様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２８ 年　 ６ 月　 ６ 日現在

機関番号：１２６１２
研究種目：基盤研究(C)（一般）
研究期間：2013～2015
課題番号：２５３３０１４３

研究課題名（和文）GPU向けOpenMP処理系の開発

研究課題名（英文）Research on OpenMP for GPU computing

研究代表者
本多　弘樹（HONDA, HIROKI）

電気通信大学・その他の研究科・教授

研究者番号：２０１９９５７４

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）：GPUコンピューティングにおいて，高い性能を達成するプログラムを作成する際の難しさを低減させるには，各種の最適化手法がユーザプログラムに自動的に適用されるプログラミング環境の構築が求められる．そこで本研究では，GPUで簡便に高性能を得ることができるプログラミング環境の実現に必要な各種最適化手法を考案するとともに，その手法を組み込んだトランスレーターとAPIの開発を行った．

研究成果の概要（英文）：On a GPU computing, to ease the difficulty of creating programs to achieve high performance, the programming environment where optimization schemes can be automatically applied on user programs. In this research, while proposing the optimization techniques which realize GPU programming environment for easily attaining high performance, the APIs and the translators based on the proposed schemes were developed.

研究分野：総合領域

キーワード：ハイパフォーマンスコンピューティング　並列処理　GPU

１．研究開始当初の背景

科学技術計算のためのハイパフォーマンスコンピューティングシステム（HPC システム）では，計算アクセラレータとしての GPU を複数備えたハイブリッドアーキテクチャのノードを複数結合することによりハードウェアを構成し，GPU コンピューティングを可能とするものが多い．また，単一ノードで構成される小型 HPC サーバでも複数 GPU を装備し，GPU コンピューティングを可能とするものもある．これは，GPU により電力及び電力量あたりの計算性能を向上させることができるなどの理由による．HPC システムの環境負荷低減が求められる中で，低消費電力な HPC システムの構築をするには今後も GPU コンピューティングが不可欠と考えられる．

一方，GPU 利用にあたっては， CUDA などの GPGPU 向けプログラミング環境を用いたプログラミングが必要となっていたが，これはこれらに習熟していないユーザにとっては大きな障壁となる．

このような中で，応募者らは，従来の並列プログラミング経験者が CUDA などを習得せずとも GPU コンピューティングを利用できるプログラムの開発を可能とし，また，従来の CPU 向けに記述された並列プログラムを GPU で実行できるようにすることを目指して，CPU 向けの並列プログラミングモデルである OpenMP で記述されたプログラムを GPU で実行されるプログラムへ変換することを世界に先駆けて着眼し，その処理系の開発に取り組んできている[1].

２．研究の目的

本研究課題では，これまでの上記の研究成果を発展させ，ユーザが GPU コンピューティングを簡便に利用するために必要となる処理系の基本技術の開発を試みるものである．特に次の３点に重点をおいて，手法の考案と実装を行う．

・タスクの最適分割と最適割当
・CPU-GPU 間データ転送最適化，
・カーネル関数内最適化

３．研究の方法

研究は３年間かけて行うこととした．まず，本研究で対象とする最適化項目の検討を行い，対象とすべき項目の選定を行う．次に，必要な各種最適化手法の考案と実装を行い手法の評価を行う．

具体的には次のとおりである．

(1) 研究計画策定の現段階では，前述の３項目を最適化項目として次の項目を候補としているが，GPU プログラミングにおけるハンドチューニング技法をサーベイの上，最適化項目として含むべきものが有るか否かを検討する．
(2) (1)で検討した各項目に対し，実プログラムにおける適用頻度や見込まれる最適化効果などをもとに優先度を策定し，どの項目を本研究期間での対象とするのかを検討する．
(3) GPU アーキテクチャ，GPU プログラミング/チューニング環境，OpenCL[2]や OpenACC[3] 等のアクセラレータ向け並列方式は早いペースで進歩しているので，他の研究・開発状況などにより本研究でも考慮すべき事項については柔軟に対処することとする．
(4) 有効性検証に用いるためのテストベッドシステムを構築するとともに，必要となるソフトウェアの導入などを行い有効性検証環境の整備を行う．
(5) 本研究の対象とした個々の最適化項目についての手法を考案する．すなわち，各最適化におけるパラメータ（データ転送量，ループ回転数，ループボディ計算量など）の同定，テストベッドシステム上での実行時間の測定による最適パラメータ値の取得，プログラム解析によるパラメータ設定自動化手法の開発などを行う．
(6) 各機能の実装については優先順位を決め，進捗が思わしくない場合には，重要な機能の実装に絞るなど対処する．
(7) 構築したテストベッド上で開発した手法の動作検証と有効性の評価を行う．
(8) 研究成果を学会等で発表するとともに研究成果をまとめる．
(9) 研究体制は，研究代表者(本多弘樹)と１名の研究協力者(各年に大学院学生１名程度)で構成する．

４．研究成果

本研究の主要な成果は，CUDA Zero Copy 方式によるデータ転送の最適化手法の開発，OpenCL におけるタスク分割・割当とカーネル関数内の最適化手法の開発，OpenACC におけるデータ転送最適化手法の開発である．
それぞれの研究成果の概要は次のとおりである．

(1)CUDA Zero Copy 方式によるデータ転送最適化手法

本研究における各課題項目の初期検討の結果，GPU コンピューティングでは GPU におけるカーネル関数自体の実行時間が短縮されても CPU と GPU 間のデータ転送に時間がかかってしまうと総合的な性能が低下してしまう問題をまずは解決すべき重要な課題であるとの認識に至った．

そこで，CUD に実装されている Zero Copy 方式によるデータ転送方式に着目し，通常の cudaMemcpy 関数を用いた Memory Copy 方式によるデータ転送と比べた場合の有効性の検証，実際のカーネル関数実行に際しての Zero Copy 方式の採用方法について考察を進めた．

その結果，カーネル関数で使用される変数のアクセス回数に応じて Memory Copy 方式と Zero Copy 方式を選択することでデータ転送

時間の削減を可能とする方式を提案し，その効果を簡便に利用できるようにするためのAPIのあり方を明らかにした．

開発した手法の特徴は，プログラムの特性に応じて Zero Copy 方式が有利な場合と Memory Copy 方式が有利な場合を判断・選択するという点にある．

Zero Copy 方式では，CPU の Mapped Memory を利用して GPU のメモリ空間を CPU のメモリ空間にマッピングすることにより，GPU が直接 CPU メモリの変数にアクセスすることが可能となり，Memory Copy による CPU と GPU 間での明示的なデータコピーが不要となる．これにより，Memory Copy に係るオーバーヘッドを削減しひいては CPU と GPU 間でのデータ転送時間を短縮することが期待できる．

一方，同じ変数に何度もアクセスするような場合には，Memory Copy によってデータをいったん GPU メモリに転送し，その GPU メモリ上の変数にアクセスした方が，アクセスコストの総量を減らすことができる．

そこで本研究では，変数へのアクセス回数に基づいて，Memory Copy 方式と Zero Copy 方式を適切に選択する手法を開発した．
具体的には，それぞれの手法による転送時間を検証し，アクセス回数が 2 回以上の変数については Memory Copy により CPU メモリから GPU メモリへデータを転送した上で GPU メモリ上の変数にアクセスすることとし，アクセス回数が 1 回の変数については Zero Copy により CPU メモリ上の変数に直接アクセスすることとした．

さらに，従来の CUDA プログラミングで用いられている Memory Copy を用いたプログラムを変数へのアクセス回数に応じて Zero Copy を用いたプログラムに変更する際のプログラマの負担を軽減することを目的として，データ転送のための API を開発した．本 API は，メモリ領域確保・解放，カーネル実行，必要なデータ転送を自動的に行う機能を有する

提案手法の評価は，Rodinia ベンチマークの Nearest Neighbor を用いて行った．評価に用いた環境の CPU は Intel Xeon X5660(4Core)2.8GHz，GPU は NVIDIA Tesla S2050 1.15GHz，CUDA のバージョンは 5.0 である．その結果，オリジナルの転送方式の実行時間に比べて，提案手法を用いた実行時間は 12%短縮することができることを確認した (表 1)．

表 1　Nearest Neighbor 実行時間

	オリジナル	提案方式
実行時間[us]	96.26	84.96

この成果は，CPU/GPU 間のデータ転送を削減することを目的に CUDA の Zero Copy を用いる際に重要な技術を明らかにした点で意義がある．

(2) OpenCL におけるタスク分割・割当とカーネル関数内の最適化手法の開発

現状の GPU コンピューティングシステムでは異なる種類の GPU が搭載された複数のコンピュータノードがネットワークを介して接続されている形態が多くなってきているとの認識のもと，タスクの最適分割と最適割当に注力することとし，また，対象を CUDA のみではなく昨今注目されている OpenCL も含めて研究を実施した．

具体的には，性能の異なる GPU を搭載したノードから構成されるヘテロジニアスなシステムにおいて，それぞれの GPU に割り当てるスレッド数や SM に割り当てるスレッド数を決定する手法について考察を進めた．

その結果，カーネルプログラムの計算量と GPU の PE から，OpenCL におけるグローバルワークサイズとローカルワークサイズを算出するモデルを明らかにした．

GPU の種類が増加した際には，各 GPU の演算コア数やその性能，メモリ性能の差異を考慮して，それぞれの GPU に割り当てる処理の分割方法を決めなければならない．

そこで本研究では，性能の異なる GPU を搭載した複数ノードから構成されるヘテロジニアスな GPU システムを対象とし，複数 GPU のそれぞれに割り当てるスレッド数や SM あたりに割り当てるスレッド数といったワークサイズを適切な値に決定する手法を考案し，これを OpenCL 上で実装することに取り組んだ．

開発した手法では，グローバルワークサイズ，ローカルワークサイズ，ワークグループ内でのワークアイテムのグループ構成を最適化する．最適化の方針としては，まず各 GPU での適切なグローバルワークサイズを決定した後，ローカルワークサイズを決定することとした．手法の概要は次のとおりである．

グローバルワークサイズ決定手法：

入力変数としては，GPU の PE 数とカーネル関数の計算量を用いる．カーネル関数の計算量はループボディ中の四則演算命令と比較命令をループ繰り返し数の積とした．
各種ベンチマークプログラムを対象にグローバルワークサイズを変化させて実験を行った結果から，二つの GPU へ割り当てるグローバルワークサイズ(GWS)を次式のとおりとするモデル化を行った．

GPU_s の GWS =
　　RE_ratio_s * k *　総ﾜｰｸｱｲﾃﾑ数
GPU_l の GWS =
　　総ﾜｰｸｱｲﾃﾑ数 - GPU_s の GWS

(GPU_s:PE の少ない GPU，GPU_l:PE の多い GPU，PE_resio_:GPU_s の PE 数割合，k=0.25(高計算量)，k=0.75(中計算量)，k=1.25(低計算量))

GPU 数が 3 以上の場合には PE 数の小さい順に上述の方法で決定していく．

ローカルワークサイズとワークグループ

構成決定手法：

各種ベンチマークプログラムを対象にローカルワークサイズとワークグループ構成を変化させて実験を行った結果から，ローカルワークサイズとワークグループ構成は次のとおりとすることとした．

すなわち，ワークグループがワークアイテムを 2 次元方向にマッピングすることが可能な場合は，ローカルワークサイズを 1024 未満で 16 の倍異数になる最大値とする．

ワークグループ構成に際しては，メモリアクセスレイテンシを削減するよう，計算に必要なデータを共有する複数ワークアイテムを x 方向もしくは y 方向でグループ化することとした．

すなわち，ワークグループがワークアイテムを 1 次元方向にのみマッピング可能な場合は，CU あたりの PE 数に等しいか，この数を超えもっとも小さい値のローカルワークサイズを選択する．

ワークグループがワークアイテムを 2 次元方向にマッピング可能な場合には，x 方向にマッピングしたワークアイテム数の約数 CD_x と y 方向にマッピングしたワークアイテム数のの約数 CD_y の組み合わせの中から，次の方針でローカルワークサイズとワークグループ構成を決定する．

1. 16 の倍数で 1024 未満のローカルワークサイズの組み合わせを抽出．
2. 上記組み合わせの中からさらに，x 方向優先構成の場合には最大の CD_x を，y 方向優先構成の場合には最大の CD_y を要素に持つ組み合わせを抽出．
3. 上記組み合わせの中から，ローカルワークサイズが最大となる組み合わせを採用する．

提案手法の有効性を検証するため，前述の手法を OpenCL に実装し，GPU として Quadro 2000D と GeForce 8800GTS をそれぞれ搭載したシステム上で，提案手法を用いた場合と用いない場合でプログラムの実行時間の比較評価を行った．対象としたプログラムは，行列積カーネル，線形補完法カーネル，実数ソートカーネル，台形公式カーネルである．

グローバル・ワークサイズの決定手法の評価には，グローバル・ワークサイズを GPU の PE 数の比で分割した場合と提案手法で分割した場合とを比較する．ローカル・ワークサイズの評価には，この値を指定しない場合，および提案手法で求めた値にした場合とを比較する．

その結果，線形補間法カーネル以外のカーネルプログラムでは，提案手法を用いることで実行時間を平均 47%短縮できることを確認した．また，全カーネルプログラムでは実行時間を平均 11%短縮できることを確認した．線形補間カーネルでは実行時間が約 85%増加してしまったが，各 GPU にグローバルワークサイズを分割・分配する際に，適切ではない分割方向を選択してしまったためと考えられ，今後の課題となる．

この成果は，複数 GPU へのタスクの分割・割当，カーネル関数内での割り当ての最適化を行う際に，GPU の PE 数という比較的単純な指標でも効果が得られることを示した点で意義がある．

本研究成果の一部は，情報処理学会のハイパフォーマンスコンピューティング研究会で発表を行った(学会発表の(1))．

(3) OpenACC におけるデータ転送最適化手法

前述のとおり，データ転送の最適化として Zero Copy を用いた手法を開発した．

さらなる最適化をめざし，現状の GPU コンピューティングシステムでの CPU と GPU 間のデータ転送最適化はタスク分割最適化との協調が必要であるとの認識のもと，タスク分割とデータ転送の協調最適化に着眼し，また，対象を GPU プログラミングの枠組みとして利用が進んでいる OpenACC をターゲットとして研究を実施した．

具体的には，単純なデータ転送のみを指定した OpenACC プログラムを解析し，タスク分割・データ分割を行うことによってデータ転送のパイプライン化を可能とすることによって，データ転送の自動最適化を行う手法について考察を進めた．さらに，その手法を実装した OpenACC to OpenACC のコードトランスレータを開発し，その結果，同手法の有効性を明らかにした．

パイプライン化には，並行実行を実現する GPU のストリーム機能を用い，深さ優先パイプライン化と幅優先パイプライン化を実現する．

深さ優先パイプラインのコードは，ストリーム数分に分割された CPU-GPU 通信，async 句を付したカーネル，GPU-CPU 通信の連続するコードをひとつのループボディとしたストリーム数分回転するパイプラインループで構成する．

幅優先パイプラインのコードは，ストリーム数分に分割された CPU-GPU 通信，async 句を付したカーネル，GPU-CPU 通信のそれぞれのコードを個別のループボディとするストリーム数分回転する個別のパイプラインループで構成する．

本方式の有効性を検証するため，姫野ベンチマー(data size:65536, 524288, 4194304) と K-NN(data size:262148, 524292, 1048580, 2097152)を対象に，パイプライン化をしない場合の実行とハンドコーディングにより本手法を適用した深さ優先および幅優先のパイプライン化(段数:4 段)を行った実行を比較した．その結果，姫野ベンチマークでは深さ優先方式で平均 13.1%，幅優先方式で 14.5%の速度向上が得られた．K-NN では深さ優先方式では速度向上は得られず，幅優先方式では 56.1%の速度向上が得られた．

また，大幅な速度向上が得られた幅優先方式による K-NN を対象として，開発した開発

した OpenACC to OpenACC トランスレータで変換したコードを実行した結果，ハンドコーディングによるコードの実行とほぼ同じ性能となるコードを生成できることを確認した．

この成果は，CPU-GPU 間のデータ転送とカーネル実行のオーバーラップを実現するために重要な技術である．

なお，本研究課題の前述の(1)～(3)の研究に関連して３名の博士前期課程学生の研究指導を行い，それぞれの学生は，「CPU と GPU 間のデータ転送時間の短縮に関する研究」，「ヘテロジニアスコンピューティングでのワークサイズ調整機構に関する研究」，「OpenACC プログラムの自動チューニングに関する研究」と題する学位論文を作成し修士学位を取得した．

＜参考文献＞

[1] 大島聡史，平澤将一，本多弘樹：既存の並列化手法を用いた GPGPU プログラミングの提案，情報処理学会研究報告(ARC-175)，pp.7-10(2007)．
[2] https://www.khronos.org/opencl/
[3] http://www.openacc.org/

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０件）

〔学会発表〕（計１件）

(1) 竹本拓未，和田康孝，近藤正章，本多弘樹：ヘテロジニアス GPU コンピューティングのためのワークサイズ自動調整手法の提案，情報処理学会　ハイパフォーマンスコンピューティング研究会 2015-HPC-148，2015 年 2 月 23 日．

〔図書〕（計０件）

〔産業財産権〕

○出願状況（計０件）

○取得状況（計０件）

〔その他〕

６．研究組織

(1)研究代表者

本多 弘樹（HONDA・HIROKI）

電気通信大学・大学院情報システム学研究科・教授

研究者番号：２０１９９５７４

(2)研究分担者

なし

(3)連携研究者

なし