2版

様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科研費 KAKENHI

科学研究費助成事業　　研究成果報告書

平成 ２７ 年　 ５ 月 ２０ 日現在

機関番号：１４１０１
研究種目：基盤研究(C)
研究期間：2012～2014
課題番号：２４５０００６０
研究課題名（和文）共有メモリモデルとスケジューリング最適化によるＧＰＧＰＵプログラミング技術

研究課題名（英文）GPGPU Programming Framework based on a Shared Memory Model and Scheduling Optimization

研究代表者
大野　和彦（OHNO, Kazuhiko）

三重大学・工学（系）研究科（研究院）・講師

研究者番号：２０３０３７０３

交付決定額（研究期間全体）：（直接経費）　　3,900,000円

研究成果の概要（和文）：グラフィック処理用のGPUを用いた高性能計算はコストパフォーマンスの高さから利用が増えているが、現状のプログラミング環境は生産性・再利用性に問題がある。そこで、現在使われている開発環境CUDAを改良したMESI-CUDAの研究を行った。MESI-CUDAではGPUの複雑なアーキテクチャを隠蔽してプログラミングを容易にすると同時に、コンパイル時に自動最適化を行うことでユーザの負担なく高性能を実現する。本研究課題では、前者の成果としてスレッドの論理マッピング記法の導入や動的データ構造のサポート、後者の成果として高速なシェアードメモリを利用しマッピングを最適化する手法を得た。

研究成果の概要（英文）：Although Graphics Processing Units (GPU) is regarded as a promising platform for high performance computing, the productivity and reusability of the current programming framework CUDA are not sufficient. Therefore, we are developing an improved framework named MESI-CUDA. MESI-CUDA provides easier framework hiding low-level architecture, while high performance is achieved by the automatic optimization. As the research results, we introduced logical thread mapping and supported dynamic data structures. To improve the execution performance, we also developed memory access optimization schemes such as utilizing the shared memories and logical-to-physical thread mapping.

研究分野：並列プログラミング言語

キーワード：高性能計算　GPGPU　CUDA　並列プログラミング言語　言語処理系　プログラミングモデル　自動最適化

１．研究開始当初の背景

近年、汎用プロセッサのマルチコア化と並んでグラフィックプロセッシングユニット(GPU)を高性能計算資源として用いるGPGPU が盛んになり、GPU 搭載型スーパーコンピュータから小規模な計算サーバまで、幅広く利用されている。現在の GPU は数千に及ぶコアを内蔵しており、2～16 コア程度の汎用プロセッサと比較して潜在的な性能が非常に高い。しかし、現状で GPGPU のプログラミングはハードウェアに密着した記述が必要であり、高性能を達成するには手動で最適化を施す必要がある。さらに、プログラムの実行性能が GPU の緒元に大きく影響されるため、異なる GPU モデル上で動作させる場合は再度手動最適化を要する。

この原因として以下の点が挙げられる。

(1) GPU は高速な画像処理を目的に設計されており、汎用プロセッサと比較すると様々な処理に対する適性の差が激しい。

(2) 一定数のコア毎にストリーミングマルチプロセッサ(SM)と呼ばれるグループを構成しており、SM 内外やチップ内外に異なる特性のメモリ･キャッシュを持つ、複雑なアーキテクチャになっている。

(3) ハードウェアの特性上 OS のようなミドルウェアを動作させることができず、アプリケーション層に対してハードウェアを抽象化・仮想化できない。

２．研究の目的

現在の標準的なGPGPUプログラミング環境である CUDA に対し、より簡潔で移植性の高いコーディングが可能であり、同時に高い実行性能を達成できる環境を実現する。

本研究課題開始前より、我々は CUDA をベースとしたMESI-CUDAを提案している。MESI-CUDA は仮想共有メモリ型のプログラミングモデルにより GPU の複雑なアーキテクチャを隠蔽し、静的解析を用いた自動最適化によりユーザの負担なく高性能を達成するアプローチをとる。本研究課題では、この MESI-CUDA に適用可能な様々な自動最適化手法を開発する。

３．研究の方法

CUDA を用いた GPGPU プログラミングでは、ハードウェア特性を踏まえた様々な最適化テクニックが知られており、個々のアプリケーション上に手動で適用されて大きな性能改善を達成している。したがって本研究では、この種のテクニックを自動的に適用する手法を開発するため、以下の研究を行った。

(1) MESI-CUDA プログラムの静的解析手法

(2) 最適化戦略決定手法

(3) 最適化コード生成手法

既知の最適化テクニックのほとんどは、個々のプログラムの挙動を踏まえて適用の可否やパラメータの決定を行う必要がある。たとえばメモリアクセス最適化のためにはプログラム中における配列アクセスの回数や範囲といった情報が必要になる。条件分岐の実行効率が悪く OS の支援を受けられない GPU の特性から、MESI-CUDA では静的解析によりこうした情報を抽出し、コンパイル時に最適化戦略を決定したうえで、最適化されたコードを静的に生成する方法をとる。これにより、実行時に動的な情報収集や判定を行うオーバヘッドが生じない。一方で静的に解析できるプログラムの挙動には限界があるが、比較的定型的な処理に用いられる GPGPU の性質から、実用上は十分である。

４．研究成果

(1) 自動的な明示キャッシュ手法

GPU は用途別に様々なメモリを持つが、とくにチップに内蔵するシェアードメモリはアクセス遅延が小さい。このため、アクセス頻度の高いデータをシェアードメモリ上に配置する手動最適化手法が知られている(図1)。汎用プロセッサではこのような機構はハードウェアキャッシュとして実現されているが、GPU ではユーザプログラムレベルで直接制御する必要がある。しかしシェアードメモリは 64KB 程度と小容量のものが SM 毎に用意されているため、これらを効率よく利用するプログラムを作成するのは難易度が高い。

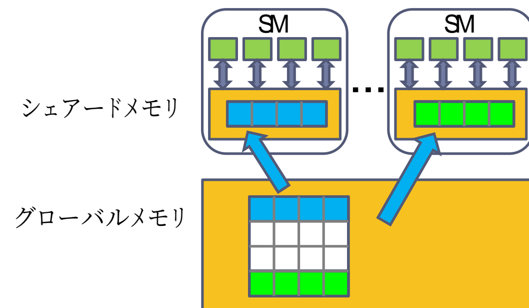

図 1 シェアードメモリへのキャッシュ

そこで、シェアードメモリをキャッシュとして自動的に利用する手法を実現した。ハードウェア･OS の支援がないため、本キャッシュ機構はユーザプログラム内に明示的なキャッシングを行うコードを挿入する。このコードの生成はコンパイル時に自動的に行われるため、通常のハードウェアレベルのキャッシュと同様、ユーザの負担は生じない。また、キャッシュ対象はコンパイル時に決定されているため、アクセス状況を動的に追跡するオーバヘッドも生じない。

① 静的解析手法

スカラー変数は通常レジスタに配置されるから、本手法では配列変数内の必要な領域をシェアードメモリにコピーする。そのためには、同じ SM 上で実行されるスレッドの集まりであるブロックに対して、ブロック内でアクセスされる範囲を得る必要がある。また、限られたシェアードメモリを活用するため

にはより再利用率の高い配列を優先するべきであるから、要素当たりの平均アクセス回数も得る必要がある。

本手法では、配列の添え字式とループ範囲に注目し、これらの情報を静的に解析する。この種の静的解析は以前から C や FORTRAN などを対象に研究されているが、一般的には条件分岐や不定回ループ、不規則な要素アクセスを考慮する必要があるため、高精度の解析を短時間で行うことが難しい。本手法においては、不定形の制御構造は GPU の性能特性から避ける傾向があること、静的なコード生成を目的とするため規則性のない解析情報は役に立たないことから、添え字式を線形式に限るなど実用上問題ない制限を加えることで、解析の計算量を大幅に削減している。

② 最適化戦略決定手法

静的解析で得られたアクセス範囲と平均アクセス回数を用いて、キャッシング対象となる配列を決定する。具体的な戦略としては、アクセス範囲の和がシェアードメモリの大きさ以下に収まる配列の組み合わせのうち、平均アクセス回数の和が最大になるものを選択する。

③ 最適化コード生成手法

キャッシング対象に選択された配列について、低速なグローバルメモリとシェアードメモリの間でデータをコピーするコードを生成する。また、これらの配列にアクセスするコードを、シェアードメモリ上のコピーへアクセスするコードに置き換える。

本手法を実装し実機上で評価を行った結果、ほとんどの場合に 1.3～3.1 倍程度の速度向上を得た(図 2)。一方、NAS Parallel

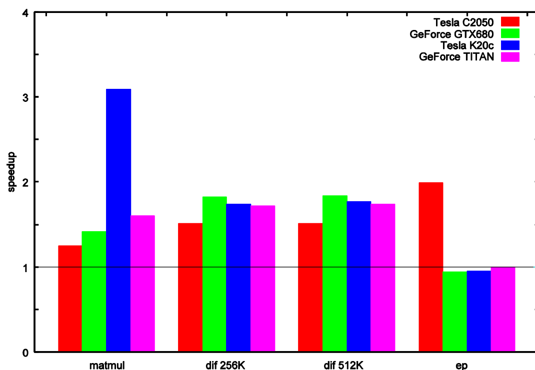

図 2 明示キャッシュによる速度向上率

Benchmark の EP では多くの GPU モデルでわずかに性能低下が見られた。このプログラムでは最もアクセス頻度の高い配列がシェアードメモリの大きさを超えており手法の効果が低い。一方で、シェアードメモリを使用することにより並行実行されるスレッド数が低下する現象が起きていると思われる。この結果が示すように、本手法は実行性能の改善に貢献するものの、副作用の方が大きいケースでは適用しないという戦略が必要である。

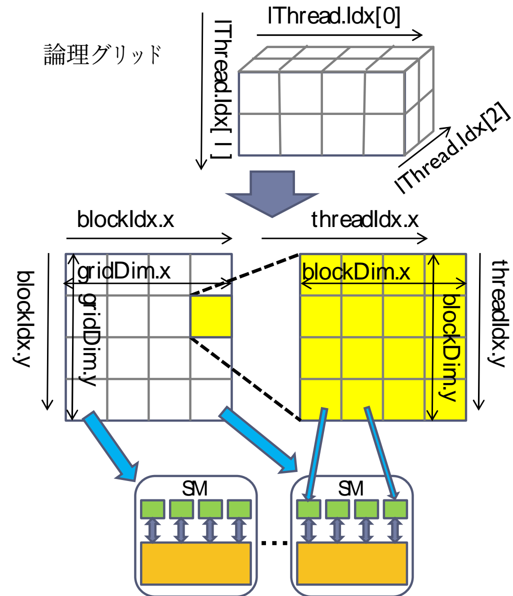

図 3 スレッドの論理マッピングと物理マッピング

(2) スレッドマッピングの自動最適化手法

CUDA では GPU スレッドとデータのマッピングをユーザが直接指定する。また、スレッドと物理コアの間のマッピングは自動的に行われるものの、ユーザが指定したブロック内のスレッド群が同一 SM 上で実行されるなど、ユーザのマッピングが物理マッピングに大きく影響する。このため、高性能を達成するためにはハードウェア特性を意識したマッピングを記述する必要がある。GPU のモデルにより SM 数などが異なるため、最適なマッピングも変化する。

そこで、論理的なスレッドマッピング記法を導入し、GPU のモデルに依存せずに問題の性質だけを考慮したマッピングを行えるようにした(図 3)。また、この論理マッピングを最適化な物理マッピングに変換する自動最適化手法を実現した。本手法もコンパイル時の自動コード変換によって実現するため、ユーザの負担や実行時のオーバヘッドが生じない。

① 言語拡張

CUDA で GPU スレッドを生成するには、スレッド起動時にブロック数とブロック内スレッド数を指定する。これらの値は SM 上のスレッド数と各 SM に割り当て可能なスレッド群の数を決定するため、物理資源を活用するには物理構造を意識した適切な値を指定しなければならない。また、ビルトイン変数により、スレッドやブロックのインデックスを取得できる。この変数を配列アクセス時の添え字式中でこれらの変数を用いることで、各スレッドが配列中のどの要素をアクセスするかという、スレッド・データ間のマッピングを行う。SM 上ではインデックスの連続する 32 スレッド毎に SIMD 型並列実行を行い、これらのスレッドが同一キャッシュラインにアクセスする際にはトランザクションを一つにまとめるコアレス化を行う。このため、連続インデックスを持つスレッド群を配列

のどの次元に対応させるかによって、メモリアクセス遅延が大きく変化する。また、この対応関係により SM 内でアクセスされる配列の範囲も変化しうるから、(1)で述べた最適化手法の効果も左右する。

これに対し本手法では、スレッド起動時に論理マッピングを指定する記法を導入した。本記法では任意の次元数・大きさの論理グリッドの形でスレッド数を指定する。また、スレッドはビルドイン変数により、各次元方向のインデックス値を得ることができる。このためユーザはハードウェアに依存せずに扱うデータ構造に合わせたスレッドグリッドを生成し、スレッド・データ間マッピングを自然に記述できる。

② 静的解析手法

本手法では、ユーザが指定した k 次元の論理マッピングを、ブロック・スレッドという CUDA の 2 次元マッピングに変換する。この際に、メモリトランザクションのコアレス化率が高く、ブロック内の配列アクセス範囲が小さくなるように両マッピングの次元を対応づけることで、実行性能を向上させる。

本手法では、配列の添え字式に注目し、論理インデックスを表すビルトイン変数の値とアクセスする配列要素との関係を抽出し、同一あるいは近接要素をアクセスするスレッドのグループを発見する。この問題の一般解を得るには非常に大きな計算量を要するが、本手法では任意のスレッドをグループ化するかわりに論理 k 次元をどの順で物理 2 次元に畳み込むかという問題に簡略化し、高速な解析を可能にしている。具体的には k 個の各論理次元方向について隣接スレッドがアクセスする要素間の距離を求め、その次元の畳み込み優先度の評価値としている。

③ 最適化戦略決定手法

静的解析結果を利用し、最適な物理マッピングを決定する。

コアレス化については、解析で得られた優先度がコアレス化率を反映しているため、優先度の高い論理次元が内側になるように、物理次元に畳み込むことで最適解が得られる。

また、このマッピング時にはブロック内のスレッド数を決定する必要がある。同一 SM 上で並行実行されるブロックは物理資源を共有するから、(1)で述べたようにシェアードメモリの使用は並行ブロック数を減少させ、実行性能を悪化させる可能性がある。現在の手法では、シェアードメモリの使用量が CUDA の推奨値である 16KB 以下に収まるようにブロック内スレッド数を決定している。

④ 最適化コード生成手法

決定した物理マッピングになるように、スレッド生成時のパラメータを生成する。また、配列の添え字式を、CUDA のビルトイン変数を用いた等価な式に置換する。

本手法を実装し実機上で評価を行った結果、最悪マッピングと比較すると、ほとんどの場合に数倍から数十倍の速度向上を得た(図 4)。新しい GPU モデルほどキャッシュなどの追加機能によりマッピングによる差が減少する傾向にあり、また初心者が CUDA プログラムを記述する際に最悪なマッピングを選択するとは限らない。しかし、最新の GPU モデルにおいてもマッピングにより大きな性能差が出ることが多いことから、自動最適化によりユーザの負担なく安定した高性能が得られることは、大きな利点である。

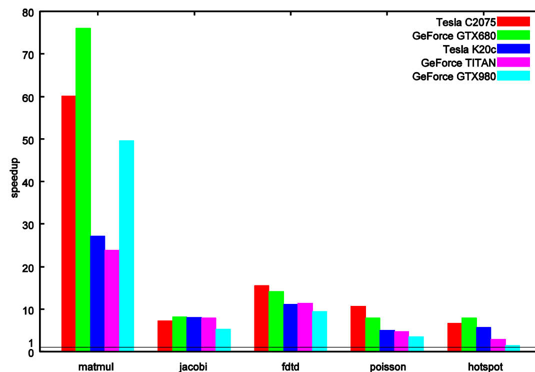

図 4　最悪マッピングに対する速度向上率

(3) マルチ GPU

GPGPU ボードは 1 台の PC に複数装着できるため、プログラムから複数の GPU を同時に使用することで、さらに高性能を実現できる。CUDA ではこのようなプログラミングが可能であるものの、明示的に GPU デバイスを切り替えながらデータ転送やスレッド起動などの操作を行う必要がある。また、負荷分散についても処理系のサポートがないため、とくに性能の異なる GPU を装着している場合に、適切な割合で負荷を割り当てるのはすべてユーザの責任となる。

そこで、MESI-CUDA の仮想共有メモリモデルを拡張し、利用可能な GPU すべてに対して自動的に並列スレッドが分配される機構を提案した。これにより、MESI-CUDA プログラムは修正なしにマルチ GPU 環境で高速動作させることができる。また、各デバイスへのスレッド配分やデータ転送は処理系が行うため、自動最適化を施すことができる。この最適化性能を左右するスケジューリング戦略については未だ研究途中であるが、予備評価では、ある程度の静的負荷分散を行ったうえで動的負荷分散により調整を行うハイブリッド方式が、どちらか一方のみの方式より安定して高性能を得られるという結果を得ている。

(4) 動的データ構造を扱うための機能拡張

GPU は定型的な処理に適したアーキテクチャであるため、MESI-CUDA は配列を主体とする静的データ構造を想定した言語設計や自動最適化手法を採用している。しかし、実プログラムでは定型的な処理であっても部分的に動的データ構造が必要になることも多

い。そこで、仮想共有メモリ上で動的データ構造を扱うための言語拡張を設計・実装した。
具体的には、可変長配列およびポインタのサポートを実現した。MESI-CUDA が提供する仮想共有変数は、複数メモリ上の領域確保とデータ転送を行う CUDA コードに変換する形で実装されており、可変長配列は容易に実現できた。ポインタについては仮想共有変数間のリンク構造に限定し、転送時にアドレスを変換するコードを自動生成することで、C 言語のネイティブポインタを直接使用できるような機構を実現した。

(5) 成果の位置づけと今後の展望

2013年に発表されたCUDA6はMESI-CUDAと同様の共有メモリ型プログラミングをサポートしている。このことは、GPGPU プログラミングの容易化に関して本研究のアプローチが妥当であったことを裏付けていると考える。一方で、CUDAでは現在に至るまで処理系による自動最適化は採用しておらず、高性能が必要な場合はユーザが手動最適化することを想定している。したがってユーザの負担を軽減しつつ高性能なGPGPUプログラムを開発するには、本研究課題のような自動最適化手法が依然として必要である。

本研究課題においていくつかの有用な自動最適化手法を実現できたが、まだ多くの課題が残されている。とくにマルチ GPU 環境については、最適スケジューリング手法の研究が必要である。

また、本研究課題の遂行にあたっては手法の開発と評価を優先したため、当初の目的の一つであったMESI-CUDA処理系の開発については試験的な実装にとどまった。今後、実用レベルの処理系を実装し、提供していきたい。

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計　２件）

① Kazuhiko Ohno, Tomoharu Kamiya, Takanori Maruyama, Masaki Matsumoto, Automatic Optimization of Thread Mapping for a GPGPU Programming Framework, International Journal of Networking and Computing, 査読有, Vol.5, No.2, 採録決定, 2015

② Kazuhiko Ohno, Dai Michiura, Masaki Matsumoto, Takahiro Sasaki, Toshio Kondo, A GPGPU Programming Framework based on a Shared-Memory Model, Parallel and Distributed Computing and Networks, 査読有, Vol. 3, pp.1-14, 2013.

〔学会発表〕（計10件）

① 神谷 智晴，丸山 剛寛，大野 和彦，GPGPU 処理系におけるカーネル関数内のコード変換によるスレッドマッピング機構の改良，PPL2015，2015年3月2日，道後プリンスホテル(松山市)

② 丸山 剛寛，神谷 智晴，大野 和彦，GPGPU フレームワーク MESI-CUDA における自動最適化のための配列インデックスの静的解析手法，PPL2015，2015 年 3 月 2 日，道後プリンスホテル(松山市)

③ Kazuhiko Ohno, Tomoharu Kamiya, Takanori Maruyama, Masaki Matsumoto, Automatic Optimization of Thread Mapping for a GPGPU Programming Framework, CANDAR2014, 2014 年 12 月 10 日，グランシップ(静岡市)

④ 丸山 剛寛，田中 宏明，水谷 洋輔，神谷 智晴，大野 和彦，GPGPU フレームワーク MESI-CUDA におけるマルチ GPU へのスレッドマッピング機構，SWoPP 新潟 2014，2014年7月30日，朱鷺メッセ(新潟市)

⑤ Tomoharu Kamiya, Takanori Maruyama, Kazuhiko Ohno, Compiler-Level Explicit Cache for a GPGPU Programming Framework, PDPTA'14, 2014 年 7 月 24 日，Las Vegas(USA)

⑥ 神谷 智晴，丸山 剛寛，大野 和彦，GPGPU 処理系の自動最適化におけるシェアードメモリへのデータ転送方式の改良，第143回HPC研究発表会，2014年3月3日，和倉温泉「あえの風」(七尾市)

⑦ 神谷 智晴，丸山 剛寛，松本 真樹，大野 和彦，GPGPU のシェアードメモリを利用する自動最適化機構，SWoPP 北九州 2013，2013年8月2日，北九州国際会議場(北九州市)

⑧ 丸山 剛寛，神谷 智晴，松本 真樹，大野 和彦，シェアードメモリを利用した GPGPU 処理系の自動最適化機構，先進的計算基盤システムシンポジウム SACSIS2013，2013年5月23日，仙台国際センター(仙台市)

⑨ Kazuhiko Ohno, Masaki Matsumoto, Tomoharu Kamiya, Takanori Maruyama, Supporting Dynamic Data Structures in a Shared-memory Based GPGPU Programming Framework, PDCS2012, 2012 年11月14日，Las Vegas(USA)

⑩ 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，先進的計算基盤システムシンポジウム SACSIS2012，2012年5月17日，神戸国際会議場(神戸市)

６．研究組織

(1)研究代表者

大野 和彦（OHNO，Kazuhiko）
三重大学・大学院工学研究科・講師
研究者番号：２０３０３７０３