連立一次方程式に対する共役勾配法の GPU 実装と実験的評価

浅野 翔太郎 †　稲木 雅人 †　永山 忍 †　若林 真一 †
† 広島市立大学大学院　情報科学研究科
〒 731-3194 広島県広島市安佐南区大塚東 3-4-1

概要 コンピュータの性能の進歩に伴い様々な分野で流体解析などの複雑な数値シミュレーションが利用されるようになった．数値シミュレーションの計算時間の大部分は連立一次方程式の求解が占めているため，その解法である共役勾配法の高速化が数値シミュレーションの高速化につながる．一方，高速に問題を解くために，大量のデータを多数のプロセッサで並列に処理できる GPU(Graphics Processing Unit) を汎用的な計算に利用する手法がある．そこで本論文では，共役勾配法の GPU 実装を複数試行し，さらなる高速化のため各処理に必要な計算時間・オーバーヘッドの解析を行う．

GPGPU Implementation of the Conjugate Gradient Method for Simultaneous Linear Equations and Its Experimental Evaluation

Shotaro ASANO†　Masato INAGI†　Shinobu NAGAYAMA†　Shin'ichi WAKABAYASHI†
†Graduate School of Information Sciences，Hiroshima City University
3-4-1 Ozuka-higashi，Asaminami-ku，Hiroshima，731-3194，Japan

Abstract Complex numerical simulations came to be performed in various fields in accordance with the increase of the performance of computers. In the computation time of such numerical simulations, solving simultaneous linear equations is dominant in many cases. Therefore, faster numerical simulations can be realized by improving the execution time of the conjugate gradient method for solving simultaneous linear equations. On the other hand, as a method for fast numerical computing, general-purpose computing on GPU is attracting much attention. In this paper, we test multiple GPU implementations of the conjugate gradient method, and analyze the execution and overhead time of each phase for further improvement.

1　はじめに

コンピュータの性能の進歩に伴い航空機・自動車・船舶・鉄道車両の設計などの様々な分野で流体解析などの複雑な数値シミュレーションが利用されるようになった．数値シミュレーションの計算時間の大部分は連立一次方程式の求解が占めているため，連立一次方程式の解法の高速化が課題となっている．

連立一次方程式の解法は大きく直接法と反復法に分けられる．式変形を繰り返す直接法の例として掃出し法 [4] が挙げられるが，直接法は計算時間が大きい．例えば掃出し法の時間計算量は，係数行列を N 次正方行列とすると，$O(N^3)$ となる．そのため，N が大きい大規模な連立一次方程式には適用できない．そこで近似解法として，解を許容誤差内に徐々に収束させていく反復法が用いられる．共役勾配法 (Conjugate Gradient method，CG 法) は連立一次方程式の反復解法としてよく用いられており，係数行列が正定値対称行列のとき掃出し法と同じ時間計算量となる．係数行列が正定値対称行列でない場合も，係数行列が正定値対称行列となる問題に容易に変換可能である．数値シミュレーションに適用する際は係数行列が疎行列の場合が多く、このときは掃出し法よりも効率的に解ける．そのため，共役勾配法のさらなる効率化・高性能実装のため研究が進められている [1]-[3].

共役勾配法は実行する前に連立一次方程式の係数行列に前処理 (不完全コレスキー分解など) を加えると前処理をしていないときと比べて速く収束する特徴があり，前

処理を伴う共役勾配法の実装に関する研究が行われている [2][3]．しかし，前処理の選びかたは解く問題の種類によって変わってくる [4]．一方，共役勾配法のハードウェア実装の容易さから FPGA に並列実装することで高速化を目指す研究がある [1]．この研究では，回路面積などの観点から係数行列の前処理を実装せず共役勾配法のみを実装している．そこで，本論文では前処理は考えず，全ての実装の基礎となる共役勾配法本体のアルゴリズムの実装について FPGA 以外の並列実装方法を検討する．

高い並列演算能力を持つ GPU[6][7] に共役勾配法を実装する研究がある [2]．GPU(Graphics Processing Unit) は描画処理専用デバイスで，GPU を汎用的な計算に利用することを GPGPU(General Purpose Computing on GPU) と呼ぶ．GPU には多くのプロセッサが搭載されているが，1 つ 1 つのプロセッサは単純な構造を持ち，その機能は，CPU に比べ非常に限定されたものである．しかし，大量のデータを複数のプロセッサで同時に，かつ並列に処理できるため，強力な演算装置として GPU を使用できる．

既存研究 [2] では，連立一次方程式に対する共役勾配法の計算時間の短縮のため，共役勾配法のアルゴリズムの中で行列-ベクトル積やベクトルの内積などの各工程を CPU で行うか，GPU で行うかを変えて複数の検討対象手法としている．しかし，この手法では CPU-GPU 間の転送時間，CPU から GPU を呼び出すときのオーバーヘッド，および GPU から CPU に戻るときの同期が問題となる可能性がある．そのため本研究では，CPU に戻らずに GPU 内部で GPU へのタスクを生成できるダイナミック並列処理を試した．実行時間の計測実験の結果，行列-ベクトル積を GPU で実装した場合，未知変数の個数が多くなるほど CPU のみで実装した場合よりも高速に解を求められることを確認した．一方で，ダイナミック並列処理を含め，その他の工程の GPU 実装は結果に大きな影響を与えなかったため，検証として各工程の処理時間およびオーバーヘッドの解析を行った．また，GPU 実装による行列-ベクトル積の計算時間の改善比を検証するため，GPU 実装の効果が高いことが知られている行列-行列積との比較を行った．結果として実験結果の妥当性が示された．

2 準備

2.1 連立一次方程式

連立一次方程式は，

$$\sum_{j=1}^{N} a_{ij}x_j = b_i, \quad i = 1, 2, ..., N \tag{1}$$

と表記される方程式である．x_j は未知変数，a_{ij} と b_j は既値の定数である ($i = 1, 2, ..., N$, $j = 1, 2, ..., N$)．

また，式 (1) を，N 次正方行列 A と 2 つの N 次元ベクトル $\mathbf{x}$，$\mathbf{b}$ を用いて表現すると

$$A\mathbf{x} = \mathbf{b} \tag{2}$$

となる．本論文では，式 (2) の行列 A とベクトル $\mathbf{b}$ が既値の定数であるとき，この式を満たすベクトル $\mathbf{x}$ の値を高速に求めるために，共役勾配法の効率的な GPU 実装を検討する．

2.2 共役勾配法

共役勾配法は連立一次方程式 $A\mathbf{x} = \mathbf{b}$ を解くための解法の 1 つである [5]．ここで，A は N 次元行列，$\mathbf{b}$ は既知の N 次元ベクトル，$\mathbf{x}$ は未知の N 次元ベクトルである．また共役勾配法は，連立一次方程式の係数行列 A が N 次対称正定値行列 ($A = {}^tA$ であり，任意のベクトル $\mathbf{x} \neq \mathbf{0}$ に対し，$\mathbf{x}^tA\mathbf{x} > 0$ を満たす行列) であるときに丸め誤差がなければ解を N 回の反復で求めることができるという特徴がある [5]．数値シミュレーションで扱うことの多い疎行列ではさらに反復回数が少なく効率的になる．

2.3 共役勾配法のアルゴリズム

本論文での GPU 実装に用いる共役勾配法のアルゴリズム [5] を以下に示す．$\mathbf{tmp}$，$\mathbf{p}$，$\mathbf{r}$ は N 次元ベクトルである．また，$(\mathbf{a}, \mathbf{b})$ はベクトル $\mathbf{a}$ とベクトル $\mathbf{b}$ の内積を，$\| \mathbf{a} \|$(ノルム $\mathbf{a}$) はベクトル $\mathbf{a}$ の全ての要素の合計，ε は許容誤差を表す．

1. 適当な初期ベクトル $\mathbf{x}$ を選んで $\mathbf{p} \leftarrow \mathbf{b} - A\mathbf{x}$，$\mathbf{r} \leftarrow \mathbf{p}$ とする．
2. 次の (1)〜(7) を順に (5) において終了条件を満たすまで繰り返す．
   - **(1)** $\mathbf{tmp} \leftarrow A\mathbf{p}$
   - **(2)** $\alpha \leftarrow \frac{(\mathbf{p}, \mathbf{r})}{(\mathbf{p}, \mathbf{tmp})}$
   - **(3)** $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
   - **(4)** $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{tmp}$
   - **(5)** $\| \mathbf{r} \| < \varepsilon$ ならば終了
   - **(6)** $\beta \leftarrow -\frac{(\mathbf{r}, \mathbf{tmp})}{(\mathbf{p}, \mathbf{tmp})}$
   - **(7)** $\mathbf{p} \leftarrow \mathbf{r} + \beta\mathbf{p}$

2.4 GPU

GPU(Graphics Processing Unit) は，描画処理に特化した専用プロセッサである．GPU を描画処理ではなく，他の用途で利用することを GPGPU(General-Purpose Computing on GPU) という．本論文では，連立一次方程式に

対する共役勾配法に GPU を用いることで計算を並列化する．

本論文で使用する GPU は NVIDA 社の Kepler アーキテクチャを有する Tesla K20c である．Tesla K20c には 13 個の Streaming Multiprocessor eXtreme(SMX) があり，各 SMX に 192 個の Streaming Processor(SP) を搭載している [7]．CPU 等のプロセッサには，命令デコードといわれる機能がある．これにより，プロセッサは次に行う作業を取得することができる．一方で，SP は命令デコードの機能をもっていない．SP を複数搭載している SMX 内に命令デコード機能があり，この SMX でデコードされた命令が SP に伝えられる．その時に，SMX に搭載されている全ての SP に同じ内容の命令が伝えられる．つまり，同じプログラムが複数の SP 上で並列に動いている．これにより，大量のデータを並列に処理することができる．

プログラムの実行はスレッドという単位で管理される [6]．各 SP は各時点で 1 スレッドのみを実行する．スレッドはブロックと呼ばれるスレッドの集合単位で扱われ，1 つのブロックは 1 つの SMX で実行される．SMX が持つ SP 数以上のスレッドからなるブロックは時分割で処理される．GPU が持つ SMX 数以上のブロックがある場合は時分割で処理される．異なるブロック内のスレッドは処理の終了によってのみ同期できる．

次に GPU のメモリアーキテクチャについて説明する．一番大きいメモリとしてグローバルメモリがある．グローバルメモリの実態は GPU とは独立したデバイスメモリであり，容量が大きいが GPU そのものから遠い位置にあるため，アクセス速度が他のメモリと比べて遅いという特徴がある．グローバルメモリは全ての SMX 内の全ての SP からアクセス可能である．SMX 内には，SMX 内の全ての SP がアクセスできるシェアードメモリがある．これは容量が小さいが，グローバルメモリと比べてアクセス速度が高速である．シェアードメモリは設定によりグローバルメモリに対するキャッシュメモリとしても動作する．本論文では考慮しないが，その他にも読み込み専用のメモリ領域であるコンスタントメモリやコンスタントキャッシュ，テクスチャキャッシュ，各 SP に付属するレジスタなどがある．

2.5 ダイナミック並列処理

ダイナミック並列処理 (Dynamic Parallelism) は，Tesla シリーズの内，Kepler 世代で追加された機能である [7]．従来の GPU は CPU の外部演算ユニットの一部として扱われ，必要に応じて随時 CPU が呼び出し処理を行う形態をとっていた．1 処理ごとに CPU との間にデータ通信を行うため，計算効率を悪化する要因となっていた．そこで，図 2 のように CPU を介することなく GPU 内部で処理を自動完結させることができるダイナミック並列処理機能が付加された．ダイナミック並列処理は，GPU がカーネル (GPU で実行するプログラム) の実行時に，その実行中のカーネルの内部から GPU 自身の新たなタスクを生成するという方法で，CPU から独立性が高い形で GPU を自律的に動作させることが可能である．

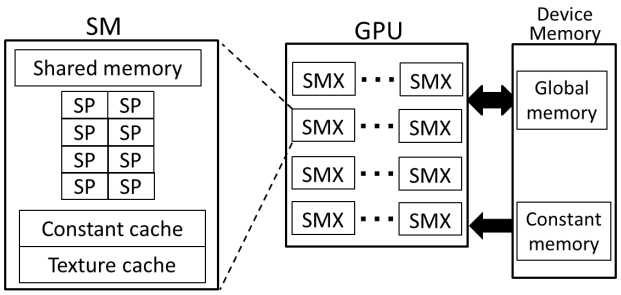

図 1: GPU のアーキテクチャ

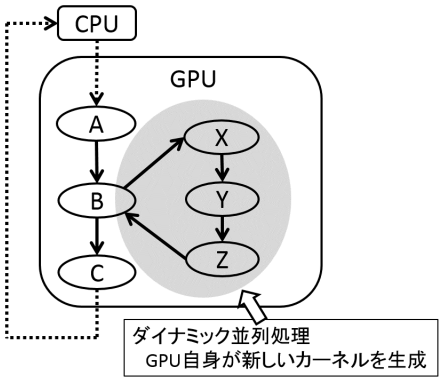

図 2: ダイナミック並列処理

3 連立一次方程式に対する共役勾配法の GPU を用いた実装

本論文では，GPU で行う処理と CPU で行う処理の割り当てを変えた 3 種類の GPU 実装について検討する．本節では，まず共役勾配法の主要な処理の GPU 実装方法を説明し，次に各 GPU 実装においてどの処理に GPU を用い，どの処理に CPU を用いるか，データ転送及び同期の仕方などについて説明する．

3.1 行列-ベクトル積

行列-ベクトル積演算 $\mathbf{c} \leftarrow A\mathbf{b}$ において，ベクトル $\mathbf{c}$ の各要素を求める演算はそれぞれ独立しているので並列に実行可能である．ここで，A は N 次元行列，$\mathbf{b}$ と $\mathbf{c}$ は N 次元ベクトルである．そのため，1 つのスレッドにベクトル $\mathbf{c}$ の各要素を求める演算を割り当てていく．また，行列 A は行ごとにベクトル $\mathbf{b}$ を何度も呼び出しているので，

GPU へ実装するときベクトル **b** をメモリアクセスの時間が長いグローバルメモリからメモリアクセスの時間が短いシェアードメモリへデータをコピーして，時間を短縮する．

3.2 ベクトル同士の内積

ベクトル同士の内積 (**a**, **b**) を求める演算は，まずベクトル **a** と **b** の各要素ごとの乗算は独立してできているので，乗算 1 つにスレッドを 1 個割り当てて並列に実行する．同期のためカーネルを起動しなおしたのち，全要素の合計を図 3 のように 1 段ずつ，隣り合う要素の加算 1 つにスレッドを 1 個割り当てて並列に行う．ただし，加算でも 1 段ごとに全ての要素が計算し終わるまで待つために同期をとる必要がある．ここで，**a** と **b** は N 次元ベクトルである．複数のブロックを用いて全要素の合計を計算する場合，各ブロックでは割り当てられた要素に対して図 3 の処理を行う，これによりブロックごとの要素の総和が得られるので，カーネルの再起動によりブロック間の同期を行ったのち，得られたブロックごとの要素の総和の総和を再度計算する．これを全体の総和が求められるまで繰り返す．

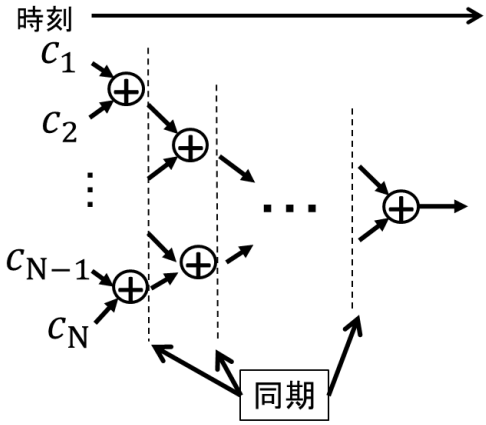

図 3: ベクトル同士の内積

3.3 CPU-GPU 間の演算の割り当て

2.3 節で述べたアルゴリズムの反復処理内の手順の CPU と GPU への割り当て案を図 4 に 3 種類示す．

3.3.1 実装 1

図 4 の実装 1 は，GPU による高速化の効果が大きいと思われる行列-ベクトル積演算のみを GPU で実行して，時間のかからない他の演算は CPU で実行する．ただし，反復手順を行う前に，N 次元行列 A と N 次元ベクトル **x** を GPU 側へ転送する．具体的に示すと，2.3 節のアルゴリズムにおいて，手順 (1) の行列-ベクトル積のみ 3.1 節のように GPU で行う．この構成では，共役勾配法の反復処理 1 回につき，N 次ベクトルのデータ転送が 1 往復分必要である．

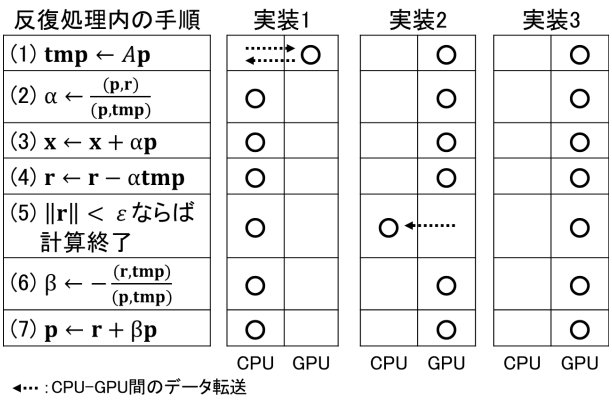

図 4: CPU-GPU の構成

3.3.2 実装 2

図 4 の実装 2 は，CPU と GPU 間のデータ転送を最小限にするために収束判定のみ CPU で行い，他のほぼすべての演算は GPU で実行する．ただし，反復処理を行う前に，N 次元行列 A と N 次元ベクトル **x**，**p**，**r** を GPU 側へデータ転送する．具体的に示すと，2.3 節のアルゴリズムにおいて，手順 (5) はノルムの計算を GPU 側で行い，その結果から終了判定を CPU 側で行う．手順 (1) は行列-ベクトル積を 3.1 節のように GPU で実行する．手順 (2), (6) はまずベクトル同士の内積を 3.2 節のように並列に実行した後，一度同期をとるためにカーネルを起動しなおしてから α と β の計算を行う．手順 (3)，(4)，(7) はベクトル同士の足し算なので，各要素ごとに並列に計算する．この構成では，共役勾配法の反復処理 1 回につき，GPU から CPU へのデータ転送が 1 回しか必要ないが，GPU が苦手とする逐次的な演算に長時間を要すと考えられる．

3.3.3 実装 3

図 4 の実装 3 は，ダイナミック並列処理を用いて全ての演算を GPU で実装する．初めに，CPU から共役勾配法のカーネルを 1 スレッドだけ生成させ，そのスレッドから反復処理を子カーネルに並列に処理させる．ただし，共役勾配法のカーネルを起動する前に，N 次元行列 A と N 次元ベクトル **x**，**b** を GPU 側へ転送する必要がある．並列化は実装 2 と同じように行う．ただし，2.3 節のアルゴリズムにおいて，手順 (5) の収束判定を CPU 側で実行しないかわりに収束判定の前で，一度同期をとる必要がある．この構成は，全ての演算を GPU で処理するため CPU-GPU 間のデータ転送時間を考える必要がない．

4 実験と評価

第 3 章で説明した 3 種類の実装方法全て，および比較対象である CPU のみでの実装について実験結果を示し，これらに対して評価・考察を行う．

4.1 実験環境

比較対象である CPU での実装は，Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz，メインメモリが 24GB の PC 上で実行した．GPU での実装は GPU は Tesla K20c，専用メモリ合計 4GB，CUDA5.5 を使用し，上記の PC 上で実行した．Tesla K20c は SMX を 13 個搭載しており，各 SMX は 192 個の SP を搭載している．つまり，Tesla K20c 全体では 2496 個の SP を搭載している．SP それぞれの動作周波数は 706MHz である．

4.2 CPU 実装および 3 つの GPU 実装の比較

4.2.1 実験内容

CPU のみによる実装，GPU 実装 1，2，3 の比較実験を行った．実験方法は，1 ブロックあたりのスレッド数は 64，ブロック数は 32 もしくは未知変数の個数/1 ブロックあたりのスレッド数のどちらか小さい方を用いて，未知変数の個数が 100，200，...，1000 のときの実行時間を 100 回測定しその平均を求めた．

4.2.2 実験結果

CPU-GPU 間の演算の割り当てを変更したときの実行時間を図 5 に示す．グラフの CPU は CPU のみで共役勾配法を実装したときの実行時間，実装 1 は行列-ベクトル積演算のみ GPU で他を CPU で実装したときの実行時間，実装 2 は収束判定以外全て GPU で実装したときの実行時間，実装 3 はすべての演算を GPU で実装した時の実行時間を表す．実行時間の単位は [ms] である．

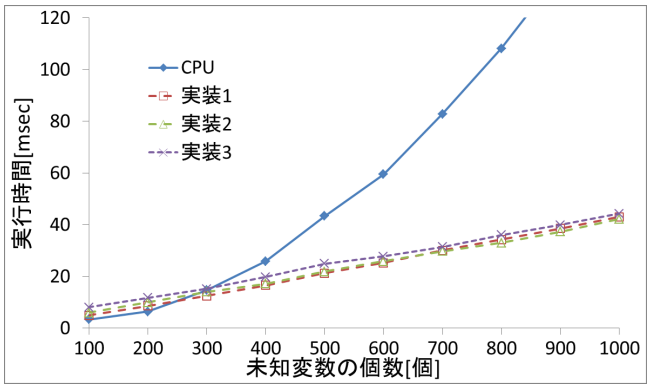

図 5: CPU-GPU 間の演算の割り当てを変更したときの実行時間

図 5 を見ると，CPU のみで実装したときと GPU も用いて実装したときを比べると GPU も用いた方が実行時間が短く，未知変数の個数が多くなるほど差が広がることがわかる．これは，未知変数の個数が多くなるほど，演算の並列化の効果が表れていることを示している．また，未知変数の個数がある程度大きくないと，演算を並列化したことによる短縮した演算時間よりも CPU-GPU 間の通信時間の方が長くなってしまうことが分かった．

実装 1，2，3 を比較すると，実装 3 のダイナミック並列処理を用いた場合も含め，実行時間に大きな差がないことがわかる．これは，共役勾配法のアルゴリズムにおいて実行時間のほとんどが行列-ベクトル積演算の処理時間で占められていることを示唆している．

4.3 共役勾配法の各手順の実行時間

4.3.1 実験内容

4.2 節での推測を検証するため，共役勾配法のアルゴリズムにおいて各演算が全体の実行時間のどれだけを占めているかを測定する実験を行った．実験では，1 ブロックあたりのスレッド数は 64，ブロック数は 32 もしくは未知変数の個数/1 ブロックあたりのスレッド数のどちらか小さい方を用いて，未知変数の個数が 1000 のときの実行時間を 100 回測定しその平均を求めた．

4.3.2 実験結果

共役勾配法のアルゴリズムにおいて各演算が全体の実行時間のどれだけを占めているか調べた．表 1 は未知変数の個数が 1000 のときに共役勾配法のアルゴリズムを CPU のみ用いて実行したときと GPU を用いて実行したとき (実装 3) の各演算にかかる実行時間である．合計は，各演算 (行列-ベクトル積，内積，ベクトルのスカラー倍，それ以外) の実行時間の合計である．実行時間の単位は [ms] である．表 1 を見ると，CPU 実装では行列ベクトル積が全体の実行時間の 98%以上を占めており，GPU 実装でも 90%近くを占めていることが分かる．この結果から，4.2 節での推測の正しさが示された．また，行列-ベクトル積の実行時間をさらに短縮できれば共役勾配法のアルゴリズム全体の高速化につながることが確かめられた．

表 1: 共役勾配法の各手順の実行時間 (単位:ms)

演算	CPU	GPU
行列-ベクトル積	189.060	38.728
内積	2.136	1.783
ベクトルのスカラー倍	0.745	0.265
それ以外	0.000	3.280
合計	191.941	44.056

4.4 行列-ベクトル積の高速化

4.4.1 実験内容

行列-ベクトル積演算の高速化について検討した．行列-行列積は並列度を容易に上げられるため GPU で効率的に実行可能であることが知られている．そこでまず行列-行列積の実行時間を計測する．次に行列-ベクトル積の GPU 実装の並列度を高め，行列-行列積同様に効率的に実行可能であるか検討した．

行列-ベクトル積演算を GPU に実装するとき行列の各行ごとにスレッドを割り当てて並列化しており，スレッドあたり $O(N)$ の計算量で実行している．これを，行列の各要素ごとにスレッドを割り当てて並列化し，スレッドあたり $O(\log N)$ の計算量 (要素同士の積に $O(1)$，各行の総和計算に $O(\log N)$．3.2 節を参照．) で実行し，比較した．1 ブロックあたりのスレッド数およびブロック数は前節の実験と同様とした．

4.4.2 実験結果

行列-行列積演算を GPU で (スレッドあたり $O(N)$ の計算量で) 実行すると，表 2 のように CPU と比べて高速に実行できた．表 2 の GPU4 は，CPU-GPU 間の行列の転送時間を除いた結果，GPU4' は転送時間も含めた結果，比率は CPU と比べ GPU4' が何倍速くなったかを示す．次に行列-ベクトル積の計算の並列度を行列-行列積

表 2: 行列-行列積の実行時間 (単位:ms)

N	CPU	GPU4	GPU4'	比率
16	0.167	0.224	0.757	0.220
32	0.397	0.238	0.776	0.512
64	2.676	0.275	0.818	3.271
128	16.253	0.476	1.062	15.304
256	140.813	2.451	3.098	45.453
512	1361.295	16.395	18.560	73.346
1024	11382.879	128.745	132.358	86.000
2048	199655.375	1025.707	1037.466	192.445

のときと同様に上げる実験を行った．結果を表 3 に示す．表 3 の GPU5 と GPU6 は，それぞれスレッドあたり計算量 $O(N)$(スレッド数 N) の実装，スレッドあたり計算量 $O(\log N)$(スレッド数 N^2) の実装において CPU-GPU 間の転送時間を除いた結果，GPU5' と GPU6' は，転送時間も含めた結果である．実行時間の単位は [ms] である．表 3 を見ると，スレッドあたり $O(N)$ の計算量で実行した時よりも $O(\log N)$ の計算量で実行した時のほうが遅くなっている．これは，行列の各要素ごとの乗算と行列の行ごとの総和計算のために生成する大量のスレッドの管理に

表 3: 行列-ベクトル積の実行時間 (単位:ms)

N	CPU	GPU5	GPU6	GPU5'	GPU6'
16	0.101	0.214	0.471	1.127	1.319
32	0.109	0.214	0.477	1.131	1.332
64	0.131	0.215	0.898	1.165	1.757
128	0.243	0.240	0.953	1.486	1.846
256	0.545	0.357	1.178	1.455	2.204
512	1.681	0.621	2.023	2.660	3.520
1024	5.505	1.370	3.810	5.205	7.092
2048	18.565	2.303	19.375	14.099	30.531

伴うオーバーヘッドのためと思われる．また，GPU5 の $N = 128$〜2048 の結果を見ると，N に対してほぼ線形となっており，データ転送時間を除くと計算時間が妥当であることが確認できた．

5 まとめと今後の課題

本論文では，連立一次方程式に対する共役勾配法の GPU 実装について検討し，実験的評価を行った．検討した手法では，並列に演算を処理可能な行列-ベクトル積やベクトル同士の内積などを GPU 上で行い，実行時間を短縮した．実験の結果，未知変数の個数が大きくなるほど GPU での実装が有効であることを示した．

今後の課題としては，メモリアクセスやデータ転送など GPU の最適な利用法を考えての実装などが挙げられる．

参考文献

[1] 芳賀祐介，永山忍，稲木雅人，若林真一，“連立一次方程式に対する共役勾配法の FPGA 実装と実験的評価”，第 27 回回路とシステムワークショップ論文集 (2014) (to appear).

[2] 勝田肇，今野佳祐，陳強，澤谷邦男，“GPU による共役勾配法の高速化に関する一検討”，信学技報，vol. 112，no. 216，AP2012-86，pp. 25-29 (2012).

[3] 石黒美佐子，陽遊美由紀，平津忍，“ 前処理付き CG 法系解法における行列の条件数と収束判定 ”，情処研報 HPC，97(121)，pp. 13-18 (1997).

[4] 渡部善隆，“ 連立 1 次方程式の基礎知識〜および Gauss の消去法の安定性について〜 ”，九州大学大型計算機センター広報, vol. 28, no. 4, pp. 291-349 (1995).

[5] 皆本晃弥，“ UNIX & Information Science-5 C 言語による数値計算入門-解法・アルゴリズム・プログラム- ”，株式会社サイエンス社，pp. 109-115 (2005).

[6] 小山田耕二，岡田賢治，“ CUDA 高速 GPU プログラミング入門 ”，秀和システム (2010).

[7] NVIDIA Corporation，“Kepler コンピュート・アーキテクチャ・ホワイトペーパー V1.0”，http://www.nvidia.co.jp/content/apac/pdf/tesla/nvidia-kepler-gk110-architecture-whitepaper-jp.pdf，July (2014).