PC 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

高橋 大介[†] 朴 泰祐[†] 佐藤 三久[†]

本論文では，PC 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．提案する並列一次元 FFT アルゴリズムは，six-step FFT アルゴリズムに基づいている．キャッシュミスの回数を減らすために，six-step FFT アルゴリズムはブロック nine-step FFT に変更することができる．このブロック nine-step FFT アルゴリズムではキャッシュメモリを効果的に利用することにより，性能が改善されることを示す．ブロック nine-step FFT アルゴリズムに基づいて，並列一次元 FFT を dual Pentium III PC SMP クラスタに実装し，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 を超える性能を得ることができた．

A Blocking Algorithm for Parallel 1-D FFT on Clusters of PCs

Daisuke Takahashi,[†] Taisuke Boku[†] and Mitsuhisa Sato[†]

In this paper, we propose a blocking algorithm for a parallel one-dimensional fast Fourier transform (FFT) on clusters of PCs. 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. The block nine-step FFT algorithm improves performance by utilizing the cache memory effectively. We use the block nine-step FFT algorithm to design the parallel one-dimensional FFT algorithm. We successfully achieved performance of over 1.3 GFLOPS on an 8-node dual Pentium III 1 GHz PC SMP cluster.

1. はじめに

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)~9)]．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 1 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，PC クラスタにおける FFT アルゴリズムでは，演算回数だけではなく，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うえで非常に効果があるといえる．

本論文では，PC 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[4),5)]は six-step FFT アルゴリズム[4),10)]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[4),11)]．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

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

ために，二次元表現に基づくブロック six-step FFT [12)]を三次元表現に基づくブロック nine-step FFT に拡張する．

多くの並列一次元 FFT アルゴリズムにおいては，全対全通信を用いてノード間でデータを入れ換える必要があることが知られている[5),6),8)]．全対全通信は，分散メモリ型並列計算機ではコストの高い通信であるので，回数をできるだけ減らす必要がある．

この全対全通信に関しては，入力のデータ配置を転置したものを出力とすることにより，全対全通信の回数を 1 回にする手法[5)]が提案されているが，この手法では入力と出力を同じデータ配置にする必要がある場合には，全対全通信が 3 回必要となる．

全対全通信の回数を減らすために，Edelman らは fast multipole method [13),14)]を用いて仮想的にデータを入れ換えることにより，入力と出力を同じデータ配置にした場合でも全対全通信の回数を 1 回にする手法を提案している[8)]．

本論文では，データ分割にサイクリック分割を適用するとともに，ノード内で行列の転置を行ってから全対全通信を行い，その後にノード内で行列の再配置を行うことにより，Edelman らの手法のような複雑な処理を行わずに，入力と出力を同じデータ配置にした場合でも全対全通信の回数を 1 回に減らす手法を示す．

提案するブロック nine-step FFT に基づく並列一次元 FFT アルゴリズムを 8 ノードの dual Pentium III PC SMP クラスタに実現し，性能評価を行う．

以下，2 章で six-step FFT について簡単に説明する．3 章で提案する nine-step FFT アルゴリズムについて，4 章で提案するブロック nine-step FFT アルゴリズムについて，5 章でブロック nine-step FFT に基づく並列一次元 FFT アルゴリズムについて述べる．6 章でノード内における in-cache FFT アルゴリズムについて述べる．7 章で本論文で示す並列一次元 FFT アルゴリズムの性能評価結果を示す．最後の 8 章はまとめである．

2. 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．

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad 0 \le k \le n-1 \tag{1}$$

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と，式 (1) における j および k は，

$$j = j_1 + j_2 n_1, \quad k = k_2 + k_1 n_2 \tag{2}$$

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．

$$x_j = x(j_1, j_2), \quad 0 \le j_1 \le n_1 - 1, \quad 0 \le j_2 \le n_2 - 1 \tag{3}$$

$$y_k = y(k_2, k_1), \quad 0 \le k_1 \le n_1 - 1, \quad 0 \le k_2 \le n_2 - 1 \tag{4}$$

したがって，式 (1) は式 (5) のように変形できる．

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

式 (5) から次に示されるような，six-step FFT アルゴリズム[4),10)]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 [4)]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 4) 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 [12)]が提案されている．

3. 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[5)]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と，式 (1) における j および k は，

$$\begin{aligned} j &= j_1 + j_2 n_1 + j_3 n_1 n_2, \\ k &= k_3 + k_2 n_3 + k_1 n_2 n_3 \end{aligned} \qquad (6)$$

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．

$$\begin{aligned} & x_j = x(j_1, j_2, j_3), \quad 0 \le j_1 \le n_1 - 1, \\ & \quad 0 \le j_2 \le n_2 - 1, \quad 0 \le j_3 \le n_3 - 1, \end{aligned} \qquad (7)$$

$$\begin{aligned} & y_k = y(k_3, k_2, k_1), \quad 0 \le k_1 \le n_1 - 1, \\ & \quad 0 \le k_2 \le n_2 - 1, \quad 0 \le k_3 \le n_3 - 1 \end{aligned} \qquad (8)$$

したがって，式 (1) は式 (6) のように変形できる．

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

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

$$x_3(k_3, j_1, j_2) = x_2(k_3, j_1, j_2) \omega_{n_2 n_3}^{j_2 k_3}$$

Step 4: 転置

$$x_4(j_2, j_1, k_3) = x_3(k_3, j_1, j_2)$$

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

$$\begin{aligned} & x_6(k_2, j_1, k_3) \\ & \quad = x_5(k_2, j_1, k_3) \omega_{n}^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2} \end{aligned}$$

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 に比べてメモリアクセスの局所性が高く，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．

n が非常に大きく，$n^{1/3}$ 点 FFT がキャッシュに載らない場合には，三次元表現を四次元表現にすることで，multicolumn FFT におけるメモリアクセスの局所性を高くすることができるが，行列の転置は 5 回必要になる．このように多次元表現を用いることで，より大きなサイズの n に対してメモリアクセスの局所性を引き出すことが可能になるが，その一方で行列の転置の回数は次元数が大きくなるに従って増えてしまう．

したがって，n の大きさや，プロセッサのキャッシュサイズ，そして主記憶のアクセス速度等により，最適な次元数は異なることに注意する．

4. 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[12)]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，キャッシュミスの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，$n = n_1 n_2 n_3$

```
COMPLEX*16 X(N1,N2,N3),Y(N3,N2,N1)
COMPLEX*16 U2(N3,N2),U3(N1,N2,N3)
COMPLEX*16 YWORK(N2+NP,NB),ZWORK(N3+NP,NB)
DO J=1,N2
  DO II=1,N1,NB
    DO KK=1,N3,NB
      DO I=II,II+NB-1
        DO K=KK,KK+NB-1
          ZWORK(K,I-II+1)=X(I,J,K)
        END DO
      END DO
    END DO
    DO I=1,NB
      CALL IN_CACHE_FFT(ZWORK(1,I),N3)
    END DO
    DO K=1,N3
      DO I=II,II+NB-1
        X(I,J,K)=ZWORK(K,I-II+1)*U2(K,J)
      END DO
    END DO
  END DO
END DO
DO K=1,N3
  DO II=1,N1,NB
    DO JJ=1,N2,NB
      DO I=II,II+NB-1
        DO J=JJ,JJ+NB-1
          YWORK(J,I-II+1)=X(I,J,K)
        END DO
      END DO
    END DO
    DO I=1,NB
      CALL IN_CACHE_FFT(YWORK(1,I),N2)
    END DO
    DO J=1,N2
      DO I=II,II+NB-1
        X(I,J,K)=YWORK(J,I-II+1)*U3(I,J,K)
      END DO
    END DO
  END DO
  DO J=1,N2
    CALL IN_CACHE_FFT(X(1,J,K),N1)
  END DO
END DO
DO II=1,N1,NB
  DO JJ=1,N2,NB
    DO KK=1,N3,NB
      DO I=II,II+NB-1
        DO J=JJ,JJ+NB-1
          DO K=KK,KK+NB-1
            Y(K,J,I)=X(I,J,K)
          END DO
        END DO
      END DO
    END DO
  END DO
END DO
```

図 1 ブロック nine-step FFT アルゴリズム
Fig. 1 A block nine-step FFT algorithm.

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る．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 n_b を示しており，`NP` はパディングサイズ，`YWORK`, `ZWORK` 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．

ブロックサイズ n_b が大きいほど，主記憶とキャッシュ間のデータ転送の効率が上がるが，逆に作業用配列がキャッシュに収まらなくなる可能性が出てくる．つまり，問題サイズ n が比較的小さいときには，ブロックサイズ n_b を大きくしても，作業用の配列 YWORK，ZWORK はキャッシュに載るが，n が大きくなった場合には n_b を小さくする必要がある．このように，n の大きさや，プロセッサのキャッシュサイズ，そして主記憶のアクセス速度等により，最適なブロックサイズ n_b は異なることに注意する．

5. 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[4),5)]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[11)]．

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．

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

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

$$\hat{x_2}(K_3, \hat{J}_1, J_2) = \sum_{J_3=0}^{N_3-1} \hat{x_1}(J_3, \hat{J}_1, J_2)\omega_{N_3}^{J_3K_3}$$

Step 3: ひねり係数の乗算およびプロセッサ内再配置

$$\begin{aligned}\hat{x_3}(\hat{J}_1, J_2, \tilde{K}_3, P_3) &= \hat{x_2}(P_3, \tilde{K}_3, \hat{J}_1, J_2)\omega_{N_2N_3}^{J_2K_3} \\ &\equiv \hat{x_2}(K_3, \hat{J}_1, J_2)\omega_{N_2N_3}^{J_2K_3}\end{aligned}$$

Step 4: 全対全通信

$$\hat{x_4}(\tilde{J}_1, J_2, \hat{K}_3, P_1) = \hat{x_3}(\hat{J}_1, J_2, \tilde{K}_3, P_3)$$

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

$$\hat{x_6}(K_2, \tilde{J}_1, \hat{K}_3, P_1) = \sum_{J_2=0}^{N_2-1} \hat{x_5}(J_2, \tilde{J}_1, \hat{K}_3, P_1)\omega_{N_2}^{J_2K_2}$$

Step 7: ひねり係数の乗算およびプロセッサ内再配置

$$\begin{aligned}\hat{x_7}(J_1, K_2, \hat{K}_3) &\equiv \hat{x_7}(P_1, \tilde{J}_1, K_2, \hat{K}_3) \\ &= \hat{x_6}(K_2, \tilde{J}_1, \hat{K}_3, P_1)\omega_N^{J_1(\hat{K}_3+K_2N_3)}\end{aligned}$$

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

$$\hat{x_8}(K_1, K_2, \hat{K}_3)$$

表 1 最内側ループにおける基数 2，4，8 の DIF Stockham アルゴリズムに基づく FFT カーネルの実演算回数（PentiumIII）

Table 1 Real inner-loop operations for radix-2, 4 and 8 FFT kernels based on the DIF Stockham algorithm on PentiumIII processor.

	基数 2	基数 4	基数 8
ロード＋ストア回数	8	16	32
乗算回数	4	12	32
加算回数	6	22	66
総浮動小数点演算回数 ($n \log_2 n$)	5.000	4.250	4.083
浮動小数点演算命令数	10	34	98
浮動小数点演算命令数とロード＋ストア回数の比	1.250	2.125	3.063

$$= \sum_{J_1=0}^{N_1-1} \hat{x_7}(J_1,\, K_2,\, \hat{K_3})\omega_{N_1}^{J_1 K_1}$$

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．
- 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 はともに正順となる．

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．

6. ノード内における 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ノード内における in-cache FFT には Stockham アルゴリズム[15]を用いた．

2 べきの FFT では，基数 2 の FFT に比べて演算回数およびメモリアクセスのより少ない基数 4，8 の FFT [16]を適用することにより，効率を高くすることができる[4]．したがって，本論文では基数 2，4，8 の組合せで実現および評価を行った．

表 1 は最内側ループにおける基数 2，4，8 の DIF Stockham アルゴリズムに基づく FFT カーネルの実演算回数を示している．表 1 における浮動小数点演算命令数とは，浮動小数点の乗算，加算をそれぞれ 1 命令とした場合の演算命令数である．

表 1 から分かるように，大きい基数の FFT カーネルはロード＋ストア回数の面からも演算回数の面からも有利である．さらに，浮動小数点演算命令数とロード＋ストア回数の比は，基数 8 の FFT は基数 2 の FFT に比べて 2.45 倍であり，基数 4 の FFT に比べても約 1.44 倍となっている．これは，基数を大きくするに従ってデータを再利用できる回数が増えるためにロードとストアの回数が減るからである[4]．

2 点 FFT を除く 2 べきの FFT では，基数 4 と基数 8 の組合せにより 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[9]．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．

7. 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）[7]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．また，提案するブロック nine-step FFT に基づく並列一次元 FFT において，図 1 のブロックサイズを `NB=4` とし，パディングサイズを `NP=2` としている．

PC クラスタとしては，8 ノードの dual PentiumIII PC SMP クラスタ（Coppermine 1 GHz, i840 chipset，総メモリ容量 8 GB RDRAM，Linux 2.2.16 + RWC SCore 3.3.1）を用いた．PC 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 [17]を用いた．なお，PC SMP クラスタにおいて今回の性能評

表 2　dual Pentium III PC SMP クラスタにおける並列一次元 FFT の性能
Table 2　Performance of parallel one-dimensional FFTs on dual Pentium III PC SMP cluster.

P (Nodes × CPUs)	N	Block nine-step FFT Time	Block nine-step FFT MFLOPS	FFTW Time	FFTW MFLOPS	Time Ratio
1×1	2^{23}	5.40606	178.45	10.50152	91.86	1.943
1×2	2^{23}	3.33968	288.86	7.49437	128.72	2.244
2×1	2^{24}	7.46566	269.67	16.55127	121.64	2.217
2×2	2^{24}	4.99214	403.29	11.96556	168.26	2.397
4×1	2^{25}	8.22695	509.82	17.79209	235.74	2.163
4×2	2^{25}	6.01907	696.84	15.44108	271.63	2.565
8×1	2^{26}	8.68712	1004.26	19.33295	451.26	2.225
8×2	2^{26}	6.58020	1325.82	18.06414	482.95	2.745

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 2** 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．なお，表 2 の一番右の列の Time Ratio は FFTW の実行時間を，提案するブロック nine-step FFT に基づく並列一次元 FFT の実行時間で割った値を示している．表 2 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．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．

図 2 に，dual Pentium III PC SMP クラスタにおける，提案するブロック nine-step FFT に基づく並列一次元 FFT と FFTW の性能を $N = 2^{23}$ に固定した場合について示す．図 2 から，提案するブロック nine-step FFT に基づく並列一次元 FFT は FFTW に比べて高い性能が発揮できていることが分かる．

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 を実現しているので，$\sqrt{N}$ 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 6 章で述べた in-cache FFT において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とっている．提案するブロック nine-step FFT に基づく並列一次元 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，
- 1 ノード，1 CPU の（つまりノード間通信がない）場合においても，提案するブロック nine-step FFT は FFTW に比べて約 1.94 倍高速であることから分かるように，全対全通信回数の削減だけではなく，明示的なブロック化も性能向上に大きく貢献している，

があげられる．

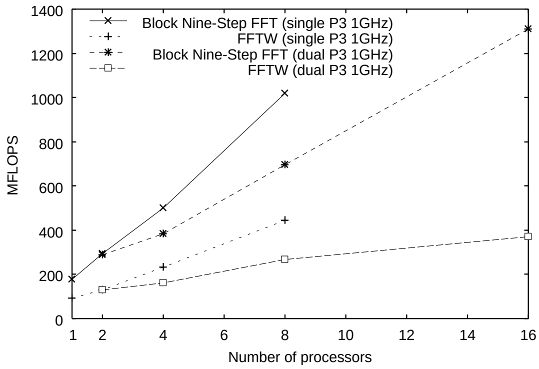

図 2　dual Pentium III PC SMP クラスタにおける並列一次元 FFT の性能（$N = 2^{23}$）
Fig. 2　Performance of parallel one-dimensional FFTs on dual Pentium III PC SMP cluster ($N = 2^{23}$).

つまり，提案するブロック nine-step FFT に基づく並列一次元 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，FFTW に比べて優れていることが分かる．また，表 2 から，8 ノードの dual

表 3　dual Pentium III PC SMP クラスタにおける全対全通信の性能
Table 3　All-to-all communication performance on dual Pentium III PC SMP cluster.

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

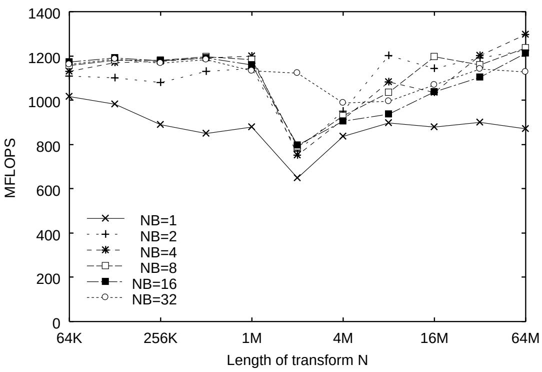

図 3　dual Pentium III PC SMP クラスタにおけるブロック nine-step FFT に基づく並列一次元 FFT の性能（8 ノード，16 CPU）
Fig. 3　Performance of block nine-step FFT-based parallel one-dimensional FFTs on dual Pentium III PC SMP cluster (8 Nodes, 16 CPUs).

Pentium III 1 GHz PC SMP クラスタでは提案するブロック nine-step FFT に基づく並列一次元 FFT において，$N = 2^{26}$ の場合に 1.3 GFLOPS を超える性能が得られていることが分かる．

表 3 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 3 から分かるように，今回評価に用いた dual Pentium III PC SMP クラスタでは，全対全通信が並列一次元 FFT の実行時間のうち，大きな割合を占めている．したがって，全対全通信を 1 回にする手法は，ブロック化と並んで性能を向上させるうえでは有効な手法であることが分かる．

図 3 に，ブロックサイズ NB を変化させた場合の dual Pentium III PC SMP クラスタにおける，提案するブロック nine-step FFT に基づく並列一次元 FFT の性能を 8 ノード，16 CPU の場合について示す．図 3 において，$N = 2^{21}$ および $N = 2^{22}$ の場合に性能が低下しているが，これは $N = 2^{21}$ および $N = 2^{22}$ の場合だけ通信性能が極端に悪くなっているのが原因である．図 3 から，N を変化させた場合，最適なブロックサイズ NB が異なっていることが分かる．

8. ま と め

本論文では，PC 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した．提案するブロック nine-step FFT に基づく並列一次元 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，PC 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 に基づく並列一次元 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，従来の並列一次元 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，並列一次元 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 を超える性能を得ることができた．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金奨励研究（A）（課題番号 12780190）および IPA（情報処理振興事業協会）の平成 12〜13 年度未踏ソフトウェア創造事業の支援を受けた．

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) Johnsson, S.L. and Krawitz, R.L.: Cooley-Tukey FFT on the Connection Machine, *Parallel Computing*, Vol.18, pp.1201–1221 (1992).
4) Van Loan, C.: *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, PA (1992).
5) 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).

6) Hegland, M.: Real and Complex Fast Fourier Transforms on the Fujitsu VPP 500, *Parallel Computing*, Vol.22, pp.539–553 (1996).
7) Frigo, M. and Johnson, S.G.: The Fastest Fourier Transform in the West, Technical Report MIT-LCS-TR-728, MIT Laboratory for Computer Science (1997).
8) Edelman, A., McCorquodale, P. and Toledo, S.: The Future Fast Fourier Transform?, *SIAM J. Sci. Comput.*, Vol.20, pp.1094–1114 (1999).
9) Takahashi, D.: High-Performance Parallel FFT Algorithms for the HITACHI SR8000, *Proc. 4th International Conference/Exhibition on High Performance Computing in Asia-Pacific Region* (*HPC-Asia 2000*), pp.192–199 (2000).
10) Bailey, D.H.: FFTs in External or Hierarchical Memory, *The Journal of Supercomputing*, Vol.4, pp.23–35 (1990).
11) 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).
12) Takahashi, D.: A Blocking Algorithm for Parallel 1-D FFT on Shared-Memory Parallel Computers, *Proc. 6th International Conference on Applied Parallel Computing* (*PARA 2002*), Lecture Notes in Computer Science, Vol.2367, pp.380–389, Springer-Verlag (2002).
13) Greengard, L. and Gropp, W.D.: A Parallel Version of the Fast Multipole Method, *Comput. Math. Applic.*, Vol.20, pp.63–71 (1990).
14) Katzenelson, J.: Computational Structure of the N-Body Problem, *SIAM J. Sci. Stat. Comput.*, Vol.10, pp.787–815 (1989).
15) Swarztrauber, P.N.: FFT Algorithms for Vector Computers, *Parallel Computing*, Vol.1, pp.45–63 (1984).
16) Bergland, G.D.: A Fast Fourier Transform Algorithm Using Base 8 Iterations, *Math. Comput.*, Vol.22, pp.275–279 (1968).
17) Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T. and Ishikawa, Y.: High Performance Communication using a Commodity Network for Cluster Systems, *Proc. 9th International Symposium on High Performance Distributed Computing* (*HPDC-9*), pp.139–146 (2000).

(平成 14 年 1 月 24 日受付)
(平成 14 年 5 月 12 日採録)

高橋　大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

朴　　泰祐（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，日本応用数理学会，IEEE 各会員．

佐藤　三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学電子・情報工学系教授．同大学計算物理学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．日本応用数理学会会員．