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

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
指導教員 朴 泰祐
2012 年 6 月 21 日

1 はじめに

従来は 3D グラフィックスを描画するための装置としてのみしか利用されていなかった GPU に，汎用的な計算をさせる General Purpose computing on GPU (GPGPU) が高性能計算分野で脚光を浴びている．GPU は，CPU と比較して高い並列演算性能とメモリバンド幅を持ち，NVIDIA 社の Tesla M2090 では，倍精度演算性能で 665 GFLOPS，メモリバンド幅で 177GB/sec に達する．

近年の GPGPU の普及と，1 台のマシンが接続できる PCI Express のレーン数の増加に伴い，1 台のマシンに 3 台や 4 台の GPU を搭載するシステムも登場しているため，効率的なメモリ転送の戦略や，GPU の制御方法が重要視されている [1, 2]．また，計算を全て GPU に任せ，CPU は GPU 制御やノード間通信のみを行う計算モデルだけでなく，GPU が計算を行ないつつ CPU も計算を行う協調計算型のモデルも用いられている．

GPU 単体では，プログラムの実行やデータ転送といった動作をすることはできない．CPU と GPU 間は PCI Express インターフェイスによって接続されており，CPU からの命令発行やデータ転送は PCI Express を通じて行われる．PCI Express Gen2 16 レーンの帯域は上り 8GB/sec，下り 8GB/sec の全二重通信であり，CPU のメモリ帯域や，GPU のメモリ帯域と比較して細いため，ボトルネックとなりやすい．また，CPU から GPU への命令も PCI Express を通じて送られるため，GPU の操作はオーバーヘッドを伴う．

本研究では，核融合シミュレーション用プログラム GT5D (conservative global gyrokinetic toroidal full-f five-dimensional Vlasov simulation) [3] の GPU 化のための事前評価と，GPU 化を目的とする．GT5D は，独立行政法人日本原子力研究開発機構で開発されたプログラムであり，Fortran で記述されている．MPI と OpenMP によるハイブリッド並列化が既に成されており，それらを基に GPU 化を進める．また，GPU クラスタを効率よく利用するために，1 ノード複数 GPU の利用を行う．

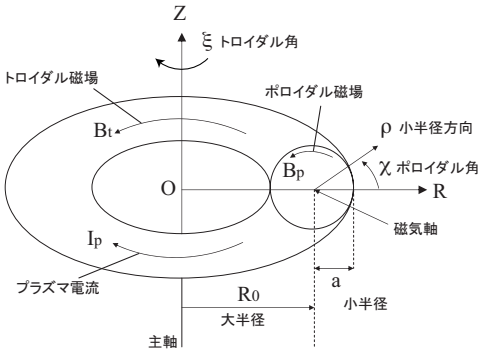

図 1: GT5D におけるトーラス配位．

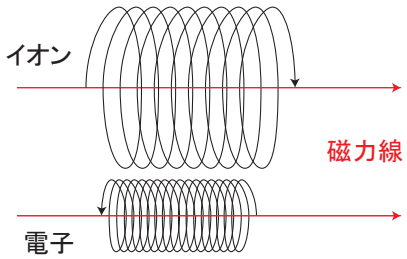

図 2: プラズマ粒子の運動．

2 GT5D

GT5D は，旋回平均された速度分布関数の時間発展を計算するコードであり，トカマクプラズマ中の乱流現象を記述する．プラズマ中の乱流現象は，プラズマ輸送などのより大きな時間・空間スケールの現象にも影響を及ぼし，例えば，異常輸送や，乱流駆動不安定性などの原因となる．

GT5D の扱う空間を図 1 と図 2 で示す．GT5D はトーラス配位の実空間 3 次元 (ρ, χ, ξ)(図 1) と，粒子の速度空間 2 次元 $(v_{\parallel}, v_{\perp})$ を位相空間変数としている．ここで，$v_{\parallel}, v_{\perp}$ はそれぞれ磁力線に平行方向の速度，垂直方向の速度である．荷電粒子は磁力線に巻き付くように運動するが，磁力線を旋回する速度は GT5D が対象とする乱流現象に比べて十分速い．このため，旋回平均によって速度空間変数から旋回位相を消去できる．

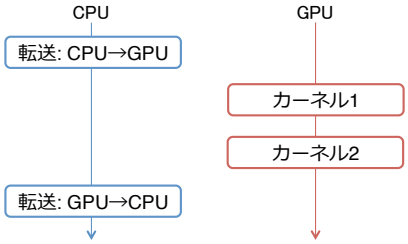

図 3: 一般的な CUDA プログラミングの流れ.

3 CUDA プログラミング

CUDA プログラミングにおいて，GPU で行う処理は関数単位で記述しカーネルと呼ばれる．CPU と GPU はメモリ空間がわかれているため，CPU から GPU のメモリ，あるいは GPU から CPU のメモリへ直接アクセスできない．したがって，計算や通信に必要なデータは，`cudaMemcpy` といった API を用いて，CPU と GPU の間でデータを転送する．図 3 の様に，計算用のデータを GPU へ送り，カーネルを起動して計算を行い，結果を GPU から転送するという手順が基本的な CUDA プログラミングの流れとなり，転送に伴うオーバーヘッドが存在する．

3.1 PGI CUDA Fortran

GT5D は Fortran で記述されているが，NVIDIA 社の提供する GPGPU 用開発環境 CUDA では，C 言語および C++言語のコンパイラのみ提供されており，そのままではソースコードを再利用できないため，本研究では，PGI 社の提供する PGI CUDA Fortran コンパイラを利用する．

PGI CUDA Fortran は，CUDA C/C++のように，Fortran 2003 の仕様に CUDA のために文法を拡張したコンパイラと，CUDA ランタイムライブラリを Fortran から呼び出すためのライブラリから構成される．PGI CUDA Fortran コンパイラは，Fortran コードを C コードに変換し，バックエンドとして CUDA C/C++コンパイラを呼び出し，GPU 向け実行ファイルを作成する．PGI CUDA Fortran のソースコード例をリスト 1 に示す．CUDA C/C++における`__global__`と同等の意味を持つ `attributes(global)` や，Shared Memory に領域を確保することを示す `shared` 属性，カーネル起動時のスレッド，ブロックの次元数を指定する`<<< >>>`といったものが，Fortran に対する CUDA 拡張である．また，CUDA ランタイムの関数は，ほぼ全て Fortran から呼べるようにバインディングが提供されている．

表 1: 計算機環境.

CPU	Intel Xeon E5-2670 × 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

リスト 1: PGI CUDA Fortran の例.

```
1  attributes(global) &
2  subroutine saxpy_kernel(alpha, x, y)
3    real, value :: alpha
4    real :: x(256), y(256)
5    real, shared :: tmp(256)
6
7    tmp(threadIdx%x) = y(threadIdx%x)
8    y(threadIdx%x) = &
9      alpha * x(threadIdx%x) + tmp(threadIdx%x)
10 end subroutine saxpy
11
12 subroutine saxpy(alpha, x, y)
13   real :: alpha
14   real, device :: x(256), y(256)
15
16   call saxpy_kernel<<<1, 256>>>(alpha, x, y)
17 end subroutine saxpy
```

4 計算機環境

本研究では，筑波大学計算科学研究センターの超並列 GPU クラスタである HA-PACS を実験に用いる [2]．HA-PACS 1 ノードの性能諸元を表 1 に示す．1 つのノードに，Intel Xeon E5-2670 が 2 台，NVIDIA Tesla M2090 が 4 台，および Inifiband HBA が搭載され，図 4 のように接続されている．CPU1 と CPU2 の間は，Intel の CPU 相互接続用シリアルバスである QuickPass Interconnect (QPI) で接続され，CPU と各 GPU 間は PCI Express 16 レーンで接続され，CPU1 つにつき GPU が 2 つ接続されている．CPU1 と CPU2 はそれぞれ 64GB のメモリが結合され，ノードあたり 128GB のメモリを持つ NUMA (Non Uniform Memory Architecture) を構成してる．したがって，CPU1 から CPU2 のメモリ，CPU2 から CPU1 のメモリへのアクセスは，自 CPU の持つメモリより若干時間がかかる．マシン間インターコネクトとしては，Infiniband QDR を 2 レーン用いるマルチレーン環境を構成している．ノード全体の接続はファットツリー型となっており，268 ノードからクラスタが構成されている．

5 GT5D の GPU 化の方針

GT5D の GPU 化の方針としては，時間発展部分を GPU 化する対象とする．そして，時間発展部分の中で，

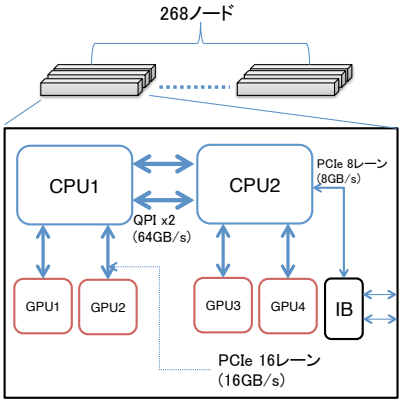

図 4: ノード内のコンポーネント間接続の概念図.

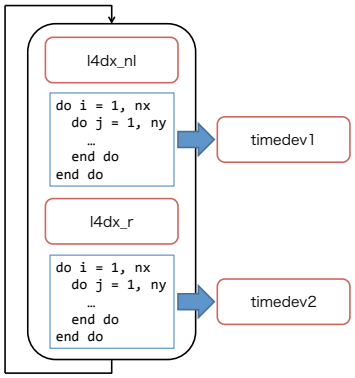

図 5: GT5D の時間発展部分の GPU 化の概要図.

特に重い関数である l4dx_s, l4dx_r, l4dx_l, l4dx_nl の 4 関数を中心に考える．これらの関数は，通信を含まない計算のみの関数である．l4dx_r，l4dx_l，l4dx_nl の 3 関数は，時間発展 1 回につき 2 回呼ばれる．l4dx_s は状況に寄らず固定で呼ばれる 4 回に加えて，時間発展の内部ループ 1 回につき 1 回呼ばれている．内部ループはある値が一定値以下に収束するまで繰替えされるため，l4dx_s の呼び出し回数は実行パラメータによって異なる．

CPU と GPU の間の通信速度はあまり早くなく，性能面から CPU と GPU の間のデータ移動は，必要最低限に抑える必要がある．l4dx_s, l4dx_r, l4dx_l, l4dx_nl の周辺に，小さな DO ループがいくつかある．各ループは計算量としては多くないが，CPU で実行するとなると，CPU~GPU 間のデータ転送が発生し，性能に悪影響を及ぼすため，図 5 のように関数化 (timedev1~timedev9) し，GPU で実行する．ある処理を GPU 化するかどうか決定する際は，計算量だけでなく，CPU~GPU 間のデータ転送量も重要となる．また，GPU よりも CPU で実行する方が速い処理の場合でも，データ移動の時間を含めて比較検討しなければならない．

5.1 プロセス毎の GPU の割り当ての方針

GT5D の GPU 化にあたり，1 つのプロセスがいくつの GPU を制御するかを考える．本研究では，図 6 に示す割り当て方針を採用する．1 つのプロセスに対して 1 つの GPU を割り当てを行い，プロセスは担当している GPU のみを制御する．HA-PACS では，1 ノードにつき，2 つの CPU が搭載され，各 CPU に対して 2 つの GPU が接続されている．したがって，1 ノードあたり 4 プロセスを起動し，MPI 並列におけるプロセス番号と 4 の剰余を取り，操作する GPU を決定する．また，HA-PACS はノードあたり 16 コアを持つため，プロセス当りの OpenMP のスレッド起動数を OMP_NUM_THREADS 環境変数を使用し 4 に設定する．

HA-PACS は NUMA 構成となっているため，他の CPU にあるメモリへのアクセスは速度面でペナルティがある．また，GPU も同様に，他の CPU の配下にある GPU へのデータ転送は，避けなければならない．前述した 2 つの条件を満すために，numactl コマンドを使用し，あるプロセスが実行される CPU を固定する．numactl は NUMA 環境でのリソースを制御するために用いるコマンドであり，プロセスが利用する CPU コアとメモリを限定できる．例えば，リスト 2 の様にコマンドを実行すると，GT5D をノード 0 番の CPU で実行し，0 番 CPU に接続されているメモリ (ローカルメモリ) を利用するという意味になる．

リスト 2: numactl コマンドの例.

```
1 $ numactl --cpunodebind=0 --localalloc -- ./GT5D
```

本方針では，GT5D が持つ既存の MPI 並列化のコードを再利用でき，開発が容易であること，また，プロセス毎にデータ参照の局所性があり，NUMA の対応を取りやすいこと，といった利点があるが，一方で，同じノードに接続されている GPU 間のデータ交換でさえ，MPI を経由せねばならず，オーバーヘッドが発生するという欠点を持つ．

6 性能評価

6.1 関数毎の性能評価

関数毎の性能評価では，各種パラメータを次のように設定し測定を行った．GT5D のメッシュ分割数は $(N_R, N_\zeta, N_Z, N_{v_\parallel}, N_\mu) = (64, 64, 64, 64, 4)$ とし，MPI 並列は使用せず 1 プロセスのみ動かし，CPU 側の OpenMP スレッド数は 4，GPU 使用数は 1 とする．

timedev1~timedev9 関数を GPU 化し，CPU と性能を比較した結果を表 2 と図 7 に示す．最も性能が改

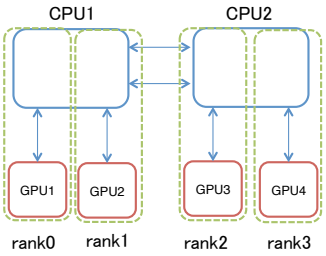

図 6: プロセス毎の GPU の割り当ての方法.

表 2: timedev1∼timedev9 関数の性能評価.

関数名	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

善した関数は timedev1 のケースで，CPU と比べ 3.37 倍高速になった．また，timedev1∼timedev9 関数の平均では，CPU と比べ 2.66 倍高速になった．

l4dx_r, l4dx_s, l4dx_l, l4dx_nl 関数を GPU 化し，CPU と性能を比較した結果を表 3 と図 8 に示す．最も性能が改善した関数は l4dx_r 関数のケースで，CPU と比べ 2.16 倍高速になった．l4dx_s, l4dx_nl 関数でも速度向上がみられるものの，l4dx_l 関数は CPU と比べて 0.77 倍高速と，GPU で実行する方が遅くなってしまった．

6.2 時間発展全体の性能評価

時間発展全体の性能評価では，各種パラメータを次のように設定し測定を行った．GT5D のメッシュ分割数は $(N_R, N_\zeta, N_Z, N_{v_\parallel}, N_\mu) = (64, 64, 64, 64, 4)$ とし，HA-PACS 1 ノードを使用した．プロセス，スレッド，GPU の割り当ては前章で延べた通りである．

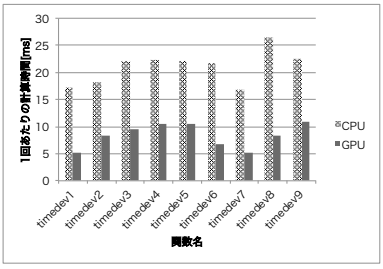

図 7: timedev1∼timedev9 関数の性能評価のグラフ.

表 3: l4dx_r, l4dx_s, l4dx_l, l4dx_nl 関数の性能評価.

関数名	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
l4dx_nl	148.0	123.9	1.19

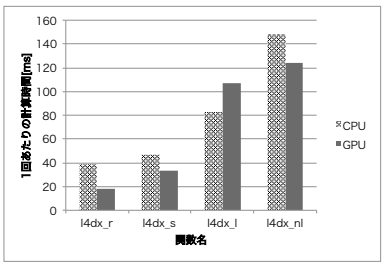

図 8: l4dx_r, l4dx_s, l4dx_l, l4dx_nl 関数の性能評価のグラフ.

時間発展 1 回あたりの計算時間は，CPU が 7.05 秒，GPU が 5.81 秒と，GPU の方が 1.21 倍高速に計算できるという結果が得られた．なお，上記計算時間は MPI 通信および CPU∼GPU 間の通信時間を含んでいる．

7 まとめと今後の課題

本研究では，核融合シミュレーションコード GT5D の GPU 化を行なった．関数単位では，CPU と比較し，最大で 3.37 倍の高速化が得られたものの，CPU よりも遅い関数があり，最適化が不十分である．

1 ノードあたり 4GPU を利用し，時間発展部全体を実行した場合，1.21 倍の高速化が達成できた．しかしながら，通信と計算のオーバーラップなど，さらなる最適化を行う余地は残っている．

参考文献

[1] TSUBAME2 ハードウェア構成 — TSUBAME 計算サービス. http://tsubame.gsic.titech.ac.jp/hardware-architecture

[2] HA-PACS ベースクラスタ — 筑波大学 計算科学研究センター. http://www.ccs.tsukuba.ac.jp/CCS/research/project/ha-pacs/cluster

[3] Y.Idomura, M.Ida, T.Kano, N.Aiba, S.Tokuba, Conservative global gyrokinetic toroidal full-f five-dimensional Vlasov simulation, Computer Physics Communications, 179, 391-403, 2008