CPU および GPU を併用する FFT ライブラリの提案と評価

尾形泰彦[†] 遠藤敏夫[†] 松岡聡[†,††]

General-purpose GPU (GPGPU) を科学技術計算のために利用する手法が注目されている。GPU は非常に高速な並列計算を行うことが可能で、かつ価格性能比が非常に高いという特徴を持つ。一方で、CPU のマルチコア化も近年急速に進みつつあり、マルチコア CPU と GPU という、性能特性の異なる 2 つの並列計算リソースが一つの計算機内に存在するようになってきた。このため、計算機を有効に利用するためには両者をより効率よく併用する必要があり、このヘテロな計算環境への計算量の割り振りを行う必要が出てきている。我々は、このヘテロな計算環境への計算量の割り振りについて、性能モデルを用いて予測することを提案する。この性能モデルは、問題のサイズに対する計算量やデータ量等を元に構築し、実測値からパラメータを与える。また、この性能モデルを用いることで、全体の計算量および GPU と CPU への各割り振り率に対する実行時間を予測する。この予測した実行時間の最小値を探索することで最適割り振り率を得る。本提案の実証のために GPU と CPU を併用する 2D-FFT ライブラリを実装し、同 FFT ライブラリに対する性能モデルの構築を行った。同ライブラリと性能モデルを用いた実行性能の予測値と実測値との比較を行い、性能モデルの正確さを評価した。その結果、予備実行の 16^2 倍までの計算量の問題について、最適割り振り率を高々5%の誤差、実行時間も 15%程度の誤差に抑えられ、この性能モデルが十分な予測精度を持つことを確認した。

Proposal and Evaluation of a FFT Library That Uses CPU and GPU Together

Yasuhiko Ogata ,[†] Toshio Endo [†] and Satoshi Matsuoka[†,††]

General purpose computation on graphics processing units (GPGPU) is becoming popular in HPC field, in expectation of excellent peak performance of GPUs. Their effective performance is, however, not so far from that of recent multi-core CPUs. Therefore we can expect to improve performance by using GPUs and CPUs cooperatively. One of the key challenges in such heterogeneous environments is to determine optimal load balancing ratio among processors. It depends not only on characteristics of target computation and processors, but also on problem sizes. Our approach is to construct a performance prediction model that covers computational cost and data transfer cost of target computation. We train the model with a small number of test runs to determine model parameters. Then we use the model to obtain optimal load balancing ratio for arbitrary problem sizes. According to this approach, we have implemented a two-dimensional FFT library for heterogeneous environments and constructed its performance model. We have evaluated accuracy of our model by comparing prediction and real performance on a heterogeneous system with a GeForce8800GTX GPU and a Core2Duo CPU. After training the model with test runs of 512^2 FFT, we have evaluated larger (up to 8192^2) problem sizes. The results show that our model succeeds to predict the optimal load balancing ratio within 5% accuracy, while prediction errors in execution time are 15% or less.

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)[3] と呼ばれる手法が注目されている。最新の CPU のピーク性能が 25GFLOPS 程度であるのに対して、近年のハイエンド GPU は 500GFLOPS を誇る。また、GPU は値段が安く、ClearSpeed[4] などの SIMD アクセラレータよりも非常に低コストで性能を向上させることが可能である。

† 東京工業大学 (Tokyo Institute of Technorogy)
†† 国立情報学研究所 (National Institute of Informatics)

CPU と GPU を併用するためには、複数の CPU コアと GPU が存在するヘテロな環境において、各々のプロセッサへの計算量割り振りの予測が必要である。これまで多くの GPGPU の研究がなされてきたが、ほとんどが GPU 上のアルゴリズムの提案や、GPU 一台での性能に重点を置いている。[7][8] しかし、実効性能は GPU と CPU の間の差はそれほど大きくない。現在の GPU は、CPU と GPU の通信オーバヘッドなどのため、実効性能はピーク性能の 10%以下であることも珍しくない。一方で CPU においてはマルチコア化が急速に進みつつあり、多くの科学技術計算はそれらを有効利用できる。このような現状において、いずれか一方のみ用いるよりも、両方を併用する方が高

性能を引き出すことができる。

我々は、この計算量の割り振り問題を、性能モデルを構築し、各プロセッサの特性に応じた最適な割り振り量を推定することを提案する。その第一歩として本論文では、二次元高速フーリエ変換 (2D-FFT) 計算を対象とし、CPU および GPU を併用するライブラリを実装し、そのライブラリに対する性能モデルを構築する。また、構築した性能モデルを用いて、実行特性の予測を行い、実際の実行時間と比較した。その結果、小さい問題サイズでの 2D-FFT の実行から得られたパラメータを用いて、その 16^2 倍の問題サイズの場合について最適割り振り率を予測した。その際、最高性能を得られる割り振り率を 5%以内の精度、また実行時間も 15%程度のずれに抑えることに成功した。

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[8] などのアルゴリズムが提案されている。その多くは GPU 一台での性能に注目している。その中で大島ら[10] 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 GPU と CPU に計算領域を分割する。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、本研究ではマルチコア 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きている。Buck ら[5] は、グラフィックの知識なしに GPGPU を行うことが出来る言語処理系 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おうとしている他、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[9] の研究が挙げられる。これは、実際の実行時間からパラメータを決定するのではなく、メモリバンド幅等のパラメータを用いて性能予測を行う。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が、正しく計算量と通信量を見積もるためにはコードの詳細設計が必要になると考えられる。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するものではない。

3. GPU と CPU を併用する 2D-FFT ライブラリ

我々は、2 次元の FFT について CPU と GPU を併用することが出来るライブラリを実装し、測定およびモデル化のサンプルとして利用した。

2 次元の FFT は、X/Y 各軸方向に 1D-FFT により計算が可能である。このアルゴリズムは Row-Col 法と呼ばれる。計算機内ではデータは 1 次元配列で表されており、2 次元の配列にアクセスする場合、特定の軸に関しては連続なアクセスとなるが、別の軸では離散したアクセスとなる。典型的な FFT ライブラリでは入力に連続データを要求する。このため、前処理として転置が必要となる。これらにより、2 次元の FFT は X/Y 各軸方向への 1D-FFT と軸方向を修正するための転置に分解される。

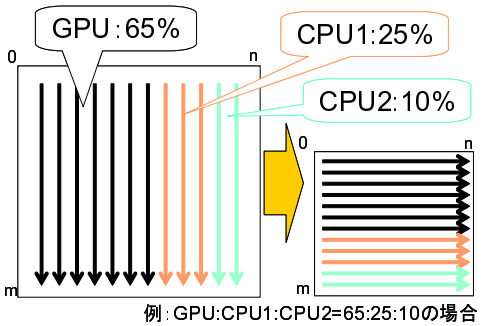

図 1 GPU/CPU への 1D-FFT の分割

我々は、Row-Col 法を用いて分解した 1D-FFT を、CPU 用および GPU 用の汎用な 1D-FFT ライブラリ各々に割り当て、CPU と GPU への計算量の分割を行う。また、現在の GPU 上での実行が単精度であるという制約により、本ライブラリは単精度のみの実装である。

3.1 本ライブラリで用いた 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[2] および CUDA[1] で提供された FFT ライブラリを用いた。CPU 側および GPU 側の各 FFT ライブラリは差し替えることが可能である。

FFTW[6] 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[2] 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、NV_fragment_program を用いている。

CUDA (Compute Unified Device Architecture)[1] 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。なお、今回は用いることは出来なかったが、現在は CUDA ver1.0 がリリースされている。CUDA ver1.0 では、GPU 上のプログラム実行時に CPU 時間を消費しない CPU と GPU の

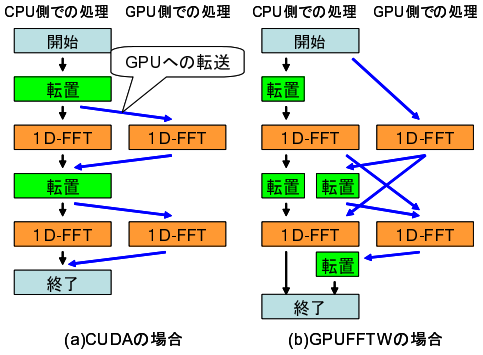

図 2 実行の流れ

非同期な実行が行える。今回は主に CUDA 上の FFT ライブラリである CUFFT ライブラリを使用した。

3.2 本ライブラリの構成

図 2 に本ライブラリにおけるデータフローを示す。本ライブラリでは、先に列方向の 1D-FFT を行った後、行方向の 1D-FFT を行う。今回用いたライブラリのうち、FFTW および、CUFFT は Row-major のデータ並び順を要求し、GPUFFTW は Col-major のデータ順を要求する。このため、GPUFFTW を利用する図 2(b) では、CPU 側行方向の前後、GPU 側では列方向の前後と転置の順序が異なり、一方で、図 2(a) に示されている CUDA 版では、転置は行方向の前後に行われている。

転置は、CPU 上のみで行い、CPU 上の FFT 実行を管理するスレッド、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 CPU への 1D-FFT の割り当て率、GPU への 1D-FFT の割り当て率と同じ割合で転置を振り分ける。

本ライブラリでは、GPU 側に GPU メモリサイズ以上のデータ量の FFT を割り振る場合、GPU メモリに収まるサイズに調整した数の 1D-FFT を複数回呼び出す。これは、GPU のメモリサイズ制限以上のデータを扱うためである。GPU のメモリは総じて CPU のメモリよりも少ない傾向にある。一般に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高々 512MB 程度である。しかし、CPU 側に搭載されているメインメモリは 4GB 以上積まれていることが多い。また、一般的な GPGPU においては、計算を、テクスチャメモリから読み出してフレームバッファメモリに書き込むことで行う。このため、これらそれぞれにメモリ領域を割り付ける必要があり、実際にデータを格納できる容量はさらに減る。以上により、2D-FFT をすべて GPU 上で計算しようとした場合、GPU が一度に計算可能な行列のサイズは CPU と比較して 10 分の 1 以下になる。本ライブラリでは GPU 側に 1D-FFT 計算のみを配分したことで、GPU 単体では計算不可能な大きさの問題を解くことを可能とした。

CPU 側の実装では、複数スレッドに 1D-FFT を自由な配分で割り振れるように設計した。これは、GPU 側のライブラリが計算中に、アイドル状態の CPU パワーを生かすためである。GPU 側のコントロールスレッドが使用する CPU コアと、そうでない CPU コアで、計算量を変えられるようにした。

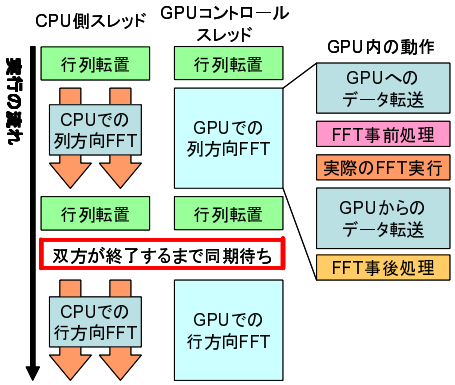

図 3 実行フローの詳細 (CUDA 版)

4. 本ライブラリの性能モデルの構築

本稿で構築した性能モデルは、ライブラリを細かい実行単位に切り分け、各実行単位に対してメモリアクセス量・計算量から問題サイズ n との関係を予測し、実測値から係数を与えることで構築を行う。例えば、GPU へデータを転送する際には、送信するデータ量に比例する時間が掛かることが予想され、($T = K * n^2$,T は転送時間,K は係数,n は問題サイズ) と実行時間をモデル化する。

図 3 に、本ライブラリの実行フローを、今回モデル化した実行単位ごとに示す。この実行単位ごとの分割は、ライブラリ内部の関数呼び出し等を基準に分割し、この中で全体の実行時間に特に大きな影響を与える 9 種類をモデル化の実行単位として扱う。実行単位のパラメータを、表 1 に示す。

図 3 では、GPU 内の動作が 1 セットしか書かれていないが、実際の GPU 側の動作は GPU で可能なサイズの 1D-FFT のセットを複数回繰り返す。しかし、FFT の実行時間は 1D-FFT は本数に対して線形に増加し、かつそれを複数回行った場合も同じ増加率で実行時間が増加する。このため、今回行ったモデル化では繰り返す回数については考慮せず、問題サイズのみ考慮したモデルとした。以下に実際のモデル式を示す。

$$T_{total} = max(T_{gpu1}, T_{cpu1}) + max(T_{gpu2}, T_{cpu2}) \quad (1)$$

(1) の変数は、T_{total}：全体の実行時間,T_{gpu1}, T_{gpu2}：GPU 側列/行各方向の実行時間、T_{cpu1}, T_{cpu2}：CPU 側列/行各方向の実行時間である。本ライブラリでは、FFT を計算する軸を切り替える際に同期を取る必要があり、その同期の前後の時間についてそれぞれ GPU 側/CPU 側の実行時間の最大値を取り、それらの合計時間を全体の時間とした。

次に、$T_{gpu1}, T_{gpu2}, T_{cpu1}, T_{cpu2}$ の内訳を示す。数式内の n は問題サイズ、k は GPU 側への割り当て率である。($0 \leq k \leq 1$) また、各パラメータに関しては表 1 に示す。

$$
\begin{aligned}
T_{gpu1} &= kn^2(K_{gMa} + K_{c2g} \\
&\quad + K_{gP} + K_{gDP} + K_{g2c}) \\
&\quad + kn^2 * M_{trans} * (K_{trans} * 2) \\
&\quad + kn^2 log(n) K_{gFFT} \qquad (2) \\
T_{gpu2} &= kn^2(K_{c2g} + K_{gP} \\
&\quad + K_{gDP} + K_{g2c} + K_{gMf}) \\
&\quad + kn^2 log(n) K_{gFFT} \qquad (3) \\
T_{cpu1} &= (1-k)n^2 * M_{trans} * (K_{trans} * 2) \\
&\quad + (1-k)n^2 log(n) K_{cFFT} \qquad (4) \\
T_{cpu2} &= (1-k)n^2 log(n) K_{cFFT} \qquad (5) \\
M_{trans} &= (-0.7 * |k - 0.5|/0.5 + 1.7) \qquad (6)
\end{aligned}
$$

表 1 性能モデルのパラメータ

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
M_{trans}	転置係数

行列転置は 2CPU で実行した場合には実性能が 2 倍にはならず、約 1.2 倍程度の性能となる。このため、並列実行時は係数 M_{trans} を掛ける事で、約 1.2 倍程度の性能となることを再現した。

この実行性能のモデルは CPU 側 GPU 側ともに、多くの部分では実行時間が問題サイズに依存するとした、粒度が荒い近似である。しかし、この単純なモデルでも実行時間を高い精度で予測可能である。このモデルを用いた予測結果は 5.2 にて検証した。

5. 評　　価

本章では、ライブラリの実行性能とモデル化の精度について評価を行った。また、改善案を提示し、それらについての性能予測を行った。

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 の環境で行った。測定は、一辺の要素を 256〜8192 とする 2 次元複素数行列の FFT について実行時間を測定した。

図 4 に、問題サイズに対する GPUFFTW 版、CUFFT 版の最適分割率付近での実行時間、及び比

表 2 評価環境

CPU	Core 2 Duo E6400(2.13Ghz*2)
memory	PC6400 4GB
HDD	250GB
GPU	Geforce 8800GTX
OS	Linux kernel2.6.18
Driver	NVIDIA Display Driver ver97.46
	CUDA ver0.8

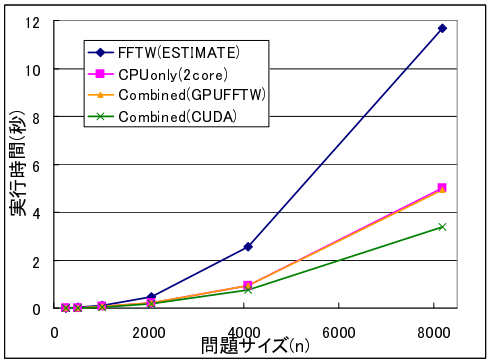

図 4 問題サイズに対する実行性能

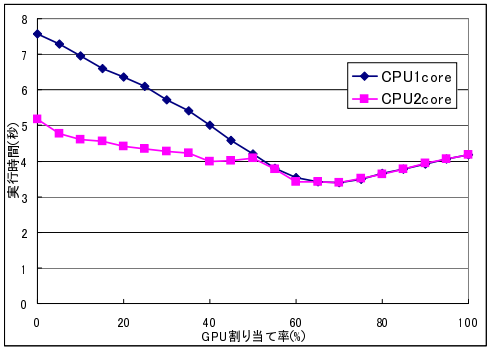

図 5 GPU 割り当て率に対する実行性能 (CUFFT 版)

較のために FFTW および本ライブラリを CPU のみで用いた場合の実行時間を示す。CUFFT 版で問題サイズが 8192 の場合、FFTW と比較して 3.5 倍弱の性能向上、本ライブラリを CPU のみで使用した場合と比較して 1.5 倍の性能向上となった。

図 5 に CUFFT 版について、問題サイズ 8192 の場合の GPU 側への割り当て率の変動と実行時間の関係のグラフを示す。図 5 では、GPU 割り当て率が 0%の場合はすべての計算を CPU 上で行っていることを表し、100%の場合はすべての計算を GPU で行っていることを示す。全体の性能は、GPU 側への割り当てが 70%付近で最高性能を引き出すことができ、CPU と GPU を併用したことで、CPU 上で FFT を 1 スレッドで計算を行った場合に対して 2.2 倍、CPU2 スレッドで計算を行った場合に対して 1.5 倍、GPU のみで行った場合に対して 1.2 倍の性能を引き出せた。

GPU への割り当て率が、最適性能に近くなる場合、CPU を 1 コアで実行した場合と 2 コアで実行した場合では性能に差が存在しない。これは、CPU の 2 スレッド目の計算配分量を変更しても同様である。この原因は、GPU を用いる際、GPU コントロールスレッドが CPU の計算能力を消費するためである。GPU への割り当て率が最適値付近を超える場合は、GPU コントロールスレッドにより、1 コア分の CPU 時間が消費された。このため、CPU 側の 1D-FFT を各 CPU コア毎に分割したことによる性能向上は得られ

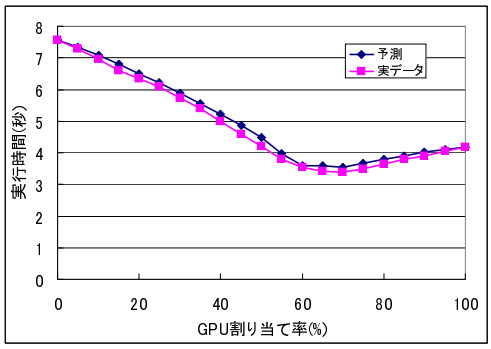

図 6 モデルと実性能の比較 (CUFFT 版,8192to8192)

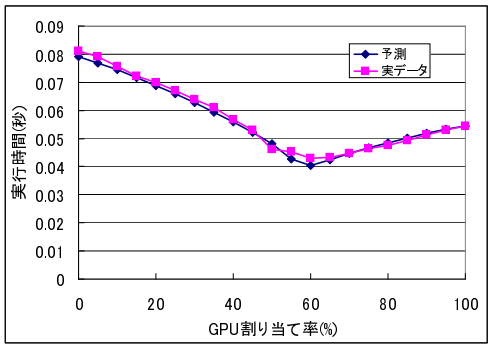

図 7 モデルと実性能の比較 (CUFFT 版,512to1024)

なかった。

5.2 性能モデルの検証

まず、図 5 で示された CUFFT 版、問題サイズ 8192 の場合について性能モデルから性能の推定を行う。図 5 で示した通り、CPU 側を 2 スレッド用いた場合でも性能が向上しない。このため、CPU 側 1 スレッドの場合のみ予測できれば、最適な GPU と CPU の分割率を決定出来る。

図 6 に CUFFT 版、問題サイズ 8192、CPU 側が 1 スレッドの場合の実測の実行時間と、性能モデルから予測した実行時間を示す。この性能モデルのパラメータは、同じ問題サイズの CPU 側に FFT をすべて割り振った場合と GPU 側に割り振った場合の 2 点から決定した。図 6 では、最適な GPU への割り当て率を予測できている。また、予測した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。

次に、異なる問題サイズから予測する場合を図 7 に示す。図 6 では、同じ問題サイズで GPU および CPU のどちらか片側に全て割り振った場合の実測値からモデルのパラメータを決定したが、図 7, 図 8 では、問題サイズ 512 の GPU および CPU にすべて割り振った場合の実測値からパラメータを決定し、予測を行った。

図 7 は、問題サイズが 1024 の場合の実測値と予測値を重ねて示したものである。図 7 では、最適分割率は 5%刻みでデータを予測した場合には完全に一致し、それより細かい粒度で予測を行った場合にも 5%以内の精度で予測できている。予測実行時間の誤差は平均

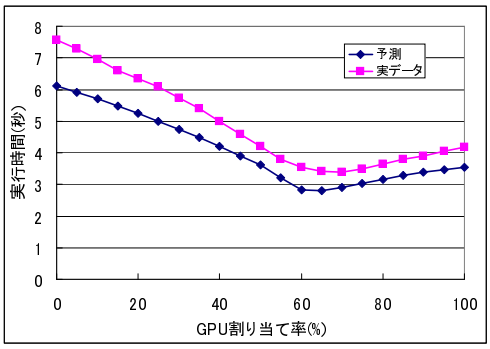

図 8 モデルと実性能の比較 (CUFFT 版,512to8192)

で 2%程度、最大で 6%以下となっている。図 7 の場合は問題サイズが 2 倍になった場合の予測であるが、問題サイズが 2 倍になることで、実際の計算時間は 4 倍以上になるため、このように 2 分の 1 の問題サイズから非常に精度の良い予測が出来ることは重要である。

図 8 に、図 7 と同じパラメータを用いて問題サイズが 8192 の場合の実測値と予測値を示す。図 8 の場合は、実行時間を全体として少なく見積もってしまっているが、最適分割率は 5%程度の誤差で予測できる。

この実行時間のずれは、主に CPU 側の予測が低く予測してしまうためである。特に、CPU 上で行っている転置の予測のずれによる影響が大きく、この転置のずれにより、CPU が律速の場合/GPU が律速の場合が共に少なめに見積もっている。このずれの原因としては、何らかの理由によりモデルのパラメータが跳ね上がることがあることで、特に CPU 上の転置が顕著である。しかし、このパラメータの上昇は、最大でも 20%程度の上昇であり、これによって予測した最適分割率が受ける影響は軽微である。本稿における性能モデルの主な目的は、GPU と CPU 間での最適分割率の推定であり、この最適分割率の推定については本稿で用いた非常に単純なモデルでも十分に推定可能であることを示す。

このモデル化による最適分割率の予測は、図 8 の場合、実際の実行が 3.5 秒程度の計算を行うために、問題サイズ 512 の場合における、すべてを CPU または GPU で行った 2 回の実行時間、計 0.032 秒の事前実行で最適分割率が決定できている。この計 0.032 秒の事前実行を行うことで、CPU2 コア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る。

5.3 GPU 上で転置を行った場合の性能予測

今回の実装では、転置をすべて CPU 上で行った。GPUFFTW 等一般的な GPGPU では計算可能なデータ量が実際の GPU メモリよりも少ないため、転置を GPU 上で行うことが現実的ではない。また、GPUFFTW のライブラリ内に CPU へのデータ転送等が含まれており、転置を GPU 上に実装を行うとライブラリを分解する必要がある。このため、転置を GPU 上で行う場合、ライブラリ単位で差し替えることを可能にするという用件を満たせない。

しかし、CUDA は GPU メモリとして実装されてい

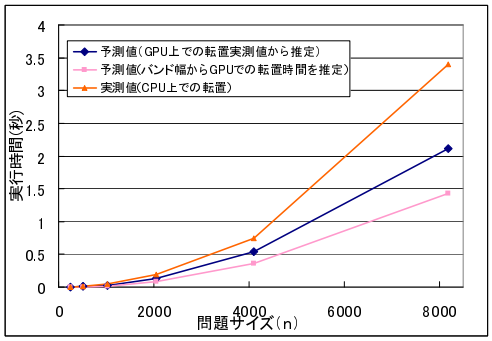

図 9 GPU 上で転置を行う場合の性能予測

るメモリをすべて利用することが可能である。また、FFT ライブラリ自体とメモリ管理部分は切り離されている。このため、CUDA を用いることで GPU 上で転置を行うのにデータ制限を受けにくく、ライブラリ自体には手を入れずに実装することが可能となる。

そこで、GPU 上で転置を実行した場合の実行時間に対して性能モデルを構築し、性能を予測した。GPU 上での転置以外の部分は 5.2 で用いたパラメータを利用し、GPU 上での転置部分に関しては GPU 上での転置を仮実装し、そこから予測されたパラメータを用いた。しかし、この仮実装の転置の性能は、メモリバンド幅等の GPU のスペックを大きく下回る性能となった。このため、メモリバンド幅から予測した転置の実行時間を用いた実行予測時間も併記する。

図 9 に問題サイズに対応する FFT ライブラリ全体の実行時間の最適値を、GPU 上で行った転置の実測時間をパラメータに用いた FFT ライブラリ全体の予測実行時間、GPU のメモリバンド幅から予測される転置実行時間をパラメータに用いた FFT ライブラリ全体の予測実行時間、および CPU 上の転置を行った場合の FFT ライブラリ全体の実行時間の実測値を示す。CPU 上で転置を行うことと比較して、GPU 上の転置の実測値をパラメータに用いた場合、1.6 倍程度性能が向上する。また、GPU のメモリバンド幅から推定したパラメータを用いた場合、CPU 上で転置を行った場合と比較して 2.3 倍の性能向上が予測される。

このように、性能のモデル化を行ったことでプログラムの挙動に修正を加えようとする際に、修正後のプログラムの挙動を予測することが出来る。

6. まとめと今後の課題

本稿では、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、性能のモデル化を行い、構築した性能モデルを用いて、CPU と GPU への最適な割り振り率を予測する方法の提案を行った。構築した性能モデルを用いることで、最適な CPU と GPU への割り振り率を最大で 5%程度の誤差で予測することが可能で、この性能モデルのパラメータを求めるための事前計算は実際の計算時間の 100 分の 1 の実行時間で済むことを示した。また、ライブラリ内の挙動を変更した場合の実行時間も予測した。

今後の課題は、このモデルを利用した自動チューニング機構を作成することである。また、転置を実際に GPU 上で行った場合の性能を測定し、予測と合致するかどうか検証する。さらに、今回転置を CPU と GPU に分割することを考えなかったが、転置も CPU と GPU に分割するとどのような実行性能が得られるかを予測し、実際の結果と合致するか検証することを予定している。

謝辞 本研究の一部は科学研究費補助金特定領域研究 (18049028) の補助による。

参 考 文 献

1) http://developer.nvidia.com/object/cuda.html.
2) http://gamma.cs.unc.edu/gpufftw/.
3) http://gpgpu.org/.
4) http://www.clearspeed.com/.
5) Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, Vol. 23, No. 3, pp. 777–786, 2004.
6) Frigo, et al. The design and implementation of FFTW3. *Proceedings of the IEEE*, Vol. 93, No. 2, pp. 216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".
7) Nico Galoppo, Naga K. Govindaraju, Michael Henson, and Dinesh Manocha. LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, p. 3, Washington, DC, USA, 2005. IEEE Computer Society.
8) Moreland K and Angel E. The FFT on a GPU. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings*, pp. 112–119, July 2003.
9) 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.
10) 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.