核融合シミュレーションコードのGPUクラスタ向け最適化

2012/06/21

システム情報工学研究科
コンピュータサイエンス専攻2年
201120742 藤田 典久

研究背景

- GPUをアクセラレータとして利用するクラスタの増加
- スーパーコンピュータ ランキング top500 (2011/11)
  - Tianhe-1A (2位)
  - Nabulae (4位)
  - TSUBAME 2.0 (5位)
- 筑波大学 HA-PACS

研究背景

- 1台のノードに複数のGPUを搭載するマシンの増加
  - 筑波大学 HA-PACS: 4GPU/Node
  - 東京工業大学 TSUBAME 2.0: 3GPU/Node
- 複数のGPUを効率よく使用する必要
- GPUアプリケーション例
  - 合金の凝固過程における樹枝状結晶のシミュレーション
  - 次世代気象予報モデル ASUCA

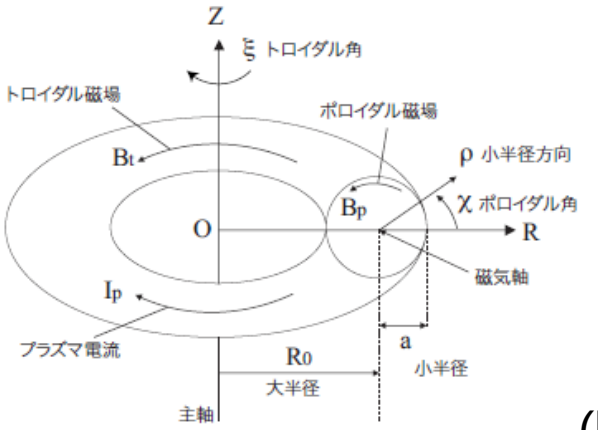
GT5D

- GT5D
  - 核融合シミュレーションを行うプログラム
  - トカマクプラズマ中の乱流現象を対象とする
  - 独立行政法人日本原子力研究開発機構で開発
  - とても大規模な計算を必要とする

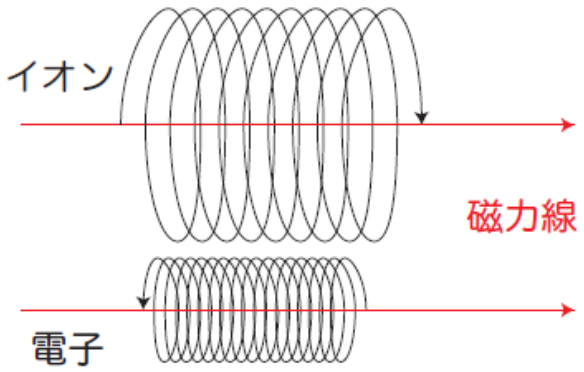
- ソフトウエア構成
  - Fortranで記述
  - OpenMP + MPIによるハイブリッド並列化

GT5Dの座標系

- (a) トーラス配位: 実空間変数(ρ, χ, ξ)
  - ポロイダル磁場を作る方法でトカマク、ヘリカルなどにわけられる
- (b) プラズマ粒子の運動: 速度空間変数(v, θ)
  - 荷電粒子は磁力線に巻きつくように運動する
  - 速度空間変数は$(v_{\|},\ v_{\perp},\ v_{\phi})$にわけられる
    - 磁力線周りを旋回する速度は乱流現象に比べて十分速いので消去できる(旋回平均)
  - 速度: $v = \sqrt{v_{\perp}^2 + v_{\|}^2}$　ピッチ角: $\tan\theta = v_{\perp}/v_{\|}$



(a)



(b)

目的

- GT5DのGPU化
  - 日本原子力研究開発機構との共同研究
  - GPU化のための事前評価
  - GPU化したコードの性能評価
- 1ノード複数GPUを活用するプログラムの開発
  - HA-PACSを利用する

CUDA

- Compute Unified Device Architectureの略語

- NVIDIA製GPUで汎用計算を行うための開発環境
  - コンパイラ
  - ドライバ
  - プロファイラ

CUDAプログラミング

- GPUで実行するプログラムコードのことを「カーネル」と呼ぶ

C/C++

- NVIDIAが提供するC/C++コンパイラ
- CUDAにおける一般的な開発環境

```
__global__
void add(int* out, int* x, int* y) {
  for (int i = 0; i < 10; ++i) {
    out[i] = x[i] + y[i];
  }
}
```

PGI CUDA Fortran

- PGIが提供するコンパイラ
- C/C++ CUDAコードを生成する
  - バックエンドはCUDAを使用
- 本研究ではこちらを使用

```
attributes(global) &
subroutine add(out, x, y)
  integer, intent(out) :: out(10)
  integer, intent(in) :: x(10), y(10)
  integer :: i

  do i = 1, 10
    out(i) = x(i) + y(i)
  end do
end subroutine add
```

CUDAプログラミング

- CPUとGPUはメモリ空間がわかれている
  - 直接、データの参照はできない
  - 通信や計算に必要なデータはCPUと転送する
    - 転送オーバーヘッドが存在

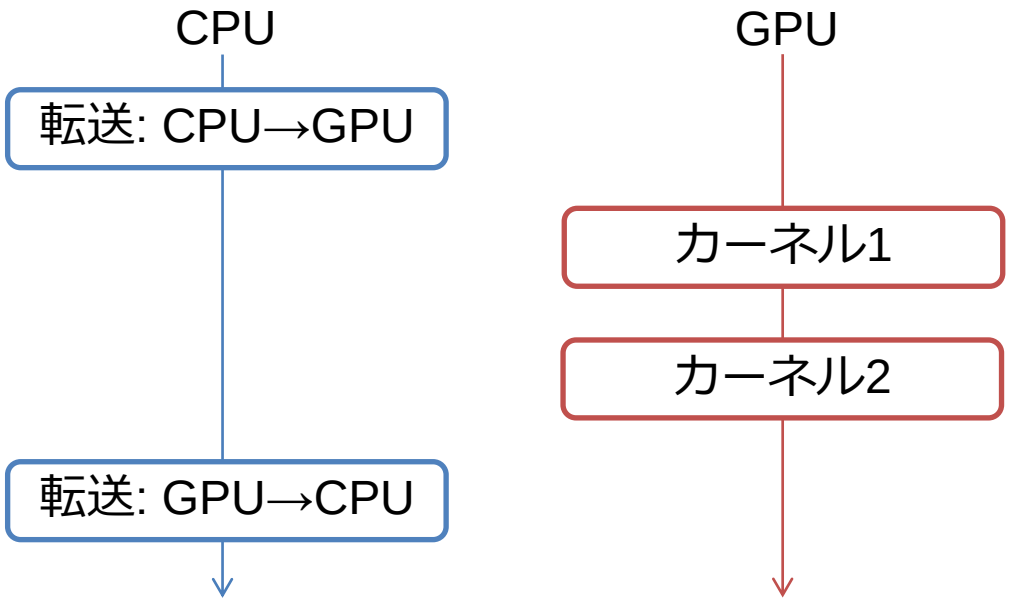


図2: 一般的なCUDAプログラムの流れ

計算機環境

HA-PACS	
CPU	Intel Xeon E5-2670@2.6GHz × 2
CPUメモリ	128GB
GPU	NVIDIA Tesla M2090 × 4
GPUメモリ	6GB/GPU
CUDA Toolkit	4.1
PGI Compiler	12.2
MPI	MVAPICH2 1.8
インターコネクト	Infiniband QDR × 2

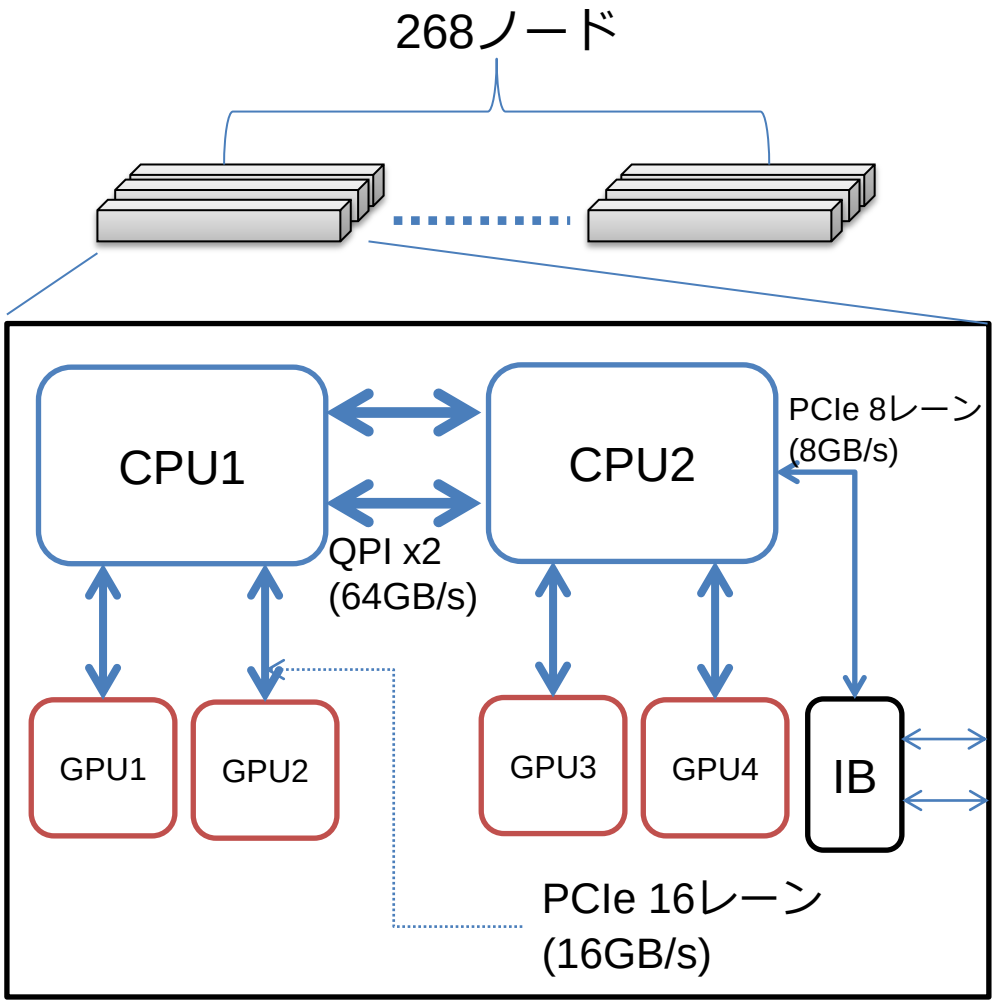


図3: クラスタの構成

GPU化の方針

- 時間発展部分をGPU化する
- main内に直接書かれているループがいくつかある
  - 各ループを関数としてGPU化する
    - timedev1 ～ timedev9
  - 個々のループはとても小さい
- 特に重い関数: l4dx_s, l4dx_r, l4dx_l, l4dx_nl
  - 通信を含まない、計算のみの関数

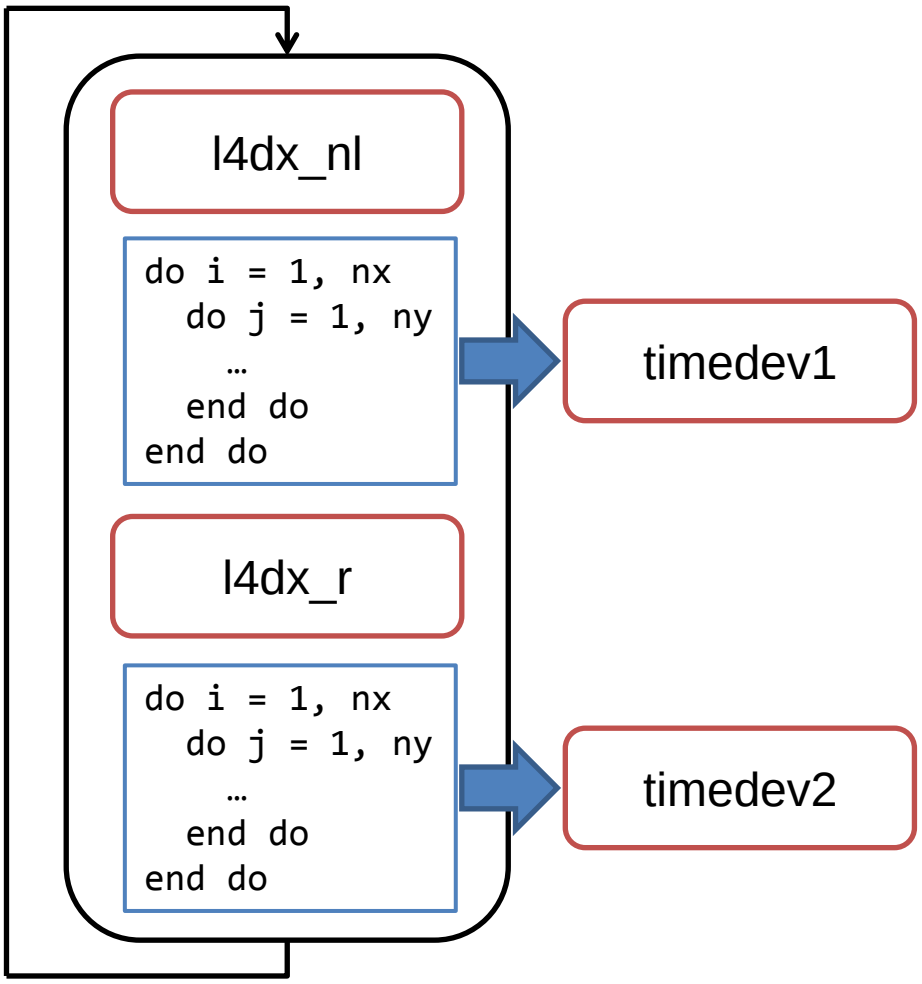


図4: GT5DのGPU化の概要図

データ移動の最適化

- 小さい関数でもGPU化を行う場合がある
  - GPU化された関数に隣接する関数
  - 例: timedev1～9, l4dx_nl
- データ移動の最適化のため
  - CPU～GPU間の通信速度はあまり速くない

データ移動の最適化

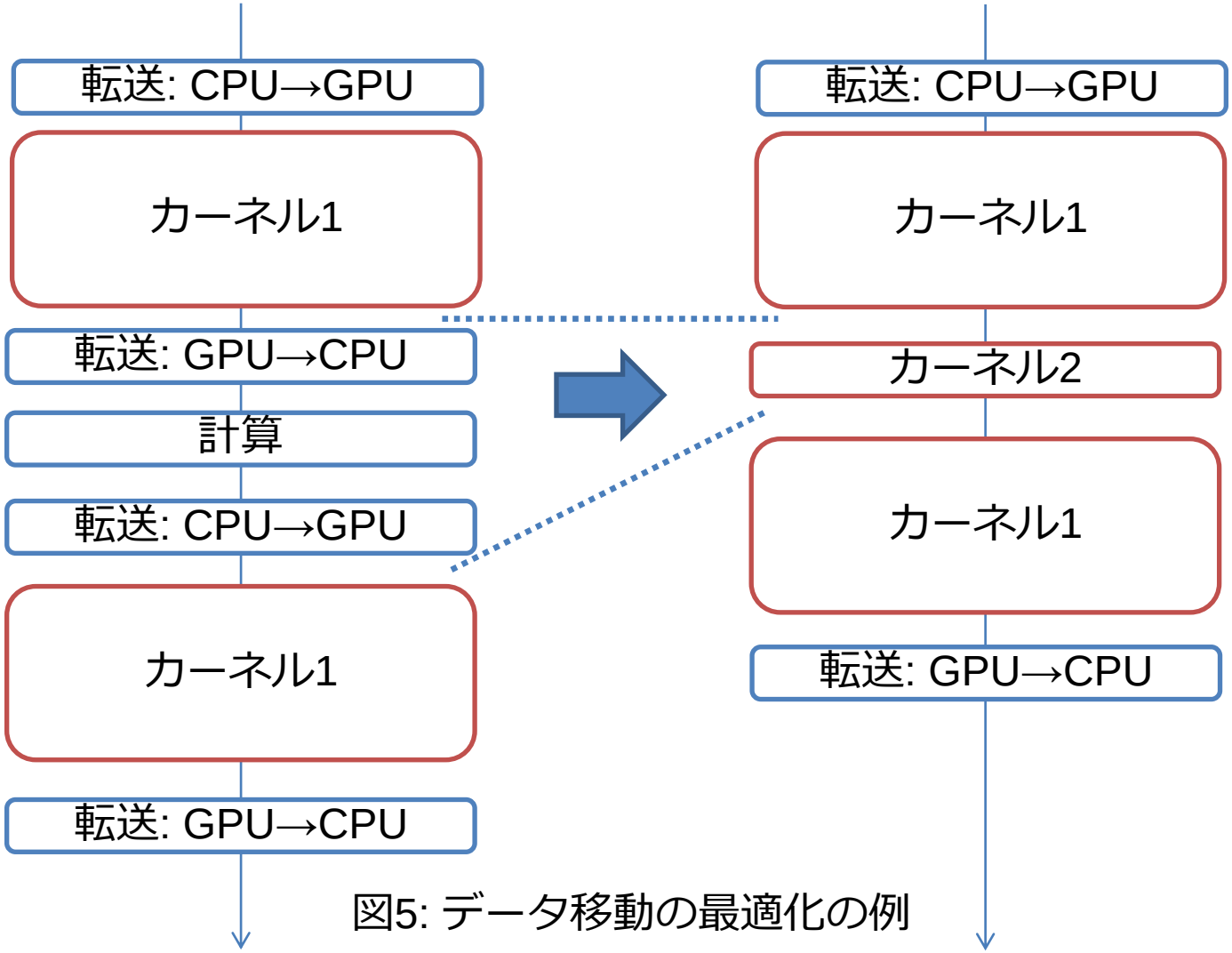


図5: データ移動の最適化の例

マルチGPU化の方針

- 1GPU = 1MPI Processの考え方を採用
  - 個々のプロセスが1つのGPUを制御
  - OpenMPによるスレッド並列も併用する
    - 4Core/Process

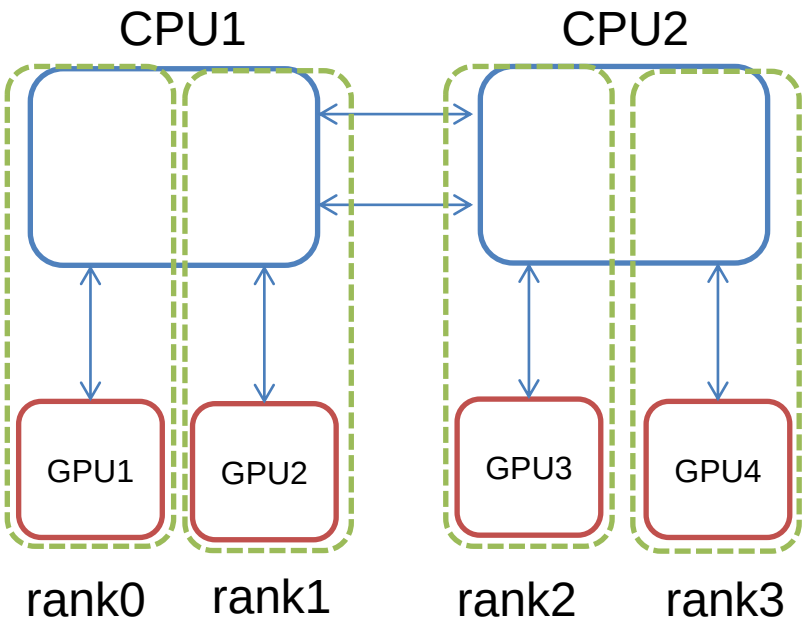


図6: MPIプロセスの割り当て

- 利点
  - 既存コードを流用可能
  - NUMAを意識しなくていい
- 欠点
  - GPU間のデータ交換にオーバーヘッドが発生する

性能評価の条件

- GT5D 問題サイズ (メッシュ分割数)
  - $(N_R, N_\zeta, N_Z, N_{v||}, N_\mu)$ = (64,64,64,64,4)
- 関数毎の性能評価の場合
  - 1プロセス、4スレッド、1GPU
- 時間発展全体の性能評価の場合
  - 1ノード、4プロセス、4GPU

関数毎の速度比較

関数	CPU[ms]	GPU[ms]	Speedup
l4dx_r	38.9	18.0	2.16
l4dx_s	47.0	33.0	1.42
l4dx_l	82.6	106.6	0.77
l4xd_nl	148.0	123.9	1.19

l4dx_r, s, nlの3つの関数では、
GPUの方がCPUよりも高速だが、
l4dx_lに関してはGPUの方が遅い結果が得られた。

関数毎の速度比較

関数	CPU[ms]	GPU[ms]	Speedup
timedev1	17.2	5.1	3.37
timedev2	18.2	8.3	2.19
timedev3	22.1	9.5	2.33
timedev4	22.4	10.5	2.13
timedev5	22.2	10.5	2.11
timedev6	21.8	6.7	3.25
timedev7	16.9	5.1	3.31
timedev8	26.4	8.3	3.18
timedev9	22.6	10.9	2.07

timedev1～timedev9の関数のGPU化では、
最大で3.37倍、平均で2.66倍の高速化が得られた。

全体の速度比較

	CPU[s]	GPU[s]	Speedup
時間発展部1回	7.05	5.81	1.21

GPU版はCPU版と比較して、1.21倍の高速化が
得られたものの、個々の関数の最適化や、
GPU化する範囲を広げる、
通信と計算のオーバーラップなど、
更なる高速化の余地が残っている。

まとめと今後の課題

- 核融合シミュレーションコードGT5Dの時間発展部のGPU化を行った
- 複数GPUを利用するコードが動作している
- 各関数の最適化や、袖領域の通信を行う際の計算のオーバーラップが課題