jh220036

大規模アプリケーションの高性能な実用的アクセラレータ対応手法

下川辺　隆史（東京大学）

概要　近年、電力と設置面積の制約のもと、最大限に計算性能を高くするために、多くのスパコンで GPU などの演算加速装置が導入されている。本課題では、スパコンで動作している CPU アプリケーションを演算加速装置を搭載したスパコンへ移植し、その移植の方法を確立することを目的とする。様々な分野の研究者が開発した CPU アプリケーションを移植することを念頭に、演算加速装置で高い性能を達成することを目指しながらも、専用言語を多用せず、標準的・汎用的手法を用いて移植を実現する。実用的な移植方法として、OpenMP、OpenACC、do concurrent などの活用を検討する。本年度は、地震波伝播・強震動シミュレーションコード OpenSWPC を対象として do concurrent を用いて GPU 化し、GPU スパコンである東京大学 Wisteria-Aquarius でで性能測定した。インライン展開や通信最適などを導入し、1 GPU で 8 CPU 程度の計算性能を達成し、64GPU（8 ノード）まで比較的良好な弱スケーリング結果を得た。

1. 共同研究に関する情報

(1) 共同利用・共同研究を実施している拠点名（該当するものを残す）

東京大学 情報基盤センター

大阪大学 サイバーメディアセンター

(2) 課題分野（該当するものを残す）

大規模計算科学課題分野

(3) 共同研究分野（HPCI 資源を利用している研究課題のみ，該当するものを残す）

超大規模数値計算系応用分野

(4) 参加研究者の役割分担

- 下川辺 隆史（東京大学）：移植コード開発と最適化
- 額田 彰（筑波大学）：移植コード開発と最適化
- 古村 孝志（東京大学）：OpenSWPC に関する助言
- 羽角 博康（東京大学）：COCO に関する助言
- 川崎 高雄（東京大学）：COCO に関する助言
- 山岸 孝輝（高度情報科学技術研究機構）：COCO に関する助言
- 朴 泰祐（筑波大学）：演算加速装置に関する助言
- 高橋 大介（筑波大学）：演算加速装置に関する助言
- 三木 洋平（東京大学）：演算加速装置に関する助言
- 塙 敏博（東京大学）：大規模計算に関する助言
- 中島 研吾（東京大学）：計算科学に関する助言
- 星野 哲也（東京大学）：演算加速装置に関する助言
- 大森 拓郎（東京大学）：移植コード開発
- 畠山 昂（東京大学）：移植コード開発
- 佐久間 大我（東京大学）：移植コード開発
- Ziheng Yuan（東京大学）：移植コード開発
- 勢見 達将（筑波大学）：移植コード開発

2. 研究の目的と意義

近年、電力と設置面積の制約のもと、最大限に計算性能を高くするために、多くのスパコンで GPU などの演算加速装置が導入されて

いる。今後この傾向はさらに加速すると予想され、将来のスパコンでは、大多数のアプリケーションで高い性能を引き出すために演算加速装置の利用が必須となる。このような中、東京大学情報基盤センターと筑波大学計算科学研究センターと共同で設置する最先端共同 HPC 基盤施設（JCAHPC）では、Oakforest-PACS（OFP）の後継機（OFP-II）を 2024 年 4 月に稼働予定であり、その OFP-II においても演算加速装置を導入する方向で検討を進めている。OFP をはじめとした両センターが運営に関わるスパコンでは、CPU に最適化された多数のアプリケーションが実行されている。OFP-II の稼働開始に向け、これらのアプリケーションを演算加速装置を導入した OFP-II へ効率的に移植していく指針の確立とその手法の構築が必須である。

本研究課題では、スパコンで動作している CPU アプリケーションを演算加速装置を搭載したスパコンへ移植し、その移植の方法を確立することを目的とする。様々な分野の研究者が開発した CPU アプリケーションを移植することを念頭に、演算加速装置で高い性能を達成することを目指しながらも、高性能計算分野でない研究者も難なく扱えるように、演算加速装置専用の開発言語を多用せず、標準的・汎用的手法を用いて移植を実現する。実用的な移植方法として、OpenMP、OpenACC、do concurrent などの標準構文による GPU 化を検討する。実用的な手法による GPU への移植に関する知見が得られたのちに、実例として海洋大循環モデル COCO と地震波伝播・強震動シミュレーションコード OpenSWPC の 2 つのスパコンで動作する実 CPU アプリケーションを対象に移植を行い、性能評価を行う。

本課題は、個別のアプリケーションで最高性能を目指して、ただ GPU へ移植することを目的にするものではなく、様々なアプリケーションへ汎用的に適用できる現実的な移植の方法論を検討し確立することを目指す。スパコンにおける演算加速装置の利用は主流となってきており、様々な CPU アプリケーションを演算加速装置である GPU へ移植する方法論の確立を目指す意義は大きい。

3. 当拠点の公募型研究として実施した意義

本研究課題は、既存のアプリケーションを演算加速装置へ移植し、その手法を汎用的な方法として確立する。これにはアプリケーション分野と高性能計算分野の専門家による密な連携が必須である。

アプリケーション分野の専門家として COCO の開発者、OpenSWPC の開発者が参画する。アプリケーションのコードや計算に必要なデータの提供とともに、データ構造に関する知見を提供する。また、アプリケーションの将来性を考えると、移植したコードはそれぞれのコミュニティで継続して開発できることが重要となる。これにはアプリケーション分野からの視点は必須であり、この役割を担う。高性能計算分野からは演算加速装置への最適化の専門家が参画し、コードの原型を保ちつつ、最大の性能を引き出す手法を検討する。我々はこれまでに CPU と GPU 両方で実行可能な性能可搬性が高いフレームワークの開発経験があり、この知見を生かす。このように様々な専門家が密に連携することにより、実用に耐えうる汎用的な移植手法の確立を目指す。

本研究課題は、演算加速装置として NVIDIA A100 GPU を対象とし、様々なスパコンにおける移植可能性を検証する計画であり、東京大学の Wisteria-Aquarius と大阪大学の SQUID (GPU)は本研究を遂行する上で最適な計算環境である。

このように本研究は当拠点公募型共同研究として計算環境を効果的に利用し、共同研究を遂行することで着実に成果を出すことができている。

4. 前年度までに得られた研究成果の概要

該当しない。

5. 今年度の研究成果の詳細

本年度前半は、OpenMP、OpenACC による並列化など、実用的な移植方法の検討を進めた。特に、拡散方程式を用いて、これらの生産性と高い性能を得るための最適化について検討した。

並行して、COCO と OpenSWPC のコードの構造の分析と検証を行った。本研究課題は 2 年計画で、これらの 2 つのアプリケーションを GPU へ移植する。検討の結果、初年度となる本年度はコード規模が比較的小さく、構造がわかりやすい OpenSWPC を GPU 化する方針とした。

本年度後半は、実際に OpenSWPC の GPU 化を進めた。OpenSWPC は Fortran で記述され、OpenMP+MPI のハイブリッド並列化が既に行われている。検討の結果、Fortran の標準並列化構文 DO CONCURRENT で GPU 化できることがわかり、GPU 移植をおこなった。その結果、東京大学の Wisteria-Aquarius ノードを用いると、1 GPU で 8 CPU 程度の計算性能を達成できた。

以下では、OpenMP、OpenACC を用いて GPU 化した拡散方程式の性能比較と OpenSWPC の GPU 化についての研究成果について説明する。

5.1 OpenMP 、OpenACC を用いて GPU 化した拡散方程式の性能比較

OpenMP Offload は 、OpenMP を用いて GPU 等の加速器デバイスで演算を行うための機能である。現在の主な GPU ベンダーである NVIDIA, AMD において公式にサポートされており GPU ベンダーに束縛を受けない API として注目を集めている。

OpenMP Offload における演算性能を調べるため、拡散方程式においてネイティブな環境と OpenMP Offload での実行性能の比較を行う。GPU としては NVIDIA A100 GPU と AMD MI100 GPU を用いる。データ転送は計測区間外で行われている。AMD MI100 に関しては GPU フレームワークである ROCm が発展途上なこともありバージョン 4.5.2 と 5.1.3 に関して評価する。NVIDIA に関しては OpenMP Offload に関しては NVIDIA HPC SDK22.2, CUDA に関して は CUDA11.4 を用いる。NVIDIA GPU においてはネイティブコードは NVCC コンパイラ, OpenMP Offload コードは NVC++ コンパイラでコンパイルする。AMD GPU においてはネイティブコードは HIP コンパイラ, OpenMP Offload コードは Clang++ コンパイラを用いる。実験条件としては (nx, ny, nz) = (512, 512, 512) とし 100,000 ステップの演算を行なう。単一 GPU での拡散方程式における性能測定の結果を図 1 に示す。CPU-GPU 間でメモリコピーを介在しない演算性能としては NVIDIA A100 に関して OpenMP Offload は CUDA 比で 40.3% の演算性能を持つことがわかる。AMD MI100 に関しては ROCm4.5.2 において HIP 比で 26.5%, ROCm5.1.3 において HIP 比で 35.2% の演算性能を持つ。

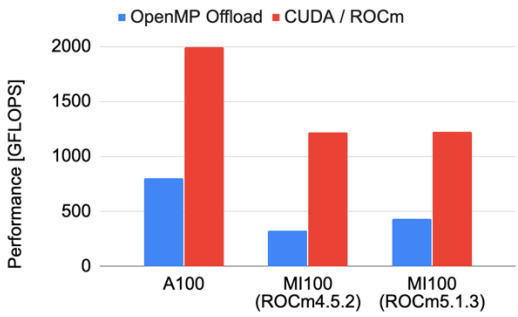

図 1 : ネイティブな環境と OpenMP Offload での拡散方程式の実行性能の比較。NVIDIA A100 および AMD MI100 GPU を用いている。

次に、現状では NVIDIA 向け限定であるものの指示文を用いたプログラミング手法として OpenACC が存在する。ここでは、NVIDIA GPU に対して、OpenMP Offload との性能比較を行う。図 2 に、OpenACC と OpenMP を用

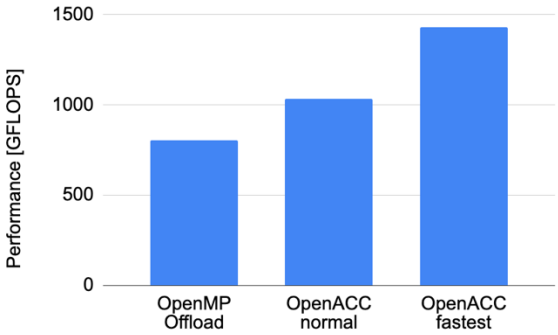

図 2：OpenACC と OpenMP Offload を用いて GPU 化した拡散方程式の性能測定の結果。

いた場合の拡散方程式における性能測定の結果を示す。まず OpenACC normal はパラメータ調整なしの実装で OpenMP Offload 比で 128% の性能が出る。OpenACC fastest は実行スレッド数のパラメータ調整を最適化したもので、OpenMP Offload 比で 138% の性能が出る。パラメータ調整には GPU に関する知識や経験が必要とはいえ、現状 NVIDIA の GPU を指示文で扱う上では OpenACC に性能面では分があると言える。

5.2 DO CONCURRRENT 構文による OpenSWPC の GPU 化

5.2.1 OpenSWPC の GPU 化手法

OpenSWPC はオープンソースの地震波の伝播シミュレーションコードである。Fortran で記述され、OpenMP+MPI のハイブリッド並列化が既に行われている。このコードを GPU に対応するにあたり、ノード内の OpenMP 並列化部分に関して GPU で実行するべき箇所を Fortran の標準並列化構文”do concurrent”に変更し、CPU のマルチスレッドで実行する箇所は引き続き OpenMP 並列化を用いるという方針を採用した。NVIDIA HPC SDK の nvfortran コンパイラで-stdpar=gpu と-mp のオプションを付けた際には意図する通りに GPU 実行と CPU 並列実行を使い分けることが可能である。

OpenSWPC は複数の 3 次元配列に対してステンシル計算のような処理を繰り返す部分が支配的である。CPU 用の OpenMP 並列化では大きなスレッド数を想定しておらず最外ループの並列化だけが行われていた。この部分を GPU で実行する際にはより多くの並列性が必要となるため 3 重ループをまとめて並列実行するように変更する。OpenSWPC は空間を 2 次元分割しているため効率を優先し再内ループ（＝メモリ上で連続）は鉛直方向になるが、材質によって計算内容のが異なるため地形に合わせて再内側の処理が異なる 3 重ループが複数実行されるコードになっている。まず再内ループの反復回数が一定でないと効率の良い GPU 実行ができず、また正しいコードが生成されないというコンパイラのバグも見つかっている。そこで再内ループの範囲を一度シミュレーション領域全体に置き換え、ループ内で実際に対象範囲であった場合にのみ計算を行う条件文を追加した。次に複数の 3 重ループをマージして一回の GPU カーネル実行に変更した。これによりメモリアクセス回数の低減やキャッシュメモリ利用率の向上に繋がり、またカーネル呼び出しのオーバヘッドも削減される。

“do concurrent”内の処理には制限があり、関数呼び出しをする場合には純粋関数（pure function）でなければならないという言語仕様がある。さらに NVIDIA のコンパイラは現状でこの純粋関数の呼び出しにも対応していないという問題があり、関数呼び出し部分ではインライン展開をする必要があった。

NVIDIA の実装では全てのメモリを CUDA managed memory という種類のメモリを採用することで GPU 化を実現している。このメモリは CPU と GPU のどちらからもアクセス可能であり、そのメモリページは最後にアクセスしたデバイスに自動的に移動する。このためたとえ並列性が低く CPU で行うべき処理であってもメモリページが移動してしまうことを避けるために”do concurrent”構文を使う

ことが望ましい。

GPU による並列実行では GPU 内での計算は高バンド幅のデバイスメモリを用いて効率的に行うことができるが、一方で GPU 間のデータ転送はネットワーク帯域等に制限される。DO CONCURRENT では ALLOCATABLE で動的に確保されたメモリ領域が全て CUDA Unified Memory として確保され、通常はこれが MPI 通信における通信バッファとして使われる。この Unified Memory はホストメモリとデバイスメモリの間でページ単位で動的に移動する。このため正常に通信を行うために InfiniBand HCA などの 通信デバイスが転送を行う場合一旦ホストメモリに別途確保されたバッファにコピーし，その後で通信が行われると いう実装になっており、通信性能が著しく低下する。

成瀬らは C++標準の並列化構文による GPU 実行において同様の問題に直面し，その回避方法として通信用バッファとなるメモリ領域には CUDA C/C++用の cudaMalloc() API を用いてデバイスメモリを確保し、GPUDirect RDMA によって他の GPU との間で効率よくデータ転送できるようにしている。DO CONCURRENT では Fortran をベースとするため cudaMalloc を呼ぶことはできない。そこで図 5 に示すような方法を採用する.まず CUDA Fortran の機能を利用して明示的にデバイスメモリを確保する.さらに DO CONCURRENT ループでカーネルがこのデバイスメモリにアクセスするために，OpenACC の指示詞でデバイスメモリを指していることを明示する。これによって GPUDirect RDMA を利用したデバイスメモリ間のデータ転送を行うことができ、通信性能が大きく改善される。

5.2.2 性能評価と評価環境

性能評価には OpenSWPC の既存コードと、これまでに示した各種最適化を適用した GPU コードを用いて行う。シミュレーション用のデータとして、関東エリアを対象範囲とする小田原地震データ及び OpenSWPC に付属するテスト用データ (input.inf) の 2 種類を用いる。小田原地震データは地形情報を含むフルシミュレーションが行われるデータで強スケーリングの評価に用いる.並列度によらず空間グリッド間隔は各方向 0.25km、グリッド数は 1024 × 1024 × 400 である。テスト用データは領域サイズ を変更し GPU コードの弱スケーリングの評価に用いる。

評価環境として，東京大学情報基盤センターが運用する Wisteria/BDEC-01 の Aquarius を用いる。コンパイラは CPU コードには Intel 社の ifort，GPU 用コードには NVIDIA の nvfortran コンパイラを用いる。GPU 上でコードを実行するためにはコンパイルオプションに-stdpar=gpu を指定する必要がある。環境変数 CUDA VISIBLE DEVICES を用いて 1 ランクに 1GPU が割り当てられるようにしている。

5.2.3 小田原地震データによる性能評価結果

実行結果は各環境において入力されたデータを規定タイムステップ数で計算したものである。図 3 に GPU 用コードによる小田原地震データでの実行時間を示す。global__comm で始まるものは MPI による通信処理、kernel__ や absorb__ で始まるものはステンシル計算，source_stressdrop は震源の処理，output_*や report_progress は出力関連の処理，*_setup,vmodel は 初期化処理である。

通信最適化を適用したコードが A、最適化されていない DO CONCURRENT のみのコードが B である。8GPU までは全ての通信はノード内で行われ、16GPU 以上でノード間通信が行われる。いずれの状況でも通信最適化により global_comm_stress，global_comm_vel ともに 大きく時間短縮されている。並列度

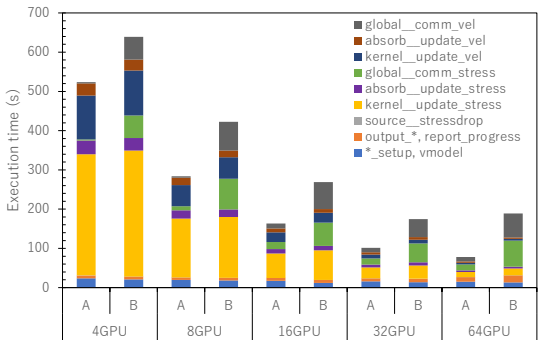

図 3: GPU コードによる小田原地震データの実行時間

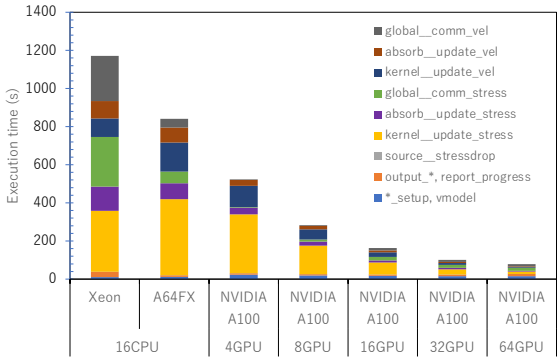

図 4: GPU コードとオリジナルコード (Aquarius の Xeon CPU と Odyssey の A64FX CPU での実行) による小田原 地震データの計算時間の比較

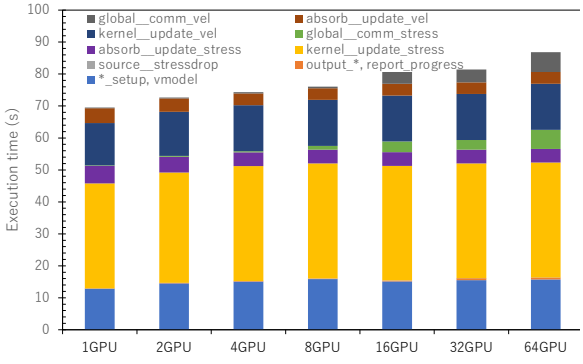

図 5: GPU コードでの弱スケーリングの評価

を上げると kernel__*, absorb__*等の GPU でのステンシル計算の実行時間は順調にスケールしているが、output_*の実行時間は逆に伸びている。これは定期的に snapshot を保存する処理で、多数のノードからデータを縮約してファイルへの書き込みを行う。結果として 64GPU の計算速度は 4GPU の約 5 倍 に留まっている。以降では通信最適化を行ったコードを用いる。

図 4 に GPU 用コードの各 GPU 数の実行と，16CPU を 利用した CPU 用コードの実行時間の比較を示す。また、図 4 には比較のため Wisteria-Odyssey ノードの A64FX CPU で実行した結果も併せて記載する。CPU と比較すると GPU の 方が計算性能は高く、4GPU での実行時間との比較によると、1GPU の性能が Xeon 9CPU 分、A64FX 6CPU 分相当であると考えられる。

5.2.4 テスト用データによる弱スケーリング結果

GPU 用コードでは CPU 用コードほど良いスケーリングが確認できなかったため、その原因として問題サイズが十分ではないことが考えられ、GPU 用コードでの弱スケーリングの評価を行った結果を図 5 に示す。データはテスト用データを用い、512 × 512 × 400 グリッドを基準に使用 GPU 数ごとに領域サイズを増加させている。

16GPU までは GPU 数が増えるごとに通信方向が増えるため、それに伴い通信処理 (global__comm_*) にかかる時間が長くなっている。16GPU 以降は通信方向、通信デー タ量も一定であるが、64GPU で通信時間が増加している。16GPU、32GPU の時にはともに東西方向に 4 分割している。この場合 8GPU を搭載する計算ノードでは東西方向はノード内通信となり、南北方向についても一方はノード内通信となっている。これに対して 64GPU の時は東西南北それぞれ 8 分割され、南北方向は両方ノード間通信となるためノード間通信が倍増する。GPU 数がさらに増加すると東西方向でもノード間通信が必要になり通信時間がもう 一段階 (25%) 増加すると考えられるが、1GPU あたりの処理する

データサイズが十分であれば性能はスケールすることが期待できる。

6. 進捗状況の自己評価と今後の展望

今年度は、当初の計画どおり、OpenMP と OpenACC により GPU 化された拡散方程式の性能評価を行い、ネイティブな環境を用いた実装と比較し、OpenMP ではネイティブ環境と比べて約 40%程度の性能にとどまることが明らかになった。

次に、当初の計画通り、COCO と OpenSWPC のコードを検証し、今年度はコード規模が比較的小さい OpenSWPC の GPU 化を進めた。OpenSWPC は Fortran で記述され、OpenMP+MPI のハイブリッド並列化が既に行われている。検討の結果、Fortran の標準並列化構文 DO CONCURRENT で GPU 化できることがわかり、インライン展開の適用や通信最適化などを導入し、GPU 移植をおこなった。その結果、東京大学の Wisteria-Aquarius ノードを用いると、1 GPU で 8 CPU 程度の計算性能を達成できた。

来年度は、まず OpenSWPC を対象に、ノード間の GPU 間通信手法とそれを伴うデータ入出力の性能向上を行い、アプリケーション全体としての性能向上を目指す。ノード間の GPU 間通信の性能向上のためには、データを GPU または CPU メモリに固定する必要があり、OpenACC または CUDA を利用した手法を開発する。これの実現方法の一つは 5.2.1 で述べた方法であり、当初の計画よりも進んでいる。また、シミュレーションのデータ入出力は一般にノード間の GPU 間通信を伴うため、これも性能向上する方法を検討し、導入する。特にデータ入出力ではストレージ性能にも大きく依存するため、ノード間通信に加えてストレージ性能を考慮した手法を開発する。これらの高速化手法を導入後、OpenSWPC アプリケーション全体の性能を評価する。

来年度後半は、前半で OpenSWPC に対して確立した高性能なノード間の GPU 間通信手法とそれを伴うデータ入出力は、実アプリケーションで多様されるため、これを汎用的に利用できるように、フレームワークやライブラリなどの形で整備する。また、計画通りに、COCO の GPU への移植を開始する予定である。性能測定はには、引き続き、東京大学の Wisteria-Aquarius と大阪大学の SQUID (GPU)を利用する予定である。

7. 研究業績

(1) 学術論文 （査読あり）

該当なし

(2) 国際会議プロシーディングス （査読あり）

該当なし

(3) 国際会議発表 （査読なし）

[1] Takuro Omori, Takashi Shimokawabe, “Performance Optimization Of Lattice Boltzmann Method On A64FX,” 15th World Congress on Computational Mechanics & 8th Asian Pacific Congress on Computational Mechanics, Online, August 2022.

[2] Tatsumasa Seimi, Akira Nukada, “GPU Acceleration of OpenSWPC using DO CONCURRENT”, GPU Technology Conference 2023 Spring, Poster, Online, Mar. 2023.

(4) 国内会議発表 （査読なし）

[3] 大森 拓郎, 下川辺 隆史, 朝比 祐一, “OpenMP Offloading を用いた GPU での格子ボルツマン法実行における性能評価”, 第 27 回計算工学講演会, オンライン, 2022 年 6 月.

[4] 勢見 達将, 額田 彰,”DO CONCURRENT 構文による OpenSWPC の GPU 化”,研究報告ハイパフォーマンスコンピューティング（HPC）,2023-HPC-188(19),1-7 (2023-03-09), 2188-8841

(5) 公開したライブラリなど

該当なし。

(6) その他（特許，プレスリリース，著書等）

該当なし。