Storage Scheme)이 필요하다. 이는 많은 수의 0를 저장하지 않음으로써 기억장소를 줄일 수 있는 이점이 있다. 본 연구에서는 Uncompressed Storage Scheme을 사용하였는데 이 방법은 세개의 일차원 벡터를 사용하는 것으로 그림3은 이를 나타낸다. 여기에서 벡터 a는 0이 아닌 원소를 나타낸다. 벡터 ja는 a에 있는 원소의 열위치를 나타내며 벡터 ia는 각 행에 있는 첫번째로 0이 아닌 원소의 위치를 지시하기 위해서사용한다. 이러한 저장계획을 사용하면 n x m 행렬사용시 0이 아닌 원소가 x개이면 총 2*x + n+1 개의 기억장소가 필요하다.

$$\begin{pmatrix} 0 & 0 & a_{13} & 0 & 0 \\ a_{21} & 0 & 0 & 0 & a_{25} \\ 0 & 0 & 0 & 0 & 0 \\ a_{41} & a_{42} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix}$$

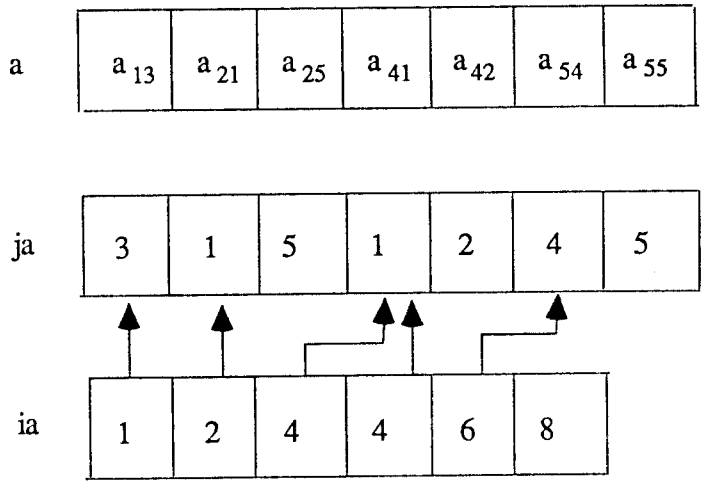

그림 3. 데이타 저장계획(Uncompressed Storage Scheme)

2) 데이타 분할 계획

병렬처리는 동일하며 반복적인 계산을 여러 Transputer에 나누어 처리함으로써 계산시간을 절약하자는데 목적이 있다. Conjugate Gradient Method를 병렬처리 하는데는 행렬과 벡터의 연산(그림 1에서 $A^k(d)$)이 전체 계산중 가장 많은 비중을 차지함으로 이를 4개의 Transputer에 나누어 처리하였다. 우선 소(疏)한 형렬의 각 행의 0이 아닌 원소수를 파악하여 계산량이 4등분이 되도록 데이타 저장계획에 따라 a 벡터를 분할한후, 이와 관련이 있는 ia, ja 벡터의 원소들만을 각 Transputer로 전달하였다. 다음에는 모든 Transputer에 공통적으로 전달되어야 하는 Conjugate Direction 벡터(그림 1의 d^k)를 분할없이 전달하였다.

3) 데이타 커뮤니케이션 계획

데이타 커뮤니케이션은 그림 4. 에서 보는 바와 같이 Root Transputer 에서 각각의 Transputer로 전달하여, 연산후 받아들이는 과정을 화살표로 나타냈다. 여기서 화살표 1은 Root Transputer에서 각각의 Transputer로 a, ia, ja 벡터의 원소들을 보내는 것으로 한 번만 전달한다. 화살표 2는 Conjugate Direction 벡터를 전달하는 것으로 t 2(Transputer 3)는 Root Transputer로부터 직접 받는 것이 아니고 t 1(Transputer 2)으로부터 전달받는다. 이는 Root Transputer 에서 각 Transputer로 보내는 것보다 Transputer 1에서 Transputer 2로 보내는 동시에

Root Transputer에서 Transputer 3으로 보냄으로써 Transputer 2로 보내는 시간을 절약할 수 있는 것이다. Driver, Tsk1, Tsk2, Tsk3는 각 Transputer에서 수행되는 프로그램을 말하며 Filter와 Afserver는 Root Transputer와 퍼어스널 컴퓨터를 연결시키는 기능을 갖는다.

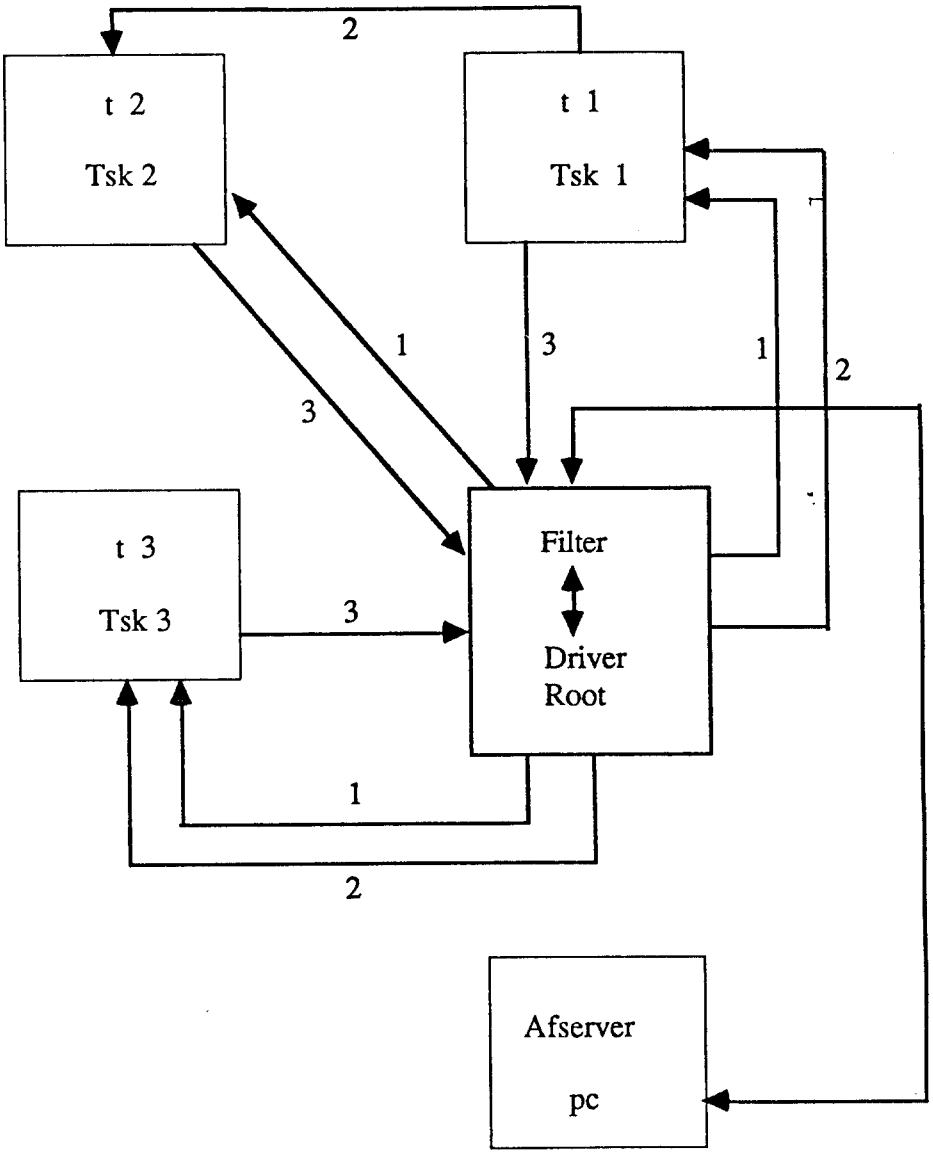

그림 4. 데이타 커뮤니케이션

3. 실험결과

3-1. 입력자료 생성과정

소(疏) 선형 시스템 , Ax=b를 만들기 위해서 다음 그림 5 에서 보는바와 같이 우선 주대각선에 위치하는 원소를 $n^2 \times k$ ($k = 1,...n$) 으로 하고, 1을 n 개씩 더하여 Density를 1/n 씩 증가시키는 방법을 사용하여 A행렬을 만들었다.

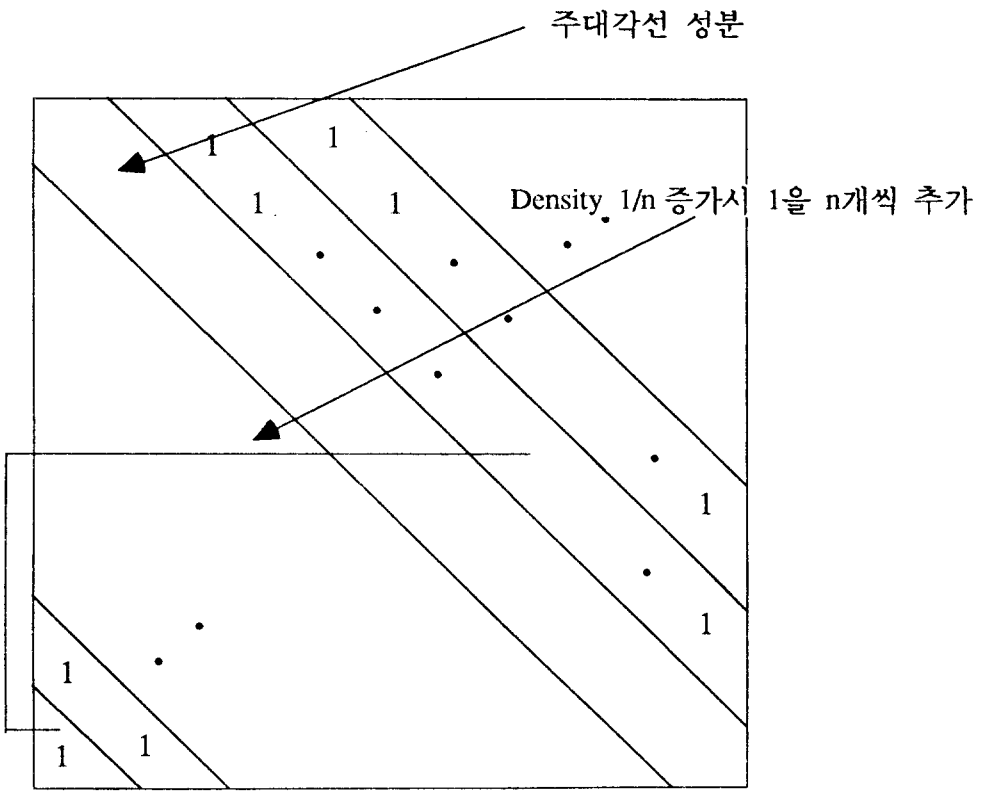

그림 5. 소(疏)한 행렬의 생성 과정

이러한 과정을 통해서 얻어진 정방행렬 A는 모든 행의 0이 아닌 원소의 갯수가 같으며, Positive Definite 행렬이 된다. Right Hand Side, b는 해가 (1,...,1) 이 되도록 A행렬에 모든 원소를 1로 갖는 벡터, x를 곱하여 얻어진 값을 사용하였다.

3-2. 실험결과

5가지의 Density (20%, 40%, 60%, 80%, 100%) 를 갖는 행렬을 대상으로 (크기는 각 Density에 대하여 40x40, 80x80, 120x120, 160x160, 200x200 5가지임) Single Transputer와 4개 Transputer를 사용하여 얻은 속도비 (Speed-up)는 그림 7과 같으며 속도비는 다음과 같이 표시할 수 있다.

$$\text{Speed-up} = \frac{f(n, d, m)}{g(n, d, \kappa, \delta, m)}$$

f (n, d, m)는 Single Transputer 이용시, 해를 구하는데 걸리는 총시간을 행렬의 차원 (n), Density (d), Conjugate 스텝 수 (m)의 함수로 나타낸 것이며, g(n, d, κ, δ, m)은 4개 Transputer의 경우로 κ, δ는 n개의 데이타를 주고 받는데 걸리는 시간을 (κ+ δn)으로 표시하기 위한 상수이며, 이를 간단한 실험을 통하여 구해 보았다. 추정식은 다음과 같다.

$$f(n, d, m) = ((n^2 \times d) \times 2a + 14na) \times m$$

$$g(n, d, \kappa, \delta, m) = (0.25 \times (n^2 \times d) \times 2a + 14na + 2.75(\kappa + \delta n)) \times m + 3n \times 14a + 2(n^2 \times d) \times 14a$$

n : 행렬의 차원
d : Density
a : 계산시간 (덧셈, 곱셈)
m : Conjugate 스텝
κ, δ : 전달시간과 관련된 상수

f(n, d, m) 에서 ($n^2 \times d) \times 2a$는 그림 1에서 밑줄친 A (d^k)를 구하는 부분이며, 14na는 이를 제외한 나머지 덧셈, 곱셈등의 연산을 매 스텝마다 하는데 걸리는 시간이다. 따라서 Conjugate 스텝수가 m이므로 이와같은 연산시간에 m을 곱한 시간이 총 시간이 된다. g (n, d, κ , δ, m) 는 Conjugate Gradient Method를 병렬처리한 알고리즘을 바탕으로 추정되었다 (그림 6 참조). 0.25

x (n^2x d) x 2a 는 A (d^k)를 구하는 시간으로 병렬처리를 하면 4개 Transputer 가 이를 1/4씩 나누어 구하므로 Single Transputer를 사용한 경우의 1/4의 시간이 소요된다. 14na는 Single 경우와 마찬가지이며, 2.75 (κ+ δ n) 는 그림 6의 (2)에서 보듯이 (d^k)를 각 Transputer에 주고 계산된 A (d^k)를 다시 Root Transputer로 받아들이는데 걸리는 총 시간이다. 3n x 14a + 2(n^2x d) x 14a 는 그림 3에 있는 a, ja, ia 벡터를 알고리즘의 초기화시에 각 Transputer로 보내는데 걸리는 시간이다. 이는 d^k벡터와는 달리 단 한번만 전달하면 된다(그림 6의 (1) 참조). 위의 추정식을 이용하여 얻어진 이론적인 속도비를 나타낸 것이 그림 8이다. 추정치와 실제치에는 어느 정도 차이가 있으나 전체적인 형태는 같음을 알 수 있다. 곡선의 형태로 미루어 볼 때, 행렬의 크기가 커지면 이에 따라서 Speed-up이 계속 증가하지 않고 어느 정도에 이르게 되어 다시 감소 하리라는 것을 추측할 수 있다. 행렬의 크기를 고정시킨 상태에서, Density가 커지면 Speed-up이 증가하게 되는데 이는 Density가 증가함에 따라서 계산량이 많아지게 되므로 전체계산시간중에서 계산부분이 차지하는 비율이 커지게 되므로 4개 Transputer 쪽이 빠르다고 할 수 있다 (단위 데이타 전달시간은 계산시간에 비해 상당히 큼). 또한 Density가 고정된 경우에도 행렬의 크기가 커지면 계산부분의 비율이 커지므로 Speed-up 이 증가하게 된다. 그림 9과 그림 10은 각각 이를 나타내고 있으며 실측치는 이론치보다는 다소 못하나 이론치와 거의 같은 형태를 나타내고 있음을 볼 수 있다.

choose x^0, $r^0 = A x^0 - b$, $d^0 = -r^0$

각 Transputer로 a, ja, ia 벡터(그림 3 참조) 를 4등분하여 전달한다 (1).

for k=0,1,2,.....

(d^K) 를 각 Transputer로 전달시킨다. 각 Transputer는 A (d^K)를 1/4씩 계산한다. 계산된 값을 Root Transputer로 전달한다 (2).

$$\alpha^k = -(r^k)^T(r^k) / (d^k)^T A (d^k)$$

$$x^{k+1} = x^k + \alpha^k d^k$$

$$r^{k+1} = r^k + \alpha^k A (d^k)$$

test for convergence

$$\beta^k = (r^{k+1})^T(r^{k+1}) / (r^k)^T(r^k)$$

$$d^{k+1} = -r^{k+1} + \beta^k d^k$$

그림 6. 병렬처리된 Conjugate Gradient Method

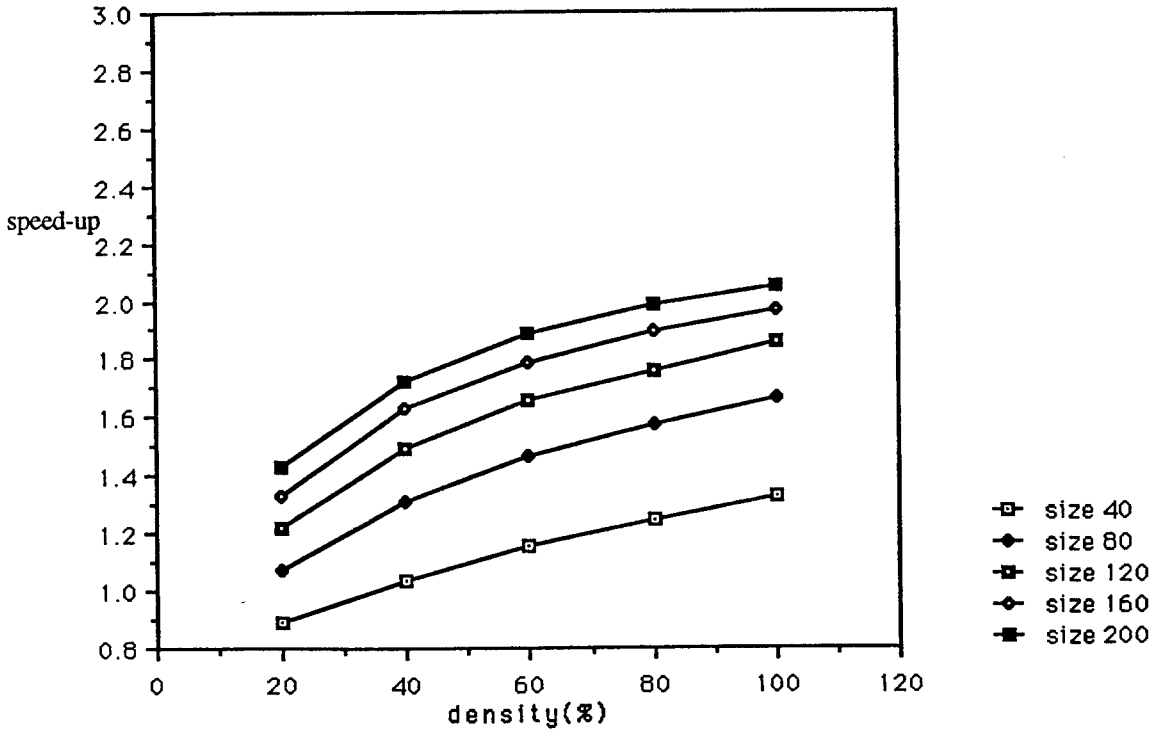

그림 7. density에 따른 실제치 사용시 속도비

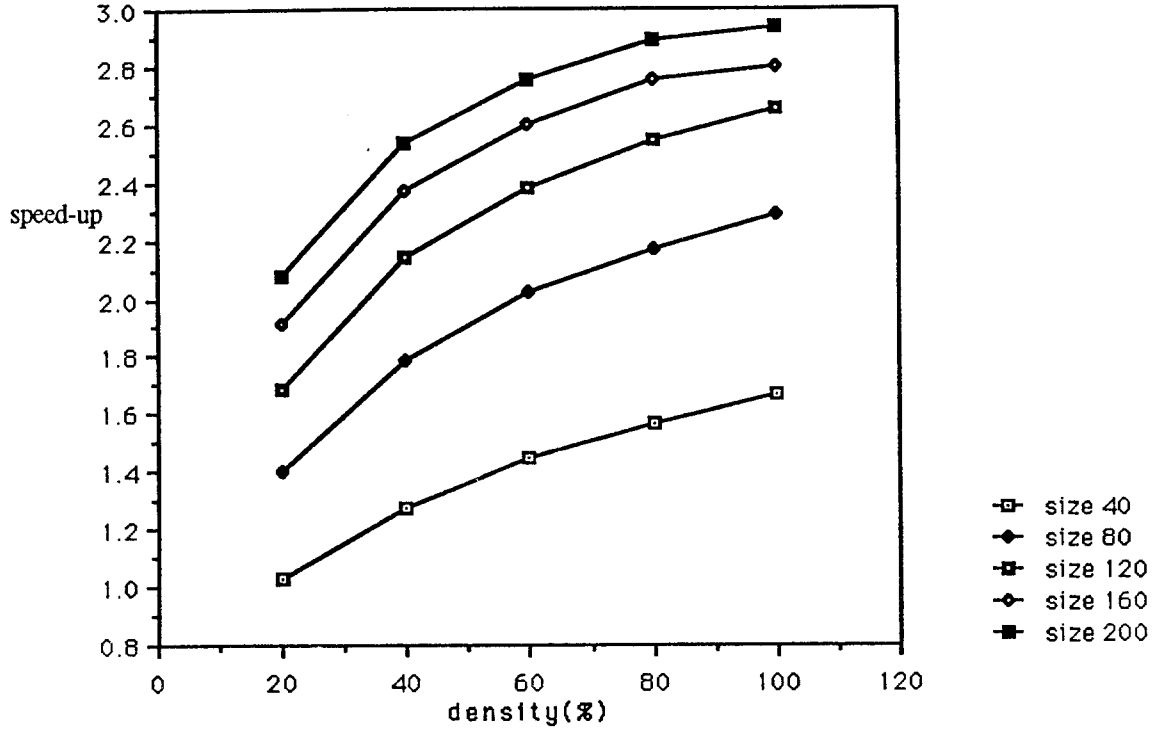

그림 8. density에 따른 이론적 속도비

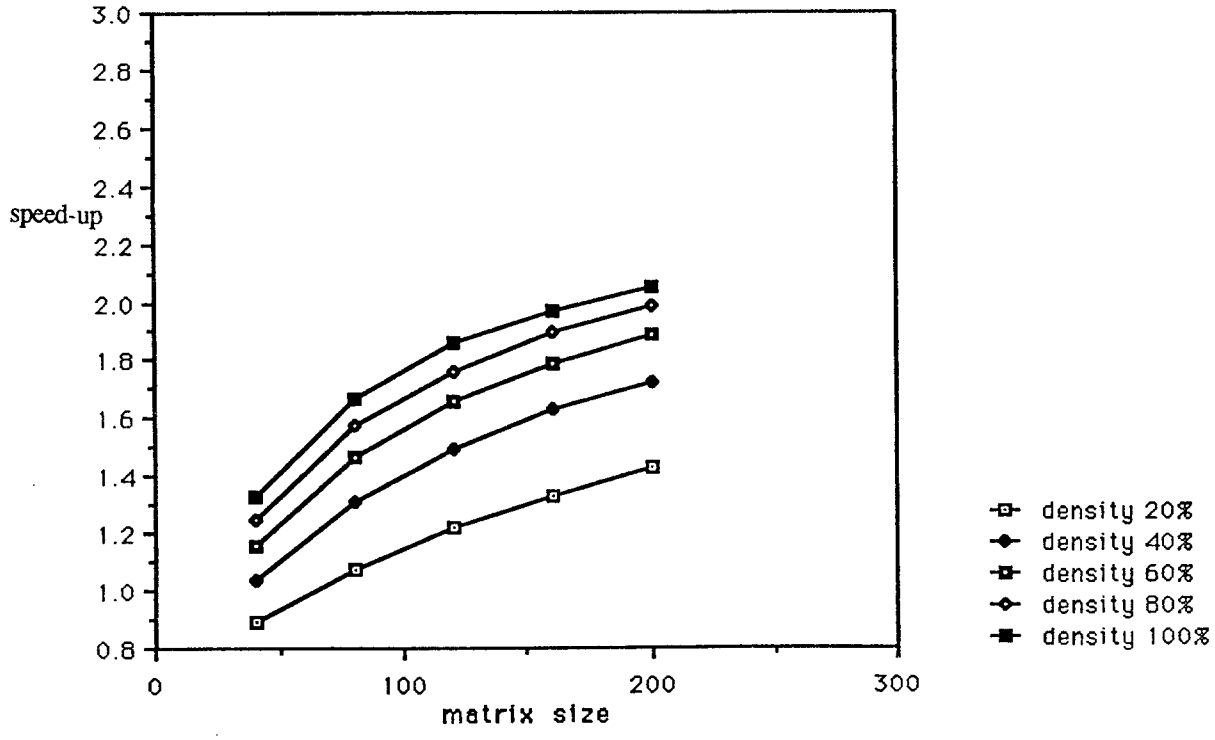

그림9. 행렬크기에 따른 실제치 사용시 속도비

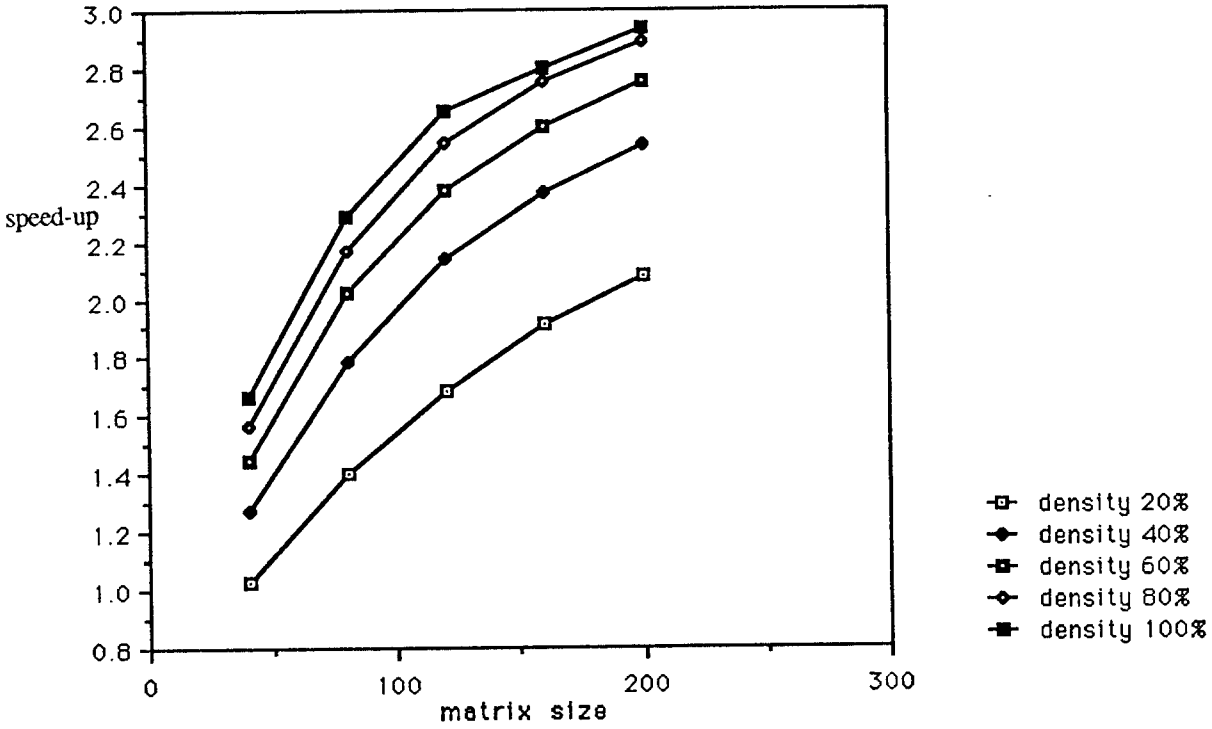

그림10. 행렬 크기에 따른 이론적 속도비

4. 결론

본 연구에서는 대규모, 소(疏) 선형시스템을 풀기 위해 Conjugate Gradient Method를 사용하였으며, 4개의 Transputer가 부착된 퍼어스널컴퓨터용 병렬처리보드를 이용한 병렬처리를 시도하였다.

본 연구에 사용된 Conjugate Gradient Method와 병렬처리는 선형시스템을 해결하는데 유용한 방법이 될 수 있음을 실험을 통하여 발견할 수 있었으며, 행렬의 크기가 크고 Density가 높을수록 4개 Transputer를 사용한 경우가 계산속도가 빠르나 행렬의 크기가 어느 정도에 다다르면 더 이상 Speed-up이 증가하지 않으리라는 것을 추측할 수 있다.

앞으로 4개 이상의 Transputer를 사용하여 병렬처리를 구현하는 경우에는 데이타 커뮤니케이션 문제가 보다 복잡해지므로 무엇보다 이에 대한 연구가 선행되어져야만 할 것으로 믿는다.

참고 문헌

[1] Luenbeger, D.G., " Linear and Nonlinear Programming," Addison Wesely, New York, 1984.

[2] Duff, I.S., " Sparse Matrices and Their Uses," Academic Press, London, 1981.

[3] Rodrigue, G., " Parallel Processing for Scientific Computing, " SIAM, New York , 1987.

[4] Schendel, U.,"Introduction to Numerical Methods for Parallel Computers," Ellis Horwood Limited, London,1984.

[5] Evans, D.J., " Sparsity and its Applications," Cambridge University Press," Cambridge, 1985.

[6] Ortega, J.M., " Introduction to Parallel and Vector Solution Systems," Plenum Press, New York, 1988.