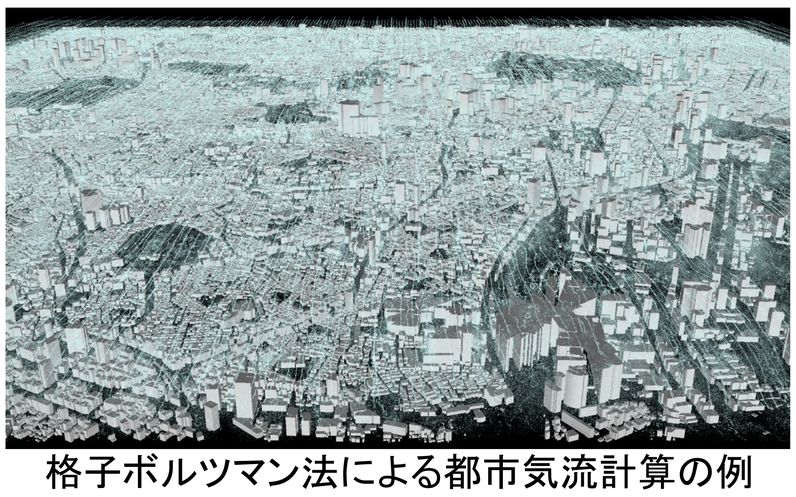
jh150047-NA25

下川辺隆史（東京工業大学）

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

1 研究背景と研究目的

従来のCPUと比べ、低消費電力で高性能である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。我々は、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導入できる構造格子用GPUフレームワーク開発している。本年度は、これを拡張し、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。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。そして、フレームワークを用い、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都市気流計算コードを従来よりも高生産に開発する。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を開発する技術を確立する。



格子ボルツマン法による都市気流計算の例

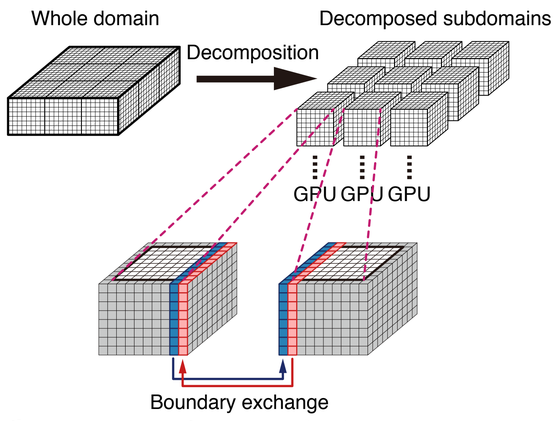
2 高生産フレームワークの概要

- ✓ 陽解法による直交格子上のステンシル計算を対象
- ✓ C++/CUDAを用いCPUおよびNVIDIA GPU に対応
- ✓ フレームワークはC++から利用可能。言語拡張や標準でないプログラミングモデルを利用しない。
  - → フレームワークおよびユーザコードの可搬性の向上　既存ライブラリと既存コードとの連携が可能
- ✓ ユーザはステンシル関数を定義し、フレームワークで実行

3 フレームワークの実装とプログラミングモデル

3.1 フレームワークの構造

- ✓ 複数GPU計算に対応
- ✓ フラットMPIによる並列化
- ✓ ステンシル計算はGPUあるいはOpenMP並列によりCPUで実行



3.2 直交格子データ構造

- ✓ データの大きさと位置を保持した構造
- ✓ 効率的な記述のために、このデータ構造 ETArray を利用する

```
// 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);
```

3.3 ステンシル計算の記述

- ✓ ある格子点を更新する関数（C++ファンクタ）をユーザが定義
- ✓ ArrayIndexによりインデックスを表現

```
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
  }
};
```

3.4 ステンシル計算の実行

- ✓ フレームワークを用いユーザ定義関数を全格子点へ適用
- ✓ 自動チューニングされ実行される（現在、GPUのみ対応）

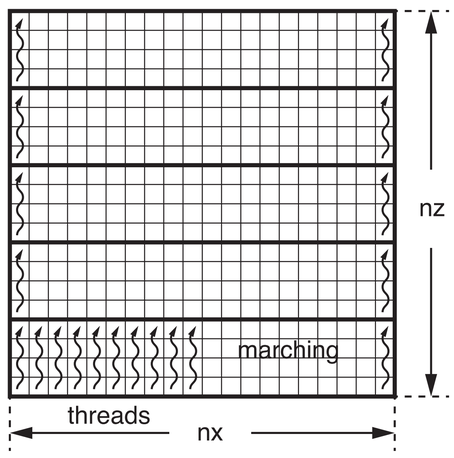
```
viewat(fn,inside,info) = funcf<float>(Diffusion3d(), ptr(f), idx(f),
                                      ce, cw, cn, cs, ct, cb, cc);
// f,fnはETArray, insideはRange3Dで関数適用範囲の指定
// viewat により fn のinside領域内の格子点のみ値を更新
// infoは自動チューニング用パラメータと結果を保持
// ptrはETArrayのrawポインタ、idxはインデックス表現 ArrayIndex を渡す
```

3.5 GPUにおける自動チューニング

CUDAのブロックをxy平面に配置し、各スレッドをある格子点 *(i ,j)* に割り当て、z方向にマーチングしながら計算する。x, y方向のスレッド数、z方向のマーチングする格子数が自動チューニングされる。

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



3.6 全要素への計算

- ✓ 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
- ✓ 左辺のETArrayの領域を適用範囲とする（明示的範囲指定は不要）
- ✓ 式からGPU/CPUコードの生成（式テンプレート）

```
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)));
// 拡散方程式の初期化の例。左辺の f は ETArray。posにより位置情報が渡される。
```

3.7 GPU間通信

- ✓ 指定された計算中心領域と ETArray データ自身の持つ大きさから通信を必要とする境界領域を自動判定

```
PBoundaryExchange<Range3D> exchange(inside, rank, connect);
// 境界でない計算中心領域を示す inside (Range3D型)
// MPIの自身と隣接ランク情報 rank, connect を渡す
exchange.append(&f); // 任意の型の ETArray を登録
exchange.transfer(); // 登録された ETArray の境界領域を隣接GPU間で交換
```

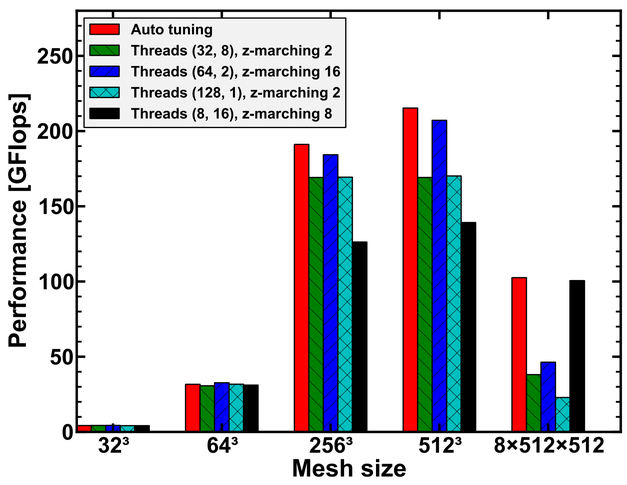
4 評価実験

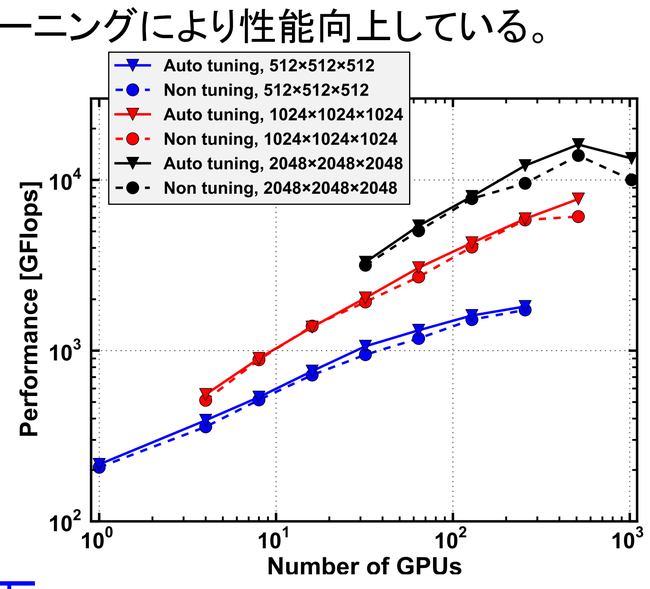
4.1 計算環境：東京工業大学 TSUBAME 2.5

- ✓ NVIDIA Tesla K20X GPU: 4224台
- ✓ 各ノード：3 GPUs, 2 Intel Xeon X5670 CPU, 2 Infiniband QDR

4.2 結果

フレームワークで実装された拡散方程式の性能評価。左図：単一GPUで自動チューニング機能を用いた場合と用いない場合の実行性能。右図：複数GPU計算の強スケーリング。自動チューニングにより性能向上している。





5 まとめと今後の研究計画

フレームワークに直交格子データ構造を追加し、ステンシル計算、全要素への計算へ対応した。今後は、計算による通信の隠蔽手法、効率的データ入出力、リダクション計算などを導入する。都市気流計算へフレームワークを適用し、GPU/CPUスパコン上での高性能な実アプリケーションの開発技術を確立する。