jh170044-NAH

AMR 法フレームワークの様々なアーキテクチャへ向けた発展

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おり、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。本研究課題では、前年度課題を発展させ、単一の GPU と CPU だけでなく、スパコンに搭載された最新の GPU や近年スパコンへ搭載されているメニーコアプロセッサ Xeon Phi による複数ノード計算へ対応したフレームワークへ発展させた。AMR 法フレームワークの基盤となるステンシル計算用フレームワークを C++11 のラムダ式を用いるように再構築することで、Xeon Phi へ対応させた。再構築したステンシルフレームワークと前年度に開発した単一 GPU 用 AMR 法フレームワークを基盤に、複数 GPU 計算へ対応した AMR 法フレームワークを構築した。このフレームワークを二つの異なる計算手法で実装された圧縮性流体計算および金属凝固成長計算に適用し、その有効性を検証した。

1. 共同研究に関する情報

(1) 共同研究を実施した拠点名

東京大学

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（東京大学 情報基盤センター）：研究全体の統括、AMR 法フレームワークの完成と流体中を流れ成長する金属凝固計算への適用
- 高木 知弘（京都工芸繊維大学 機械工学系）：流体中を流れ成長する金属凝固計算コードの提供と AMR 法導入の検討
- 青木 尊之（東京工業大学 学術国際情報センター）：AMR 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
- 小野寺 直幸（日本原子力研究開発機構システム計算科学センター）：格子ボルツマン法に関する助言
- 星野 哲也（東京大学 情報基盤センター）：Xeon Phi 向け最適化手法に関する助言

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 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。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、GPU と CPU でステンシル計算を高速実行できる高性能・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を開発に成功し（平成 26 年度課題）、都市気流計算コードを高生産に開発を進め（平成 27 年度課題）、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。

近年、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）が有効である。そこで前年度課題では、これまでに開発したステンシル計算フレームワークを基盤として、高性能 AMR フレームワークの構築を進め、単一 GPU 用 AMR 法フレー

ムワークを構築し、圧縮性流体計算に成功した。

本年度は、前年度課題を発展させ、GPUとCPUだけでなく、近年、スパコンに搭載されてきているメニーコアプロセッサ Xeon Phi へ対応させ、(1) GPU/CPU/Xeon Phi へ対応したAMRフレームワークを構築し、AMRアプリの開発技術を確立する。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に加え、Xeon Phiで高速実行できるコードを生成できるようにする。また、複数GPU、複数CPU、複数Xeon Phi計算できるようフレームワークを高度化する。

次に、開発したAMR法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、本年度は、(2) 二つの計算手法を併用した流体中を流れ成長する金属凝固計算へフレームワークを適用し、より複雑な問題においても局所的に高解像度となる計算を実現できることを目指す。最終的には、(1)で構築したフレームワークを用いることで、GPU/CPU/Xeon Phiを搭載したスパコン上で、流体中の金属凝固成長計算の大規模計算を実現することを目指す。

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やXeon Phiなどのメニーコアプロセッサが搭載され、日本、米国、中国などで稼働している。流体中を流れ成長する金属凝固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異なるプロセッサを搭載した様々なスパコンで実現させる意義は大きい。

大規模アプリケーションは一般に特定のアーキテクチャに対してのみ最適化され動作するため、他のアーキテクチャでは動作せず、生産性や保守性が低い。さらに実装が複雑となるAMR法を導入したアプリケーションの開発コストは極めて高い。申請者らは、これを解決するため、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AMR法フレームワークの構築を進めた。本課題は、これを発展させるもので、本フレームワークを主要なメニーコアプロセッサの一つである Xeon Phi に対応させ、その適用範囲を拡大する。

次世代スパコンでは、GPUやXeon Phi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、異なるプロセッサを搭載した様々なスパコン、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を実現するAMR法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/CPU/Xeon Phiを搭載したスパコン上で、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流体中を流れながら成長する金属凝固成長計算を実現する。GPU を搭載する東京工業大学の TSUBAME2.5/3.0 と、Xeon Phi を搭載する東京大学の Oakforest-PACS は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、流体中を流れ成長する金属凝固成長計算に AMR 法を適用するだけでなく、その開発技術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への適用が重要となる。本研究は、大規模アプリケーションの専門とする下川辺（東大）が中心となり、凝固成長計算を専門とする高木（京都工芸繊維大）、流体計算の専門とする青木（東工大）、小野寺（原研）、計算機科学を専門とする星野（東大）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様々なアーキテクチャに対応した実用に耐えうる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 年度課題でステンシル計算用の 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ステンシル関数から最適化された GPU 実行コードを生成し、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。

平成 27 年度は、平成 26 年度課題で開発した構造格子用 GPU フレームワークを高度化し、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。CPU 計算では OpenMP による並列計算に対応した。GPU 計算において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の開発に成功した。

前年度（平成 28 年度）は、平成 27 年年度課題で開発した 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、移流計算および圧縮性流体計算において局所的に高解像とできる計算に成功した。

5. 今年度の研究成果の詳細

本研究計画では、AMR 法フレームワークのメニーコアプロセッサ Xeon Phi への対応に向け、その基盤となるステンシル計算用フレームワークの Xeon Phi への対応を行った。当初の計画では、前年度の研究で開発した GPU および CPU に対応したステンシル計算用フレームワークをそのまま拡張し Xeon Phi へ対応する予定であったが、既存のフレームワークは式テンプレートを多用しており、このために Xeon Phi では高い性能が得られないことが判明した。このため、本研究計画では、C++11 のラムダ式を用いてステンシル計算用フレームワークを再構築した。これにより Xeon Phi および GPU で高い性能が得られることに成功した。また、前年度の研究で開発した単一 GPU 用 AMR 法フレームワークと C++11 のラムダ式を用いてステンシル計算用フレームワークを基盤とすることで、複数 GPU 計算へ対応した AMR 法フレームワークを構築した。これを用いて圧縮性流体計算に成功し、金属凝固成長計算へ適用し有効性を検証した。構築した AMR 法フレームワークは、Xeon Phi へ実行できるものの、研究計画になかったステンシルフレームワークの再構築に時間を要し、AMR 法フレームワークへと拡張した部分の Xeon Phi への最適化が完了しなかったため、当初の研究計画で期待したような高い性能を達成できず、Xeon Phi への対応は完成に至らなかった。以下では具体的な研究成果について説明する。

5.1 ステンシルフレームワークの様々なプロセッサへの対応

これまでの研究で開発したステンシル計算用フレームワークは、GPU および従来の CPU で高速計算が可能である。本研究計画ではこれを Xeon Phi へ対応させる。

5.1.1 Xeon Phi 向け最適化手法

Xeon Phi では従来の CPU 用コードをそのまま実行することができるが、高い性能を得るためには、様々な最適化を導入する必要がある。最適化による実行性能向上の効果を検証するため、流体計算の基礎的な方程式である拡散方程式に対し、以下の最適化を適用する。

(1) OpenMP の適用
(2) 最内ループの SIMD 化
(3) オンパッケージの高バンド幅メモリである MCDRAM の利用、OS 割り込み処理などによるジッタの影響を回避するための明示的なコア割り当て
(4) メモリアライメントの適用（64 バイトでアライメント）

(1)、(2)、(4)はソースコードレベルの最適化、(3)は実行時オプションで実現される。

図 1 に 3 次元拡散方程式に対して最適化(1)-(4)を順に適用した場合の実行性能変化について示す。計算格子サイズは 256^3 で、計算は単精度で行う。性能測定は、東京大学の Oakforest-PACS を用いる。Oakforest-PACS は 8000 台を超える Intel Xeon Phi 7250（開発コード Knights Landing; KNL)が搭載されている。ソースコードレベルの最適化である(1)、(2)、(4)をユーザコードに適用できることは必須であり、それを実現できるようフレームワークを高度化する。

5.1.2 直交格子データ構造の拡張

最適化(4)をフレームワークで実現するため、フレームワークの提供する独自データ型 Array を拡張する。Array 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
Array<float, Range3D> f_h(whole,MemoryType::HOST_MEMORY);
Array<float, Range3D>
                  f_d(whole, MemoryType::DEVICE_MEMORY);
Array<float, Range3D>
                  f_k(whole, MemoryType:: HOST_ALIGNED64_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や GPU のデバイスメモリ上に確保することができる。最適化(4)に対応するため、MemoryType:: HOST_ALIGNED64_MEMORY を導入する。

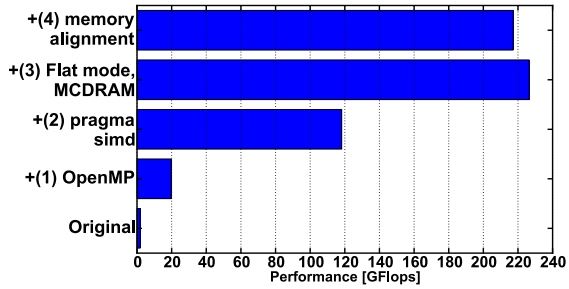

図 1 単一 Xeon Phi における拡散計算の最適化適用による実行性能の変化（単精度計算）

5.1.3 様々なプロセッサへ対応したステンシル計算方法

次に最適化(1)、(2)をフレームワークで実現するため、ステンシル計算の定義と実行を拡張する。本フレームワークでは、従来はステンシル計算の実行に C++の式テンプレート技術を利用していたが、式テンプレートでは補助的な型を多用し、Xeon Phi ではこれが並列化を阻害するためか、高い性能を達成できない。従来のフレームワークに最適化(1)-(4)を適用した場合でも、計算格子サイズ 256^3 の単精度計算で 69.9 GFlops しか達成できなかった。これは図 1 の+(4)の性能から大幅な性能低下である。

これを解決するため、式テンプレート技術を利用せずに、C++11 で導入されたラムダ式を活用し、ステンシル計算の定義と実行する機構を構築する。

ラムダ式を用いて新たに構築した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、ラムダ式として定義する。これをステンシル計算関数と呼ぶ。データ構造 Array を用いることで、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し実行することができる。

```
Engine_t engine;
engine.run(inside,
  [=] __host__ __device__ (const ArrayIndex3D &idx,
       const float *f, float *fn) {
       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)];
  }, idx(fa.range()), ptr(&fa), ptr(&fan));
```

ArrayIndex3D 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。

engine は、第 2 引数にラムダ式で表されたステンシル計算を取り、これに第 3 引数以降を渡す。fa、fan は Array 型、inside は Range3D 型で、ptr()はステンシル計算関数中で Array の inside の開始点のポインタを取得できる。ArrayIndex3D と組み合わせることで、格子点 (i, j, k) のアドレスを取得できる。engine は、ステンシル関数を inside 領域内のみ実行する。従来は式テンプレートを用いて記述した式をラムダ式内に記述することが可能となっている。

5.1.4 GPU における性能評価

ラムダ式によって実装されたステンシル計算関数の実行性能を検証する。まず、GPU での性能を評価するため、東京工業大学の GPU スパコン TSUBAME3.0 を用い、本フレームワークを用いて実装した拡散方程式の性能を評価する。TSUBAME3.0 は 2000 台を超える NVIDIA Tesla P100 GPU が搭載されている。計算は全て単精度で行う。

図 2 に複数 GPU 計算の強スケーリングを示す。1 ノードあたり 4GPU を用いる。計算格子サイズ 256^3 では 1GPU で 461.1 GFlops を達成している。参照実装として CUDA を用い実装した拡散方程式では、同じ計算格子サイズで 407.2 GFlops であり、本フレームワークでは参照実装よりも高い性能を達成することに成功した。

5.1.5 Xeon Phi における性能評価

次に Xeon Phi での性能を評価するため、東京大学の Oakforest-PACS(OFP)を用い、GPU の場合と同じく本フレームワークを用いて実装した拡散方程式の性能を評価する。計算は全て単精度で行う。

図 3 に複数 Xeon Phi を用いた計算の強スケーリングを示す。計算格子サイズ 256^3 では 1 台の Xeon Phi で 209.1 GFlops を達成している。5.1.1 に示した手で実装した拡散方程式では 226.4 GFlops を達成しており、フレームワークで達成した性能はこれよりも低い性能ではあるが 92%の性能を達成できている。

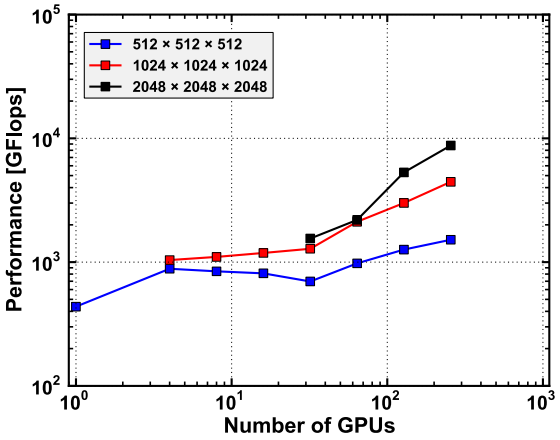

図 2 ラムダ式によるステンシル計算フレームワークを用いた複数 GPU による拡散計算の強スケーリング

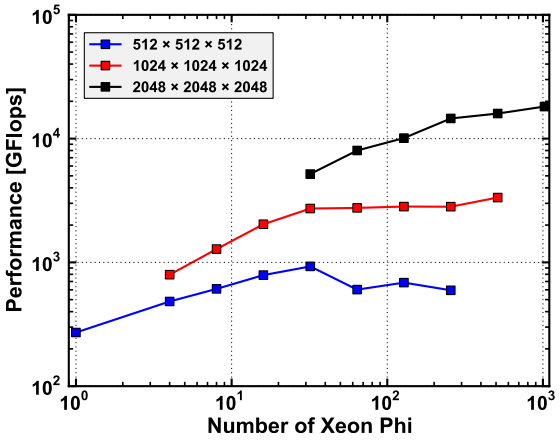

図 3 ラムダ式によるステンシル計算フレームワークを用いた複数 Xeon Phi による拡散計算の強スケーリング

図 3 に示すように良い強スケーリングが得られている。図 2 の GPU による強スケーリングと比較すると、1 台の GPU と Xeon Phi では GPU の方が高い性能が得られるが、利用する台数を増やすと、TSUBAME3.0 と比較して OFP が高い性能を得られることがわかった。これは OFP が 1 ノードに 1 台の Xeon Phi が搭載されているのに対して、TSUBAME3.0 は 1 ノードあたり 4GPU が搭載されているためと考えられる。

5.2 複数 GPU 計算のための AMR フレームワーク

5.1 で構築した C++11 のラムダ式を用いたステンシル計算フレームワークと前年度に構築した単一 GPU 用の AMR 法フレームワークを基盤とすることで、複数 GPU 計算へ対応した AMR 法フレームワークを構築する。構築したフレームワークは Xeon Phi へ対応はしているものの、期待したような高い性能を達成できず、Xeon Phi への対応は完成に至っていない。

5.2.1 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。図 4 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。GPU は連続したメモリ領域へアクセスするときに高い実行性能となるため、格子ブロックは物理変数ごとに一つの大きな連続メモリ領域に確保する。各リーフノードは、直接は格子ブロックを保持せず、格子ブロックを特定する ID を保持する。この ID から割り当てられた格子ブロックの連続メモリ領域における位置を求め、格子ブロック上のデータを参照する。

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、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。

```
// 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 };

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

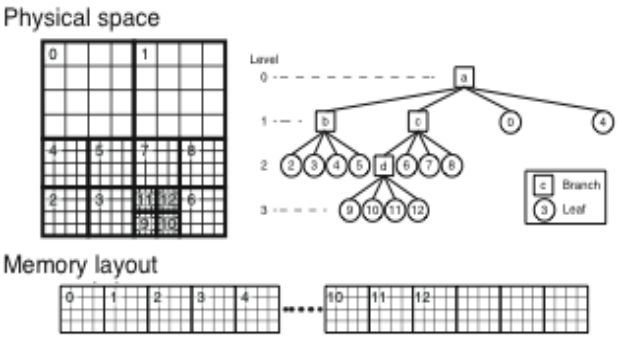

図 4　多数の格子の木構造による物理空間配置とメモリ空間配置

複数 GPU による計算では MPI で並列化され、各プロセスが計算領域全体の木構造を表現する Field 型データ を保持する。(1) 各リーフに対する解像度の変更の指示、 (2) 特定のリーフのデータをある GPU から別の GPU へ移行させる指示、はプロセス毎に発行することが可能で、発行の後、この情報を MPI 通信によって全プロセスで共有する。共有後に、各プロセスは自分の保持する計算領域全体を表現する木構造を変更する。各プロセスの木構造自身は通信により明示的に同期することはないが、木構造を変更する上記の二つの「指示」を共有することで、全てのプロセスは常に同一の木構造を保持する。この計算領域全体の木構造の情報をもとに、適切に境界領域の交換などを行う。

5.2.2 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 MArrayIndex3D 等を用い、C++11 で 導入されたラムダ式を使い定義する。フレームワークの提供するデータ構造 MArray を用いることで、効率的な記述を実現する。MArray は、 大きさと位置の情報を持つ Range3D 型と配列を保持するクラスであり、この配列を連続メモリ領域として使う。図 4 のように分割して複数の格子ブロックとし、木構造のリーフへ分配し利用する。

AMR 法フレームワークではステンシル計算を多数の格子ブロックに同時に実行する。C++11 で

導入されたラムダ式を使い定義するステンシル計算は、5.1.3 で示した直交格子用の従来のステンシル計算関数とほぼ同じである。従来のステンシル計算関数と異なり、ArrayIndex の代わりに MArrayIndex を用いる必要がある。このクラスは AMR に対応するため内部的な処理は異なるが、これが持つ関数(インターフェース)は同等である。

AMR 法フレームワークでは、ステンシル計算を複数の解像度の異なる格子ブロックに対して次のように実行する。

```
Range3D inside; // inside は格子点のうち，更新した領域 Engine_t
engine;
engine.run(amrcon, inside, LevelGreaterEqual(1),
    diffusion3d, idx(fa.range()), level(),
    ce,cw,cn,cs,ct,cb,cc, ptr(fa), ptr(fan));
```

engine は、第 1 引数に AMR データ構造を表す Field などを保持するオブジェクト、第 4 引数にラムダ式で表されたステンシル計算を取り、これに第 5 引数以降を渡す。fa、fan は MArray 型、inside は Range3D で、ptr() はステンシル計算関数中で MArray の inside の開始点のポインタを取得する。level() は計算が適用される格子ブロックの AMR のレベルを取得する。これによりレベル毎に異なる計算を行うことが可能である。engine は、ステンシル関数を第 2 引数で渡された inside 領域内に対し、第 3 引数の条件を満たすレベルの格子ブロックに対して適用する。LevelGreaterEqual(1) を指定することで、このステンシル計算関数は、レベル 1 以上の格子ブロックに適用される。

格子ブロックは前節で述べたように連続メモリ領域に確保されるため、フレームワークは、あるステンシル関数を複数の解像度の異なる格子ブロックに対して同時に適用することができる。

5.2.3 同一 GPU 内における格子ブロック間の袖領域の交換

本フレームワークでは、リーフ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、異なる解像度間の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。袖領域の交換のための計算を GPU 上で実行する際に、GPU 上で木構造にアクセスすると性能低下する。これを回避するため、本フレームワークでは、木構造から一時的に格子ブロックの隣接関係を表現したテーブル構造を作成し、これを利用する。特にテーブル構造では固定長の列とすることで、GPU 上でのアクセスを容易にしている。

5.2.4 異なる GPU 内における格子ブロック間の袖領域の交換

複数 GPU 計算では、時間発展させるためには、同一 GPU 内における格子ブロック間の袖領域データ交換とともに、異なる GPU に割り当てられた格子ブロックの袖領域データも交換する必要がある。ステンシル計算に必要な格子ブロックを隣接 GPU から転送するとき、まずフレームワークは、各プロセスにおいて、連続メモリ領域から未使用な格子ブロックの一部を一時領域として確保する。次に、MPI 通信を利用して実際に隣接 GPU からステンシル計算に必要となる格子ブロックを転送し、一時領域へ保存する。各プロセスでのステンシル計算は、この一時領域を参照して実行される。

異なる GPU 間の袖領域の交換では、ステンシル計算で必要となる格子ブロックの袖領域だけでなく、格子ブロックの全領域のデータを転送する。これは大きな性能低下につながるが、ステンシル計算では複数の時間ステップに必要な通信をまとめて実行できる時間ブロッキング法が有効であることがわかっており [1]、これを利用することで、この通信による性能低下を抑えている。

5.2.5 格子解像度の変更

リーフ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、フレームワークが提供する解像度変更を指示する関数を用いて、解像度を変更する格子ブロックを明示的に指

定する。複数 GPU 計算では、この指示が全プロセスで共有される。

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 1 段レベルが上がる。本フレームワークでは隣接格子ブロック間で高々1 段のレベル差しか許していないため、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。

解像度を変更する格子ブロックが最終的に決定した後に、フレームワークは、解像度変更のための格子ブロック間の値のコピーを実際に並列で実行する。解像度変更のためのコピーでは、値の補間が必要であり、現在、この値は 1 次関数により補間される。連続メモリ領域から未使用な格子ブロックを確保し、これをコピー先として用いる。不要になったコピー元の格子ブロックは将来の利用のため未使用領域として分類される。

5.2.6 格子ブロックの GPU 間の移行

AMR 法では、時間発展とともに高解像度となる局所的な領域の大きさや位置が変化する。複数 GPU による AMR 法では、性能低下を抑えるために、常にそれぞれの GPU にほぼ同数の格子ブロックが割り当てられるようにして、計算負荷を均等にする必要がある。

フレームワークは、格子ブロックをある GPU から他の GPU へ移行させることを指示する関数を提供する。 ユーザが格子ブロックに対して移行を指示し、それが完了した後に、フレームワークは実際にその指示に従い移行を実行する。複数 GPU 計算では、移行実行前にこの指示が全プロセスで共有される。フレームワークは未使用の格子ブロックを連続メモリ領域から確保し、ここに他の GPU からの格子ブロックのデータを移し、これ以後、この格子ブロックを利用する。

5.2.7 AMR 法フレームワークの適用例

(1) 圧縮性流体計算

複数 GPU に対応した AMR フレームワークの有効性 を確認するため、2 次元 3 次精度風上手法を用いた圧縮性流体計算へ適用する。図 5 に 8 台の NVIDIA Tesla P100 GPU による圧縮性流体計算による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 20^2 格子 で、5 レベルの AMR を用いる。最大格子の幅は最小格子の幅の 16 倍となる。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。

3 次元 3 次精度風上手法を用いた圧縮性流体計算へ 適用し、実行性能を評価する。512 × 512 × 1024 の均一格子と同等の計算領域を 4 レベルの AMR を適用し、 東京工業大学の TSUBAME3.0 スパコンの 4 ノードを用い、各ノードから 2 台、合計 8 台の NVIDIA Tesla P100 GPU で計算する。図 6 は、各時間ステップに要した計算時間を示す。1 つの格子ブロックは 20^3 格子で、最大格子の幅は最小格子の幅の 8 倍となる。動的負荷分散 (Migration) と時間ブロッキング法 (TB) を有効または無効にして実行する。導入した動的負荷分散手法と時間ブロッキング法を用いることで、GPU 間の袖領域交換手法の実行性能が向上し、各時間ステップに要する計算時間が短くなっている。

(2) 金属凝固成長計算

次に本フレームワークをフェーズフィールド法による金属凝固成長計算へ適用する。本計算は、TSUBAME3.0 の NVIDIA Tesla P100 GPU 1 台を用いる。図 7 に計算例を示す。金属凝固している領域に合わせて格子が集約されていることがわかる。

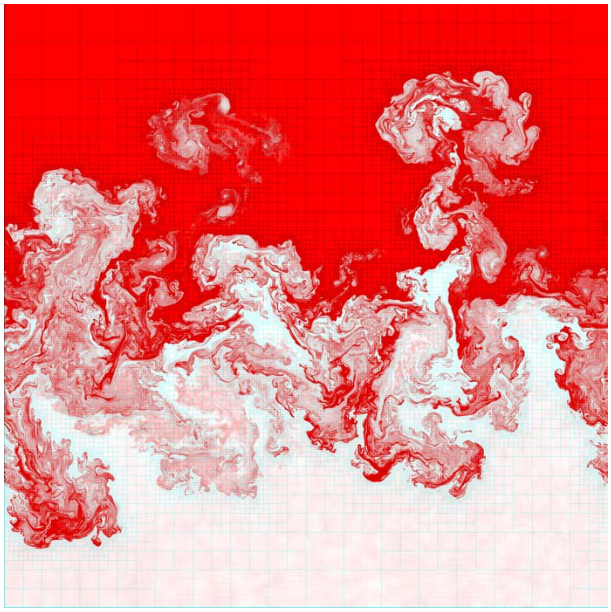

図 5 AMR 法フレームワークを用いたレイリーテイラー不安定性の複数 GPU による計算結果

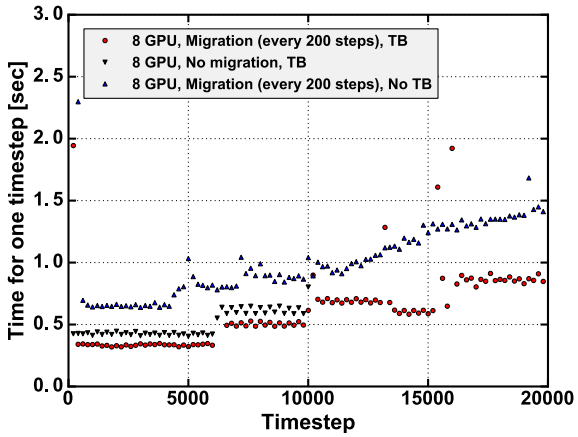

図 6 AMR 法を適用した 3 次元圧縮流体計算における各時間ステップの計算時間

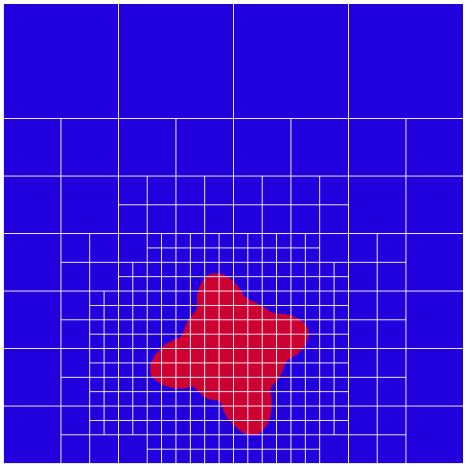

図 7 AMR 法フレームワークを用いたフェーズフィールド法による金属凝固成長計算の例

6. 今年度の進捗状況と今後の展望

本研究計画では、AMR 法フレームワークのメニーコアプロセッサ Xeon Phi への対応に向け、その基盤となるステンシル計算用フレームワークの Xeon Phi への対応を行った。当初の計画では、前年度の研究で開発した GPU および CPU に対応したステンシル計算用フレームワークをそのまま拡張し Xeon Phi へ対応する予定であったが、既存のフレームワークは式テンプレートを多用しており、このために Xeon Phi では高い性能が得られなかった。本研究計画では、C++11 のラムダ式を用いてステンシル計算用フレームワークを再構築した。これにより Xeon Phi および GPU で高い性能が得られることに成功した。また、前年度の研究で開発した単一 GPU 用 AMR 法フレームワークと C++11 のラムダ式を用いてステンシル計算用フレームワークを基盤とすることで、複数 GPU 計算へ対応した AMR 法フレームワークを構築した。これを用いて圧縮性流体計算に成功し、金属凝固成長計算へ適用した。

当初の研究計画では、複数 GPU 計算へ対応に加えて、複数 Xeon Phi へ対応した AMR 法フレームワークの構築を目標としていた。しかし Xeon Phi へ対応するためには当初の研究計画になかった C++11 のラムダ式によるステンシル計算用フレームワークの再構築が必要となった。これに時間を要したため、構築した AMR 法フレームワークは、Xeon Phi へ実行できるものの、最適化が完了しないない。このため当初の研究計画で期待したような高い性能を達成できず、Xeon Phi への対応は完成に至らなかった。

一方で、C++11 のラムダ式によりステンシル計算フレームワークを再構築したことで、従来よりも柔軟な記述が可能となり、結果的には構築した AMR 法フレームワークは記述性の優れたものとなり、再構築自体は非常に有効であった。

今後は、圧縮性流体計算および金属凝固成長計算に AMR 法フレームワークを適用できたので、これを併用した流体中を流れ成長する金属凝固計算への適用を進めていく。また、構築した AMR 法フ

レームワークは複数 GPU 計算できるが、計算負荷の分散や通信の最適化に高度化の余地がある。GPU スパコン上で実行時間を最小化するためのフレームワークの高度化を進める。合わせて Xeon Phi への最適化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, *Toshio Endo*, Naoyuki Onodera and Takayuki Aoki, "A Stencil Framework to Realize Large-scale Computations Beyond Device Memory Capacity on GPU Supercomputers," 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, Sep. 2017, pp. 525-529.

(3) 国際会議発表

[2] Takashi Shimokawabe, Takayuki Aoki, and Naoyuki Onodera, "An AMR Framework for Realizing Effective High-Resolution Simulations on Multiple GPUs," 18th SIAM Conference on Parallel Processing for Scientific Computing, Tokyo, Japan, Mar. 2018.

[3] Takashi Shimokawabe, ″AMR Framework for Realizing Effective High-Resolution Simulations on GPU,″ GTC 2018, San Jose, CA, USA, Mar. 2018. (ポスター)

(4) 国内会議発表

[4] 下川辺隆史、青木尊之、小野寺直幸, "高精細計算を実現する AMR 法フレームワークの開発", 日本計算工学会 第 22 回計算工学講演会, 大宮, 2017 年 6 月.

[5] Takashi Shimokawabe, "An AMR Framework for Realizing Effective High-Resolution Simulations on GPU," 29th Advanced Supercomputing Environment (ASE) Seminar, The University of Tokyo, Tokyo, Aug. 2017.

[6] Takashi Shimokawabe, "A Stencil Framework to Realize Large-scale Computations Beyond GPU Device Memory Capacity," 32nd Advanced Supercomputing Environment (ASE) Seminar, The University of Tokyo, Tokyo, Nov. 2017.

(5) その他（特許，プレス発表，著書等）

なし