通信情報保護學會誌
第2卷, 第2號, 1992. 6

암호 해독 응용을 위한 공유 메모리 모델상에서의 병렬처리

Parallel Gaussian elimination on Shared Memory Model with Application to Cryptoanalysis

정창성* · 최윤희**

요 약

암호응용분야에 있어서의 이산대수 문제나 인수분해 문제는 방대한 양의 데이타를 다루는 문제로 많은 계산시간이 소요되므로 이들 문제들에 대한 고속 병렬처리는 매우 중요하다. 본 논문에서는 역행렬 문제나 이산대수 문제와 인수분해 문제의 중요한 과정인 선형시스템을 푸는데 효율적인 고속 병렬 알고리즘들을 소개한다.

1. Introduction

최근 VLSI 기술의 발달로 여러개의 프로세서를 사용할 수 있는 고성능 chip의 개발이 가능하게 되어 이를 대량으로 사용한 상업용 고도 병렬 컴퓨터가 등장하고, 순차 컴퓨터상에서의 성능 향상으 물리적인 빛의 속도 한계에 도달하게 됨에 따라 고속 계산을 필요로 하는 분야에서의 병렬 처리는 매우 중요하고 필수적인 것으로 인식되고 있다. 특히 암호 처리 분야에 있어서의 이산대수 문제나 인수분해 문제는 많은 계산시간이 걸리므로 병렬처리가 매우 적절하다. 본 논문에서는 이들 문제들을 푸는데 중요한 부분을 차지하는 선형 시스템의 병렬처리에 대해 소개한다.

공유 메모리 모델은 여러개의 프로세서가 공유 메모리를 통하여 데이타를 교환할 수 있는 병렬 모델로 임의의 두개 프로세서 사이의 데이타 교환이 $O(1)$에 수행될 수 있는 이론적으로 가장 빠른 병렬 모델이기 때문에 공유 메모리 모델상에서의 선형 시스템을 최적 시간에 해결하기 위한 많은 연구가 계속되고 있다.

본 논문에서는 L. Csanky의 알고리즘과 $O(n)$의 프로세싱 소자상에서 $O(n^2)$의 overhead와 $O(n^2)$의 시간 복잡도를 가지는 Bampis의 알고리즘, 그리고, $O(\frac{3}{8}n^2)$의 프로세싱 소자에서 $2(n-1)$의 단위 시간에 Gaussian elimination을 수행할 수 있고, $O(n)$의 프로세싱 소자에서는 Bampis의 알고리즘과 동일한 overhead와 시간 복잡도를 가지며 프로세서의 efficiency를 높힐 수 있는 NGA 병렬 알고리즘을 소개한다.

2. L. Csanky의 알고리즘

Csanky는 선형 시스템을 푸는 알고리즘의 시간

* 고려대학교 전자공학과 교수
** 포항공과대학 전자계산학과 대학원

복잡도의 lower bound가 $2\log n+O(1)$로 증명되었으나 기존의 알고리중 가장 빠른 병렬 알고리즘의 시간복잡도가 $2n+O(1)$으로 기존의 Gaussian elimination, Jordan Elimination, Jacobi의 method, 그리고 Strassen의 method 등이 선형 시스템을 푸는데 있어서는 최적 알고리즘이 아니라는 것을 증명하고, 역행렬을 구하는 문제와 선형 시스템의 해를 구하는 문제는 같은 문제로 바꿀수 있으므로 역행렬 문제를 $O(\log^2 n)$의 시간복잡도를 가지고 푸는 다음과 같은 알고리즘을 제시하였다[1].

A를 크기가 n인 행렬이고 det(A), adj(A), tr(A)를 각각 행렬 A의 determinant, adjoint, 그리고 trace라 정의하고, 행렬 A의 charateristic polynomial $f(\lambda)$의 근을 $\lambda_1, \lambda_2, \cdots, \lambda_n$라고 가정하자.

step 1 : s_k를 다음과 같은 방법으로 계산하여 행렬 S를 구한다.

$$s_k=\sum_{i=1}^{n}\lambda^k \quad \text{for } 1\le k\le n$$
$$=tr(A^k) \quad \text{for } 1\le k\le n$$

$$S=\begin{bmatrix} 1 & & & & & & \\ s_1 & 2 & & & & & \\ s_2 & s_1 & 3 & & & & \\ s_3 & s_2 & s_1 & 4 & \ddots & & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & \\ \vdots & \vdots & \vdots & \vdots & & & \\ s_{n-1} & s_{n-2} & s_{n-3} & s_{n-4} & \cdots & s_1 & n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ \vdots \\ s_n \end{bmatrix}$$

$$S_c=-s$$

step 2 : 다음과 같은 식을 이용하여 행렬 S의 역행렬을 구한다. 행렬 S의 역행렬 S^{-1}는

$$S^{-1}=\frac{S^{n-1}+d_1S^{n-2}+\cdots+d_{n-1}I}{d_n}$$

으로 d_n은 행렬 S의 characteristic polynomial $g(\lambda)$을 전개하여 계산한다.

$$g(\lambda)=det(\lambda I-S)$$
$$=(\lambda-1)(\lambda-2)\cdots(\lambda-n)$$
$$=\lambda^n+d_1\lambda^{n-2}+\cdots+d_{n-1}\lambda+d_n$$

step 3 : S의 역행렬 S^{-1}가 구해지면

$$c=-S^{-1}s$$

를 계산한다.

step 4 : step 3에서 계산한 c를 이용하여 A^{-1}를 구한다.

$$A^{-1}=\frac{A^{n-1}+c_1A^{n-2}+\cdots+c_{n-1}I}{c_n}$$

위의 과정에 의해 역행렬을 구할 때 사용되는 프로세싱 소자의 수와 시간복잡도는 다음과 같다.

step 1 : A^k를 계산하기 위해서는 $\log k$의 행렬의 곱셈이 필요하고 행렬의 곱은 n^3의 프로세싱 소자를 가지고 $O(\log n)$의 시간에 구할 수 있으므로 $\log k \times \log n$의 시간이 요구되며 $A^2, A^3, \cdots, A^n$을 계산하기 위해서는 $\frac{1}{2}n^4$의 프로세싱 소자상에서 $\log^2 n$의 시간이 요구된다. 또한 s_k를 구하는데 $O(\log n)$의 시간이 필요하므로 S는 $\log^2 n+O(\log n)$의 시간에 구할 수 있다.

step 2 : $g(\lambda)$는 $\log n+O(1)$의 시간에 구해지고, S^k을 구하는 것은 step 1에서 A^k를 구하는 방법과 동일하므로 $S^2, S^3, \cdots, S^{n-1}$은 $\frac{1}{2}n^4$개의 프로세싱

소자를 가지고 log^2n의 시간에 구해진다. 그리고, 이렇게 구한 d_k와 $S^k(1\leq k\leq n)$을 이용하여 S^{-1}를 구하는데 걸리는 시간은 $\log n+O(1)$으로 전체적으로 $log^2 n+O(\log n)$의 시간에 S^{-1}를 구할 수 있다.

step 3 : c는 $O(n^2)$개의 프로세싱 소자를 이용하여 $\log n+O(1)$ 시간에 구해진다.

step 4 : 역행렬 A^{-1}는 step 2의 S^{-1}를 계산하는 것과 동일한 시간 복잡도를 가지고 구해진다.

결과적으로 역행렬 A^{-1}는 $\frac{1}{2}n^2$개의 프로세싱 소자를 가지고 $O(\log^2 n)$ 시간에 구해진다.

위에서 보는 바와같이 Csanky의 알고리즘은 역행렬을 $O(log^2n)$의 시간에 구할 수 있는 알고리즘이다. 하지만 이 알고리즘을 수행하기 위해서는 $\frac{1}{2}n^4$의 프로세싱 소자가 필요하므로 프로세싱 소자의 수와 계산시간의 곱이 $O(n^4 \log n)$으로 $O(n^2)$의 프로세싱 소자상에서 $2n+O(1)$에 구하는 알고리즘에 비하여 프로세싱 소자의 overhead가 너무 크고, 문제의 크기가 큰 경우에는 사용하기 어려워 실제로 구현하기에는 적합하지 않은 알고리즘이다.

3. Bampis와 J. C. Konig의 알고리즘

Bampis는 공유 메모리 모델병렬 처리 컴퓨터상에서 $O(n)$개의 프로세싱소자를 가지고 $O(n^2)$의 시간에 $O(n^2)$ overhead를 가지고 Gaussian Elimination을 수행하는 scheduling 알고리즘을 제시하였다[7]. 프로그램은 task들로 구성되어 있고 수행순서는 task들간의 precedence relation으로 정의된다. $T_a \ll T_b$는 T_a가 T_b의 수행전에 수행되어야함을 뜻한다. 이 때 T_a는 T_b의 previous task이고 T_b는 T_a의 next이다. precedence graph는 task들의 수행순서를 도시한 graph로 precedence relation에 의해 정의되는 precedence constraint에 의해 만들어진다. Bampis의 알고리즘은 프로세싱 소자의 full efficiency를 더욱 오래 유지하기 위해 Gaussian elimination의 task들을 더욱 작게 세분하는 것을 기본으로 한다. $n\times n$ 행렬에 대하여 Gaussian elimination을 수행하는 병렬 알고리즘은 $2(n-1)$ step으로 구성되어 있고 각 step은 동시에 수행할 수 있는 elementary task들로 이루어져 있다.

```
Parallel Gaussian algorithm GA
for t : =1 to 2(n-1)
  If t is odd(=2k-1), then
      for j : =k+1 to n
        Execute task T_kjk (a_kj : =a_kj/a_kk)
  If t is even(=2k), then
      for j : =k+1 to n
        for i : =k+1 to n
          Execute task T_kji (a_ij : =a_ij-a_ik*
      a_kj)
```

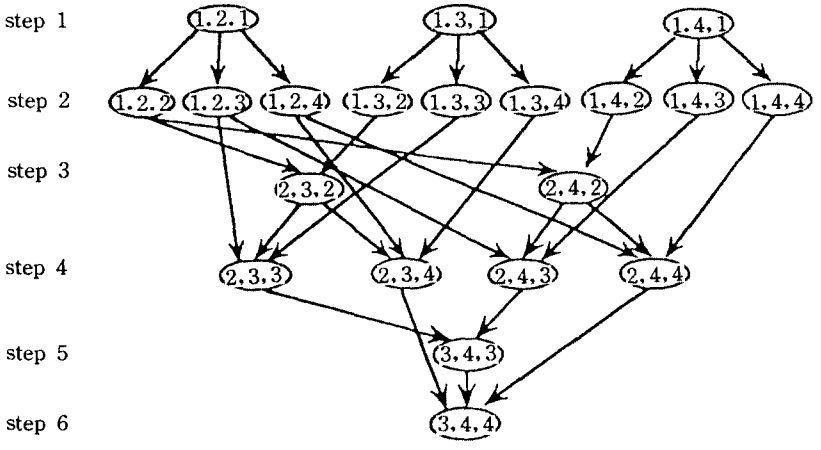


그림 1. Precedence graph G_4

with the precedence constraints for $k<n$:

$T_{k,j,k} \ll T_{k,j,i}$ for all i such that $k<i\leq n$
$T_{k,k+1i} \ll T_{k+1,j,i}$ for all j such that $k+1<j\leq n$
$T_{k,j,i} \ll T_{k+1,j,i}$ for all $i>k+1$ and $j>k+1$

End Parallel Algorithm.

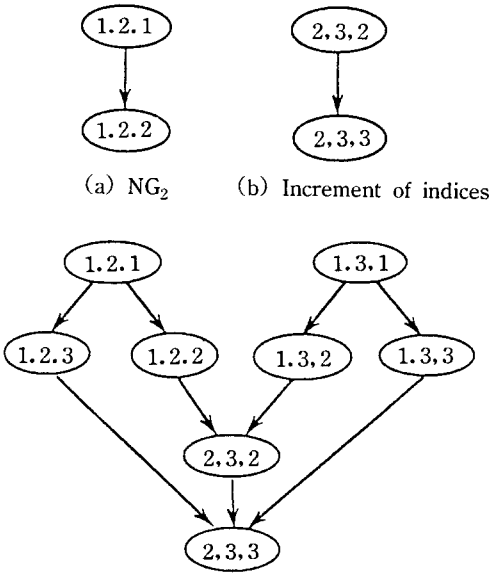
Bampis의 알고리즘에서 각 step $(2k-1)$는 task $T_{k,j,k}$, $k+1\leq j\leq n$의 집합으로, step $2k$는 task $T_{k,j,i}$, $k+1\leq i$의 집합으로 구성되어 있고 step $2k-1$과 step $2k$은 각각 $n-k$개와 $(n-k)^2$개의 task를 가진다. 그러므로 Bampis의 알고리즘에서 한 step이 가지는 최대의 task수는 step 2에서의 $(n-k)^2$개 이고 $O(n^2)$개의 프로세서에서 수행할 경우 $(n-1)^2$개의 프로세싱 소자를 가지고 $2(n-1)$의 단위 시간에 풀 수 있다. 또한 각 step은 greedy한 형태로 수행되어지며, 이 알고리즘을 $p(=O(n))$의 프로세서상에서 수행할 경우에는 한 step이 가지는 task 수를 m이라하면 크기가 $\lceil m/p \rceil$ 혹은 $\lceil m/p \rceil$인 job으로 분할하여 각 프로세서에 할당한다. 이러한 방법으로 task들을 수행하면 level $2k$에서는 job의 크기가 $\lceil m/p \rceil$로 step $(n-\sqrt{p})$까지는 idle한 프로세서가 없고 각 step에서의 idle time은 p 보다 작아지므로 전체 overhead가 $O(n^2)$이 된다.

4. NGA Algorithm

가. Basic Idea

새로운 병렬 알고리즘 NGA[8]는 Bampis의 스케줄링 알고리즘에서 한 step이 필요로 하는 최대의 task수를 줄임으로써 전체 step수를 늘리지 않으면서 프로세싱 소자의 수를 줄여 최적의 시간 복잡도를 이루기 위한 알고리즘이다. precedence graph G_4에서 보는 바와 같이 Bampis의 precedence graph상의 task들 중에는 다음의 step으로 수행 순서를 늦추어도 전체 step의 수에 영향을 주지 않는 것들이 있다. 예를들면 만약 step t의 한 task의 next task가 step $\hat{t}$에 있다면 이 task는 step t와 step $(\hat{t}-1)$ 사이의 어떤 step에서도 수행가능하다. 새로운 병렬 알고리즘의 기본 아이디어는 이런 task들의 수행을 최대한으로 늦추어 한 step이 가지는 최대 task의 수를 줄이고 각 step간에 task의 수를 좀 더 고르게 분배하여 load balancing을 이루는 것이다. 새로운 병렬 알고리즘의 기본 아이디어를 $(m-1)\times(m-1)$ 행렬에 대한 새로운 precedence graph NG_{m-1}을 이용하여 $m\times m$ 행렬에 새로운 precedence graph NG_m을 만드는 과정을 보여주므로써 간단히 설명한다.

우선 2×2 행렬에 대한 새로운 precedence graph NG_2를 그림 2. a와 같이 구성한 후 NG_3를 바탕으로 하여 2×3 행렬에 대한 새로운 precedence graph NG_3를 다음과 같은 방법으로 구성한다 : NG_3의 $k=2$인



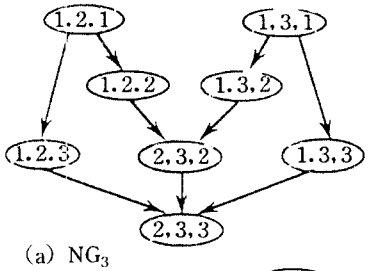
(a) NG_2 (b) Increment of indices

(c) Addition of tasks

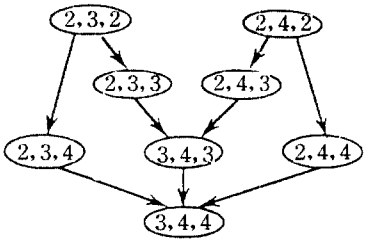
(d) Transfer of tasks

그림 2. Constructing NG_3 from NG_2

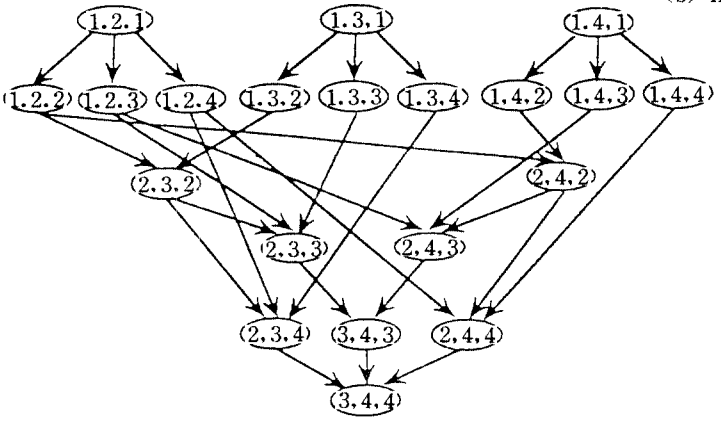
task T_{kij}를 구성하기 위하여 NG_2의 task들의 1씩 증가시킨후(그림 2. b), step 1과 step 2에 $k=1$인 task T_{kij}를 추가한 다음(그림 2. c) step 2의 각 task들을 각각의 next task들이 수행되기 바로 전 step으로 옮긴다. 예를들면 T_{123}과 T_{133}을 그림 2. d에서와 같이 step 3으로 옮긴다. 마찬가지 방법으로 4×4 행렬에 대한 새로운 precedence graph NG_4를 그림 3와 같이 구성할 수 있다. 일반적으로 precede-



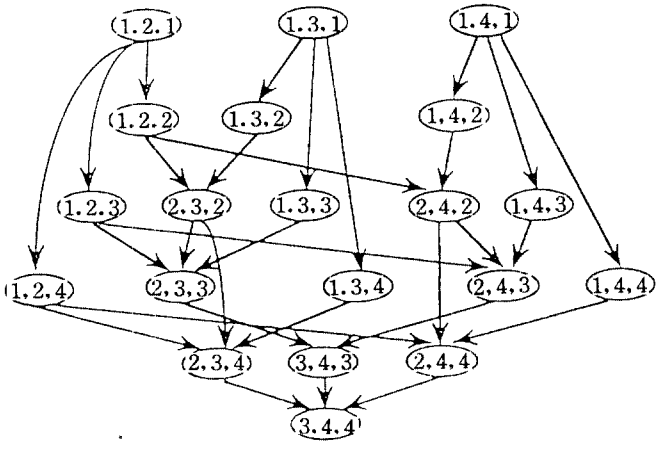
(a) NG_3



(b) Increment of indices



(c) Addition of tasks



(d) Transfer of tasks

그림 3. Constructing NG_4 from NG_3

nce graph NG_m은 NG_{m-1}의 task T_{kij}의 인덱스 k, j, i를 각각 1씩 증가시킨 다음 $k=1$인 task T_{kij}들을 추가시키고 이들 중 step 2의 task들을 각 task의 next task가 있는 바로전의 step으로 옮김으로써 만들어진다. 그림 4는 NG_4로부터 NG_5를 만든 것이다.

새로운 precedence graph는 각각의 task들이 수행되어지는 step과 인덱스간에 다음과 같은 관계가 성립한다.

Lemma 1 : 새로운 병렬 알고리즘의 task T_{pqr}은 step $p+r-1$에서 수행된다.

proof. $(i)2\times2$ 행렬에 대하여 성립한다.

$(ii)(n-1)\times(n-1)$ 행렬에 대하여 성립한다고 가정하면 NG_n의 step 3에서 step $2(n-1)$까지의 task들은 NG_{n-1}의 task들의 인덱스를 1씩 증가시킴으로써 얻어진 것이므로 위의 lemma 1을 만족한다. 또한 step 1의 task들은 옮겨지지 않았으므로 step 1의 task T_{1jl}도 또한 lemma 1을 만족한다. task $T_{1ji}(2\le i, j\le n)$는 precedence constraints에 의하여 각각 T_{2*i}를 next task로 가지고, T_{2*i}는 step $i+1$에 수행되어지므로 각각의 task T_{1ji}는 step i에 옮겨진다. 그러므로 NG_n의 모든 task들도 lemma 1을 만족한다.

Lemma 2 : 새로운 병렬 알고리즘의 step t는 $1\le t\le n$인 t에 대해 $1\le p\le\lceil\frac{t}{2}\rceil$이고 $n\le t\le 2(n-1)$인 t에 대해 $t-n+1\le p\le\lceil\frac{t}{2}\rceil$를 만족하는 task T_{pqr}로 구성되었다.

proof. $(i)2\times2$ 행렬에 대하여 성립한다. □

$(ii)(n-1)\times(n-1)$ 행렬에 대하여 성립한다고 가정하면 NG_n의 step 3에서 step $2(n-1)$까지의 task들은 NG_{n-1}의 task들의 인덱스를 1씩 증가시키고 수행되는 step을 2씩 증가시킴으로써 얻어진 것이므로 step 3에서 step $n+1$ 사이의 step t의 task T_{pqr}는 $2\le p\le\lceil\frac{t}{2}\rceil$를 만족하고, step$(n+2)$에서 step $2(n-1)$ 사이의 step t의 task T_{pqr}는 $t-n+1\le p\le\lceil\frac{t}{2}\rceil$를 만족하기 때문에 step 2에 추가된 task들을 옮기기전까지는 lemma 2를 만족한다. 그리고, step 1과 step 2에 task T_{1pr}를 추가시킨 다음 step 2의 task $T_{1pr}(2\le q)$을 lemma 1에 의해 step $r(r\le n)$에 옮기면,

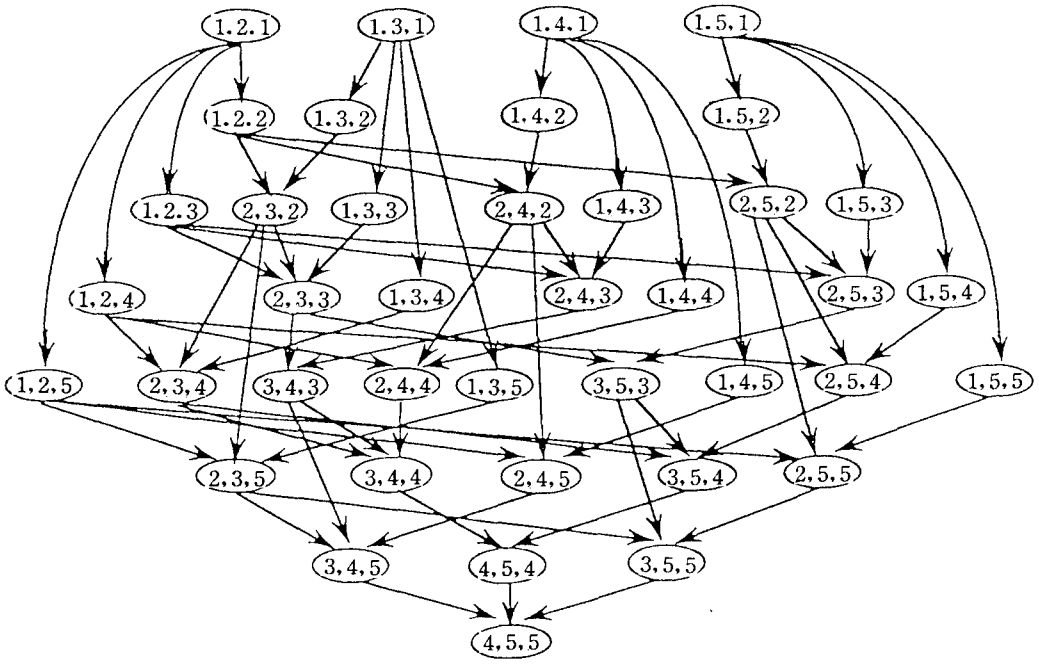


그림 4. New precedence graph NG_5 for 5×5 matrix

step 1에서 n 사이의 step t의 각 task T_{pqr}은 $1\le p\le \lceil \frac{t}{2} \rceil$를 만족하고, step $(n-2)$에서 step $2(n-1)$ 사이의 *step* t의 task T_{pqr}는 $t-n+1\le p\le \lceil \frac{t}{2} \rceil$를 만족한다. □

나. 병렬 알고리즘

이 절에서는 Gaussian elimination을 $O(\frac{3}{8}n^2)$의 프로세싱 소자를 가지고 최적 시간에 수행할 수 있는 알고리즘을 소개한다. 새로운 병렬 알고리즘의 *precedence graph*는 lemma 1과 lemma 2에 의해 구성되고, 각각의 step에서 수행되는 task들은 다음의 병렬 알고리즘에 의해 규칙적으로 생성된다.

Parallel New Gaussian Algorithm(NGA)
for t : =1 to n
 for k : =1 to $\lceil t/2 \rceil$
  for j : =$k+1$ to n
   Execute task $T_{k,j,t+1-k}$
    (a_{kj} : =$a_{kj}/a_{k,k}$) if $k=t+1-k$
    ($a_{t+1-k,j}$: =$a_{t+1-k,j}-a_{t+1-k,k}*a_{k,j}$
    otherwise
for t : =$n+1$ to $2(n-1)$
  for k : =$t-n+1$ to $\lceil t/2 \rceil$
   for j : =$k+1$ to n
   Execute task $T_{k,j,t+1-k}$

with the precedence constraints for $k<n$:

$T_{k,j,k} \ll T_{k,j,i}$ for all i such that $k<i\le n$
$T_{k,k+1,i} \ll T_{k+1,j,i}$ for all j such that $k+1<j\le n$
$T_{k,j,i} \ll T_{k+1,j,i}$ for all $i>k+1$ and $j>k+1$

End Parallel Algorithm.

step 1과 step n 사이의 step t의 모든 task T_{kjr}은 $1\le k\le \lceil \frac{t}{2} \rceil$ 와 $k+r-1=t$를 만족하고, step $n+1$과 step $2(n-1)$ 사이의 step t의 모든 task T_{kjr}은 $t-n+1\le k\le \lceil \frac{t}{2} \rceil$ 와 $k+r-1=t$를 만족하므로 각 step이 가지는 task의 수는 step 1에서 step n 사이에서는 증가하며, step 1에서 step n 사이에서는 감소한다.

Lemma 3 : 새로운 precedence graph의 한 step이 가지는 task의 최대수는 n이 홀수인 경우에는 $(3n^2-1)/8$이고, 짝수인 경우에는 $(3n^2-2n)/8$이다.

proof. $1\le t_1\le n$인 step t_1에 수행되는 task의 수 N_{t1}는

$$N_{t1}=\sum_{k=1}^{\lceil t_1/2 \rceil}(n-k)$$

$$=\begin{cases}(t_1+1)(4n-t_1-3)/8 & \text{if } t_1 \text{ is odd}\\ t_1(4n-t_1-2)/8 & \text{otherwise}\end{cases}$$

이고, $n+1\le t_2\le 2(n-1)$인 *step* t_2에 수행되는 task의 수 N_{t2}는

$$N_{t2}=\sum_{k=t_2-n+1}^{\lceil t_2/2 \rceil}(n-k)$$

$$=\begin{cases}\{3(2n-t_2)^2-3\}/8 & \text{if } t_2 \text{ is odd}\\ \{3(2n-t_2)^2-2(2n-t_2)\}/8 & \text{otherwise}\end{cases}$$

이므로 step n이 가장 많은 수의 task를 가지고 있고 task의 수는 n이 홀수인 경우에는 $(3n^2-1)/8$이고, 짝수인 경우에는 $(3n^2-2n)/8$이다. □

Lemma 1과 lemma 2로 부터 NGA은 correct하며, lemma 3에 의하여 다음과 같은 theorem이 성립한다.

Theorem 1 : Gaussian elimination은 공유 메모리 모델병렬 처리 컴퓨터상에서 $O(\frac{3}{8}n^2)$개의 프로세싱 소자를 가지고 $2(n-1)$의 단위시간에 풀 수 있다.

또한 새로운 병렬 알고리즘을 $O(n)$개의 프로세싱 소자를 가진 병렬처리 컴퓨터에 적용할 경우에는 Bampis의 알고리즘과 동일한 overhead를 가지면서 각 step에 task들이 훨씬 고루 분포되어 있기 때문에 Bampis의 알고리즘에서 프로세싱 소자가 idle해지가 시작하는 step 보다 훨씬 뒤의 step에서부터 idle한 프로세싱 소자가 생기기 시작한다.

5. 결 론

본 논문에서는 공유 메모리 모델상에서 선형시스템을 효율적으로 풀 수 있는 병렬 알고리즘들에 대하여 설명하였다. 그중 가장 빠른 알고리즘인 L. Csanky의 알고리즘은 $O(log^2n)$의 시간 복잡도를 가지지만 $\frac{1}{2}n^4$개의 프로세싱 소자를 필요로 하므로 실제로 사용하기는 어려운 알고리즘이고, precedence graph를 기

본으로하는 Bampis의 알고리즘과 NGA는 $O(n)$개와 $O(n^2)$개의 프로세싱 소자상에서 구현할 수 있는 알고리즘으로 NGA가 Bampis의 알고리즘에 비해 훨씬 적은 수의 프로세싱 소자를 사용하는 효율적인 알고리즘이다.

앞으로는 Gaussian elimination을 분산기억장치 모델상에서 처리하는 알고리즘에 관한 연구와 더불어 선형시스템의 고속처리외에도 암호처리 분야의 문제들 중 많은 계산 시간이 요구되는 이산대수 문제, 인수분해 문제의 전체적인 알고리즘을 병렬화하는 것에 관한 연구가 필요하다.

참 고 문 헌

[1] L. Csanky, Fast Parallel Matrix Inversion Algorithms, em SIAM J. Compt(1976) 618-623.

[2] M Cosnard, M. Marrakchi, Y. Robert and D. Trystram, Parallel Gaussian Elimination on an MIMD compute, *Parallel Computing 6* (1988) 275-296.

[3] Nilolas M. Missirlis, Scheduling Parallel Iterative Methods on Multiprocessor system, *Parallel Computing 5* (1987) 295-302.

[4] J. M. Ortega and C. H. Romine, The ijk Forms of Factorizatin Methods II. Parallel System, *Parallel Computing 7* (1988) 149-162.

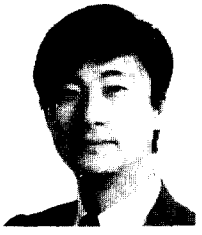
[5] M. Marrakchi and Y. Robert, Optimal Algorithm for Gaussian Elimination on an MIMD Computer, *Parallel Computing 12* (1989) 183-194.

[6] Michael R. Leuze, Independent Set Orderings for Parallel Matrix Factorization by Gaussian Elimination, *Parallel Computing 10* (1989) 177-191.

[7] E. Bampis and J. C. Konig, Impact of Communications on the Complexity of Parallel Gaussian Elimination, *Parallel Computing 17* (1991) 55-61.

[8] C. S. Jeong and Y. H. Choi, Parallel gaussian Elimination on Shared Memory Model, *ISMM Interantional Conference on Computer Applications in Design, Simmulation and analysis, Orlando, U. S. A.* (1992)

□ 著者紹介



정 창 성

1981년 2월 서울대학교 공과대학 전기공학과 졸업
1985년 8월 Nortwestern University 전자계산학 석사
1987년 8월 Nortwestern University 전자계산학 박사
1987년 9월～1992년 3월 포항공과대학 교수
1992년 4월～현재 고려대학교 전자공학과 교수
관심분야 : 병렬처리, 암호학, 계산 이론

□ 著者紹介

최 윤 희

1991년 2월 포항공과대학 전자계산학과 졸업
1991년 3월～현재 포항공과대학 전자계산학과 대학원