FFT の多次元化による並列処理の研究

東海大学 工学部 通信工学科

指導教員 清 水 尚彦 講師

30et3238　渡邉 岳彦

提出日 平成９年４月７日

Abstract

本論文では、１次元 FFT を３次元化することで、演算と通信のオーバーラップが可能となり、並列処理の高速化が実現できることを述べる。また、２次元化 FFT との比較により性能評価をおこなった。プログラムは、３次元化 FFT を C 言語で作成しメッセージ交換ライブラリーは、MPI(Message-Pssing interface) を使用し、NEC Cenju3(PE 数:128) で実行した。

目 次

Chapter 1

はじめに

離散 Fourier 変換 (discrete Fourier transform, DFT) を $O(Nlog_2N)$ 回の演算で行なう高速 Fourier 変換 (fast Fourier transform,FFT) というアルゴリズムは、近年、科学技術計算において広く使用されている。そのため、FFT の並列処理におけるデータの大容量化と高速化は、重要な課題である。

この FFT の並列処理は、問題を分割し各々の部分を並列に実行させることで実現できる。これは１次元 FFT を２次元化することで可能となる [?]。しかし、２次元化 FFT では２回の１次元 FFT との間に、全 PE から全 PE への通信が必要となりその時間はデータ量に比例して増加して行く。

本論文では、１次元 FFT を３次元化することで、演算と通信のオーバーラップが可能となり、並列処理の高速化が実現できることを述べる。また、２次元化 FFT との比較により性能評価をおこなった。プログラムは、３次元化 FFT を C 言語で作成しメッセージ交換ライブラリーは、MPI(Message-Pssing interface) を使用し、NEC Cenju3(PE 数:128) で実行した。

以下、２節では１次元 FFT の３次元化について、３節では３次元化 FFT の並列化について、４節では通信と演算のオーバーラップについて述べる。５節では３次元化 FFT の実験結果と２次元化 FFT と比較した性能評価について述べる。６節では全体のまとめを述べる。

Chapter 2

１次元 FFT の３次元化

効率的なアルゴリズムを開発する有効な手法は、大規模な問題を複数の小規模な複数の問題に分割することである。DFT においては、インデックス変数を線形変換することで、一次元の問題を多次元の問題に写像する方法が利用できる。これは、インデックス写像 (変数変換) と呼ばれる [?]。1D-FFT の 3D-FFT 化をこのインデックス写像を用い具体的におこなってみる。

2.1 DFT の３次元化

DFT の基本的な定義は、

$$g_k = \sum_{n=0}^{N-1} x_n \omega_N^{nk} \tag{2.1}$$

で与えられる。だたし、$\omega_N = e^{\frac{-2\pi jnk}{N}}$ そして $j^2 = -1$ である。まず、データ数 N を $N = N_1N_2N_3$に因数分解し、インデックス変数 n と k を以下のように線形変換する。

$$n = n_1N_2N_3 + n_2N_3 + n_3$$
$$k = k_1 + k_2N_1 + k_3N_1N_2$$

線形変換したインデックス変数を、定義式に代入し整理すると、

$$g_{k_1k_2k_3} = \sum_{n_3=0}^{N_3-1} \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_1n_2n_3} \omega_{N_1}^{n_1k_1} \omega_{N_1N_2}^{n_2k_1} \omega_{N_2}^{n_2k_2} \omega_N^{n_3k_1} \omega_{N_2N_3}^{n_3k_2} \omega_{N_3}^{n_3k_3} \tag{2.2}$$

が求まる。これは、一次元の DFT とひねり係数の乗算をそれぞれ 3 回おこなうことを表している。この式における入力データと出力データの関係を図?? に示す。

Figure 2.1: 入出力データの配列

2.2 ３次元化 FFT の流れ

３次元化 FFT は、式??を下式のように x 軸方向の XFFT、y 軸方向の YFFT、z 軸方向の ZFFT に、大きく３段階に分け実行する。つまり、XFFT の出力は YFFT の入力となり YFFT の出力は ZFFT の入力となる。最後の ZFFT の出力は、前述した出力データの配列で x 軸 y 軸 z 軸の順で出力される。また、３次元化 FFT のアルゴリズムを１～６に示す。

$$XFFT \Rightarrow X_{k_1 n_2 n_3} = \sum_{n_1=0}^{N_1-1} x_{n_1 n_2 n_3} \omega_{N_1}^{n_1 k_1} \omega_{N_1 N_2}^{n_2 k_1} \tag{2.3}$$

$$YFFT \Rightarrow Y_{k_1 k_2 n_3} = \sum_{n_2=0}^{N_2-1} X_{k_1 n_2 n_3} \omega_{N_2}^{n_2 k_2} \omega_{N}^{n_3 k_1} \omega_{N_2 N_3}^{n_3 k_2} \tag{2.4}$$

$$ZFFT \Rightarrow Z_{k_1 k_2 k_3} = \sum_{n_3=0}^{N_3-1} Y_{k_1 k_2 n_3} \omega_{N_3}^{n_3 k_3} \tag{2.5}$$

$$g_{k_1 k_2 k_3} = Z_{k_1 k_2 k_3} \tag{2.6}$$

Figure 2.2: 3D-FFT の流れ

1. １次元入力データ n を $n = n_1 N_2 N_3 + n_2 N_3 + n_3$に並び変える。
2. y-z 平面に対し、x 軸方向に N_1の長さの１次元 FFT を実行。
3. 全データに対し、ひねり係数$\omega_{N_1 N_2}^{n_2 k_1}$の乗算を実行。
4. z-x 平面に対し、y 軸方向に N_2の長さの１次元 FFT を実行。
5. 全データに対し、ひねり係数$\omega_{N}^{n_3 k_1} \omega_{N_2 N_3}^{n_3 k_2}$の乗算を実行。
6. x-y 平面に対し、z 軸方向に N_3の長さの１次元 FFT を実行。

Chapter 3

３次元化 FFT の並列化

並列化は、各 PE に分割されたデータを個別に実行することで実行時間の高速化がはかられる。しかし、個別実行した結果がそのまま全体の結果の一部とはならないのが普通であり、PE どうしでのデータ転送が必要となるため、転送時間や転送量によっては、PE 台数に見合った高速化がはかられない場合も多い。

3.1 ３次元データの PE 分割

３次元化データの入力となる１次元データ数は、２巾乗であることを前提としている。３次元化するにあたっても、各軸方向のデータ数が２巾乗であることが FFT のアルゴリズムを利用する条件となる。また、各軸のデータ数は、同一に近づけることが片よりのない FFT を実行でき、キャッシュに収まる確率をあげる。よって、３軸のうち２軸が同一のデータ数となり、残りの軸が１巾乗分大きいか少ないデータ数となる。この２軸が等しくなることを利用し、この２軸に対し PE 分割を行う。図**??**は、PE 数４台とし y 軸と z 軸が等しい時の例である。並列３次元化 FFT のアルゴリズムを１～８に示す。

1. １次元入力データ n を $n = n_1N_2N_3 + n_2N_3 + n_3$に並び変える。
2. z 軸を分割する形で各 PE に配置する。
3. y-z 平面に対し、x 軸方向に N_1の長さの１次元 FFT を実行。
4. ひねり係数$\omega_{N_1N_2}^{n_2k_1}$の乗算を実行。
5. z-x 平面に対し、y 軸方向に N_2の長さの１次元 FFT を実行。
6. ひねり係数$\omega_N^{n_3k_1}\omega_{N_2N_3}^{n_3k_2}$の乗算を実行。
7. 各 PE から各 PE へ、データの転送を実行。
8. x-y 平面に対し、z 軸方向に N_3の長さの１次元 FFT を実行。

Figure 3.1: ３次元データの PE 分割例

Chapter 4

通信と演算のオーバーラップ

前節では、３次元化 FFT の並列化を述べたが、これは通信と演算のオーバーラップがなされていないことがわかる。通信と演算のオーバーラップを考えたとき、まずそれが可能であるかの検討が必要である。２次元化 FFT の場合、各 PE で行う最初の１次元 FFT をすべて完了しなければ、次の１次元 FFT に必要なデータを各 PE に転送することができない。３次元化 FFT では、１ PE での YFFT が全て終了ぜずとも部分的に完了したデータは、各 PE に転送することが出来る。

4.1 YFFT の分割演算と分割転送

図??に示すように、YFFT の部分的な結果をノンブロッキング転送し、転送中に次の部分を実行することで通信と演算のオーバーラップが可能となる。しかし、この場合最後の分割転送部分 (TRANS-4) がオーバーラップされないという問題がおきる。この場合の並列３次元化 FFT のアルゴリズムを１～８に示す。

1. １次元入力データ n を $n = n_1N_2N_3 + n_2N_3 + n_3$に並び変える。
2. z 軸を分割する形で各 PE に配置する。
3. y-z 平面に対し、x 軸方向に N_1の長さの１次元 FFT を実行。
4. ひねり係数$\omega_{N_1N_2}^{n_2k_1}$の乗算を実行。
5. 分割された z-x 平面に対し、y 軸方向に N_2の長さの１次元 FFT を実行。
6. 分割されたデータに対し、ひねり係数$\omega_N^{n_3k_1}\omega_{N_2N_3}^{n_3k_2}$の乗算を実行。
7. 分割されたの部分の各 PE への転送を起動し、５～６を並列で実行。
8. 転送終了後 x-y 平面に対し、z 軸方向に N_3の長さの１次元 FFT を実行。

Figure 4.1: YFFT の分割演算と分割転送

4.2 YFFT&ZFFT の分割演算と分割通信

前述した問題の対策として、分割演算後、分割転送により転送されたデータにより部分的に ZFFT を実行する方法が考えられる。この方法だと、通信とオーバーラップさせる演算を分割 $YFFT$ と分割 ZFFT となり、通信時間の遅いマシンでもオーバーラップ可能と考えられる。この演算と通信のオーバーラップイメージを図?? に示す。この場合の並列３次元化 FFT のアルゴリズムを１～８に示す。

1. １次元入力データ n を $n = n_1N_2N_3 + n_2N_3 + n_3$に並び変える。
2. z 軸を分割する形で各 PE に配置する。
3. y-z 平面に対し、x 軸方向に N_1の長さの１次元 FFT を実行。
4. ひねり係数$\omega_{N_1N_2}^{n_2k_1}$の乗算を実行。
5. 分割された z-x 平面に対し、y 軸方向に N_2の長さの１次元 FFT を実行。
6. 分割されたデータに対し、ひねり係数$\omega_N^{n_3k_1}\omega_{N_2N_3}^{n_3k_2}$の乗算を実行。
7. 初回転送時、分割部分の各 PE への転送を起動し５～６を並列で実行。
8. ２回目以降転送時、分割部分の各 PE への転送を起動し、前回転送を受けた部分の x-y 平面に対し、z 軸方向に N_3の長さの１次元 FFT を実行。

4.3 通信量の調整

オーバーラップをより効率的に行うためには、分割した演算を実行中に通信を完了させることが必要となる。つまり、いかに通信時間が演算時間の影に隠れるかである。これは、マシン性能、演算量、通信時間、通信量、通信先 PE 数に大きく影響される。そこで、効率的なオーバーラップの実現方法として、演算量 (=通信量) を調整することが必要となる。調整するための値を index 値と呼ぶ。これは、分割 YFFT の演算量を決めるもので x 軸の範囲で、２の巾乗で変更可能である。例えば、図??では、index=２なので YFFT-1(データ数８の１次元 FFT を４回) を実行し、結果計３２個の複素データを４PE に８個づつ転送し、各 PE では他の４ PE から受信した計３２個の複素データにより、ZFFT-1 を実行する。現在この index 値は、パラメータとして手入力し最高の性能が出るよう調整する必要がある。今回の実験結果ではこの index 値で良い性能が出たとき値をを使用している。また、この index 値をｘ軸と同じにする事でオーバーラップ処理をスキップすることができる。

Figure 4.2: YFFT&ZFFT の分割演算と分割転送

4.4 プロセッサ間通信

通信が特定のプロセッサに集中しないように、送信側は自プロセッサに+1 した先に送信し、受信側は自プロセッサに-1 した先から受信する。

Figure 4.3: プロセッサ間通信

Chapter 5

実験結果と性能評価

今回、実験で使用した並列計算機は、NEC Cenju3(128PE, ローカルメモリ 64M/PE, 浮動小数点演算ピーク性能 50MFLOPS/PE) である。測定方法は、２の巾乗の複素データ数と PE 数を変化させ順変換の実行時間を測定した。測定値は３回の平均とし、ばらつきの多いデータに限り３〜５回の平均とした。

5.1 2D-FFT

2D-FFT における実行時間を表 ??に示す。表中のMはメモリネックのため実行できなかったことを示している。この表??より台数効果を見てみると、速度アップ率の図??では、台数を増やすと性能が落ちるということがわかる。MFLOPS 値の図??では、2^{14}〜2^{18}までは、台数効果がリニアに上がらず $2^{14}, 2^{16}$においては PE 数が３２、６４台の時のほうが１２８台の時より良い性能がでている。$2^{20}, 2^{26}$ においては、ほぼリニアに台数効果が現れていることがわかる。これは、少ないデータ量の時は、転送する PE 数が多くなると１ PE に転送するデータ量も少なくなり、転送時間にしめる通信の立上り時間が無視できなくなるためである。

Table 5.1: 2D-FFT における実行時間 [sec](順変換)

PE 数	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	0.485	2.461	11.566	M	M	M	M
2	0.266	1.253	6.180	27.273	M	M	M
4	0.138	0.629	3.042	13.915	62.854	M	M
8	0.073	0.319	1.469	7.026	31.579	M	M
16	0.043	0.171	0.740	3.519	16.217	70.906	M
32	0.034	0.096	0.378	1.673	8.024	35.192	M
64	0.043	0.075	0.214	0.840	3.927	17.605	M
128	0.078	0.095	0.167	0.474	1.956	8.834	38.473

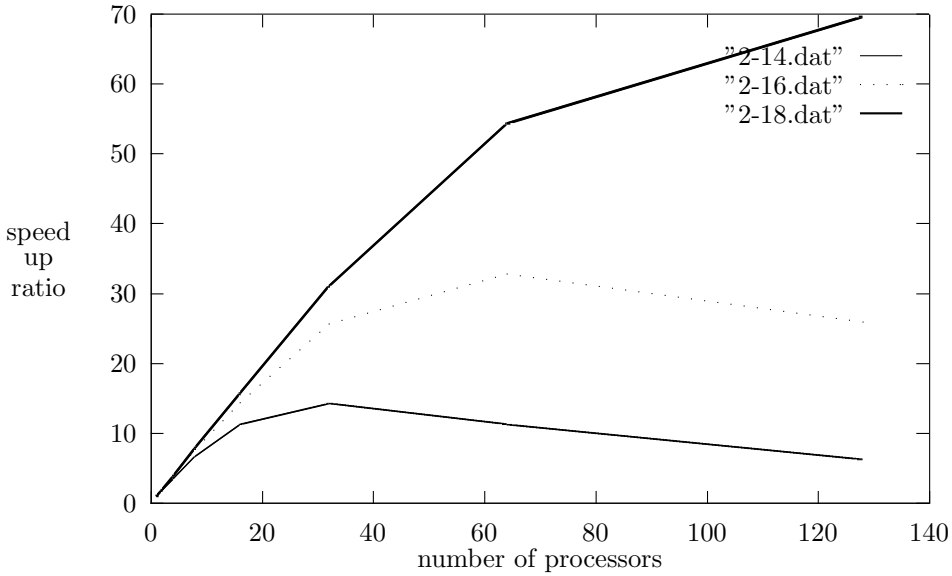

Figure 5.1: Cenju3 における 2D-FFT の台数効果 (その１)

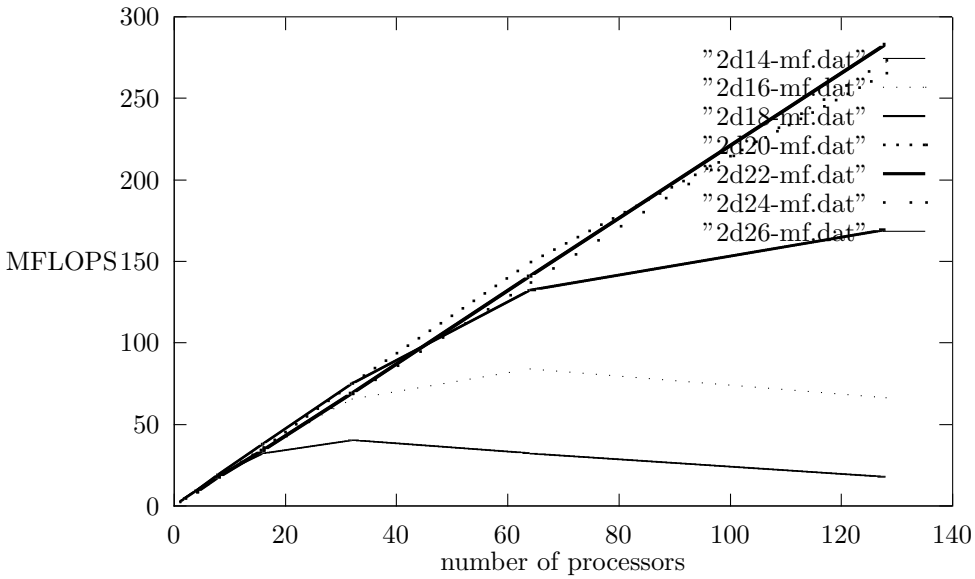

Figure 5.2: Cenju3 における 2D-FFT の台数効果 (その２)

5.2 3D-FFT

3D-FFT における実行時間を表??に示す。表中のMはメモリネックのため実行できなかったことを示している。また、＊はプログラム上 xyz の各軸より PE 数が多いこと制約となっているため、実行できなかったことを示している。この表??より台数効果を見てみると、速度アップ率の図??では、$2^{16}, 2^{18}$の時のデータが PE 数１２８台まで取れないため、途中までのリニアな部分だけしか確認できなかった。MFLOPS 値の図??では、2^{16}〜2^{18} については速度アップ率同様確認できないが、2^{20}〜2^{24}についてはほぼリニアに台数効果が現れていることがわかる。

Table 5.2: 3D-FFT における実行時間 [sec](順変換)

PE 数	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	0.541	2.579	11.485	M	M	M	M
2	0.279	1.263	5.856	25.830	M	M	M
4	0.140	0.627	2.896	12.916	M	M	M
8	0.072	0.353	1.426	6.508	28.702	M	M
16	0.037	0.164	0.710	3.211	14.404	M	M
32	*	0.086	0.348	1.597	7.242	31.448	M
64	*	*	0.188	0.801	3.645	15.868	M
128	*	*	*	*	1.854	8.146	M

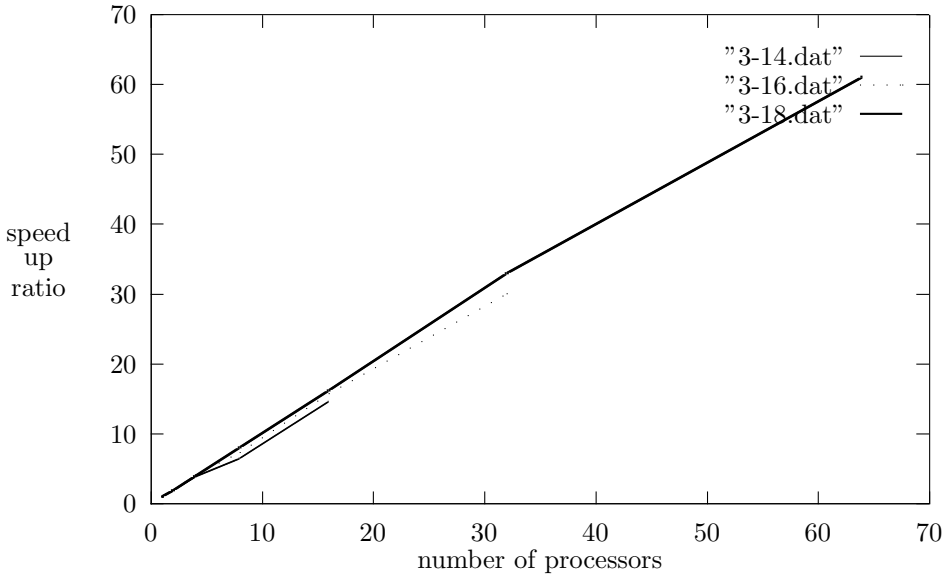

Figure 5.3: Cenju3 における 3D-FFT の台数効果 (その１)

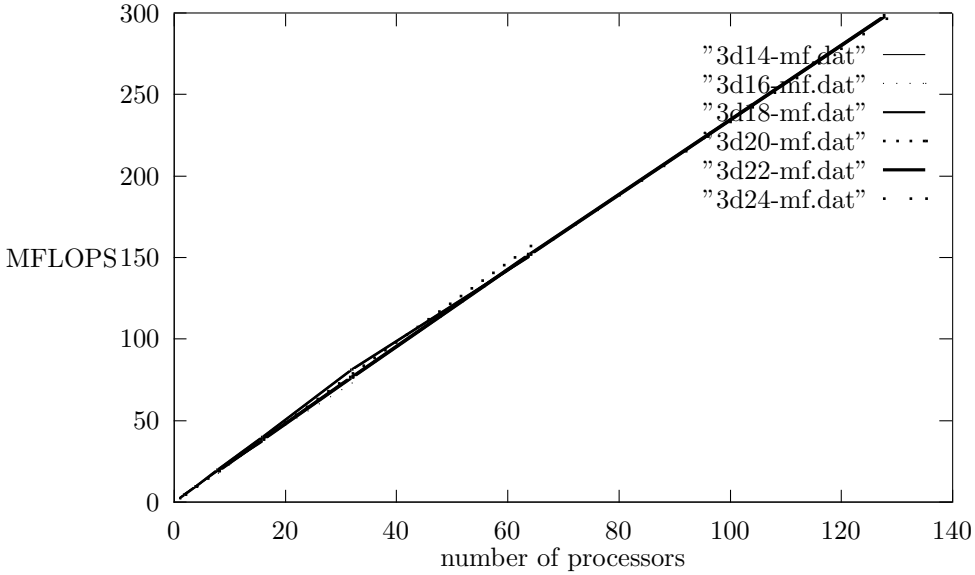

Figure 5.4: Cenju3 における 3D-FFT の台数効果 (その 2)

5.3 2D-FFT と 3D-FFT の比較

2D-FFT と 3D-FFT を性能比較したのが表??で、MFLOPS 値を比較したのが表??である。この表は 2D-FFT の性能や MFLOPS 値を基準とし 3D-FFT の比率を表したものである。この２つの表はほとんど同じ値を示している。この表より、2^14 で PE 数４台までと、2^16 で PE 数８台までは 2D-FFT のほうが性能が良く、それ以上の PE 数の時とデータ数の時は 3D-FFT のほうが性能が約５%～１３%良いことがわかる。また、この表をグラフ化したのが図??と図?? である。これによりオーバーラップによる高速化がはかれたことがわかる。

Table 5.3: 2D-FFT と 3D-FFT の性能比較

PE 数	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	0.896	0.953	1.011	M	M	M	M
2	0.953	0.991	1.055	1.055	M	M	M
4	0.985	1.003	1.051	1.084	M	M	M
8	1.013	0.903	1.030	1.077	1.096	M	M
16	1.162	1.042	1.042	1.095	1.131	M	M
32	*	1.116	1.086	1.048	1.112	1.107	M
64	*	*	1.138	1.048	1.078	1.125	M
128	*	*	*	*	1.054	1.079	M

Table 5.4: 2D-FFT と 3D-FFT の MFLOPS 値比較

PE 数	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}
1	0.896	0.954	1.007	M	M	M	M
2	0.954	0.992	1.055	1.055	M	M	M
4	0.985	1.003	1.050	1.077	M	M	M
8	1.013	0.903	1.030	1.079	1.100	M	M
16	1.162	1.042	1.042	1.095	1.125	M	M
32	*	1.116	1.086	1.047	1.107	1.119	M
64	*	*	1.138	1.048	1.077	1.109	M
128	*	*	*	*	1.055	1.084	M

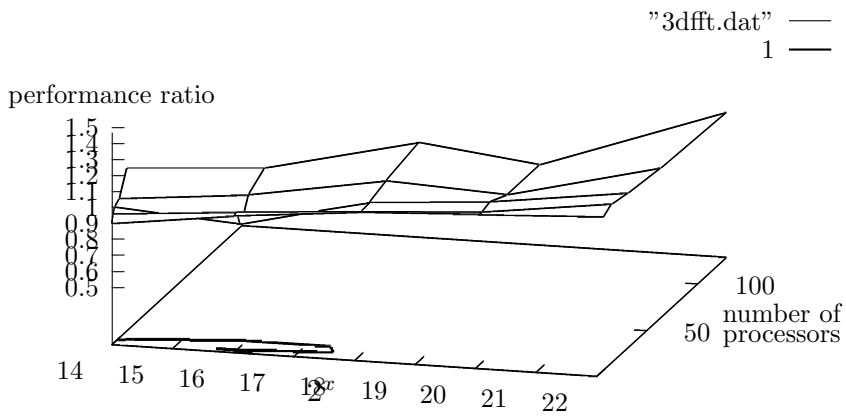

Figure 5.5: Cenju3 における 2D-FFT と 3D-FFT の比較 (その１)

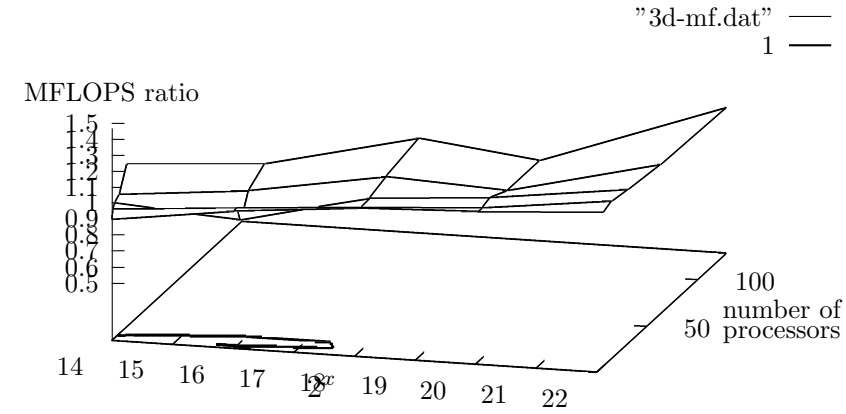

Figure 5.6: Cenju3 における 2D-FFT と 3D-FFT の比較 (その２)

Chapter 6

まとめと今後の課題

本論文では、１次元 FFT を３次元化することで演算と通信のオーバーラップを実現できることを述べた。また、２次元化 FFT との比較により性能評価をおこなった。プログラムは、３次元化 FFT を C 言語で作成しメッセージ交換ライブラリーは、MPI(Message-Pssing interface) を使用し、NEC Cenju3(PE 数:128) で実行した。その結果、データ数が 2^{18}〜2^{24}の時は、2D-FFT と比較し性能が約５%〜１３%向上し、並列処理の高速化が実現できることがわかった。しかし、2^{14}〜2^{16}の時は 2D-FFT の方が性能が良いため、アプリケーションとしては、扱うデータ数に応じて使い分ける必要がある。

今後の課題として、

1. メモリネック問題の解決。
2. PE 数に影響されない３次元化の実現。
3. index 値の自動設定を組み込み。
4. オーバーラップ転送の改良。

があげられる。

Chapter 7

謝辞

本研究は、多くの人からの協力により完成することができました。特に、若輩ものの私に、多大な御指導と助言を頂きました我が研究室の指導教員であられる清水尚彦先生に厚く御礼申し上げます。また、実験環境を提供して頂いた東海大学計算機センターの方々に感謝します。その他、協力を頂きました方々には大変感謝しております。

Bibliography

[1] Swartztrauber, P.N.: Multiprocessor FFTs.Parallel Computing,no.5, (1987)1.97-210.

[2] Jae S.Lim Alan V.Oppenheim ed.: 現代ディジタル信号処理理論とその応用, (青山友紀監訳), 丸善株式会社,1992.

[3] 青山 幸也: 並列プログラミングの虎の巻 MPI 版, 日本アイ・ビー・エム株式会社, 1996.

[4] NEC C&C 研究所: 並列処理センター利用者の手引,1996.