jh150047-NA25

大規模 GPU/CPU 計算に向けた高生産フレームワークの構築とこれを用いた都市気流計算コードの開発

下川辺　隆史（東京工業大学）

概要　様々なアーキテクチャで高速に実行可能にする GPU/CPU 両対応した高性能・高生産フレームワークを構築し、これを用いることで従来よりも高性能な都市気流計算コードを高生産に開発する。GPU スパコンおよび CPU スパコン上での高生産・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。本課題では、開発中のフレームワークに、単一のユーザコードを GPU および CPU などの様々なアーキテクチャで高速実行する機構などを導入した。元々のフレームワークは GPU 計算に対して高速化を行っていたが、CPU 計算において OpenMP による並列計算が可能となる。また、GPU 計算ではオーバーラップ手法、および、実行時に自動チューニングする機構を導入し、さらなる高度化に成功した。これを用い拡散方程式および都市気流計算コードを実装し、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、GPU/CPU 両対応の格子計算用フレームワークおよびフレームワークベースの都市気流計算コード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 および CPU 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）：数値計算手法に関する助言と最適化手法の検討
- 小野寺 直幸（海上技術安全研究所）：都市気流計算コードに関する助言とそれに対するフレームワーク適用の検討

2.　研究の目的と意義

2.1 研究の目的

著者らは気象計算など格子に基づいたアプリケーションを 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、従来のプロセッサの CPU と比べて計算の処理能力が非常に高く、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いる。

GPU によるステンシル計算は高い性能が得られるものの、それにはアーキテクチャに適した高度な最適化を導入する必要がある。前年度の研究では、著者らは、通常の C++を記述するだけで、GPU スパコンに必須な最適化をアプリケーションへ簡便に導入できる構造格子用 GPU フレームワークを提案した。これを用い、ユーザコードに GPU 固有の最適化を記述せず、通信の隠蔽などが導入された GPU スパコンに最適化された気象計算コードの開発に成功し、スパコン分野で最高峰の国際会議 SC14 で発表した。これによって、フレームワーク技術は GPU 実アプリケーションを高生産に開発する上で有効であることを示した。

今年度は、前年度に開発したフレームワークを拡張し、GPU だけでなく CPU など (1) 様々なアー

キテクチャで高速に実行可能にする GPU/CPU 両対応した高性能・高生産フレームワークを構築する。大規模 GPU アプリケーションは、開発コストが高いだけでなく、CPU では動作せず、生産性や保守性が低い。今年度は、フレームワークを拡張し、機種固有のコードの差異を吸収し、単一の計算コードから様々な GPU/CPU で高速実行できるコードを生成できるようにする。さらに、アーキテクチャに適した最適化、自動チューニング機構を導入し、高度化する。開発したフレームワークが様々なアプリケーションへ適用できることを実証するため、今年度は、前年度の気象計算と数値計算手法が異なるアプリケーションであり、B/F 値の小さい次世代の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って実装された大規模な都市気流計算コードへフレームワークを適用する。これによって、(2) フレームワークを用いることで従来よりも高性能な都市気流計算コードを高生産に開発できることを示す。前年度と今年度を通して、フレームワークを用いて複数の実アプリケーションを開発することで、(3) GPU スパコンおよび CPU スパコン上での高生産・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。

2.2 研究の意義

ペタスケールのスパコンでは、低消費電力かつ高性能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。気象計算や都市気流計算などの格子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、今年度の都市気流計算では、高層ビルが立ち並び複雑な構造をした都市部で詳細な気流を解析するため高解像度格子を用いて広範囲に計算する必要がある。著者らは、これまでに TSUBAME の 4032 台の GPU を用い、皇居などを含む東京都心部の 10km 四方のエリアを 1m 格子で計算することに成功した。

このように都市気流計算を始めとした格子計算では GPU が極めて有用であるが、一方で、高い性能を得るためには、それぞれのアーキテクチャに適した高度な最適化を導入する必要がある。また、一般に GPU コードは CPU コードとは独立に開発され、生産性や保守性が低い。本研究で開発する GPU/CPU 両対応したフレームワークは、これらを解決する有効な方法であり、本課題は GPU スパコンおよび CPU スパコン上での高生産・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ものである。

次世代スパコンでは、GPU などのメニーコアプロセッサの利用が性能、消費電力面から必須であるが、高性能を得るには、ますます高度な機種固有の最適化を導入する必要がある。フレームワークは、機種固有の高度な最適化を隠蔽しながら、それをアプリケーションへ導入できる技術である。本研究が完成すると、CPU スパコン、GPU スパコン、さらには次世代スパコンでもユーザコードを変更せずに一貫して動作するアプリケーションを高生産に開発する基盤技術になり、波及範囲は広い。

3 当拠点公募型共同研究として実施した意義

本研究は、GPU スパコンおよび CPU スパコン上で、高生産に高性能な実アプリケーションを開発するためのフレームワークを開発し、高性能な都市気流計算コードを高生産に開発し、大規模計算を実現する。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 および CPU を搭載したスパコンであり、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、フレームワーク構築にも重点をおく。有用なフレームワークの構築には、高度な計算機科学の知見に合わせて、様々な実アプリケーション開発の経験が問われる。本研究は、大規模アプリケーションを専門とする下川辺（東工大）が中心となり、計算機科学を専門とする丸山(理研)、流体計算を専門とする青木(東工大)、小野寺(海技研)が共同研究を行うことで、アプリケーション開発者の立場から実用に耐えうるフレームワークを構築する。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

本課題では、前年度までに格子計算用の GPU コンピューティング・フレームワークの基本機能を開発した。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。前年度は、これを用い気象庁が開発する気象計算コード ASUCA を GPU スパコンへ実装した。ユーザコードに GPU 固有の最適化を記述せずに、GPU スパコンに最適化されたコードを開発した。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し、国際会議 SC14 で発表した。

5　今年度の研究成果の詳細

本研究課題では、前年度の研究で開発した構造格子用 GPU フレームワークを拡張し、単一のユーザコードを GPU および CPU などの様々なアーキテクチャで高速実行する機構などを導入した高生産フレームワークを完成させた。元々のフレームワークは GPU 計算に対して高速化を行っていたが、今年度は CPU 計算において OpenMP による並列計算に対応した。また、GPU 計算では計算条件によって自動チューニングする機構を導入し、さらなる高度化に成功した。これを用い、拡散方程式および都市気流計算コードを実装した。以下では、具体的な研究成果について説明する。

5.1 ステンシル計算用フレームワークの概要

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。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。また複数 CPU および GPU 計算に対応する。プログラマは格子点上での計算についてのみ記述し、格子全体の処理はフレームワークが行う。ユーザはフレームワークを用いることで、ユーザアプリケーションにフレームワークの提供する最適化手法を簡便に適用することが可能となる。

5.2 直交格子データ構造の導入

生産性をさらに向上させるため、今年度は、本フレームワークに独自データ型 ETArray を導入する。ETArray は配列データに加えて、その大きさと位置を保持した構造である。これを用い、以下のように配列データを確保することができる。

```
// 直交格子のデータの大きさ、位置
unsigned int length[] = {nx+2*mgnx, ny+2*mgny, nz+2*mgnz};
int          begin [] = {-mgnx, -mgny, -mgnz};
Range3D whole(length, begin); // 大きさ、位置を表す

// ホスト上、デバイス上に直交格子用データを作成する
ETArray<float, Range3D> f_h(whole, MemoryType::HOST_MEMORY);
ETArray<float, Range3D> f_d(whole, MemoryType::DEVICE_MEMORY);
```

Range3D は大きさと位置の情報を持つ範囲を指定する補助クラスである。配列データはホストメモリおよび GPU のデバイスメモリ上に確保することができる。

5.3 ステンシル計算関数の定義と実行

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。5.2 で述べた今年度導入したデータ構造 ETArray を用いることで、これまでよりもより効率的な記述を実現する。3 次元の拡散計算では、次のようにステンシル計算関数を定義できる。

```
struct Diffusion3d { // ユーザ定義のステンシル関数（例は拡散方程式）
__host__ __device__
  float operator()(const float *f, const ArrayIndex &idx,
    float ce, float cw, float cn, float cs, float ct,
    float cb, float cc) {
    const float fn = + 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)];
    return fn; // 戻り値は、ある一点の更新する値
  }
};
```

ArrayIndex は、対象とする格子サイズ(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、その点および隣接点へアクセ

スする関数を提供する。例えば、idx.ix()、idx.ix(-1,-2,0) は それぞれ (*i, j, k*)、 (*i*-1, *j*-2, *k*) を表すインデックスを返す。

ステンシル計算関数は次のように実行する。

```
ETArray<float, Range3D>
   f(whole, MemoryType::DEVICE_MEMORY);
ETArray<float, Range3D>
   fn(whole, MemoryType::DEVICE_MEMORY);
Range3D inside; // inside は格子点のうち、更新したい領域

...

view(fn,inside)
   = funcf<float>(Diffusion3d(), ptr(f), idx(f),
                  ce, cw, cn, cs, ct, cb, cc);
// カーネル関数の実行
// ptr は ETArray の raw ポインタ、
// idx はインデックス表現 ArrayIndex を渡す
```

f、fn は ETArray データである。funcf()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、funcf()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 CUDA の __host__、__device__で定義でき、ホスト、デバイス両方へ対応する。 ptr()は、ステンシル計算関数中で ETArray のある格子点 (*i, j, k*) のポインタを取得できる。view()は、fn の inside 領域内の格子点の値へ右辺の式を代入する。ファンクタは,CPU 上では対象の格子点に対して for 文内で、 GPU 上では CUDA のグローバル関数に包み実行される。新たに導入した ETArray データ型によって view()によりカーネル関数が実行でき、より効率的にカーネル関数の実行を記述できる。

5.4 ETArray による配列の全要素への計算

配列の全要素への計算は、ステンシル・アプリケーションでしばしば見られる。本フレームワークでは ETArray データ型によって、配列の全要素への計算機能を提供する。

3 次元の拡散方程式の初期化を次のように記述できる。

表 1　フレームワークが提供する ETProperty

ETProperty	提供機能
HostProperty	ホストでの実行
OmpProperty	OpenMP で実行
DeviceProperty	デバイス (GPU) で実行
DeviceATProperty	GPU で自動チューニングを用いて実行
DeviceOverlapProperty	GPU でオーバーラップ手法を用いて実行
DeviceOverlapATProperty	GPU でオーバーラップ手法と 自動チューニングを用いて実行

```
int main() {
   ...
   f = 0.125 * (1.0 - cos(k[0] * dx*(pos(axis0) + 0.5)))
             * (1.0 - cos(k[1] * dy*(pos(axis1) + 0.5)))
             * (1.0 - cos(k[2] * dz*(pos(axis2) + 0.5)));
   // 左辺の f は ETArray。pos により位置情報が渡される。
   ...
}
```

f は ETArray データである。式テンプレートと呼ばれる C++技術を利用し、式から GPU および CPU に対応したカーネル関数を生成する。式の適用範囲は左辺 f の範囲とし、明示的な範囲指定は不要とし、コードのインライン化も可能となり、見通しがよく簡便な記述を実現する。

5.5 ステンシル計算関数の実行環境の拡張

本フレームワークは、これまで GPU 計算と単一コアによる CPU 計算に対応していたが、OpenMP を用いた CPU による並列計算やユーザ定義の実行方法を指定できるよう拡張する。

これを実現するため、ステンシル計算関数の実行時の情報を保持および記録する ETProperty クラスを新たに提供する。5.3 で述べた view()はこれに対応し、次のように利用する。

```
ETProperty *prop = new DeviceATProperty;
view (fn, inside, prop)
   = funcf<float>(Diffusion3d(), ptr(f), idx(f),
                  ce, cw, cn, cs, ct, cb, cc);
```

本フレームワークは、表 1 に示した ETProperty クラスを提供する。これらにより OpenMP による実行、通信を計算で隠蔽するオーバーラップ手法、自動チューニング機構を提供する。ETProperty を用いたオーバーラップ手法、自動チューニング機構の詳細については 5.6、5.7 で述べる。

5.6 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、複数 GPU 計算に対して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 1 にオーバーラップ手法の計算方法を示す。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。

本フレームワークは、ETArray に対応したオーバーラップ手法を提供する。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。以前のフレームワークと比べ、実現する最適化は同等であるものの、記述がより簡便となる。

```
PBoundaryExchange
   exchange(inside, rank, neighbor_connect);
DeviceOverlapProperty *prop
   = new DeviceOverlapProperty(Range3D::ndim);

vieweb(fn, &exchange, prop)
   = funcf<float>(Diffusion3d(), ptr(f), idx(f),
                  ce, cw, cn, cs, ct, cb, cc);
```

vieweb()は、境界領域のデータ転送を制御する PBoundaryExchange オブジェクトに対応した view()で、右辺の式を境界領域と中心領域に分割し実行する。PBoundaryExchange 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。

5.7 自動チューニング機構

フレームワークでは、CUDA のブロックを二次元に確保し、xy 平面に平行に配置しステンシル計算関数を実行する。CUDA のブロック内の各スレッドを(i, j)に割り当て、z 方向にマーチングしながら順に計算を行う。一般に、計算領域に割り当てるスレッド数を多くすると GPU では性能が向上しやすいため、 z 方向にいくつかの領域に分割し、CUDA のブロックを割り当てる。

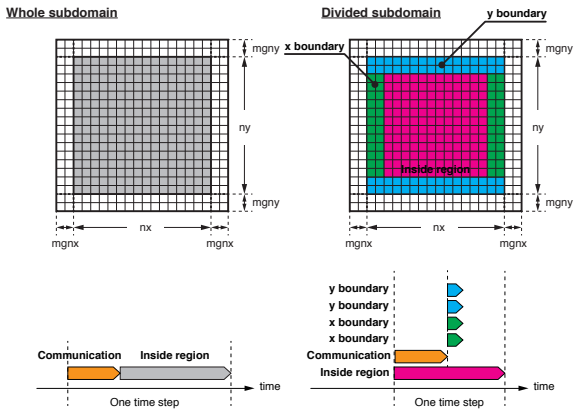

図 1　カーネル分割による計算と通信のオーバーラップ手法

GPU によるステンシル計算では、その性能はこれらのスレッド数や分割するなどの実行時パラメータに大きく依存する。本フレームワークでは、CUDA のブロック内の x と y 方向のスレッド数と z 方向にマーチングする格子数をチューニングするパラメータとする自動チューニング機構を新たに提供する。ユーザは表 1 の DeviceATProperty あるいは DeviceOverlapATProperty を用いることで利用できる。表 2 に選択され得る値を示す。

DeviceATProperty を引数にとる view()によってステンシル計算関数が実行されると、実行毎に DeviceATProperty によって新しい自動チューニング・パラメータが指定され、その実行時間が計測され、DeviceATProperty のオブジェクトへ記録される。DeviceATProperty は全てのパラメータで計測を完了すると、それ以降のステンシル関数の実行では、実行時間を最短とする最適なパラメータを用いる。自動チューニングは、プログラムの実行中に行われる。チューニング中は性能低下が見られるが、一般的に格子を用いた計算では、ステンシル関数は数万回を超え実行されるため、チューニング・パラメータ決定のための性能低下は

表 2　自動チューニングされるパラメータ

Number of threads in x direction	4, 8, 16, 32, 64, 128
Number of threads in y direction	1, 2, 4, 8, 16
Number of mesh in z-marching	1, 2, 4, 8, 16

アプリケーション全体の実行時間と比較すると無視することができる。`DeviceOverlapATProperty` は `DeviceATProperty` の機能に加え、オーバーラップ手法を用いステンシル計算関数を実行する。

5.8 その他の機能

実アプリケーションを開発する上で、生産性の向上は重要である。本フレームワークでは、実アプリケーションである都市気流計算を開発する上で必要とした下記の機能を追加している。

(1) `ETArray` の持つ要素のうち最大、最小、総和などのリダクション計算を行う関数群。

(2) `ETArray` 型のデータを入出力する関数群。`ETArray` は配列データに加えて、その大きさと位置を保持している。この構造を利用することで任意の計算格子で計算を再開することが容易となっている。

5.9 性能評価

本フレームワークの提供する機能の性能を検証するため、本課題では、流体計算の基礎的な方程式である拡散方程式と、実アプリケーションである都市気流計算コードを本フレームワークで実装し、性能評価する。

性能測定は、東京工業大学の GPU スパコン TSUBAME2.5 を用いる。TSUBAME2.5 は 4000 基を超える NVIDIA Tesla K20X GPU が搭載されている。Tesla K20X は単精度計算で 3.95 TFlops のピーク性能を持つ。 TSUBAME2.5 の 1 ノードには Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core が 2 ソケット、Tesla K20X が 3 基搭載されている。計算は全て単精度で行う。

5.9.1 拡散方程式

まず流体計算の基礎的な方程式である拡散方程式を本フレームワークで実装し、性能評価する。

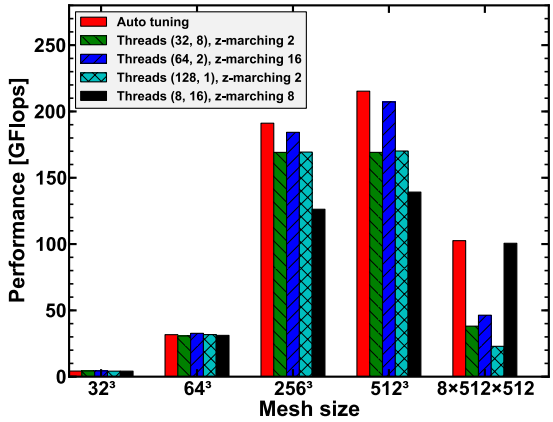

図 2 単一 GPU における自動チューニング機能を用いた拡散計算の実行性能

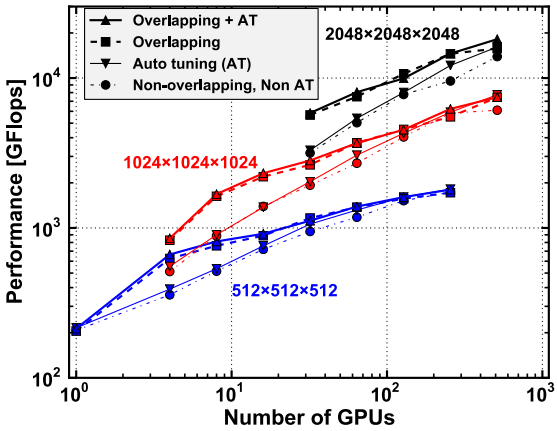

図 3　オーバーラップ手法、自動チューニング、それを併用した複数 GPU 計算と最適化なし複数 GPU 計算による拡散方程式の強スケーリング

図 2 は単一 GPU で自動チューニング機能を用いた場合と代表的なスレッド数、分割数に固定した場合の実行性能を示す。図に示すように、自動チューニングを取り入れた計算は、全ての格子サイズの領域に対して高速な実行速度を示し、512^3 の計算格子では 215.3 GFlops を達成した。(62, 2) のスレッドと z 方向のマーチング格子点数に 16 を指定して実行すると多くの計算格子サイズの条件で良い実行性能が達成できるものの、8×512×512 では性能が低下し、46.4 GFlops となる。これに対して、自動チューニングを用いると 102.6 GFlops を達成した。

図 3 は複数 GPU 計算の強スケーリングを示す。

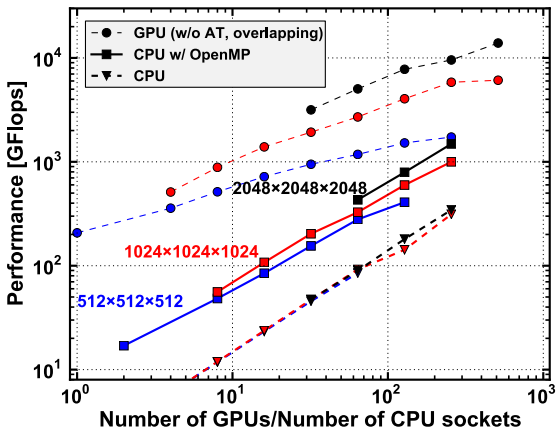

図 4　複数 CPU 計算（1 ノードあたり 1 コア）、OpenMP により並列化された複数 CPU 計算、複数 GPU 計算（最適化なし）の拡散方程式の強スケーリングの比較

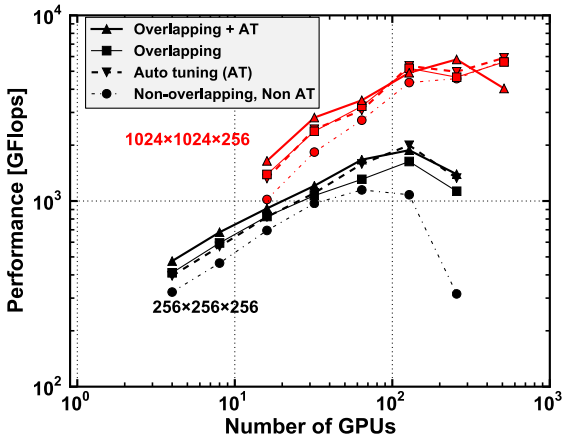

図 5　オーバーラップ手法、自動チューニング、それを併用した複数 GPU 計算と最適化なし複数 GPU 計算による都市気流計算の強スケーリング

オーバーラップ手法、自動チューニング、それらの併用した場合およびそれらを利用しなかった場合の実行性能を比較する。1 ノードあたり 3GPU を使用する。自動チューニングを入れることで、入れない場合と比較し、性能が向上していることがわかる。また、オーバーラップ手法により通信コストが隠蔽されることで、大幅な性能向上を達成している。64GPU を用いた 1024^3 の計算格子では、オーバーラップ手法と自動チューニングの併用、オーバーラップ手法のみ、自動チューニングのみ、それらを利用しなかった場合で、それぞれ 3.78 TFlops、3.70 TFlops、3.06 TFlops、2.70 TFlops となる。

図 4 は 1 ノードあたり 1CPU コア、1 ノードあたり 2CPU ソケット（12 コア）を使用した計算の実行性能を示す。1 ノードあたり 2CPU ソケット利用する場合は、ノード内を OpenMP で並列化する。比較として図 3 に示した 1 ノードあたり 3GPU を使用した実行性能（最適化なし）を合わせて示す。計算には全て同一コードを使用しており、GPU 計算に加えて、CPU 計算において OpenMP を用いることで高速化されることがわかる。64CPU ソケットを用いた 1024^3 の計算格子では、シングルコアで計算した場合、OpenMP を用いた場合で 、それぞれ 92.0 GFlops、327 GFlops を達成した。

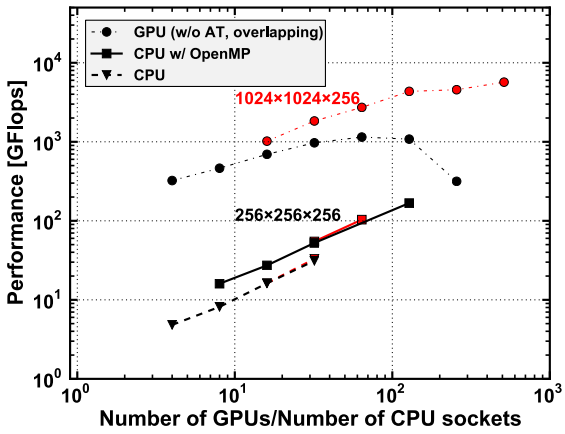

図 6　複数 CPU 計算（1 ノードあたり 1 コア）、OpenMP により並列化された複数 CPU 計算、複数 GPU 計算（最適化なし）の都市気流計算の強スケーリングの比較

5.9.2 都市気流計算による性能評価

次に実アプリケーションである都市気流計算コードを本フレームワークで実装し、性能評価する。

都市気流計算コードは、小野寺ら(HPCS2013, 2013)によって開発された東京の 10km 四方の領域を対象に、実際の建物データを使用して 1m 間隔の格子解像度で気流シミュレーションを行い、ビル風などを再現するアプリケーションである。計算手法として格子ボルツマン法が用いられている。拡散方程式とは異なる計算手法であるが、本フレームワークを格子ボルツマン法へ適用することに成功した。都市気流計算コードの開発を通して、

5.8 に示した機能を追加している。

図 5 は複数 GPU 計算の強スケーリングを示す。図 3 と同じくオーバーラップ手法、自動チューニング、それらの併用した場合およびそれらを利用しなかった場合の実行性能を比較する。図が示すように、本フレームワークは実アプリケーションである都市気流計算でも有効であり、オーバーラップ手法、自動チューニングをそれぞれ単独に適用することによって性能が向上していることが確認できる。さらに、オーバーラップ手法と自動チューニングを併用することで大幅に性能が向上していることがわかる。64GPU を用いた 1024×1024×256 の計算格子では、オーバーラップ手法と自動チューニングの併用、オーバーラップ手法のみ、自動チューニングのみ、それらを利用しなかった場合で、それぞれ 3.49 TFlops、3.24 TFlops、2.90 TFlops、2.72TFlops となる。なお、フレームワークを用いた都市気流計算の実装では、2 つの手法を併用することで単一 GPU あたり最大で 129 GFlops を達成しており、これは、高嵜ら(HPCS2015, 2015)によって報告された小野寺らの開発による従来の最適化されていないナイーブに実装された参照コードで達成した単一 GPU あたり 75.3GFlops を超えるもので、フレームワークを用いた実装は高い性能を達成している。

図 6 は図 4 と同じく都市気流計算において 1 ノードあたり 1CPU コア、1 ノードあたり 2CPU ソケット(12 コア)を使用した計算の実行性能を示す。実アプリケーションにおいても、同一コードで GPU 計算に加えて、単一コアによる CPU 計算および OpenMP による並列 CPU 計算が可能である。32CPU ソケットを用いた 1024×1024×256 の計算格子では、シングルコアで計算した場合、OpenMP を用いた場合で 、それぞれ 32.9 GFlops、54.8 GFlops を達成した。

6 今年度の進捗状況と今後の展望

本課題では、前年度の研究で開発した構造格子用 GPU フレームワークを拡張し、単一のユーザコードを GPU および CPU などの様々なアーキテクチャで高速実行する機構などを導入した高生産フレームワークを完成させた。既存のフレームワークは、C++のテンプレート機能を利用し、通常の C++で記述されたステンシル計算コードを NVIDIA 社の GPU で高速に実行するコードを生成できる。このフレームワークを拡張し、今年度は CPU 計算において OpenMP による並列計算に対応した。また、GPU 計算では計算条件によって自動チューニングする機構を導入し、さらなる高度化に成功した。また、実アプリケーションの開発では生産性が高いことが重要である。本課題では、独自の型である `ETArray` を追加することで、配列型の容易な確保、配列の全要素への計算、リダクション計算、簡便なデータ入出力を実現した。

本課題では、開発したフレームワークを拡散方程式および都市気流計算コードへフレームワークを適用した。都市気流計算コードは前年度の気象計算コードとは全体構造や計算手法が全く異なるため、新たなフレームワークの実アプリケーションへの適用例となり、その生産性や最適化手法の有効性を検証することができた。都市気流計算コードへの適用を通して、それに必要としたリダクション計算、簡便なデータ入出力を実現する機能を追加し、フレームワーク全体の完成度を高めることに成功した。前年度および今年度の課題を通して、開発したフレームワークが大規模計算に適用でき高い性能を得られることを示した。しかしながら、本課題で対象とした都市気流計算に対しては、TSUBAME に搭載された全 GPU や CPU を用いて計算を実行する機会を得るに至らなかったため、引き続き研究を進めて行く。

大規模 GPU 計算が可能となり、近年は広大な計算領域の場所によって求められる精度が異なる問題に有効な手法が要求されてきている。GPU 計算では、GPU が得意なステンシル計算を活用しながら、高精度が必要な領域を局所的に高精細にできる適合細分化格子法（Adaptive Mesh Refinement; AMR 法）が有効である。今後は、本課題で開発したフレームワークを基盤として、局所的に高精細となる計算を実現する GPU スパコン用の AMR 法を

様々なアプリケーションへ適用可能とする AMR 法フレームワークを構築する。AMR 法では、様々な解像度の格子で多数のステンシル計算を行うため、本課題の成果を活かすことができる。AMR 法フレームワークを構築することで、CPU スパコン、GPU スパコン、さらには次世代スパコンでもユーザコードを変更せずに一貫して動作する高精細計算を実現する AMR 法を導入したアプリケーションを高生産に開発する基盤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 for Large-scale GPU/CPU Stencil Applications," IHPCES/ICCS 2016, San Diego, USA, June 2016.（採択済）

(3) 国際会議発表

[2] Takashi Shimokawabe, "Large-scale GPU-based CFD Applications based on a High-productivity Stencil Framework," Parallel CFD 2016, 神戸, May. 2016. (招待講演)

[3] Takashi Shimokawabe, "Advanced High-Productivity Framework for Large-Scale GPU/CPU Stencil Computations," GTC 2016, San Jose, CA, USA, April 2016. (GTC Poster Award finalist, ポスター)

[4] Takashi Shimokawabe, Takayuki Aoki and Naoyuki Onodera, “Advanced High-productivity Framework for Stencil Applications on GPU Supercomputers,” The 3rd International Workshops on Advances in Computational Mechanics, Tokyo, Japan, Oct. 2015.

(4) 国内会議発表

[5] Takashi Shimokawabe, ″High-resolution Weather Prediction Code based on High-productivity Framework for Multi-GPU computation,″ 2nd Annual Meeting on Advanced Computing System and Infrastructure (ACSI2016), 福岡, 2016 年 1 月.

[6] 下川辺隆史, 青木尊之, 小野寺直幸, “自動チューニング機構の導入によるステンシル計算のための GPU コンピューティング・フレームワークの高度化”, 日本応用数理学会 2015 年 年会, 金沢, 2015 年 9 月.

[7] 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高生産 GPU コンピューティング・フレームワークの高度化”, 日本計算工学会第 20 回計算工学講演会論文集, つくば国際会議場, つくば, 2015 年 6 月.

(5) その他（特許, プレス発表, 著書等）

なし