核融合シミュレーションコードのGPU化と並列言語XcalableMPによる実装

朴　泰祐
筑波大学計算科学研究センター
H25年度学際共同利用：NUFUSEプロジェクト

研究の背景と枠組み

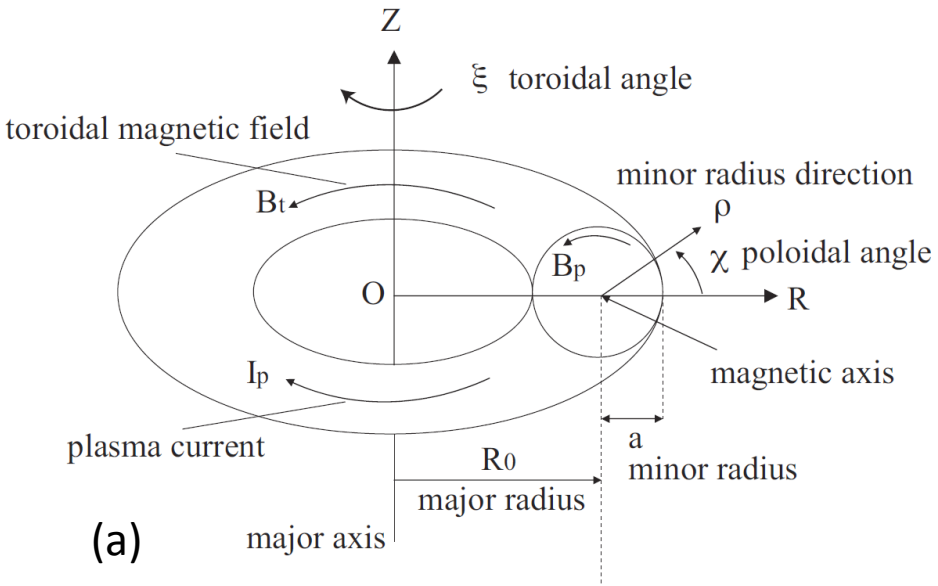
- JSPS-G8 (多国間国際研究協力事業)における異分野・多国籍共同研究
  “Nuclear Fusion Simulations at Exascale” (NuFuSE)
- 参加国・機関
  - 日本、英国、フランス、ドイツ、米国、ロシア
  - 筑波大、原研、核融合研
- 目的
  - 各機関で開発中の大規模核融合シミュレーションコードの超並列化・高機能化・高速化を連携研究の枠組みで行う
  - 京をはじめとする国内のスパコンリソースを使った開発とポーティングを行う
- 研究メンバー（学際共同利用）
  - 筑波大：朴泰祐、奴賀秀男、藤田典久、津金佳祐
  - 原研：井戸村泰宏、矢木雅敏、前山伸也、内藤裕志
  - 核融合研：中島徳嘉、藤堂泰、渡邉智彦、坂上仁志
  - プリンストン大学：William Tang, Bei Wang, Stephane Ethier
  - エジンバラ大学：Adrian Jackson, Toni Collis

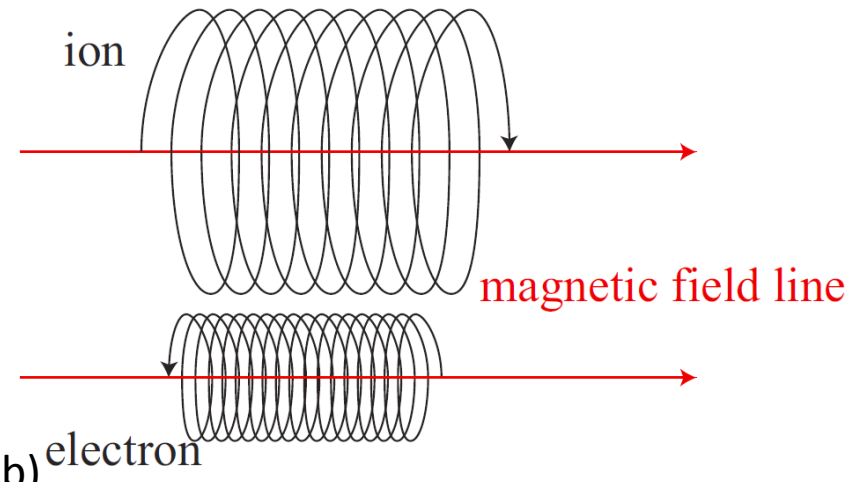
H25学際共同利用における研究

- 原研で開発中の核融合シミュレーションコードGT5DのGPUによる高速化
  - 筑波大・原研の共同研究
  - HA-PACSを利用
  - オリジナルコード（OpenMP + MPI, Fortran）をPGI-CUDA Fortran + OpenMP + MPIによりGPU化
- プリンストン大学で開発中の核融合シミュレーションコードGTC-PのXcalableMP化
  - 筑波大・プリンストン大の共同研究
  - 学際共同利用の別プロジェクトであるXcalableMP開発チームと連携
    中尾昌広（理研AICS）、佐藤三久（筑波大/理研AICS）
  - XcalableMP言語が global view と local view という2つのデータ分散モデルをサポートすることをPICコードに適用
  - 将来のPICコードの開発手法とXcalableMPの実応用の両面を研究
- いずれもプロダクトランではなくコード開発と性能評価が目的

GT5D

- GT5D
  - 核融合炉のシミュレーターであり、炉におけるプラズマ乱流を記述する
  - 日本原子力研究開発機構 (JAEA) で開発されている
- GT5Dの計算はメモリ帯域によって律速
  - ステンシル系の計算でありGPUに適する
    (GPUメモリ帯域: 200~ GB/s)

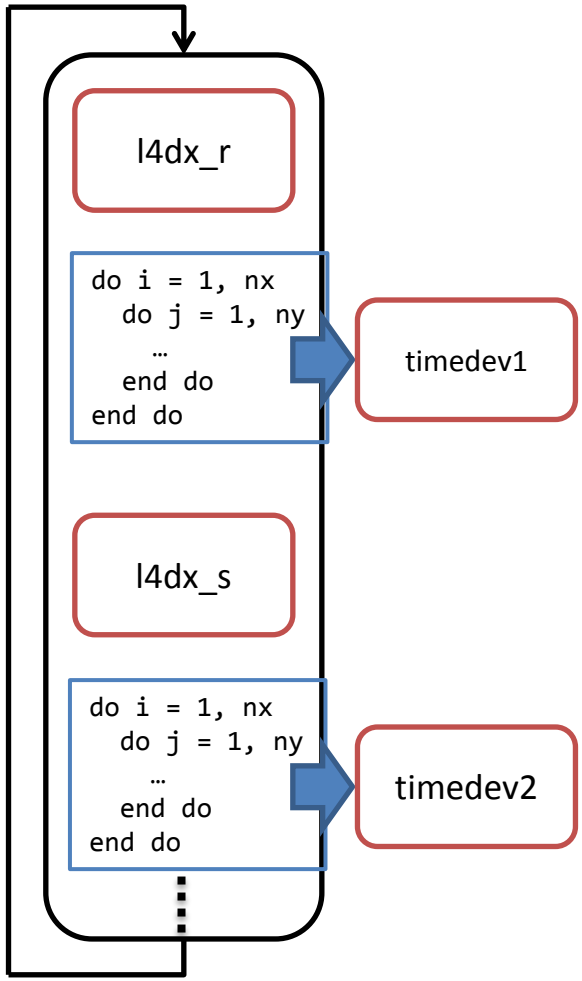




GT5DのGPU対応

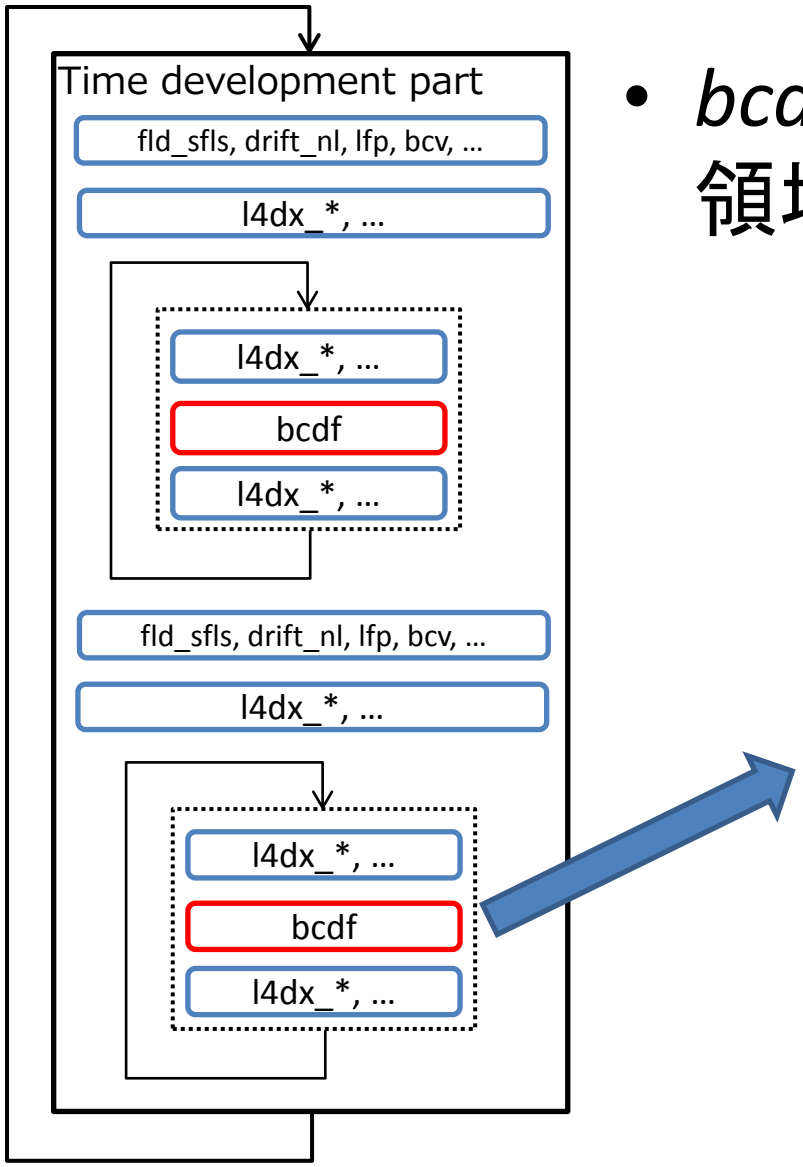
- GT5DはFortranで書かれている
  - しかしながら、CUDA ToolkitはFortranコンパイラを提供しない
- GPUを利用するためにPGI CUDA Fortranを使用する
  - 全体のコードの書き換えを防ぎ、CPU部のコードを再利用できる
- GT5Dの時間発展部分をGPU化し、MPI通信を除く部分をGPU化を完了
  - 計算に必要なデータはGPUに置かれたまま
  - MPI通信に必要なデータのみ移動される

GPUカーネルの作成

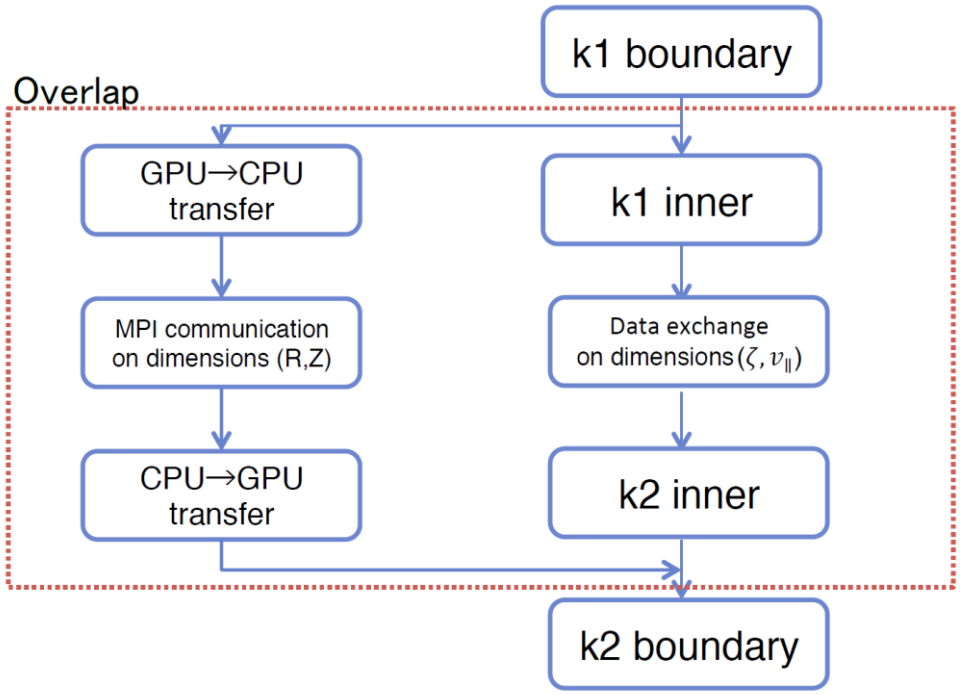


- CPUの関数に対応するGPUカーネルを作成する
  - 1対1に移植を行う
- いくつかのループは関数として分離されていない
  - *timedev1* ～ *timedev9* と命名
  - 関数の周囲にあり、それらに対してもGPUカーネルを作成する
  - データ移動の最適化のため

通信オーバーラップ



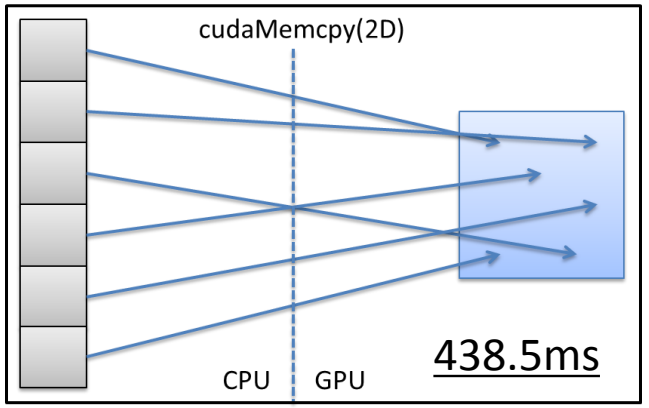
- *bcdf* はステンシル計算の袖領域の交換を行う関数



データアンパックの最適化

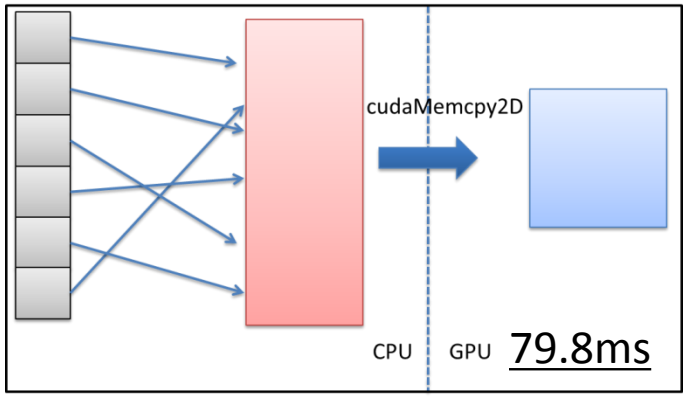
・Method A:
各チャンクに対して
cudaMemcpyを行う

A)



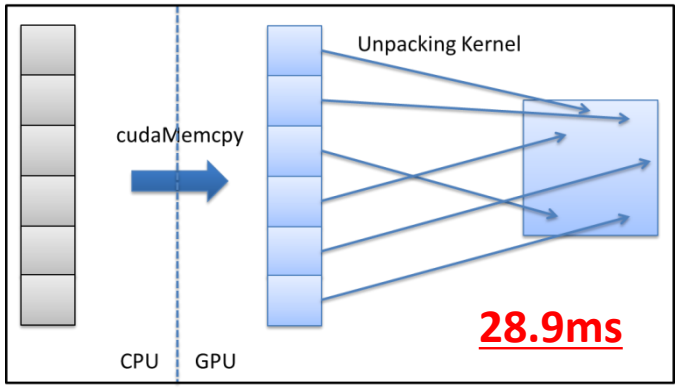
・Method B:
CPU上でデータの並べ替えを
行い、GPUへ転送する

B)



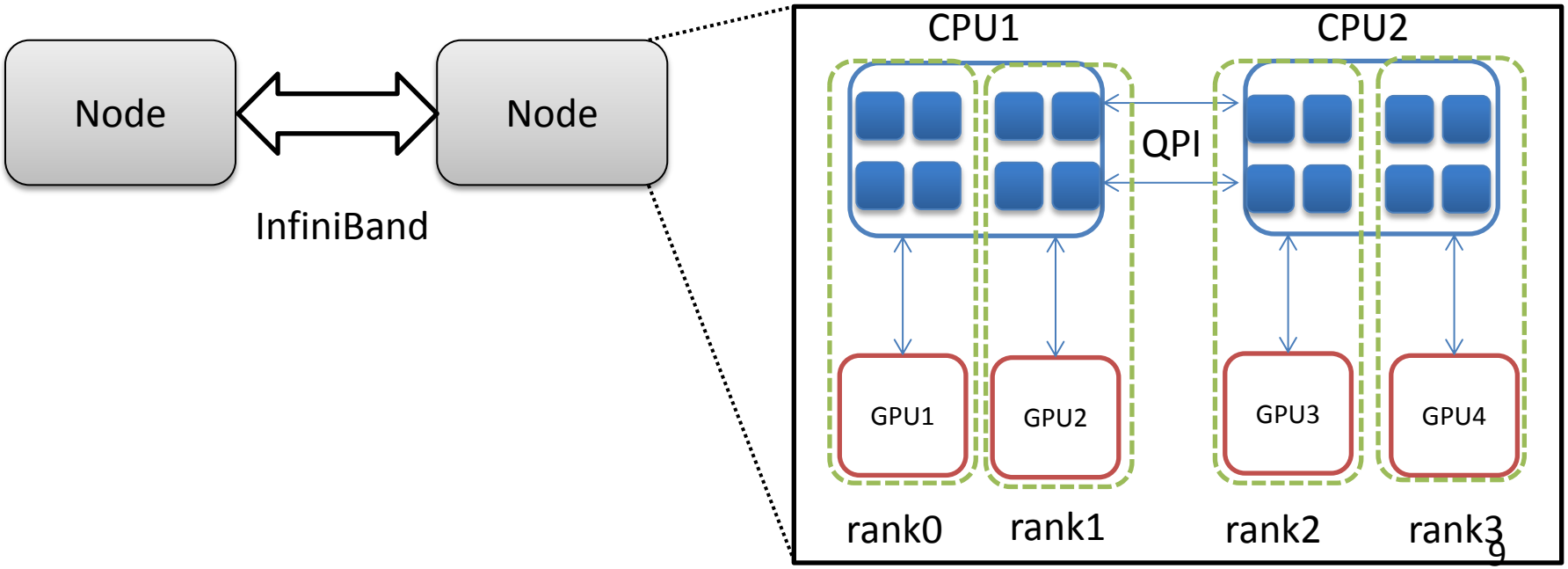
・Method C:
バッファをGPUへ転送し、
それからGPU上でデータの
並び換えを行う

C)



CPUとGPUの割り当て方法

- HA-PACSを利用して性能測定を行う
- プロセスベースのマルチGPU利用
  - それぞれのプロセスが別々のGPUを扱う
  - 1 MPI Process = 1 GPU = 4 CPU Threads (OpenMP)

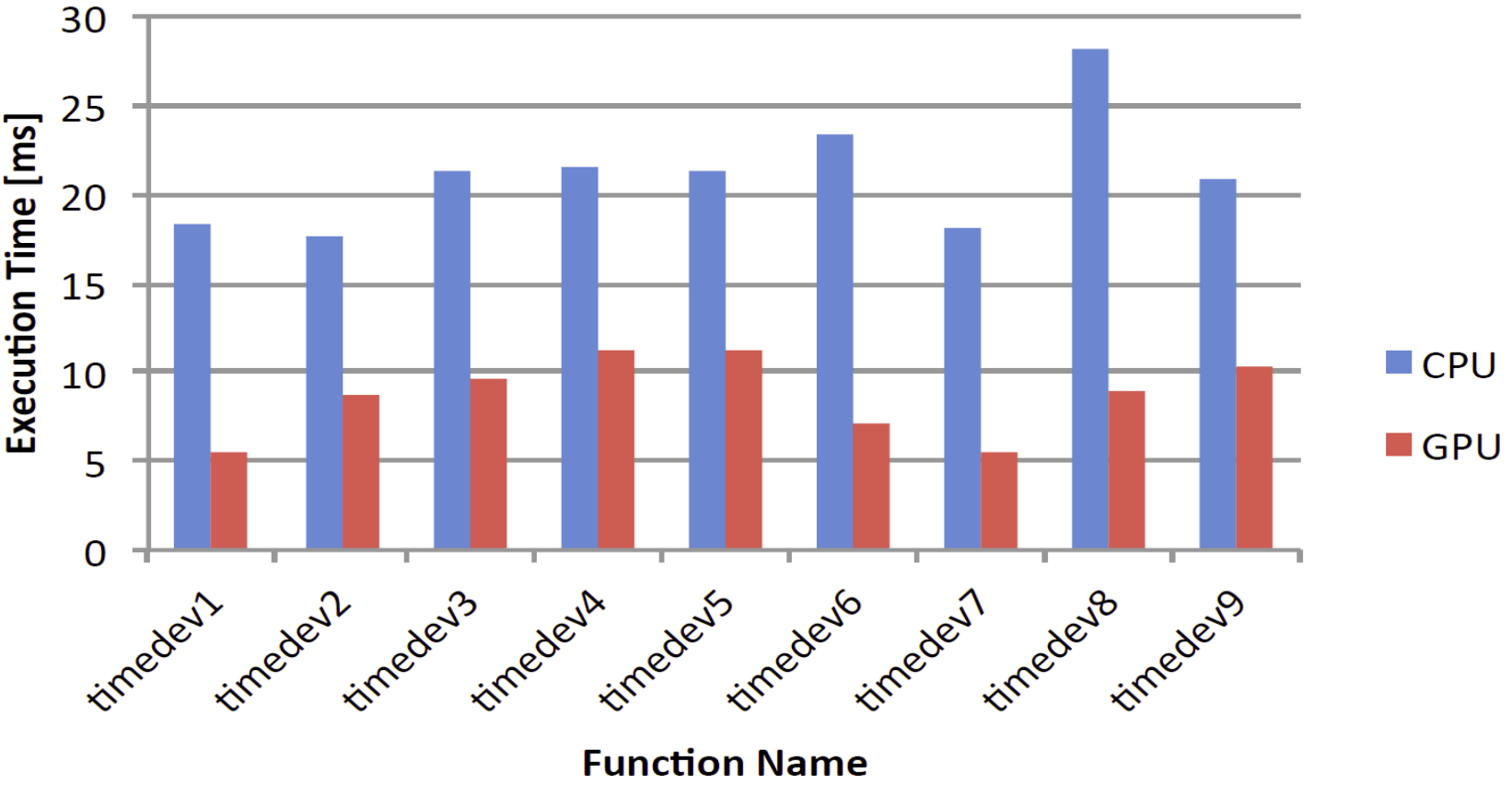


関数単位の性能評価

- 関数単位の性能評価を以下のパラメータで行う
  - 計算のみの比較で試験用プログラムを作成し実施
  - 1GPUのみ利用し、MPIによる領域分割はない

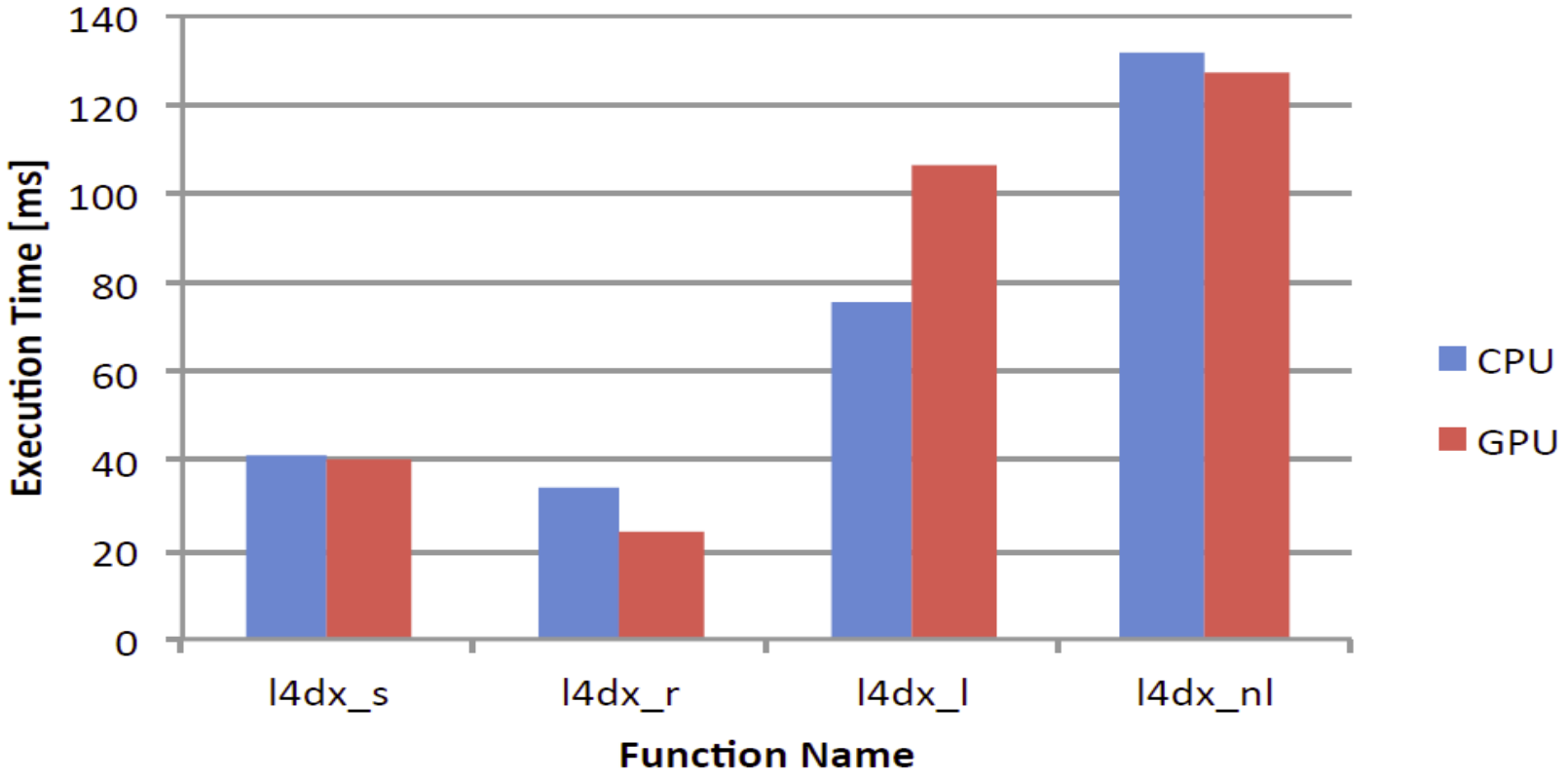
パラメータ	
ノード数	1
MPIプロセス数	1
MPI 分割 (n_R, n_Z, n_m)	なし
CPUスレッド数	4
ノードあたりのGPU数	1
全体のGT5Dメッシュサイズ $(N_R, N_z, N_Z, N_v, N_m)$	(128,128,128,128,4)

timedev系関数の性能評価



- timedev1関数のGPUでの実行はCPUでの実行よりも3.35倍高速である.
  全体の平均speedupは2.58倍である
- これらの関数はメモリ帯域に律速される

l4dx系関数の性能評価



- l4dx_r関数はGPUでの実行の方がCPUでの実行よりも1.43倍高速である
  しかしながら，l4dx_l関数はGPUで実行する方が遅い (0.71倍)
- timedev系関数よりも複雑であり，計算性能に律速
- 時間発展部のおよそ35%がl4dx_s関数であり，
  l4dx_s関数が全体の性能のボトルネックとなる

時間発展部全体の性能評価

- 時間発展部全体の性能評価を以下の条件で行う
  - ノードにある全てのGPUを利用する
  - 16ノード (256 core), 64GPUを利用

	計算時間[s]	Speedup
CPU only	15.12	-----
GPU without overlap	12.89	1.17
GPU with overlap	7.90	**1.91**

* “overlap” はbcdf関数におけるオーバーラップを表す

XcalableMP (XMP)

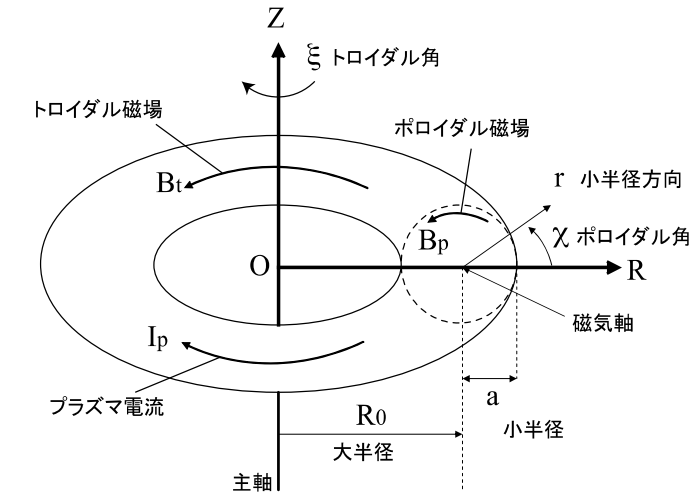
- 分散メモリ環境を対象とした並列言語
  - 既存言語(C, Fortran)の拡張
- OpenMPのような指示文形式
  - 明示的な並列化と通信

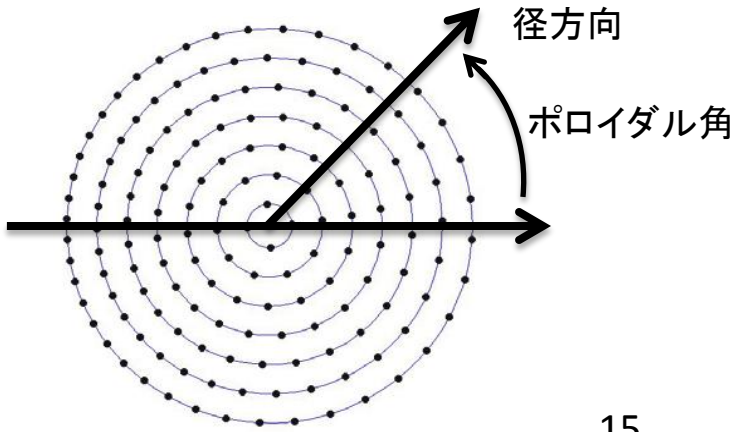
  ⇒逐次版と同等のプログラム
- 2種類のプログラミングモデルを実装
  - グローバルビュー
    - 指示文によりデータ分散, 並列実行, 通信・同期
    - ステンシル演算のような, 静的な領域分割に適している
  - ローカルビュー
    - 各ノードが持つローカルデータに対して通信を行うモデル
    - ローカルデータサイズが動的に変化する場合等に適している

GTC-P

- 磁場閉じ込め型核融合プラズマの乱流現象解析を行うコード
  - プリンストン大学が開発
  - 3次元PICコード
  - MPI+OpenMPによるハイブリッド並列化
- MPIによる領域分割
  - トロイダル方向, 径方向, 粒子数
- ポロイダル断面上の格子点
  - 等ポロイダル角ではなく, 等径方向距離

⇒演算量の均等化のために径方向の
領域分割は非均等に行われる

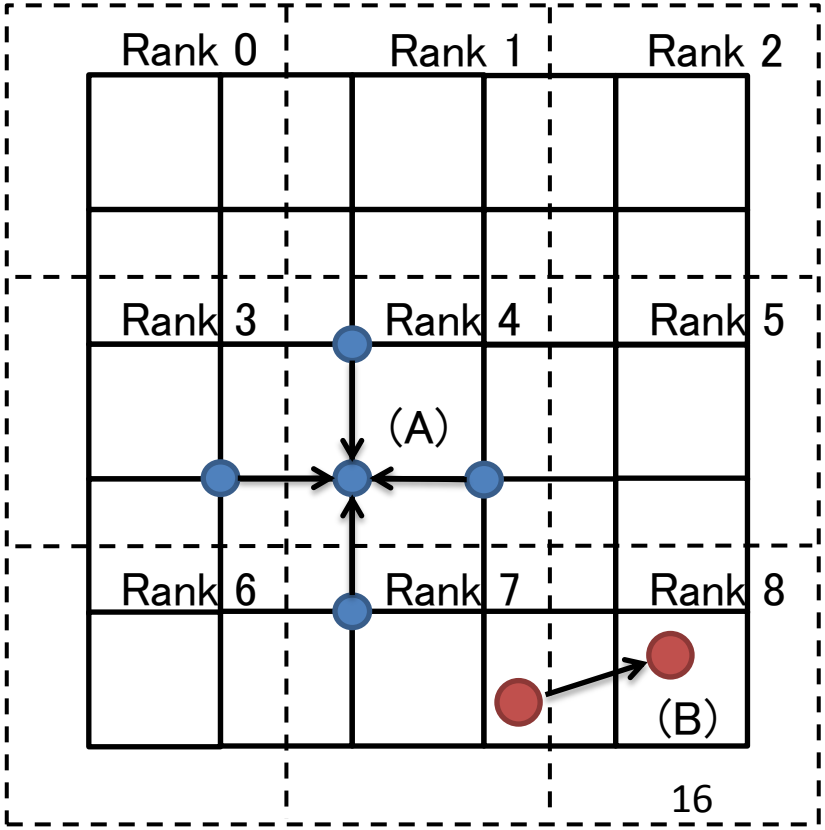




PICコード

- PIC(Particle-in-Cell)法
  - 核融合プラズマ分野における粒子シミュレーション手法の一種
  - 場の計算を行う計算格子と粒子軌道演算で構成
    - 場の計算は静的な領域分割
      - グローバルビュー
    - 粒子演算は動的に演算量が変化
      - ローカルビュー

⇒二つのモデルを組み合わせた
ハイブリッドビューが必要



(A) 場の計算
(B) 粒子軌道演算

ハイブリッドビューによる実装

- XMP宣言部の実装例
  - 径方向の分割はプロセス毎に分割サイズが異なる
  - 非均等なブロックサイズによる分割が必要

  ⇒ gblockを使用
    - ユーザ定義のブロック分割

- MPI実装では，動的にブロックサイズを決定
  - XMPは，動的な2次元分散をする事が現在不可能

  ⇒例同様，静的に記述

```
#define n_t        2 /* トロイダル方向の分割数 */
#define n_r        4 /* 径方向の分割数 */
#define n_rp       2 /* 粒子数の分割数 */
#define nloc_over_all          107722

real phitmp_g  [nloc_over_all][2 * n_t];
int  b [n_r * n_rp]
  ={10967,10967,14086,14086,16164,16164,12644,12644};
    /* gblock分散時の各プロセスのブロックサイズ */

#pragma xmp nodes P2(n_r * n_rp, n_t)
    /* ノードの分割数の指定 */
#pragma xmp template T(0:nloc_over_all-1, 0:2*n_t-1)
    /* テンプレート長の指定 */
#pragma xmp distribute T(gblock(b), block) onto P2
    /* テンプレートの分割方法の指定 */
#pragma xmp align phitmp_g     [i][j]    with T(i, j)
    /* 配列とテンプレートの対応 */
#pragma xmp shadow phitmp_g    [0][1:0]
    /* 袖領域の確保 */
```

通信部分の実装

- ローカルビューによる実装
  - MPI_Sendrecvをcoarrayで記述

```
double *sendr, *recvl;

for(i=0;i<nloc_over;i++)
 sendr[i]=phitmp[i*(mzeta+1)+mzeta];

MPI_Sendrecv(sendr,nloc_over,double,right_pe,
        isendtag,recvl,nloc_over,double,left_pe,
        irecvtag,toroidal_comm,&istatus);
```

MPI

```
double Xsendr[nloc_over],Xrecvl[nloc_over];
#pragma xmp coarray Xrecvl:[*]

for(i=0;i<nloc_over;i++)
 Xsendr[i]=phitmp[i*(mzeta+1)+mzeta];

Xrecvl[0:nloc_over]:[right_pe]=Xsendr[0:nloc_over];
xmp_sync_all(NULL);
```

XMP-localview

- ハイブリッドビューによる実装
  - 分散配列により近傍格子間の通信を指示文1行へ

```
#pragma xmp reflect (phitmp) width (0,/periodic/1:0)
```

XMP-hybridview

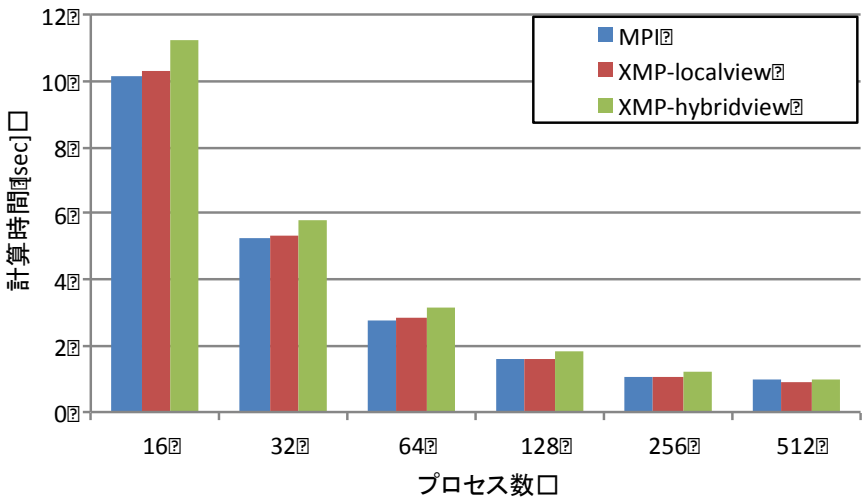
実験環境

- HA-PACS
  - 1ノード16MPIプロセス, 最大32ノード512MPIプロセスで計測

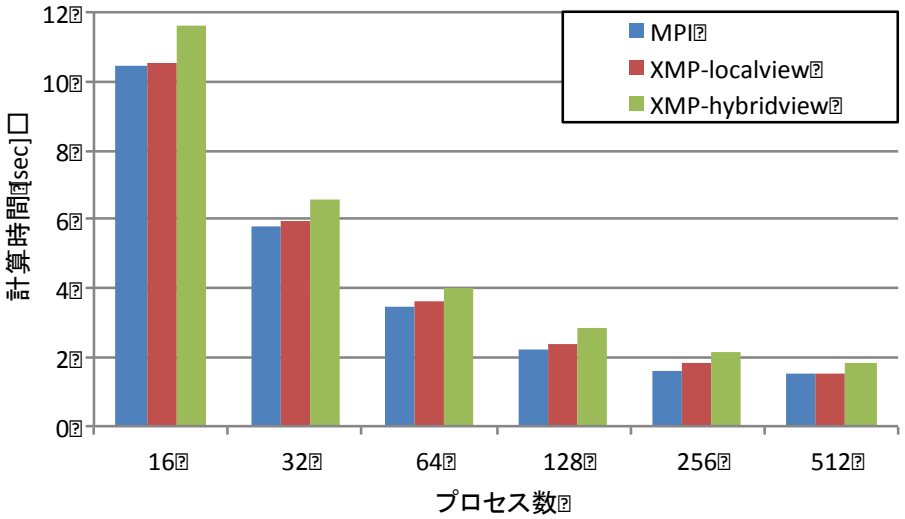
CPU	Intel Xeon E5-2670 x2 (2.6GHz)
	CPU (8 cores/CPU) x2 = 16 cores
Memory	128 GB, DDR3 1600 MHz
Interconnect	Infiniband QDR 4 Lanes x2 Rails
OS	CentOS 6.1
C Compiler	GCC 4.4.7
MPI	MVAPICH2 2.0

- トロイダル方向, 径方向, 粒子数の分割数
  - 各分割数でStrong scaling, Weak scalingを計測
- 1次元の分割数を変動させ, 他2次元は2x2に固定
  - トロイダル方向のみGTC-Pの制約上, 演算量を揃えるため2x4

評価 (Strong scaling)



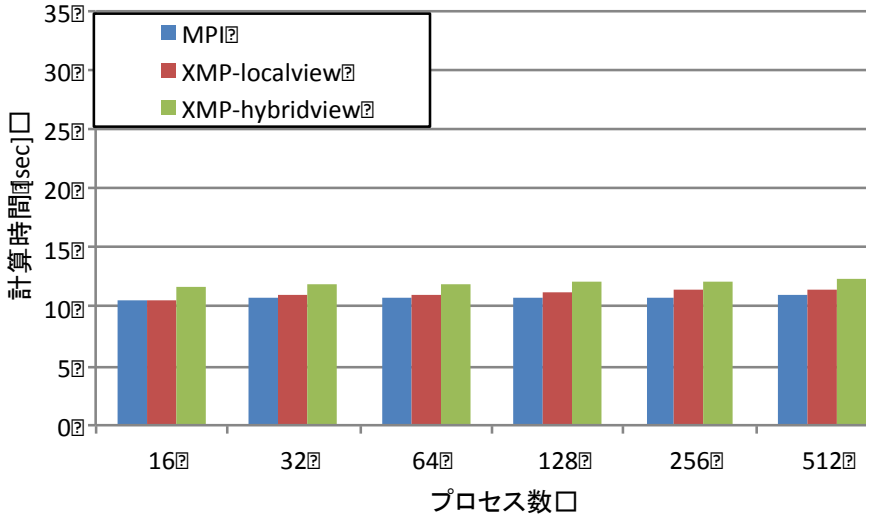
径方向分割 (2xNx2)



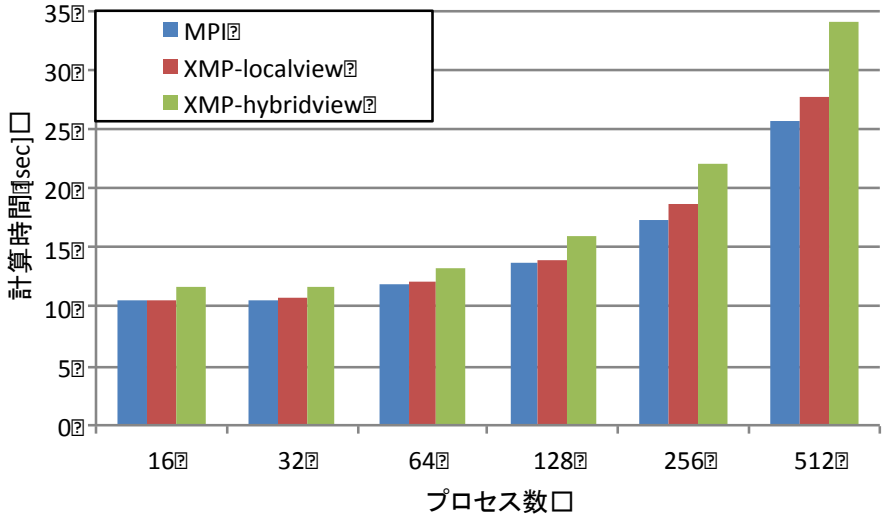
粒子数分割 (2x2xN)

- XMP-localviewとMPI実装は, ほぼ同等の性能
- XMP-hybridviewは5 ~ 25%の性能低下

評価 (Weak scaling)



粒子数分割 (2x2xN)



径方向分割 (2xNx2)

- XMP-localviewとMPI実装は，ほぼ同等の性能
- XMP-hybridviewは8 ~ 25%の性能低下
  - トロイダル領域分割は粒子数分割と同様の傾向のため省略
- 径方向分割は他の分割方法と比較して，全体的に性能が低い

考察 (Weak scaling)

- 各プロセスの実行時間の調査 (演算する格子点数)
  - 通信時間を除く

径方向分割

プロセス数	Min	Max
16	9.902 (10967)	10.144 (16164)
32	9.905 (12104)	10.405 (24200)
64	9.916 (14130)	11.354 (33462)
128	9.974 (17422)	12.798 (74745)
256	10.197 (23198)	14.737 (138127)
512	10.670 (34522)	18.165 (276110)

粒子数分割

プロセス数	Min	Max
16	10.236 (19805)	10.399 (19916)
32	10.241 (19805)	10.425 (19916)
64	10.238 (19805)	10.431 (19916)
128	10.220 (19805)	10.444 (19916)
256	10.225 (19805)	10.447 (19916)
512	10.223 (19805)	10.462 (19916)

  - 径方向分割時のロードバランスが悪い
    - プロセス当たりの演算する格子点数に大きな差があるため

⇒GTC-P自体の問題

生産性

- ローカルビューによる実装
  - coarray記法による配列代入文形式
- ハイブリッドビューによる実装
  - reflect指示文による1行での袖領域通信
  - MPI版と比較してシリアル版からの変更点が少ない

Serial	MPI	XMP-hybridview
4386	5319	5052

ソースコード(行数)

ℒいずれの実装も見通しが良く生産性が高い

対外発表

- N. Fujita, H. Nuga, T. Boku, Y. Idomura, "Nuclear Fusion Simulation Code Optimization and Performance Evaluation on GPU Clusters", Proc. of PDSEC2014 (with IPDPS2014), Phoenix, May 2014.
- K. Tsugane, H. Nuga, T. Boku, H. Murai, M. Sato, W. Tang, B. Wang, “Hybrid-view Programming of Nuclear Fusion Simulation Code in the PGAS Parallel Programming Language XcalableMP”, Proc. of ICPADS2014, Hsinchu, Dec. 2014 (to appear).
- 津金 佳祐，奴賀 秀男，朴 泰祐，村井 均，佐藤 三久，Willia Tang. "並列言語XcalableMPによる核融合シミュレーションコードの実装と評価", 2014-HPC-145, 2014.
- 奴賀 秀男, 朴 泰祐, 藤田 典久, 中尾 昌広, 佐藤 三久, William Tang, "並列言語XcalableMPによる核融合シミュレーションコードの開発", 2013-HPC-142, 2013.
- 藤田 典久, 奴賀 秀男, 朴 泰祐, 井戸村 泰宏, "GPUクラスタHA-PACSにおける核融合シミュレーションコードの性能評価", 2013-HPC-140, 2013.
- 藤田 典久, 奴賀 秀男, 朴 泰祐, 井戸村 泰宏, "GPUクラスタにおける核融合シミュレーションコードの実装", 2013-HPC-138, 2013.

まとめ

- G8多国籍国際協力研究フレームワークにおいて、GPU及び超並列向け言語を大規模核融合シミュレーションに適用
- 核融合コードのプロダクトランではなく、同分野研究者と計算機科学者の共同研究として、コード開発と性能評価を行った
- GT5DのGPU化では、HA-PACS (base cluster)でCPUのみの場合に対し約2倍の性能向上
- GTC-PのXcalableMP実装では、コード記述量を減らし、MPIや local view only実装に比べ見通しのよいコードを、非常に小さい性能低下で実現可能
- 今後、佐藤グループで開発中のXcalableACCによるGPU化を進める