卒業論文

題目

# 異なる性能のマルチGPUへのデータ振り分け手法

指導教員

大野 和彦 講師

4年

三重大学 工学部 情報工学科
計算機ソフトウェア研究室

水谷 洋輔 (407851)

# 内容梗概

近年，様々な分野で大規模計算の需要が高まっており，GPU に汎用的な計算を行わせる GPGPU に関心が高まっている．GPU を 1 基搭載して動作させるシングル GPU に対し，GPU を 複数搭載して並列動作させるマルチ GPU という技術が存在する．これを用いることでより高度な処理を行うことが出来るが，性能の異なる GPU を用いた環境ではそれぞれの GPU 性能に適したデータ振り分けを行わなければ性能を発揮することが出来ない．これに対処するため，静的振り分け手法と動的振り分け手法を提案する．静的振り分け手法は一定数に分割した仕事をそれぞれの GPU 性能に適した割合で配分する．動的振り分け手法では一定数に分割した仕事を，それぞれの GPU へ１つづつ振り分け，処理の終わった GPU へ仕事を配分していく．提案手法を用いることで，性能の同じ GPU に用いられる均等にデータ振り分けする手法より高速化することに成功した．

# Abstract

In recent years,the demand of large-scale calculation is increasing in various fields, and concern is increasing in GPGPU which makes general-purpose calculation perform to GPU. It is GPU to the single GPU which carries one GPU and operates it. More than one are carried and the technology of multi-GPU which carries out parallel operation exists. Although more advanced processing can be performed by using this, if data distribution suitable for each GPU performance is not performed, performance cannot be demonstrated in the environment using GPU from which performance differs. In order to cope with this, the static distribution technique and the dynamic distribution technique are proposed. The static distribution technique distributes the work divided into fixed numbers at a rate suitable for each GPU performance. By the dynamic distribution technique, the work divided into fixed numbers is distributed to each one GPU of every, and work is distributed to GPU which processing finished. It succeeded in accelerating by using the proposal technique from the technique of carrying out data distribution equally used for the GPU with same performance.

# 目 次

# 図 目 次

# 表 目 次

# 1　はじめに

## 1.1　背景

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．GPU を 1 基搭載して動作させるシングル GPU に対し，GPU を 複数搭載して並列動作させるマルチ GPU という技術が存在する．これを用いることでより高度な処理を行うことが出来る．

## 1.2　研究目的

シングル GPU に比べ，マルチ GPU を用いることで性能は向上する．しかし性能の異なる GPU を用いた環境では，それぞれの GPU 性能に適したデータ振り分けを行わなければ，シングル GPU の場合より性能が低下する場合がある．そこで本研究では性能の異なる GPU を用いたマルチ GPU へのデータ振り分け手法を考案及び評価した．

## 1.3 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GPU及びCUDAの概要について述べ，第３章に提案手法の説明，第４章に性能の評価を行い，最後に第５章にてまとめを行う．

# 2　背景

## 2.1　GPU と CUDA

GPU とは Graphics Processing Unit の略称であり，多くのプロセッサが集まった構造となっている．１つ１つのプロセッサの機能は CPU に比べ限定されているが，大量のデータを複数のプロセッサで同時かつ並列処理することが出来る．

CUDA[1] は NVIDIA 社より提供されている GPGPU 用の SDK であり，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 プログラムを容易に開発することができる．GPGPU プログラムの一覧の流れは，まず CPU 側 (ホスト) が必要なデータを作成し，GPU 側 (デバイス) に転送する．次にホストが GPU に実行させる関数 (カーネル) を起動させ，デバイスに処理を行わせる．デバイスの処理が終了した後，結果をホストへ転送し，デバイスから受け取ったデータをホストが処理をすることで終了する．

## 2.2　マルチ GPU

マルチ GPU とは，コンピュータ 1 台に対して複数の GPU を搭載して動作させる技術のことである．複数の GPU を用いることで，GPU の

数だけ並列処理を多く行うことが出来，またGPUに搭載されているメモリもより多く使用することが出来る．マルチGPU環境はその時点での最高級のGPUを複数搭載することが理想的ではあるが，コストが高くなる．そこで計算機の新調などで余ったGPUを利用して，マルチGPU環境を構築し，より高速な処理を行えるようにしたい． 同一のGPUを用いたマルチGPU環境はそれぞれのGPUは全く同一の性能であるため，データは均等に振り分ければよい．しかし，性能の異なるGPUを用いるときは，均等なデータ振り分けでは性能の劣るGPUの処理がボトルネックとなり，性能の優れたGPU単体で処理させた場合に比べ劣る場合がある．

# 3　提案手法

## 3.1　概要

2.2 節の問題に対処するため，性能の異なる GPU を用いたマルチ GPU 環境では，それぞれの GPU へ均等にデータ処理を行わせるのではなく，それぞれの GPU の性能に見合ったデータ量を適切に振り分ける手法が必要となる．そこで GPU へのデータ振り分け手法を 2 種類提案する．事前に GPU の性能を計測し，それに見合った割合でデータを振り分ける手法と，処理の終わった GPU へデータを振り分ける手法である．本論文では，前者を静的振り分けと呼び，後者を動的振り分けと呼ぶ．

### 3.1.1　非同期 API

本手法では CUDA の持つ非同期 API を用いる．この API を用いることで，依存関係のない演算とデータ転送を同時に実行することが可能である．依存関係のあるデータ転送と GPU 上で実行される関数はストリームで結びつけられる．ストリーム上ではデータ転送・演算がそれぞれ登録順に実行されていく．ストリームに登録された処理が終了したかは cudaStreamQuery 関数を用いることで判別可能である．マルチ GPU 環境では，この非同期 API を用い，それぞれの GPU に非同期で転送・演

算を行わせ，GPU を並列に動作させる必要がある．

また，CUDA4.0 からはカーネル実行のオーバーラップが可能である．最大 16 ストリームまで並列にカーネル実行を行わせることが出来る．カーネル実行をオーバーラップさせられる数はデバイス毎に決まっており，計算の重みが十分小さい場合にはマルチプロセッサの数と，マルチプロセッサの最大スレッド数に依存している．そのため，性能の異なるマルチ GPU 環境ではデバイス毎に適したストリーム数は異なる．

### 3.2 データ振り分け

対象となるプログラムのデータを一定数に均等分割し，分割したデータは手法に基いて 振り分けを行う．データを分割することで，分割されたデータ単位の転送と処理時間を短くし，GPU 間でのオーバーラップを高める事が可能である．分割されたデータは任意の GPU に 振り分けられるため，対象とするプログラムの処理を行うデータに依存関係はないものとする．本論文ではデータは 16 に分割することとする．

最適な分割数はデバイス毎に異なる．計算の重みが十分小さい場合は分割したデータ内のスレッド数がデバイスでの最大スレッド数になるような分割数が最適であると考えられる．

各 GPU にはそれぞれストリームを用意し，命令はストリームを用いて

行う．fig .3.1 に二次元配列をデータ分割し，複数ストリームへの振り分けを行ったモデルを示す．また，単一 GPU に複数ストリームを準備することでカーネル実行のオーバーラップにより処理を高速化することができる．詳細は 4.2.1 節に後述する．本論文では各 GPU に 8 ストリームを用意することとする．

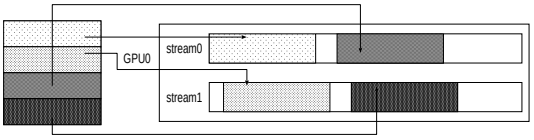


fig . 3.1: データ振り分けモデル

## 3.3 静的振り分け

静的振り分け手法では，分割されたデータをそれぞれの GPU へ交互に振り分ける．一方の GPU へ 最適な量を振り分けた時点で，その GPU への振り分けを止め，もう一方の GPU へ残りのデータを振り分ける．最適量を調べるため，事前に対象のプログラムを少ないデータ量で処理させ，振り分け割合の計測を行う．同一プログラムであれば，データ量によって最適な振り分け割合は変わらないものとする．詳細は 4.1.3 節に後述する．

### 3.4 動的振り分け

動的振り分け手法では，各GPUに２セットのストリームを用意する．それぞれのストリームへ 分割されたデータを１セットずつ，１つのGPUに 計２セットのデータを転送・演算させる．まずcudaStreamQuery関数を用い，ストリームの処理の終了を待つ．次に処理の終わったストリームへデータの振り分けを行い，ストリームの 処理の終了を待つ．この工程をすべてのデータを振り分けるまで続ける．これにより，処理速度の早いGPUに優先的にデータ振り分けがされ，最適な振り分けを行う．

### 3.5 手法比較

静的振り分けでは事前に最適配分量を調べる工程が必要となるが，同一プログラムであればデータ量に依らず再度最適配分量を計測する必要はない．また，振り分けのオーバーヘッドが小さいと考えられる．動的振り分けでは，事前の最適配分量を調べる必要がなく，データ量に依る変化にも対応をすることができるが，処理速度の早いGPUの処理終了直前に最後のデータを処理の遅いGPUへ振り分けた場合に，処理速度が遅くなる可能性がある．これはデータ分割数を多くすることで軽減が可能である．

# 4 性能評価

## 4.1 評価環境

提案した手法の有効性を示すために，3.1 節の手法を用いた CUDA プログラムと GPU 単体での CUDA プログラム，データ振り分け制御を行わないプログラムでの実行時間の評価を行った．評価環境は Core i7-3820 3.60GHz，メモリ 16GB，GTX680，TeslaC2075 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である．正方行列の一辺の大きさ $N$ が 1024，2048，4096 の場合の実行時間を計測した．この結果を Table 4.1 に示す．

CPU での演算に比べ GPU を用いたプログラムの実行時間が早いことは明らかである．

表 4.1: 振り分け手法による実行時間 (秒)

| | 1024 | 2048 | 4096 |
|---|---|---|---|
| CPU 単体 | 1.6273 | 76.759 | 698.11 |
| TeslaC2075 単体 | 0.0829 | 0.4274 | 3.8116 |
| GTX680 単体 | 0.0570 | 0.2925 | 2.3563 |
| 均等振り分け | 0.0433 | 0.2201 | 2.4483 |
| 静的振り分け | 0.0365 | 0.1891 | 1.5445 |
| 動的振り分け | 0.0365 | 0.1954 | 1.6294 |
| 動的振り分け (最適化) | 0.0283 | 0.1831 | 1.4300 |

#### 4.1.1 均等振り分け

GPU 単体での実行時間に比べ，均等にデータ振り分けしたプログラムの実行時間は概ね短くなっているが，$N$=4096 となるデータでは単体よりも長くなっている．これはデータを一定数に分割を行わずに，データを二等分しそれぞれの GPU へ振り分けているため，１回のデータ転送と演算にかかる時間が長くなり，オーバーラップが低くなっているからだと考えられる．

#### 4.1.2 提案手法

均等にデータ振り分けしたプログラムに比べ，提案手法を用いたプログラムでは明らかに実行時間が短くなっていることがわかる．これはデータ分割により，データ転送開始時の片側の GPU のみでデータ転送・演算が行われている時間が短くなり，更にそれぞれの GPU の処理性能に適したデータ振り分けが行われたことで，オーバーラップが高まったためであると考えられる．

動的振り分け手法では静的振り分け手法よりも実行時間が長くなっているのは，振り分けの際のオーバーラップ，及び GPU への振り分け比率が一定ではないことが考えられる．動的振り分け手法では TeslaC2075 へ

の振り分け割合は凡そ 31.25-37.5 %であった．

### 4.1.3 データサイズによる振り分け割合

動的振り分け手法での振り分け割合は変化した．そこで静的振り分け手法での振り分け割合による実行時間の変化を調べる．評価環境での，TeslaC2075 へのデータ振り分け割合を基に，データサイズ毎のプログラム実行時間を Table 4.2 に示す．プログラムは評価に用いたものと同一のものとする．Table 4.2 からわかるように最も実行時間が短い割合はどのデータサイズであっても，37.5 %の時であるということがわかる．

表 4.2: 振り分け割合による実行時間 (秒)

| | 0 % | 31.25 % | **37.5 %** | 43.75 % | 100 % |
|---|---|---|---|---|---|
| 1024 | 0.0570 | 0.0389 | **0.0353** | 0.0382 | 0.0848 |
| 2048 | 0.2925 | 0.2051 | **0.1864** | 0.1941 | 0.0433 |
| 4096 | 2.3563 | 1.6389 | **1.5449** | 1.6852 | 3.8107 |

## 4.2 最適化

提案手法では操作可能なパラメータがいくつか存在する．ここでは，振り分け割合の計測が必要がなく，パラメータの操作が容易な動的振り分け手法の各パラメータの最適化を行う．

### 4.2.1 ストリーム数

ストリーム数による実行時間の変化を調べた．GTX680 を 1 台に対し，評価に用いたプログラムのストリーム数を変化させたときの実行時間を Table 4.3 に示す．どのデータサイズであってもストリーム数が増える毎に実行時間は約半分ほど短くなっている．これはストリーム数に応じてオーバーラップが高まっているからだと考えられる．しかし一定のストリーム数から実行時間に変化は少なくなり，実行時間が長くなっている例も見られる．また，データサイズが大きくなるにつれ，実行時間の変化が少なくなるストリーム数は少なくなっている．

これはカーネル実行のオーバーラップが，GPU のマルチプロセッサ，またはコアに仕事をしていないものがあった場合に他のオーバーラップ可能なストリームのカーネル実行を処理しているからだと考えられる．そして，データサイズが増えることでストリーム毎に割り当てられる処理スレッド数が増え，ストリーム毎に割り当てられるマルチプロセッサ，もしくはコア数が増えたことにより，空いたマルチプロセッサに処理を振ることが出来なくなったために，ストリーム数を増やすことでの実行時間の短縮が行えなくなったと考えられる．そのため，ストリームはデバイスのマルチプロセッサ数に準じた数を用意しておき，ストリームへは

マルチプロセッサの最大スレッド数を考慮したカーネル実行時のスレッド数を設定するべきだと考えられる．

表 4.3: ストリーム数による実行時間 (秒)

| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| 1024 | 0.3575 | 0.1829 | 0.0964 | 0.0548 | 0.0388 | 0.0383 |
| 2048 | 1.4232 | 0.7507 | 0.4219 | 0.2880 | 0.0302 | 0.3020 |
| 4096 | 6.1360 | 3.5560 | 2.3790 | 2.3518 | 2.3536 | 2.3506 |

### 4.2.2 分割数

データ分割数の最適化を行った．今回は振り分け割合を計測する必要のない動的振り分け手法を用いる．TeslaC2075 のデバイス毎の最大スレッド数は 21504 であり，GTX680 のデバイス毎の最大スレッド数は 16384 である．今回は GTX680 に合わせて最適化を行った．分割した処理に含まれるスレッド数を 16384 とするため，分割数はそれぞれ $N$=1024,2048,4096 での分割数を 64,256,1024 とした．その結果，静的振り分け手法より高速化することに成功した．データ分割数の最適化では TeslaC2075 への振り分け割合はそれぞれのデータサイズで 31.25 %,41.01 %,39.84 %前後であった．

Table 4.2 からはデータサイズに依らず最適な振り分け割合は一定であるように見られるが，データサイズ毎の演算の重みにより最適なデータ

分割数は変化し，データ分割数毎に最適な振り分け割合は変化するものであると考えられる．

# 5　おわりに

本研究では，性能の異なる GPU を用いたマルチ GPU 環境に適したデータ振り分けを行う手法を提案・実装し評価を行った．本手法を用いることで GPU の使用効率が上がり，実行時間の高速化を行うことが出来た．

今後の課題として，分割したデータに依存関係がある場合にも対応する必要がある．

# 謝辞

本研究を行うにあたり，ご指導，ご助言いただきました下さいました大野和彦講師に深く感謝いたします．また，様々な局面にてお世話になりました計算機ソフトウェア研究室の皆様にも心より感謝いたします．

# 参考文献

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

# A　プログラムリスト

## A.1　カーネル起動

カーネル起動時のスレッド数は，デバイスのマルチプロセッサの最大スレッド数で割り切れる数にすると良い．性能が異なるマルチ GPU 環境下では，デバイス毎にカーネル起動時のスレッド数は変えた方が良いと考えられる．

## A.2　ストリーム割り当て

作成したプログラムでは，行列を行単位で分割している．振り分けでは分割されたデータを１行毎にカーネル実行させ，ストリームに割り振っている．

## A.3　動的振り分け

動的振り分けでは分割したデータでの仕事が終わったかどうかを調べるためにストリームを２セット用意している．それぞれのセット内のストリーム数はデバイス毎の最適ストリーム数であると良い．それぞれのセットに仕事をさせ，無限ループによりストリームの仕事終了を監視する．仕事が終わったストリームのセットへ分割したデータの仕事を追加

していく．こうすることで，ストリームへの仕事追加する際のオーバーヘッドをオーバーラップし隠すことが出来る．

### A.4　注意事項

マルチ GPU の場合，デバイス毎にメモリ確保をすることになる．一方のデバイスで確保したメモリ領域に，もう一方のデバイスでアクセスすると正しい結果が返されない．メインメモリを経由した転送が必要となる．GPU Direct ver.2 ではそうした場合でもアクセス可能であるとされている．

上記のような場合が存在するため setDevice により使用するデバイスを指定する場合には注意が必要である．

## B　評価用データ