社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－164 (24)
2005／8／4

CPUとGPUの並列処理による行列積和演算方式の提案

大島聡史† 吉瀬謙二†
片桐孝洋† 弓場敏嗣†

数値計算にGPU(Graphics Processing Unit)を利用する研究が盛んになりつつある．本稿ではGPUを用いてより高い演算性能を得るため，GPUで数値計算をおこなうだけではなく，対象問題を分割し，CPUとGPUで並列に問題を解く方式を提案する．行列の積和演算をCPUとGPUで並列実行した結果，CPUのみで解いた場合に比べて，最大38.1%の性能向上が得られた．また，予備実験によって得たCPUとGPUそれぞれのFLOPS値を利用し，最適な問題配分をおこなうための方法を検討した．その結果，実験による測定値に近い値を，計算によって求めることができた．

Proposal of Matrix Multiply and Add Method by Parallel Processing Using CPU and GPU

SATOSHI OHSHIMA,† KENJI KISE,† TAKAHIRO KATAGIRI† and TOSHITSUGU YUBA†

A research that uses GPU for numerical calculation is becoming active. In this paper, we do not only solve numerical problem using GPU but also propose a method that divide a problem and calculate it using CPU and GPU. Measure a execution time of matrix multiply and add, and because of parallel processing the performance was improved of 38.1% than the case solved only with CPU. Moreover, we exmine a method for a best problem distribution using each FLOPS of CPU and GPU obtained by a preliminary experiment. As a result, we could obtain values by prediction nearby experimental values.

1. はじめに

画像処理ハードウェアであるGPU(Graphics Processing Unit)の性能向上が進んでいる．GPUはCPUと比べて並列処理やベクトル処理に適したハードウェア構成を持っており，画像処理以外の様々な処理ができるように進化している．この機能を利用して，汎用演算(General-Purpose computation)への活用が模索されている．GPUを用いた汎用演算はGPGPU(General-Purpose computation on GPU's)と呼ばれており，様々な用途に向けた研究がおこなわれている[1]．

一方，数値計算の分野では常に高速な計算環境が求められる．例えば，数値計算において良く利用される重要な演算の一つに行列の積和演算がある．CPUにおける高速な行列の積和演算の実装としては，ATLAS[2]の提供するものが有名である．ここではCPUの性能を効率良く引き出すことに成功している．GPUを用いて高速に行列の積和演算をおこなう研究もなされている[3][4][5][6]．これらの研究の中には，行列の積和演算ではなく行列積の研究をおこなっているものもある．行列の積和演算を高速化するためには行列積の高速化が重要である．CPU上における高速な行列積の実装は，行列の積和演算として提供されている．そこで本稿では，行列積の研究も行列の積和演算の研究と同様に扱う．GPUを用いた行列の積和演算の研究には，良い性能を得られているものも見られる．

本研究では，行列の積和演算をはじめとした数値計算問題に対し，対象問題を分割してCPUとGPUで同時に演算をおこなうことで，より高速に問題を解決することを目指す．本稿では対象問題を行列の積和演算にしぼり，CPUのみで演算をおこなったときおよびGPUのみで演算をおこなったときに比べて，より高速に演算をおこなうことのできる，CPUとGPUの並列演算方式を提案し評価する．

本稿の構成は以下のとおりである．まず2章でGPGPUと数値計算に関する先行研究について論じる．3章でCPUとGPUによる並列処理方式につい

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

て検討し，4章ではCPUとGPUの並列処理方式による行列の積和演算の実装と評価をおこない，5章でまとめる．

2. 関連研究

GPUのハードウェア的な特徴として，CPUと比べて並列計算やベクトル計算に適しているという点が挙げられる．これはGPU本来の処理である画像処理，特にポリゴン表示のための演算の特徴に起因する．ポリゴン表示に必要な座標計算や色計算には，4次元ベクトルを用いた演算や4×4行列を用いた演算が多く用いられる．また，これらの演算は互いに独立しておこなえることが多いため，高速化のためにGPU内部の処理ユニットが多重化されることが多い．そのため，GPUはCPUと比較して並列計算やベクトル計算に適したハードウェアとなっている．並列計算やベクトル計算を利用すると様々な数値計算が高速化できるため，GPUを数値計算に使う研究がおこなわれるようになった．例えば，松井らによるLU分解の実装[7]などが挙げられる．また，GPU本来の役割である画像処理と関連付けて，数値計算と可視化をGPUによって同時に高速に実行するような試みもなされている．このような例としては，小松原らによる流体シミュレーションの研究[8]などが挙げられる．

行列の積和演算は数値計算の分野において最も良く利用される計算の一つである．行列の積和演算は問題サイズが大きくなると急激に計算量が増え実行時間が延びてしまうため，高速化のための研究が盛んにおこなわれてきた．現在では，行列の積和演算を含む行列やベクトルに関する様々な演算をまとめたライブラリBLAS[9]が広く利用されている．BLASは関数インターフェースの集合であるため，実行環境にあわせたBLASの高速実装を入手することで，BLASのインターフェースを利用したプログラムを高速化させることができる．そのため，BLASは多くの科学技術計算ライブラリの内部で利用されている．BLASの高速な実装の代表にATLAS[2]が挙げられる．ATLASは行列の積和演算において，最大でCPUの理論性能の90%という高い性能を引き出すことに成功している．

行列積をGPU上で高速に解くための研究がいくつか報告されている．GPUにおいては，行列積と行列積和ではCPU-GPU間のデータ転送量に大きな差が生じるが，ある程度問題サイズが大きければデータ転送時間は演算時間に隠蔽されるはずである．また計算量を考えた場合も，ある程度問題サイズが大きければ，行列和の追加による計算量の増加量は行列積の計算量に比べて極めて小さくなる．よって，行列積のアルゴリズムや実装が行列の積和演算においても重要であると考え，ここではGPUによる行列積の先行研究に注目する．

Larsenら[3]は，描画処理を利用した行列積の解法について提案した．ここでは行列積$C = A \times B$において，行列Cの(i,k)要素が行列Aの(i,*)要素と行列Bの(*,k)要素を参照することを利用している．行列Aの各列と行列Bの各行を順番に乗算し，足しあわせて行列積とする手法である．これは，GPUの基本的な処理であるテクスチャの描画に良く対応した手法である．その後，GPUの内部処理をソフトウェアレベルである程度自由に変更することのできるプログラマブルシェーダが普及してからは，プログラマブルシェーダを利用した計算手法の提案が多くおこなわれるようになった．Thompsonら[10]やHallら[5]はGPUのアーキテクチャを意識したより効率的な行列積の実装方法について議論や実装をおこなった．ここではベクトル化による計算の効率化と，テクスチャキャッシュを意識した高速化に関する検討が中心となった．これらを踏まえて，Fatahalianら[6]は実機による性能評価をおこなった．その結果，環境次第ではPentium4のATLAS並に高速に行列積をおこなうことができた．しかしながら，GPUの理論性能はCPUを遥かに上回ると言われている[11]のに対して，低い実効性能しか得られていなかった．その理由として，GPU内部のキャッシュバンド幅が足りないことを指摘している．

以上から，GPUを用いた行列の積和演算については，これまでの研究においてある程度良い性能が得られている一方，理論性能に対する実効性能という点では十分な性能が得られていないのが現状であると言える．

3. CPUとGPUによる並列処理方式

3.1 CPUとGPUによる並列処理方式の提案

GPGPUにおけるCPUの主な役割は，CPU-GPU間のデータ転送とGPUへの命令発行である．CPUは，GPUへデータを転送し，描画の開始を指示した後は，GPUからデータの書き戻しが可能になるまで何もすることがなくなってしまう．近年のGPUはデータ転送にハードウェア機能を利用できるようになっていることもあり，プログラマブルシェーダによる演算時間がGPGPUの実行時間の多くを占める場合，CPUがGPUの演算終了を待つ時間の割合は大きい．この待ち時間にCPUも演算をおこなえば，CPUの演算

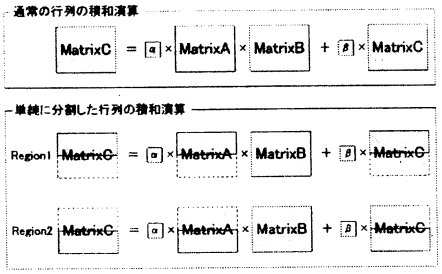

図 1 行列積和の分割実行

Fig. 1 Divided execution of matrix multiply and add.

能力と GPU の演算能力を同時に得ることが可能になり，対象問題をより高速に解けるようになることが期待できる．

例えば，既に述べたように，GPU を用いて CPU の高速実装である ATLAS 並に高速に行列の積和演算が実行できた報告が存在する[6]．このとき，CPU がほとんど処理をおこなっていないならば，積和演算を部分問題に分割し，CPU と GPU で同時に別の部分問題を並列実行することで，更に高速化がおこなえることが期待できる．

ここで述べている問題解決方法は，異なる種類のプロセッサを複数用いて処理をおこなっていることから，分散処理と呼ぶのが正しいかも知れない．しかし，ここでは CPU と GPU が同じ計算を同時におこなっていることを強調するため，**CPU と GPU による並列処理**と呼ぶことにする．これまでの数値計算における GPGPU では，いかにして GPU に高速に数値計算をおこなわせるかが注目されてきた．しかし，GPGPU を行う環境には CPU も備えられているのが普通である．複数の高機能なプロセッサが備えられているのであれば，それら全てを有効に利用すべきであるが，これまでそのような提案はおこなわれていない．なお，グラフィックス分野，特にリアルタイムムービーなどにおける GPGPU の場合は，CPU が物理演算をおこない GPU が照明演算をおこなうような CPU と GPU による問題分割はおこなわれている．

次章では CPU と GPU の並列処理による行列の積和演算を実装し，性能を評価する．

3.2 行列の積和演算に対する方式の適用

行列の積和演算は，行列 A，行列 B，行列 C，スカラー α，スカラー β に対して，$C = \alpha \times A \times B + \beta \times C$ をおこなうものである．そこで，**図 1** のように行列を分割する．この分割方法では，データの更新が行われるのは行列 C のみであり，また更新する要素以外に行列 C の参照はおこなわれないため，それぞれの積和演算を同期をとらずに同時に実行することができる．そのため，CPU 用と GPU 用にそれぞれ一つずつ，合計 2 つのスレッドを用意する．CPU 用のスレッドでは ATLAS(version 3.6.0) を利用し，行列 B 全体と，行列 A および行列 C の上側を用いて行列の積和演算をおこなう．GPU 用のスレッドでは行列 B 全体と，行列 A および行列 C の下側を担当する．担当領域のデータを CPU から GPU へ転送し，プログラマブルシェーダを用いて演算をおこない，GPU から CPU へのデータの書き戻しをおこなう．また CPU と GPU の担当する領域の境界を変更し，それぞれの行う計算量と性能の関係を調査する．なお，CPU や GPU が複数搭載されたシステムであれば，更に多くのスレッドで分割実行をすることでより良い性能が得られる可能性も考えられるが，これは今後の課題とする．

4. 性能評価

4.1 実験環境

ここでは CPU と GPU による並列処理の有効性を確認するための実験をおこなう．実験対象の問題は，32bit 単精度浮動小数点データからなる行列の積和演算 (BLAS における SGEMM) とする．なお，この演算精度は現在の GPU がサポートする最大の演算精度である．

本稿では GPU プログラムの作成および実行に，グラフィックス API として DirectX を，シェーダ言語として HLSL を利用した[12]．GPU による行列の積和演算の実装については，Fatahalian ら[6] による行列積の方式を利用した．彼らは大きくわけて Multi と Single の二つの方式を提案している．二つの方式の主な違いは，行列データのテクスチャへの配置方法である．どちらの方式も，テクスチャをもう一枚用意して加算をおこなうように変更すれば，行列の積和演算に利用できる．後に示す二つの実験環境において，それぞれの実装方法を元に行列の積和演算をおこなった結果，Single の方が Multi に比べて最大で 9.24%高速に問題を解くことができた．しかしながら，Single では CPU と GPU で行列データをやりとりするために，GPU 上のメモリ配置にあわせて CPU 上の行列データを再配置する必要がある．一方 Multi ではそのような余計な処理は不要であり，並列処理の実装がおこないやすい．そのため，今回は Multi を元に並列プログラムの実装をおこなった．以降の予備実験や評価実験は，全て Multi を元にした実装に対しておこなう．

なお，DirectX の初期化関連処理や，HLSL プログラムの読み込み処理など，演算とデータ通信以外の各

表 1　予備実験および評価実験を行った実験環境
Table 1　Experimental environment of preliminary experiments and evaluation experiments.

	環境 AGP	環境 PCI-E
CPU	Pentium4 2.4GHz	Pentium4 3.0GHz
MainMemory	512MB	2GB
OS	WindowsXP	WindowsXP
GPU	GeForce 6800GT	GeForce 6600GT
GPU 接続バス	AGP x4	PCI-Express x16
VRAM	512MB	256MB
GPU コアクロック/メモリクロック	350MHz / 1.00GHz	300MHz / 1.00GHz
頂点処理ユニット数 / ピクセル処理ユニット数	6 / 16	3 / 8
チップセット	Intel 845	Intel 915
ビデオドライバ	ForceWare 76.44	ForceWare 76.44

表 2　CPU と GPU それぞれの行列の積和演算における実行時間および FLOPS 値
Table 2　Execution time and FLOPS of matrix multiply and add CPU and GPU respectively.

問題	実行時間 (sec)		GFLOPS		GPU/CPU
サイズ	CPU	GPU	CPU	GPU	性能比 (%)
環境 AGP					
1024	0.34	0.48	5.88	4.17	70.8
1536	1.13	1.47	5.97	4.59	76.9
2048	2.61	3.36	6.13	4.76	77.7
環境 PCI-E					
1024	0.31	0.52	6.45	3.85	59.6
1536	1.03	1.67	6.55	4.04	61.7
2048	2.42	3.95	6.61	4.05	61.3

種処理はスレッド生成前に済ませている．そのため，CPU と GPU の並列処理をライブラリとして実用化する場合には，これらの処理のオーバーヘッドによってある程度性能が低下することは避けられない．これらを考慮した評価は今後の課題とする．

実験は**表 1** に示す二つの環境で行った．二つの環境の主な違いとしてはビデオカードの接続バスが挙げられる．また実験機材の都合上，環境 PCI-E の方が CPU の性能が高いが，GPU の性能は環境 AGP の方が高い実験環境となっている．

4.2 予備実験

性能評価をおこなう前に，予備実験として各実験環境における CPU のみ，および GPU のみの積和演算の性能を測定した．結果を**表 2** に示す．なお表中の問題サイズについては行列の一辺の長さを示しており，問題サイズ 1024 の積和演算とは 1024×1024 の正方行列による積和演算を意味する．今回の実験環境における，行列の積和演算を対象としたときの CPU に対する GPU の性能は，環境 AGP で 75.1%，環境 PCI-E で 60.9%であることがわかった．(いずれも対象とした 3 種類の問題サイズでの平均値．) なお，GPU による行列の積和演算の実行時間には，CPU-GPU 間の通信時間など主要な計算時間以外の時間も含まれる．

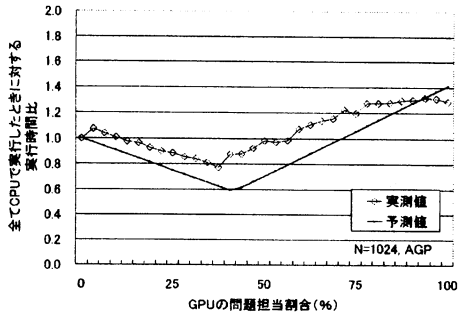

図 2　予測値と実測値の比較 (問題サイズ 1024, 環境 AGP)
Fig. 2　Comparison between predicted value and measurement value (N=1024, AGP).

行列の積和演算は計算に使うデータの量に対して計算量が多いため，この予備実験における主要な計算時間以外の時間は実行時間の 10%以下に抑えられた．よって，こうした時間は並列実行する上での致命的な障害にはならないものと考える．

ここで，CPU と GPU で行列の積和演算を並列実行した際の性能を予測することを考える．上記の予備実験から算出された FLOPS 値と，CPU と GPU それぞれが担当する問題の量を利用すれば，各々の実行時間を予測することができる．全体の実行時間を予測するには，CPU の実行時間と GPU の実行時間のうち，大きな方をとればよい．予測の結果は，次節の評価実験の中で実験結果とともに示す．

4.3 評価実験

CPU と GPU の並列処理の評価実験として，行列の積和演算を並列実行して性能を測定した．また，その結果を前節でおこなった性能予測の結果と比較した．**図 2** から**図 7** にそれぞれ比較結果を示す．各グラフには，前節で求めた実行時間の予測値と，実験によって測定された実行時間の実測値を示す．横軸は GPU の問題担当割合 (右側ほど GPU が多くの領域を担当す

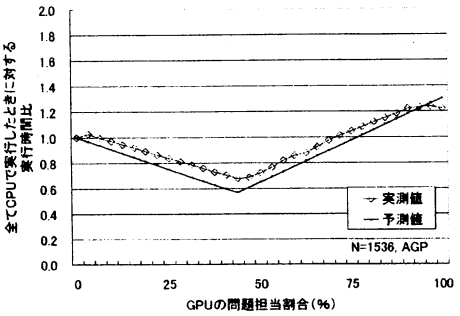

図 3 予測値と実測値の比較 (問題サイズ 1536, 環境 AGP)
Fig. 3 Comparison between predicted value and measurement value (N=1536, AGP).

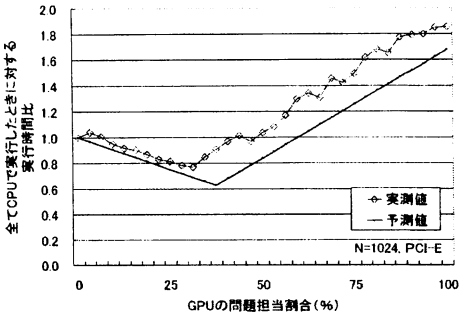

図 5 予測値と実測値の比較 (問題サイズ 1024, 環境 PCI-E)
Fig. 5 Comparison between predicted value and measurement value (N=1024, PCI-E).

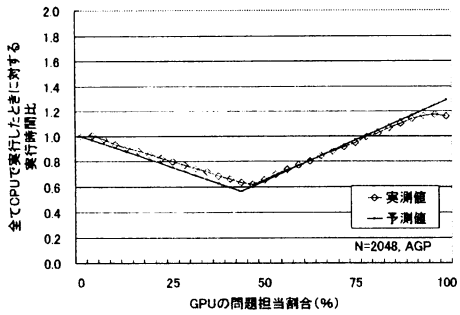

図 4 予測値と実測値の比較 (問題サイズ 2048, 環境 AGP)
Fig. 4 Comparison between predicted value and measurement value (N=2048, AGP).

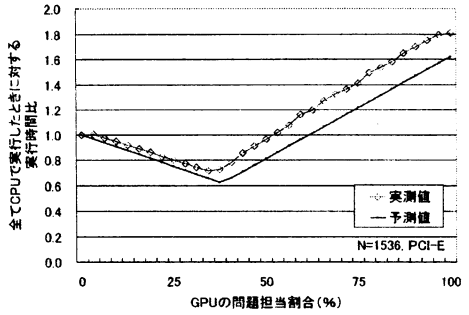

図 6 予測値と実測値の比較 (問題サイズ 1536, 環境 PCI-E)
Fig. 6 Comparison between predicted value and measurement value (N=1536, PCI-E).

る) で，縦軸は問題全体を ATLAS で実行した場合の実行時間を 1 とした実行時間比を示している．なお，CPU か GPU のどちらかが全ての計算を担当する場合は，もう一方のスレッドは何もせずに終了している．そのため CPU が全てを担当した場合の実行時間は，ATLAS を直接利用した場合と比較してもほとんど差は生じなかった．

実験の結果，どのグラフも同様の傾向を示した．左右の端が高く，中央やや左よりが一番低いという傾向である．これは，CPU と GPU で問題を分割し並列実行することで，CPU か GPU のみで実行した場合に比べて実行時間を短縮することができていることを示している．問題サイズが大きい方が並列実行の効果が高くなっていることや，実測値のグラフが予測値のグラフに近いこともわかる．実行環境や問題サイズに関係なくある程度の性能を予測できていることから，あらかじめ CPU と GPU それぞれの FLOPS 値を測定しておくことで，最適な問題分割をおこなうことができる可能性があると見て良いだろう．なお一番性能が良い点に注目すると，環境 AGP では問題サイズ 2048 において，GPU に 46.9%の問題を割り当てたところで CPU のみに対して最大 38.1%の速度向上，同様に環境 PCI-E では問題サイズ 2048 において，GPU に 34.4%の問題を割り当てたところで最大 31.1%の速度向上を達成できた．また FLOPS 値で評価すると，並列処理をおこなうことによって環境 AGP で最大 9.06GFLOPS，環境 PCI-E で最大 9.92GFLOPS という高い性能を得ることができた．

5. ま と め

本稿では，CPU の演算性能と GPU の演算性能を同時に利用して高い演算性能を引き出す，CPU と GPU による並列処理について検討を行った．また，これを用いて行列の積和演算を並列実行し，CPU のみで実行した場合に対して最大で 38.1%の性能向上を得ることができた．

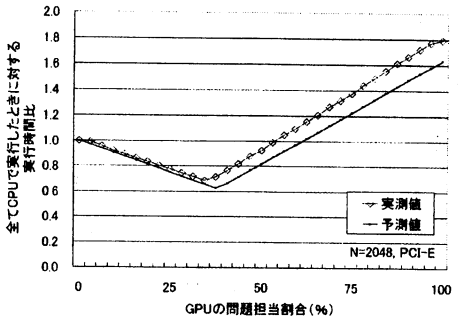

図 7 予測値と実測値の比較 (問題サイズ 2048, 環境 PCI-E)
Fig. 7 Comparison between predicted value and measurement value (N=2048, PCI-E).

今後は，本稿の結果を元に CPU と GPU を用いて高速に行列の積和演算などの問題を解くための方式をまとめ，実装および評価を行う．具体的には，搭載されている CPU と GPU の性能を元に，CPU と GPU への最適な問題の割り当てをおこなう機能を ATLAS に組み込むことを検討している．また CPU と GPU の演算精度について評価をおこない，数値計算分野における GPU の利用に向けて検討と評価を進める．

参 考 文 献

1) GPGPU: General-Purpose Computation Using Graphics Hardware, http://gpgpu.org/.
2) Whaley, R. C., Petitet, A. and Dongarra, J. J.: Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing*, Vol. 27, No. 1–2, pp. 3–35 (2001).
3) Larsen, E. and McAllister, D.: Fast matrix multiplies using graphics hardware, *Proceedings of the 2001 ACM/IEEE conference on Supercomputing* (2001).
4) Ádám Moravánszky: Dense Matrix Algebra on the GPU, ShaderX2 (2003).
5) Jesse D. Hall, Nathan A. Carr, J. C. H.: Cache and Bandwidth Aware Matrix Multiplication on the GPU, Technical report, University of Illinois Dept. of Computer Science (2003).
6) K.Fatahalian, J.Sugerman and P.Hanrahan: Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication, *Graphics Hardware 2004* (2004).
7) 松井学, 伊藤文彦, 萩原兼一: プログラマブル GPU における LU 分解の設計と実装, 先進的計算基盤システムシンポジウム SACSIS2005, pp. 359–367 (2005).
8) 小松原誠, 森眞一郎, 中島康彦, 富田眞治: 汎用グラフィクスカードを用いた格子ボルツマン法による流体シミュレーション, 情報処理学会研究報告, 2005-ARC-163, pp.37-42 (2005).
9) Higham, N. J.: Exploiting Fast Matrix Multiplication Within the Level 3 BLAS, *ACM Transactions on Mathematical Software*, Vol. 16, No. 4, pp. 352–368 (1990).
10) Thompson, C. J., Hahn, S. and Oskin, M.: Using Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis, *Proceedings of the 35th annual ACM/IEEE International Symposium on Microarchitecture*, pp. 306–317 (2002).
11) John Montrym, H. M.: THE GEFORCE 6800, *IEEE MICRO 2005*, Vol.25, No.2 (2005).
12) Microsoft Corporation: DirectX and HLSL, http://www.microsoft.com/japan/windows/directx/default.mspx (DirectX ホームページ).