jh140013-NA09

# マルチ GPU コンピューティング・フレームワークを用いた高精度気象計算コードの開発

下川辺　隆史（東京工業大学）

概要　気象計算はスパコンで実行される重要なアプリケーションの一つであり、GPU スパコンで高速化されてきている。GPU スパコンで効率的に実行するためには、複雑な最適化手法を導入する必要がある。本研究では、これらをアプリケーションに簡単に導入することが可能な GPU コンピューティング・フレームワークを開発した。フレームワークは、ユーザが記述した格子計算のコアとなる格子点を更新する関数から最適化された GPU 実行コードを生成し、アプリケーション全体を簡単に並列化し、実装の複雑な通信を隠蔽するオーバーラップ手法等をアプリケーションへ簡便に導入することができる。これを用いて次期気象予報のために気象庁で開発されている非静力気象モデル ASUCA を GPU スパコンへ実装した。これを、東京工業大学の GPU スパコン TSUBAME2.5 で実行し、高い生産性・可搬性を実現しながら、高い実行性能を達成することに成功した。

## 1.　共同研究に関する情報

### (1) 共同研究を実施した拠点名

東京工業大学

### (2) 共同研究分野

■　超大規模数値計算系応用分野

□　超大規模データ処理系応用分野

□　超大容量ネットワーク技術分野

□　超大規模情報システム関連研究分野

### (3) 参加研究者の役割分担

- 下川辺 隆史（東京工業大学 学術国際情報センター）：研究全体の統括、格子計算用フレームワークの開発、これを用いた GPU 版 ASUCA の開発と計算の実施
- 丸山 直也（理化学研究所 計算科学研究機構）：格子計算用フレームワークへの GPU 向け最適化手法の検討と導入
- 青木 尊之（東京工業大学 学術国際情報センター）：Kepler コアを搭載した K20X GPU へチューニング
- 小野寺 直幸（東京工業大学 学術国際情報センター）：乱流モデルおよび高精度な移流計算手法の開発と導入
- 室井 ちあし(気象庁)：気象庁開発の ASUCA に関する情報提供と高精度計算手法の検討
- 河野 耕平(気象庁 数値予報課)：気象庁開発の ASUCA に関する情報提供と高精度計算手法の検討

## 2.　研究の目的と意義

### 2.1 研究の目的

著者らは気象庁が次期気象予報に向けて開発を進めている次世代気象シミュレーションコード ASUCA 全体を GPU で実行する研究に取り組んできた。GPU は元々は画像処理用のプロセッサであったが、ここ数年、これを汎用計算に適用する GPGPU 研究が盛んに行われている。GPU は従来のプロセッサである CPU と比べて消費電力当たりの演算性能が高いため、ペタスケールのスパコンでは GPU を大規模に搭載してきている。

本研究では、気象庁での開発が進み ASUCA がほぼ完成状態に至ったので、 完全版 ASUCA をフル GPU 化し GPU スパコンで実行し、気象計算精度が高精細格子を用いることで格段に向上することを実証する。現在の気象庁では日本列島全土を水平解像度 2km で気象計算している。これに対し、本研究では GPU のアーキテクチャに合った最適化手法を導入し、東京工業大学の TSUBAME2.5 の 1,000 個

以上の GPU を用いることで約 8,000×8,000×100 という計算格子で水平解像度 300m 以下という高解像度の計算を目指す。300m の水平解像度に必要となる乱流モデルや高い空間解像度を活かした高精度な移流計算を検討し、導入して、最終的に気象予測としての精度向上を追求する。

実装には開発中の格子計算用のフレームワークを用い、格子に基づいたアプリケーションの大規模GPU化技術を確立する。大規模GPU計算では、高性能を達成するためノード間通信の隠蔽などの最適化手法の導入が必須であるが、その導入コストは非常に高い。著者らは複数 GPU 計算に必須な最適化手法を簡便に導入できる格子計算用のフレームワークを開発中で、これを完成させ ASUCA に用いることで、高性能な実アプリケーションを高生産に開発する GPU 利用技術を確立する。

### 2.2 研究の意義

気象分野では GPU 計算の導入は高い注目を集めている。米国大気研究所を中心に開発されている次世代大気シミュレーションコード WRF では、GPU を用いた高速化の取り組みがいち早く行われてきた。しかし WRF では、コードのごく一部分を GPU 化できたのみで、全体の性能を約 30% 向上させるに留まっている。著者らは WRF と異なり ASUCA 全体を GPU 上で実行するフル GPU 化に取り組んできた。ASUCA の既存のソースを全て書き直し、GPU 計算に合った最適化を考案し、スパコン分野で最高峰の国際会議 SC10 で論文として発表し、気象計算において GPU が極めて有用であることを世界で初めて示した。本研究はこれまでの研究を発展させるもので、TSUBAME で高速かつ高精細な格子を用い気象予測精度が向上すれば、気象および高性能計算分野で特色のある研究となり大きなリードとなる。

ペタスケールのスパコンでは、低消費電力に抑えながら高性能演算能力を達成するため数千台を超える GPU が搭載され、日本、米国、中国などで稼働している。気象計算はスパコンを利用する代表的なアプリケーションであり、気象庁の次期運用で用いられる気象計算コード ASUCA の完成版を GPU スパコンで高性能に動作させる意義は大きく、世界での注目は間違いない。

本研究の影響は気象分野にとどまらない。ここ数年、GPU の高い演算能力から様々な分野で GPGPU 研究が盛んに行われている。 GPU は消費電力あたりの演算性能が高く、今後は大規模アプリケーションでは GPU を代表としたメニーコアプロセッサの利用が性能面、消費電力面、コスト面から必須である。しかし、その利用の難しさ、生産性や保守性の低さが導入の障壁となっている。本研究が完成すると、GPU 化を簡便に行えるフレームワークが完成し、これを用いることで高い生産性で気象庁の行う気象計算が GPU スパコンで行えることを実証でき、実アプリケーションで GPU を利用する成功事例になると確信している。

## 3. 当拠点公募型共同研究として実施した意義

本研究では、大規模 GPU 計算によって高精度な気象計算を実現する。東京工業大学のスパコン TSUBAME2.5 は世界的にみても大規模に GPU を搭載した GPU スパコンであり、本研究を遂行する上で最適な計算機環境である。

本研究は、大規模アプリケーションの専門家、計算機科学の専門家、気象予測計算の専門家による共同研究によって進められている。丸山（理研）が計算機科学の知識を提供し、青木（東工大）が K20X GPU のチューニング手法を開発し、下川辺（東工大）がこれらを取り入れた GPU 用フレームワークを構築した。オリジナルの気象計算コード ASUCA を開発した室井、河野(気象庁)の助言のもと、下川辺がフレームワークを用い GPU 版の ASUCA を実装した。小野寺（東工大）の開発する乱流モデルと数値計算手法の導入を検討した。このように効果的に共同研究を遂行することにより、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

該当なし

## 5. 今年度の研究成果の詳細

格子計算用の GPU コンピューティング・フレームワークを開発し、気象庁が開発を進める気象計算コード ASUCA をこのフレームワークを用い GPU スパコン上へ実装し、大規模高性能計算を実行した。これらの成果をスパコンの国際会議 SC14 に投稿し、採択され、2014 年 11 月に発表した（Takashi Shimokawabe, Takayuki Aoki, and Naoyuki Onodera, SC14, 2014, pp. 1-11, 研究業績[1]）。以下では、具体的な研究成果について説明する。

### 5.1 GPU コンピューティング・フレームワーク

これまでの気象計算 ASUCA の GPU 化では、GPU 上で性能を出すために配列の次元を変更し、GPU アーキテクチャに向けた最適化手法を導入した。また、大規模計算では、通信コストを隠蔽する通信と計算のオーバーラップ手法などの計算手法を導入した。

本研究では、これらの最適化手法を簡便に導入し、高い生産性を実現するフレームワークを開発し、これを用いて気象計算 ASUCA を実装した。本フレームワークは、直交格子型の解析を対象とし、各格子点上で定義される物理変数（プログラム上は配列となる）の時間変化を計算する。また、当該物理変数の時間ステップ更新は陽的であり、ステンシル計算によって行われる。TSUBAME に導入されている NVIDIA 社製 GPU で実行することを目指し、実装には、ホストコードは C/C++言語、デバイスコードは CUDA を用いる。

フレームワーク設計における主な目標を述べる。

- プログラマは格子点上での計算についてのみ記述する。格子全体の処理はフレームワークが行う。格子全体の処理がユーザコードからフレームワークへ分離され GPU に合った最適化手法を隠蔽する。また、バックエンドとして様々なプロセッサを採用することができ、拡張性と高い生産性を持つ。現在、フレームワークでは、GPU および CPU で実行可能である。
- フレームワークが提供するテンプレートを用いた C++クラスを用い格子点上の計算を記述する。言語拡張や標準的でないプログラミングモデルを利用すると、既存コードからの乖離も大きく利用しにくい。また、拡張部分がフレームワークを他の環境へ移植する妨げになりうる。その点、C++のテンプレートやクラスは広く使われており、基盤とできる。
- 複数ノードでの GPU 計算に対応するため、ノード間およびノード内通信を簡便に記述するクラスを提供する。このクラスを用いることで、プログラマは MPI や OpenMP を明示的に使用した通信を記述すること必要がなくなる。

以上まとめると、ユーザは、(1)フレームワークの提供するテンプレート関数およびクラスを用い、ステンシル計算を行う関数を記述する。(2)フレームワークの提供するクラスを用いて、GPU 間通信を記述する。以下で詳細について記述する。

#### 5.1.1 ステンシル計算関数の定義

本フレームワークでは、ステンシル計算は、フレームワークの提供する `ArrayIndex3D` 等を用い、C++ファンクタとして定義し、ステンシル計算関数と呼ぶ。 3 次元の拡散計算では、次のように関数を定義できる。

```
struct Diffusion3d {
__host__ __device__
void operator()(const ArrayIndex3D &idx,
       float ce, float cw, float cn, float cs,
       float ct, float cb, float cc,
       const float *f, float *fn) {
fn[idx.ix()] =
  cc*f[idx.ix()]
+ ce*f[idx.ix<1,0,0>()] + cw*f[idx.ix<-1,0,0>()]
+ cn*f[idx.ix<0,1,0>()] + cs*f[idx.ix<0,-1,0>()]
+ ct*f[idx.ix<0,0,1>()] + cb*f[idx.ix<0,0,-1>()];
}
};
```

`ArrayIndex3D` は、対象とする配列のサイズ $(n_x, n_y, n_z)$ を保持し、ある特定の格子点を表すインデックス $(i, j, k)$ を設定できる。対象とする配列が `f` であるとき、`idx` を `ArrayIndex3D` のオブジェクトとすると `f[idx.ix()]` として使われ、これは配列 `f` の $(i, j, k)$ 点の値を返す。`ArrayIndex3D` はテンプレートを用いたメンバ関

数が定義されており、例えば、idx.ix<+1,0,0>()、idx.ix<-1,-2,0>()とすると、(*i*+1, *j*, *k*)、(*i*-1, *j*-2, *k*)を表すインデックスを返す。テンプレートを用いることで、GPU および CPU でのインデックス計算の高速化を図っている。

ステンシル計算関数の第一引数は固定で、計算対象となる格子のインデックス情報を持つ idx を受け取らなければならない。関数実行時には、格子点(*i*, *j*, *k*)の値が設定されているため、(*i*, *j*, *k*)を中心としたステンシル計算を関数内に記述する。f, fn は配列へのポインタであり、これに対しステンシルアクセスすることとなる。

### 5.1.2 ステンシル計算関数の実行

本フレームワークは、全ての格子点に計算を行う Loop3D 等のクラスを提供する。これを用い、ステンシル計算関数の実行は以下のように行う。

```
Loop3D loop3d(nx+2*mgnx, mgnx, mgnx,
              ny+2*mgny, mgny, mgny,
              nz+2*mgnz, mgnz, mgnz);
loop3d.run(Diffusion3d(), ce, cw, cn, cs,
           ct, cb, cc, f, fn);
```

Loop3D はステンシル関数を適用する範囲指定するパラメータで初期化する。Loop3D::run()は任意個の異なる型を引数にとるテンプレート関数として定義されている。C++のテンプレート関数の型推論を利用し、Loop3D::run()は、与えられた全ての引数を第二引数以降に持つファンクタ Diffusion3d()を呼び出す。ファンクタは、NVIDIA CUDA の__host__、__device__で定義することができ、ホスト、デバイス両方へコンパイルされており、Loop3D::run()が CPU 上のデータに対しても、GPU 上のデータに対しても同じ関数を実行する。Loop3D 内部の実装としては、CPU 上で実行する場合はファンクタを全格子点に対して for 文で実行し、GPU 上で実行する場合は内部で生成される CUDA のグローバル関数に包み、適当な CUDA の block 数、thread 数が渡され全格子点に対して実行される。第二引数以降で初めて出てくるポインタが GPU メモリの配列を指すか CPU メモリの配列を指すかを判定し、GPU で実行するか CPU で実行するかを自動的に決定する。

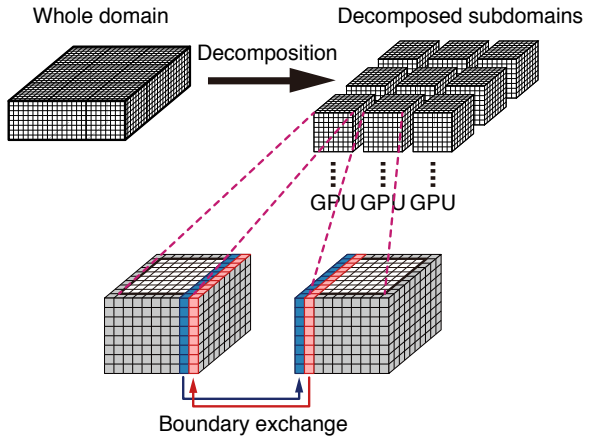


図 1 複数 GPU による格子に基づいた計算

### 5.1.3 変数の GPU 間通信

複数 GPU による格子を用いた計算では、図 1 に示す領域分割法を用いる。領域分割法では、隣接領域間の境界領域の交換が必要であり、配列として確保された変数は隣接 GPU から送信されたデータを格納する境界領域を持つ。本フレームワークでは、この隣接領域間の境界領域を交換するための GPU 間通信を簡単に記述するためのクラス BoundaryExchange を提供する。複数 GPU における計算を効率的に行うため、フレームワークは、ノード内並列では OpenMP で並列化し、GPU 間の通信を効率的に行うため GPUDirect による peer-to-peer 通信を用いる。一方、ノード間並列では MPI を利用する。MPI によるノード間通信では、転送のための一時領域をホストメモリに確保している。

クラス BoundaryExchange を用いて、以下のように GPU 間通信を行う。

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(array1);
exchange->append(array2);
exchange->append(array3);
exchange->transfer();
```

domain はクラス Domain のオブジェクトで、計算領域サイズ、隣接ノード番号等を保持している。クラス BoundaryExchange は、Domain を参照す

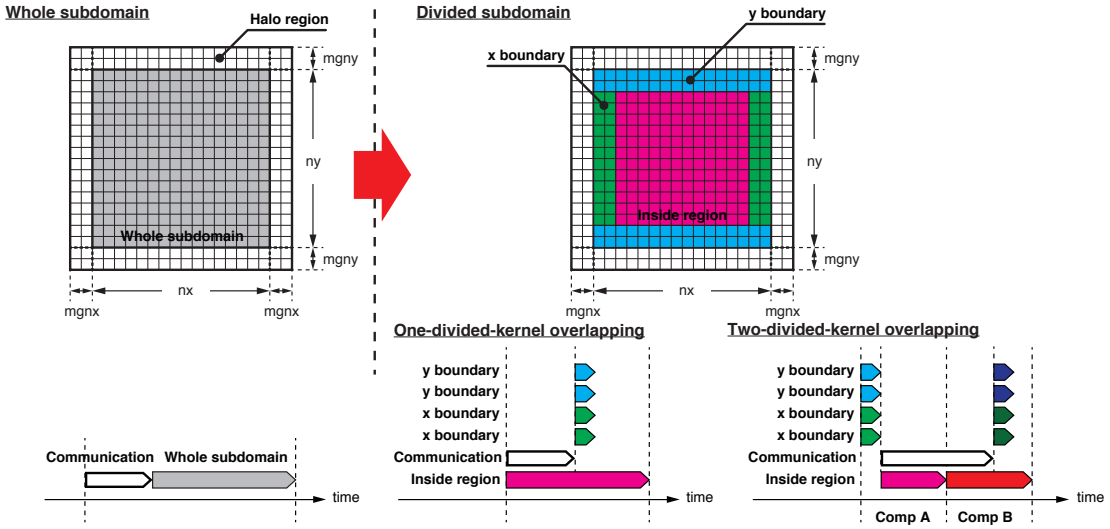


図 2　非オーバーラップ手法とカーネル分割による計算と通信のオーバーラップ手法。1 つのカーネル分割によるオーバーラップ手法、2 つのカーネル分割によるオーバーラップ手法を提供する。

ることで計算領域サイズ等を取得している。これらの情報を用いることで、`exchange->append()`で登録された変数 `array1`, `array2`, `array3` に関して `exchange->transfer()`で境界領域を転送する。この関数内で、OpenMP, MPI を用いた転送が行われる。

### 5.1.4 通信と計算のオーバーラップ手法

大規模計算では、GPU 間の通信時間は全実行時間に対して無視できない。通信コストを計算で隠蔽するオーバーラップ手法の導入を大規模計算における性能向上にとって重要となる。

本フレームワークは、カーネル分割によるオーバーラップ手法を提供する。この手法は、ある物理変数内でのデータの非依存性を利用する。ある変数のそれぞれの要素は他の要素の計算と独立に計算できるため、1 つの GPU が担当する計算領域を境界領域と残りの中心領域に分割して、独立に計算することが可能である。図 2 に一つのカーネルを分割し通信をオーバーラップする one-divided-kernel オーバーラップ手法を示す。このオーバーラップ手法では、中心領域の計算を開始し、それと同時に境界領域の計算に必要なデータを取得するための通信を行う。通信が終了した後、境界領域の計算を行う。これによって、通信の隠蔽を行う。

しばしば一つのカーネルのみ分割する one-divided-kernel オーバーラップ手法だけでは通信コストを完全に隠蔽できない。ASUCA のような実アプリケーションでは、ある変数を用いた計算後、通信により境界領域を交換し、その後、その通信した変数を用いた計算を行うという計算手順がしばしば使われる。本フレームワークでは、この計算手順を活用し、ある通信の前後の二つのカーネルを分割し、その通信をオーバーラップする two-divided-kernel オーバーラップ手法も提供する。図 2 にこのオーバーラップ手法も示す。

本フレームワークでは、ユーザコードにカーネル分割によるオーバーラップ手法を適用するために、クラス `CompCommBinder` を提供する。これを用い、拡散計算では、次のようにオーバーラップ手法を記述できる。

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(f);
CompCommBinder<Loop3D> ccbinder(exchange);
ccbinder.set_post_func(&loop,
   create_funcholder<Loop3D>(Diffusion3d(),
      ce, cw, cn, cs ,ct, cb, cc, f, fn));
   ccbinder.set_use_overlapping();
   ccbinder.run();
```

`CompCommBinder` は、`Loop3D` による計算範囲とステンシル関数および GPU 間通信を制御する `BoundaryExchange` オブジェクトによって初期化される。`CompCommBinder` は、`Loop3D` を境界領域と中心領域に対応する複数の `Loop3D` へ分割し、

これらの Loop3D で CompCommBinder::set_post_func で指定されたステンシル関数を実行する。BoundaryExchange による通信を適当なタイミングで行い、この通信を隠蔽する。以上が one-divided-kernel オーバーラップ手法である。もう一つステンシル関数を CompCommBinder::set_pre_func で指定すると、フレームワークは二つのステンシル関数へ自動的に two-divided-kernel オーバーラップ手法を適用する。

### 5.2 フレームワークを用いた気象計算 ASUCA の性能評価

本研究では、気象計算 ASUCA を本フレームワークを用い実装した。計算領域を 2 次元分割し東京工業大学の TSUBAME2.5 に搭載された NVIDIA Tesla K20X GPU を複数用い、性能測定を行う。

図 3 に強スケーリングの結果を示す。オーバーラップ手法を用いた場合、用いない場合の性能を示す。参考として 1 つの GPU を 1MPI プロセスで計算する Flat-MPI の結果も合わせて示す。計算格子 1536×1280×60 および計算格子 3072×2560×60 を単精度計算した。TSUBAME の各ノードには 3GPU 搭載されているが、この計算ではこのうち GPUDirect で通信可能な 2GPU を用いている。オーバーラップ手法を用いたことによる性能向上がみられ、計算格子 3072×2560×60 の 512GPU 計算では、18.9 TFlops を達成した。

オーバーラップ手法による性能向上の効果を分析するため、オーバーラップ手法を用いた場合と用いない場合について、二つの演算について計算時間と通信時間の内訳を調べる。図 4 に速度の座標変換の計算時間とそれに付随する通信時間を示す。非オーバーラップ手法では、計算と通信の順に逐次に実行される。x 方向の通信の一つで GPUDirect による peer-to-peer 通信を利用している。図に示すように、この通信は、その他の通信と比較して通信時間が短く、通信性能の向上に貢献している。この計算では、オーバーラップ手法として one-divided-kernel オーバーラップ手法が用いられる。GPU 計算では、しばしば多数のスレッドを用いると性能が向上する。このため、複数のカーネルに分割され小さい領域を複数に分けて計算するオーバーラップ手法ではカーネルの計算性能自体は低下する。非オーバーラップ手法では単一カーネルによる計算は 723.9 $\mu$ sec であるのに対し、オーバーラップ手法では複数に分割されたカーネルにより全体で 1209.3 $\mu$ sec かかる。オーバーラップ手法

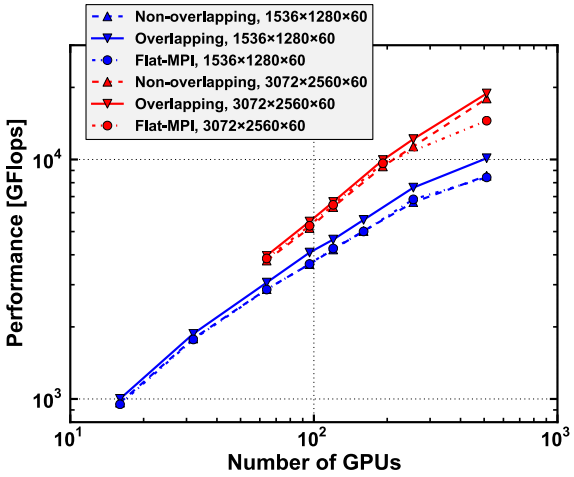


図 3　ASUCA の実行性能（強スケーリング）

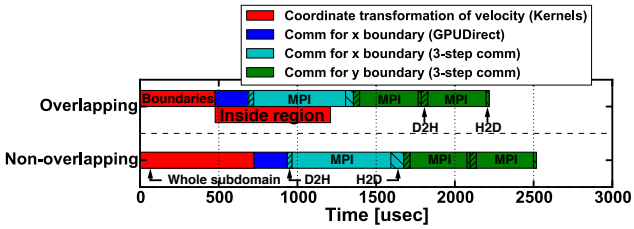


図 4　速度の座標変換の計算時間とそれに付随する通信時間の内訳。one-divided-kernel オーバーラップ手法が使われている。

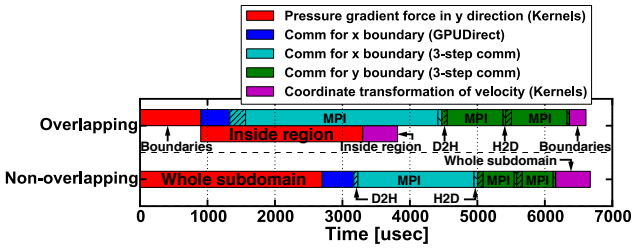


図 5　圧力勾配による力と速度の座標変換の計算時間とその間に実行される通信時間の内訳。two-divided-kernel オーバーラップ手法が使われている。

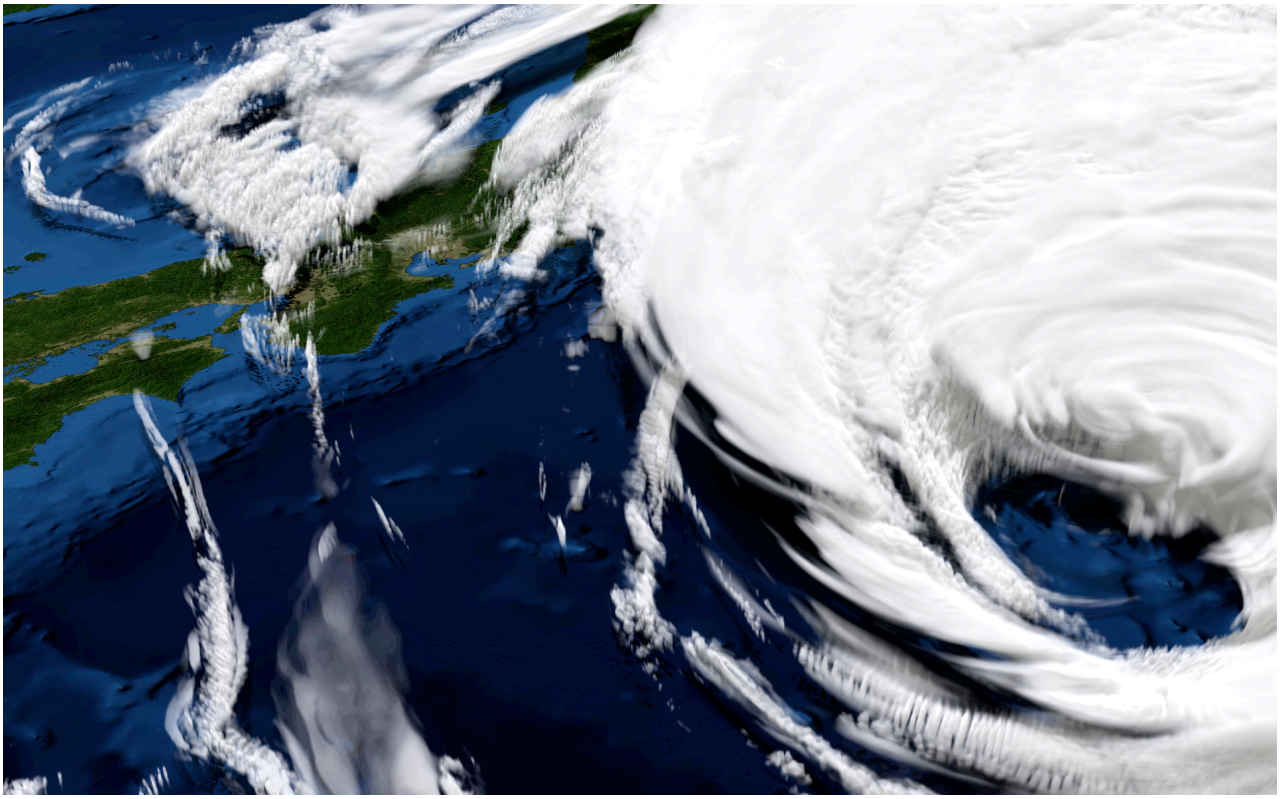

図 7　TSUBAME2.5 の 672GPU を用いた台風の気象計算の例。計算格子 5,376×4,800×57 を用い水平解像度 500m で計算している。

を適用することで計算時間自体は長くなったが、中心領域の計算が一部の通信（732.0 $\mu$ sec）を隠蔽できるため、全体として性能が向上している。

図 5 に圧力勾配による力と速度の座標変換の計算時間とそれに付随する通信時間の内訳を示す。この通信は二つの計算の間に実行されるため、この一連の計算に two-divided-kernel オーバーラップ手法を適用する。図 4 と同様に、オーバーラップ手法で用いられた複数の分割されたカーネル全体では非オーバーラップ手法で用いられた単一カーネルよりも実行時間は長いが、オーバーラップ手法では一部の通信（2916.7 $\mu$ sec）が隠蔽されているため、一連の計算全体では性能向上している。

図 6 に弱スケーリングの結果を示す。表 1 にこのときに用いた GPU 数と格子サイズを示す。1 GPU あたり計算格子 768×128×60 を用い単精度計算した。強スケーリングの測定と同様、オーバーラップ手法と用いた場合と用いない場合の測定結果を示す。強スケーリングの測定とは異なり、最大の実行性能を得るため、各ノード

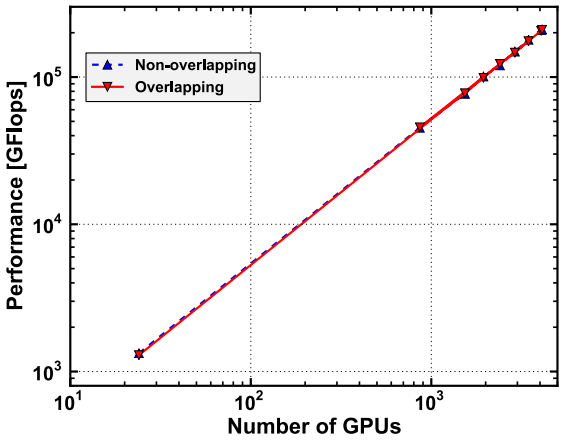


図 6　ASUCA の実行性能（弱スケーリング）

表 1　ASUCA の弱スケーリング測定に用いた GPU 数と格子サイズ

| Number of GPUs ($P_x \times P_y$) | Mesh size ($nx \times ny \times nz$) |
|---:|---:|
| 24 (2 × 12) | 1536 × 1536 × 60 |
| 864 (12 × 72) | 9216 × 9216 × 60 |
| 1536 (16 × 96) | 12288 × 12288 × 60 |
| 1944 (18 × 108) | 13824 × 13824 × 60 |
| 2400 (20 × 120) | 15360 × 15360 × 60 |
| 2904 (22 × 132) | 16896 × 16896 × 60 |
| 3456 (24 × 144) | 18432 × 18432 × 60 |
| 4056 (26 × 156) | 19968 × 19968 × 60 |
| 4108 (26 × 158) | 19968 × 20224 × 60 |

に搭載された 3GPU 全てを使用した。オーバーラップ手法を用いることで性能向上がみられ、4,108 GPU で 209.6 TFlops を達成した。

図 7 に現在の数値予報で使用されている初期値データと境界値データを用いた台風の気象計算を行った例を示す。TSUBAME2.5 の 672 GPU を用い計算格子 5,376×4,800×57（実際の格子間隔は水平 500m）で計算している。

## 6. 今年度の進捗状況と今後の展望

本課題では、開発中であった格子計算用のフレームワークを完成させ、これを用い GPU 版 ASUCA を完成させ、東京工業大学のスパコン TSUBAME2.5 で大規模実行に成功した。

フレームワークは、ユーザが記述した格子計算のコアとなる格子点を更新する関数から最適化された GPU 実行コードを生成し、アプリケーション全体を簡単に並列化し、アプリケーションへ実装の複雑な通信を隠蔽するオーバーラップ手法等を簡便に導入することができる。GPU コードの生成には、NVIDIA 社の CUDA を用い、Kepler コアの GPU に対して最適化を行った。

開発したフレームワークを用い、気象庁が開発を進める気象計算コード ASUCA を GPU 上へ実装した。この GPU 版 ASUCA を東京工業大学のスパコン TSUBAME2.5 で大規模実行し、単精度計算で 4,108 GPU を用い 209.6 TFlops を達成した。気象庁が現在の数値予報で使用している初期値データと境界値データを用い、672 GPU による水平解像度 500m という高精細格子を用いた精度の高い気象計算に成功した。

この研究成果をスパコンの国際会議 SC14 にて発表した（研究業績[1]）。本課題では、研究計画通り、フレームワークの構築とこれを用いた ASUCA の開発を通して、GPU アプリケーションを高生産に開発する方法を確立した。また、高精細格子による ASUCA の大規模計算に成功した。高精細格子による高い空間解像度を十分に活かすことができる乱流モデルや高精度な移流計算手法について個々に検討したものの、ASUCA 本体への導入を完了するに至らなかったため、引き続き導入を進めていく。

今後は、本課題で開発したフレームワークを基盤として、様々なアーキテクチャで高速に実行可能な GPU/CPU 両対応した高性能・高生産フレームワークへ発展させる。これを用い、次世代スパコンに向いたアルゴリズムである格子ボルツマン法による都市気流計算コードを高生産に開発する。これらの開発を通して、GPU スパコンおよび CPU スパコン上で、高生産・高性能・大規模な実アプリケーションの開発技術の確立を目指す。

## 7. 研究成果リスト

### (1) 学術論文

なし

### (2) 国際会議プロシーディングス

[1] Takashi Shimokawabe, Takayuki Aoki, and Naoyuki Onodera, "High-productivity Framework on GPU-rich Supercomputers for Operational Weather Prediction Code ASUCA," in Proceedings of the 2014 ACM/IEEE conference on Supercomputing (SC'14), New Orleans, LA, USA, Nov 2014, pp. 1-11.

### (3) 国際会議発表

[2] Takashi Shimokawabe, "A High-Productivity Framework for Multi-GPU Computing of Weather Prediction Code ASUCA," GTC 2015, San Jose, CA, USA, Mar 2015. (Poster, GTC Poster Award finalist)

[3] Takashi Shimokawabe, "High-productivity Framework on GPU-rich Supercomputers for Weather Prediction Code," CREST-ECRC workshop, King Abdullah University of Science and Technology, Saudi Arabia, Sep. 2014.

### (4) 国内会議発表

[4] 下川辺隆史, 青木尊之, 小野寺直幸, "GPU スパコンによる格子に基づいたシミュレーションのための GPU コンピューティング・フレームワーク", AXIES 大学 ICT 推進協議会 2014 年度年次大会, 仙台, 2014 年 12 月.

［5］下川辺隆史，青木尊之，小野寺直幸，″ステンシル計算のための GPU コンピューティング・フレームワークのチューニング高度化″，第 205 回 ARC・第 147 回 HPC 合同研究発表会（HOKKE-22），小樽，2014 年 12 月.

[6] Takashi Shimokawabe, “A High-productivity Framework on GPU-rich Supercomputers for Mesh-based applications,” Workshop on HPC and Cloud Accelerators, 神戸、2014 年 8 月.

［7］下川辺隆史，青木尊之，小野寺直幸，″GPU コンピューティング・フレームワークを用いた気象計算コードの開発″，日本計算工学会 第 19 回計算工学講演会論文集，広島国際会議場，広島，2014 年 6 月.

**(5) その他（特許，プレス発表，著書等）**

なし