

# 分散メモリ型並列計算機による 2, 3, 5 基底一次元 FFT の実現と評価

高橋 大介<sup>†</sup> 金田 康正<sup>†</sup>

本論文では、2, 3, 5 基底の一次元 FFT を分散メモリ型並列計算機において実現し評価した結果について述べる。FFT アルゴリズムは行列の分解に帰着する。この行列分解の考え方を 2, 3, 5 基底の一次元 FFT に適用して並列複素 FFT アルゴリズムを構成した。さらに、MIMD 型分散メモリ型並列計算機 HITACHI SR2201 および IBM SP2 上に実現し、実機による測定結果に基づき理論的解析との適合性を検証するとともに、FFT の問題サイズを与えたとき、最小の実行時間を与えるプロセッサ数を求めることができることを示した。

## Implementation and Evaluation of Radix-2, 3 and 5 1-D FFT on Distributed Memory Parallel Computers

DAISUKE TAKAHASHI<sup>†</sup> and YASUMASA KANADA<sup>†</sup>

This paper discusses how the 1-D fast Fourier transform (FFT) with radix-2, 3 and 5 was implemented and the evaluation on distributed memory parallel computers. Our 1-D parallel complex FFT algorithm with radix-2, 3 and 5 is derived from the matrix factorization. Experimental results on distributed memory parallel computers, HITACHI SR2201 and IBM SP2, are given with confirmation to the theoretical analysis. Formula which gives optimal number of processing elements to the minimum execution time for a given FFT problem size is also analyzed.

### 1. はじめに

高速フーリエ変換 (fast Fourier transform, FFT)<sup>1)</sup> 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、並列計算機における FFT アルゴリズム<sup>2)~4)</sup> が多く提案されている。

逐次計算機やベクトル計算機では、データ数  $N$  が  $N = 2^p$  や  $N = 2^p 3^q 5^r$  と表される場合についての FFT アルゴリズム<sup>5)~7)</sup> が提案されており、実際に数値計算ライブラリとして提供されることが多い。

しかし分散メモリ型並列計算機では、 $N = 2^p$  の並列一次元 FFT アルゴリズムは提案されているが、 $N = 2^p 3^q 5^r$  の並列一次元 FFT に関する実現や評価は十分に行われていないのが現状である。

そこで本論文では、分散メモリ型並列計算機において 2, 3, 5 基底の一次元複素 FFT を実現し、その性能を評価した結果を述べる。なお、特に断わらない限

り本論文で取り扱う FFT は複素 FFT を意味することとする。

FFT アルゴリズムは行列の分解に帰着できることが知られている。この行列分解の考え方を 2, 3, 5 基底の FFT に適用することで、並列 FFT アルゴリズムが構成できることを示す。この並列 FFT アルゴリズムを MIMD 型分散メモリ型並列計算機 HITACHI SR2201 および IBM SP2 上に実現し、性能評価を行う。また並列計算機の要素プロセッサのアーキテクチャの違いによって、適合する並列 FFT アルゴリズムが異なることを示す。さらに実行時間の理論的な解析を行い、実機による実行時間と比較し、これがほぼ一致すること、そしてその理論的な解析から処理すべき FFT の問題サイズを与えたとき、最小の実行時間を与えるプロセッサ数を求めることができることも示す。

以下、2 章で高速フーリエ変換について、3 章で並列 FFT アルゴリズムについて、4 章で本論文で示すアルゴリズムの性能評価結果を示す。最後の 5 章はまとめである。

<sup>†</sup> 東京大学大型計算機センター

Computer Centre, University of Tokyo

## 2. 高速フーリエ変換

FFTは、離散フーリエ変換 (discrete Fourier transform, 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および逆DFTは次式で定義される。

$$F_j(x) = \sum_{k=0}^{n-1} x_k e^{-2\pi ijk/n}, \quad 0 \leq j < n \quad (1)$$

$$F_j^{-1}(x) = \frac{1}{n} \sum_{k=0}^{n-1} x_k e^{2\pi ijk/n}, \quad 0 \leq j < n \quad (2)$$

$n$  が  $n = lm$  と分解できるものとする、式 (1), (2) における  $j$  および  $k$  は、 $j = pm + q$ ,  $k = rl + s$  と書くことができる。ここで、 $p, s$  は  $0 \sim l-1$  の値をとり、 $q, r$  は  $0 \sim m-1$  の値をとるものとする。そうすると、式 (1) は次のように書くことができる。

$$\begin{aligned} F_{pm+q}(x) &= \sum_{s=0}^{l-1} \sum_{r=0}^{m-1} x_{rl+s} e^{-2\pi i(pm+q)(rl+s)/n} \\ &= \sum_{s=0}^{l-1} \left[ \sum_{r=0}^{m-1} (x_{rl+s} e^{-2\pi iqs/n}) e^{-2\pi iqr/m} \right] e^{-2\pi ips/l}, \end{aligned}$$

( $0 \leq p < l, 0 \leq q < m$ )

上式は、 $n$  点 DFT が  $l$  点 DFT と  $m$  点 DFT に分解されることを示している。

これから、たとえば  $n = n_1 n_2$  と分解されるとき、次に示すような “four step” FFT アルゴリズムとして知られている方法が導かれる<sup>8),9)</sup>。

1.  $n_1$  組の  $n_2$  点 FFT を  $n_1 \times n_2$  の複素数行列に対して行う。
2.  $n_1 \times n_2$  行列  $A_{jk}$  を考え、それに  $e^{-2\pi ijk/n}$  を掛ける。
3.  $n_1 \times n_2$  行列を  $n_2 \times n_1$  行列に転置する。
4.  $n_2$  組の  $n_1$  点 FFT を  $n_2 \times n_1$  の複素数行列に対して行う。

この “four step” FFT アルゴリズムの特徴は、

- $n_1 = n_2 \equiv \sqrt{n}$  とした場合、最内側のループ長が  $\sqrt{n}$  と固定にできるので、ベクトルプロセッサに適している。
- 行列の転置が 1 回で済む。
- 各ステップにおいて行方向の FFT を行っているため、メモリ参照の局所性が低く、キャッシュを前提とした RISC プロセッサには適さない。

といった点である。

また “four step” FFT の変形として、“six step” FFT アルゴリズムとして知られている以下に示す方法も導かれている<sup>8),9)</sup>。

1.  $n_1 \times n_2$  の複素数行列を  $n_2 \times n_1$  行列に転置する。
2.  $n_1$  組の  $n_2$  点 FFT を  $n_2 \times n_1$  行列に対して行う。
3.  $n_2 \times n_1$  行列  $A_{jk}$  に  $e^{-2\pi ijk/n}$  を掛ける。
4.  $n_2 \times n_1$  行列を  $n_1 \times n_2$  行列に転置する。
5.  $n_2$  組の  $n_1$  点 FFT を  $n_1 \times n_2$  行列に対して行う。
6.  $n_1 \times n_2$  行列を  $n_2 \times n_1$  行列に転置する。

この “six step” FFT アルゴリズムの特徴は、

- 各ステップにおいて列方向の FFT しか行わないので、メモリ参照の局所性が高く、キャッシュを前提とした RISC プロセッサに適している。
- 行列の転置が 3 回必要になる。

といった点である。

## 3. 並列 FFT アルゴリズム

### 3.1 アルゴリズム (1)

並列 FFT アルゴリズムを考えるにあたって “six step” FFT の考え方を適用する。

一次元 FFT はデータ数  $N$  を合成数で表すと、多次元の FFT として表現できる<sup>3)</sup>。データの長さを  $N = N_1 \cdot N_2$ ,  $P$  をプロセッサ数とした場合、次の手順で FFT を計算することができる<sup>3)</sup> (以下、アルゴリズム (1) と呼ぶ)。

1.  $N_1 \times N_2$  の入力データ (行列と考える) に対して、各プロセッサでは、 $N_1 \cdot (N_2/P)$  のデータを分散して持つ。
2.  $N_1 \times N_2$  行列を  $N_2 \times N_1$  行列に転置する。
3.  $N_1/P$  組の  $N_2$  点 FFT を  $N_2 \times N_1$  行列に対して行う。
4.  $N_2 \times N_1$  行列  $A_{jk}$  に、 $\exp(-2\pi ijk/N)$  を掛ける。
5.  $N_2 \times N_1$  行列を  $N_1 \times N_2$  行列に転置する。
6.  $N_2/P$  組の  $N_1$  点 FFT を  $N_1 \times N_2$  の行列に対して行う。
7.  $N_1 \times N_2$  行列を  $N_2 \times N_1$  行列に転置する。

このアルゴリズム (1) は、上記ステップ 3, 6 において  $N_1 = N_2 = \sqrt{N}$  としたとき、独立な  $\sqrt{N}$  点 FFT が  $\sqrt{N}/P$  回繰り返される形になるのが特徴である。

上記ステップ 2, 5, 7 の各部分で行列の転置が行われるが、これは全対全通信を必要とする。上に示したアルゴリズムはデータの分散をブロック分割にした場合のアルゴリズムであるが、サイクリック分割にした場合には最初から計算すべきデータが各プロセッサ内にあるため、ステップ 2 の部分のデータ交換が不要になる。したがって、ブロック分割では 3 回必要なプロセッサ間のデータ交換がサイクリック分割では 2 回で済むので、通信時間を  $2/3$  に減少させることができる。なお、このアルゴリズムでは  $N_1$  および  $N_2$  が

ロセッサ数  $P$  で割り切れる場合には各プロセッサの演算量は均一となることに注意しておく。

並列計算機の要素プロセッサが最新の RISC プロセッサであるような場合 (例: IBM SP2) は, 十分大きなデータ数  $N$ , たとえば  $N = 2^{20}$  と仮定しても,  $\sqrt{N} = 2^{10}$  となり, ワーキングセットサイズがキャッシュサイズ以下の大きさとなるので, 高速に実行できると予想される。

次に並列計算機の要素プロセッサがベクトルプロセッサであるような場合 (例: HITACHI SR2201) について考える。

ベクトルプロセッサで  $n$  点 FFT を計算するとき, 最内側のループ長は,  $n/2, n/4, \dots, 1$  と次第に短くなっていくので, ループ長をできるだけ長くするために, ループを途中で入れ換えるようなコーディングを行う。こうすることによって平均ループ長は  $\sqrt{n}$  のオーダーになる。

さてこのアルゴリズムでは各プロセッサは  $N_1 = N_2 = \sqrt{N}$  点 FFT を繰り返し実行するので, 最内側のループを入れ換えたとしても平均ループ長は  $N^{1/4}$  のオーダーになる。したがって十分大きな  $N$ , たとえば  $N = 2^{20}$  としても, 平均ループ長は 32 となってしまう, ベクトルプロセッサの演算パイプラインの立上りにもよるが, ループの長さとしては不十分となる。

今回評価に用いた HITACHI SR2201 の要素プロセッサはソフトウェアによる疑似ベクトル処理<sup>10)</sup>を行っているため, 図 1 から分かるように, 他のベクトルプロセッサに比べると演算パイプラインの立上りがきわめて良い。しかし, それでも飽和性能に達するループ長は 100 以上であるため, 前述の並列アルゴリズムはベクトル並列アーキテクチャの性能を十分に引き出しているとはいえない。

### 3.2 アルゴリズム (2)

ベクトル並列アーキテクチャの性能を十分に引き出すために, ループ長を長くする方法として, 各プロセッサ内で  $N/P$  点 FFT が実行できるようにすることを考える。

その方法として 1 章で述べた “four step” FFT の考え方を適用する。

データの長さを  $N = N_1 \cdot N_2 \cdot N_3$ ,  $P$  をプロセッサ数とした場合, 次の手順で FFT を計算することができる (以下アルゴリズム (2) と呼ぶ)。

ここで  $W_N = \exp(-2\pi i/N)$  とする。

1.  $N_1 \cdot N_2 \cdot N_3$  の入力データに対して, 各プロセッサでは  $N_1 \cdot N_2 \cdot (N_3/P)$  のデータを分散して持つ。
2.  $N_1 \cdot N_2 \cdot (N_3/P)$  のデータをプロセッサ間で交

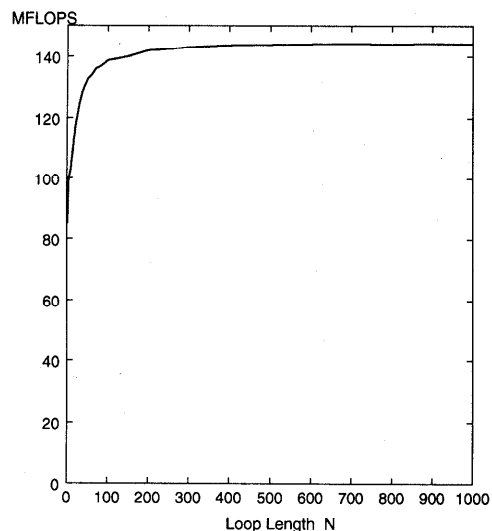


図 1 FFT カーネル (基数 4) の性能 (HITACHI SR2201 1 PE)

Fig. 1 Performance of FFT Kernel (radix-4) (HITACHI SR2201 1 PE).

換して  $N_3 \cdot N_1 \cdot (N_2/P)$  のデータを持つようにする。

3.  $N_1 \cdot (N_2/P)$  組の  $N_3$  点 FFT を行う。
4. 結果のデータを  $X(j_1, j_2, k_3)$  とするとき, それらに  $W_N^{(j_2 N_1 + j_1) k_3}$  を掛ける。
5.  $N_3 \cdot N_1 \cdot (N_2/P)$  のデータをプロセッサ間で交換して  $N_2 \cdot N_1 \cdot (N_3/P)$  のデータを持つようにする。
6.  $N_1 \cdot (N_3/P)$  組の  $N_2$  点 FFT を行う。
7. 結果のデータを  $X(j_1, k_2, k_3)$  とするとき, それらに  $W_{N_1 N_2}^{j_1 k_2}$  を掛ける。
8. 各プロセッサ内で  $N_1 \times N_2$  行列を  $N_2 \times N_1$  行列に転置する。
9.  $N_2 \cdot (N_3/P)$  組の  $N_1$  点 FFT を行う。
10.  $N_3 \cdot N_1 \cdot (N_2/P)$  のデータをプロセッサ間で交換して  $N_1 \cdot N_2 \cdot (N_3/P)$  のデータを持つようにする。

上記ステップ 2 の部分で,  $N_1 \cdot N_2 \cdot (N_3/P)$  のデータをプロセッサ間で交換しているのは, 計算すべきデータを各プロセッサ内に持ってくる必要があるためである。ステップ 5, 10 の各部分についても同様である。これは全対全通信となる。

3.1 節で示したアルゴリズム (1) と同様に, このアルゴリズムでもサイクリック分割の方が全対全通信が 2 回で済む。通信以外の演算量は, ブロック分割とサイクリック分割では同じとなり, サイクリック分割が優れていることから, 本論文では以降, サイクリック

分割による FFT アルゴリズムについてのみ考えることにする。

なお、このアルゴリズムでは  $N_2$  および  $N_3$  がプロセッサ数  $P$  で割り切れる際には各プロセッサの演算量が均一となる。そのためには、 $N = N_1 \cdot N_2 \cdot N_3$  であるので、 $N$  が  $P^2$  で割り切れるように  $N$  および  $P$  を選択する必要があることに注意しておく。

### 3.3 各アルゴリズムのプロセッサアーキテクチャへの適合性

これらのアルゴリズム (1), (2) において、要素プロセッサがベクトルプロセッサである場合について適しているかどうかを判断するために、各プロセッサ内の演算の最内側の平均ループ長を検討する。

アルゴリズム (2) において、簡単のため  $N_1 = N_2 = N_3 = N^{1/3}$  とし、ステップ 3, 6, 9 のそれぞれで “four step” FFT の考え方を適用すると、最内側の平均ループ長は  $N^{2/3}/P$  となる。最内側の平均ループ長が  $N^{1/4}$  となるアルゴリズム (1) と比較すると、 $N^{2/3}/P > N^{1/4}$  となる条件は  $N^{5/12} > P$  となる。この領域については上記のアルゴリズム (2) が最内側の平均ループ長が長くできることになるが、並列 FFT を実行する際はほとんどこの条件を満たすと考えられるので、アルゴリズム (2) は要素プロセッサがベクトルプロセッサである場合については有利になることが分かる。

次に、アルゴリズム (1), (2) において、要素プロセッサがキャッシュメモリを前提にした RISC プロセッサである場合について適しているかどうかを判断するために、各プロセッサ内の演算のワーキングセットサイズを検討する。

アルゴリズム (2) では各プロセッサが一度に  $N/P$  点 FFT を実行することと等価になるので、ワーキングセットサイズは  $N/P$  となる。一方、アルゴリズム (1) ではワーキングセットサイズが  $\sqrt{N}$  となる。これから、アルゴリズム (1) のワーキングセットサイズがアルゴリズム (2) より小さくなる条件は、 $\sqrt{N} < N/P$  より  $\sqrt{N} > P$  であり、並列 FFT を実行する際はほとんどこの条件を満たすと考えてよい。

したがって、要素プロセッサがキャッシュメモリを前提にした RISC プロセッサである場合についてはアルゴリズム (1) が有利になることが分かる。

### 3.4 プロセッサ内の逐次 FFT アルゴリズム

基数 2, 3, 5 の FFT アルゴリズムとしては、Singleton<sup>5)</sup>によるものや、Temperton<sup>6)</sup>による Mixed-Radix FFT が知られている。

今回は、フーリエ変換すべきデータ数  $N = 2^p 3^q 5^r$

において “four step” FFT アルゴリズムを適用した。

まず  $2^p 3^q$  点 FFT を “four step” FFT アルゴリズムにより計算し、その結果と  $5^r$  点 FFT の結果を “four step” FFT アルゴリズムで統合する。

基数 2 の FFT アルゴリズムとしては Stockham のアルゴリズム<sup>11)</sup>を用いた。この Stockham のアルゴリズムは、Cooley-Tukey のアルゴリズム<sup>1)</sup>に比べて入力と出力が重ね書きできないために、メモリは 2 倍必要となる。しかし、最内側のループの配列アクセスが連続的になるので、キャッシュのヒット率が良くなるという特徴がある。またベクトルプロセッサにも適しているアルゴリズムとしても知られている<sup>11)</sup>。

基数 3 の FFT アルゴリズムとしては、Stockham のアルゴリズムを拡張し、Rader の Small- $n$  アルゴリズム<sup>12)</sup>を適用した。

基数 5 の FFT アルゴリズムは、基数 3 と同様に Stockham のアルゴリズムを拡張して、Rader の Small- $n$  アルゴリズムを適用した。Small- $n$  アルゴリズムとしては、ほかに Winograd による WFTA (Winograd Fourier Transform Algorithm)<sup>13)</sup>がある。WFTA の Small- $n$  アルゴリズムによる基数 5 の変換は、Rader のアルゴリズムに比べると乗算は 2 回減るが、加算が 2 回増えてしまう。並列計算機の各プロセッサにおいて、乗算のサイクル数が加算に比べて多い場合には WFTA の Small- $n$  アルゴリズムが有効であるが、今回評価に使用した HITACHI SR2201, IBM SP2 のそれぞれの要素プロセッサは積和計算を 1 サイクルで実行できるアーキテクチャであるため、所要サイクル数は加算の回数で決まる。したがってこのような場合については、Rader の Small- $n$  アルゴリズムが高速に実行できることが分かる。

また基数 2 の FFT においては、演算回数の少ない基数 4, 8 の FFT を部分的に適用することにより効率を高くすることができる。同様に、基数 6 ( $= 2 \times 3$ ) や基数 9 ( $= 3 \times 3$ ) などでも適用可能であり、演算回数をより少なくすることができるが、基数を大きくするに従ってアルゴリズムが複雑となるので、基数 2, 3, 4, 5, 6 の組合せが実用上限度とされている<sup>6)</sup>。

今回は基数 2, 3, 4, 5 の組合せで実現および評価を行った。

### 3.5 基数 2 の FFT

基数 2 における Stockham のアルゴリズムについて説明する。

$n = 2lm$  とする。ここで  $l$  および  $m$  は 2 のべきであるものとする。 $l$  の初期値は  $n/2$  とし、反復ごとに 2 で割っていく。 $m$  の初期値は 1 とし、反復ごとに 2

倍していく。配列  $X$  は入力データであり、 $Y$  は出力データである。実際にインプリメントするときには、反復において、 $X$  から  $Y$ 、 $Y$  から  $X$  に出力されるようにすると、メモリコピーを減らすことができる。ここで  $\omega_p = e^{-2\pi i/p}$  である。

$$\begin{aligned}c_0 &= X(k + jm) \\c_1 &= X(k + jm + lm)\end{aligned}$$

$$\begin{aligned}Y(k + 2jm) &= c_0 + c_1 \\Y(k + 2jm + m) &= \omega_{2l}^j (c_0 - c_1) \\0 \leq j < l \quad 0 \leq k < m\end{aligned}$$

### 3.6 基底 3 の FFT

基底 3 の場合に拡張した Stockham のアルゴリズムは次のようになる。

$n = 3lm$  とする。ここで  $l$  および  $m$  は 3 のべきであるものとする。 $l$  の初期値は  $n/3$  とし、反復ごとに 3 で割っていく。 $m$  の初期値は 1 とし、反復ごとに 3 倍していく。ここで  $\omega_p = e^{-2\pi i/p}$  である。

$$\begin{aligned}c_0 &= X(k + jm) \\c_1 &= X(k + jm + lm) \\c_2 &= X(k + jm + 2lm) \\d_0 &= c_1 + c_2 \\d_1 &= c_0 - \frac{1}{2}d_0 \\d_2 &= -i \left( \sin \frac{\pi}{3} \right) (c_1 - c_2)\end{aligned}$$

$$\begin{aligned}Y(k + 3jm) &= c_0 + d_0 \\Y(k + 3jm + m) &= \omega_{3l}^j (d_1 + d_2) \\Y(k + 3jm + 2m) &= \omega_{3l}^{2j} (d_1 - d_2) \\0 \leq j < l \quad 0 \leq k < m\end{aligned}$$

### 3.7 基底 4 の FFT

基底 4 の場合に拡張した Stockham のアルゴリズムは次のようになる。

$n = 4lm$  とする。ここで  $l$  および  $m$  は 4 のべきであるものとする。 $l$  の初期値は  $n/4$  とし、反復ごとに 4 で割っていく。 $m$  の初期値は 1 とし、反復ごとに 3 倍していく。ここで  $\omega_p = e^{-2\pi i/p}$  である。

$$\begin{aligned}c_0 &= X(k + jm) \\c_1 &= X(k + jm + lm) \\c_2 &= X(k + jm + 2lm) \\c_3 &= X(k + jm + 3lm) \\d_0 &= c_0 + c_2 \\d_1 &= c_0 - c_2 \\d_2 &= c_1 + c_3 \\d_3 &= -i(c_1 - c_3)\end{aligned}$$

$$\begin{aligned}Y(k + 4jm) &= d_0 + d_2 \\Y(k + 4jm + m) &= \omega_{4l}^j (d_1 + d_3) \\Y(k + 4jm + 2m) &= \omega_{4l}^{2j} (d_0 - d_2)\end{aligned}$$

$$\begin{aligned}Y(k + 4jm + 3m) &= \omega_{4l}^{3j} (d_1 - d_3) \\0 \leq j < l \quad 0 \leq k < m\end{aligned}$$

### 3.8 基底 5 の FFT

基底 5 の場合に拡張した Stockham のアルゴリズムは次のようになる。

$n = 5lm$  とする。ここで  $l$  および  $m$  は 5 のべきであるものとする。 $l$  の初期値は  $n/5$  とし、反復ごとに 5 で割っていく。 $m$  の初期値は 1 とし、反復ごとに 5 倍していく。ここで  $\omega_p = e^{-2\pi i/p}$  である。

$$\begin{aligned}c_0 &= X(k + jm) \\c_1 &= X(k + jm + lm) \\c_2 &= X(k + jm + 2lm) \\c_3 &= X(k + jm + 3lm) \\c_4 &= X(k + jm + 4lm) \\d_0 &= c_1 + c_4 \\d_1 &= c_2 + c_3 \\d_2 &= \left( \sin \frac{2\pi}{5} \right) (c_1 - c_4) \\d_3 &= \left( \sin \frac{2\pi}{5} \right) (c_2 - c_3) \\d_4 &= d_0 + d_1 \\d_5 &= \frac{\sqrt{5}}{4} (d_0 - d_1) \\d_6 &= c_0 - \frac{1}{4}d_4 \\d_7 &= d_6 + d_5 \\d_8 &= d_6 - d_5 \\d_9 &= -i \left( d_2 + \frac{\sin(\pi/5)}{\sin(2\pi/5)} d_3 \right) \\d_{10} &= -i \left( \frac{\sin(\pi/5)}{\sin(2\pi/5)} d_2 - d_3 \right)\end{aligned}$$

$$Y(k + 5jm) = c_0 + d_4$$

$$\begin{aligned}Y(k + 5jm + m) &= \omega_{5l}^j (d_7 + d_9) \\Y(k + 5jm + 2m) &= \omega_{5l}^{2j} (d_8 + d_{10}) \\Y(k + 5jm + 3m) &= \omega_{5l}^{3j} (d_8 - d_{10}) \\Y(k + 5jm + 4m) &= \omega_{5l}^{4j} (d_7 - d_9)\end{aligned}$$

$$0 \leq j < l \quad 0 \leq k < m$$

### 3.9 演算回数

基底 2, 3, 5 の各 FFT の最内側ループ中における実数演算回数は、表 1 のようになる。今回の実現においては、データ数が 2 のべきで表される場合については基底 2 と基底 4 の FFT を組み合わせているが、演算回数としては基底 2 の FFT を実行したのとして計算している。

ここで、基底 2, 3, 5 の各 FFT における 1 点あたりの演算回数は、表 1 に示されている実数演算回数を基底で割った値となるので、 $N = 2^p 3^q 5^r$  とした場合の FFT における加算の回数  $A(N)$  と乗算の回数

表1 基数2, 3, 5のFFTの最内側ループ中における実数演算回数<sup>6)</sup>

Table 1 Real arithmetic operation counts at the innermost loop on the radix-2, 3 and 5 FFT<sup>6)</sup>.

基数	加算	乗算
2	6	4
3	16	12
5	40	28

$M(N)$  は,

$$A(N) = 2N(1.5p + 2.667q + 4r - 1) + 2$$

$$M(N) = 2N(p + 2q + 2.8r - 2) + 4$$

となる<sup>6)</sup>。したがって総計算量は、 $A(N) + M(N) = 2N(2.5p + 4.667q + 6.8r - 3) + 6$  となる。

### 3.10 演算時間

プロセッサ単体の演算性能を  $S_{comp}$  (MFLOPS)、プロセッサ数を  $P$  とすると、演算時間  $T_{comp}$  (sec) は、

$$T_{comp} = \frac{2N(2.5p + 4.667q + 6.8r - 3) + 6}{P \cdot (S_{comp} \times 10^6)} \quad (3)$$

となる。ただし、 $S_{comp}$  は基数2の  $N$  点FFTの理論演算量  $5N \log_2 N$  から算出した値とする。

### 3.11 通信時間

プロセッサ間のデータの交換において全対全通信が発生する。

この通信では、各プロセッサは残りのすべてのプロセッサに対し、 $N/P^2$  個の複素数倍精度配列、つまり  $16 \times (N/P^2)$  バイトのデータを送信する。

したがって、通信時間  $T_{comm}$  (sec) は、 $M$  を通信回数、 $P$  をプロセッサ数、 $L$  をレイテンシ ( $\mu$ s)、 $N$  をデータ数、 $B$  をデータ転送速度 (MB/s) とすると、

$$T_{comm} = M \times (P - 1) \times \left( L \times 10^{-6} + \frac{16 \times (N/P^2)}{B \times 10^6} \right) \quad (4)$$

と表される。

ブロック分割をした場合、プロセッサ間でデータを3回交換することから  $M = 3$  であり、サイクリック分割をした場合は  $M = 2$  となる。

### 3.12 コピー時間

各プロセッサ内における行列の転置および“four step”FFTアルゴリズムのステップ2で行っているような、ひねり係数の乗算において、それぞれ  $16 \times (N/P)$  バイトのデータをコピーする。コピー性能を  $S_{copy}$  (MB/s) とすると、コピー時間  $T_{copy}$  (sec) は、

$$T_{copy} = 2 \times \frac{16 \times N}{P \cdot (S_{copy} \times 10^6)} \quad (5)$$

となる。

### 3.13 合計時間

全実行時間  $T_{total}$  (sec) は、

$$T_{total} = T_{comp} + T_{comm} + T_{copy} \quad (6)$$

となる。

ここで、サイクリック分割の場合の全実行時間  $T_{total}$  を最小にする目安となるプロセッサ数  $P_{opt}$  を、式(6)の極値として求めると、

$$P_{opt} \approx \sqrt{\frac{N}{L} \left( \frac{2.5p + 4.667q + 6.8r - 3}{S_{comp}} + \frac{16}{B} + \frac{16}{S_{copy}} \right)} \quad (7)$$

となる。ここで、データ数  $N$  が  $P_{opt}$  で整除できない場合には、データ分散が不均一になり、各プロセッサの負荷が均等でなくなるため、必ずしも式(7)が全実行時間  $T_{total}$  を最小にするとは限らないことに注意しておく。

## 4. 並列FFTの性能評価

並列化の評価では、 $N = 2^p 3^q 5^r$  の  $p, q, r$  およびプロセッサ数  $P$  を変化させて複素順方向FFTを10回実行し、その平均の経過時間を測定することにより行った。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。

並列計算機として、MIMD型並列計算機HITACHI SR2201 (1024 PE, 総主記憶256 GB, 理論ピーク性能307.2 GFLOPS)、IBM SP2 thin-node (16 PE, 総主記憶4 GB, 理論ピーク性能4.3 GFLOPS) を用いた。

### 4.1 HITACHI SR2201による測定結果

HITACHI SR2201の通信ライブラリとして、メモリコピーの発生しないリモートDMA転送<sup>14)</sup>を用いた。プログラムはすべてFORTRANで記述した。コンパイラは日立の最適化FORTRAN77 V02-03を用い、最適化オプションとして-W0,'pvec(pvfunc(1)),opt(o(s), fold(2),prefetch(1),rapidcall(1),ischedule(3), reroll(1),scope(1),split(1),uinline(2))'を指定した。

アルゴリズム(1),(2)のそれぞれの実行時間を測定した結果を図2に示す。図2から、HITACHI SR2201ではアルゴリズム(2)がアルゴリズム(1)に比べて高い性能が発揮できていることが分かる。これは、アルゴリズム(2)では最内側のループ長を長くできるので、単体プロセッサで高い性能を発揮できているためである。

アルゴリズム(1),(2)ともに、 $N = 2^{20}$ 点FFTにおいては128台まではプロセッサ数の増加にともない実行時間が短縮されているが、256台以上になると逆

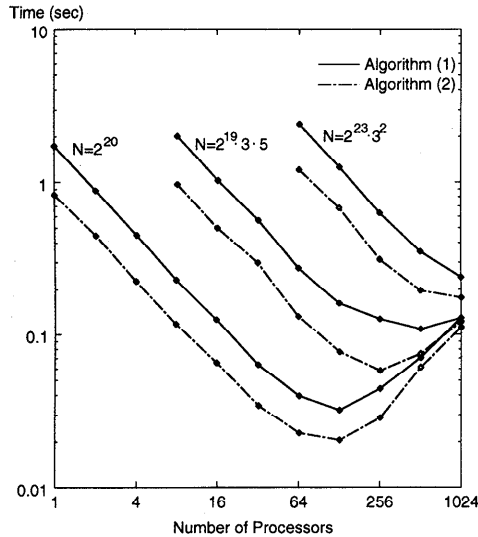


図2 並列FFTの実行時間 (HITACHI SR2201)  
Fig. 2 Execution time of parallel FFT  
(HITACHI SR2201).

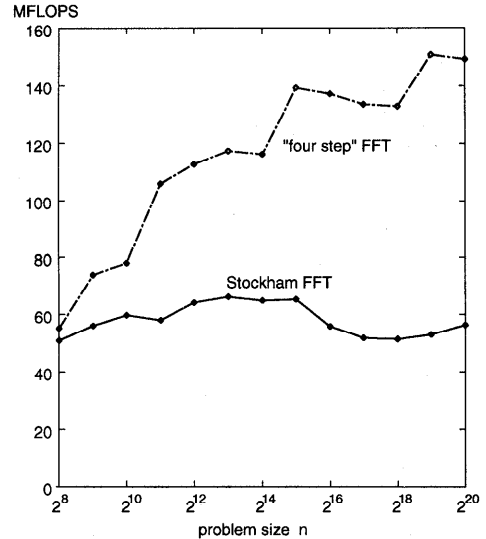


図3 単体プロセッサにおけるFFTの性能  
(HITACHI SR2201 1 PE)  
Fig. 3 Performance of FFT on single processor  
(HITACHI SR2201 1 PE).

に実行時間が増えてしまっている。これは式(4)からも分かるように、プロセッサ数が増加するに従って一度に送るデータ量がプロセッサ数の自乗に反比例して小さくなるため、通信の立ち上がり時間が無視できなくなってくるためと考えられる。

また式(3)~(6)で与えられる実行時間の予測値のパラメータとして、通信に関しては実測値を一次式で近似して得た、レイテンシ  $L = 41.9$  ( $\mu\text{s}$ ) とデータ転送速度  $B = 240.8$  (MB/s) を、コピー性能については実測値より得た  $373.8$  (MB/s) を使用した。演算性能については単体プロセッサでFFTを実行した場合の実測値(図3)を用い、式(3)からプロセッサ単体の演算性能  $S_{comp}$  (MFLOPS) を求めた。具体的には、アルゴリズム(1)では各プロセッサで  $\sqrt{N}$  点FFTが  $\sqrt{N}/P$  回繰り返されることから、単体プロセッサの演算性能は、 $\sqrt{N}$  点のStockham FFTを実行した場合の性能とし、アルゴリズム(2)では各プロセッサで  $N/P$  点FFTを実行するのと等価になることから、単体プロセッサの演算性能は、 $N/P$  点の“four step”FFTを実行した場合の性能とした。これらのパラメータからプロセッサ数を4096台まで増やした場合の予測値を求め、実測値と比較したものを図4と図5に示す。

ここで  $N = 2^{20}$  の場合において、演算性能はプロセッサ数  $P$  によらず一定で、アルゴリズム(1)では図3におけるStockham FFTのピーク演算性能として  $S_{comp} = 60$  (MFLOPS)、アルゴリズム(2)で

は図3における“four step”FFTのピーク演算性能として  $S_{comp} = 150$  (MFLOPS) と仮定すると、式(7)より実行時間が最小になる目安となるプロセッサ数  $P_{opt}$  はアルゴリズム(1)では約154台、アルゴリズム(2)ではアルゴリズム(1)より少ない約103台となる。この関係は式(7)より、 $S_{comp}$  が大きくなると  $P_{opt}$  は小さくなることから説明できる。またこれらの  $P_{opt}$  は、図2から得られる結果、すなわち128台とほぼ一致していることが分かる。

図4と図5で  $N$  が小さい場合の実測値と予測値は、実行時間が最小になるプロセッサ数は一致しているがそれ以上では、ずれが大きくなっている。これは、式(4)においてプロセッサ数  $P$  が多く、なおかつ  $N$  が小さくなっている場合にはレイテンシ  $L$  が通信時間に大きく影響してくるためである。今回の性能評価では、レイテンシ  $L$  が一定であると仮定したのが実測値と予測値のずれの原因と考えられるが、図4と図5から分かるように、異なる2つのアルゴリズムにおいても実行時間の実測値と予測値はほぼ同様な傾向を示している。これより式(3)~(7)を与える計算モデルは少なくとも実測値の範囲においては妥当であるといえる。

#### 4.2 IBM SP2による測定結果

IBM SP2の通信ライブラリとしてMPI<sup>15)</sup>を用いた。プログラムはすべてFORTRANで記述した。コンパイラはIBMのXL Fortran version 3.2を用い、

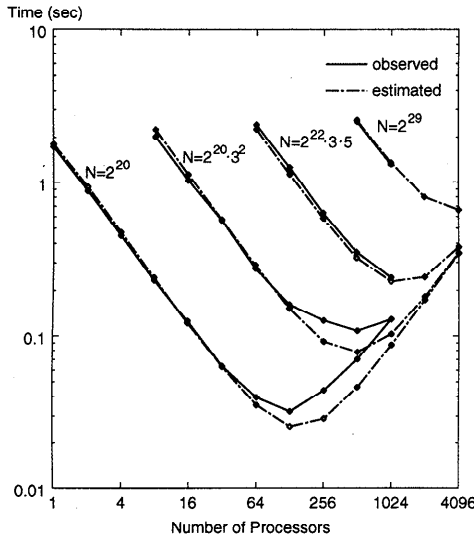


図4 並列FFTの予測時間と実行時間の比較 (アルゴリズム(1), HITACHI SR2201)

Fig. 4 Comparison between estimated time and execution time (Algorithm (1), HITACHI SR2201).

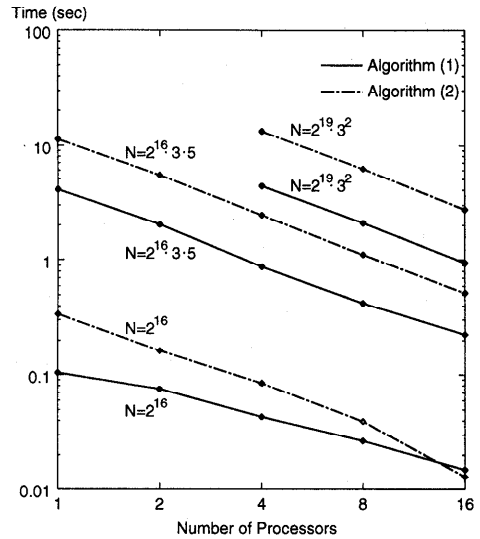


図6 並列FFTの実行時間 (IBM SP2)

Fig. 6 Execution time of parallel FFT (IBM SP2).

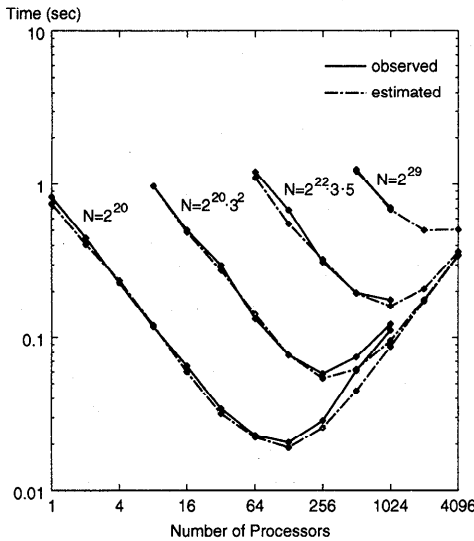


図5 並列FFTの予測時間と実行時間の比較 (アルゴリズム(2), HITACHI SR2201)

Fig. 5 Comparison between estimated time and execution time (Algorithm (2), HITACHI SR2201).

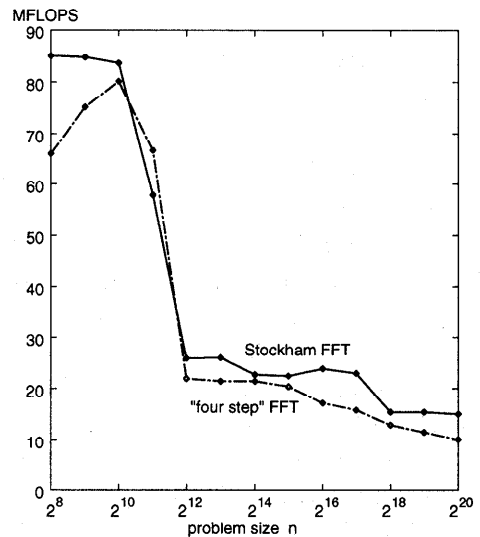


図7 単体プロセッサにおけるFFTの性能 (IBM SP2 1 PE)

Fig. 7 Performance of FFT on single processor (IBM SP2 1 PE).

最適化オプションとして-O3 -qarch=pwr2 -qhot -qtune=pwr2を指定した。アルゴリズム(1), (2)のそれぞれの実行時間を測定した結果を図6に示す。図6から、IBM SP2では、アルゴリズム(1)がアルゴリズム(2)に比べて高い性能が発揮できていることが分かる。これはアルゴリズム(1)ではワーキングセットサイズが小さくなっており、キャッシュのヒット率が高くなるからである。

また式(3)~(6)で与えられている実行時間の予測値のパラメータとして、通信に関しては実測値を一次式で近似して得た、レイテンシ  $L = 126.8 (\mu s)$  とデータ転送速度  $B = 22.1 (MB/s)$  を、コピー性能については実測値より  $85.0 (MB/s)$  を使用した。

演算性能についてはHITACHI SR2201の場合と同様に、単体プロセッサでFFTを実行した場合の実測値(図7)を用いた。これらのパラメータからプロセッサ数を256台まで増やした場合の予測値を求め、実測



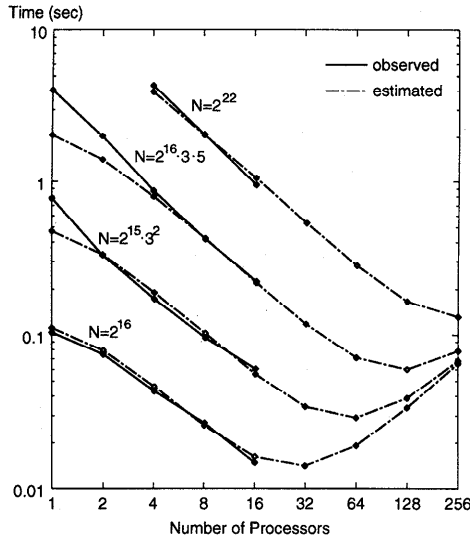


図 8 並列 FFT の予測時間と実行時間の比較 (アルゴリズム (1), IBM SP2)

Fig. 8 Comparison between estimated time and execution time (Algorithm (1), IBM SP2).

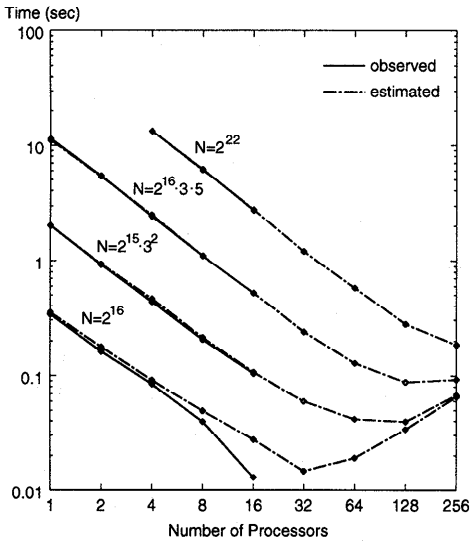


図 9 並列 FFT の予測時間と実行時間の比較 (アルゴリズム (2), IBM SP2)

Fig. 9 Comparison between estimated time and execution time (Algorithm (2), IBM SP2).

値と比較したものを図 8 と図 9 に示す。

図 8 においてプロセッサ数が 1~2 台の場合、 $N = 2^{15} \cdot 3^2$ ,  $N = 2^{16} \cdot 3 \cdot 5$  の場合に予測値と実測値のずれが大きくなっており、図 9 においては  $N = 2^{16}$ 、プロセッサ数が 16 台の場合に予測値と実測値のずれが大きくなっている。

これらの原因としては、単体プロセッサにおける FFT の性能と並列 FFT の性能を測定したときでは、

表 2 データ数の違いによる実行時間の変化 (単位 秒, 16PE)

Table 2 Execution time on various data size (in sec, 16 PE).

N	SR2201	SP2
$2^{16} \cdot 3 \cdot 5 = 983040$	0.0657	0.5556
$2^{20} = 1048576$	0.0652	0.5217
$2^{17} \cdot 3^2 = 1179648$	0.0723	0.6553
$2^{18} \cdot 3 \cdot 5 = 3932160$	0.2703	2.9121
$2^{22} = 4194304$	0.2546	2.7132
$2^{19} \cdot 3^2 = 4718592$	0.3088	3.0372
$2^{20} \cdot 3 \cdot 5 = 15728600$	1.0999	15.6788
$2^{24} = 16777216$	0.9932	15.0075
$2^{21} \cdot 3^2 = 18874368$	1.2940	17.2011

表 3 理論計算量  $n \log_2 n$  に対する実数演算量の比 (相対効率)<sup>5)</sup>

Table 3 Ratio of the real arithmetic operation counts to theoretical arithmetic complexity of  $n \log_2 n$ <sup>5)</sup>.

基数	相対効率
8	0.333
4	0.375
2	0.500
3	0.631
5	0.689
7	0.763

キャッシュの振舞いに違いが出てくるのが 1 つの要因と考えられるが、図 8 および図 9 から分かるように、IBM SP2 においても異なるアルゴリズムでの実行時間の実測値と予測値はほぼ同じような傾向を示している。このことから、式 (3)~(7) を与える計算モデルは少なくとも実測値の範囲においては妥当であるといえる。

### 4.3 2, 3, 5 基底を組み合わせさせた場合

HITACHI SR2201 および IBM SP2 でプロセッサ数を 16 とした場合について、アルゴリズム (2) をデータ数  $N = 2^p 3^q 5^r$  を変化させて実行した場合の順 FFT の実行時間を表 2 に示す。

実行時間が短くなるのはデータ数が 2 のべきだけからなる場合の FFT であることが表 2 から分かる。これは、表 3 で示すように、基数 3, 5 の FFT は基数 2 の FFT に比べて効率が悪くなっているためである。

このことから、基数 2, 3, 5 を組み合わせさせて FFT を計算する際には、2 のべきの部分なるべく多くして、効率の良い基数 2 の割合を高くすると、性能が向上することが分かる。

## 5. ま と め

本論文では、分散メモリ型並列計算機において 2, 3, 5 基底の FFT を実現し、評価した結果について述べた。また、並列計算機の要素プロセッサのアーキテクチャの違いによって適合する並列 FFT アルゴリズム

ムが異なることを示した。さらに実行時間の理論的な解析を行い、実機による実行時間と予測時間を比較し、両者がほぼ一致すること、そしてその理論的な解析から処理すべきFFTの問題サイズを与えたとき、最小の実行時間を与える目安となるプロセッサ数を求めることができることも示した。

特に、今回の評価においてはプロセッサ単体の演算性能として、基数2のFFTの性能を用いたが、2以外の基数を含む場合においても実測値と予測値がほぼ一致していることから、この計算モデルの有効性を示しているものと判断する。

今後の課題としては、実機による少ないパラメータによる実行性能予測精度の向上および、2, 3, 5基底の並列実数FFTの実現と評価があげられる。

### 参考文献

- 1) Cooley, J.W. and Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series, *Math. Comp.*, Vol.19, pp.297-301 (1965).
- 2) Swarztrauber, P.N.: Multiprocessor FFTs, *Parallel Computing*, Vol.5, pp.197-210 (1987).
- 3) 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).
- 4) Hegland, M.: Real and Complex Fast Fourier Transforms on the Fujitsu VPP 500, *Parallel Computing*, Vol.22, pp.539-553 (1996).
- 5) Singleton, R.C.: An algorithm for computing the mixed radix Fast Fourier Transform, *IEEE Trans. Audio Electroacoust.*, Vol.17, pp.93-103 (1969).
- 6) Temperton, C.: Self-Sorting Mixed-Radix Fast Fourier Transforms, *J. Comput. Phys.*, Vol.52, pp.1-23 (1983).
- 7) Agarwal, R.C. and Cooley, J.W.: Vectorized Mixed Radix Discrete Fourier Transform Algorithms, *Proc. IEEE*, Vol.75, pp.1283-1292 (1987).
- 8) Bailey, D.H.: FFTs in External or Hierarchical Memory, *The J. Supercomputing*, Vol.4, pp.23-35 (1990).
- 9) Van Loan, C.: *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, PA (1992).
- 10) Nakazawa, K., Nakamura, H., Imori, H. and

Kawabe, S.: Pseudo Vector Processor based on Register-Windowed Superscalar Pipeline, *Proc. Supercomputing '92*, pp.642-651 (1992).

- 11) Swarztrauber, P.N.: FFT Algorithms for Vector Computers, *Parallel Computing*, Vol.1, pp.45-63 (1984).
- 12) Rader, C.M.: Discrete Fourier transforms when the number of data samples is prime, *Proc. IEEE*, Vol.56, pp.1107-1108 (1968).
- 13) Winograd, S.: On computing the DFT, *Math. Comp.*, Vol.32, pp.175-199 (1978).
- 14) 日立製作所: HI-UX/MPP リモート DMA 転送使用の手引 6A20-3-021 (1996).
- 15) Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard, Version 1.1* (1995).

(平成 9 年 6 月 20 日受付)

(平成 10 年 1 月 16 日採録)



高橋 大介 (正会員)

1970 年生。1991 年呉工業高等専門学校電気工学科卒業。1993 年豊橋技術科学大学工学部情報工学課程卒業。1995 年同大学大学院工学研究科情報工学専攻修士課程修了。1997 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。並列数値計算アルゴリズムに関する研究に従事。日本応用数理学会会員。

金田 康正 (正会員)

1949 年生。1973 年東北大学理学部物理第二学科卒業。1978 年東京大学理学系研究科博士課程修了。理学博士。1978 年名古屋大学プラズマ研究所助手。1981 年東京大学大型計算機センター助教授を経て現在同教授。その間英国ケンブリッジ大学計算機研究所客員研究員、名古屋大学プラズマ研究所客員助教授、核融合科学研究所客員助教授。昭和 58 年度情報処理学会論文賞受賞。著書「 $\pi$  のはなし」(東京図書)、共著「アドバンスド・コンピューティング—21 世紀の科学技術基盤」(培風館)、編著「Trends in Supercomputing」(World Scientific)。日本応用数理学会、プラズマ・核融合学会、ACM、SIAM 各会員。研究テーマは「大規模数値計算」および「研究の研究」。