学際大規模情報基盤共同利用・共同研究拠点 公募型共同研究 平成29年度採択課題
9th Symposium
Joint Usage / Research Center for Interdisciplinary Large-scale Information Infrastructures
jh170044-NAH

下川辺隆史（東京大学）

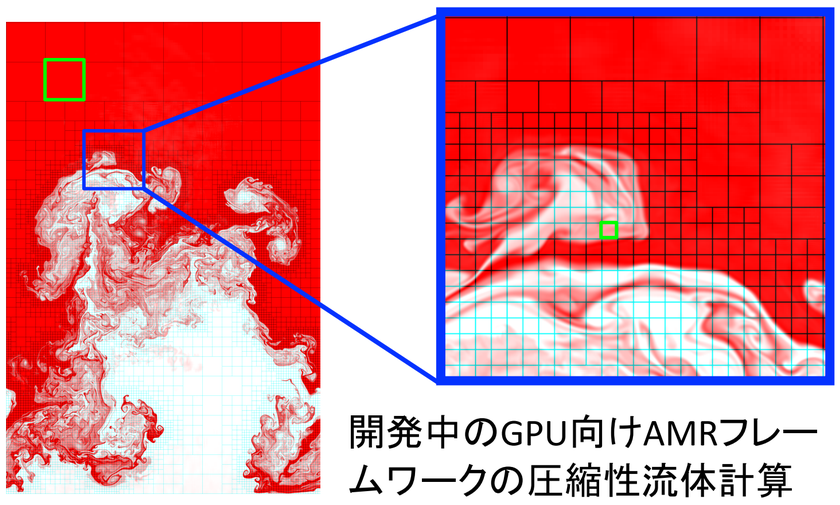
AMR法フレームワークの様々なアーキテクチャへ向けた発展

1 研究背景と研究目的

近年、ステンシル計算を用いた格子に基づいたシミュレーションでは、大規模なGPU計算が可能となり、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。

本研究では、開発中のGPU/CPU向けの高生産・高性能AMRフレームワークを、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 へ対応したフレームワークへ発展させる。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。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、最終的に、フレームワークを完成させ、GPUスパコン、CPUスパコン、Xeon Phiを搭載した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。

本年度は、開発したフレームワークが、より複雑なアプリケーションへ適用できることを実証するため、現在取り組んでいる流体中を流れながら成長する金属凝固計算にフレームワークを適用する。



開発中のGPU向けAMRフレームワークの圧縮性流体計算への適用例。緑のブロックは同一数の格子点を持つ。

2 AMR法フレームワーク

AMR法フレームワークの概要を述べる。平成26、27年度課題で開発したGPU/CPUで高性能を実現するステンシル計算フレームワークを発展させ構築されている。

2.1 フレームワークの対象

- ✓ 直交格子をベースとしたブロック型のAMR
- ✓ 各格子点上で定義される物理変数（配列）の時間変化を計算
- ✓ 物理変数の時間ステップ更新は陽的でステンシル計算で行う

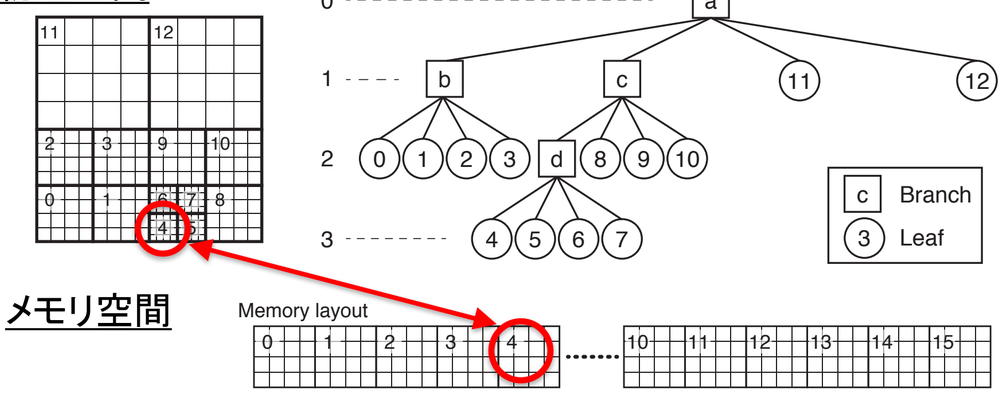
2.2 フレームワークの設計

- ✓ フレームワークはC++で構築
- ✓ ユーザは基本的にステンシル計算についてのみ記述
- ✓ AMRでは様々な解像度のブロックが存在するが、ユーザはあたかも単一解像度への計算として記述できる
  → これを実現するため各ブロックは袖領域の格子を持つ
- ✓ AMRデータは木構造で管理するが、これを意識しないプログラミング
- ✓ 任意の数の物理変数（配列）を扱える

2.3 AMR法のデータ構造

- ✓ 構造格子を再帰的に細分化し、木構造で表す
- ✓ 物理空間ではリーフノードに格子ブロックを配置
- ✓ メモリ空間では複数の格子ブロックをフラットに配置
- ✓ リーフノードとメモリ上の格子ブロックは整数値(Id)で対応付け

物理空間



メモリ空間

2.4 ステンシル関数の記述と実行

- ✓ メモリレイアウトをフラットな構造とし、各格子ブロックで袖領域を持つことで、直交格子用のステンシル計算関数を利用可能

```
struct Diffusion3d { // User-written stencil function
__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; // this function updates one point.
  }};
```

(i, j, k)

(i, j, k-1)

- ✓ フラットなメモリレイアウトにより複数の格子ブロックを同時に計算できる

```
Range3D inside; // ステンシル関数の実行範囲

viewma(fn, inside, level >= 0)
          = funcf<float>(Diffusion3d(), ptr(f), idx(f),ce,cw,cn,cs,ct,cb,cc);
```

3 Xeon Phi対応に向けた拡張

Xeon Phi に有効な最適化を導入したコードをフレームワークで生成するため、Xeon Phi に最適化された参照コードを実装し性能評価を進める。

3.1 Xeon Phi

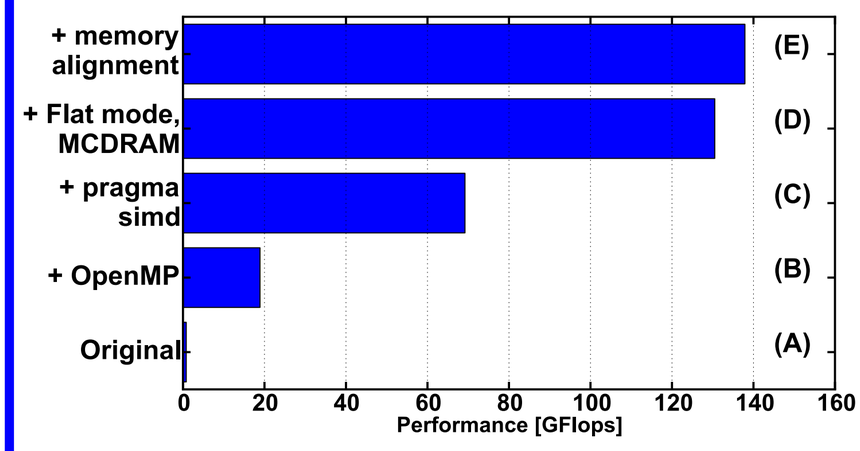
- ✓ 東京大学/筑波大学の Oakforest-PACS の Intel Xeon Phi 搭載ノードを利用
  - ✓ Intel Xeon Phi 7250 (Knights Landing)、68コア、1.4 GHz、理論ピーク性能 3.0 TFlops
  - ✓ メモリ：MCDRAM、16 GByte, 実効490 GB/sec

3.2 Xeon Phi に有効な最適化手法の検討

- ✓ OpenMPの適用とベクトル化
- ✓ メモリアライメント
- ✓ MCDRAM（オンパッケージの高バンド幅メモリ）の利用
- ✓ 明示的なコア割り当て（OS割り込み処理などによるジッタの影響の回避）

Xeon Phi 向けの拡散方程式のコード例

```
#pragma omp parallel for collapse(2)
for(int k = 0; k < nz; k++) {
  for (int j = 0; j < ny; j++) {
#pragma simd
    for (int i = 0; i < nx; i++) {
      const int ix = nx*ny*k + nx*j + i;
      const int ip = i == nx - 1 ? ix : ix + 1;
      const int im = i == 0      ? ix : ix - 1;
      const int jp = j == ny - 1 ? ix : ix + nx;
      const int jm = j == 0      ? ix : ix - nx;
      const int kp = k == nz - 1 ? ix : ix + nx*ny;
      const int km = k == 0      ? ix : ix - nx*ny;
      fn[ix] = cc*f[ix] + ce*f[ip] + cw*f[im] + cn*f[jp]
             + cs*f[jm] + ct*f[kp] + cb*f[km];
}}}
```



拡散方程式に様々な最適化を適用した場合の実行性能の変化。
(A) オリジナル、
(B) OpenMPを適用、
(C) (B) + pragma simd、
(D) (C) + Flatモード, MCDRAMの利用
(E) (D) + メモリアライメント

4 まとめと今後の研究計画

前年度までに構築を進めたGPU向けAMR法フレームワークを拡張し、Xeon Phi対応に向けた拡張を進めている。Xeon Phi に有効な最適化の検討を行なった。

今後は、AMR法フレームワークのXeon Phi への対応に向けた要素技術の開発を進める。また、流体中を流れ成長する金属凝固計算へAMR法フレームワークを適用し、高解像度計算を実現する。最終的には、GPU/CPU/Xeon Phiへ対応したAMRフレームワークを完成させ、これを流体中を流れ成長する金属凝固計算へ適用し、GPU/CPU/Xeon Phiを搭載したスパコンで大規模計算を目指す。