産業技術研究(江原大學校 産業技術研究所 論文集) 第14輯 (1994)
Research Report, The Institute of Industrial Technology, Kangwon Nat'l Univ., Korea, Vol. 14, 1994

MP-2에서의 타원형 편미분 방정식 병렬계산

김 형 중* · 이 용 호**

Parallel Computation of Elliptic Partial Differential Equation on MP-2

Hyoung-Joong Kim * · Yong-Ho Lee **

ABSTRACT

We can get a tridiagonal block Toeplitz linear system by the finite difference approximation of 2-D Poisson equation. To exploit the nice property of this linear equation, we transform the equation into a Lyapunov equation and apply DST (discrete sine transform) to get diagonal matrix based Lyapunov equation. DST can be performed using FFT, which enables high-speed computaion. All the computations are performed on an SIMD parallel computer, the MasPar MP-2 with 4,096 processing elements. In this paper, parallel algorithm, mapping method of the algorithm onto the MP-2, and timing results are presented.

1. 서 론

유한 차분법을 사용하여 2차원 Poisson 방정식을 근사화시키면

$$Au = f \qquad (1)$$

라는 선형방정식을 얻게되며 여기서 pentadiagonal(혹은 tridiagonal block-Toeplitz) 행렬 A는 다음

$$A = \begin{bmatrix} T & -I & 0 & \cdots & 0 \\ -I & T & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & T & -I \\ 0 & \cdots & 0 & -I & T \end{bmatrix} \qquad T = \begin{bmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{bmatrix}$$

* 공과대학 제어계측공학과 조교수
** 공과대학 제어계측공학과 석사과정

과 같이 표현된다[5]. 행렬 T는 대각선을 따라 상수인 Toeplitz 구조를 갖는다.

선형방정식 (1)은 다음과 같은 Lyapunov 행렬방정식

$$LU + UL = F \tag{2}$$

로 변형될 수 있으며 행렬 L은

$$L = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}$$

로 표현된다.

행렬L의 독특한 형태로 인해 이산정현변환 DST (discrete sine transform)를 통해 문제 해결을 쉽게 이룩할 수 있다. DST는 FFT의 특수한 형태이므로 DST대신 FFT를 이용하여 고속 계산이 가능하도록 한다. FFT는 데이타 병렬성이 높아 SIMD 컴퓨터에서 계산하는데 적합하다. MasPar에서 제작된 MP-2는 SIMD 컴퓨터로 최대 16,384개의 프로세서를 보유할 수 있다. 본 논문에서는 4,096개의 프로세서를 이용하였다. MasPar는 MPF (MasPar Fortran) 병렬 프로그래밍 언어와 MPML (MasPar Mathematics Library)이라는 선형대수 계산용 라이브러리를 제공한다[1]. 따라서 본 논문에서는 MPML에 있는 FFT프로그램을 이용해 MPF로 프로그래밍한 다음 계산시간을 조사한다.

MPF를 사용할 경우 두가지 FFT루틴을 쓸 수 있고, FFT도 1차원과 2차원의 두 종류가 있으며, 거기에 실수 FFT와 복소수 FFT가 있어 어느 조합이 가장 효율적인지, 또 문제점은 무엇인지 알아보고자 한다.

식 (2)는 아래 방정식으로 바꿀 수 있다.

$$DX + XD = Y \tag{3}$$

여기서,

$$D = QLQ, \tag{4}$$

$$Q = q_{ij} = \sqrt{\frac{2}{M+1}} \cdot \sin\left(\frac{ij\pi}{M+1}\right), \tag{5}$$

$$D = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_M \end{bmatrix}, \tag{6}$$

$$\lambda_i = 2-2\cos\left(\frac{i\pi}{M+1}\right) \quad i, j = 1, 2, \cdots, M, \tag{7}$$

$$X = QUQ, \tag{8}$$

$$Y = QFQ, \tag{9}$$

과 같이 주어지며, Q는 DST행렬이고 $Q = Q^T = Q^{-T}$이다.

그리하여 식 (3)은 아래와 같이 단순한 대수 방정식으로 줄일 수 있다.

$$(\lambda_i + \lambda_j) X_{ij} = Y_{ij} \tag{10}$$

이제 우리는 식 (10)으로부터 방정식 (2)의 행렬 U를 얻을 수 있다.

$$U = QXQ \tag{11}$$

모든 진행 절차는 단순히 2차원 DST 즉, $Y = QFQ$와 $U = QXQ$를 수행한 것으로 충분하다. 우리는 이것을 프로세서상에 어떻게 배정할 것인지와 실행 과정에서 나타나는 문제점, 그리고 성능의 차이가 어떻게 나타나는지를 보일 것이다. 전체의 병렬처리 과정은 그림 1과 같다.

1단계 : $Au = f$에서 초기조건을 가지고 병렬로 f를 구함.
2단계 : $Au = f$를 $LU + UL = F$로 변환
3단계 : 병렬로 $Y = QFQ$를 구함.
4단계 : 병렬로 $X_{ij} = \frac{Y_{ij}}{\lambda_{ij}}$ 를 구함.
5단계 : 병렬로 $U = QXQ$를 구함.

그림 1. 병렬처리 과정

식 (9)에서 Q는 실수 원소로 구성되며 F의 모든 원소도 실수이므로 Y도 실수 행렬이 된다. 그런데 식 (9)를 FFT로 계산하려고 하면 크게 두가지 문제점이 생긴다. 하나는 DST에서는 Q가 실수행렬인데 반해 Q에 해당하는 FFT에서의 트위들 팩터 행렬이 복소수라는 점이다. 또 한가지는 실수 입력 F에 대한 FFT의 결과가 복소수로 나올 뿐만 아니라 그 결과도 DST한 결과와 다르다는 것이다. 따라서 FFT결과가 DST결과와 일치하게 만드는 방법이 요구된다.

다음 2절에서는 FFT결과와 DST결과를 일치시키는 방법을 1차원과 2차원의 두가지 방법에 대해 모두 언급할 것이다. 여기서는 주로 벡터 입력에 대한 논의가 이루어진다. 3절에서는 MasPar의 SIMD 컴퓨터인 MP-2를 사용하여 FFT를 구현한 방법과 MasPar에서 제공하는 FFT루틴들의 차이점을 비교 설명할 것이다. 여기서는 2절의 벡터 입력과는 달리 행렬 입력의 경우가 다루어진다. 4절에서는 MP-2에서 시뮬레이션한 결과로, FFT를 수행함에 있어서의 문제점과 계산오차 및 방정식의 해를 구하는데 필요한 시간에 대해 언급할 것이다. 마지막으로 5절에서는 결론으로 MP-2에서의 알고리듬 수행결과 나타난 제반 현상을 요약해서 기술한다.

2. 벡터 데이타 변환 방법

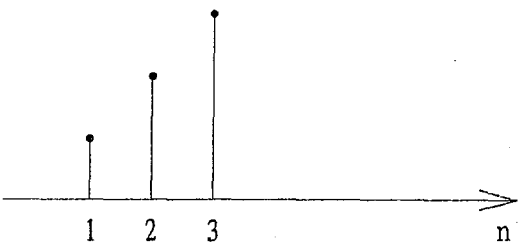
DST는 DCT (discrete cosine transform)와 마찬가지로 FFT 알고리즘의 특별한 경우이므로, 더욱 빠른 계산을 위해 DST 대신 FFT 알고리즘을 사용할 수 있다[2,3]. 앞서 지적한 것처럼 동일한 벡터 입력 $x(n)$에 대한 DST 결과와 FFT 결과는 다르다. 따라서 $x(n)$을 변형시켜서 두 결과를 일치시키는 방법이 요구된다. 다음은 $x(n)$을 DST한 결과와 $x(n)$을 변형시켜 FFT한 결과를 같게 하는 방법에 대해 논의한다. FFT는 1차원과 2차원에 따라 그 방법이 다르므로 두 방법에 대해 모두 언급하기로 한다.

가) 1차원 FFT의 수행

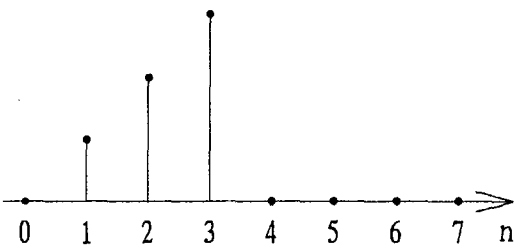
1차원 FFT를 하기 위해, DST 입력을 $x(n)$이라 하고 FFT 입력을 $y(n)$이라 하여 다음과 같이 두 입력 사이에 관계식을 갖도록 한다.

$$\begin{aligned} y(n) &= 0 \quad n = 0 \text{ 또는 } n = N,\ N+1,\ \cdots,\ 2N-1 \\ y(n) &= x(n) \quad n = 1,\ 2,\ \cdots,\ N-1 \end{aligned} \tag{12}$$

여기서 $x(n)$은 주어진 벡터로 DST 입력이 된다. 편의상 벡터의 갯수는 M으로 놓는다. $x(n)$은 M개이나 $y(n)$은 $2M+2$개의 데이타로 늘어나게 된다. 단 $M = N-1$이며 N은 FFT 입력 갯수를 나타낸다. 만약 $M = 3\ (N = 4)$일때 $x(n)$과 $y(n)$의 관계는 다음 그림 2와 같다.



(a) 3-point 입력 데이타 : $x(n)$



(b) 8-point 입력 데이타 : $y(n)$

그림 2. 1-D FFT를 위한 데이타 변환

식(12)의 $y(n)$을 가지고 1차원 FFT를 수행한 후 M개($1, 2, \cdots, N-1$)값의 허수 부분만 취하여 $-\sqrt{2/(M+1)}$ 를 곱한 결과는 $x(n)$을 가지고 1차원 DST한 결과와 같다. 즉,

$$DST = -\sqrt{\frac{2}{M+1}} \cdot imag(FFT) \qquad (13)$$

가 된다. 식(13)에서 비록 $y(n)$이 실수일지라도 FFT한 결과는 복소수가 되며 이 가운데 허수 부분만 취해야 한다. 그러면 그림 1의 3단계, 5단계에서의 DST 계산을 FFT로 대치할 수 있다. 그 결과는 그림 3과 같다.

1단계 : $Au = f$에서 초기조건을 가지고 병렬로 f를 구함.
2단계 : $Au = f$를 $LU + UL = F$로 변환.
3단계 : 병렬로 $Y = QFQ$를 구함.
  1) 병렬로 1차원 FFT 수행.
  2) 병렬로 transpose.
  3) 병렬로 1차원 FFT 수행.
  4) 병렬로 $\frac{2}{M+1}$ 곱하기.
4단계 : 병렬로 $X_{ij} = \frac{Y_{ij}}{\lambda_{ij}}$ 를 구함.
5단계 : 병렬로 $U = QXQ$를 구함.
  1) 병렬로 1차원 FFT 수행.
  2) 병렬로 transpose.
  3) 병렬로 1차원 FFT 수행.
  4) 병렬로 $\frac{2}{M+1}$ 곱하기.

그림 3. 1차원 FFT를 사용한 병렬처리 과정

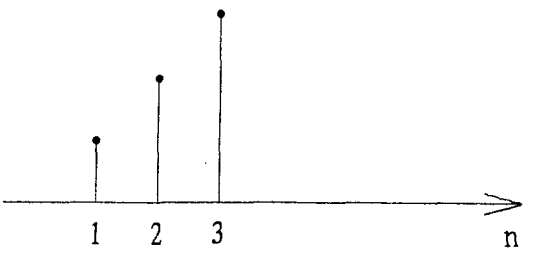
그림 3의 3단계, 5단계에서 1차원 FFT 수행 전의 입력은 DST의 입력 벡터(F 또는 X의 각 행이나 열)이므로 식(12)에서와 같이 FFT를 수행할 수 있도록 바꾸어야 하고, FFT 수행 후에는 반대로 DST의 입력 벡터 형태로 변환한 다음 허수 부분만을 취한다.

나) 2차원 FFT의 수행

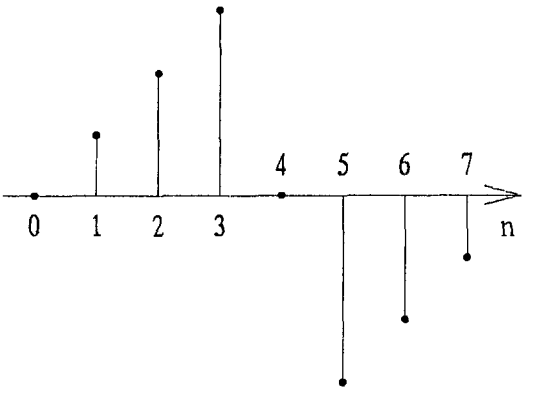
2차원 FFT를 하기 위해, DST 입력을 $x(n)$이라 하고 FFT 입력을 $y(n)$이라 하여 다음과 같이 두 입력 사이에 관계식을 갖도록 한다.

$$\begin{aligned} y(n) &= 0 \quad n = 0 \text{ 또는 } N \\ y(n) &= x(n) \quad n = 1, 2, \cdots, N-1 \\ y(n) &= -x(2N-n) \quad n = N+1, N+2, \cdots, 2N-1 \end{aligned} \qquad (14)$$

만약 $M = 3 (N = 4)$ 일때 $x(n)$과 $y(n)$의 관계는 다음 그림 4와 같다.



(a) 3-point 입력 데이타 : $x(n)$



(b) 8-point 입력 데이타 : $y(n)$

그림 4. 2-D FFT를 위한 데이타 변환

식(14)에서와 같이 $y(n)$을 만든 다음,

$y(n)$에 대해 FFT 하고$\sqrt{\frac{2}{M+1}}$을 곱하면,

$$\sqrt{\frac{2}{M+1}}\ Y(m) = \sqrt{\frac{2}{M+1}} \sum_{n=0}^{2N-1} y(n) W_{2N}^{nm}$$
$$= \sqrt{\frac{2}{M+1}} \left(\sum_{n=1}^{N-1} x(n) W_{2N}^{nm} - \sum_{n=N+1}^{2N-1} x(2N-n) W_{2N}^{nm} \right)$$
$$= \sqrt{\frac{2}{M+1}} \left(\sum_{n=1}^{N-1} x(n) W_{2N}^{nm} - \sum_{n=1}^{N-1} x(n) W_{2N}^{(2N-n)m} \right)$$
$$= \sqrt{\frac{2}{M+1}} \sum_{n=1}^{N-1} x(n)\left[W_{2N}^{nm} - W_{2N}^{-nm} \right] \qquad (15)$$

과 같은 결과를 얻었다. 여기서

$W_{2N} = e^{-\frac{j2\pi}{2N}}$ 이다.

식(15)의 양변에$-\frac{1}{2j}$ 을 곱하면,

$$\sqrt{\frac{2}{M+1}} \left(-\frac{1}{2j}\right) Y(m)$$
$$= \sqrt{\frac{2}{M+1}} \sum_{n=0}^{N-1} x(n) \sin\left(\frac{nm\pi}{N}\right) \qquad (16)$$

와 같은 관계가 성립한다. 식(16)의 우변은 $x(n)$에 1차원 DST한 결과이고, 좌변의 $Y(m)$은 식(15)에서 본 바와 같이 $y(n)$에 1차원 FFT를 한 결과이다. DST한 결과와 FFT한 결과가 서로 일치하지 않기 때문에 (16)에서와 같이 $Y(m)$에$\sqrt{\frac{2}{M+1}} \left(-\frac{1}{2j}\right)$ 을 곱해야 한다. 즉,

$$DST = \sqrt{\frac{2}{M+1}} \left(-\frac{1}{2j}\right) FFT \qquad (17)$$

가 된다. 이때 FFT(입력은 실수벡터)한 결과의 실수 부분은 모두 0이다. 또한 허수 입력에 대한 FFT 결과의 허수 부분은 모두 0이다. 앞에서 설명한 1차원 FFT에서는 허수 부분을 취하는 과정이 포함되어 있다. 따라서 1차원 FFT를 두번 수행함으로써 2차원 FFT를 수행한 것과 같은 효과를 얻고 싶다면 앞서 기술한 1차원 FFT 방법을 쓰면 된다. 그러나 2차원 FFT 한번으로 원하는 결과를 얻고 싶다면 앞의 방법을 사용할 수 없다. 왜냐하면 기존의 FFT 루틴에는 도중에 허수 부분만 취하게 하는 과정이 포함되어 있지 않기 때문이다. 따라서 실수 입력에 대해 1차원 FFT하면 실수값은 모두 0이므로 허수값만 출력되고 다시 그 허수값에 1차원 FFT를 수행시켜 허수값은 모두 0이므로 실수값만 출력되도록 함으로써 2차원 FFT가 실행되게 하는 특별한 $y(n)$을 만드는 법은 (14)와 같다. 그래서 2차원 FFT 루틴을 그대로 사용하여도 2차원 DST를 한 결과를 얻을 수 있다. 2차원 FFT에 대한 식은 다음과 같다.

$$2D\_DST = -\frac{1}{2(M+1)}\ 2D\_FFT \qquad (18)$$

그러면 그림 1의 3단계, 5단계에서의 DST 계산은 다음 그림 5와 같다.

1단계 : $Au = f$에서 초기조건을 가지고 병렬로 f를 구함.
2단계 : $Au = f$를 $LU + UL = F$로 변환.
3단계 : 병렬로 $Y = QFQ$를 구함.
  1) 병렬로 2차원 FFT 수행.
  2) 병렬로 $-\frac{1}{2(M+1)}$ 곱하기.
4단계 : 병렬로$X_{ij} = \frac{Y_{ij}}{\lambda_{ij}}$ 를 구함.
5단계 : 병렬로 $U = QXQ$를 구함.
  1) 병렬로 2차원 FFT 수행.
  2) 병렬로 $-\frac{1}{2(M+1)}$ 곱하기.

그림 5. 2차원 FFT를 사용한 병렬처리 과정

그림 5의 3단계, 5단계에서 2차원 FFT 수행 전의 입력은 DST의 입력 벡터(F 또는 X)이므로 식(14)에서와 같이 FFT를 수행할 수 있도록 바꾸어야 하고, FFT 수행 후에는 반대로 DST의 입력 벡터 형태로 변환한 다음 실수 부분만을 취한다. 이때 허수 부분은 모두 0이다.

3. MP-2에서의 구현 방법

MasPar에서 제공하는 FFT 프로그램은 다음의 5가지가 있다[1].

(가) 1D complex minimum latency FFT
(나) 1D complex highly parallel FFT
(다) 1D real minimum latency FFT
(라) 1D real highly parallel FFT
(마) 2D complex minimum latency FFT

2절에서는 입력이 벡터인 경우가 언급되었다. 3절에서는 입력이 행렬로 주어진 경우를 다루게 된다. 본 논문에서는 식(9)와 (11)을 계산하는 것이 목표이므로 행렬을 입력으로 하며 이 계산 과정에서의 병렬처리가 주된 관심사이다.

$M=3$인 행렬 입력의 경우 즉, 2차원 DST를 하기 위해 3×3 행렬 입력이

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

와 같이 주어졌다고 가정한다.

1) 1차원 FFT

2차원 DST를 1차원 FFT로 구현하기 위해 식(12)와 같은 방법으로 FFT를 위한 입력을 만들면 그림 6의 (a)와 같고, 그 과정은 그림 3의 3단계, 5단계와 같다. 그러나 이 경우 출력으로 C는 2부터 4까지, L은 2부터 4까지의 9개의 값만을 필요로 하므로 그림 6의 (b)에서처럼 크기를 3×8행렬로 줄일 수 있다. 이 입력으로 1차원 FFT를 수행한다. 다음으로 이 9개 값의 허수 부분만을 취하여 transpose한 후 그림 6의 (b)에서와 같은 입력을 만들어 한번 더 1차원 FFT를 수행한다. 그런 다음 다시 0이 아니었던 앞서와 같은 부분의 9개 값의 허수 부분만 취하여 $\frac{2}{M+1}$ 를 곱하면 2차원 DST를 한 결과를 얻는다.

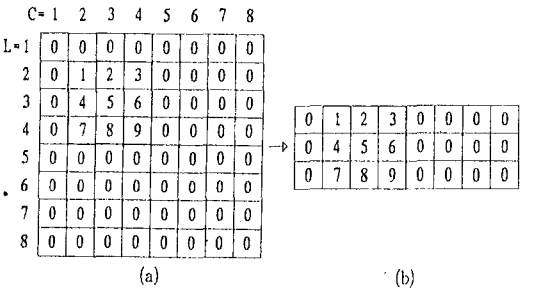


그림 6. 실제 1-D FFT 입력 데이타 형태 (M=3)

2) 2차원 FFT

2차원 DST를 2차원 FFT로 구현하기 위해서 식(14)와 같이 처리하면 그림 7의 (a)와 같고, 그 과정은 그림 5의 3단계, 5단계와 같다. 이 경우 출력도 1차원의 경우처럼 9개의 값만을 필요로 하므로 그림 7의 (b)처럼 입력을 바꿀 수 있다. 이 과정은 그림 7의 (a)처럼 뒷부분을 나열하지 않아도 되므로 입력을 나열하는데 드는 시간을 줄일 수 있다. 이 입력으로 2차원 FFT를 한 후 9개 값의 실수 부분만 취하여 $-\frac{2}{M+1}$ 를 곱하면 2차원 DST를 한 결과를 얻는다.

MasPar에서 제공하는 2차원 FFT 라이브러리에서는 complex FFT 루틴 밖에 없어 2차원 complex FFT를 수행하였다.

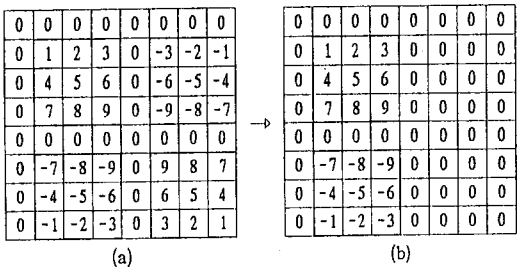


그림 7. 2-D FFT 입력 데이타 형태 ($M=3$)

각 루틴의 특징을 요약하면 다음과 같다.

먼저 minimum latency FFT와 highly parallel FFT의 차이는 입력 데이타를 프로세서에 배당하는 방법에 있다. 만약 그림 6의 (a)와 같은 입력으로 1차원 FFT를 할 경우, minimum latency FFT는 C개의 데이타를 C개의 프로세서에 하나씩 배당하여 L번 FFT를 수행한다. highly parallel FFT에서는 $C\times L$개의 데이타를 L개의 프로세서에 배당하여 한번에 FFT를 수행한다. 이때는 한 프로세서에 C개씩 데이타를 배당한다. 따라서 L개의 프로세서는 독립적으로 FFT를 수행한다. 즉 L개의 프로세서로 분산처리 한다고 보면 된다.

다음으로 complex FFT와 real FFT의 차이를 비교하기 위해 $M=3$(DST 입력 $x(n)=$[1 2 3])인 경우를 예로 살펴보자. FFT 입력은 $y(n)=$[0 1 2 3 0 0 0 0]와 같이 만들 수 있다.

1) Complex FFT

FFT에 사용되는 입력 $y(n)$은 순수한 실수이다. 그러나 complex FFT library를 사용하기 위해 입력으로 허수부분이 0인 복소수를 만들어 complex FFT를 수행하였다.

다시 말해 $y(n)=$[0+j0, 1+j0, 2+j0, 3+j0, 0+j0, 0+j0, 0+j0, 0+j0]으로 표시해야 하며 그 입력 패턴은 아래

입력 : [(0, 0) (1, 0) (2, 0) (3, 0) (0, 0) (0, 0) (0, 0) (0, 0)]

와 같이 된다. 이때 complex FFT 결과 패턴은

출력 : [(6, 0) $(-\sqrt{2}, \underline{(-2-2\sqrt{2})\,j})$ $\underline{(-2, 2j)}$ $(\sqrt{2}, \underline{(2-2\sqrt{2})\,j})$ (-2, 0) $(\sqrt{2}, (-2+2\sqrt{2})\,j)$ (-2, -2j) $(-\sqrt{2}, (2+2\sqrt{2})\,j)$]

와 같다. 여기서 우리가 원하는 DST 출력은 밑줄친 부분으로 허수값만 취하면 된다. 다시 말해 DST 결과는$[-2-2\sqrt{2}, 2, 2-2\sqrt{2}]$가 된다.

2) Real FFT

입력이 순수한 실수 값을 가질 경우 FFT한 결과를 $y(n)$이라 하면 그 결과의 $n=1$ 부터 $N-1$까지의 값을 $n=N+1$부터 $2N-1$까지의 값과 비교하면 다음과 같다[4].

실수 부분 : $y(1)=y(2N-1)$, $y(2)=y(2N-2)$, …, $y(N-1)=y(N+1)$

허수 부분 : $y(1)=-y(2N-1)$, $y(2)=-y(2N-2)$, …, $y(N-1)=-y(N+1)$

이러한 결과의 중복성으로 인해 생기는 계산 시간을 단축하기 위해, 또한 FFT한 결과의 M개 ($n=1, 2, \cdots, N-1$: 밑줄친 부분)의 값만을 필요로 하므로 real FFT library를 사용하기 위해 실수값을 pack하는 루틴으로 $2N$개의 데이타를 N개의 데이타로, 실수 입력을 순서대로 실수와 허수로 pack 한 후 real FFT를 수행하였다. 입력 패턴은 아래

입력 : [(0, 1) (2, 3) (0, 0) (0, 0)]

와 같이 된다. 이때 real FFT 결과 패턴은

출력 : [(6, -2) $(-\sqrt{2}, \underline{(-2-2\sqrt{2})\,j})$ $\underline{(-2, 2j)}$ $(\sqrt{2}, \underline{(2-2\sqrt{2})\,j})$]

와 같다.

4. 실 행 결 과

1차원 FFT에서는 그림 8에서 보는 바와 같이 오차는 어느 경우나 비슷한 결과를 보였다. 이 오차값은 각각의 계산결과(31×31, 63×63, 127×127, 255×255, 511×511의 격자)의 모든 원소를 비교하여, 오차의 절대값을 취해 그중 가장 큰 값으로 하였다. 이렇게 오차가 증가하는 것은 n의 증가에 따라 A의 condition number가 증가하는데 따른 것이다.

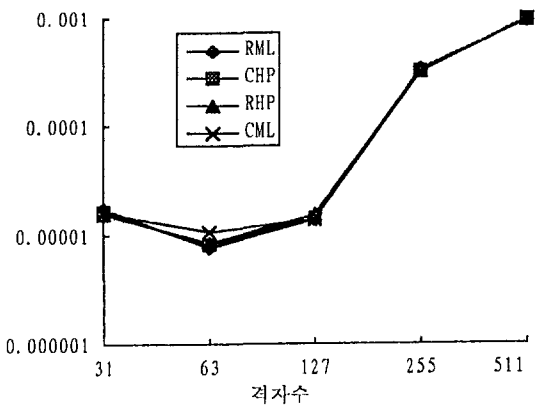


그림 8. 계 산 오 차

5가지 방법에 대해 1차원 n-point FFT만을 하였을 경우를 그림 9에 나타냈다. 앞절에서 보였듯이 real FFT는 입력을 pack하여 계산하므로 complex FFT에 비해 입력 데이타의 크기가 반밖에 안된다. 그러므로 일반적으로 real FFT는 같은 크기의 입력에서는 complex FFT보다 계산시간도 적게 걸리고, 처리할 수 있는 데이타의 양도 2배나 크다. 1차원 n-point highly parallel FFT를 이용할 경우 n개의 데이타를 한개의 프로세서에 배당하여 FFT를 수행해야 하는데, 이때 MPF프로그램상에서는 병렬로 한번에 프로세서에 입력 데이타를 배당할 수 있도록 컴파일 되지 않는다. 그러므로 이와 같은 MPF 프로그램상의 병렬처리 문제점으로 highly parallel FFT가 minimum latency FFT보다 M이 커질수록 계산 시간이 많이 소요된다. MP-2는 각각의 프로세서마다 메모리를 가지고 있다. 이런 메모리를 가지고 1차원 n-point FFT를 할 때 highly parallel FFT는 한개의 프로세서에서 n개의 데이타를 n×n의 트위들 팩터와 곱해 FFT를 하고, minimum latency FFT는 한개의 프로세서가 한개의 데이타만 처리하면 된다. 이렇게 프로세서에 데이타를 배당하는 방법의 차이로 인해 highly parallel FFT보다 minimum latency FFT가 단일 FFT를 계산할 경우 처리할 수 있는 데이타양이 많다.

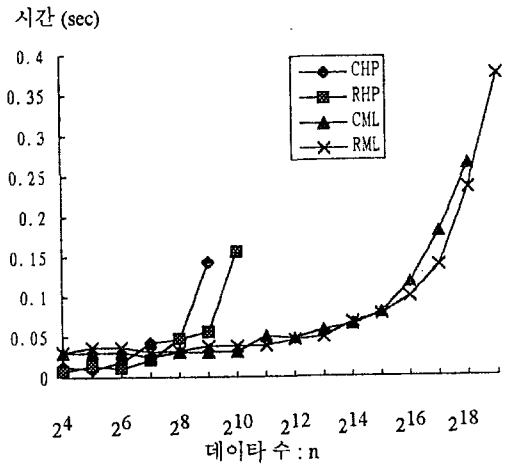


그림 9. FFT한번 계산하는데 소요되는 시간

3절에서와 같은 방법으로 해를 구하는데 필요한 시간을 그림 10에 나타냈다. 그림 10으로부터 다음의 결과를 얻었다.

1) $L \times C$ 행렬 입력을 1차원 FFT할 경우 highly parallel FFT를 수행하려면 $C \times L$개의 데이타를 C개씩 L개의 프로세서에 배당해야 한다. 그런데 MPF에서 프로세서를 2차원으로 $L1 \times L2 (L = L1 \times L2)$와 같이 배열할 수 있도록 프로그램해야 한다. 그러므로 $L \times C$ 개의 데이타를 분배하기 때문에 근본

적으로는 $L1 \times L2 \times C$와 같은 3차원 배열로 처리해야 한다. 그러나 컴파일할 때 병렬로 데이타가 분배되지 않기 때문에 M이 커질수록 mimimum latency FFT를 수행했을 때보다 총 계산 시간이 증가하게 된다.

2) $L \times C$ 행렬 입력을 1차원 FFT할 경우 mimimum latency FFT는 C개의 데이타를 병렬로 C개의 프로세서에 배당하여 L번 FFT를 수행한다. 이 때 real minimum latency FFT는 pack 루틴을 한번 더 수행한다. 그러므로 그림 6의 (b)와 같이 행렬 입력을 나열한 후, real minimum latency FFT에서 pack 하는 시간이 complex minimum latency FFT에서 복소수 입력을 만들기 위해 병렬로 입력의 허수부분을 0으로 채우는 시간보다 더 소요된다.

3) 2차원 complex minimum latency FFT의 경우는 1차원 complex minimim latency FFT에서와 같이 반복 수행에 따른 병렬처리 문제가 발생하지 않으므로 5가지 방법 중에서 가장 빠른 시간에 수행 결과를 얻을 수 있다. 그러나 M이 커질수록 트위들 팩터의 크기가 커져 프로세서의 메모리 내에서 처리할 수 있는 데이타 양의 한계로 M이 255를 넘을 경우는 계산 도중에 PE 스택 오버플로우 에러가 발생한다.

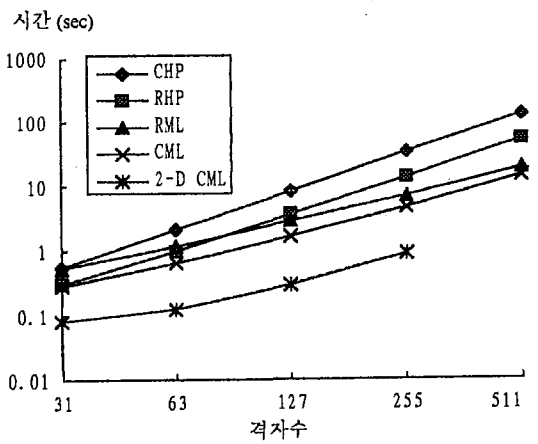


그림 10. 전체 계산시간

5. 결 론

우리는 2차원 Poisson 방정식을 풀기 위해 이 편미분 방정식을 근사화하여 선형방정식을 얻었다. 이 방정식을 Lyapunov 방정식으로 변환시킨 다음 이산정현변환(DST)을 이용해서 대수 방정식을 풀었다. 이 과정에서 DST의 결과가 FFT한 결과와 같게 하는 방법과 병렬처리 방법을 제시하였다. 또한 이 방법을 SIMD 컴퓨터 MP-2에서 실행시켜 보았다. 2차원 FFT를 수행하였을 때 격자의 갯수가 255×255개인 입력을 사용하여 0.887초 정도에서 해를 구했다.

6. 후 기

본 연구는 기초전력공학연구소(관리번호 94-001)의 지원에 의해 이루어 졌다.

요 약

일반적으로 2차원 Poisson 방정식을 풀기 위해 유한 차분법을 이용하여 tridiagonal block Toeplitz 선형방정식을 얻는다. 이 선형방정식의 독특한 형태를 활용하기 위해 Lyapunov 방정식으로 변환시킨 다음 이산정현변환(DST)을 이용해서 대각선 행렬로 만들면 계산이 용이해진다. 또 DST는 FFT를 이용해 계산할 수 있으므로 고속 계산이 가능하다. FFT를 병렬로 처리하기 위해 프로세서가 4,096개인 SIMD 컴퓨터 MP-2에서 시뮬레이션했다. 본 논문에서는 알고리즘 유도, 매핑 및 시뮬레이션 결과를 제시했다.

참 고 문 헌

1. MasPar Computer Corporation, MasPar Mathematics Library(MPML) Reference Manual, 1991.

2. J. Makhoul, "A Fast Cosine Transform in One and Two Dimensions," *IEEE Trans. Acoust., Speech, and Signal Process.*, vol. ASSP-28, no. 1, pp. 27-34, 1980.
3. K. R. Rao and P. Yip, *Discrete Cosine Transform : Algorithms, Advantages, Appl-ications*, Academic Press, 1990.
4. H. V. Sorensen, D. L. Jones, M. T. Heideman and C. S. Burrus, "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on Acoust., Speech, and Signal Process.*, vol. ASSP-35, no. 6, pp. 849-863, 1987.
5. P. N. Swarztrauber, "The Methods of Cyclic Reduction, Fourier Analysis and The FACR Algorithm for The Discrete Solution of Poisson's Equation on a Rectangle," *SIAM Rev.* vol. 19. no. 3, pp. 490-501, 1977.