GPUの気象・気候・海洋モデルへの適用とその最適化を目指した取り組み

Activities towards an application of GPU to numerical weather/climate/ocean model and its optimization

高度情報科学技術研究機構
山岸　孝輝

次世代のエクサスケール･スーパーコンピュータでは、性能及び電力効率の高さが求められる。その両方を備えたシステムとして、演算性能とメモリ転送性能に特化した、アクセラレータを用いたヘテロジニアスなシステムが広がりを見せている。演算性能、電力効率の世界ランキングの両方においてアクセラレータを用いたシステムが多数上位に入っており、その性能に対する期待は大きい。その一方、アクセラレータは数値計算に特化したプロセッサであることに加え、CPUとの複合システムであることから、ユーザの開発及び最適化の負担が大きい事が問題となっている。本稿では、これらのアクセラレータの内、最も普及しているNVIDIA社のGPUを取り上げ、現在及び今後のスーパーコンピューティングにおけるGPU適用の背景、基礎並びに今後の課題について概説し、併せて我々が行っている気象・気候・海洋モデルへのGPUの適用とその最適化について紹介する。

1．はじめに

地球温暖化を初めとする気候変動の予測、極端現象・異常気象による災害の予防と即時対応の手段として、またそれらの基礎となる物理プロセスを理解する手段として、気象、気候及び海洋の数値シミュレーションの重要度は高まる一方である。それ故、気象･気候･海洋モデルの高度化の需要は大きい。本稿では、気象・気候・海洋モデルの有効な高度化手段の一つとして、アクセラレータ（NVIDIA GPU）への対応についてその背景、基礎並びに今後の課題について概説し、併せて我々の研究成果を紹介する。

2．気象・気候・海洋モデルの特徴と開発の背景

気象・気候・海洋モデルでは、複雑かつ多くの種類の現象をまとめて扱う事が特徴である。例えば、気象の数値モデルで扱う大気現象をスケール（代表的な大きさ）の違いで考えると、空間スケールは地球1周4万km、大気拡散の運動は数mmである。季節の移り変わりは数ヶ月、気候変動となると数十年、風の流れは数時間から数分である。つまり、時間・空間スケールの幅は非常に大きい。関連する物理現象も、大気放射、エアロゾル（大気中の塵）、大規模凝結、積雲対流、降水に地表面過程など多岐にわたり、これらが相互作用しながら、いわゆる気象システムを形成している。気象の数値モデルは、これらの現象の相互作用を時間・空間的に広いスケールで扱う、高度な数値モデルといえる。

より良い精度の予報並びに予測とそのプロセスの理解には、モデルをより高解像度で実行することが求められる。加えて、防災という観点からは予報の即時性が求められるため、予報が得られるまでの経過時間の減少も重要な要素である。また、統計的手法により予報の精度を上げるためには十分なアンサンブル数の確保が必要であり、経過時間の減少

は重要である。数値モデルはその名の通り、実際の自然現象を近似して数値化しているのであるが、より詳しく現実を再現・理解するためにはなるべく近似を行なわず、より多くの力学・物理過程を直接表現すること、つまりモデルの精微化が必要となる。

以上に挙げた高度な気象・気候・海洋モデルへの要求事項（高解像度化、高速化、精微化）に対する制約の一つは、計算機の能力と実効性能である。例えば、メッシュサイズに依って数値モデルに含む要素やモデル化の手法も決まり、その結果、研究の対象にも制約が生じる。計算機の能力・実効性能と研究対象の関係は線形では無く、ある閾値を境に質的な変化をも含む。これは同じ計算機の能力でも、高度化の内容次第で研究の質ががらりと変わりうるということである。

気象・気候・海洋モデルは社会的な重要度が大きくかつ影響を与える分野が広いため、利用者の数も使い方も自ずと増える。直接開発に携わらない人や専門的な領域のみ詳しい人が占める割合が増える事に加え、数値モデルを実行する環境も多様化する。それ故、なるべく汎用性が高くて実行環境を選ばないこと、簡単に開発・高度化できること、コードの理解に誰もが簡単に取り組めること等が重要となる。

これまでに挙げた課題を計算科学の言葉で言い換えると、気象・気候・海洋モデルの高度化は、弱いスケーリング（高解像度化）と強いスケーリング（高速化）の両方が求められていて、そのコードはさらなる複雑化を求められ、生産性（容易な開発）やポータビリティ（多様な実行環境に対応）も求められるという、実に挑戦的な取り組みといえる。

これらの課題に対してどのように対応してきたかについて、これまでの日本での気象・気候・海洋モデル開発の歴史を振り返ってみると、ベクトル計算機の並列クラスタである地球シミュレータから、超並列なスカラ計算機への移行を経て京コンピュータでの大規模実行を可能とする開発と実行など、革新的かつ多大な努力が多方面から払われてきた。この開発の流れに今後さらに影響を与えるものに、電力効率改善の要求がある。その要求に対応するため、次世代のエクサスケール・スーパーコンピュータでは、これまでの超並列化の流れに加えて、アーキテクチャやシステム構成の変更をも伴う大きな転換を求められている。

電力効率を上げる方法として、プロセッサの動作周波数を下げると共にコア数を増やし(半導体プロセスの集積度の増加分をコア数増加に転換させて)、電力消費量を抑えたままでチップあたりの演算性能は向上させる方法と、目的に特化した設計として無駄を省く方法がある。ベクトル計算機からマルチコアを搭載した超並列なスカラ計算機への移行においては、前者の方法が主に取られてきた。今後の流れでは、さらに後者の方法も重要とされている。超並列化及び構造の特化で無駄を省く方法の両方を採用するプロセッサとして近年広がりを見せているのは、従来のCPUと組み合わせて用いるアクセラレータである。CPUとアクセラレータの組み合わせを採用したヘテロジニアスな構成となるスーパコンピュータ（以下スパコン）は、演算性能(TOP500)、電力効率（Green500）の世界ランキングの両方において多数上位に入っており、TOP500での最新の１位はGPUを搭載した米国のSummitである。日本では産業技術総合研究所のABCI（５位)、東京工業大学のTSUBAME3. 0（19位）などがランク入りしている。

アクセラレータの広まりと同時に課題として上がってきたのは、プロセッサの超並列化とスパコンシステムのヘテロ構成化（CPU＋アクセラレータの組み合わせ）がアプリ制作側の負担となることである。これは、気象・気候･海洋モデルのみが抱える問題では無い。

本稿では、特に気象・気候・海洋モデルがこれから迎える超並列化の時代に抱えるであろう課題とその対応について、関連する背景や基礎を紹介しつつ纏めていく。後述するが、気象モデルで扱うカーネルは構造格子でメモリアクセスも系統的である故に他のアプリにも適用可能な高度化のノウハウを含むものであり、他の事例にも適用可能と考えている。また、近年最も広まっているアクセラレータはNVIDIA社のHPC向けグラフィックカード（GPU）を採用したものであり、本稿では全てNVIDIA社のGPUを前提とする。

3．GPUとは

本章では、GPUの基礎について、気象・気候・海洋モデルに関係が深い部分を中心にまとめる。さらなる解説等はNVIDIAの公開文書［1，2］を参照されたい。

元来はグラフィック用のプロセッサをHPC向けに利用したもので、HPCで主に使われる上位機種では、グラフィック処理ではさほど必要としない倍精度演算が強化されており、浮動小数点の表現ではIEEE754－2008に準拠し、デバイスメモリにはECCが搭載されているなど、科学技術計算において求められる機能を備えている。コンパイラやライブラリ、プロファイラ等のソフトウェアも提供され、ハード・ソフトの両面で十分な機能を備えている。

これまでの一般的なCPUに比べて、数値計算に特化したことで性能に対する電力の効率が高いことが特徴である。しかしながら、特化したが故にCPUとは異なる並列化手法が求められ、プログラミングにおいていくつかの制約が発生する。

数値計算に特化して構造を単純化させたことは、使いにくさと使いやすさの両方につながる。GPUは基本的なやり方を守って実装しないと十分な性能を出すことができず、理論演算性能で劣るCPUよりも実効性能が出ない事が殆どである。これは、多くのユーザが抱える問題ではあるが、筆者は、単純であるが故の利点、例えば、使いやすさもあると考える。構造が単純である場合、ハードの機能に依る余計なレイテンシが発生する機会が少なくなり（例えばGPUのL１キャッシュはコヒーレンシを取らない）、加えてモデルの振る舞いを把握しやすく、書き手が操作しやすくなる。モデルの最適化では、性能の阻害要因を特定し、その原因を調査した上で、コードを修正していく。複雑な構造の場合、原因を特定するためのコストが大きくなり、加えて書き手の狙い通りに修正することが難しい。

この欠点と利点のどちらが大きいかはアプリの特性や書き手のGPUに対する習熟度によるが、筆者らは、気象・気候・海洋モデルについては、先行研究や筆者らの事例では良い結果が出ていることから、利点が上回ると考える立場である。

ここで、GPUの実行モデルについて要点を纏める。デバイスであるGPUはホストであるCPUから独立しており、GPUメモリの確保と転送はCPUから行う。GPU上のメモリをCPUから確保した上で、CPUメモリからGPUメモリにデータを転送する（ホストデバイス転送）。GPUカーネルは、転送されてきたデータを更新した後、CPUメモリに転送してGPUでの処理を終える。GPUカーネルでの処理は、CPUではデータが連続する方向にループを回転させることで多数のデータを連続して処理していくが、GPUは１つのデータを１つのスレッドに割り当て、CPUでのループの回転をスレッドの割り当てに置き換える。このように多数のスレッドをGPUが持つ数千の演算コアに、データ並列性を活かすように割り当てていくことで、多数のデータの処理を行う（図１）。

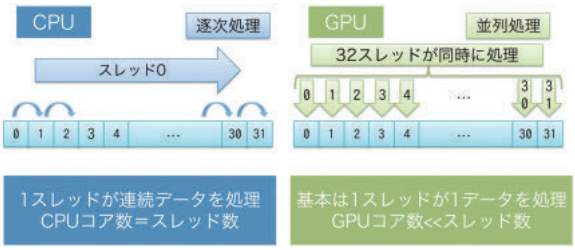

図１　GPUの実行モデル

科学技術計算にてオーバヘッドまたはメモリの読み込みなど「待ち」が発生することは大きな問題であるが、CPUはこの問題をキャッシュとその他の機能（例：SIMD処理、プリフェッチ、分岐予測）で対応する。一方、GPUは多数のスレッドを立て、「待ち」の発生時にはスレッドを切り替えることで対応する。GPUは処理の粒度を極力細かくした大量のデータを扱うことを前提しており、あるスレッドが「待ち」の時は実行の準備が完了した他のスレッドに切り替えて先に実行させてしまう。その実行の裏で「待ち」の処理をさせて、次のスレッドの切り替えに備えさせる。裏を返すと、GPUは高いデータ並列性が無いと「待ち」のレイテンシを隠蔽することができず、無駄なコストの増加につながる。

このように、GPUとCPUは実行モデルが全く違うものである。シリアル処理が多い問題に対しては、CPUが得意とする。それに対し、データ並列性が高い処理を含む問題に対しては、GPUが得意とする。

GPU実装で問題となるのは、CPUとGPUの複合システムであるが故に、物理メモリが２つ存在し、その間で転送が必要なことである。転送を記述することでプログラミングが煩雑になり、レイテンシを含む転送コストはパフォーマンスに直接影響する。現状、GPUに対応する気象・気候・海洋モデルは、初期処理をCPUで行い、メインループ内の演算処理は全てGPUで実施している。また、MPI通信やファイル入出力処理は本質的にCPUのみでしかできないため、ホストデバイス転送は避けられない。これらに対しては、GPUDirect RDMA通信、ストリームを用いた非同期処理などの手法でコストの増加を抑える事が可能である（手法の詳細は［３］を参照）。現状の実装例の殆どでは、気象・気候・海洋モデルのメインループが回転している間は、MPI通信やファイル入出力処理以外ではCPUはアイドル状態にあり、見方を変えるとCPU資源の無駄が生じているといえる。

アプリケーションをGPU対応させる方針として、３つ挙げられる。GPU対応ライブラリへの置き換え、指示行挿入またはCUDAによる直接的な記述である。

ライブラリの利用では、通常のCPUコードで既存のライブラリを利用する場合とほぼ同様に利用できる。この場合、ライブラリのコール時にホストデバイス間の転送を伴うため、その転送コストとGPUによる高速化のどちらの効果が大きいのか正しく評価する必要がある。

指示行挿入によるものでは、仕様の更新・改良の頻度が高く、最も広く使われているものにOpenACCがある［４］。OpenACCはOpenMPと同じく、並列化のオープンな規格であり、複数のメーカによってそれを実装したコンパイラがリリースされている。機能豊富でライブラリ利用よりも実装での自由度が高い。利用方法はOpenMPと同様、並列化するループの前後に指示行を挿入することで、そのループをGPUでの並列化に対応させる。

OpenACC対応コンパイラのオプションにて、指示行の有効・無効を指定できる。これにより、CPU用･GPU用の実行形式を同一のソースコードから作成することができるため、コードの可搬性が高い実装方法といえる。後述するCUDAに比べて容易に実装可能であり、開発・メンテナンスのコストを削減できる。

しかしながら、オープンな規格故にNVIDIA GPUに特化した機能を含んでおらず、ハー

ドが持つ全ての機能を活用できない。例えば、シェアードメモリの活用など、後述するCUDAにてその有効性が示されている手段を取れないという問題点がある。

最も自由度が高い実装方法に、CUDAを用いた記述がある。CUDAはNVIDIA GPU向けの言語拡張で、現在主なものではC言語向けとFortran向けがリリースされている。GPUのスレッドを陽に扱う事ができるため、柔軟なコーディングが可能となり、より深い最適化が可能になる。しかしながら、CPU版のコードからポーティングする場合、コードの大半を書き換えることが必要となるため、CPU･GPU間でのコードの互換性を保つことは難しい。

このように、実装の容易さと性能の出しやすさは相反するものとなっている。これら3つの実装方法はそれぞれ異なる特徴を持つが、実装において排他的では無く、併用した実装が可能である。アプリに応じて選択し、適切に当てはめていくことが求められる。

4．気象・気候・海洋モデルの基礎

4.1　モデルの構造とモデル化

気象・気候・海洋モデルの基本的な構成、モデル化の方法、数値解法などの基礎のうち、GPU向け実装に関係が深い部分を中心にまとめる。気象・気候・海洋モデルは大きく分けて力学過程と物理過程の2つから成る。気象モデルの場合、力学過程は、流体の運動方程式を基本として、連続の式、熱力学の式、水蒸気の式などで構成される。物理過程は、力学以外の要素の総称で、気象モデルの場合、大規模凝結、積雲対流、大気放射計算、エアロゾルや地表面過程などが含まれる。以上から、気象・気候・海洋モデルは、非常に複雑な非線形数値モデルといえる。

力学過程は水平方向の差分化でスペクトル法と格子点法に分類されるが、隣接ノード間での通信が主となる格子点法を採用する気象モデルが増加しており、本稿で紹介する気象モデルは全て格子点法を用いている。格子点法を用いて力学過程を構成する方程式を離散化するにあたり、高次の離散スキームを取ることは少なく、隣接する格子点との間で離散化されるスキームとなることが殆どである。構造格子かつ系統的なメモリアクセスではあるものの隣接格子への3次元方向での参照が多く、加えて物理過程に比べて複雑な演算を含まないことから、メモリバンド幅ネックになりやすい。物理過程は、各格子で計算が閉じるか、鉛直方向のみで依存するものが大半である。力学過程に比べて処理が複雑となり、加えて特殊関数を多く含むなど処理の中で演算が占める割合が多く、演算ネックになりやすい。以上から、気象・気候・海洋モデルは、計算特性が異なるものを含む、計算科学の視点からも複雑な数値モデルといえる。

自然現象全てを微細なプロセスまで全て計算しているわけでは無く、より大きなスケールの変数から経験的に微細なプロセスの役割を見積もる、いわゆるパラメタリゼーション化がなされている。例えば、積雲が発生する水平の空間スケールは数百メートルであるため、数十キロの格子間隔の気象モデルでは表現することができない。その場合、その数十キロの格子を代表する物理量のうち積雲の発生と関係が深いものから、物理的・経験的な手法によって、その格子にて発生する積雲の平均的な値を見積もることで、小さいスケール（積雲）が大きいスケール（数十キロの格子の状態）に与える影響を評価している。この手法は積雲パラメタリゼーション化と呼ばれている。

パラメタリゼーション化の存在は予報にある程度の不確定性を与え得るため、解決策として、計算の空間解像度を上げて直接計算を行い、パラメタリゼーションを外してしまうことが挙げられる。水平解像度を細かくして、積雲の発生を直接計算する全球非静力学

気象モデルの開発も行われてきた［5］。直接計算するか否かでモデルの結果には質的な違いが生じ、それまで表現できなかった物理を捉えることも多く、計算能力の向上・効率化がもたらす気象・気候・海洋モデルの精微化に対する需要は大きい。

気象・気候・海洋モデルは、以上に挙げた全ての要素が相互作用しながら時間発展していく非線形問題である。初期データ（予報変数の初期値、境界条件）を読み込んだ後は、全ての要素を含んだ時間ループを何度も繰り返していく。しかしながら、全ての要素に適切な予報変数を設定し、それらの連立方程式を解くのは計算量が膨大となってしまう。実際のモデル実装では、いくつかの仮定の下で近似が行われ、時間スケールや処理の性質に応じて分割された上で計算される。例えば、力学過程（風速、地表面気圧、雲水など）を計算する際、物理過程（大気放射、エアロゾル、積雲対流、降水や地表面過程など）は更新せずに一定としている。そして、更新した力学の場を元に、物理過程のそれぞれの要素を逐次に更新する。

各要素の更新の順番、組み合わせ、更新の頻度をどのように設定するかは、気象･気候･海洋モデルにとって不確定要素の１つといえる。それは、予報の精度並びに現象の解釈に影響を与えるのみならず、モデルの実行速度にも関係している。それ故、気象学・計算科学の両方の視点から、気象・気候・海洋モデルの構成を検討する必要がある。

4.2　気象・気候・海洋モデルへのGPU適用事例

求められる即時性という社会的な重要性と、構造格子でメモリアクセスが系統的という数値モデルとしての性質の良さから、GPUに対応させる取り組みは早くから行われてきた。しかし、初期の事例では、CUDAによる部分的なポーティングに留まるものであった。メインの時間ループに含まれる処理全てをGPU上で初めて実行したのは、CUDA CによるGPU実装を行った気象モデルASUCAのTSUBAME2. 0での大規模実行の事例である［6］。GPU単体での性能改善やマルチGPUでの通信の隠蔽等により、高い実効効率の元での大規模実行を達成した。CUDA Cは最適化の自由度が高く、高い性能を引き出すことができたが、CPU版コードとの互換性は保たれておらず、開発・メンテナンスにおける効率が良いものでは無かった。

文献［6］で示した事例を含む初期の事例では、CUDAを用いたGPU実装で高い性能を示したものの、研究室または研究室間での試みにとどまり、気象予報等での実運用を前提とした開発ではなかった。近年の開発では、コードのポータビリティ、開発・メンテナンスの工数削減の必要性から、CUDAでの開発から指示行挿入によるOpenACCを中心とした開発が主流となってきた。また、大学・研究機関・企業での共同体制が多くのプロジェクトで確立されている。これは、気象モデルのGPU対応に十分な意義があることの現れといえる。

OpenACCを用いて、大規模システムで多数ノードまで性能をスケールさせた例としては、TSUBAME2. 5でのNICAM力学コアのOpenACC実装［7］の他、スイス連邦工科大学･スイス気象庁によるプロジェクト［8］にて、気象モデルCOSMOを、OpenACCでの実装に加えDSLやライブラリとの併用で、力学過程･物理過程全てをGPUに実装し、数千GPUまで性能がスケールすることを示した［9］。またスイス気象庁ではGPUスパコンであるPiz Kesch 上でGPUを用いた初の気象予報が行われており、GPUが気象予報の実運用に耐えうることが初めて示された。この他、NCAR等の米国の研究機関が中心となって開発してきた領域気象モデルWRFや全球気象モデルMPASでは、NVIDIAを含む

民間企業も参加した国際的なプロジェクトとして、OpenACCを軸としたGPUへの対応が進められている［10］。

5．計算科学としての気象・気候・海洋モデルの課題と今後

ここまでは、気象・気候・海洋モデルのGPUへの適用と高度化について、その背景や基礎を概説した。本章では、それらを踏まえて、課題と今後の対応についてまとめる。

気象・気候・海洋モデルでは、特に力学過程においてメモリアクセスがカーネルの性能を律速する要因となることが多い。一般にGPUではメモリ転送性能の演算性能に対する割合がCPUに比べて低いため（メモリ転送性能の絶対値ではCPUよりも優れているものの）、演算の実行効率という視点からは問題となり得る。また、CPU･GPU共に、近年のスーパーコンピューティングでは、メモリ転送性能の演算性能に対する割合が低くなる傾向にあり､気象モデルのこの性質は性能を向上させる上で大きな問題の一つとなっている。

また、カーネル内に一時変数を持つことも多く、レジスタ消費量も多い。特に物理過程に含まれる経験則に基づく式は、複数のパラメータを含む事が多く、レジスタ使用量が多くなりがちである。レジスタ消費量が大きいと、同時に実行状態となるスレッドの数が減少し、データ並列性を確保できない、また、レジスタスピルが発生してキャッシュ等への変数の一時退避が発生するなど、性能を阻害する要因となる。

ノード間通信は毎時間ステップにおいて複数回発生し、通常のMPIで通信する場合はその都度GPU上のメモリからCPU上のメモリに転送する必要があり、CPU単独での実行に比べて、余計なコストとなり得る。

モデルの実行コスト分布に関しては、コード全体で共通して使われるような処理が少ないため、計算コストが集中するような箇所が存在しないことが特徴である。全球スペクトル気象モデルの性能分析例では、最もコストが集中したサブルーチンは、全体の数％程度のコストでしか無く、大半のサブルーチンは１％未満のコストしか占めなかった。

モデルの開発コストに関しては、その行数が非常に多いこと、複数の研究者による共同での開発であること並びにCPUでの開発が基本であることが特徴である。複数の要素から構成されるため、必然的にコードの分量は大きくなる。また、構成要素（例えば、積雲対流など）それぞれを担当する研究者が、独自の方針でCPUにて数値モデルを作成・検証し、持ち寄ったモデルを結合したものを１つの気象・気候・海洋モデルとして共有している。それ故、開発者全てが、モデルの高度化に詳しいとは限らない。

気象・気候・海洋モデルのGPUでの高度化は複数の課題を抱えており、その課題を解決するには、計算科学の専門家のみでは不可能で、地球科学を専門とする側との協力がいろいろなレベルで必要となる。そのためには、理解しやすく、実装に時間がかからないプログラミング手法が必須といえる。

ここまで、コードの特性からのGPU向け最適化の困難さ、気象モデルコミュニティ故の開発に対する問題点の二つを述べたが、後者についてはOpenACCが唯一の解である。実際、昨今はOpenACCでの指示行挿入での開発がほぼ全ての気象・気候・海洋モデルで行われている。しかしながら、第２章で挙げた「高解像度化」、「高速化」、「精微化」全てに対応するには機能面の制約から限界があると言わざるを得ない。また、OpenACCの元でGPUでの実行を優先して最適化すると、CPU実行での性能が低下しかねない。

今後の取り組みとしてまず筆者が重要と考えることは、OpenACCとCUDAを併用した実装である。加えて将来的な案として、先述した気象モデルの複合的な性質を元に、気象

システムを構成する要素の並列化による高速化を紹介する。

OpenACCでの高度化が難しい箇所はCUDAまたはGPU対応ライブラリを用いる。その際、CPU版コードと併存させることになり、プリプロセッサなどの指定でコンパイル時に切り替えることが良いと考える。その部分については2種類のコードが存在することになり、メンテナンスのコストは2倍となる。気象・気候・海洋モデルは大量のコードを含むが、長い間更新されない、いわゆるレガシー化された部分もあり、そのような部分はCUDA版とCPU版のコードを併用して運用しやすい。性能計測を行いコスト上位からOpenACCとCUDAを併用していくやり方もあるが、コードの更新頻度も併用する部分を選択する指標の一つとなる。先述の通り、CUDAは最適化のスイートスポットが小さいが、幸い気候・気象・海洋モデルは構造格子でかつ系統的なメモリアクセスであるため、最適化の方針は立てやすい。CUDAを用いた先行研究で高い実効効率を出したことがそれを示している。

OpenACCとCUDAを併用した実装の優先度が高いが、将来的な案として、気象システムを構成する要素の並列化による高速化の案を紹介する。先述したとおり、気象モデルは自然現象の内、実際は同時に行われる要素（例えば雲の凝結と大気の放射計算など）をモデル化の段階で逐次処理に設定している。この場合、GPUまたはCPUで逐次に更新していくことになる。複数の要素を、並列に処理する様にモデル化することも、その要素の性質によっては不可能では無い。これに基づき、CPU向けの処理とGPU向けの処理を並行処理できるようなモデル化を検討する。空いているリソース（この場合はCPU）を活用することに加え、CPU向けの処理はCPUで高速に処理させることができれば、モデル全体の高速化につながる。コアの単体性能が高く、キャッシュの活用でレイテンシが大きい問題に対応可能な構造であるCPUと、多数のコアによる多数の並列計算を可能にするGPUという、異なる性格を併せ持つシステムは、気象モデルのような複雑な要素を多く含むモデルには本質的に相性が良いといえる。

今後必要となるOpenACCとCUDAを併用した実装であるが、OpenACCはCPUコード内のループをCUDAで記述されたGPUカーネルに置き換えるものであるため、CUDA実装の理解はCUDAでの実装部分のみならず、OpenACCでの最適化にも役立つものである。そこで、次章では非静力学海洋モデルをCUDAを用いてGPU実装並びに高速化した筆者らの研究を紹介する。

6．CUDAによる非静力学海洋モデルkinacoのGPU実装と高速化

非静力学海洋モデルkinacoのCUDAを用いたGPU実装と高速化の事例を紹介する。kiancoの基本方程式等は［11］を参照されたい。計算のメイン部分は、前処理としてマルチグリッド法を用いたCG法でPoisson方程式を解く部分である。高度化のさらなる詳細については、参考文献［12, 13 , 14］を参照されたい。

CG法前処理への混合精度の適用［12］

気象モデルに含まれる浮動小数点演算は倍精度に設定されることが多い。GPUは、単精度演算のユニットと倍精度演算のユニットは別のハードで構成されていて、チップ当たりの演算性能は単精度演算の方が優れている。メモリ転送性能も、単精度は倍精度に比べて実質2倍の性能に相当する。高速に処理する上で、倍精度から単精度への移行を検討することは十分な意味がある。精度が落ちてしまうため、計算結果への影響は評価する必要があり、適用は慎重に行うべきである。ここでは、運動方程式のソルバである前処理付きCG法の前処理のみに単精度での処理を適

用し、ソルバ本体であるCG法は倍精度のままで実行した。その結果、収束回数は前処理含めて全て倍精度に設定時と同じ回数となり、計算結果の精度も、結果の解釈に影響を及ぼさなかった。

コアレスアクセスの促進［12］

Poisson方程式に今回含まれる行列ベクトル積では、各格子とその周囲6点の参照を係数行列によって表現している。合わせて7点の係数は、各格子固有の値をとる。オリジナルのCPUコードでは、各コア（スレッド）が逐次にメモリをアクセスしていくが、その際に係数7つがキャッシュライン上に連続して並ぶため、キャッシュラインを有効に活用できる。しかしながら、複数のスレッドが同時にアクセスするGPUでは、ストライド幅が7つのアクセスとなってしまう(図2)。そこで、配列要素の次元を並び替え、コアレスアクセスとすることで、メモリへのアクセス効率を向上させた。

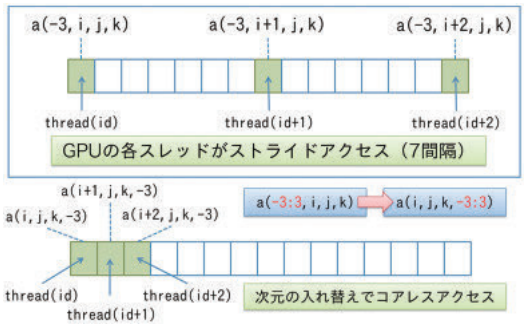

図2　コアレスアクセスの促進

シャッフル関数の活用［14］

CG法の前処理に用いたマルチグリッド法では、格子を粗くする・細かくする操作が複数回実施される。GPUでのこの操作は、各スレッドによる細かい格子からのロードが非コアレスアクセスとなってしまう（図3）。キャッシュの活用により、2回目以降のアクセスでは最適化されるものの、本件ではさらなる高速化を目指し、ウォープシャッフル関数を活用した。まず、コアレスアクセスでロードし、レジスタに保存する。シャッフル関数を用いて、他スレッドのレジスタに保存された値を自スレッドにロードして足し込み、粗い格子を作成した。

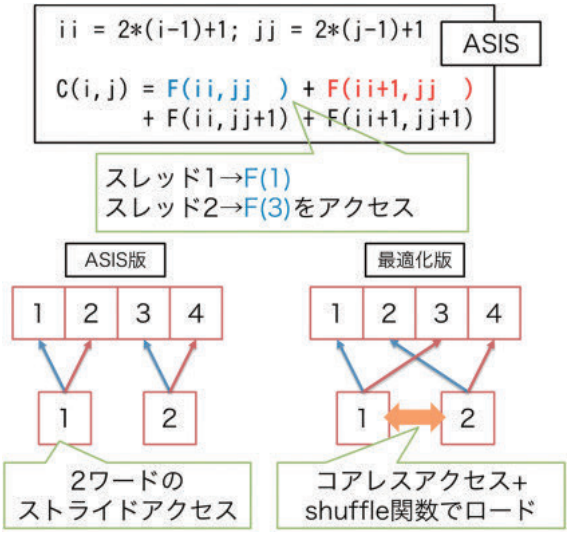

図3　粗い格子を作成するアルゴリズム

ループ融合とレジスタの活用によるデータ並列性の確保［13］

物理過程に含まれるカーネルにおいて、CPUコードではある3次元の物理量を計算し、それに対して連続にアクセスさせて別の物理量を計算していた。この計算方法は、最初に計算した3次元の物理量をキャッシュを介して次の連続アクセスで再利用することが目的であったが、GPUの場合はレジスタを活用することで、キャッシュよりもより高速にデータを再利用できる(図4)。前述の2つ

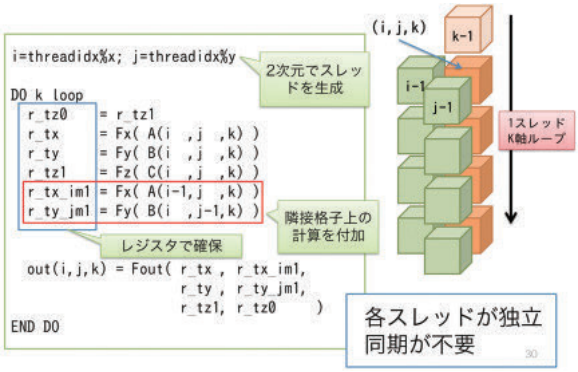

図4　ループ融合とレジスタの活用によるデータ並列性の確保

の処理を融合し、レジスタを介して計算を各スレッドで閉じる。同時にループ間の暗黙の同期が削除されるため、２次元スレッドが独立に動き、データ並列性を活用できる。

ブロック形状変更によるキャッシュブロック化の促進［14］

kinacoの格子形状の仕様から、Poisson方程式の係数が鉛直方向にてほぼ一様（海面付近、海底付近を除く）という特性がある。CPUコードでは、同一の値となる係数は、同じメモリアドレスに格納してキャッシュの利用効率を上げる最適化がなされていた（図５）。この最適化はGPUのL1キャッシュでも有効であるが、キャッシュヒット率を上げるために、ブロックの形状を鉛直方向の要素数が大きくなるように確保した。同じブロックが処理を終了するまでストリーミングマルチプロセッサから解放されない特性を活用している。評価例を挙げると、[32, 8 , 1] のブロック形状に対して、[32, 1 , 16] はおよそ２倍高速である。

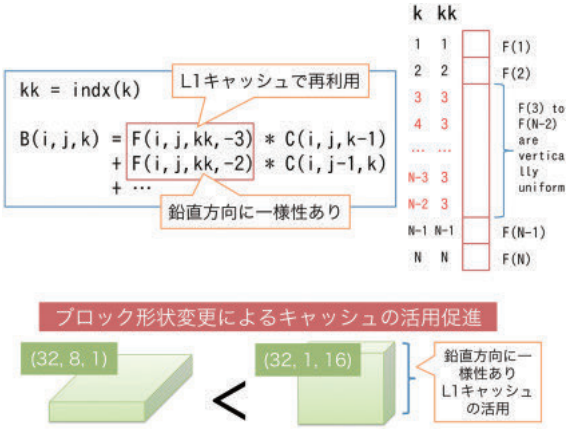

図５　ブロック形状変更によるキャッシュブロック化の促進

メモリアライメント調整（ノード間通信用袖領域の削除）［14］

各ノードでの予報変数は、A（１：n1, 1：n2, 1：n3）としてn1*n2*n3個の要素を有する場合、袖領域は各次元で２個、つまりA（−1：n1＋２，−１：n2＋２，−1：n3＋２）としてメモリを確保している。モデルの仕様上、n1らは２の階乗となる場合が殆どであり、その場合、メモリへのアクセスでアライメントが適切なものにならず、どの次元においてもトランザクションが最低で一つ無駄になる。筆者らは、オリジナルのCPUコードではMPI通信時のオーバヘッド回避のために、転送データ（袖領域）を別途用意した１次元配列に詰め替えた後に通信を行っていることに着目し、この１次元配列をGPUカーネルから直接参照させて予報変数には袖領域を用意しないことにした（図６）。これにより、メモリアクセスで無駄なトランザクション無くなることに加え、受信した１次元配列から予報変数への詰め替えのコストも削減出来た。

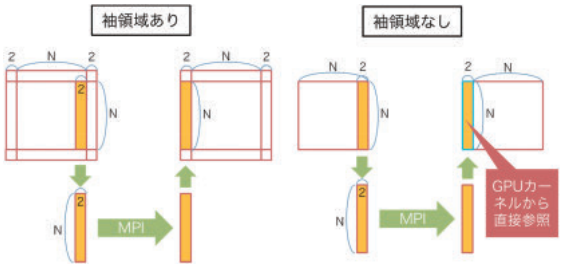

図６　袖領域を持たない隣接通信

コストが小さいカーネルの融合［14］

マルチグリッド法を用いた前処理では、粗い格子（例えば、[８, ８], [４, ４] 程度）での演算が複数回実施される。その中に含まれる行列ベクトル積のカーネルは、代表的なコストは経過時間にして数マイクロ秒程度であるが、カーネルの起動に伴うオーバヘッドは20マイクロ秒程度と、カーネル本体の計算に比べて無視できない大きさである。反復法であるが故に、呼び出し回数はモデル本体のステップ数の100倍以上と、非常に大きい。そこで、カーネルを融合することで、起動に伴うオーバヘッドを削減した（図７）。オリジナルのCPUコードでは、隣接間通信の前後に複数のカーネルが組まれていたが、通信以外の

カーネルを全て融合した。余計なロードストアが減ったことと、カーネル終了時にグリッド全体でとられていた同期が、ブロック内での同期となったことによるコスト減少の効果も含まれる。

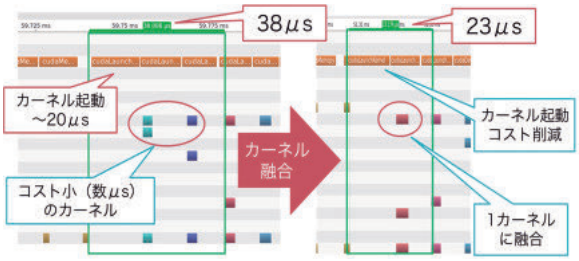

図7　コストが小さいカーネルの融合

粒子追跡［13］

算出した流速分布から海中の粒子（栄養塩や放射性物質など）分布を決める粒子追跡計算にて、CPU版は粒子のデータ構造に連結リストを使用していた。連結リストでは粒子の追加や消滅などを表現する際の扱いが容易という利点はあるが、時間が経過するにつれて近傍の粒子がメモリアドレス上で散逸していき、GPUで求められるコアレスアクセスに比べて効率が低くなる。そこで、連結リストから通常の配列に移行した。前後の粒子を指し示すポインタを排除し、粒子位置に応じて粒子を配列内でソートすることで、粒子をコアレスにアクセスさせるようにした。ソートのコストは生じるものの、メモリアクセス最適化による高速化及び連結リストでのポインタの計算が不要になった効果の方が大きい。

GPUDirect RDMA［14］

異なるノード間にてGPU同士で通信する場合、GPUDirectを用いることで、ネットワークのチップをGPUから直接経由させて、途中CPUにデータが達すること無く、受信するGPUがネットワークのチップからデータを直接受け取る事ができる。この直接通信はソフト・ハード共に仕様の要求要件があるので、詳細は［1，2］等を参照して欲しい。

我々は、kinacoに含まれる袖通信を全てGPUDirectにて実装することで、CPUで通信した場合とほぼ同等の通信速度となる事を確認した。

7．まとめ

気象・気候・海洋モデルのGPU対応について、GPU並びに気象・気候・海洋モデルの基礎を含めて今後の課題をまとめた。気象・気候・海洋モデルのコード特性並びに開発者・利用者の背景から、今後の気象・気候・海洋モデルの開発は、OpenACCとCUDAを併用した実装が最適と判断した。将来の並列化案として、気象システム内の要素を並列化させる事も検討していく。OpenACCとCUDAの併用で、気象・気候・海洋モデルに求められる条件（「高解像度化」、「高速化」、「精微化」）を満たすにはCUDAの知識と経験が必須である。我々は、これまで行ったCUDAによる非静力学海洋モデルのGPU実装をさらに推し進め、気象・気候・海洋モデルを始めとする数値モデルのGPU実装と高度化に役立つ知見を蓄積し、次世代のエクサスケール・スーパーコンピュータにて成果を出していくことを目指している。

参考文献

1．CUDA C Programming Guide, NVIDIA
2．CUDA C Best Practice Guide, NVIDIA
3．John Cheng, Max Grossman, Ty McKercher,CUDA C プロフェッショナルプログラミング，インプレス
4．OpenACC, https://www.openacc.org/
5．Satoh, M., et al. (2014). The Non-hydrostatic Icosahedral Atmospheric Model: description and development. Progress in Earth and Planetary Science 1(1): 18.
6．Shimokawabe, T., et al. (2010). An 80 Fold Speedup, 15.0 TFlops Full GPU

Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code. Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society: 1-11.

7. Yashiro et al. (2015), A Simulation of Global Atmosphere Model NICAM on TSUBAME2.5 Using OpenACC, GTC 2015, San Jose.
8. Consortium for Small-scale Modeling, http://www.cosmo-model.org/
9. Fuhrer, O., et al. (2018). Near-global climate simulation at 1km resolution: establishing a performance baseline on 4888GPUs with COSMO 5.0. Geosci. Model Dev. 11(4): 1665-1681.
10. Posey S. and Adie J. (2019), Sunny Skies Ahead! Versioning GPU accelerated WRF to 3.7.1, GTC 2019, San Jose.
11. Matsumura, Y. and Hasumi, H (2008). A non-hydrostatic ocean model with a scalable multigrid Poisson solver. Ocean Modelling 24(1-2): 15-28.
12. Yamagishi, T. and Matsumura, Y. (2016), GPU Acceleration of a Non-hydrostatic Ocean Model with a Multigrid Poisson/Helmholtz Solver, Procedia Computer Science, 80, 1658-1669.
13. Yamagishi, T. and Matsumura, Y. (2016). GPU Acceleration of a Non-Hydrostatic Ocean Model with Lagrangian Particle Tracking. Supercomputing Conference 2016.
14. Yamagishi, T., et al. (2018), An MPI-CUDA Acceleration for a Non-Hydrostatic Ocean Model with GPUDirect RDMA Transfers, GTC Japan 2018.