情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 1　40–50 (June 2008)

性能モデルに基づくCPU及びGPUを併用する効率的なFFTライブラリ

尾 形 泰 彦[†1,†2]　遠 藤 敏 夫[†1,†2]
丸 山 直 也[†1,†2]　松 岡 　聡[†1,†2,†3]

General-purpose GPU（GPGPU）を HPC の分野で利用する手法が，その非常に高いピーク性能のために注目されている．しかし，ホストとの転送 I/O 帯域幅やメモリサイズの制限等のため，実効性能は大幅に低下する傾向にある．一方で，CPU のマルチコア化も近年急速に進みつつあるため，GPU と CPU 上のアプリケーションの実効性能の乖離は小さい場合が多く，両者を併用することにより性能の改善が期待される．このとき，効率的な実行のためにはヘテロなプロセッサへのタスクの分割率を適切に決める必要がある．しかし，最適な分割率は問題サイズ等に依存して変化するために，自明な問題ではない．我々は 2D-FFT を対象問題として取り上げ，CPU と GPU を併用するライブラリを実装する．そして最適な分割率を得るために 2D-FFT のアルゴリズムを詳細に考慮した性能モデルを構築する．モデルのパラメータは小規模な予備実行により決定され，それをもとに任意の問題サイズと分割率に対して並列実行時間を予測することができる．実験の結果，性能モデルは予備実行の 16 倍のサイズの問題について，実行時間を 15%以内の誤差で予測した．予測から得られた最適分割率は 5%の誤差に抑えられ，この誤差に起因する性能低下は 1%以内であった．また，最適分割率における並列実行により，CPU 1 コアや GPU 単体の場合と比較して 1.19 から 1.55 倍の性能向上が得られた．

An Efficient, Model-based CPU-GPU Heterogeneous FFT Library

Yasuhiko Ogata,[†1,†2] Toshio Endo,[†1,†2]
Naoya Maruyama[†1,†2] and Satoshi Matsuoka[†1,†2,†3]

General Purpose computing on Graphics Processing Units (GPGPU) is becoming popular in HPC because of it's high peak performance. However, in spite of the potential performance improvements, it might not necessarily perform better than the current high-performance CPUs, especially with recent trends for increases in their number of cores on a single die. This is because the GPU performance can be severely limited by such restrictions as memory size and I/O bandwidth. For this reason, we can expect that performance is improved by using CPU and GPU simultaneously. In heterogeneous environments, we need to find optimal load distribution ratio. We implement a 2D-FFT library that uses heterogeneous CPU-GPU computing resources. To find optimal load distribution ratios, we construct a performance model that predicts execution time of 2D-FFT that captures the respective contributions of CPU vs. GPU. The model parameters are determined by pre-stage performance profiling; based on this, we predict the overall execution time of 2D-FFT for arbitrary problem sizes and load distributions. Preliminary evaluation shows that the performance model can predict the execution time of problem sizes that are 16 times as large as the profile runs with less than 15% error, and that the predicted optimal load distribution ratios have less than 5% error; performance overhead caused by this error is less than 1%. We show that the resulting performance improvement by such heterogeneous parallelization can be 1.19 to 1.55 times compared to using only a CPU core or a GPU.

1. はじめに

高性能な科学技術計算のために Graphic Processing Unit（GPU）を用いる，General-Purpose GPU（GPGPU）[1)]と呼ばれる手法が HPC 分野でも注目されている．最新の CPU のピーク性能が 25 GFLOPS 程度であるのに対して，近年のハイエンド GPU は 500 GFLOPS の理論性能を持つ．また価格性能比が良好であり，数値計算専用に設計された ClearSpeed [4)] や GRAPE [10)] 等のアクセラレータと比較して非常に低コストで性能を向上させることが可能である．

このような特性を持つ GPU を有効利用しようとする研究が数多く行われており[7),9)]，それらの多くは多様なアルゴリズムに対して，GPU の性能を引き出すことに重点が置かれている．しかしその実効性能は，以下に述べる 3 つのオーバヘッドのために低下する傾向にある．1 つ目のオーバヘッドは，GPU 上のメモリへのデータ転送によるオーバヘッドである．GPU 内部では 100 GB/s にも迫るバンド幅が存在する一方で，CPU/GPU 間のデータ転

†1 東京工業大学
Tokyo Institute of Technology
†2 独立行政法人科学技術振興機構，CREST
Japan Science and Technology Agency, CREST
†3 国立情報学研究所
National Institute of Informatics

© 2008 Information Processing Society of Japan

送は 1 GB/s 程度のオーダに落ち込んでしまう．2 つ目は GPU メモリの搭載量の制限によるオーバヘッドである．GPU は本来グラフィック用途に設計されているため，多くのグラフィックボードでは多くても 512 MB 程度のメモリしか搭載されていない．しかし，HPC アプリケーションでは，数 GB 程度のメモリを必要とする場合も多いため，データを参照するたびに送り直す必要がある等，データ転送によるコストが増大する．3 つ目はプログラミング API のオーバヘッドである．実行特性が画面描画向けの特性になっており，元から GPGPU を考慮しているわけではないため，GPU のポテンシャルを生かしきれない．また，従来の DirectX や OpenGL ではメモリ管理が難しいことや，GLSL や HLSL 等シェーダ言語の記述は非常に難しいため，必要以上にプログラマに負荷がかかる．近年，NVIDIA CUDA [11] をはじめとする効率的な GPGPU 環境が整備されてきており，この問題は大きく軽減されつつあるが，それ自身がメモリサイズ制限にともなう問題を完全に解決するわけではない．

このような GPGPU のオーバヘッドに加えて，近年 CPU のマルチコア化が進んでいるため，GPU と CPU の実効性能の差は大きくない場合も多い．このため，GPU に加え CPU を併用する並列処理により性能の向上が期待できる．しかし，その際に必要な，異種のプロセッサ間にどのような割合で計算を割り振るべきか決定する問題は自明ではない．GPU は上記の 3 つの問題のために CPU と異なる性能特性を持ち，最適な分割率は問題サイズ等に依存して変化することがありうるためである．

我々はこの問題を解決するために，性能モデルに基づき，CPU および GPU を併用するライブラリを提案する．モデルを用いた性能予測を行うことにより，最適な計算の分割率を決定するアプローチをとる．その第 1 歩として本稿では，信号解析等広い応用分野を持つ二次元高速フーリエ変換（2D-FFT）計算を対象とし，ライブラリと性能モデルを構築する．

このライブラリは Row-Column 法に基づいており，計算が各軸方向への 1 次元 FFT（1D-FFT）へ分解できることを利用して，計算の割り振りを行う．この 1D-FFT 計算には，GPU 側 CPU 側ともに既存のライブラリを利用した．CPU 側のライブラリとして，著名な高性能ライブラリである FFTW [6] を，GPU 側のライブラリとして，Govindaraju らによる GPUFFTW [8] と，CUDA パッケージに含まれる CUFFT を用いた．また，必要に応じて GPU 側計算を複数回に分けて実行させるため，GPU メモリサイズ以上の問題サイズの計算も可能とした．

性能モデルは 2D-FFT アルゴリズムを実行単位へ分解し，この分割された単位ごとにデータ量に対する GPU，CPU それぞれの実行時間を予測する．その際のアーキテクチャに依存するパラメータは小規模な予備実行により決定される．モデルは各単位の実行時間を総合して，任意の問題サイズと分割率に対して合計実行時間を予測する．これにより最適分割率を予測することができる．

実験の結果，予備実行の 16 倍の問題サイズの場合の実行時間を予測した場合，誤差は 15%以内であった．このときモデルが予測した最適分割率は，実際と 5%以内の誤差に抑えられ，予測に基づく分割率を用いて実行した場合，本来の最適実行時間に対する性能低下は 1%以内であった．また，CPU 1 コア，GPU 単体の場合と比較して 1.19 から 1.55 倍の性能向上が得られることが分かった．

2. 関連研究

GPGPU の研究はすでに多くなされており，LU 分解 [7]，FFT [9] 等のアルゴリズムが提案されている．それらの多くは GPU 単体での性能に注目している．その中で大島ら [13] は，GPU と CPU を併用する行列積（GEMM）ライブラリの提案と実装を行い，単体のみの場合よりも高性能を達成している．このライブラリでは行列積演算を GPU と CPU に分割する．GPU への計算命令をまず（OpenGL API を用いて）発行し，その後 CPU の計算を行い，そして GPU からの結果を待つ．行列積と比べ，本研究が対象とする 2D-FFT では，フェーズ間の行列転置のために GPU と CPU 間の連携が多く必要となり，構築するモデルはそれを考慮に入れている．また，大島らの研究ではシングルコア CPU を仮定しているのに対し，本研究のライブラリはマルチコア CPU を考慮に入れている．

GPU における性能モデルの構築は今までにいくつか行われている．Buck ら [3] は，グラフィックの知識なしに GPGPU を行うことができる言語処理系 BrookGPU を提案し，その中で転送量と計算量に各々係数を与えるという本手法に近いモデル構築を行っている．しかし，Buck らはすべてを線形近似で行うほか，実性能との比較は行っていない．

そのほかの GPU 上の性能モデルの構築としては，伊藤ら [12] の研究があげられる．このモデルでは，メインメモリ，GPU メモリ，GPU 演算器間の通信量に注目して性能予測を行う．このため GPU プログラム実装前から実行時間を予測が可能という特徴を持つ．一方我々は，通信量だけでは説明できない，GPU メモリ確保や解放等の処理が，実行時間の無視できない割合を占めることを確認している．またこの研究自体は，ヘテロなプロセッサの利用を目的としない．

速度の異なる CPU 群を持つヘテロな環境での並列計算は広く研究されており，速度に応じてデータを分割する手法も知られている [5]．一方，我々が対象とする環境では，問題サイ

© 2008 Information Processing Society of Japan

ズに対する特性が大きく異なる GPU と CPU を含むため，本研究では特性を考慮したモデルを構築する．

3. GPU と CPU を併用する 2D-FFT ライブラリ

本ライブラリは，入力データとして二次元複素数配列をとり，Row-Column 法と呼ばれる手法により二次元 FFT（以下，2D-FFT）計算を行う．Row-Column 法では，まずすべての列に対してそれぞれ一次元 FFT（以下，1D-FFT）を行い，次にすべての行について同様に 1D-FFT を行う．本ライブラリでは，**図 2** のように，この複数の 1D-FFT 処理を CPU および GPU に割り振ることにより，CPU と GPU の併用を行う．この実際の処理のために，本ライブラリから，CPU 用および GPU 用に設計された既存の FFT ライブラリを呼び出す．CPU と GPU への処理量の割り振りは，後述のように与えられる分割率に応じて行う．また，マルチコアプロセッサを利用するために，CPU 側には複数のスレッドにより計算を行うことが可能である．スレッド間の割り振り率も変更可能であるが，本稿の実験では，スレッド間は均等としている．

図 1 に実行の流れを示す．列方向 1D-FFT と行方向 1D-FFT の間，およびその前後には，以下の理由により二次元行列の転置処理を行う．二次元配列は一次元のメモリ上に配置されるため，特定の行または列に注目すると，一方は連続したメモリに配置されるが，他方は飛び飛びに配置される．既存の FFT ライブラリでは，効率的な実行のために入力に特定のデータ配置を要求し，しかもライブラリによってそれが異なりうる．このため，転置処理を適切な箇所で行うことにより（このために図中 (a) と (b) の差異が生じる），前半でも後半でも効率的実行が可能なようにする．

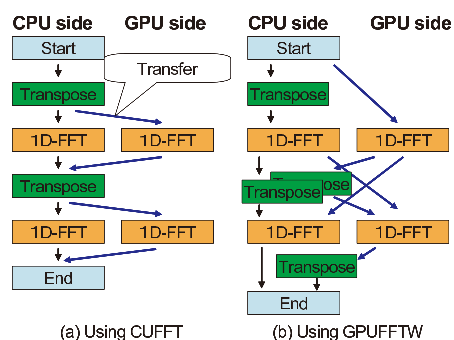

図 1　実行の流れ
Fig. 1　Execution flows of the CUFFT based and GPUFFTW based libraries.

本ライブラリは GPU のメモリサイズ以上のデータを以下のようにして扱うことができる．GPU 側にメモリサイズ以上のデータ量の FFT を割り振る場合には，収まるサイズに調整した数の 1D-FFT を複数回呼び出す．GPU のメモリは総じて CPU のメモリよりも少ない傾向にあり，市販されているグラフィックボードに搭載されているメモリはたかだか 768 MB 程度である．一方で，HPC 用システムではメインメモリは応用ユーザの要求に応えるために，CPU あたり数 GB 積まれることが多い．そのため，本ライブラリでは GPU 単体で単純には計算不可能な大きさの問題にも対応している．

現在の実装の制約として，本ライブラリは 2 のべき乗の問題サイズのみに対応する．また現在の GPU が倍精度計算に対応しないという制約により，本ライブラリは単精度にのみ対応する．

3.1 本ライブラリが利用する FFT ライブラリ

今回，CPU 側のライブラリとして FFTW [6]，GPU 側のライブラリとして GPUFFTW [8] および CUDA [11] が提供する FFT ライブラリの 2 種類を用いた．これらは差し替えることが可能であるが，後述するライブラリインタフェースの差異には注意する必要がある．

FFTW は MIT の Matteo Frigo および Steven G. Johnson. により開発された CPU 用 FFT ライブラリで，実行前の仮実行に基づき，実行時間の最適なアルゴリズムを自動で選択可能なライブラリである．

GPUFFTW は，UNC Chapel Hill の Naga K. Govindaraju らが開発した GPU 上での FFT ライブラリである．このライブラリは，グラフィックメモリの構造を考慮した最適化を行うことが特徴である．GPUFFTW は OpenGL の NVIDIA 社による拡張である，

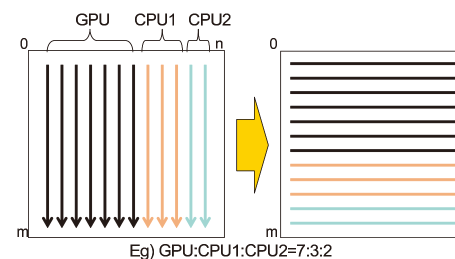

図 2　GPU/CPU への 1D-FFT の分割
Fig. 2　Example distribution of FFT computation to one GPU and two CPUs.

© 2008 Information Processing Society of Japan

NV_fragment_program を用いているため，NVIDIA 社の GPU にのみ対応する．

CUDA（Compute Unified Device Architecture）[11] は NVIDIA が開発した GPGPU 専用環境である．NVIDIA Geforce8000 系以降の GPU に対応しており，CUDA は GPU 上のプログラムを，C 言語に近い形で簡単に書くことが可能である．このためユーザは，グラフィック API を用いずに GPGPU を行うことが可能である．また，一般的な GPGPU の実装を Geforce8000 系列上で実行した場合と比較して，より高い性能を引き出すことが可能である．今回は CUDA プログラムを記述するのではなく，CUDA パッケージ中の CUFFT ライブラリを使用した．

3.2　実行の詳細

本ライブラリでは，1 つ以上の CPU 計算スレッドと，1 つの GPU コントロールスレッドが協調的に動作する．前者は FFTW を呼び出し，後者は GPUFFTW もしくは CUFFT を呼び出す．それぞれのライブラリが要求するデータ構造にあわせるため，二次元配列の転置が必要である．今回用いたライブラリのうち，FFTW および CUFFT は Row-major のデータ構造を要求し，GPUFFTW は Column-major のデータ順を要求する．このため，図 1 の (a) と (b) の差異が生じる．CUFFT 版 (a) では各次元の 1D-FFT 処理の前に転置が行われる．一方，GPUFFTW 版 (b) では，CPU が担当する領域と GPU が担当する領域とで，転置の扱いを変更する必要がある．このような差はあるが，CPU-GPU 間転送量には両者で差はない．

二次元配列の転置は，現在の実装では CPU 上のみで行い，CPU 計算スレッドのうちの 1 つおよび GPU コントロールスレッドの計 2 スレッドで実行する．また，各スレッドには 1D-FFT 処理の割当て率と同じ割合で転置処理を振り分ける．列方向計算および転置が終了した時点で，行方向計算を行う前に，全スレッド間で同期をとっている．

4.　本ライブラリの性能モデルの構築

本章では，実装した 2D-FFT ライブラリの実行時間を予測するための性能モデルを述べる．この予測実行時間を用いることにより，実際の計算より前に最適な割当て率を探索することが可能である．現在のところ，モデルは CPU 側の計算を 1 スレッドで行う場合を対象としている．また本章では，CUFFT を用いる場合を述べるが，GPUFFTW を用いる場合のモデルも同様に構築している[14]．以下の説明で，問題サイズは $n \times n$ とする．

我々が構築した性能モデルは，2D-FFT の実行を細かい実行単位に切り分け，各実行単位の実行時間を問題サイズ n，割当て率，アーキテクチャパラメータで表現する．たとえば，CPU から GPU へデータを転送するための時間は，送信するデータ量に比例すると考え，$K_{c2g} * r * n^2$（K_{c2g} は通信性能に依存する係数，r は GPU への割当て率，n は問題サイズ）とモデル化する．アーキテクチャパラメータについては，予備実験において各実行単位の時間を実測し，そこから求める．

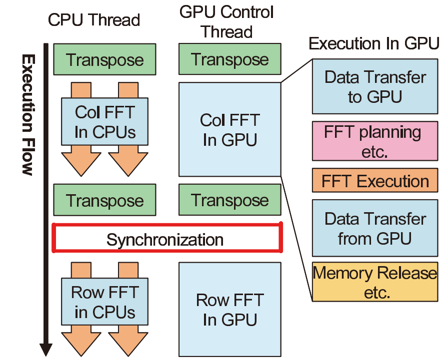

図 3　実行フローの詳細（CUFFT 版）
Fig. 3　Execution steps in our model for the CUFFT-based library.

表 1　性能モデルのパラメータ
Table 1　Model parameters.

K_{trans}	CPU での転置
K_{gMa}	GPU メモリ確保
K_{c2g}	CPU から GPU へのデータ転送
K_{g2c}	GPU から CPU へのデータ転送
K_{gP}	GPU 側 FFT 前処理
K_{gDP}	GPU 側 FFT 後処理
K_{cFFT}	CPU 側 FFT 実行
K_{gFFT}	GPU 側 FFT 実行
K_{gMf}	GPU メモリ開放

図 3 に，本ライブラリ（CUFFT 版）の実行フローを，モデルにおける実行単位ごとに示す．また各実行単位に対応するアーキテクチャパラメータを**表 1** に示した．実行単位は，CPU または GPU 上の 1D-FFT 計算や通信処理以外にも，既存ライブラリを呼び出す際の前処理，後処理も含んでいる．これは，既存ライブラリの利用において，実行条件を示す “Plan” の作成等に，無視できない時間がかかるためである．

式 (1)，(2) に CPU 計算スレッドの実行時間を示す．ここで n は問題サイズ，r は GPU

© 2008 Information Processing Society of Japan

側への割当て率である（$0 \le r \le 1$）．T_{cpu1} は列方向計算，T_{cpu2} は行方向計算についての予測実行時間を表す．2 回のデータ転置の時間は，列方向に含めている．

$$T_{cpu1} = (1-r)n^2 * 2K_{trans} * M_{trans}(1-r) + (1-r)n^2 \log n K_{cFFT} \quad (1)$$

$$T_{cpu2} = (1-r)n^2 \log n K_{cFFT} \quad (2)$$

式 (3)，(4) に同様に GPU 側のモデル式を示す．GPU 側は図 3 に示したとおり，実際の FFT 計算に加え CPU-GPU 間の転送や前処理・後処理に関する項を含む．転置については GPU コントロールスレッドが CPU 上で行っており，これに関する項も加わる．

$$T_{gpu1} = rn^2(K_{gMa} + K_{c2g} + K_{gP} + K_{gDP} + K_{g2c}) + rn^2 * 2K_{trans} * M_{trans}(r) + rn^2 \log n K_{gFFT} \quad (3)$$

$$T_{gpu2} = rn^2(K_{c2g} + K_{gP} + K_{gDP} + K_{g2c} + K_{gMf}) + rn^2 \log n K_{gFFT} \quad (4)$$

なお図では，GPU 内の動作が 1 セットしか描いていないが，実際には GPU メモリに収まるサイズに列/行を分割し，複数回繰り返す．しかし，1 回の関数呼び出しで十分な数の列/行をまとめる場合，計算時間と通信時間の両方が，列/行の本数に比例するため，今回のモデル化では GPU 計算処理時間は n，r とアーキテクチャにのみ依存し，繰返し回数に依存しないとした．

これまでに示した各式を，列方向・行方向ごとに CPU/GPU の大きいほうの実行時間を得て，合計することで全体の実行時間を式 (5) のように求める．これは，計算の方向が切り替わる際に同期を行うためである．

$$T_{total} = \max(T_{gpu1}, T_{cpu1}) + \max(T_{gpu2}, T_{cpu2}) \quad (5)$$

4.1　転置時間の補正

転置処理時間は，担当データサイズの比例するとしたうえで定義したモデルでは実測値とずれが存在する．このずれを解析するため，我々は予備評価を行った．その結果を**図 4** に示す．図 4 は問題サイズ 8,192 で実行した本ライブラリ中での GPU 側の 1 回目の転置実行時間を，スレッドが担当する割合に対して 1 スレッドでのデータ量あたりの実行時間を 1 とし，相対的な実行時間としてプロットしたグラフである．図 4 に示されるとおり，データ量あたりの相対的な実行時間はスレッドへの転置の割当て率が 50%まではほぼ一定値で

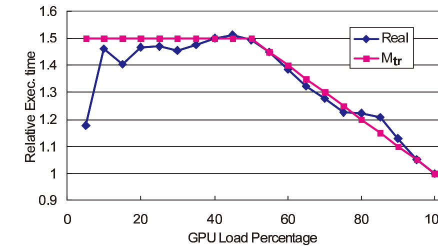

図 4　データ量あたりの相対的な実行時間（問題サイズ 8192^2，1 スレッドでの実行時を 1 とする）
Fig. 4　Relative execution time to transpose a 8192^2 matrix.

あり，その後線形に 1 に向かって線形に減少している．もし，転置を 2 スレッドで実行し性能が 2 倍になる場合，図 4 に示される “From Real Data” は 1 で一定であるはずである．この性能を表現するため，スレッドに対する計算量の割当て率 r_t に依存する各転置時間を補正する項として M_{trans} を設けた．

式 (6) に M_{trans} の定義を示す．我々は補正項 M_{trans} を設けこの特性をモデルに適用した．M_{trans} は図 4 に示されたデータ量あたりの相対的な実行時間同様にスレッドへの転置の割当て率が 50%までは一定値，その後線形に 1 に向かって線形に減少する関数とした．ここで用いているパラメータ C_{tr} は環境に依存する変数と思われ，我々が用いた実験環境では $C_{tr} = 1.5$ とした．

$$M_{trans}(r_t) = \begin{cases} (1-C_{tr})\frac{(r_t - 0.5)}{0.5} + C_{tr} & \text{if } r_t > 0.5 \\ C_{tr} & \text{if } r_t \le 0.5 \end{cases} \quad (6)$$

転置実行時間が図 4 で示すとおりに推移する理由は以下のとおりであると推測される．前提として，我々のライブラリでは，データの転置の際に CPU 計算スレッドおよび GPU コントロールスレッドの両者が各々の担当範囲を並列に転置する．また，2 スレッドが等量の転置を並列に実行する場合，双方のスレッドの実行時間が C_{tr} 倍される．一方，2 スレッドが等量の転置を行わない場合，データ量が少ない側は転置終了まで他のスレッドが存在する状態で転置を行うが，データ量が多い側はデータ量が少ない側の転置終了後は 1 スレッドでの実行となり性能低下が起こらない．このとき，1 スレッドでの実行がデータ量の偏りが大きいほど長いため，実行時間が割当て率が 50%以上の場合は線形に減少するものと推測される．パラメータ C_{tr} はメモリバンド幅による性能の制限等に拠る環境依存の変数と考えており，転置するデータ量がキャッシュサイズを上回る場合は一定に近くなる．図 4 のス

© 2008 Information Processing Society of Japan

レッドでの割当てが 5%のときのように，キャッシュに転置する領域が収まる場合は M_{trans} が異なる挙動をするが，現在は解析およびモデル化はできていない．これらについては，今後の研究で究明およびモデルのさらなる改良を行う予定である．

以上に述べたモデルでは，現在はキャッシュ等の影響を考慮しておらず，また前処理・後処理の時間も GPU の担当データサイズに単純に比例すると仮定した．上で示したモデルで，実行時間をどの程度の精度で予測可能かを，5.2 章で示す．

5. ライブラリ性能とモデルの精度の評価

5.1 本ライブラリの実行性能

本ライブラリの評価を**表 2** に示すような Dual Core CPU を持つ環境で行った．なお，今回用いた GPU である Geforce8800GTX は 1.35 GHz で動作する 128 個のストリームプロセッサを搭載し，グラフィックメモリとして 768 MB の GDDR3 メモリを搭載する．

図 5 に，本ライブラリの GPUFFTW 版，CUFFT 版の最適分割率での実行時間を示す．加えて，比較のために CPU 上の FFTW と GPU 上の CUFFT の，二次元 FFT 関数（以下，2D-FFTW および 2D-CUFFT）の実行時間を示す．横軸は問題サイズ，縦軸は実行時間である．また本ライブラリ，FFTW においては CPU 上の計算のために 2 スレッド利用している．

我々のライブラリは FFTW 単体よりも性能が向上しており，CUFFT 版で問題サイズが 8,192 の場合，約 3.5 倍の性能向上となっている．また CPU/GPU 併用の場合，GPUFFTW 版よりも CUFFT 版の方が約 30%高速である．これは GPU 側ライブラリの性能の差によるものであり，その差は併用する場合の性能にも現れた．

一方で本ライブラリは，すべてを GPU 上で実行する 2D-CUFFT と比較して性能が出ていない．推測される原因として，本ライブラリが CPU 上で転置を行うことを前提に置いていることがあげられる．このため，本ライブラリにおいては CPU-GPU 間の転送量が 2D-CUFFT の倍となっており，さらに転置処理自体の性能も，メモリバンド幅のために GPU より CPU の方が低速と考えられる．ただし，2D CUFFT が高速なのは GPU メモリサイズに収まる問題サイズに限られ，4096^2 を超えるサイズでは実行できない．本ライブラリはそれ以上のサイズでも計算可能であるし，また 2D-CUFFT においても大きな問題サイズに対応しようとすると，やはり CPU-GPU 通信の量は同様に増加すると考えられる．

なお今後，GPU 側でも転置を行うライブラリを実装する予定である．また，2D-CUFFT のようなすべて GPU で行う場合をモデルに加え，問題サイズに応じて適応的に選択する手法も開発したい．

図 6 に，本ライブラリにおいて，GPU 割当て率 r を 0%，100%，最適値（Optimal）とした場合の性能を示す．CPU 側計算には 2 スレッドを用いている．GPUFFTW 版と CUFFT 版それぞれについて問題サイズに対する実行時間を示した．CUFFT 版においては 0%（CPU のみ），100%（GPU のみ）よりも最適値の場合が高速である．CPU のみと比較して 1.5 倍，GPU のみと比較して 1.2 倍性能が向上している．一方，GPUFFTW 版（optimal）においては，100%（GPU のみ）より向上しているが，CPU のみと比べ性能改善が見られていない．この点については，以下で議論する．

図 7 に GPUFFTW 版について，GPU 側割当て率 r の変動と実行時間の関係のグラフを示した．問題サイズ n は 8,192 である．CPU 側計算スレッドを 1 つ用いた場合と，2 つ用いた場合を示す．1 スレッド用いた場合には，割当て率を 40%とした場合が最速とな

表 2　評価環境
Table 2　Evaluation enviroment.

CPU	Core 2 Duo E6400(2.13 Ghz*2)
Memory	PC6400 4 GB
Chipset	intel 975X Express
HDD	250 GB
GPU	Geforce 8800GTX
GPU Mem.	GDDR3 768 MB
OS	Linux kernel2.6.18
Driver	NVIDIA Display Driver ver97.46
	CUDA ver0.81

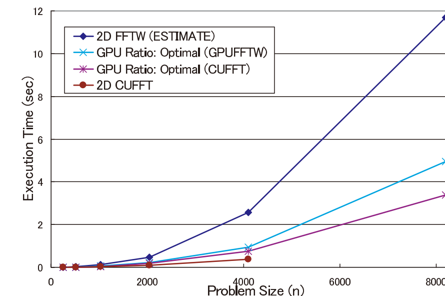

図 5　各 2D ライブラリの実行時間
Fig. 5　Performance comparison with existing 2D-FFT libraries.

© 2008 Information Processing Society of Japan

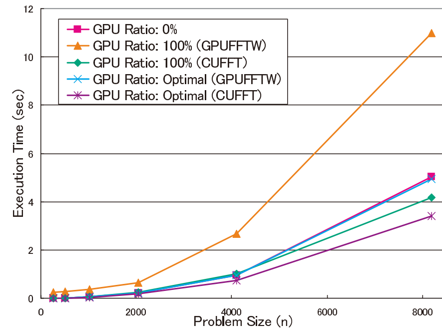

図 6　本ライブラリの各実装における実行時間
Fig. 6　Performance comparison with varying problem sizes.

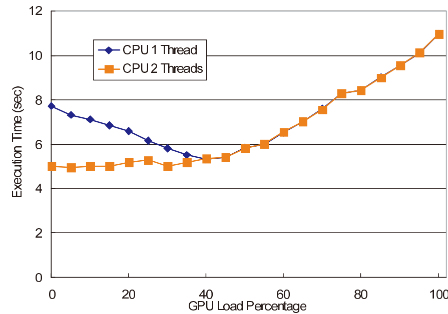

図 7　GPU 割当て率に対する実行時間の変化（GPUFFTW 版）
Fig. 7　Performance of GPUFFTW-based our library with varying distribution ratios.

り，CPU/GPU を併用する利益が現れている．このとき，0%時よりも 31%，100%時よりも 52%改善されている．一方で 2 スレッド用いた場合には，すでに示したとおり，CPU に加えて GPU を併用する利点が現れず，さらに r が 40%以上の場合は 1 スレッド時の性能とほぼ同じとなった．この理由を以下のように推測している．利用した GPUFFTW では，GPU 側の計算が終了するまで OpenGL の関数内でビジーウェイトしてしまい，CPU を消費することが分かっている．このため，2 コア CPU 上で，GPU コントロールスレッドと 2 つの CPU 計算スレッドの計 3 スレッドが同時に CPU を消費してしまう．これらのために CPU 側の計算が遅くなり，全体性能を低下させていると推測される．

次に CUFFT 版について，同様の評価結果を，**図 8** に示す．全体の性能は，GPU 側への

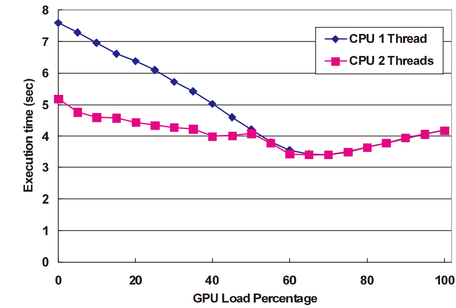

図 8　GPU 割当て率に対する実行時間の変化（CUFFT 版）
Fig. 8　Performance of CUFFT-based our library with varying distribution ratios.

割当てが 70%付近で最高となった．CPU 計算スレッドが 1 つの場合，r = 0%の場合に比べ 55%，r = 100%の場合に比べ 19%高速となった．一方，50%以上の場合に，1 スレッドの場合と 2 スレッドの場合の性能がほぼ等しくなっている．この原因は GPUFFTW 版と同様に，CUDA においても，今回使ったバージョンにおいては GPU の計算終了を待つ間 CPU を消費する⋆1ためと推測している．一方で，GPUFFTW 版と異なり，2 スレッドの場合でも，r = 0%よりも併用時が向上している．この理由は，CUFFT 単体の性能が GPUFFTW より優れており，GPU の計算中に CPU を無駄に消費する時間が少なくなり，全体性能の低下を抑えているためと考えられる．

5.2　性能モデルの検証

性能モデルの評価を，予測時間と実測時間の比較，および最適な GPU と CPU の分割率を推定できるか否かについて行う．性能モデルの各パラメータについては，特定の問題サイズについて予備実験を行い，各実行単位の実行時間を測定することにより決定する．本節では CUFFT 版について，CPU 計算スレッドは 1 つのときの検証結果を示す．

表 3 に予備実験から得られたパラメータを示す．予備実験時に問題サイズ 512 と用いた場合と，8,192 を用いた場合を示す．モデルが挙動を完全に予測可能であれば，両者のパラメータは一致するはずだが，現在は両者の間に差が存在し，予測に誤差が生じている．これらについては後で詳しく議論する．

⋆1 NVIDIA Forums [2] トピック 28524 の NVIDIA 開発者のコメントより．また，我々もその現象を，実行中プロセスの CPU 利用率を監視することにより観測している．

© 2008 Information Processing Society of Japan

表 3　予備実験から得られたパラメータ
Table 3　Learned model parameters using either 512-length or 8192-length profile runs.

予備実験サイズ	8,192	512
K_{trans}	1.47×10^{-8}	1.11×10^{-8}
K_{gMa}	6.93×10^{-11}	5.95×10^{-10}
K_{c2g}	5.92×10^{-9}	6.16×10^{-9}
K_{g2c}	5.09×10^{-9}	5.68×10^{-9}
K_{gP}	2.83×10^{-10}	2.73×10^{-11}
K_{gDP}	2.46×10^{-9}	7.28×10^{-12}
K_{cFFT}	3.21×10^{-9}	2.61×10^{-9}
K_{gFFT}	1.88×10^{-10}	1.65×10^{-10}
K_{gMf}	6.08×10^{-10}	3.01×10^{-9}

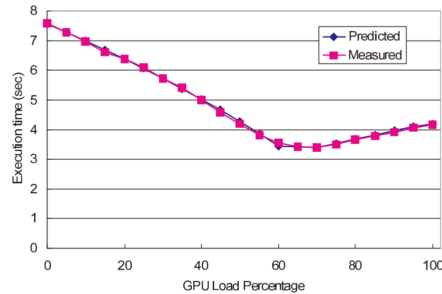

図 9　モデルから推定した実行時間と実際の実行時間の比較（CUFFT 版，問題サイズ 8,192，モデルは 8,192 のデータからパラメータを取得）
Fig. 9　Comparison of the real performance of 8192-length 2D-FFT with the predicted performance using profile runs with the same data size.

図 9 に，CUFFT 版，問題サイズ 8,192 の実行時間の実測値と予測値を示す．このときの性能モデルのパラメータは，同じ問題サイズ 8,192 による予備実行から決定した．この場合，我々のモデルは非常に正確に実行時間を予測していることが分かる．予測実行時間の誤差は最大で 6.9%，多くの場合は 5%未満である．また，最適な GPU への割当て率である 70%を予測できた．本稿では割当て率を 5%刻みとしたため，これは最適割当て率を 5%以内の精度で予測したことを意味する．

次に，小さい予備実験から得られたパラメータで，大きい問題サイズの性能を予測した場合を**図 10**，**図 11** に示す．図 10，図 11 では，問題サイズ 512 で予備実験を行い，問題サイズ 1,024 および 8,192 の場合を予測した結果を示す．

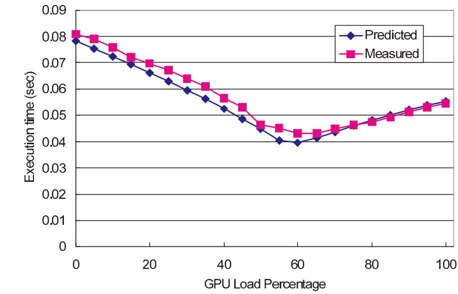

図 10　モデルから推定した実行時間と実際の実行時間の比較（CUFFT 版，問題サイズ 1,024，モデルは 512 のデータからパラメータを取得）
Fig. 10　Comparison of the real performance of 1024-length 2D-FFT with the predicted performance using 512-length profile runs.

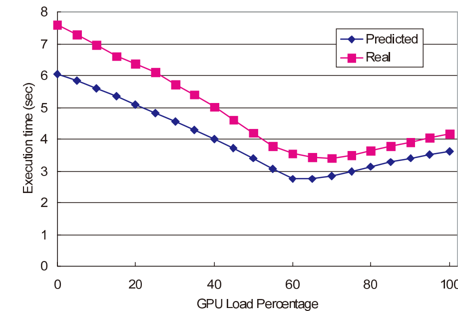

図 11　モデルから推定した実行時間と実際の実行時間の比較（CUFFT 版，問題サイズ 8,192，モデルは 512 のデータからパラメータを取得）
Fig. 11　Comparison of the real performance of 8192-length 2D-FFT with the predicted performance using 512-length profile runs.

図 10 の場合は，実行時間を平均で 2%程度の誤差，最大誤差で 6%程度で予測している．また，最適割当て率である 60%の予測に成功している．一方図 11 の場合は，実行時間を少なく見積もってしまう傾向にあり，最大 15%程度の誤差が出ている．

この誤差の原因を調査するため，実行時間の内訳および表 3 で示されたパラメータを比較した．**図 12** に問題サイズ 8,192 の実測値の内訳と，512 から得られたパラメータを用いて 8,192 の場合を予測した予測値の内訳を積み上げ棒グラフとして示した．グラフの下から，実行単位が実行順に示され，点線はスレッド間同期を示す．図 12 から，予測と実測の

© 2008 Information Processing Society of Japan

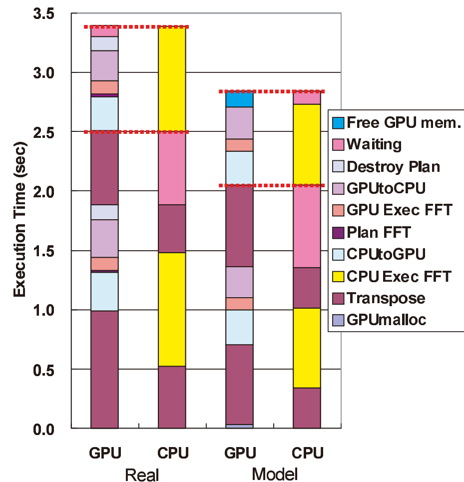

図 12　問題サイズ 8,192 の実際の実行の内訳と 512 から得られたパラメータで 8,192 を予測した内訳の比較（CUFFT 版，GPU 割当て率 70%）
Fig. 12　Breakdown comparison of the real performance of 8192-length 2D-FFT with the predicted performance using 512-length profile runs.

差に大きな要因は転置処理と CPU 側 1D-FFT であることが分かる．転置については，表 3 でも問題サイズ 512 から得たパラメータと 8,192 から得たパラメータを見ても K_{trans} の差が大きい．すでに係数 M_{trans} の導入により並列性能の特性を考慮したが，現在は不完全であり，より正確なモデルが必要である．たとえば，転置するデータ量がキャッシュに収まる場合と収まらない場合の差について考慮が必要であると考えており，将来の課題の 1 つである．

次に，CPU 側 1D-FFT の誤差は，以下の原因が推測される．

- CPU 側の前処理（Plan 作成等）を FFT 処理と同一実行単位に含めている．これらを GPU 側と同様に分離することにより改善が可能と推測している．
- 予備実行のサイズが小さいため，本実行とキャッシュミス率が異なり，その影響も考えられる．
- 図 12 の場合には，CPU 側 1D-FFT と，GPU コントロールスレッドによる転置が並列に実行されることがありうるため，メモリバンド幅の影響を考慮する必要があると考えられる．

なお，表 3 中では，K_{trans}，K_{cFFT} 以外にも差の大きいパラメータが見られるが，実行時間への影響については，上記の 2 つよりは小さい．ただし，その中では GPU の後処理や GPU 上のメモリ解放についての誤差が大きく，それらについてもモデルの改善を行う予定である．

以上の問題はあるが，それに起因する最適割当て率の誤差は比較的小さく，実際の 70%の代わりに 65%が得られた．なお，予測された割当て率を採用して実行を行った場合，本来の最適実行時間に比べた性能低下は 1%以内であった．

最後に，最適分割率を決定するためにかかる時間を述べる．図 11 の場合，実際の実行が 3.5 秒程度であるのに対し，予備実行は計 0.032 秒であった．この予備実行を行うことで，CPU2 スレッドで実行する場合と比較して，1.78 秒の実行時間短縮が可能であり，この場合はパラメータ決定を行うコストを払っても，合計実行時間は得になる．

6. まとめと今後の課題

本研究では，GPU と CPU を併用する FFT ライブラリを提案し実装した．またそのライブラリについて性能モデルを構築し，それを用いて CPU と GPU への最適な割振り率を予測した．複数の問題サイズを用いた実験により，最適な CPU と GPU への割振り率を 5%以内の誤差で予測することを示した．小さい問題サイズで予備実験を行い，それによるモデルから割振り率を得て実行を行った場合，本来の最適実行時間と比較して 1%以内の性能低下率であった．また，そのときの予備実験の時間は実際の計算時間の 100 分の 1 以下であった．

今後の課題は以下のとおりである．まずあげられる課題は，性能モデルの精度の向上である．現状の性能モデルは，主にデータ転置時間の予測精度に改善の必要があるため，キャッシュミスの影響等を取り入れる予定である．また，現在のところモデルは CPU 側計算を 1 スレッドで行い，GPU は 1 つである場合にのみ対応している．より多数のスレッドによる計算や，複数の GPU，さらには異種の GPU からなる環境にも対応できるよう性能モデルを改良していく．なお本研究の実験では 2 コア計算機で 2 スレッドを計算に用いた場合の性能が想定より低くなった．これはグラフィック API が CPU を占有しているためと推測しているが，その問題が軽減されている新しいバージョンの CUDA での性能評価および，モデルの対応と検証を行う．また，一般的な問題サイズにおいてさらなる高速化を行うために，CUDA の特徴を活かしたライブラリの改良を行う．たとえば GPU 側でのデータ転置を取り入れることで，転置自体の高速化と通信削減が可能となると考えられる．今後，それ

© 2008 Information Processing Society of Japan

らの手法を取り入れた場合の，GPU/CPU 併用の有用性をモデルと実験から調査していく予定である．最後に，今回は 2D を対象としたが，実際の応用では 3D がより重要である．3D の場合は必要とされるメモリ量が増大する傾向にあるため，本ライブラリのように大きな問題サイズへ対応することはより有用であると期待される．また，2D-CUFFT を内部で用い，さらに CPU の併用を行うことにより，効率的に大規模な 3D FFT を実現できると考えられる．

謝辞　本研究の一部は JST-CREST「ULP-HPC：次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」および Microsoft Technical Computing Initiative の援助による．

参　考　文　献

1) GPGPU.org. http://gpgpu.org/
2) NVIDIA Forums. http://forums.nvidia.com/
3) Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M. and Hanrahan, P.: Brook for GPUs: stream computing on graphics hardware, *ACM Trans. Graph.*, Vol.23, No.3, pp.777–786 (2004).
4) ClearSpeed Technology plc.: ClearSpeed white paper: CSX processor architecture. http://www.clearspeed.com/
5) Crandall, P.E. and Quinn, M.J.: Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network, *Proc. IEEE International Symposium on High Performance Distributed Computing* (*HPDC*), pp.42–49 (1993).
6) Frigo, M. and Johnson, S.G.: The design and implementation of FFTW3, *Proc. IEEE*, special issue on Program Generation, Optimization and Platform Adaptation, Vol.93, No.2, pp.216–231 (2005).
7) Galoppo, N., Govindaraju, N.K., Henson, M. and Manocha, D.: LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware, *SC '05: Proc. 2005 ACM/IEEE conference on Supercomputing*, p.3, Washington, DC, USA, IEEE Computer Society (2005).
8) Govindaraju, N.K. and Manocha, D.: GPUFFTW: High Performance Power-of-Two FFT Library using Graphics Processors, http://gamma.cs.unc.edu/GPUFFTW/
9) Moreland, K. and Angel. E.: The FFT on a GPU, *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003*, pp.112–119 (July 2003).
10) Makino, J., Kokubo, E. and Fukushige, T.: Performance evaluation and tuning of GRAPE-6 — towards 40 "real" Tflops, *SC '03: Proc. 2003 ACM/IEEE conference on Supercomputing*, p.2, Phoenix, AZ (2003).
11) NVIDIA Corporation: NVIDIA CUDA Compute Unified Device Architecture Programming Guide. http://developer.nvidia.com/cuda
12) 伊藤信悟，伊野文彦，萩原兼一：GPGPU アプリケーションの開発を支援するための性能モデル，先進的計算基盤システムシンポジウム SACSIS2007 論文集，pp.27–34 (May 2007).
13) 大島聡史，吉瀬謙二，片桐孝洋，弓場敏嗣：CPU と GPU を用いた並列 GEMM 演算の提案と実装，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG12(ACS 15), pp.317–328 (2006).
14) 尾形泰彦，遠藤敏夫，松岡　聡：CPU および GPU を併用する高効率の FFT ライブラリ，先進的計算基盤システムシンポジウム SACSIS2007 論文集ポスターセッション，pp.175–176 (May 2007).

(平成 19 年 10 月 9 日受付)
(平成 20 年 1 月 31 日採録)

尾形　泰彦（学生会員）
1984 年生．2007 年東京工業大学理学部情報科学科卒業．現在，同大学大学院数理計算科学専攻修士課程在学中．主に，GPU を含むアクセラレータの HPC への利用に興味を持つ．

遠藤　敏夫（正会員）
1974 年生．2001 年東京大学大学院理学系研究科情報科学専攻博士課程修了．博士（理学）．東京大学情報理工学系研究科特任助手，東京工業大学学術国際情報センター特任講師等を経て，2007 年より東京工業大学グローバル GCOE「計算世界観の深化と展開」特任准教授．主に分散処理や不均一型アーキテクチャ上での並列計算の研究に従事．日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

© 2008 Information Processing Society of Japan

丸山　直也（正会員）

2008 年東京工業大学数理計算科学専攻博士課程修了．2008 年 4 月より東京工業大学学術国際情報センター産学官連携研究員．並列・分散コンピューティング，プログラム最適化に関する研究に従事．理学博士．ソフトウェア科学会会員．

松岡　　聡（正会員）

1986 年東京大学理学部情報科学科卒業，1989 年同大学大学院博士課程から，学情報科学科助手に採用，同大学情報工学専攻講師を経て，1996 年に東京工業大学情報理工学研究科数理・計算科学専攻助教授．2001 年 4 月に東京工業大学学術国際情報センター教授，2002 年より国立情報学研究所の客員教授を併任．博士（理学）（東京大学）．高性能システム，並列処理，グリッド計算，クラスタ計算機，高性能・並列オブジェクト指向言語処理系等の研究に従事．わが国のナショナルグリッドプロジェクトである NAREGI プロジェクトのサブリーダーであり，また 2006 年時点でわが国最高性能のスーパコンピュータ TSUBAME を構築．1996 年度情報処理学会論文賞，1999 年情報処理学会坂井記念賞 2006 年学術振興会賞（JSPS Award）等を受賞．ISOTAS’96，ECOOP’97，ISCOPE’99，ACM OOPSLA’02，IEEE CCGrid’03，HPC Asia 05，Grid2006 等のプログラム（副）委員長 IEEE/ACM Supercomputing’04–Network Track Chair，Reflection’01，IEEE CCGrid’06 大会委員長．また，グリッド国際標準化団体の Global Grid Forum の Steering Group 委員等を務める．

© 2008 Information Processing Society of Japan