jh250083

“All Core Computing”によるプログラム最適化に関する研究

大島聡史（九州大学）

概要

高い演算性能と高速なメモリ転送性能を有する GPU は様々なアプリケーションの高速化に有効であり、国内外を問わず多くのスーパーコンピュータへの採用が進んでいる。特に近年は生成 AI などの需要の高まりによりさらに GPU の需要が増加している。今日の GPU は大量の計算コアを有しており、その性能を十分に引き出すには対象問題にも高い並列度が要求される。そのため並列度が十分でないために計算コアを使い切れず性能効率が悪い GPU プログラムも珍しくない。さらに GPU スパコンではホスト CPU も高性能化しているが、多くの GPU プログラムは CPU 性能をほとんど活用していない。そこで本研究では、**GPU に搭載された大量の計算コアをフル活用する技術**と、**CPU と GPU を適切に用いる技術**をまとめて **All Core Computing** と称し、実現のための技術開発とそれを用いたプログラム最適化を行う。

1　共同研究に関する情報

1.1　共同研究を実施した拠点名

- 北海道大学 情報基盤センター
- 東北大学 サイバーサイエンスセンター
- 東京科学大学 情報基盤センター
- 九州大学 情報基盤研究開発センター

1.2　課題分野

- 大規模計算科学課題分野

1.3　参加研究者の役割分担

- 大島 聡史（代表）：総括、テーマ 1 の主担当
- 野村 哲弘（副代表）：TSUBAME 4.0 向け最適化補助、プログラム最適化支援
- 南里 豪志：テーマ 2 の主担当
- 遠藤 敏夫：TSUBAME 4.0 向け最適化補助、プログラム最適化支援
- 深谷 猛, 伊田 明弘, 河合 直聡：数値計算プログラムの知識とコードの提供および最適化協力

2　研究の目的と意義

最新の GPU は大量の計算コアを有しており、十分な計算性能を得るためには対象問題に高い並列性が要求される。そのため並列度が十分ではないために計算コアを使い切れず性能効率が悪い GPU プログラムも珍しくない。さらに GPU スパコンではホスト CPU も高性能化しているが、多くの GPU プログラムは CPU 性能をほとんど活用しておらず、計算資源がもったいないことも珍しくない。

そこで本研究では、**GPU に搭載された大量の計算コアをフル活用する技術**と **CPU と GPU を適切に用いる技術**をまとめて **All**

Core Computing と称し、実現のための技術開発とそれを用いたプログラム最適化を行う。具体的には、GPU スパコンのノード内でのマルチプロセス実行や、GPU 資源の分割や制限に関わる Multi-Process Service (MPS), Multi-Instance GPU (MIG), Green Context といった技術の活用を中心とした GPU の活用技術を研究する。

本研究は GPU スパコンの先進的な利用法に取り組む研究である。我々が関心を持つ対象アプリケーションの高性能化に貢献することは当然ながら、本研究の成果は我々以外の GPU スパコン利用者や、新たに GPU スパコン利用者として既存の CPU 向けプログラムを GPU 化しようとする者にとっても有益となることが期待できる。さらに、GPU スパコンの運用にも役立つ可能性のあるものであると我々は考えている。

3 当拠点公募型研究として実施した意義

本研究は最新の HPC 向け GPU の活用における課題の解決に資する研究課題である。今回利用を希望する九州大学 玄界 ノードグループ B、東京科学大 TSUBAME 4.0、JCAHPC Miyabi-G は最新の NVIDIA GPU を多数有しており、研究対象環境として申し分ない。玄界と TSUBAME 4.0 は同一型番の GPU を搭載しているが、CPU は 60 コア Intel Xeon（玄界）と 96 コア AMD EPYC（TSUBAME 4.0）であり異なる構成の多コア CPU であることから、性能の分析や比較を行うのに適している。

研究体制としては、GPU スパコンの運用経験が豊富であり GPU スパコンの構成や利用法に詳しいシステム系のメンバー（大島、南里、野村、遠藤）と、数値計算に関する知識と実績の豊富なメンバー（深谷、伊田、河合）による協力体制を敷いている。また得られた成果を各センタースパコンの利用者や運用者にフィードバックすることで、利用者や運用者の利益にも資することが期待される。特に MPS や MIG といった GPU の資源分割に関する技術は GPU スパコンの効率的な資源利用に重要であり、この点についての貢献も期待できる。

以上のように、利用可能な計算資源と研究内容の両方の点から、本研究は JHPCN 研究として実施する意義が大きい。

4 前年度までに得られた研究成果の概要

該当なし

5 今年度の研究成果の詳細

本研究では主なテーマ（対象問題）として以下の 2 つを挙げ取り組んだ。

- テーマ 1：ブロック低ランク行列（BLR 行列）に対する QR 分解（BLR-QR）を中心とした数値計算問題
- テーマ 2：CPU プログラムと GPU プログラムによる連携・連成計算問題

また次年度に向けた準備の意味も含めて

- テーマ 3：高度なマルチプロセス実行に関する研究

に着手した。

5.1 ブロック低ランク行列（BLR 行列）に対する QR 分解（BLR-QR）を中心とした数値計算問題

現在の HPC 向け GPU は数千〜数万の計算コアと高速なメモリを搭載している。その演算性能を十分に活用するには対象問題に高い並列度が必要であるが、GPU はコンテキストスイッチが高速であり、メモリアクセスのレイテ

ンシをコンテキストスイッチにより隠蔽することで高い総演算性能を発揮するアーキテクチャであるため、理想的な並列度は数万以上である。この並列度は大規模な行列計算などであれば現実的な数字ではあるが、例えばブロック化された行列を扱うような問題では個々の計算規模は小さくなってしまい、GPU の性能を使い切ることができない。これは GPU プログラム最適化における既知の課題であり、有望な解決策としては、小規模計算を多数行うバッチ計算が活用されている。バッチ計算は AI・機械学習処理にて広く利用されている計算方法であり、主要な数値計算ライブラリによる機能提供も行われいている。しかし、バッチ計算は全ての計算を一度に発行する必要があり、また 1 プロセスによる実行が基本である。そのため、例えばこれを MPI+OpenMP ハイブリッド並列化されたプログラムから利用するにはプログラムの実行方法やデータ構造の抜本的な見直しが必要となるため、適用は容易ではない。

本研究では 1 つの GPU を複数のプロセスで共有して利用するマルチプロセス GPU 実行に注目し、ブロック低ランク（Block Low Rank, BLR）行列に対する QR 分解（BLR-QR）の高速化を進めている。NVIDIA 社の GPU を利用する際には複数のプロセスによる GPU の共有利用が可能であるが、何の工夫もなく GPU を共有利用するとプロセスが GPU 資源を奪い合い、全体的な性能の低下が生じてしまう。特に数値計算問題アプリケーションでは高性能と生産性向上のために cuBLAS などの数値計算ライブラリが活用されるが、それらは GPU 資源を使えるだけ使って最大性能を達成しようとするのが自然であり、GPU 資源の奪い合いを避けることは難しい。

そこで我々は、プロセスあたりの利用可能資源を制限することができる Multi-Process Service(MPS) や、GPU を複数のサブ GPU に分割できる Multi-Instance GPU(MIG) に注目し、これらを用いたマルチプロセス GPU 実行の最適化に取り組んでいる。本年度は主に Miyabi-G に搭載されている GH200 プロセッサを対象とした性能の最適化と評価を行った。我々の BLR-QR プログラムはデータ構造がやや複雑（動的に作られた木構造）であることから、CPU-GPU 間のデータ転送を明に行わない実装方式を用いてきた。従来の実装ではこれを GPU ドライバが自動的にデータ転送を行う Managed Memory により実現してきたが、GH200 では Unified Memory が利用可能なため、Unified Memory 向けの実装を追加し、性能を評価した。

性能評価の結果を図 1 に示す。この結果から以下が明らかとなった。

1. H100(玄界), GH200(Miyabi-G) ともに、GPU を単純に 4 基使うよりも、MIG を用いて 1GPU 内マルチプロセス実行した方が高性能。
2. CUDA Fortran を使う場合、Managed Memory や Unified Memory はコンパイルオプション任せの実装よりも、本当に必要なメモリのみを手動で Managed/Unified 管理した方が高速。（実装の手間と性能にトレードオフがある。）
3. GH200 における Managed Memory コンパイルオプション任せの性能が低い。H100 と比べても大きく劣っており、何か問題があるように見える。

これらを含む成果を国際会議 MCSoC2025 にて発表した。

一方、現在の GPU は CPU と比べて高い演算性能やメモリ転送性能を有しているが、GPU を MIG による分割などでマルチプロセス利用

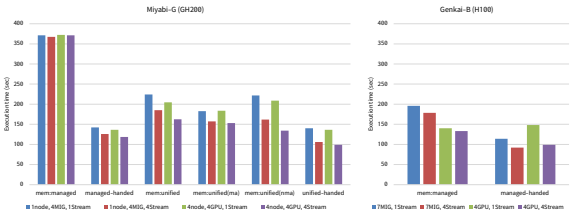

図 1　BLR-QR の性能：Miyabi-G と玄界における、各種の実装方法に対する実行時間の比較。handed の記載があるものは手動でメモリ管理をしたもの。

する場合、1 プロセスあたりの GPU 性能に対して CPU 性能が相対的に無視できない（無視するのはもったいない）程度に上昇する。そこで本研究では GPU に加えて CPU も活用することでより高い性能を得る技術についても研究を進めている。従来の GPU コンピューティングにおける CPU の介在は、GPU の動作を邪魔し性能を劣化させるものであり、GPU に加えて CPU の性能も活用することは挑戦的な課題である。実際にプログラムの設計を検討し実装を試みているが、性能が低下してしまうケースや、プログラムの構造によっては言語仕様上は適切なはずであるがコンパイラ・実行時ライブラリの挙動により期待通りの動作をしないケースなどがあり、NVIDIA 社の技術者とも連絡を取りながら継続して最適化を進めている。

5.2　CPU プログラムと GPU プログラムによる連携・連成計算問題

テーマ 2 について、GPU クラスタにおける CPU と GPU の計算能力を最大限に活用する手段として、CPU 側のみで計算を行う CPU プログラムと、主な計算を GPU に依頼する GPU プログラムによる連成計算を実現するために、通信バッファインタフェースを実装した。実装には南里らが開発し公開している連成フレームワーク CoToCoA (`https://github.com/tnanri/cotocoa`) を用いた。これは、別々に開発された複数の MPI プログラムに対して、簡単な通信関数を追加するだけで相互に連成させることを可能とするフレームワークである。一方、本研究で実装した通信バッファインタフェースは、連成計算に必要となる通信の効率化を目的として、バッファ領域に対する Put/Get による非同期通信、不連続領域のデータを通信するための pack/unpack、および CPU プログラムと GPU プログラムの間でのデータ分割方法の違いを吸収するデータ整形、の機能を提供するものである。

このうち非同期通信は、起動時に CPU 側のメモリ上に用意したバッファ領域に対して、CPU プログラム、GPU プログラムのいずれからも MPI の RMA 通信を利用して Put/Get を可能とすることで実現した。また、バッファ領域として送信側の数ステップ分の計算結果を保存可能な容量を確保することで、送信側は受信側の読み出しを待つことなく複数ステップの実行を進めることも可能とした。さらに GPUDirectRDMA を有効にすることで、低遅延でのバッファの読み書きを可能とした。

不連続領域の pack/unpack については、特に GPU プログラムでは pack/unpack のためのカーネルをそれぞれ作成した。これにより、バッファへの書き込み前に pack カーネルを、バッファからの読み出し後に unpack カーネルを、それぞれ実行することで通信の効率化を可能とした。

データ整形については、まずプログラム起動時に、各転送対象データについて、送信側、受信側、それぞれの分割次元、およびそれぞれの次元の分割数を登録する。その後、転送時の pack 時に、受信側の分割に合わせ、受信プロセスごとに必要なデータをまとめて pack する。そのため受信プロセスは自分のデータのみを読

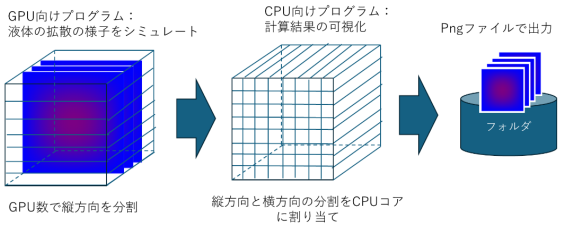

図 2　CPU プログラムと GPU プログラムの連成計算

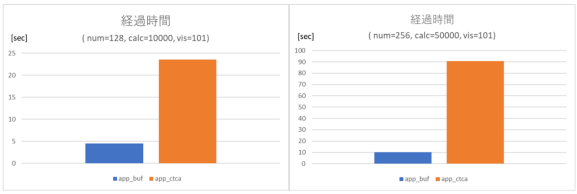

図　3　連成計算の実行時間　(左: 128x128x128、右: 256x256x256)

みだして unpack することが可能であり、データ転送量を必要最小限に維持できる。

この通信バッファインタフェースの性能を検証するために、GPU によるシミュレーションプログラムと CPU による可視化プログラムの連成計算を行った。シミュレーションプログラムは 3 次元ステンシル計算で液体の温度の拡散状況をシミュレートするもので、計算空間を縦方向で分割して各 GPU に分担させる。一方可視化プログラムは、3 次元の温度データを 2 次元平面で輪切りにしてそれぞれの面を PNG 形式で可視化する。可視化の各面について 2 次元で分割して各 CPU コアに分担させる。連成計算の様子を図 2 に示す。

この計算を玄界ノードグループ B で実行した。計算は 1 ノードで行い、シミュレーションプログラムには 4GPU、可視化プログラムは 64CPU コアをそれぞれ使用した。CoToCoA で通信バッファを用いずに実行した場合（app_ctca）と、通信バッファを用いた場合（app_buf）の実行時間を計測した結果を 3 に示す。図の左は計算空間の大きさが 128x128x128 でシミュレーションの計算ステップ数を 10000 回としたもの、右は計算空間の大きさが 256x256x256 でシミュレーションの計算ステップ数を 50000 回としたものである。いずれもの場合も、初期値の可視化を含めた可視化の回数が 101 回となるように可視化の間隔を調整している。

実験結果から、CoToCoA により同じノード内の CPU プログラムと GPU プログラムの連成計算を行えることが確認できた。今回示したのは 1 ノードによる実行であるが、CoToCoA の通信基盤は MPI であるため複数ノードでも同様に実行できることを確認できている。また、図より、本研究で実装した通信バッファインタフェースにより、大幅な性能向上が得られていることが確認できた。

5.3　高度なマルチプロセス実行に関する研究

テーマ 3 では、プロセスごとの計算負荷が不均一となる状況を想定し、GPU ごとの負荷不均衡を緩和するための高度なマルチプロセス実行方式を新たに検討している。具体的には、ノード内に GPU 数以上のプロセスが存在する場合に、各プロセスの計算量に基づいて利用する GPU を決定し、GPU 間の負荷をできるだけ均一化することを目指す。提案するプロセスと GPU の対応関係、および GPU へのプロセス割り当ての概要を図 4 に示す。本方式では、GPU へのプロセス割り当てを、GPU 間の負荷分散を目的とした Load Variance 最小化問題として捉える。実装では、Largest Processing Time first（LPT）に基づき、計算量の大きいプロセスから順に、現時点で割り当て済み負荷が小さい GPU へ割り当てる方式を採用した。これにより、プロセスごとの計算負荷に偏りがある場合でも、GPU 間の負荷をできるだけ

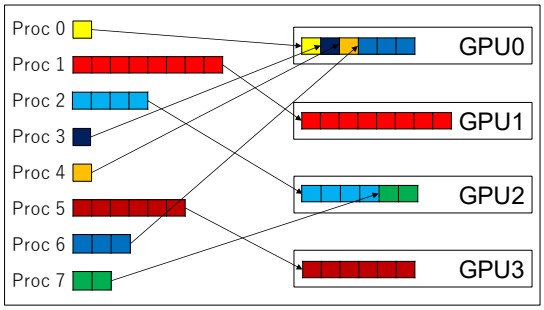

図 4　GPU ロードバランシング概念図（1 ノード、4GPU、8 プロセス）

均一化することを狙っている。基本的な実装は完了しており、現在は性能評価と実装方式の改良を進めている。簡易評価の結果、Managed Memory を利用した実装では GPU 間のデータ migration に伴うオーバーヘッドが非常に大きく、GPU 間で動的に計算を割り当てる本テーマの実行形態では性能上の大きな制約となることが判明した。そのため、Managed Memory を前提とした実装のみでは十分な性能を得ることが難しいと判断し、CUDA を用いて migration を明示的に制御・最適化する実装を進めている。現在、この CUDA ベースの最適化実装について評価を行っている段階である。なお、本研究は、別途研究を進めている DCB（Dynamic Core Binding）ライブラリの GPU 対応への拡張としても位置づけられる。今後、CPU コアに加えて GPU を含むノード内資源を計算負荷に応じて柔軟に割り当てるための基盤技術として発展させることを目指す。

6　進捗状況の自己評価と今後の展望

テーマ 1 については、「MIG, MPS, Green Context を活用したプログラムの最適化実装」「Grace Hopper 向けの最適化」を掲げており、概ね期待通りの成果が得られた。Green Context については、現時点では本テーマでは活用が難そうであるが、テーマ 3 での活用を予定している。「CPU と GPU による分業も含めた実装」については様々な実装方法を試みているが、まだ実装のアイディアを試し切れておらず、次年度に向けた継続課題である。

テーマ 2 については、同じノード内の CPU と GPU にそれぞれ別のプログラムを割り当て、それらを連成させる計算を、CoToCoA フレームワークを用いて実現できることを確認した。また、プログラム間の通信を仲介する通信バッファインタフェースを整備し、片側通信による双方のプログラムの非同期実行、不連続領域データの pack/unpack による通信頻度低減、およびデータ整形によるプログラム間のデータ分割方式の違いを吸収を実現した。さらに、実験により、この通信バッファインタフェースにより連成計算の性能が向上することを検証した。今後の課題として、異なる組み合わせでの連成計算によりデータ整形や pack/unpack の汎用性の向上を図る。また、テーマ 1 と連携し、CPU と GPU の分業という視点での連成計算の可能性を検討する。

テーマ 3 については、GPU 間の負荷不均衡を緩和するため、各プロセスの計算量に基づく GPU 割り当て方式を実装した。具体的には、GPU へのプロセス割り当てを Load Variance 最小化問題として定式化し、LPT に基づいて計算量の大きいプロセスから順に割り当て済み負荷が小さい GPU へ割り当てる方式を実装した。一方で、簡易評価により、Managed Memory を用いた実装では GPU 間 migration のオーバーヘッドが非常に大きく、動的な GPU 割り当てを行う本テーマでは大きな性能低下要因となることが判明した。この結果から、Managed Memory ベースの実装のみで高性能化を図ることは難しいと考えられる。現在は CUDA を用いて migration を明示的に制御す

る実装へ移行し、GPU 間データ移動の最適化と性能評価を進めている。今後は、CUDA ベースの migration 最適化の有効性を評価し、GPU 間データ移動のオーバーヘッドを低減する実装方式を確立する。また、プロセス数、GPU 数、計算負荷の偏りが異なる条件で評価を行い、本方式が有効となる実行条件を明らかにする。さらに、テーマ 1 で得られた MPS 等を用いたマルチプロセス GPU 実行の知見を本テーマにも適用し、複数プロセスが同一 GPU を利用する場合の資源制御や性能安定化について検討を進める。最終的には、別途研究している DCB ライブラリの GPU 対応としても活用できるよう、GPU を含むノード内資源を計算負荷に応じて柔軟に割り当てる方式へ発展させることを目指す。