jh160051-NAH

高精細計算を実現する AMR 法フレームワークの構築

下川辺　隆史（東京大学）

概要　格子に基づいたシミュレーションでは、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（AMR）法が有効である。本研究では、様々な解像度のステンシル計算を高性能に実行する手法や GPU 間の計算負荷を均等とする動的負荷分散を実現する GPU スパコン用の AMR 法を確立し、これを汎用的に利用できるフレームワークを構築する。本課題では、前年度までに開発した構造格子用のフレームワークを発展させ、AMR 法に必要なデータ構造や計算機構を開発し、単一 GPU 用の AMR 法フレームワークを構築した。AMR 法フレームワークを用い、AMR 法を導入した移流計算および、実装と計算がより複雑なレイリーテイラー不安定性シミュレーションに成功した。

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、適合細分化格子法（AMR 法）に対応したフレームワークの構築と実アプリケーションへの適用
- 高木 知弘（京都工芸繊維大学 機械工学系）：フェーズフィールド法のコードの提供と AMR 法導入に関する検討
- 青木 尊之（東京工業大学 学術国際情報センター）：AMR 法に対応した数値計算手法に関する助言と最適化手法の検討
- 丸山 直也（理化学研究所 計算科学研究機構）：AMR 法フレームワークへの GPU および CPU 向け最適化手法の検討
- 小野寺 直幸（海上技術安全研究所）：格子ボルツマン法のコードの提供と助言

2.　研究の目的と意義

2.1 研究の目的

申請者らは気象計算や金属材料の凝固成長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、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。前年度までの研究では、これを簡便に導入し生産性を高めるため、申請者らは、通常のC++を記述するだけで、 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てきた。これを用い、高性能な気象計算コード（平成26年度課題）、都市気流計算コード(平成27年度課題)を高生産に開発することに成功し、フレームワーク技術は様々な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。

一方、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) が有効であり、申請者らは、これまで単一GPU用のAMR法を開発した。

本年度は、これまでの研究で開発した単一GPU用のAMR法を拡張し、(1)複数GPU計算に対応させ、GPUスパコンで局所的に高解像度な大規模計算を実現する。GPU上での様々な解像度の格子を効率的に管理する技術や、局所的に高解像度となってもGPU間の計算負荷が均等になる動的負荷分散を実現する。凝固成長計算を対象としたGPUスパコン用AMR法を開発する。東京工業大学のGPUスパコンTSUBAMEの500台を超えるGPUを用い、解析対象（凝固の先端)を100倍の高解像度とした大規模計算を実現する。

次に、これを昨年度課題で開発したフレームワークへ導入することで、(2) GPU/CPU両対応した高性能AMRフレームワークを構築し、汎用的に適用できる技術とする。アプリケーション開発者視点から、AMR法用のデータ構造や入出力機能などアプリケーション開発に必須な要素も開発する。凝固成長計算などへフレームワークを適用し、これをTSUBAMEのGPUおよびCPUで大規模に実行し、最適化手法、生産性、達成できる実行性能を検証する。最終的にフレームワークを完成させ、GPUスパコンおよびCPUスパコン上で局所的に高精細にできるAMR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。凝固成長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、局所的に高精細とした大規模計算をGPUスパコンで実現させる意義は大きい。

しかしながら、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。さらに実装が複雑となるAMR法を導入したGPUアプリケーションの開発コストは極めて高い。本研究で開発するフレームワークは、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する。フレームワークの開発を通して、様々なアーキテクチャで動作する高性能アプリケーションを高生産に開発する基盤を確立することを目指す。本課題で開発したフレームワークは、誰からも使える局所的に高精細が必要な大規模GPU/CPUアプリケーションの開発を支援する基盤技術となり、波及範囲は広い。

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となる。

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、局所的に高精細を実現する高性能・高生産 AMR フレームワークを開発し、これを用いて局所的に高解像度となる凝固成長計算の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、 GPU スパコン上で AMR 法を実現するだけでなく、それをフレームワークとして汎用的に適用できる技術とする。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)、凝固成長計算を専門とする高木（京都工芸繊維大）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 AMR 法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、平成 26 年度課題で格子計算用の 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。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。

前年度（平成 27 年度）は、平成 26 年度課題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、都市気流計算コードを開発することに成功した。

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 GPU および CPU に対応した構造格子用フレームワークを発展させ、AMR 法フレームワークの構築に必要となるデータ構造とそれに対する計算機構の開発を行い、単一 GPU 用 AMR 法フレームワークを構築した。AMR 法に対応した汎用的なデータ構造やそのデータ構造に対して高速に計算できるコードを簡便に記述できる機構などを開発し、フレームワーク全体とそのプログラミングモデルを決定し、構築を進めた。これを用い、移流計算および、実装と計算がより複雑なレイリーテイラー不安定性シミュレーションにおいて局所的に高解像とできる計算に成功した。当初の研究計画では、複数 GPU による AMR 法フレームワークの構築と高解像度計算の実現を目標としていたが、レイリーテイラー不安定性シミュレーションへのフレームワークの適用を通して行ったフレームワーク自身の実装の再構築や単一 GPU 用コードからの複数 GPU コードへの展開に時間を要してしまい、複数 GPU への対応は完成には至らなかった。以下では、具体的な研究成果について説明する。

5.1 AMR 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。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。各時間ステップにおける格子の解像度は局所的に変化する。本フレームワークは直交格子上にブロック領域を定義し、その領域内を再帰的に細分化するブロック AMR 法として実装する。ブロック AMR 法では、計算領域内に様々な解像度のブロックが存在するが、ユーザは単一解像度の格子点上での計算についてのみ記述し、格子全体の処理、解像度の変更、解像度の異なるブロック間での袖領域のデータ交換などはフレームワークが行う。

5.2 多数の直交格子を保持するデータ構造

AMR 法では解像度の異なる多数の格子を扱う。前年度までの研究で、直交格子用フレームワークは独自の直交格子データ型 ETArray を導入することで、実行性能を保ちつつ、大幅にアプリケーションの生産性を向上させることに成功した。ETArray を用いることで、以下のように直交格子用の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。

今年度は、AMR 法を実現するため ETArray 型を基盤として多数の ETArray を保持する ETMArray 型を導入する。これを用いると以下の方法で、多数の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 narrays 個の直交格子用データを作成する
ETMArray<float, Range3D> fm_h(whole, narrays,
                              MemoryType::HOST_MEMORY);
ETMArray<float, Range3D> fm_d(whole, narrays
                              MemoryType::DEVICE_MEMORY);
```

narrays が新たに導入されたパラメータで、narrays 個の配列データを持つデータが生成される。GPU で性能が出やすくするため narrays 個の配列データは連続アドレスとして確保される。

5.3 多数の格子を管理する木構造

本フレームワークの対象とする AMR 法では構造格子を再帰的に細分化し、その空間的配置を木構造で表す。木構造の各リーフノードには、一つの格子ブロックを割り当てる。格子ブロックは典型的には 2 次元で 16×16 格子程度である。ETMArray 型で確保した多数の配列をこれらの格子ブロックとして割り当てる。図 1 に多数の格子の物理空間配置とメモリ空間の配置の模式図を示す。3 次元計算では八分木、2 次元空間では四分木となる。

木構造は空間的に点対称な広がりを持つため、単一の木構造で計算対象とする物理空間を充填させると、計算対象の形状によっては無駄な領域が発生する。そこで、図 2 のように複数の木構造を物理空間に配置する。これによって任意の領域形状を柔軟に表現することが可能となる。この複数の木構造を生成するため、本フレームワークでは、Field 型を提供する。次のように用いる。

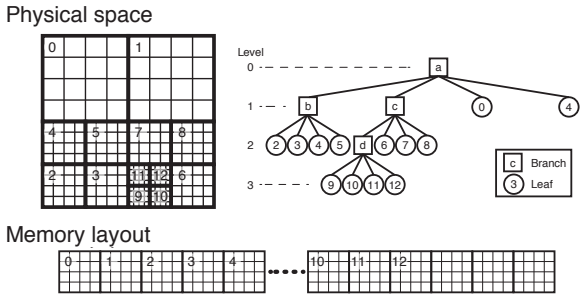

図 1　多数の格子の木構造による物理空間配置とメモリ空間配置

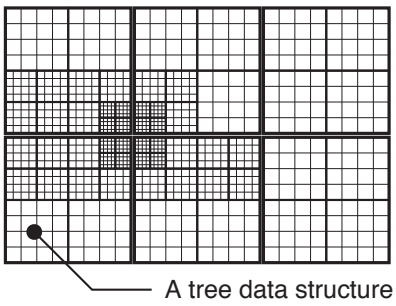

図 2　複数の木構造によるデータ構造

```
// 木構造の次元と各次元に配置する個数
const unsigned int ndim    = 2;
const unsigned int length[] = { 4, 4 };

// 木構造の生成
Field field(ndim, length);
// 全木構造を 2 段階成長
field.grow(2);
```

Field は木構造の次元と各軸に配置する木構造の数を指定して初期化される。上の例では 4×4 = 16 の木構造が生成される。Field::grow()関数により全木構造を指定レベル成長させることが可能となっている。Field 型自身は ETMArray 型のデータは保持しない。Field 型の各リーフノードは、ETMArray の何番目の配列が割り当てられているかを管理している。

5.4 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。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。このステンシル計算関数は、前年度に開発した構造格子用フレームワークと同一であり、これによって構造格子用コードを簡単に AMR 法用コードへ移植可能となる。

5.2 で述べた多数の配列を保持できるデータ構造 ETMArray を用いることで、ステンシル計算は次にように実行する。

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

...

viewma(fmn,inside, level >= 0)
   = funcf<float>(Diffusion3d(), ptr(fm), idx(fm),
                  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 ArravIndex を渡す
```

fm、fmn は ETMArray 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 ETMArray のある格子点 (i, j, k) のポインタを取得できる。viewma()は、fmn の保持する全ての格子に対して 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 の一つのグローバル関数に包み実行される。ETMAarray は木構造に関する情報を保持していない。level は field オブジェクトから生成されたもので、ETMAarray の各配列が AMR 法のいくつのレベルのリーフノードに配置されているかを表す。配置されていない配列は level<0 を持つため level >= 0 で有効な配列を選択している。これによって Field のリーフノードに割り当てられた配列にのみ右辺の関数が実行される。

5.5 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 ETMArray 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。

```
ETMArray<float, Range3D>
   am(whole, narrays, MemoryType::DEVICE_MEMORY);
ETMArray<float, Range3D>
   bm(whole, narrays, MemoryType::DEVICE_MEMORY);

am = 4.0;
bm = sqrt(am); // = 2.0
```

am, bm は ETMArray 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 am, bm 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、一般にデータ構造が複雑な AMR 法で見通しがよく簡便な記述を実現する。

5.6 2 つの格子ブロック間の袖領域データ交換

本フレームワークでは、リーフノード上の格子ブロックは、ステンシル計算のために袖領域を持つ。時間ステップを進めるためには、異なる解像度間で袖領域のデータ交換が必要である。低解像度の格子データから高解像度の格子データを生成する場合、新たに挿入される格子点には補間され

た値を代入する。現在の実装では、この値は 1 次関数により補間される。

袖領域のデータ交換は次の順に行う。まず、(1) 値の補間が必要ない同一解像度の格子ブロック間で袖領域のデータ交換を行う。次に (2) 値の補間が必要ない高解像度格子ブロックから低解像度格子ブロックへ袖領域のデータを転送する。格子点上に値が定義されている場合は、高解像度格子の値を間引くことで低解像度格子の値を決定する。最後に(3) 値の補間を必要とする低解像度格子ブロックから高解像度格子ブロックへ袖領域のデータを転送する。(3) では、補間のために隣接格子点を参照するため、先に(1)、(2)を実行する。

本フレームワークでは、この袖領域のデータ交換を簡便に記述するため、HaloExchange を提供する。HaloExchange を用いることで、ETMArray の袖領域のデータ交換を次のように記述できる。

```
ETMArray<float, Range3D>
   fm(whole, narrays, MemoryType::DEVICE_MEMORY);

HalloExchange<Range3D> exchange(inside, &field);
exchange.append(&fm);
exchange.transfer();
```

HaloExchange は、袖領域を除いた中心領域を示す Range 型の inside と格子ブロックの空間的配置が格納されている Field 型を引数に取り初期化される。Exchange::append()によって、任意の個数の ETMArray を追加することができ、Exchange::transfer()を実行すると、登録された全ての ETMArray は与えられた field によって解像度とブロック間の空間的位置関係が判断され、上の(1),(2),(3)が実行される。

5.7 格子ブロックの解像度の変更

リーフノードの持つ格子ブロックの解像度の変更は、物理量の変化等に着目する必要があるため、フレームワーク側では自動的には行わず、フレームワークは変更を指示する関数を提供する。

ユーザがある格子ブロックをより高い解像度にする、すなわちレベルを上げるよう指示すると、そのブロックは強制的に一段レベルが上がる。隣接するブロックと 2 段以上のレベル差ができた場合は、隣接するブロックのレベルを 1 段上げる。これを繰り返す。一方、ユーザがある格子ブロックをより低い解像度にするよう指示すると、隣接する格子が同じレベルか 1 段下のレベルの時のみ、1 段下げる。隣接ブロックに高い解像度の格子がある場合、物理的にその近傍で高い解像度が必要なことが示唆されているため、解像度は下げない。

格子ブロックの解像度の変更後、変更前後の異なる解像度の格子ブロック間でデータコピーを行う。5.6 で説明した袖領域のデータ交換を格子全体に拡げることで実行される。本フレームワークでは、これを簡便に記述するため、MeshModifier を提供する。MeshModifier を用いることで、ETMArray の持つ配列間で解像度を考慮したデータコピーを次のように記述できる。

```
ETMArray<float, Range3D>
   fm(whole, narrays, MemoryType::DEVICE_MEMORY);

MeshModifier<Range3D> modifier(inside, &field);
modifier.append(&fm);
modifier.refine_and_coarsen();
```

field に対して、事前に解像度の変更を指示することで、MeshModifier::append()で追加された任意の個数の ETMArray は MeshModifier::refine_and_coarsen() で実際に異なる解像度間でデータコピーされる。

5.8 AMR 法フレームワークのシミュレーションコードへの適用

5.8.1 移流計算

まず、本フレームワークの有効性を評価するため、2 次元 3 次精度風上手法を用いた移流計算を行う。高次精度数値計算手法である 3 次精度風上手法は、解像度の変化による誤差に対して敏感であり、異なる解像度間のコピーや袖領域のデータ転送が正しく作用しないと、非物理的な振動が発生する。

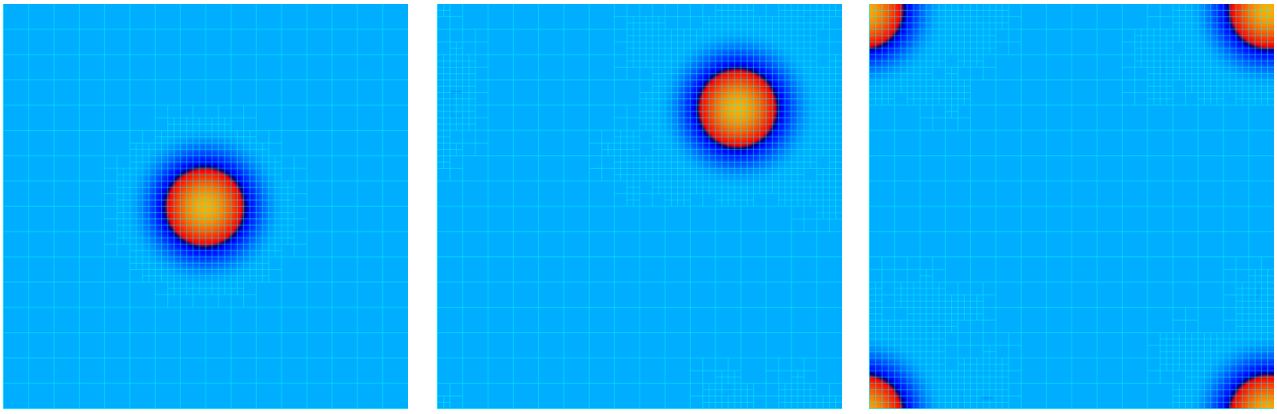

図 3 AMR 法を用いたコサインベルプロファイルの移流計算

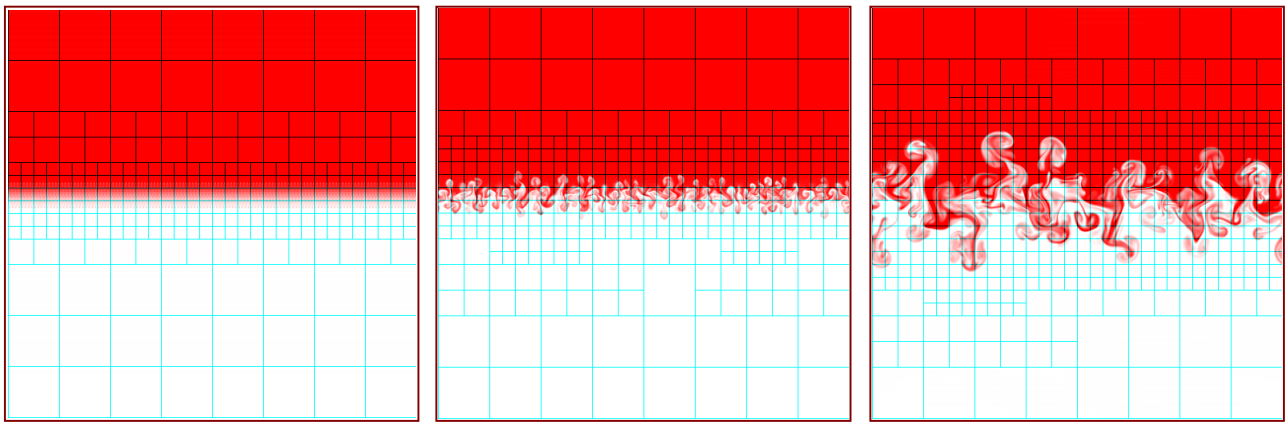

図 4 AMR 法を用いたレイリーテイラー不安定性の計算結果

ここでは、局所的に正の値を持つコサインベルを初期プロファイルとする。コサインベルプロファイルを計算領域の中心から x, y 方向に移流させた時のプロファイルの変化を図 3 に示す。図の格子状の水色線は各リーフノードが持つ格子ブロックを表す。一つの格子ブロックは 20×20 格子で、3 レベルの AMR を用いる。中心付近にコサインベルのプロファイルがあり、それを含む領域を AMR 法で高解像度にしている。AMR による高解像度領域は移流するプロファイルを良く追尾している。

5.8.2 レイリーテイラー不安定性シミュレーション

次に、より実装が複雑な問題における提案フレームワークの有効性を評価するため、3 次元オイラー方程式を解くことで圧縮性流体計算を行う。移流計算と異なり、この計算では密度、速度、エネルギーに関する 5 つの変数を扱う。移流項は、2 次元 3 次精度風上手法 に 3 次 TVD ルンゲ・クッタ法を用いる。このように解像度の変更を複数の変数に対して同時に行う必要があり、管理が複雑である。

図 4 に，この計算コードで計算したレイリーテイラー不安定性の計算結果を示す．図の格子状の水色線は各リーフノードが持つ格子ブロックを表す．一つの格子ブロックは 20×20 格子で，3 レベルの AMR を用いる．流体の色は二つの異なる密度を表している．高解像度が必要となる界面を含む領域を AMR 法で高解像度にしている．非物理的な振動を発生せずに計算することに成功した．

6 今年度の進捗状況と今後の展望

本研究課題では、前年度の研究で開発した GPU および CPU に対応した構造格子用フレームワークを発展させ、AMR 法フレームワークの構築に必要となるデータ構造とそれに対する計算機構の開発

を行い、単一 GPU 用 AMR 法フレームワークを構築した。AMR 法に対応した汎用的なデータ構造やそのデータ構造に対して高速に計算できるコードを簡便に記述できる機構などを開発し、フレームワーク全体とそのプログラミングモデルを決定し、構築を進めた。これを用い、移流計算および、実装と計算がより複雑なレイリーテイラー不安定性シミュレーションにおいて局所的に高解像とできる計算に成功した。

当初の研究計画では、複数 GPU による AMR 法フレームワークの構築と高解像度計算の実現を目標としていたが、フレームワーク自身の実装の再構築や単一 GPU 用コードからの複数 GPU コードへの展開に時間を要してしまい、複数 GPU への対応は完成には至らなかった。具体的には、AMR 法ではデータ構造が複雑となり、メモリアクセスが不規則となるため、2 つの格子ブロック間の袖領域データ交換や格子ブロックの解像度の変更で性能低下する。複数 GPU に対応した AMR 法では、これらをノードを超えて管理し、実行する必要がある。このように複雑で最適化された実装を必要とするため、その研究開発に時間を要した。

今後は、上記の問題を解決し、複数 GPU による AMR 法フレームワークを完成させる。その後、近年、メニーコアプロセッサ Xeon Pho を搭載したスパコンが稼働してきているが、本フレームワークは、これに対応していない。そこで、GPU 用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、さらには Xeon Phi で高速実行できるコードを生成するように拡張を進めることを目指す。

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,” 6th International Workshop on Advances in High-Performance Computational Earth Sciences: Applications and Frameworks, IHPCES2011, the 2016 International Conference on Computational Science, ICCS 2016, San Diego, US, June 2016, Pages 1646-1557.

(3) 会議発表(口頭，ポスター等)

[2] 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)

[3] Takashi Shimokawabe, “Large-scale GPU-based CFD Applications based on a High-productivity Stencil Framework,” Parallel CFD 2016, Kobe, Japan, May 10, 2016. (招待講演)

[4] 下川辺隆史, 遠藤敏夫, 青木尊之, “GPU デバイスメモリを超える計算を可能とするためのステンシル計算フレームワークの拡張とその性能評価”, 第 21 回計算工学講演会論文集, 新潟, 2016 年 6 月

[5] Takashi Shimokawabe, Toshio Endo, Naoyuki Onodera, Takayuki Aoki, ″Performance Evaluation of Wind Simulation Based on a GPU-computing Framework to Realize Large-scale Stencil Computations Beyond Device Memory Capacity,″ The 7th AICS International Symposium, Kobe, February 2017. (ポスター)

[6] Takashi Shimokawabe, ″Large-scale GPU Applications Based on a High-productivity Stencil Framework,″ Computational Sciences Workshop 2017, Kanagawa, March 2017. (招待講演)

(4) その他（特許，プレス発表，著書等）

なし