卒業論文

題目

MESI-CUDAにおける
メモリアクセス効率化のための
配列インデックス解析機能

指導教員

大野和彦　講師

2013年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

丸山剛寛 (409850)

内容梗概

近年，気象予測や分子動力学といった様々な分野において大規模計算の需要が高まっており，より高性能なコンピュータが求められている．そのため，CPU と比べ性能向上のめざましい GPU に汎用的な計算を行わせる GPGPU への関心が高まっている．しかし，GPU プログラムは CPU と GPU の異なるハードウェアを使用するためコーディングは困難である．

そこで，本研究室ではデータ転送を自動化するフレームワーク MESI-CUDA を開発している．しかし，現状の MESI-CUDA はメモリ階層が最適化された CUDA コードを生成できず，最適化された高速な CUDA コードが生成できているとはいえない．

そこで，本研究ではシェアードメモリを使用する最適化された CUDA コードを自動生成するために，MESI-CUDA に配列のインデックス解析機能を設計及び実装した．その結果，本機能を用いることで適切な配列のアクセス解析が行われ，シェアードメモリを使用する最適化された CUDA コードの自動生成を達成した．

Abstract

The performance of Graphics Processing Units (GPU) is improving rapidly. Thus, General Purpose computation on Graphics Processing Units (GPGPU) is expected as an important method for high-performance computing. Although programming frameworks, such as CUDA and OpenCL, are provided, they require explict specification of memory allocations and data transfers.

Therefore, we are developing a new programming framework *MESI-CUDA*, which hides such low-level description from the user. The current implementation of MESI-CUDA generate CUDA code which memory hierarchy is not optimized, so sometimes the code's execute time is long.

In the present study, I designed and implemented array index analysis for memory usage optimization in MESI-CUDA. Consequently, MESI-CUDA generate optimized CUDA code which use shared memory . And the code's execute time is shorter.

目 次

1 はじめに **1**
1.1 研究目的 1
1.2 本文構成 2

2 背景 **3**
2.1 CUDA 3
2.1.1 CUDA の説明 3
2.1.2 動作概要 3
2.1.3 スレッド管理 4
2.1.4 メモリモデル 5
2.2 MESI-CUDA 8
2.2.1 MESI-CUDA の説明 8
2.2.2 従来の MESI-CUDA の問題点 9
2.2.3 最適化 9

3 提案手法 **10**
3.1 概要 10
3.2 解析方法 11
3.3 実装方法 14

4 性能評価 **16**
4.1 評価内容 16
4.2 考察 17

5 おわりに **19**

謝辞 **20**

参考文献 **21**

A プログラムリスト **22**

B 評価用データ **22**

図 目 次

2.1 CUDA プログラムの流れ 4
2.2 CUDA のスレッド管理 5
2.3 CUDA のメモリモデル 7
2.4 MESI-CUDA のメモリモデル 8
3.5 アクセス範囲が重なる例 13
3.6 fig .3.5 のアクセスの様子 13
3.7 アクセス範囲が重ならない例 13
3.8 fig .3.7 のアクセスの様子 13

表 目 次

3.1 for 文用のエントリ 15
3.2 配列用のエントリ 15
4.3 実行時間 (秒) 17
4.4 解析時間 (ミリ秒) 17

1 はじめに

1.1 研究目的

近年，気象予測や分子動力学といった様々な分野において大規模計算の需要が高まっており，より高性能なコンピュータが求められている．そのため，CPU と比べ性能向上のめざましい GPU に汎用的な計算を行わせる GPGPU への関心が高まっている．しかし，GPU プログラムは CPU と GPU の異なるハードウェアを使用するためコーディングは困難である．そこで，本研究室ではデータ転送を自動化するフレームワーク MESI-CUDA[1] を開発している．しかし，現状の MESI-CUDA はメモリ階層が最適化された CUDA コードを生成できず，最適化された高速な CUDA コードが生成できているとはいえない．そこで，本研究ではシェアードメモリを使用する最適化された CUDA コードを自動生成するために，MESI-CUDA に配列のインデックス解析機能を設計及び実装した．

1.2 本文構成

本文の構成は以下のようになっている．第 2 章でまず背景として CUDA と MESI-CUDA の概要，第 3 章に今回提案する解析方法について述べる．第 4 章で，本機能を実装した MESI-CUDA と従来の MESI-CUDA との性能比較の評価結果を示す．最後に第 5 章にてまとめを行う．

2　背景

2.1　CUDA

2.1.1　CUDA の説明

CUDA[2][3][4][5] は NVIDIA 社より提供されている GPGPU 用の SDK であり，コンパイラやライブラリ，デバッガなどから構成されている．ユーザーは C 言語を拡張した文法とライブラリ関数を用いて GPGPU プログラミングを容易に開発することができる．CUDA では，CPU をホスト，GPU をデバイス，GPU 上で実行するプログラムをカーネルと呼ぶ．

2.1.2　動作概要

fig .2.1 に CUDA プログラムの流れを示す．CUDA プログラムではホスト側でプログラムを起動させると，カーネルがデバイスにロードされる．そして，ホスト側でデータを作成し値をデバイスに転送する．それから，ホストがデバイスに対し，カーネルを起動させ処理を開始する．処理の終了後，計算結果をホスト側に転送させる．最後に，ホストが受け取ったデータを処理する．

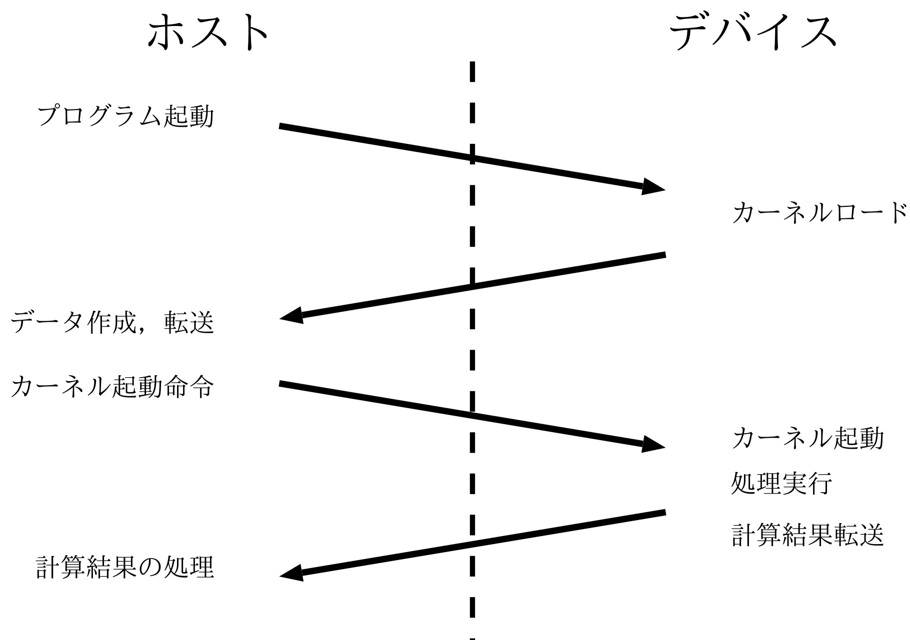

fig . 2.1: CUDA プログラムの流れ

2.1.3　スレッド管理

CUDA では，大規模な並列処理を実現するために，グリッド，ブロック，スレッドという単位で階層的なスレッド管理を行っている．スレッドはカーネルを動作させたときの多数のプログラムの最小単位である．GPU ではコアに対し数千〜数万と圧倒的な数のスレッドが並列に動作することにより，その高い性能を引き出している．fig .2.2 にグリッド，ブロック，スレッドの概念図を示す．ブロックはスレッドをまとめたもので，グリッドはブロックをさらにまとめたものである．ブロック，スレッドはそれぞれ固有の ID が割り振られている．

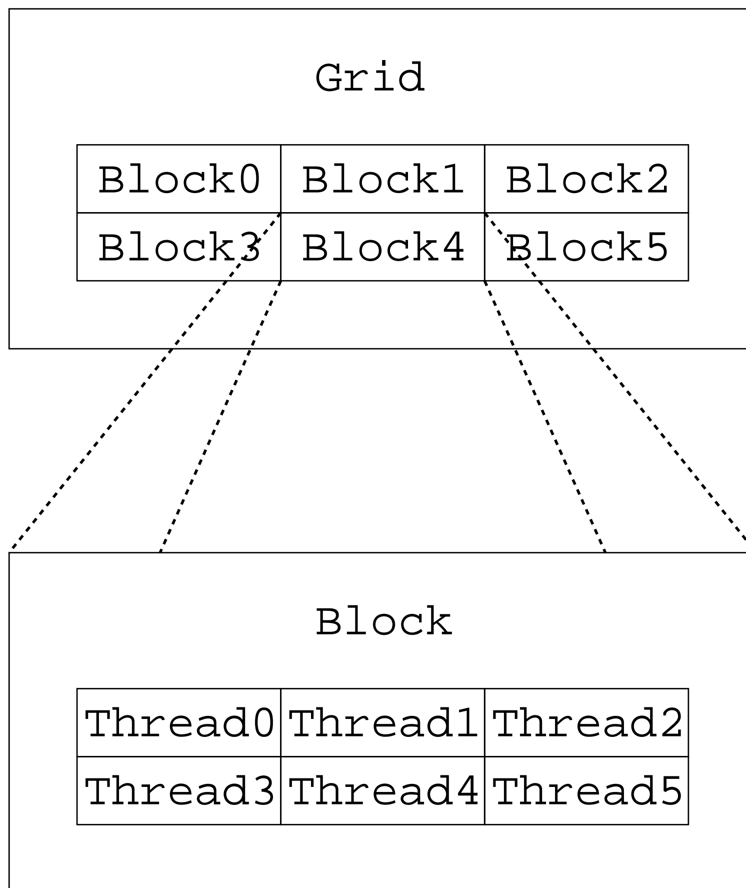

fig . 2.2: CUDA のスレッド管理

2.1.4 メモリモデル

fig .2.3 に CUDA のメモリモデルを示す．CUDA の基本的なメモリモデルは，多数のスレッドがグローバルメモリを共有している構造である．しかし，メモリは複雑に階層化されており，それぞれ性能が大きく異なる．スレッド毎にレジスタやローカルメモリを有し，ブロック毎にシェアードメモリを有する．レジスタやシェアードメモリは容量は非常に小

さいが，高速にアクセスすることができる．読み込み専用だが高速なメモリであるコンスタントメモリやテクスチャメモリもあり処理に合わせて用いる．従って，GPU の性能を最大限に引き出すためにはこれらのメモリ階層を使い分ける必要がある．特にシェアードメモリは，CUDA プログラムの高速化手法としてよく用いられる．これは，通常使用するグローバルメモリと比べ約百倍高速にアクセスできるため，メモリアクセスのレイテンシが小さいためである．

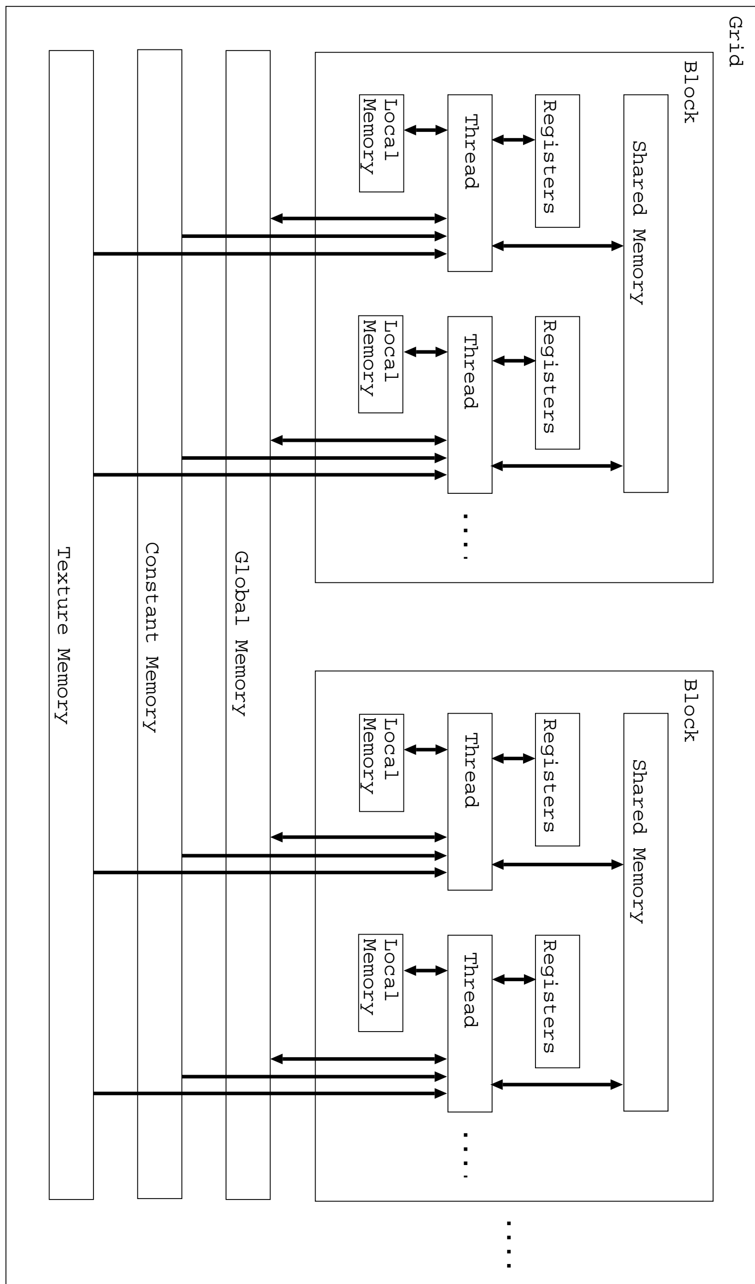

fig . 2.3: CUDA のメモリモデル

2.2 MESI-CUDA

2.2.1 MESI-CUDA の説明

MESI-CUDA はホスト・デバイス間のデータ転送コード，メモリ確保・解放コード，ストリーム生成・破棄のコードを自動的に生成することで，ユーザーの負担を軽減させるフレームワークである．fig .2.4 に MESI-CUDA のメモリモデルを示す．ホストとデバイスへの処理の振り分けやカーネルの記述はユーザ自身が従来の CUDA に準じる形でコーディングを行う．MESI-CUDA では，データ転送やカーネル処理のスケジューリングを自動的に行う．そのために，fig .2.4 のような仮想的な共有メモリ環境のモデルを採用し，ホスト・デバイス両方よりアクセス可能な共有変数を提供する．

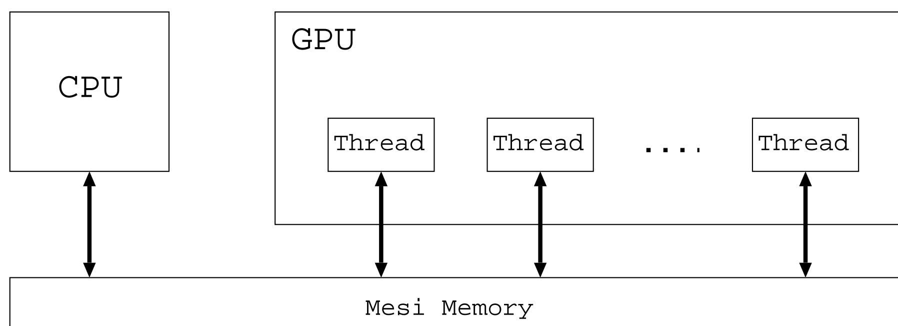

fig . 2.4: MESI-CUDA のメモリモデル

2.2.2 従来の MESI-CUDA の問題点

現状の MESI-CUDA は，グローバルメモリのみを使用するコードを生成しており，メモリ階層が最適化された CUDA コードを生成できない．そのため，手動で最適化したコードと比較すると実行速度が遅い CUDA プログラムを生成してしまうという問題がある．

2.2.3 最適化

本研究室では，aaaa 節の問題を解決するためにシェアードメモリを使用する最適化された CUDA コードの自動生成方法を提案している．この自動最適化方法は，MESI-CUDA コード中のシェアードメモリに格納すべき配列について，シェアードメモリの領域確保，シェアードメモリとグローバルメモリ間のデータ転送，シェアードメモリへのアクセスのコードを自動的に生成するものである．

3 提案手法

3.1 概要

aaaa 節の自動最適化を行うために，シェアードメモリに格納する配列を適切に選択する必要がある．これは，シェアードメモリの容量が 64KB と非常に小さく，プログラム中で使用する全ての配列を格納することは困難なためである．従って，ブロック内でアクセスする範囲を検出する必要がある．これは，従来の範囲解析方法で検出できる．また，複数の配列がブロック内でアクセスする範囲が十分小さくシェアードメモリを利用できる場合，シェアードメモリを効率的に用いるためにはブロック内の使用頻度の高い方のデータを優先的にシェアードメモリに格納する必要がある．これは，従来の範囲解析方法では検出できない．そこで，配列のアクセスするインデックスの解析を行い，ブロック内の使用頻度が高い配列の区間を検出する．今回実装する解析の対象とした MESI-CUDA プログラムは, 1 次元のグリッド，ブロックで一重ループ中の 1 次元配列を扱うプログラムである．

3.2 解析方法

対象とするプログラムの簡単な例を fig .3.5 に示す．これは配列 A のスレッド ID 番目の要素に配列 B の 0 番目の要素から 4*k 番目の要素までを足すプログラムである．このプログラムを k 個のスレッドで実行した時の配列 B のアクセスの様子を fig .3.6 に示す．k 個のスレッドでアクセス範囲が重複していることがわかる．次に，別の例を fig .3.7 に示す．これは配列 A のスレッド ID 番目の要素に配列 B のスレッド ID*4 番目の要素から 4 つの連続する要素を足すプログラムである．このプログラムを k 個のスレッドで実行した時の配列 B のアクセスの様子を fig .3.8 に示す．k 個のスレッドのアクセス範囲は重複していない．この 2 つの例では，ブロック内のアクセス範囲は一致するが，fig .3.5 の方がブロック内での使用頻度が高い．従来の範囲解析ではアクセスするかどうかは解析できるが，使用頻度を解析することはできない．このように，実際のアクセス範囲を解析すればブロック内の各スレッドからのアクセス範囲が重複しているか重複していないかを判別することができる．しかし，実際に用いられる CUDA プログラムは大規模なデータを大規模なスレッド数で処理するためこのような方法で解析することは困難である．

そこで，ブロック内のアクセス回数をブロック内のアクセス範囲で割

ることで，配列の要素あたりの平均アクセス回数を求める．この値が 1 より大きい場合，ブロック内の各スレッドからのアクセス範囲が重複していると推測できる．ブロック内のアクセス回数は，スレッドあたりのアクセス回数とスレッド数の積で求めることができる．スレッドのアクセス回数は，ループ回数と一致するので for 文を解析することで取得できる．スレッド数は，カーネル関数呼び出し命令から取得できる．ブロック内のアクセス範囲は，配列の添字中のループ変数やスレッド ID にその最小値・最大値を代入することで求めることができる．例 1 ではブロック内アクセス回数は 4*k*k 回でアクセス範囲は 4*k であるので平均アクセス回数は k 回，例 2 ではブロック内アクセス回数は 4*k 回でアクセス範囲は 4*k であるので平均アクセス回数は 1 回という解析結果が得られ，これは実際のコードの挙動と一致している．

```
for (i=0;i<4*k;i++)
  A[threadIdx.x]+=B[i];
```

fig . 3.5: アクセス範囲が重なる例

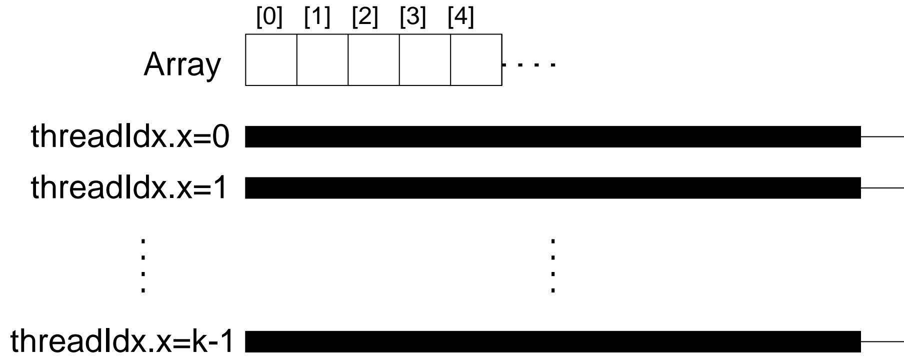

fig . 3.6: fig .3.5 のアクセスの様子

```
for (i=0;i<4;i++)
  A[threadIdx.x]+=B[4*threadIdx.x+i];
```

fig . 3.7: アクセス範囲が重ならない例

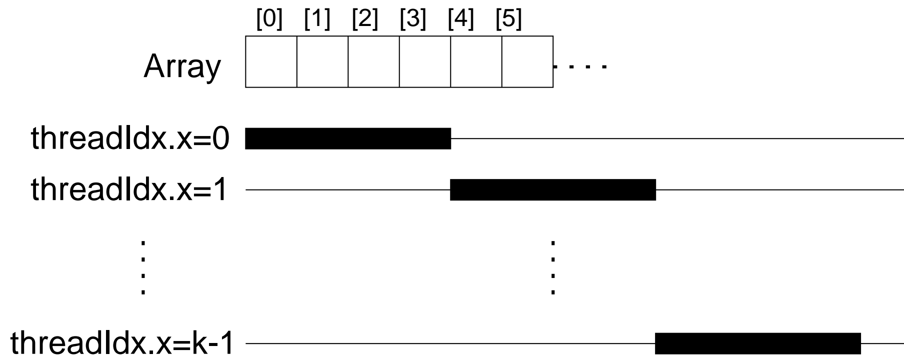

fig . 3.8: fig .3.7 のアクセスの様子

3.3 実装方法

前述の解析を行うために，複数のテーブルを実装した．プログラム中の for 文に対して Table 3.1 のようなエントリを作成する．fig .3.7 の場合，ループ変数名:i，ループ変数の変域:0〜3，ループ変数の変化間隔:1，ループ回数:4 という値を持つエントリとなり，これを for 文用のテーブルに登録する．このテーブル情報を用いることで，for 文中の配列の添字の変化を求めることができる．プログラム中の配列に対して Table 3.2 のようなエントリを作成する．スレッドの添字の変域，スレッドのアクセス回数は for 文用のテーブルを用いて添字を変化させて求めることができる．ブロック内の添字の変域，ブロック内のアクセス回数はカーネル関数呼び出し命令文から起動するスレッド数を探索して添字を変化させて求めることができる．fig .3.7 の場合，配列名:B，添字:4*threadIdx.x+i，スレッドでの添字の変域:4*threadIdx.x+0〜4*threadIdx.x+3，スレッドでのアクセス回数:4，ブロック内の添字の変域:0〜4*(k-1)+3，ブロック内のアクセス回数:4*k，平均アクセス回数:1 という値を持つエントリとなり，これを配列用のテーブルに登録する．

表 3.1: for 文用のエントリ

ループ変数名	i
ループ変数の変域	0〜3
ループ変数の変化間隔	1
ループ回数	4

表 3.2: 配列用のエントリ

配列名	B
添字	4*threadIdx.x+i
スレッドにおける添字の変域	4*threadIdx.x+0〜4*threadIdx.x+3
スレッドにおけるアクセス回数	4*k
ブロック内の添字の変域	0〜4*(k-1)+3
ブロック内のアクセス回数	4*k
平均アクセス回数	1

4 性能評価

4.1 評価内容

実装した解析機能の有用性を示すために，本機能で検出した配列にaaaa節の自動最適化を行うMESI-CUDAが生成したCUDAプログラムと従来のMESI-CUDAが生成したCUDAプログラムの実行時間の評価を行った．評価環境はCore i7 930 2.80GHz，メモリ6GB，TeslaC2050を搭載した計算機を使用した．評価に用いたプログラムは正方行列を対象とした行列積を求めるプログラムと逆行列を求めるプログラムである．正方行列の一辺の大きさを256，512，1024，2048の場合の実行時間を測定した．この結果をTable 4.3に示す．また，本解析機能によって実行時間が短縮しても解析時間が多大に掛かる場合，有用であるとはいえない．このため，本解析機能の実行時間を計測した．本機能はRubyを用いて実装した．評価環境はPentium 4 2.80GHz，メモリ1GBを搭載した計算機を使用した．この結果をTable 4.4に示す．

表 4.3: 実行時間 (秒)

		256	512	1024	2048
行列積	最適化	0.0282	1.346	2.956	7.551
	非最適化	0.0276	1.437	5.708	23.632
逆行列	最適化	0.969	3.042	20.190	53.19
	非最適化	1.520	6.434	44.730	120.751

表 4.4: 解析時間 (ミリ秒)

	配列数	実行時間
行列積	3	12.4
逆行列	10	46.9

4.2 考察

Table 4.3 からわかるように，両方のプログラムで実行時間が未実装の場合と比べ短縮されている．これは，本機能によって適切な配列のアクセス解析がされており，これによって前述したシェアードメモリを効果的に用いることが可能になりメモリアクセスのレイテンシが短縮されたためである．また，行列の一辺の大きさが大きいほど短縮率が大きくなっていることがわかる．これは，行列の一辺の大きさが大きいほどメモリアクセスが増えるのでその分シェアードメモリによるメモリアクセスのレイテンシの短縮量が増えたためである．また，解析時間においては Table 4.4

からほとんど掛からないことがわかる．以上から，本機能は有用であると考えられる．

5　おわりに

本研究では, MESI-CUDA に配列のインデックス解析機能を設計・実装し，評価を行った．その結果，本機能を用いることで適切な配列のアクセス解析が行われ，シェアードメモリを使用する最適化された CUDA コードの自動生成を達成した．また，解析もほとんど時間が掛からずに行うことができた．

今後の課題として，本研究では簡単な配列のアクセスにのみ対応しているが，多重ループ，多次元配列や多次元のグリッド，ブロックにおいても対応していく必要がある．また，本研究では配列アクセス文毎でしか解析していないが，カーネル関数全体で考えた場合での最適な配列を検出する必要もある．

謝辞

本研究を遂行するにあたり，日頃からご指導，ご助言をいただきました近藤利夫教授，大野和彦講師，佐々木敬泰助教に感謝いたします．また，様々な面でお世話になった計算機アーキテクチャ研究室の方々に感謝の意を表します．

参考文献

[1] 道浦悌, 大野和彦, 松本真樹, 佐々木敬泰, 近藤利夫. GPGPU におけるデータ転送を自動化する MESI-CUDA の提案. In 先進的計算基盤システムシンポジウム SACSIS2012, pp. 201–209, 2012.

[2] https://developer.nvidia.com/category/zone/cuda-zone. NVIDIA Developer Zone.

[3] 青木尊之, 額田彰. はじめての CUDA プログラミング. 工学社, 2009.

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

[5] http:/www.nvidia.co.jp/docs/io/51174/nvidia_cuda_programming_guide_1.1_jpn.pdf. NVIDIA CUDA Compute Unified Device Architecture プログラミング・ガイド（日本語版）.

A　プログラムリスト

B　評価用データ