

高速化した計算機システムにおける 高速フーリエ変換ソフトウェア

Fast Fourier transform software on the high-performance computer system

高橋 大介¹⁾²⁾

Daisuke TAKAHASHI

1) 埼玉大学大学院理工学研究科 (〒338-8570 埼玉県さいたま市下大久保 255)

2) 現在 筑波大学電子・情報工学系 (〒305-8573 茨城県つくば市天王台 1-1-1)

E-mail: daisuke@is.tsukuba.ac.jp)

ABSTRACT. In this paper, we propose a blocking algorithm for parallel one-dimensional fast Fourier transform (FFT) on symmetric multiprocessor (SMP) clusters. Our proposed parallel FFT algorithm is based on the six-step FFT algorithm. The six-step FFT algorithm can be altered into a block nine-step FFT algorithm to reduce the number of cache misses. We show that the block nine-step FFT algorithm improves performance by effectively utilizing the cache memory. We use the block nine-step FFT algorithm to implement the parallel one-dimensional FFT algorithm. We succeeded in obtaining performance of over 1.3 GFLOPS on an 8-node dual Pentium III 1 GHz PC SMP cluster.

1 背景

高速 Fourier 変換 (fast Fourier transform, 以下 FFT) [1] は, 科学技術計算において今日広く用いられているアルゴリズムである. FFT において大量のデータを高速に処理するために, 分散メモリ型並列計算機における FFT アルゴリズムが多く提案されている [2, 3, 4]. 多くの FFT アルゴリズムは処理するデータがキャッシュメモリに載っている場合には高い性能を示す. しかし, 問題サイズがキャッシュメモリのサイズより大きくなった場合においては著しい性能の低下をきたす. FFT アルゴリズムにおける一つの目標は, いかにしてキャッシュミスの回数を減らすかということにある.

近年のプロセッサの演算速度に対する主記憶のアクセス速度は相対的に遅くなってきており, 主記憶のアクセス回数を減らすことは, より重要になっている. したがって, SMP クラスタにおける FFT アルゴリズムでは, 演算回数だけでなく, 主記憶のアクセス回数も減らすことが今まで以上に重要である. ここで, キャッシュミスの回数を減らすことができれば, 主記憶のアクセス回数を減らす上で非常に効果があるといえる.

2 目的

本論文では, SMP クラスタにおける並列一次元 FFT のブロックアルゴリズムを提案する.

従来提案されている並列一次元 FFT アルゴリズム

[3, 4] は six-step FFT アルゴリズム [5, 3] に基づいているものが多い. この six-step FFT アルゴリズムは 2 回の multicolumn FFT と 3 回の行列の転置を必要とする. ここで, 3 回の行列の転置がキャッシュメモリを搭載したプロセッサにおいてボトルネックとなる.

このボトルネックを解消するために, six-step FFT に基づいたいくつかの FFT アルゴリズムが提案されている [3, 6]. しかし, これらの FFT アルゴリズムでは multicolumn FFT の部分と行列の転置の部分が分離されており, キャッシュ内のデータの再利用の点からはまだ改善の余地がある.

本論文で提案する並列一次元 FFT のブロックアルゴリズムでは, キャッシュ内のデータを有効に再利用し, キャッシュミスの回数を減らすために, 従来の six-step FFT では分離されていた multicolumn FFT と行列の転置を統合する. さらにキャッシュミスの回数を減らすために, 二次元表現に基づくブロック six-step FFT [7] を三次元表現に基づくブロック nine-step FFT に拡張する. このブロック nine-step FFT に基づいて並列一次元 FFT アルゴリズムを 8 ノードの dual Pentium III PC SMP クラスタに実装し, 性能評価を行う.

3 Six-Step FFT アルゴリズム

FFT は, 離散 Fourier 変換 (discrete Fourier transform, 以下 DFT) を高速に計算するアルゴリ

ズムとして知られている。DFT は次式で定義される。

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad 0 \leq k \leq n-1 \quad (1)$$

ここで, $\omega_n = e^{-2\pi i/n}$, $i = \sqrt{-1}$ である。

$n = n_1 \times n_2$ と分解できるものとする, 式 (1) における j および k は,

$$j = j_1 + j_2 n_1, \quad k = k_2 + k_1 n_2 \quad (2)$$

と書くことができる。そのとき, 式 (1) の x と y は次のような二次元配列 (columnwise) で表すことができる。

$$x_j = x(j_1, j_2), \quad 0 \leq j_1 \leq n_1 - 1, \quad 0 \leq j_2 \leq n_2 - 1 \quad (3)$$

$$y_k = y(k_2, k_1), \quad 0 \leq k_1 \leq n_1 - 1, \quad 0 \leq k_2 \leq n_2 - 1 \quad (4)$$

したがって, 式 (1) は式 (5) のように変形できる。

$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} x(j_1, j_2) \omega_{n_2}^{j_2 k_2} \omega_{n_1 n_2}^{j_1 k_2} \omega_{n_1}^{j_1 k_1} \quad (5)$$

式 (5) から次に示されるような, six-step FFT アルゴリズム [5, 3] が導かれる。

Step 1: 転置

$$x_1(j_2, j_1) = x(j_1, j_2)$$

Step 2: n_1 組の n_2 点 multicolumn FFT

$$x_2(k_2, j_1) = \sum_{j_2=0}^{n_2-1} x_1(j_2, j_1) \omega_{n_2}^{j_2 k_2}$$

Step 3: ひねり係数の乗算

$$x_3(k_2, j_1) = x_2(k_2, j_1) \omega_{n_1 n_2}^{j_1 k_2}$$

Step 4: 転置

$$x_4(j_1, k_2) = x_3(k_2, j_1)$$

Step 5: n_2 組の n_1 点 multicolumn FFT

$$x_5(k_1, k_2) = \sum_{j_1=0}^{n_1-1} x_4(j_1, k_2) \omega_{n_1}^{j_1 k_1}$$

Step 6: 転置

$$y(k_2, k_1) = x_5(k_1, k_2)$$

Step 3 における $\omega_{n_1 n_2}^{j_1 k_2}$ は, ひねり係数と呼ばれる 1 の原始根であり, 複素数である。

従来の six-step FFT アルゴリズムの特徴を以下に示す。

- $n_1 = n_2 = \sqrt{n}$ とした場合, \sqrt{n} 組の \sqrt{n} 点 multicolumn FFT [3] が Step 2 と 5 で行われる。この \sqrt{n} 点 multicolumn FFT はメモリ参照の局所性が高く, キャッシュメモリを搭載したプロセッサに適している。
- 行列の転置が 3 回必要になる。

Step 1, 4, 6 の行列の転置および, Step 3 のひねり係数の乗算をブロック化した six-step FFT アルゴリズムが文献 [3] に示されている。しかし, この FFT アルゴリズムでは multicolumn FFT の部分と行列の転置の部分が分離されているため, multicolumn FFT においてキャッシュメモリに載っていたデータが行列の転置の際に有効に再利用されないという問題点がある。

この問題点を解決するために, ブロック six-step FFT [7] が提案されている。

4 Nine-Step FFT アルゴリズム

前述の six-step FFT アルゴリズムにおいて, \sqrt{n} 点の各 column FFT は L1 キャッシュに載ると想定していたが, 問題サイズ n が非常に大きい場合には各 column FFT が L1 キャッシュに載らないことも十分予想される。このような場合は二次元表現ではなく, 多次元表現 [4] を用いて, 各 column FFT の問題サイズを小さくすることにより, L1 キャッシュ内で各 column FFT を計算することができることが知られている。

本論文では, six-step FFT アルゴリズムにおける二次元表現を三次元表現に拡張し, nine-step FFT アルゴリズムを提案する。

式 (1) において, $n = n_1 n_2 n_3$ と分解できるものとする, 式 (1) における j および k は,

$$j = j_1 + j_2 n_1 + j_3 n_1 n_2, \quad k = k_3 + k_2 n_3 + k_1 n_2 n_3 \quad (6)$$

と書くことができる。そのとき, 式 (1) の x と y は次のような三次元配列 (columnwise) で表すことができる。

$$x_j = x(j_1, j_2, j_3), \quad 0 \leq j_1 \leq n_1 - 1, \quad 0 \leq j_2 \leq n_2 - 1, \quad 0 \leq j_3 \leq n_3 - 1, \quad (7)$$

$$y_k = y(k_3, k_2, k_1), \quad 0 \leq k_1 \leq n_1 - 1, \quad 0 \leq k_2 \leq n_2 - 1, \quad 0 \leq k_3 \leq n_3 - 1 \quad (8)$$

したがって, 式 (1) は式 (6) のように変形できる。

$$y(k_3, k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2 n_3}^{j_2 k_3} \omega_{n_2}^{j_2 k_2} \omega_n^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2} \omega_{n_1}^{j_1 k_1} \quad (9)$$

式 (9) から次に示されるような, nine-step FFT アルゴリズムが導かれる。

Step 1: 転置

$$x_1(j_3, j_1, j_2) = x(j_1, j_2, j_3)$$

Step 2: $n_1 n_2$ 組の n_3 点 multicolumn FFT

$$x_2(k_3, j_1, j_2) = \sum_{j_3=0}^{n_3-1} x_1(j_3, j_1, j_2) \omega_{n_3}^{j_3 k_3}$$

Step 3: ひねり係数の乗算

1 COMPLEX*16 X(N1,N2,N3),Y(N3,N2,N1)	30 END DO
2 COMPLEX*16 U2(N3,N2),U3(N1,N2,N3)	31 END DO
3 COMPLEX*16 YWORK(N2+NP,NB),ZWORK(N3+NP,NB)	32 DO I=1,NB
4 DO J=1,N2	33 CALL IN_CACHE_FFT(YWORK(1,I),N2)
5 DO II=1,N1,NB	34 END DO
6 DO KK=1,N3,NB	35 DO J=1,N2
7 DO I=II,II+NB-1	36 DO I=II,II+NB-1
8 DO K=KK,KK+NB-1	37 X(I,J,K)=YWORK(J,I-II+1)*U3(I,J,K)
9 ZWORK(K,I-II+1)=X(I,J,K)	38 END DO
10 END DO	39 END DO
11 END DO	40 END DO
12 END DO	41 DO J=1,N2
13 DO I=1,NB	42 CALL IN_CACHE_FFT(X(1,J,K),N1)
14 CALL IN_CACHE_FFT(ZWORK(1,I),N3)	43 END DO
15 END DO	44 END DO
16 DO K=1,N3	45 DO II=1,N1,NB
17 DO I=II,II+NB-1	46 DO JJ=1,N2,NB
18 X(I,J,K)=ZWORK(K,I-II+1)*U2(K,J)	47 DO KK=1,N3,NB
19 END DO	48 DO I=II,II+NB-1
20 END DO	49 DO J=JJ,JJ+NB-1
21 END DO	50 DO K=KK,KK+NB-1
22 END DO	51 Y(K,J,I)=X(I,J,K)
23 DO K=1,N3	52 END DO
24 DO II=1,N1,NB	53 END DO
25 DO JJ=1,N2,NB	54 END DO
26 DO I=II,II+NB-1	55 END DO
27 DO J=JJ,JJ+NB-1	56 END DO
28 YWORK(J,I-II+1)=X(I,J,K)	57 END DO
29 END DO	

図 1: ブロック nine-step FFT アルゴリズム

- Step 4: 転置
- $$x_3(k_3, j_1, j_2) = x_2(k_3, j_1, j_2)\omega_{n_2 n_3}^{j_2 k_3}$$
- Step 5: $n_1 n_3$ 組の n_2 点 multicolumn FFT
- $$x_5(k_2, j_1, k_3) = \sum_{j_2=0}^{n_2-1} x_4(j_2, j_1, k_3)\omega_{n_2}^{j_2 k_2}$$
- Step 6: ひねり係数の乗算
- $$x_6(k_2, j_1, k_3) = x_5(k_2, j_1, k_3)\omega_n^{j_1 k_3}\omega_{n_1 n_2}^{j_1 k_2}$$
- Step 7: 転置
- $$x_7(j_1, k_2, k_3) = x_6(k_2, j_1, k_3)$$
- Step 8: $n_2 n_3$ 組の n_1 点 multicolumn FFT
- $$x_8(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} x_7(j_1, k_2, k_3)\omega_{n_1}^{j_1 k_1}$$
- Step 9: 転置
- $$y(k_3, k_2, k_1) = x_8(k_1, k_2, k_3)$$

nine-step FFT アルゴリズムの特徴を以下に示す .

- $n_1 = n_2 = n_3 = n^{1/3}$ とした場合 , $n^{2/3}$ 組の $n^{1/3}$ 点 multicolumn FFT が Step 2 , 5 と 8 で行われる . この $n^{1/3}$ 点 multicolumn FFT は six-step FFT に比べてメモリ参照の局所性が高く , キャッシュメモリを搭載したプロセッサにより適している .
- 行列の転置が 4 回必要となり , six-step FFT に比べて行列の転置が 1 回余分に必要になる .

5 ブロック Nine-Step FFT アルゴリズム

ブロック six-step FFT アルゴリズムにおけるブロック化 [7] と同様の手法を用いて , ブロック nine-step FFT アルゴリズムを構成することができる . 本論文では , キャッシュ内のデータを有効に再利用し , キャッシュミス回数を少なくするため , ブロック six-step FFT と同様に , multicolumn FFT と行列の転置を統合する . 前述した nine-step FFT において , $n = n_1 n_2 n_3$ とし , n_b をブロックサイズとする . ここで , プロセッサは multi-level キャッシュメモリを搭載しているものと仮定する . 図 1 に提案するブロック nine-step FFT アルゴリズムの疑似コードを示す . 図 1 のアルゴリズムを説明すると , 以下のようになる .

- Step 1: $n_1 \times n_2 \times n_3$ の大きさの複素数配列 X に入力データが入っているとす . このとき , $n_1 \times n_2 \times n_3$ 配列 X から n_b 列ずつデータを転置しながら , $n_3 \times n_b$ の大きさの作業用配列 $ZWORK$ に転送する . ここでブロックサイズ n_b は配列 $ZWORK$ が L2 キャッシュに載るように定める .
- Step 2: n_b 組の n_3 点 multicolumn FFT を L2 キャッシュに載っている $n_3 \times n_b$ 配列 $ZWORK$ の上で行う . ここで各 column FFT は , ほぼ L1 キャッシュ内で行えるものとする .
- Step 3: multicolumn FFT を行った後 L2 キャッシュに残っている $n_3 \times n_b$ 配列 $ZWORK$ の各要素にひねり係数 $U2$ の乗算を行う . そしてこの $n_3 \times n_b$ 配列 $ZWORK$ のデータを n_b 列ずつ転置しながら元の $n_1 \times n_2 \times n_3$ 配列 X の同じ場所に再び格納する .
- Step 4: このとき , $n_1 \times n_2 \times n_3$ 配列 X から n_b 列ずつデータを転置しながら , $n_2 \times n_b$ の大きさの

作業用配列 YWORK に転送する .

Step 5: n_b 組の n_2 点 multicolumn FFT を L2 キャッシュに載っている $n_2 \times n_b$ 配列 YWORK の上で行う . ここで各 column FFT は , ほぼ L1 キャッシュ内で行えるものとする .

Step 6: multicolumn FFT を行った後 L2 キャッシュに残っている $n_2 \times n_b$ 配列 YWORK の各要素にひねり係数 U3 の乗算を行う . そしてこの $n_2 \times n_b$ 配列 YWORK のデータを n_b 列ずつ転置しながら元の $n_1 \times n_2 \times n_3$ 配列 X の同じ場所に再び格納する .

Step 7: $n_3 n_2$ 組の n_1 点 multicolumn FFT を $n_1 \times n_2 \times n_3$ 配列 X の上で行う . ここでも各 column FFT は , ほぼ L1 キャッシュ内で行える .

Step 8: 最後にこの $n_1 \times n_2 \times n_3$ 配列 X を n_b 列ずつ転置して , $n_3 \times n_2 \times n_1$ 配列 Y に格納する .

図 1 のアルゴリズムにおいて , NB はブロックサイズであり , NP はパディングサイズ , WORK は作業用の配列である . また , 作業用の配列 YWORK , ZWORK にパディングを施すことにより , キャッシュラインコンフリクトの発生を極力防ぐことができる .

提案するブロック nine-step FFT アルゴリズムは , いわゆる *three-pass* アルゴリズムとなる . つまり , 各 column FFT に基数 2 の FFT を用いた場合 , 提案する n 点ブロック nine-step FFT の演算回数は $5n \log_2 n$ であるのに対し , 配列 X , Y へのアクセス回数の合計は $3n$ 回の load と $3n$ 回の store で済む .

6 ブロック Nine-Step FFT に基づく並列一次元 FFT アルゴリズム

並列一次元 FFT アルゴリズムとしては , six-step FFT に基づく並列一次元 FFT アルゴリズム [3, 4] が知られている . しかし , 並列一次元 FFT が主に対象とする , 問題サイズ n が非常に大きい場合には \sqrt{n} 点の各 column FFT が L1 キャッシュに載らないことが多く , キャッシュミスが多発することが指摘されている [6] .

そこで , 本論文ではブロック nine-step FFT に基づく並列一次元 FFT アルゴリズムを提案する .

一次元 FFT においてデータ数 N が $N = N_1 \times N_2 \times N_3$ と分解されるとし , P をプロセッサ数とする . すると , 一次元配列 $x(N)$ は三次元配列 $x(N_1, N_2, N_3)$ と表すことができる . P 個のプロセッサを持つ分散メモリ型並列計算機では , この配列 $x(N_1, N_2, N_3)$ は一次元目 (N_1) に沿って分散される . N_1 が P で割り切れるとすると , 各プロセッサには N/P 個のデータが分散されることになる .

やや複雑になるが , ここで $\hat{N}_r \equiv N_r/P$ の記法を導入する . そして , インデックス J_r に沿ったデータがすべての P 個のプロセッサに分散されることを示す記法を \hat{J}_r とする . なお , r は次元 r にインデックスが属しているという意味である .

これより , 分散された三次元配列は $\hat{x}(\hat{N}_1, N_2, N_3)$ と表すことができる . サイクリック分割によると , m 番目のプロセッサにおけるローカルインデックス $\hat{J}_r(m)$ は , 次のようなグローバルインデックス J_r に一致する .

$$J_r = \hat{J}_r(m) \times P + m, \quad 0 \leq m \leq P - 1, \\ 1 \leq r \leq 3 \quad (10)$$

ここで全対全通信を示すために , $\tilde{N}_i \equiv N_i/P_i$ の記法を導入する . この記法を用いると , N_i は \tilde{N}_i と P_i の二次元表現に分解される . なお , P_i は P と同じものを示しているが , このインデックスが次元 i に属していることを示している .

初期データを $\hat{x}(\hat{N}_1, N_2, N_3)$ とすると , nine-step FFT に基づく並列一次元 FFT アルゴリズムは次のようになる .

Step 1: 転置

$$\hat{x}_1(J_3, \hat{J}_1, J_2) = \hat{x}(\hat{J}_1, J_2, J_3)$$

Step 2: $(N_1/P) \cdot N_2$ 組の N_3 点 multicolumn FFT

$$\hat{x}_2(K_3, \hat{J}_1, J_2) \\ = \sum_{J_3=0}^{N_3-1} \hat{x}_1(J_3, \hat{J}_1, J_2) \omega_{N_3}^{J_3 K_3}$$

Step 3: ひねり係数の乗算およびプロセッサ内再配置

$$\hat{x}_3(\hat{J}_1, J_2, \hat{K}_3, P_3) \\ = \hat{x}_2(P_3, \hat{K}_3, \hat{J}_1, J_2) \omega_{N_2 N_3}^{J_2 K_3} \\ \equiv \hat{x}_2(K_3, \hat{J}_1, J_2) \omega_{N_2 N_3}^{J_2 K_3}$$

Step 4: 全対全通信

$$\hat{x}_4(\tilde{J}_1, J_2, \hat{K}_3, P_1) = \hat{x}_3(\hat{J}_1, J_2, \hat{K}_3, P_3)$$

Step 5: プロセッサ内再配置

$$\hat{x}_5(J_2, \tilde{J}_1, \hat{K}_3, P_1) = \hat{x}_4(\tilde{J}_1, J_2, \hat{K}_3, P_1)$$

Step 6: $N_1 \cdot (N_3/P)$ 組の N_2 点 multicolumn FFT

$$\hat{x}_6(K_2, \tilde{J}_1, \hat{K}_3, P_1) \\ = \sum_{J_2=0}^{N_2-1} \hat{x}_5(J_2, \tilde{J}_1, \hat{K}_3, P_1) \omega_{N_2}^{J_2 K_2}$$

Step 7: ひねり係数の乗算およびプロセッサ内再配置

$$\hat{x}_7(J_1, K_2, \hat{K}_3) \equiv \hat{x}_6(P_1, \tilde{J}_1, K_2, \hat{K}_3) \\ = \hat{x}_6(K_2, \tilde{J}_1, \hat{K}_3, P_1) \omega_N^{J_1(K_3 + K_2 N_3)}$$

Step 8: $N_2 \cdot (N_3/P)$ 組の N_1 点 multicolumn FFT

$$\hat{x}_8(K_1, K_2, \hat{K}_3) \\ = \sum_{J_1=0}^{N_1-1} \hat{x}_7(J_1, K_2, \hat{K}_3) \omega_{N_1}^{J_1 K_1}$$

Step 9: 転置

$$\hat{y}(\hat{K}_3, K_2, K_1) = \hat{x}_8(K_1, K_2, \hat{K}_3)$$

nine-step FFT に基づく並列一次元 FFT アルゴリズムの特徴は , 次に示すとおりである .

- $N_1 = N_2 = N_3 = N^{1/3}$ とした場合 , $N^{2/3}/P$ 組の $N^{1/3}$ 点 multicolumn FFT が Step 2 , 6 と 8 で実行される .

表 1: dual Pentium III PC SMP クラスタにおける並列一次元 FFT の性能

P (Nodes \times CPUs)	N	Block Nine-Step FFT		FFTW	
		Time	MFLOPS	Time	MFLOPS
1 \times 1	2^{23}	5.40606	178.45	10.50152	91.86
1 \times 2	2^{23}	3.33968	288.86	7.49437	128.72
2 \times 1	2^{24}	7.46566	269.67	16.55127	121.64
2 \times 2	2^{24}	4.99214	403.29	11.96556	168.26
4 \times 1	2^{25}	8.22695	509.82	17.79209	235.74
4 \times 2	2^{25}	6.01907	696.84	15.44108	271.63
8 \times 1	2^{26}	8.68712	1004.26	19.33295	451.26
8 \times 2	2^{26}	6.58020	1325.82	18.06414	482.95

表 2: dual Pentium III PC SMP クラスタにおける全対全通信の性能

P (Nodes \times CPUs)	N	Time	MB/sec
1 \times 2	2^{23}	0.46537	72.10
2 \times 1	2^{24}	2.18825	30.67
2 \times 2	2^{24}	2.00209	25.14
4 \times 1	2^{25}	2.48046	40.58
4 \times 2	2^{25}	2.60625	22.53
8 \times 1	2^{26}	3.01393	38.97
8 \times 2	2^{26}	3.46417	18.16

- 全対全通信が 1 回で済む。しかも，入力データ x と出力データ y は共に正順となる。

ブロック nine-step FFT アルゴリズムにおけるブロック化と同様の手法を用いて，ブロック nine-step FFT に基づく並列一次元 FFT アルゴリズムを構成することができる。

7 In-Cache FFT アルゴリズム

前述の multicolumn FFT において，各 column FFT がキャッシュに載る場合の in-cache FFT には Stockham アルゴリズム [3] を用いた。

2 点 FFT を除く 2 べきの FFT では，基数 4 と基数 8 の組み合わせにより FFT を計算し，基数 2 の FFT カーネルを排除することにより，ロードとストア回数および演算回数を減らすことができ，より高い性能を得ることができる [8]。具体的には， $n = 2^p$ ($p \geq 2$) 点 FFT を $n = 4^q 8^r$ ($0 \leq q \leq 2, r \geq 0$) として計算することにより，基数 4 と基数 8 の FFT カーネルのみで $n \geq 4$ の場合に 2 べきの FFT を計算することができる。

8 性能評価

性能評価にあたっては，提案するブロック nine-step FFT に基づく並列一次元 FFT と，多くのプロセッサにおいて最も高速な FFT ライブラリとして

知られている FFTW (version 2.1.3) [9] の性能比較を行った。 $N = 2^m$ の m およびプロセッサ数 P を変化させて順方向 FFT を連続 10 回実行し，その平均の経過時間を測定した。なお，FFT の計算は倍精度複素数で行い，三角関数のテーブルはあらかじめ作り置きとしている。

SMP クラスタとしては，8 ノードの dual Pentium III 1 GHz PC SMP クラスタ (i840 chipset, 総メモリ容量 8 GB RDRAM, Linux 2.2.16 + RWC SCORE 3.3.1) を用いた。SMP クラスタの各ノードは，1000Base-SX の Gigabit Ethernet (NIC: 3Com 3C985B-SX) で接続されている。

通信ライブラリとしては，MPICH-SCore [10] を用いた。なお，PC SMP クラスタにおいて今回の性能評価ではノード内 MPI を用いている。

提案する並列一次元 FFT アルゴリズムにおいて，コンパイラは g77 version 2.95.2 を用いた。一方，FFTW ライブラリにおいて，コンパイラは gcc version 2.95.2 を用いた。

提案するブロック nine-step FFT に基づく並列一次元 FFT の性能および FFTW の性能を表 1 に示す。ここで，実行時間の単位は秒であり， $N = 2^m$ 点 FFT の MFLOPS 値は $5N \log_2 N$ より算出している。

表 2 に，dual Pentium III PC SMP クラスタにおける全対全通信の性能を示す。ここで，実行時間の単位は秒であり，通信性能 (MB/sec) は全対全通信の通信量 $(P-1) \times (16N/P^2)$ (バイト) より算出している。表 1 から分かるように，提案するブロック nine-step FFT に基づく並列一次元 FFT が FFTW に比べて高い性能が発揮できていることが分かる。特に， $N = 2^{26}$, $P = 8 \times 2$ の場合には提案するブロック nine-step FFT に基づく並列一次元 FFT は，FFTW に比べて約 2.75 倍高速である。

この理由としては，

- 入力データと出力データを正順にした場合，FFTW では全対全通信が 3 回必要なのに対し，提案するブロック nine-step FFT に基づく並列一次元 FFT では全対全通信がわずか 1 回で済むので，通信時間が FFTW に比べて 1/3 程度に抑えられている。

- FFTW では，six-step FFT に基づいて並列一次元 FFT を実現しているため， \sqrt{n} 点の各 column

FFTを行っている。しかし、 n が非常に大きい場合には nine-step FFT のように $n^{1/3}$ 点の各 column FFT を行う方がキャッシュミス回数が少なくなる。

- FFTW では、各プロセッサ内でデータを再帰的に二分木状に分割することによって、最終的に小さな点数の DFT に帰着させるというアプローチを取っている。提案するブロック nine-step FFT に基づく並列次元 FFT では明示的にブロック化を行うことにより、キャッシュ上のデータが FFTW に比べて有効に活用できている。

があげられる。

つまり、提案するブロック nine-step FFT に基づく並列次元 FFT は、ノード内の演算性能、通信回数のいずれの面においても、FTFW ライブラリに比べて優れていることが分かる。また、表 1 から、8 ノードの dual Pentium III 1 GHz PC SMP クラスタでは提案するブロック nine-step FFT に基づく並列次元 FFT において、 $N = 2^{26}$ の場合に 1.3 GFLOPS を越える性能が得られていることが分かる。

9 まとめ

本論文では、SMP クラスタにおける並列次元 FFT のブロックアルゴリズムを提案した。提案するブロック nine-step FFT に基づく並列次元 FFT では、キャッシュメモリの再利用率を高くすることにより、キャッシュミスを少なくし、その結果主記憶のアクセス回数も少なくすることができた。さらに、入力データと出力データを正順にした場合でも、全対全通信が 1 回で済むことにより、SMP クラスタにおいて実行時間のかなりの部分を占める通信時間を少なくすることができた。

本論文で提案したブロック nine-step FFT に基づく並列次元 FFT は、プロセッサの演算速度と主記憶のアクセス速度との差が大きく、プロセッサ間通信性能が低い場合に、従来の並列次元 FFT に比べてより効果的であると考えられる。

提案するブロック nine-step FFT に基づいて、並列次元 FFT を dual Pentium III PC SMP クラスタに実現し、性能評価を行った。その結果、8 ノードの dual Pentium III 1 GHz PC SMP クラスタでは 1.3 GFLOPS を越える性能を得ることができた。

10 参加企業および機関

本プロジェクトへの参加企業と機関は以下の通りである。

プロジェクト実施管理組織：株式会社三菱総合研究所

謝辞

本研究は、IPA（情報処理振興事業協会）の平成 12 年度末踏ソフトウェア創造事業の支援を受けまし

た。また、PM（プロジェクトマネージャー）として、貴重な助言を頂いた東京大学大学院情報理工学系研究科の平木敬教授に深く感謝致します。

参考文献

- [1] Cooley, J. W. and Tukey, J. W.: An Algorithm for the Machine Calculation of Complex Fourier Series, *Math. Comput.*, Vol. 19, pp. 297–301 (1965).
- [2] Swarztrauber, P. N.: Multiprocessor FFTs, *Parallel Computing*, Vol. 5, pp. 197–210 (1987).
- [3] Van Loan, C.: *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, PA (1992).
- [4] Agarwal, R. C., Gustavson, F. G. and Zubair, M.: A High Performance Parallel Algorithm for 1-D FFT, *Proc. Supercomputing '94*, pp. 34–40 (1994).
- [5] Bailey, D. H.: FFTs in External or Hierarchical Memory, *The Journal of Supercomputing*, Vol. 4, pp. 23–35 (1990).
- [6] Wadleigh, K. R.: High Performance FFT Algorithms for Cache-Coherent Multiprocessors, *The International Journal of High Performance Computing Applications*, Vol. 13, pp. 163–171 (1999).
- [7] 高橋大介: 共有メモリ型並列計算機における並列次元 FFT のブロックアルゴリズム, 並列処理シンポジウム JSPP2001, pp. 359–366 (2001).
- [8] Takahashi, D.: High-Performance Parallel FFT Algorithms for the HITACHI SR8000, *Proc. Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region (HPC-Asia 2000)*, pp. 192–199 (2000).
- [9] Frigo, M. and Johnson, S. G.: The Fastest Fourier Transform in the West, Technical Report MIT-LCS-TR-728, MIT Lab for Computer Science (1997).
- [10] Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T. and Ishikawa, Y.: High Performance Communication using a Commodity Network for Cluster Systems, *Proc. Ninth International Symposium on High Performance Distributed Computing (HPDC-9)*, pp. 139–146 (2000).