GPGPUに対する理論モデル

鈴木, 拓也 / SUZUKI, Takuya

(発行年 / Year)
2013-03-24
(学位授与年月日 / Date of Granted)
2013-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2012 年度

指導教授　和田　幸一教授

論文題名

GPGPU に対する理論モデル

法政大学大学院工学研究科

情報電子工学専攻修士課程

学籍番号　11R4137

氏名　鈴木　拓也

This thesis proposes a new theoretical model for GPGPUs(General Purpose on GPUs), which are designed for general purpose computing by using a specialized processing unit to accelerate computation for building and manipulating images. Unlike PRAMs(Parallel Random Access Machines), which are typical theoretical models for parallel computers,Several characteristics of GPGPU, such as bank conflicts of the shared memory access and the coalescing of the global memory are taken into account in our model. We propose two variants of GPGPU models and verify the feasibility of our models using implementation of some parallel sorting algorithm.

目次

第1章　はじめに

近年，汎用問題に対してGPU(Graphic Processing Units)の並列高速処理能力を利用できるGPGPU(General Purpose on GPU)が注目されてきた．多数の演算器を持ち，並列計算可能なGPUは，CPUをはるかに上回る演算性能を持つようになり，GPUを用いて数値計算を高速化する研究が多くなされている．従来はCPUで行っていた計算をGPUの並列性を有効に活用して計算する事を可能にしたインターフェースが，GPU向け開発計算環境CUDA(Compute Unified Device Architecture)[1]である．

並列コンピューティングに適用可能なアルゴリズムを設計するための理論モデルとしてPRAM(Parallel Random Access Machine)が有名だが，GPUは従来のプロセッサとは異なる点が多いため，理論モデルとするには適当ではない．今回，GPUの特徴を捉えた理論モデルを提案する．この理論モデルを作成するにあたって，なるべくマシンの特徴を捉えつつシンプルなモデルを作成することが重要である．

本稿は，GPU理論モデルのDMM・UMM[2]とAGPU[3]を参考にし，よりシンプルでマシンの特徴を捉えた理論モデルを提案した．今回，この理論モデルで扱った問題は並列マージソートである．この理論モデルを応用し，行列積などの問題も理論値を求めることが可能である．

本稿の第2章では本研究で使用しているGPU ，GeForce GTX 550 Tiの説明とNVIDIA社が提供するGPU向けのC言語の統合開発環境であるCUDAの説明をする．
第3章では，DMMとUMMについて述べる．第4章では，AGPUについて述べる．第5章では実験としてCole’s merge sort algorithm[4] を用いてGPUとCPUとのマージソートの処理時間の比較を行い，その違いについての検証，さらに，DMM・UMMを統合した理論モデルの理論的計算時間の比較を行い，最後にそれらについての考察を述べる．

第2章　GPGPU

近年，コンピュータの性能は格段にあがり，パソコンであっても少し前では信じられないような機能が実現できるようになった．ビデオカード，つまりGPU(Graphics Processing Unit)の性能向上も著しく，髪の毛の一本一本まで細かく，かつ高速に表示することが可能になった．3Dグラフィックス処理の分野では，長年，並列処理技術が利用されてきた．汎用的な計算を得意とする一般のマイクロプロセッサ（CPU）で同じ処理を実行した場合，GPUほどのグラフィックス処理能力を得ることはできない．このためGPUは，グラフィックス・ボードの形でパソコンに組み込まれたり，PlayStation3やWiiといった家庭用ゲーム機にも搭載されたりしている．さらに，ここ数年ではGPUは演算性能，メモリバンド幅とともにCPUを大きく上回るようになった．

現在では，3D(3次元)グラフィックスを扱うゲームや3D CAD(Computer Aided Design)などにGPUが使用されている事はもちろん，体計算，電磁波シミュレーション，天文シミュレーション，たんぱく質などの挙動を解析する数値シミュレーション，バイオ・インフォマティクス，金融工学など， 幅広い分野への適用が試みられている．このような汎用的な問題に対してGPUの高速並列処理能力を適用させる「GPGPU（General Purpose Computation on Graphics Processing Unit）」，あるいは「GPUコンピューティング」と呼ばれる技術がここ数年，注目を集めている．

過去30年間にわたって，コンシューマ向けコンピューティングデバイスのパフォーマンスを向上させるために，プロセッサのクロック速度を引き上げる方法があった．1980年代の初めに最初のパーソナルコンピュータが登場した頃，コンシューマ向けのCPUはおおよそ1MHzの内部クロックで動作していた．30年後に登場したデスクトッププロセッサのほとんどは，クロック速度が1〜4MHzで，最初のパーソナルコンピュータのクロック速度の約1000倍になった．

しかし，集積回路構成の制限により，最近のCPU計算性能の向上に対して，クロック速度の限界が生じてきた．トランジスタのサイズが物理的な限界に迫っている事に加えて，電力や耐熱にも制限がかかる．

コンシューマコンピューティング以外の分野では，スーパーコンピュータが数十年にわたってクロック速度を上げることで計算能力向上を実現していた．その一方で，スーパーコンピュータはシングルプロセッサの性能を劇的に向上させる事と，プロセッサ自体の数を増やすことで性能を大きく飛躍させてきた．トップクラスのスーパーコンピュータでは，数万あるいは数十万ものプロセッサコアが並行して動作している．

スーパーコンピュータのマルチプロセッサのアイディアをパーソナルコンピュータに反映させ，大手CPUベンダーは2005年にプロセッサコアを2つ搭載させたプロセッサをリリースした．その後，3,4,6,8コアのCPUをリリースし，並列処理の流れに追従した．

現在のGPUには，プロセッサコアが32コア以上搭載され，一度に65535×512スレッドの並列処理が可能である．同時に単純な命令のみを処理させるならば，多くのケースでCPUよりもGPUを使った処理速度の方が速い．

2．1　GeForce GTX 550 Tiのアーキテクチャ

GPUというプロセッサの中の一つに本研究で使用するGeForce GTX 550 TiというGPUコンピューティングプロセッサボードがある．

GPUの機能である3Dグラフィックス処理をそれ以外の汎用的な計算行わせるようにできたのが「GPGPU（General Purpose Computation on Graphics Processing Unit）」，もしくは「GPUコンピューティング」と呼ばれる．

GPGPUは，画像処理はもちろんのこと，流体計算，電磁波シミュレーション，天文シミュレーション，たんぱく質などの挙動を解析する数値シミュレーション，バイオ・インフォマティクス，金融工学など， 幅広い分野への適用が行われている．

GPUコンピューティングは，GPGPUよりも，より汎用的なHPCの実現に向けた取り組みがGPUコンピューティングである．GeForce GTX 550 TiはこのGPUコンピューティングに特化したプロセッサボードである．

GeForce GTX 550 Tiは，GPUのメーカーであるNVIDIA社が開発したGPUコンピューティングプロセッサボードの1つである．GeForce GTX 550 Tiシリーズは GPUの構成から，映像の出力機能をカットしたものである．Teslaの特徴としてはその精度が挙げられる．

本研究で使用しているGeForce GTX 550 TiはGeForceシリーズの一つであり，240個のプロセッサコアに1296MHzの動作を行うことができる．GeForce GTX 550 Tiは科学者やアナリスト，その他の技術専門家のためのHPCアプリケーションに膨大なマルチスレッドアーキテクチャ を初めてもたらしたものである．

Tesla C1060にはGPUコンピューティングに特化したGeForce GPUを1基搭載しており，240基のCUDAプロセッサコアにより，最大933GFLOPSの単精度浮動小数点演算性能を実現できる．また，従来では単精度演算のみであった処理を倍精度による演算も可能となり，極めて複雑な集約的計算の解決を可能にする．GeForce GTX 550 Tiの詳細を表1に示す．

表1：GeForce GTX 550 Tiの詳細

GeForce GTX 550 Ti	GPU名:NVIDIA Tesla C1060
搭載GPU数	1基
CUDAコア数	240基
プロセッサ周波数	1296MHz
単精度演算性能	最大933GFlops
搭載メモリ	4GB
メモリインターフェース	GDDR3 SDRAM
浮動小数点	IEEE 754 単精度 / 倍精度浮動小数点
ホスト接続	PCI Express x16(PCI-E2.0対応)
メモリ転送帯域	102 GB / sec
TDP	187.8 W(標準値)
最大消費電力	225 W

2.2 CUDA

CUDA とは，GPU を用いて汎用計算処理のプログラムを開発するための C 言語の総合開発環境である．NVIDIA 社から提供されており，本研究ではこの NVIDIA 社の GPU を対象としている．

CUDA の特徴として，まず GPU 側で処理をされるカーネル関数は，スレッドと呼ばれる実行単位で実行される．そして全てのスレッドで同じプログラムが走る．他の特徴として，スレッドはそれぞれ固有のスレッド番号を持っていることが挙げられる．この固有のスレッド番号を用いて，同じプログラムでもそれぞれのスレッドに固有の処理を並列して行わせることが出来る．

CUDA でのプログラムの流れは，CPU 側をホスト，GPU 側をデバイスと呼ぶ．ホスト側のプログラムは CPU 上で動作し，ホストのメモリを使用する．デバイス上で動作するプログラムをカーネルプログラムと呼び，これば GPU 上で処理される．

CUDA でのプログラムの流れは以下の通りである．

1．ホスト側でプログラムを開始する
2．デバイス側でカーネルプログラムを読み込む
3．ホスト側のデータをデバイス側に転送し，カーネルプログラムを起動させる
4．デバイス側でカーネルプログラムを処理する
5．ホスト側へデータを転送する

また，CPU から GPU に対して起動される GPU プログラムをカーネルと呼ぶ．カーネルはマルチスレッド化されており，グリッド，グロック，スレッドの三つのスレッド階層で構成される．まず，グリッドについては階層の最上位であり，1 カーネルにつき一つのグリッドが割り当てられる．一つのグリッドは複数のブロックから構成される．ブロックは同一 SM に属するスレッドの集合であり，ブロック内のスレッドは同一のプログラムを実行する．CUDA はブロックをどの SM に割り当てるのか物理的に指定してくれるので，プログラマはブロックの SM へのアドレス指定をする必要はない．スレッドの命令は SP(Stream Processor)により実行される．この SP はコアとも呼ばれる．ハードウェア・マルチスレッディング(プロセッサのマイクロアーキテクチャにおいて複数のスレッドの実行をハードウェアで提供する事)を行うため，各 SP には複数のスレッドが割り当てられる．

CUDA ではブロック内のスレッドは 32 個ごとにまとめられ，32 個のスレッドは同一の命令を常に同期して実行する．このグループをワープと呼ぶ．これは SM 内の SP が SIMD として動作することに対応する．Tesla アーキテクチャの場合，SM 内の 8 個の SP は 4 クロックサイクルで 1 ワープ(32 スレッド)に対して命令実行を行う．CUDA プログラミングにおいては，明示的にワープに対する処理を記述することはないが，処理性能を高めるためには，ワープを考慮した設計を行うことが大切である．例えば，条件分岐においては，ワープ内

のすべてのスレッドが同じ実行パスを通るようにすることによって実行効率を高めることができる．実行パスがスレッドによって分かれると，その両方の処理が別のタイミングで実行されることになり，効率も下がる．また，後述するが，メモリアクセスについてもワープを考慮することが大切である．

CUDA ではワープ内のすべてのスレッドは常に同一の命令を実行するので，メモリアクセス命令においてもワープを意識してのプログラミングが重要である．そこで，すべてのスレッドが効率よく同時にデータを所得するための仕組みもプログラマに提供されている．まず，前提知識としてグローバルメモリと共有メモリの仕組みについて以下に述べる．CUDA の世代により，この二つの仕組みは若干異なるが，ここでは，Tesla アーキテクチャに対応した CUDA を取り上げる．

グローバルメモリへのアクセスは，ワープを半分ずつ二つに分けたハーフワープの単位で行われる．グローバルメモリは 64 バイトごとのブロックに分けられており，ハーフワープ内の 16 スレッドがこのブロックのデータに 4 バイトずつ順にアクセスする場合，このアクセスは 1 回の 64 バイトアクセスにまとめられる．このように複数のスレッドのメモリアクセスを一回のメモリアクセスにまとめて実行することをコアレスアクセスまたはコアレッシングアクセスと呼ぶ．なお，最新の CUDA ではグローバルメモリアクセスはワープ単位で行われ，コアレッシング可能な条件もより柔軟になっている．グローバルメモリへのアクセスレイテンシ(転送速度)は 400〜600 クロックサイクルと言われている．

共有メモリアクセスもハーフワープの単位で行われる．共有メモリは 16 個の物理的なバンクに分かれている．アドレスを 16 で割った際の余りが 0 となるデータがバンク 0 に割り当てられており，同様にアドレスを 16 で割った際の余りが q となるデータはバンク q に割り当てられる．共有メモリアクセスの際，ハーフワープ内の 16 個のスレッドがすべて異なるバンクにアクセスする場合はメモリアクセスは一度(同時)に行われる．しかし，複数のスレッドが同一のバンクにアクセスする場には，それらの処理はシリアライズされる．この状況をバンクコンフリクトと呼ぶ．なお，最新の CUDA では共有メモリアクセスもワープ単位で行われる．転送速度の差は，グローバルメモリ，キャッシュ，共有メモリ，レジスタの順番で速くなる．

また，他にもコンスタントメモリ，テクスチャメモリと呼ばれるメモリが存在するが，本稿では触れていないので，割愛する．

第3章　従来のGPGPU理論モデル

3.1　PRAM

PRAM(並列ランダムアクセス機械)は，並列コンピューティングに適用可能なアルゴリズムを開発するための理論モデルであり，プロセッサ間の同期や通信を省き，並列性をいかに引き出すかを議論することができる．フリンの分類によれば，PRAMはMIMD型コンピュータに相当する．PRAMは，複数のRAM(ランダムアクセス機械)とレジスタセットと入力レジスタと出力レジスタからなり，各RAMが独立にプログラムカウンタを持っている．レジスタはすべてのRAMに共通であり，各RAMはアキュムレータを持つ．PRAMでは，複数のCPUが同時に共有メモリの同じ箇所をアクセスする可能性がある．PRAMモデルにはこのような事態を想定して，4つのPRAMモデルが存在する．それらのモデルの違いは，共有メモリ上の同じ箇所への同時アクセスを許すか禁止するかである．この時のアクセスとは，読み取り(Read)，書き込み(Write)である．

1. 排他的読み取り/排他的書き込み(Exclusive Read Exclusive Write, EREW):
   各メモリセルはある時点では1つのプロセッサだけが読み書きできる．
2. 並行的読み取り/排他的書き込み(Conccurrent Read Exclusive Write, CREW):
   読み取りは同時に行えるが，書き込みは一度に1つのプロセッサのみである．
3. 排他的読み取り/並行的書き込み(Exclusive Read Conccurrent Write, ERCW):
   書き込みは同時に行えるが，読み込みは一度に1つのプロセッサのみである．
4. 並行的読み取り/並行的書き込み(Conccurrent Read Conccurrent Write, CRCW):
   複数のプロセッサが自由に読み書きできる．

3.2　DMM·UMM

GPGPUの時間計算量を測定するために提案された理論モデルにDMM(Discrete Memory Machine)とUMM(Unified Memory Machine)がある．既に広く普及しているPRAM(Parallel Memory Model)のような理論的並列計算マシンとは違い，この二つのモデルはNVIDIA社のGPUのメモリアクセスの特徴を良く捉えている．

NVIDIA社は，CUDA(Compute Unified Device Architecture)と呼ばれる並列計算アーキテクチャを提供している．CUDAは，開発者に対して仮想命令セットを用いたGPU内のメモリアクセスを可能にさせている．GPUは数百のプロセッサコアを搭載しているので，マルチコアプロセッサよりも多くのケースで効率的かつ高速に処理を行う．

CUDAはGPU内の二つのメモリタイプを使用する．グローバルメモリとシェアードメモリである．グローバルメモリは，off-chip DRAMとして実装され，約1.5Gytesという大きなメモリを持つ．しかし，アクセスレイテンシはシェアードメモリに比べて大きい．

シェアードメモリは，グローバルメモリに比べてアクセスレイテンシが小さくが，メモリ容量が16-64K bytesと小さいon-chipメモリである．グローバルメモリとシェアードメモリをできるだけ効率的に使用することは，GPUを用いるアプリケーションの実行速度を上げることにつながる．特に，考慮すべき事は，グローバルメモリのコアレスアクセス，シェアードメモリのバンクコンフリクトである．GPUとDRAMチップ間のバンド幅を最大にするために，グローバルメモリの連続アドレスをアクセスすることが必須である．したがって，グローバルメモリへのアクセス時，スレッドはコアレスアクセスをすべきである．

シェアードメモリのアドレス空間は，複数の物理メモリバンクに分けられる．もし二つ以上のプロセッサコアが，同じメモリバンクにアクセスする場合，アクセスリクエストはシーケンスに処理される．　DMMとUMMのアーキテクチャを図1に載せる．

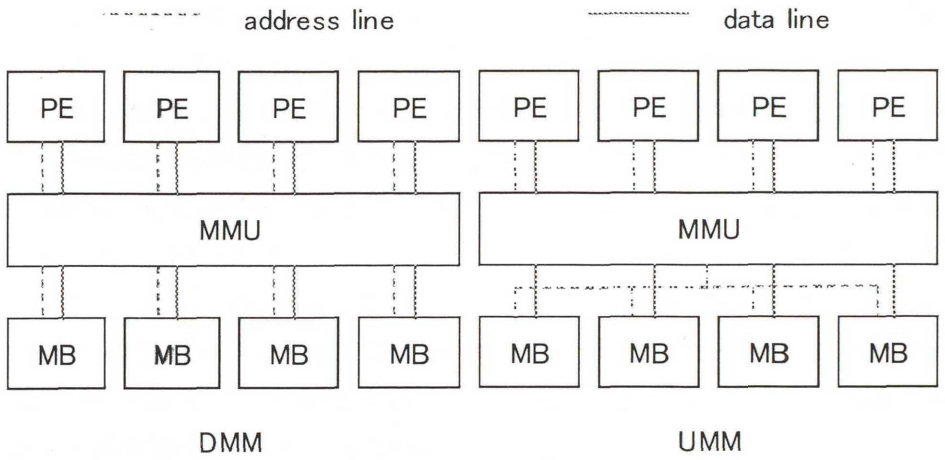

図1:DMMとUMMアーキテクチャ

両方のアーキテクチャ内のPEはプロセッサエレメントであり，メモリ制御ユニット(Memory Management Unit, MMU)を通してメモリバンク(MB)と連結している．MBが表しているメモリは，DMMでは共有メモリ，UMMではグローバルメモリである．MBは，メモリインターリーブ構成なので，アドレスiをメモリバンク数wで割った余りで該当するバンクにアクセスする．例えば，メモリのアドレス0にアクセスする場合，メモリバンク数が16の場合は 0 mod 16により0-th bankにアクセスする．

DMMとUMMのアーキテクチャの主な違いは，MBへのアドレスラインの接続方法である．DMMはそれぞれのMBへ異なるアドレスラインが接続されているが，UMMはすべてに一本のアドレスラインが接続されている．つまり，UMMはMB内の同じアドレス部分のデータのみ一度にアクセスができるということである．DMMはMB内の異なるアドレスに一度にアクセスできる．UMMはDMMと比べて制限が強いモデルである．

3.3 AGPU

AGPU モデルは多くの GPU に共通する特徴を抽象化した計算モデルである．AGPU モデルでは，処理性能に影響する特徴のみにフォーカスし，それ以外の部分は極力単純化している．まず，AGPU モデルの詳細を説明した後，CUDA との違いについて説明する．そのあと，他の計算モデルとの関係について述べる．

AGPU モデルのアーキテクチャを図 2 に示す．AGPU モデルのアーキテクチャは並列計算を行うためのデバイス(GPU)とデバイスを制御するためのホスト(CPU)の異種混載システムとなっている

プロセッサエレメント(以下 PE)はコアであり，デバイスは p 個の PE を備えている．PE は単位時間に一つの命令を実行することができる．また，PE のワード長を w ビットとし，PE はワード単位でデータにアクセスする．

AGPU モデルのデバイスは k 個の計算ユニット(以下 CU)で構成されており，各 CU は b 個の PE を備えている．すなわち p=kb である．CU はホストから起動されたプログラムを個別に実行する．各 CU は一つのプログラムのみを実行可能である．すなわち，CU は高々一つのプログラムのみを実行可能である．また，CU 内の PE はそれぞれ一つのスレッドのみを処理する．CU は他の CU との通信手段および同期手段を持たない．代わりに，CU は後述のグローバルメモリに値を書き込むことにより，ホストにデータを渡すことができる．ホストはすべての CU がプログラム実行の完了を待つことにより，CU 間の同期を行うことができる．

CU 内のすべての PE は常に同一の命令を実行する．ただし，オペランドに指定されるデータアドレスについては PE ごとに指定することができる．また，命令には実行条件を含めることができ，条件を満たす PE のみ命令を実行させることができる．

デバイスは 2 種類のメモリを備えている．一つ目はグローバルメモリで，データ転送速度が低速であるが比較的大容量という特徴を持つ．また，すべての CU 及びホストからアクセスすることができる．二つ目は共有メモリで，各 CU は内部に共有メモリを備えている．これはデータ転送速度が高速であるが，低容量という特徴を持つ．また，CU 内部の PE からのみアクセス可能である．

共有メモリは b 個の物理バンクに分かれている．各バンクにおいて，単位時間あたり一つのデータにアクセスできる．AGPU モデルでは，バンクコンフリクトが発生するような命令は発行できないものとする．すなわち，ユーザープログラムにおいて，常にバンクコンフリクトが発生しないようにしなければならない．このように制限をかけることによって，アルゴリズムの計算時間の解析を容易にすることができる．

グローバルメモリはデータサイズ bw ビットのグロックに分割されている．すなわち，1 ブロックには b ワードのデータが格納される．メモリアクセスは常にブロック単位で行われるものとする．よって，一回のメモリアクセスで同時に b ワードにアクセスできるものとする．AGPU モデルでは，グローバルメモリへのアクセス命令は以下の二つのみとする．一つ目は Read 命令であり，グローバルメモリの 1 ブロックを指定の共有メモリアドレスに

コピーする命令である．二つ目はWrite命令であり，グローバルメモリの1ブロックに対し指定の共有メモリアドレスのデータをコピーする命令である．PEがグローバルメモリのデータにアクセスする場合，グローバルメモリを一旦，共有メモリにコピーしてから共有メモリのデータにアクセスすることになる．つまり，この処理はCUDAでは1命令だが，AGPUでは2命令となる．しかし，これは計算量のオーダーには影響を与えない．また，上記のようにすることで，コアレスアクセスが可能な場合には，常にコアレスアクセスされるようになり，ユーザープログラムにおいてコアレスアクセスについて意識する必要がなくなる．

CUDAとAGPUの一番大きな違いは，CUDAでは各SPが複数のスレッドを処理することである．一方，AGPUではPEは高々一つのスレッドのみを処理する．CUDAでは，SMはハードウェア・マルチスレッディングの機構を備えており．これを利用してグローバルメモリアクセスのレイテンシを隠蔽することで高速処理を実現している．AGPUでは，PEは一つのスレッドしか処理していないものの，各CUは個別にプログラムを実行しており，これは実際の処理においてマルチスレッディングと同様の効果を持つ．そのため，AGPUを使用することで，実際にも高速に動作するアルゴリズムを設計することが可能である．

CUDAとAGPUの次の違いは，ブロックの有無についてである．CUDAはスレッドの階層としてブロックを持っているが，AGPUにはブロックは存在しない．ブロックは同一の共有メモリを持つワープ集合体である．ブロック内のワープは他のワープとの同期手段を持っており，共有メモリを読み書きすることにより，他のワープとのデータのやり取りが可能である．一方，AGPUではCUは他のCUとの通信手段や同期手段を持たないため，代わりにグローバルメモリを使用してホストに通知することになる．よって，CUDAで記載したアルゴリズムのほうがAGPUで記載したアルゴリズムよりもグローバルメモリアクセス回数が少なくなる傾向にある．別の言い方をすれば，AGPUでのグローバルメモリアクセス回数は，CUDAでのグローバルメモリアクセス回数の上限となっている．

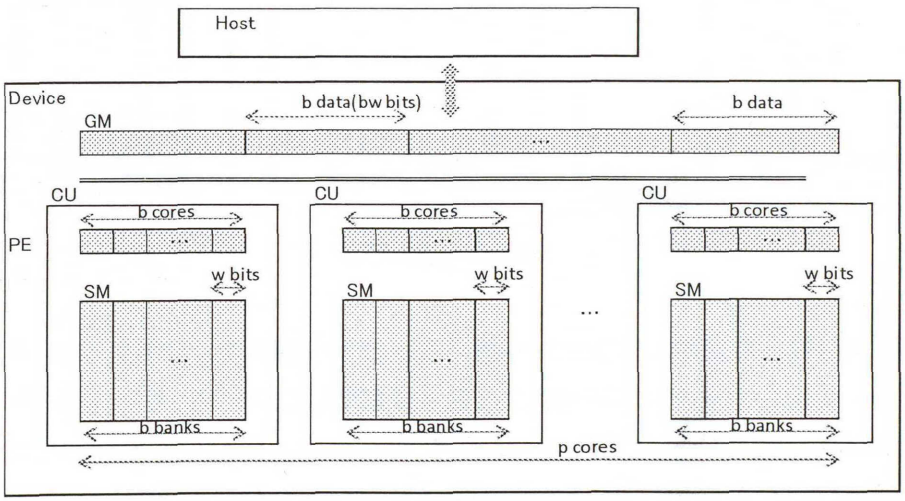

図2：AGPUモデルのアーキテクチャ

第4章　提案する理論GPGPUモデル

4.1　統合GPGPUモデルについて

提案するGPGPUモデルは,MPをPRAMもしくはDMM,MPを含めたモデル全体をUMMとした構成になっている．前章に記述したDMM･UMMを参考にし,新しく提案するモデルに合わせてDMM・UMMを拡張/変更している．

統合GPGPUモデルは,DMM・UMM同様,GM(グローバルメモリ)のコアレッシングとSM(シェアードメモリ)のメモリコンフリクトの特徴を踏まえて構成している．対象となるGPUはNVIDIA社のGPUである.統合GPGPUモデルは図3に示した.MP内のコアは同じMP内のSM(シェアードメモリ)にアクセスが可能である．コアはプロセッサであり,CUDAプログラミング上ではスレッドとも呼ぶ．また,図3で示す通りSMはk-Wayメモリインターリーブ構成であり,一つのコアがアドレスiにアクセスする際,(i mod k)-thのSMバンクにアクセスする．また,GMをメモリ,MPをコアと見なした全体モデルをUMMとした．DMMとUMMの主な違いは,コアとメモリバンクへのアドレスラインのコネクションの違いである．この点は,前述したDMM・UMMと変わらない．従って,DMM内では一度に全てのコアが別々のバンク内の任意の場所をアクセスできるが,UMM内では,コアがアクセスできる場所に制限がかかる．

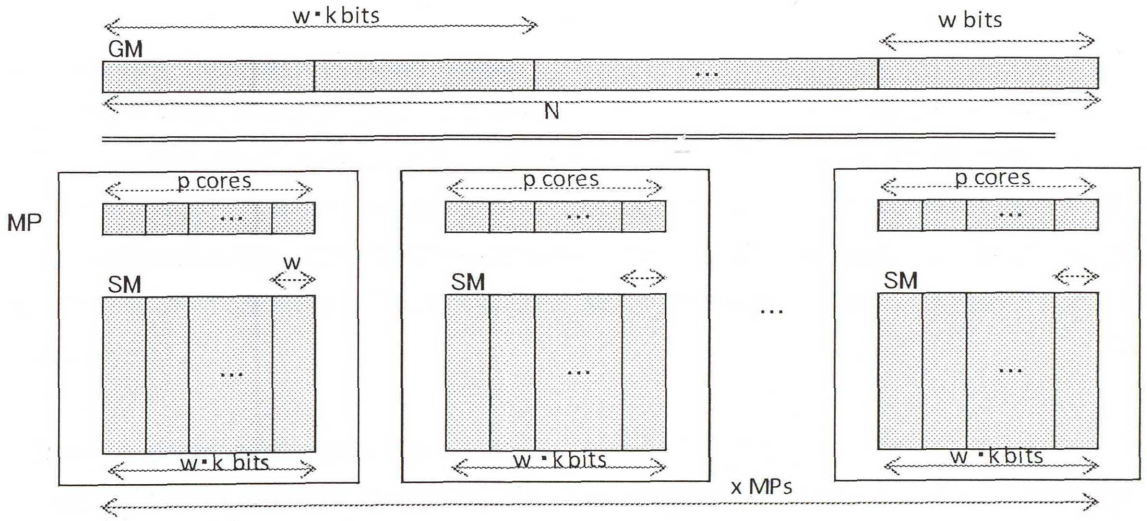

図3:統合モデル

また,MPをPRAMと見なして評価も行なった．PRAM上でのアルゴリズム評価は,常に二つのパラメータで与えられる.例えば,問題サイズnでp個のプロセッサの場合,n個の定数の和の計算はO(n/p+log p)と与えられる．MPをPRAMと見なすことができれば,DMMに比べてパラメータが減り,GPGPU上でのアルゴリズムの性能評価が易しくなるためである．

各MPはp個のコアと一つのSMを持つ．一つのMPのpコアが一度にGMからロードでき

る最大量をw bitsとしている．アクセスで気を付けるべき点は,必ずSMのメモリバンクコンフリクトを起こさず,GMのコアレッシングアクセスを毎回行う,という前提条件である．もし,SMのメモリコンフリクトが発生するケースとGMのコアレッシングアクセスができないケースをモデルに含めていくとすると,パラメータが増え,複雑な理論式になってしまうからである．よって,統合GPGPU上でメモリコンフリクトを発生せず,GMのコアレッシングアクセスをするようにアルゴリズムを工夫する必要がある．後述するCole's merge sort algorithmはSMのメモリコンフリクトを起こさず,かつGMへのコアレッシングアクセスを行う．各MPのコアは一度で最大w bitsをアクセスできるので,SMを全てGMのデータで満たすにはk回のアクセスが必要である．図4は，グローバルメモリの構成を示している．

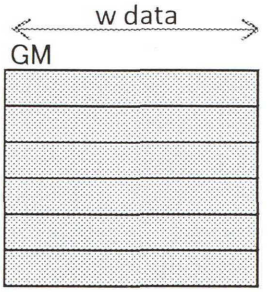

図4:グローバルメモリ(GM)の構成

4.2 モデル上でのアルゴリズム例,理論値

この統合GPGPUモデル上で実行するサンプルプログラムとして,N個のデータを並列にマージソートするCole's merge sort algorithmを採用した．このモデルは,共有メモリのメモリバンクコンフリクトの回避,グローバルメモリへのコアレスアクセスしなければいけない制限があるため,このプログラムを取り上げた．Cole's merge sort algorithmは複数のプロセッサの並列性を十分に生かすので,複数のコアを持つCUDAにも応用することが可能であった．

・Cole's merge sort algorithm

1986年,Coleは,O(n)個のPUsを用いてO(log n)で要素数nのソートをするためのCREW PRAMアルゴリズムを提唱した．このアルゴリズムは,木を用いたマージソートアルゴリズムである．よって,木の一つのレベルの各ノードでのすべてのマージングステップは並列処理することが可能である．一つのレベルでのマージングステップが終了すると,他のレベルでのマージングステップが開始できる．時間計算量を見てみると,レベルはlog nであり,総時間計算量O(log n)を得るためには各ステップをO(1)で処理しなければならないことになる．後述するCole's algorithmでは,各ステップをO(1)で処理することが可能になる．ステップごとに以前のステップでマージした際の部分情報を用いて,現在処理すべきステ

ップのマージの手助けをする．このアルゴリズムはとても効率的であるが，複雑なアルゴリズムである．

-Merge

ソート済みの二つの整数配列ＪとＫをＪ¦Ｋと定義する．以下から，もしａ＜ｘ≤ ｂならば，整数ｘはａとｂの間にあると表現する．

前提知識:

□rank

定義: 配列Ｊの要素ｘのランクとは，ｘよりも小さい(ｘと同値は含めない)Ｊの要素数である．

Rank(x,J) = card{ j∈J, j ＜ x }

Ex. A=<1,3,4,6,7> rank(5,A)=3, rank(7,A)=4

□cross-rank

定義: Ｂの中のＡのcross-rankとは，関数R[A, B] で表し，

配列Ａのｉ番目の要素が，Ｂの中のＡのｉ番目の要素のランクであることをこの関数で表すことができる．

Ex. A=<1,3,4,6,7> B=<3,5,8> R[A,B]=<0,0,1,2,2> R[B,A]=<1,3,5>

□good sampler(GS)

定義: 配列Ｌがｋ ・1で，任意のk+1の連続要素（{-∞} ・Ｌ ・{-∞}）の中で，いつも最大2k+1個のＪの要素を持つならばgood samplerと呼ばれる．

Ex. A=<1,2,3,4,5,6,7>とL=<2,4,6>があるとする．は，L=<-∞,2,4,6, ∞>となり，k=1の時，連続要素の-∞,2，つまり-∞よりも大きく，2よりも小さいＡの値をカウントする．カウント後，連続要素2と4,4と6,6と∞とカウントしていく．1≤i≤ |L| + 1

J(i) := { j∈J,Li-1 ＜ j ≤ Li+k }

Ex. J=<2,3,7,8,10,14,15,17,18,21] K=[1,4,5,9,11,12,13,16,19,20] L=[5,10,12,17]とする．

J(1)={2,3} J(2)={7,8,10}, J(3)=∅, J(4)={14,15,17}, J(5)={18,21}

K(1)={1,4} K(2)={6,9} K(3)={11,12} K(4)={13,16} K(5)={19,20}

GS を用いる事で,二つのソート済み配列のマージを高速にできる．例として,二つのソート済み配列ＪとＫをGSのＬでマージする．Ｌはl_0=-∞,$l_{|L|+1}$=+∞をセットする．

今,J = { 2, 3, 7, 8, 10, 14, 15, 17, 18, 21 },K = { 1, 4, 5, 9, 11, 12, 13, 16, 19, 20 } L = { 5, 10, 12, 17 }とする．ＬはＪとＫのＧＳである．このことは,k=1,2,…¦L¦で調べることができる．

例えば,k=1 の時,区間(-∞, 5),(5, 10),(10, 12),(12, 17),(17, +∞)のチェック,k=2 の時,区間(-∞, 10),(4, 12),(10, 17),(12, +∞)のチェックである．

k=1 の場合,
J(1)={2,3} J(2)={7,8,10}, J(3)=∅, J(4)={14,15,17}, J(5)={18,21}
K(1)={1,4} K(2)={6,9} K(3)={11,12} K(4)={13,16} K(5)={19,20}
を得て,ＬはＧＳであることが分かる．

以下に,マージを手助けするＬを用いたMerge_WITH_HELP(J, K, L)を定義する．

```
Merge_WITH_HELP(J, K, L){
        Step1: J and K are partitioned in ¦L¦+1 subsets.  …＊＊
        for(i=1;1≤i≤¦L¦+1;i++){
                J(i) = { j∈J,Li-1 ＜ j ≤ Li } and K(i) = {k∈K, Li-1 ＜ j ≤ Li }
                                }
        Step2: for(i=1;1≤i≤¦L¦+1;i++){ //as L is a GS of J and K, each subset has
at most three elements.
                //parallel do
        res(i) ← Merge( J(i),K(i) )
                                }
        Step3: J¦K ← res(1).res(2).res(3),…,res(¦L¦+1)        //partial results
are concatenated.

                Return J¦K
                }
```

ステップごとの詳細を以下に記す．
Step1:配列ＪとＫを¦L¦+1 サイズのサブ配列に分ける
　配列Ｊを分ける際には,¦J¦個のPUを用意する．それぞれのPj, j∈Jは rank(j,L)=r をリードし,J(r)にｊをインサートする．

このrank(j,L)=rをリードする時,R[J,L]を参照するだけでrが得られ,J(r)も分かる.

Ex.

J={2,3,7,8,10,14,15,17,18,21}, L={5,10,12,17},

R[J,L]={0,0,1,1,1,3,3,3,4,4,4}ならば,

P2 read rank(2,L) = 0 , inserts 2 in J(0)
P3 read rank(3,L) = 0 , inserts 3 in J(0)
P7 read rank(7,L) = 1 , inserts 7 in J(1)
P8 read rank(8,L) = 1 , inserts 8 in J(1)
P10 read rank(10,L) = 1 , inserts 10 in J(1)
P14 read rank(14,L) = 1 , inserts 14 in J(1)
:

同様に,Kもサブ配列に分ける.

R[J,L]とR[K.L]は既知であるので,このステップは¦J¦+¦K¦個のPUを用いてO(1)回で処理できる.

Step2:サブ配列を並行に併合する.

J(i)とK(i)は最大三つの要素を持ち,J(i)とK(i)を並列に併合することが可能である.

Step3:部分的なソート済みデータを統合する.

R[K,J]とR[L,K]は既知であるので,R[L,J¦K]を処理することができる.

加えて,すべて要素l (∈L)に対して

rank(l,J¦K) = rank(l,J) + rank(l,K)

が言える.

res iを担当するPUである各piは,J¦K内のli-1のrank rを計算し,res iの要素をJ¦Kとして格納する. Step3は,¦L¦+1個のPUが必要である.

このアルゴリズムのアイディアとして,パイプライン風にステップを踏むということである. まず,木の各ノードはソート済み配列val(t)をステップごとに保有していく. このval(t)は親ノードのマージソートのためのGSである. このGSが全ノードで得られるため,各ステップがO(1)回で処理することが可能になっている. よって,val(t)が生成可能であるのなら,そのノードでのマージソートが完了していなくとも先に親ノードに転送してしまう. これがパイプライン風に実行される理由の一つである.

また,プログラム内のReduce命令はステップ毎に振る舞いが変わるモードを四つ持って

いる．木のあるノードがレベルkで2^k個のソート済み要素を持っているとき,そのノードはコンプリート(complete)したと言う．もし,ステップtで,あるノードがコンプリートしたのなら,Reduce命令は以下のようにモードが変わっていく．また,val()内の要素数をxとする．

ステップt+1では,val(t+1)の(x/4)つの要素が親ノードに送られる．

ステップt+2では,val(t+2)の(x/2)つの要素が親ノードに送られる．

ステップt+3では,val(t+3)の全ての要素が親ノードに送られる．

ステップt+4では,そのノードは停止する．

ここで,0以上で同値が存在しないランダムな値が要素の配列Aに Cole’s merge sort algorithm を適用する．

Ex.A={43,81,2,37,5,86,77,26,92,34,76,56,1,35,90,55}

ここで,各MPのコア数が八つだとすると,配列Aを分割し,二つのMPでマージソートを行う．各MPでマージソートを行い,結果をGMへ書き戻す．この時,書き戻す先は読み込んだデータが存在していた場所である．図5は各MPからデータを書き戻した様子である．Sm1とsm2は各MPの共有メモリであり,各MPがGMにソート済みデータを書き込む．この時,ソート済みデータの前半部分をS1,後半部分をS’1とする．

この前半部分と後半部分に分ける際,R[L,sm1],R[L,sm2]を利用する．次に,S1とS2をsm1にリードし,MP内で Cole’s merge sort algorithm でソートする．S’1とS’2はsm2にリードし,同様にソートする．

□UMM(GM)上でのマージ

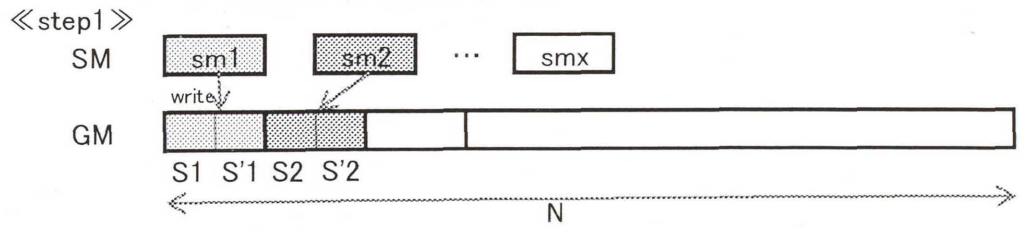

図5:UMM(GM)上でのマージステップ1

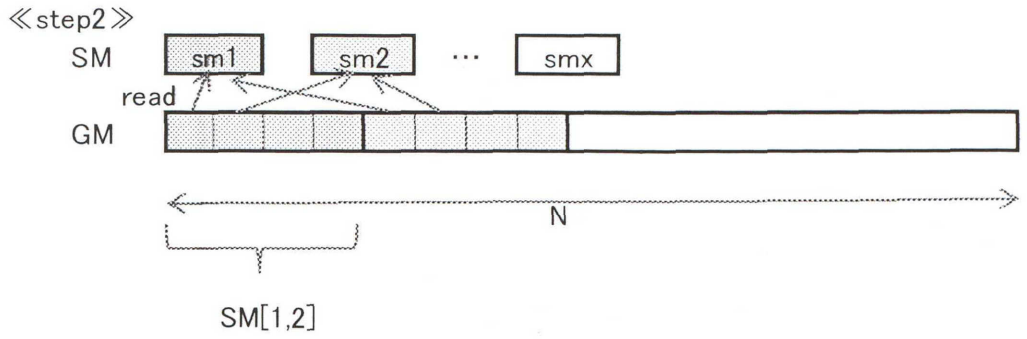

図 6:UMM(GM)上でのマージステップ 2

前提:

・同じ値を二つ以上存在しない二つの配列 X,Y(要素数は任意)を入力とする.

・GS と cross-rank R[L,X] R[L,Y],R[X,L],R[Y,L]は事前に分かっているものとする.

以下は cole のマージアルゴリズムである.

```
Cole_Merge()
        1.Receive X(t+1) from the left child and Y(t+1) from the right child
        2.Merge: val(t+1) ← Merge_with_help(X(t+1),Y(t+1),val(t))
        3.Reduce: Send Z(t+1) = Reduce(val(t+1)) to the father.
                { Reduce() keeps one value out of every four:
                Reduce({z1,z2,,,zn}) = {z4,z8,z12,,,} }
```

要素 N 個の cole’s merge sort algorithm の時間計算量は以下になる.

$(k + Lgm + sort(p, wk))$*N/kxw + (GM 上の merge sort 時間)

第5章　実験評価･考察

5．1　実験環境

本節では，実際にCUDAを用いて処理時間を測定した環境について述べていく．まず，実験にはIntel core i7-3770S CPU，GPUはNvidia Geforce GTX 580(Fermiアーキテクチャ)を用いた．提案モデル上のパラメータの名称とそれに対応するGPUの部分の名称，さらにその値を示す．レイテンシの単位はclockcycle時間．

表2:提案したGPGPU理論モデルのパラメータ

	各パラメータの説明
X	MPの数:16個
Wg	グローバルメモリのバンク:32Bbyte
Ws	シェアードメモリのバンク:使用せず
Wgk	シェアードメモリの全容量:約3.7Bbyte
Lgm	グローバルメモリレイテンシ:500
Lsm	シェアードメモリレイテンシ:4
N	要素数:可変
P	MP内のプロセッサ数:128

5．2　実験内容

要素数5万から100億個をGPGPUで並列に処理していった．一つのMPのSMに入る要素数を15000個に制限し，15000個をソートしたら，GMへ戻し，次の15000個をロードする.SMへのレイテンシLsmは4clockcycle時間,GMへのレイテンシLgmは500clockcycle時間とした.つまり，ソートを行う度に4clocksycle時間かかり，GMへソート済みデータを書き戻す度に500clocksycle時間かかることになる．

5．3　実験結果･考察

Nvidia Geforce GTX 580(Fermi アーキテクチャ)上で cole’s merge sort algorithm を実装した.対象となる要素数は 5 万から 100 億個である.図 7 には実行処理時間と理論モデル上の処理時間を載せた.理論モデルは各MP をDMMもしくはPRAMと仮定して算出している.

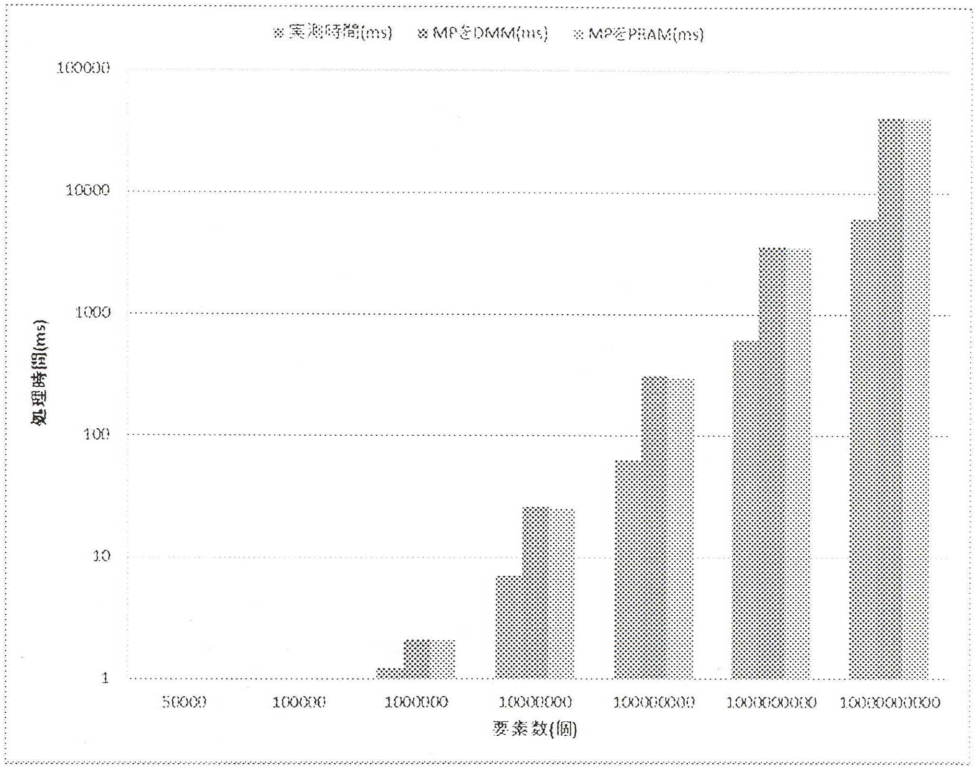

図 7:要素数 5 万から 100 億個の cole’s merge sort algorithm の GPGPU 処理時間と理論モデル上の処理時間

実際の処理時間が線形であるので,理論モデル上での時間計算量のオーダーが線形であることと一致する.このことから,提案した GPGPU 理論モデルが,cole’s merge sort algorithm のような連続したデータへのアクセスするアルゴリズムに対しては有効であることが分かる.逆に,連続しないデータに対して,つまりコアレスでないアクセスするアルゴリズムに対してはこの理論モデルはふさわしくない.

GPGPU 全体を UMM,MP を DMM として理論モデルを構成したが,グローバルメモリへのアクセスレイテンシとシェアードメモリへのアクセスレイテンシが 100 倍以上の差があるので,全体で見ると MP の時間計算量は小さい.よって,MP を PRAM と見なすモデルも考えられる.実際,行列積の時間計算量をこの理論モデルで導き出すと,PRAM でも十分,GPGPU 理論モデルとして役割を果たすことがわかっている.今後,様々な問題(アルゴリズム)に対してこの理論モデルの妥当性の検証･改善が課題となっていく.

謝辞

研究を進めるにあたり,ご指導を頂いた指導教員の和田幸一教授に感謝致します.また,日常の議論を通じて多くの知識や示唆を頂いた和田・武末研究室の皆様に感謝します.

〈参考文献〉

[1] Jason Sanders, Edward Kandrot, “CUDA BY EXAMPLE AnIntroduction to General-Purpose GPU Programming 汎用 GPU プログラミング入門(株式会社クイープ訳)”, 2011.

[2] K. Nakano, “Simple Memory Machine Models for GPUs”, IPDPS Workshops, pp.794-803, 2012.

[3] A. Koike,K. Sadakane, “GPU のための並列計算モデル”, IEICE Technical Report COMP2012-42(2012-10) pp.53-60, 2012.

[4] F.Thomson Leighton(1992), “Introduction to Parallel Algorithms And Architectures:Arrays・Trees・Hypercubes”, Morgan Kaufmann Publishers, Inc., pp.15-20.