jh140013-NA09

下川辺隆史（東京工業大学）

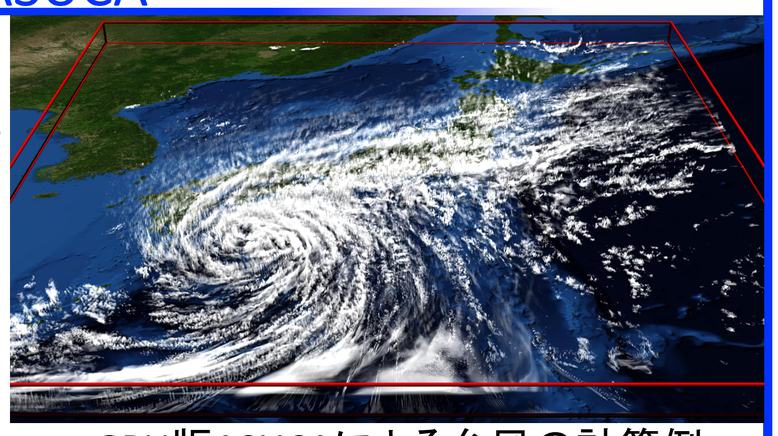
マルチGPUコンピューティング・フレームワークを用いた高精度気象計算コードの開発

1 研究背景と研究目的

短時間に精度良く計算する必要がある気象計算では、低消費電力で高性能なGPU計算の導入は高い注目を集めている。しかし、その利用の難しさ、生産性の低さが導入の障壁となっている。
我々は、気象庁が次期気象予報に向けて開発を進める気象計算コードASUCA全体を複数GPUで実行する研究に取り組んできた。
本研究では、GPU化を比較的簡単に行えるフレームワークを開発する。これを用い新たにASUCAをGPUへ実装し、GPUスパコンで高精細格子を用いることで気象予測としての精度向上を追求する。
開発を通し格子計算の大規模GPU化技術を確立する。

2 気象計算コードASUCA

- ✓ 気象庁で開発が進められている次世代高解像度気象計算コード
- ✓ 完全圧縮非静力学方程式系
  - フラックス形式
  - 一般座標系



GPU版ASUCAによる台風の計算例

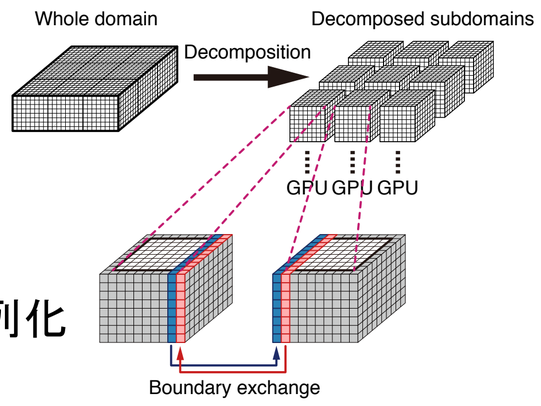
3 フレームワークの概要

- ✓ 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

4 フレームワークの実装とプログラミングモデル

1. フレームワークの構造

- ✓ 複数GPU計算に対応
- ✓ 最適化された並列化
  - ノード間並列：MPI
  - ノード内並列：OpenMP
- ✓ ユーザコード全体をOpenMPで並列化



2. ステンシル計算

- ✓ 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
- ✓ ArrayIndex3Dクラスにより座標（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__
  void operator()(const ArrayIndex3D &idx, // 格子点(i,j,k)を保持
    float ce, float cw, float cn, float cs,
    float ct, float cb, float cc, const float *f, float *fn) {
    fn[idx.ix()] = 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>()];
  }                                  /* (i,j,k-1)へアクセス */
};
```

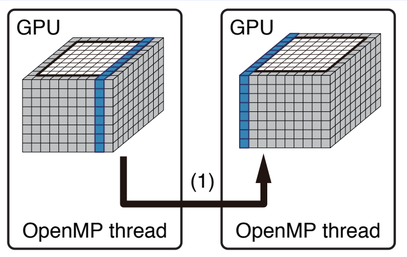
- ✓ フレームワークを用いユーザ定義関数を全格子点へ適用
- ✓ CPUあるいはGPU上で実行される

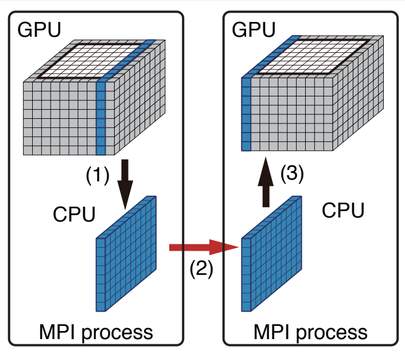
```
Loop3D loop3d(nx+2*mgnx, mgnx, mgnx,
              ny+2*mgny, mgny, mgny,  // ステンシル関数を適用する
              nz+2*mgnz, mgnz, mgnz); // 格子の範囲を指定
loop3d.run(Diffusion3d(), ce, cw, cn, cs, ct, cb, cc, f, fn);
```

3. GPU間通信

- ✓ BoundaryExchange により境界領域データを転送
  - ノード間：MPI通信
  - ノード内：GPUDIrect peer-to-peer 通信

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(array1); // 任意の型の配列を登録
exchange->append(array2); // 複数の配列を追加できる
exchange->transfer();     // 登録された全配列の境界領域を隣接GPU間で交換
```



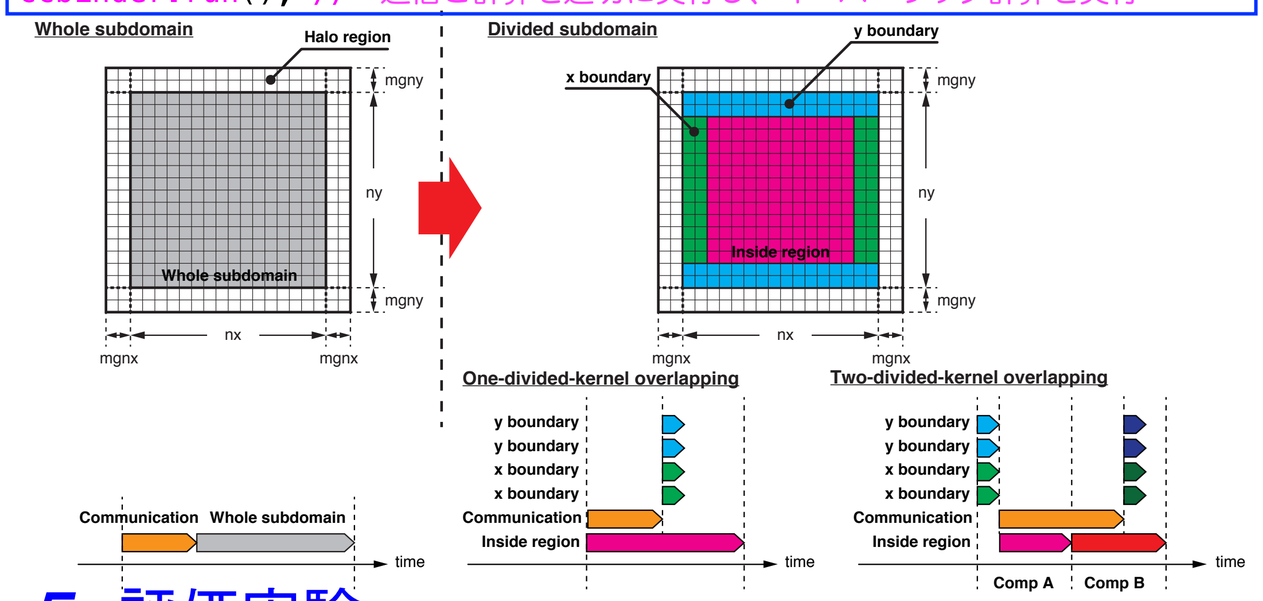


左：GPUDirectによるノード内GPU間通信
右：MPIによるノード間GPU間通信（ホストを経由する）

4. 計算による通信の隠蔽手法（オーバーラップ手法）

- ✓ ステンシル計算関数を境界領域と中心領域へ自動的に分割し、中心領域の計算で通信を隠蔽する

```
BoundaryExchange *exchange = domain.exchange();
exchange->append(f);
CompCommBinder<Loop3D> ccbinder(exchange);
ccbinder.set_post_func(&loop,      // loop領域を自動分割
    create_funcholder<Loop3D>(Diffusion3d(),
                              ce, cw, cn, cs ,ct, cb, cc, f, fn));
ccbinder.run(); //  通信と計算を適切に実行し、オーバーラップ計算を実行
```



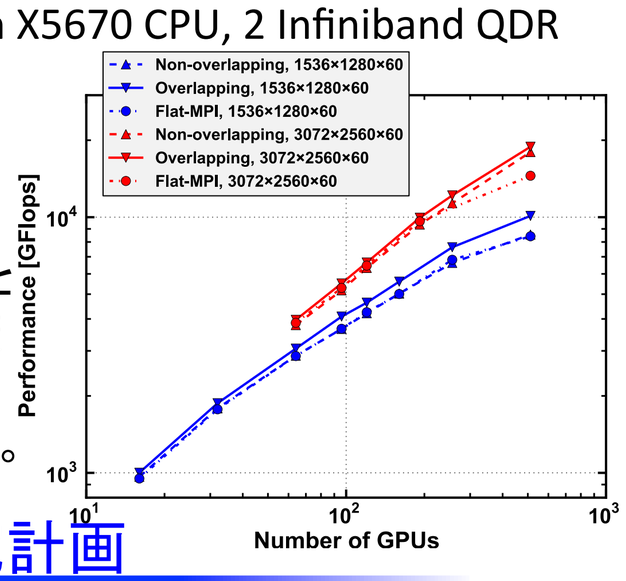
5 評価実験

1. 計算環境：TSUBAME2.5（東京工業大学）

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

2. 強スケーリング

フレームワークで実装されたASUCAの強スケーリング結果。オーバーラップ手法を用いることでGPU計算でGPU間通信を隠蔽し、30%程度性能向上し、512GPUで18.9TFlopsを達成した。



6 まとめと今後の研究計画

気象庁の開発する気象計算コードASUCAを格子計算用フレームワークを用いGPUへ実装した。今後は、実データを用いて高精細格子上で計算することで、気象予測としての精度向上を追求する。また、これを通して実アプリケーションにおける大規模GPU化技術を確立する。