

格子 QCD における CPU と GPU の協調動作についての考察

土井淳^{†1}

ペタスケールからエクサスケールへと計算機システムが巨大になるにつれ、電力効率や集積度の点から、GPU のようなアクセラレーターを組み合わせた計算機システムの重要度がますます高まっている。しかしながら、CPU 自体の計算能力も日々向上しており、計算機システム全体の計算能力を活かすためには、GPU のみならず、CPU 上でも可能な限り処理を行うことが必要である。CPU と GPU で協調動作を行うには、計算速度も性質も異なるため、工夫が必要な問題である。本研究では、格子 QCD のプログラムを用い、並列計算を行う際に生じる CPU と GPU 間で転送されるデータを利用し、一部の処理を CPU 側で行うようにする手法について考察する。

1. はじめに

格子 QCD (Quantum Chromodynamics) は、強い力の相互作用の理論をコンピュータ上でシミュレーションするのに広く利用される手法であり、古くからスーパーコンピュータシステムの重要なアプリケーションの 1 つとして知られている。特に、高いメモリバンド幅と大量の計算量を必要とし、格子 QCD がスーパーコンピュータの進化に寄与してきたものは大きく、過去には QCDPAX[1], QCDSF[2], QCDOC[3] のように格子 QCD に特化した計算機や、Blue Gene[4][5][6] シリーズのように設計思想を受け継いだ計算機として性能を加速させてきた。

格子 QCD シミュレーションによって、様々な物理現象を実験の代わりにコンピュータ上で再現することができ、カイラル対称性の自発的破れ[7]や、湯川理論[8]などが実際にコンピュータシミュレーションによって再現されてきた。今後、ペタスケールからエクサスケールへと、更なる計算機資源の拡張により、より高精細なシミュレーションが可能となり、ヒッグス粒子の発見や宇宙の起源などの、未知や未発見の現象を解き明かすことが期待される。

しかしながら、計算機の進化はその電力消費量や開発コストが問題となりつつあり、かつてのように格子 QCD に特化した計算機から、より汎用的な計算機や GPU のようなアクセラレーターを利用した計算機へとシフトしつつある。特に GPU 等のアクセラレーターを利用した計算機は、電力対性能比の観点から、主流となりつつあり、米国エネルギー省の計画する CORAL[9] のように、今後登場する大規模計算機システムの多くがアクセラレーターを搭載したシステムになる見込みである。

GPU を用いたアプリケーションの高速化においては、GPU の計算性能が CPU に比べて優れているため、一般的には GPU にすべての演算処理をオフロードするような実装を行う。また、処理能力やアーキテクチャの違いや、CPU と GPU 間のデータ転送が必要なことから、CPU と GPU で負荷分散を行うのは難しい問題である。しかしながら、GPU の進化と共に、CPU 自体も進化しており、システム全体としてみたときに CPU の計算能力を活用しないのはもった

いない。そこで、本研究では、格子 QCD を GPU クラスタ上で並列化を行う際に、CPU と GPU の双方を計算処理に利用した協調動作を行うことで、GPU のみを利用した場合と比較してどの程度性能を向上させられるのかを考察する。格子 QCD を GPU クラスタ上で並列化を行う場合、GPU と CPU 間のデータ転送は元々必要な処理であるので、この転送をうまく利用し、追加のデータ転送を無しで、CPU 上でも計算処理を行う手法について提案する。

2. 格子 QCD と並列化

2.1 格子 QCD 概要

格子 QCD は強い力場の理論を離散化してコンピュータ上でシミュレーションするための手法であり、4 次元の時空間を格子状に離散化し、格子上に物理量が定義される。格子 QCD では、図 1 に示すように格子上にスピノル場が、格子間にグルーオン場が定義され、隣接格子間における力の相互作用を用いて、線形方程式を CG 法等により解く。このとき、相互作用を計算するための演算子が、扱おうとする問題によって定義されるが、本研究においては、多くの問題で広く使われる、Wilson-Dirac 演算子を用いる。

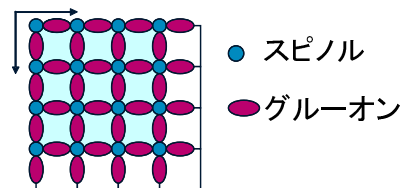


図 1 格子 QCD において格子上および格子間に定義される物理量のレイアウト。グルーオンを表す物理量（ゲージ行列）は注目する格子点から正の方向の格子間にあるものを、それぞれの次元について格子上で保持するものとする。

2.2 Wilson-Dirac 演算子

Wilson-Dirac 演算子は、式 1 に示すように、4 次元空間において隣接する 8 つの格子点との間の相互作用を計算する。

$$D(n) = \delta(n) - \kappa \cdot \sum_{\mu=1}^4 \{ (1 - \gamma_{\mu}) U_{\mu}(n) \delta(n + \hat{\mu}) + (1 + \gamma_{\mu}) U_{\mu}^{\dagger}(n - \hat{\mu}) \delta(n - \hat{\mu}) \} \quad (1)$$

式 1 において、 μ は 1~4 で、それぞれ X,Y,Z,T 軸に対応する。 $\delta(n)$ はスピノルを表し、4 つの 3 色からなるスピノルを物理量として持ち、 $U_{\mu}(n)$ はグルーオンを表し、3x3 成

^{†1} 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research – Tokyo

分のゲージ行列である。 γ_μ は、式 2 に示すような 4×4 の行列である。なお、いずれの物理量も複素数で表される。

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} \quad \gamma_2 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_3 = \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \\ i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \end{pmatrix} \quad \gamma_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad (2)$$

よって、式 1 は、隣接する 8 方向の格子点上の 3×4 のスピノルに、格子間の 3×3 のゲージ行列と 4×4 のガンマ行列を乗じたものを着目する格子点に集約する計算になる。ところで、式 2 に示すガンマ行列の対称性を利用すると、式 3 に示すように、ゲージ行列の乗算において共通項により半分の計算量にできることが知られている。

$$(1 - \gamma_1) U_1(n) \delta(n + \hat{1}, m) = \begin{pmatrix} U_1(n) \cdot (s_1 + i \cdot s_4) \\ U_1(n) \cdot (s_2 + i \cdot s_3) \\ -i \cdot U_1(n) \cdot (s_2 + i \cdot s_3) \\ -i \cdot U_1(n) \cdot (s_1 + i \cdot s_4) \end{pmatrix} = \begin{pmatrix} U_1(n) \cdot h_1 \\ U_1(n) \cdot h_2 \\ -i \cdot U_1(n) \cdot h_2 \\ -i \cdot U_1(n) \cdot h_1 \end{pmatrix} \quad (3)$$

式 3 において h で示したものは、ハーフスピノルと呼び、 2×3 の複素数で表す。ハーフスピノルはゲージ行列の演算を半分にするだけでなく、隣接格子間のスピノルの受け渡しにおいてもデータアクセスを半分にすることができ、Wilson-Dirac 演算子の並列化を行う際の通信量が半分になる。

Wilson-Dirac 演算子は次の 3 ステップによって計算される。

- (1) ハーフスピノルの生成(12 flops)
- (2) ハーフスピノルとゲージ行列の乗算(132 flops)
- (3) スピノルへの集約(48 flops, T 軸のみ 24 flops)

よって、格子点あたりの演算量は 1,488 flops であり、この計算に必要な 8 つのゲージ行列と 9 つのスピノルがロードされ、1 つのスピノルがストアされる。したがって、Wilson-Dirac 演算子は倍精度の場合 2.06 byte/flops、単精度の場合 1.03 byte/flops であり、メモリバンド幅に性能が大きく左右されることが分かる。

2.3 Wilson-Dirac 演算子の並列化

Wilson-Dirac 演算子を分散メモリ並列化するには一般的には、4 次元格子をいずれかの軸方向あるいは複数の軸方向についてブロック分割し、それぞれの分割された格子を各プロセスに割り当てて計算を行う。その際、隣接格子のデータを隣接プロセス間で交換する必要がある。なお、ここでは、周期的境界条件であるとし、元々の格子の両端の間においてもデータの交換が必要である。図 2 に示すように、分割された格子について、元々隣接していた格子のデータを隣のプロセスとの間で送りあう。このとき、ゲージ行列は格子点から見て正の方向のものをしているため、負の方向に隣接する格子点が隣のプロセスにある場合にゲー

ジ行列を参照することはできない。そのため、正方向のプロセスにデータを送る場合、あらかじめゲージ行列を乗じてから送ることで、ゲージ行列自体を送る必要を無くしている。また、このとき送信されるデータは、ハーフスピノルを用いる。

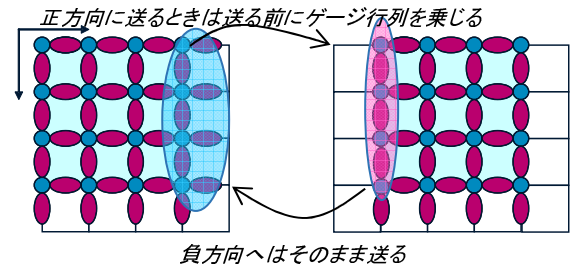


図 2 Wilson-Dirac 演算子におけるプロセス間のデータ交換。ゲージ行列の保持の仕方のため正方向と負方向で処理が異なる。

3. Wilson-Dirac 演算子の CUDA による実装

3.1 データ構造

Wilson-Dirac 演算子において、4 次元格子点上のデータ、スピノルおよびゲージ行列は、1 次元配列の形でメモリ上に保持する。それぞれ、 3×4 、 3×3 の複素数を持つが、このような構造体を配列として扱う手法として、AoS(Array of Structure)または、SoA(Structure of Arrays)が用いられる。一般的に GPU のような SIMD 演算器においては、SoA 形式を用いて隣接する格子のデータを逐次的にアクセスし処理するのが好ましいとされている。GPU においては、GPU のスレッドで連続したデータを扱う、コアレスアクセスを用いて最適化を行うために SoA 形式を用いるのが一般的である。よって、本研究では、SoA 形式を用いて、スピノルおよびゲージ場の行列を記述する。その際、複素数の配列の構造体として扱う。

3.2 GPU のスレッドへの処理の割り当て

各 GPU スレッドに 1 つの格子点を割り当てて処理を行う。このとき、X 軸方向の格子点を連続したスレッドに割り当てると、SoA 形式で保存したスピノルおよびゲージ行列についてコアレスアクセスができるようにする。このとき、連続する 32 の倍数の格子点を同一のスレッドブロックで実行するようにする。X 軸方向の格子サイズを N_x とするとき、 N_x が 32 の倍数ではない場合、最小公倍数が 32 の倍数となるような nyblock 行のブロックを同一スレッドブロックで実行するようにする。次のような CUDA コードを用いてカーネル関数を呼び出すことになる。

```
Dopr<<<dim3(Ny/nyblock,Nz,Nt),dim3(Nx,nyblock,1)>>>(...);
```

3.3 ゲージ行列の圧縮

Wilson-Dirac 演算子はメモリバンド幅ネックな処理であるので、できるだけメモリアccessを減らすことが高速化の鍵となる。ゲージ行列が SU(3)に属する場合、その対称性を用いることで、 3×3 行列のうち任意の 2 行または 2 列

から, 3×3 行列を復元できることが知られている[10]. この性質を利用することで, 3×2 の行列成分をメモリからロードし, 実行時に演算によって残りの3成分を求めることができる. 式4のようにゲージ行列を記述するとき, AおよびBの 3×2 成分を用いて, Cの3成分は, 式5によって計算できる.

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ c_0 & c_1 & c_2 \end{bmatrix} \quad (4)$$

$$C = (A \times B)^* \quad (5)$$

これにより, ゲージ行列あたり 42flops の演算量が追加されるが, GPUにおいてはメモリアクセスよりも演算の方が圧倒的に高速であるため, 実際の処理時間は短縮される.

3.4 GPUを用いたWilson-Dirac 演算子の並列化

一般的な分散メモリ並列化と同じように, GPUを用いる場合についても, GPUを分散メモリ環境における1つのプロセスであると考えて, ブロック分割された部分格子を持つ. つまり, 演算にGPUのみを用いるとすると, 元の格子をノードあたりのGPU数*ノード数分割する.

並列化を行うにあたり, 境界の部分についてGPU間および異なるノード間でデータを交換する必要がある. GPUを用いた境界の部分のデータ交換の処理は次のようになる.

- (1) ハーフスピノルの生成, 負方向へ参照するデータの場合はゲージ行列を乗算
- (2) ハーフスピノル配列をホストのメモリに転送
- (3) ハーフスピノル配列をノード間で転送 (あて先が同一ノード内のGPUではない場合)
- (4) ハーフスピノル配列をGPUへ転送
- (5) 正方向の場合ゲージ場行列を乗算, 集約計算

このとき, データ交換用にハーフスピノルを生成する処理や集約する処理について, 最内ループとなるX軸方向について行う場合, コアレスアクセスの観点から非常に効率が悪いので, X軸方向についてはブロック分割の対象からはずし, 同一GPU内で処理を完結させるものとする. 残りのY,Z,T軸方向について, ブロック分割の対象とし並列化を行い, なるべく外側から少ない軸数で分割するようにする. ただし, 本研究では同一ノード内のGPU間で並列化を行う場合, Y軸を分割するようにした.

これらのデータ交換の処理を効率良く行うために, CUDA streamを用いて非同期的にデータ転送と演算処理を重ね合わせる. ここでは, GPUあたり, ブロック分割を行う軸数*2+1個のCUDA streamを用いる. 図3は, T軸方向にノード間で分割, Y軸方向に同一ノード内のGPU間で分割した場合の5つのCUDA streamを使用した場合の実装例を示す. T,Y軸正負方向についての境界部分を処理するCUDA streamと, 通信を伴わずに計算のできる部分(inner)を非同期的に処理するが, 最後に集約を行う部分は同期が必要であるので, 図3のような階段状の順番で処理が行わ

れる.

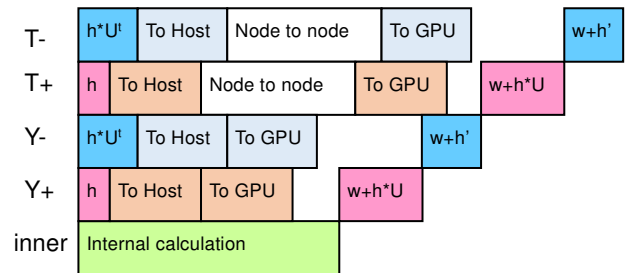


図3 CUDA streamを用いたWilson-Dirac 演算子のデータ転送処理と演算処理の重ね合わせ (T方向にノード間の分散メモリ並列化を行い, Y方向に複数GPUを用いる場合)

4. Wilson-Dirac 演算子の GPU-CPU 協調動作

4.1 CPUとGPUの協調動作と並列化

一般的に, 格子QCDのようなステンシル計算や密行列の演算等のように, 演算量が均一な処理は, GPUとCPUの演算速度やメモリバンド幅の比を用いて分割することで, 同一の処理を負荷分散するのは比較的容易である. ところが, そのような場合でも, ステンシル計算のように分割された部分について相互にデータを参照しなければならない場合にGPUとCPUの間でデータ転送が必要となるため, GPUのみを用いて計算を行う場合に比べて効率が悪くなる.

しかしながら, 複数ノードや複数GPUを用いて並列化を行う前提であれば, どちらにしてもノード間やGPU間でデータを参照しあう必要があるために, GPUとCPU間のデータ転送は生じる. このデータをうまく利用してCPU側で演算処理ができれば, GPUの計算能力に加えて, CPUの計算能力を使って, 処理速度を向上できる可能性がある.

4.2 Wilson-Dirac 演算子のT軸方向のGPU-CPU協調動作

ここでは, まずT軸方向についてノード間でブロック分割を行って並列化した場合について考える. 式1および式2から, T軸正の方向の処理は式6, T軸負の方向の処理は式7のようになる.

$$(1 - \gamma_4)U_4(n)\delta(n + \hat{4}, m) = \begin{pmatrix} 0 \\ 0 \\ 2 \cdot U_4(n) \cdot (s_3) \\ 2 \cdot U_4(n) \cdot (s_4) \end{pmatrix} \quad (6)$$

$$(1 + \gamma_4)U_4(n)\delta(n - \hat{4}, m) = \begin{pmatrix} 2 \cdot U_4(n) \cdot (s_1) \\ 2 \cdot U_4(n) \cdot (s_2) \\ 0 \\ 0 \end{pmatrix} \quad (7)$$

式6および式7から, T軸方向については, ハーフスピノルを生成するための演算処理を行わなくても, 正方向は3つ目と4つ目のスピン成分を, 負方向は1つ目と2つ目のスピン成分を取り出せば, ハーフスピノルとして使用で

きることが分かる．つまり，GPU 側で境界部分についてのハーフスピノル生成処理を行わずに，直接スピノル配列からホスト側に境界部分のハーフスピノル配列を転送すれば良い．負方向については，ゲージ行列を乗じる処理が必要であるので，本来はハーフスピノル生成処理の一部として GPU 側で処理されるものであるが，この処理を CPU 側で実行するようにすることが可能である．この方法を用いて図 3 を CPU との協調動作に対応させたものを図 4 に示す．

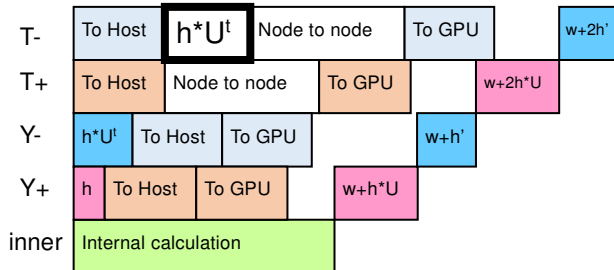


図 4 Wilson-Dirac 演算子の T 軸負方向の境界部分のゲージ行列積を CPU 側で処理するようにしたときの実装

T 軸方向の境界部分は，X,Y,Z 軸の直方体として表れ，1 次元配列上では連続して記憶される．GPU 上で SoA 形式で保存されているため，ハーフスピノルの 6 つの複素数成分を取り出すには 6 つの部分に分けて転送する必要があるが，`cudaMemcpy2D(cudaMemcpy2DAsync)`関数を使用することで効率良くホスト上のメモリに転送できる．

図 4 で，CPU で実行されるゲージ行列積の処理は `cudaStreamQuery` 関数データを受け取ったのを確認した後，非同期で実行できるように `pThread` を用いて複数スレッドを新しく生成してその上で行う．すべてのスレッドが join した後であって先のノードへハーフスピノル配列を転送する．

また，正方向についても同様にゲージ行列積を CPU 側で処理させることも考えられる．図 5 に示すように，別のノードから転送されたハーフスピノル配列を受け取った直後に CPU 側でゲージ行列積を計算してから GPU にハーフスピノル配列を転送する．実はこれら境界部分のゲージ行列は正負共に同じものが参照されるので (Even-odd 等のプリコンディショニングを行わない場合に限り)，うまくいけばキャッシュメモリ上のものが再利用でき，効率良く処理できる可能性がある．

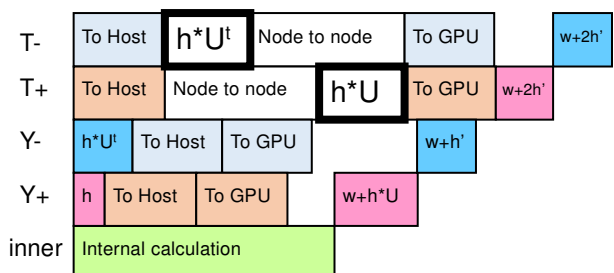


図 5 Wilson-Dirac 演算子の T 軸両方向の境界部分のゲージ行列積を CPU 側で処理するようにしたときの実装

さらに，Y 軸や Z 軸方向にも同様にノード間でブロック分割を行う場合にも境界部分のゲージ行列積を CPU で行うことも考えられる．しかしながら T 軸方向以外の軸方向については，GPU でまずハーフスピノルを生成する処理が必要となり，T 軸方向の場合のように GPU 側の処理を減らす効果は比較的大きくはないと考えられる．

5. 性能評価

5.1 実行環境

本性能評価では，表 1 に示す計算機 4 ノードから構成されるクラスターを利用して性能評価を行った．GPU は各ノード 2 枚ずつ装着されているが，それぞれ別々のソケットに接続されるため，GPU 間の peer-to-peer のデータ転送は利用できない．

表 1 実行環境

計算ノード	IBM System x iDataPlex dx360 M4
CPU	2x Intel Xeon E5-2665
メモリ	64 GB
GPU	2x Nvidia Tesla K20X
ネットワーク	Infiniband, Mellanox MT26428

また，本性能評価に使用した CUDA toolkit のバージョンは 7.0 である．

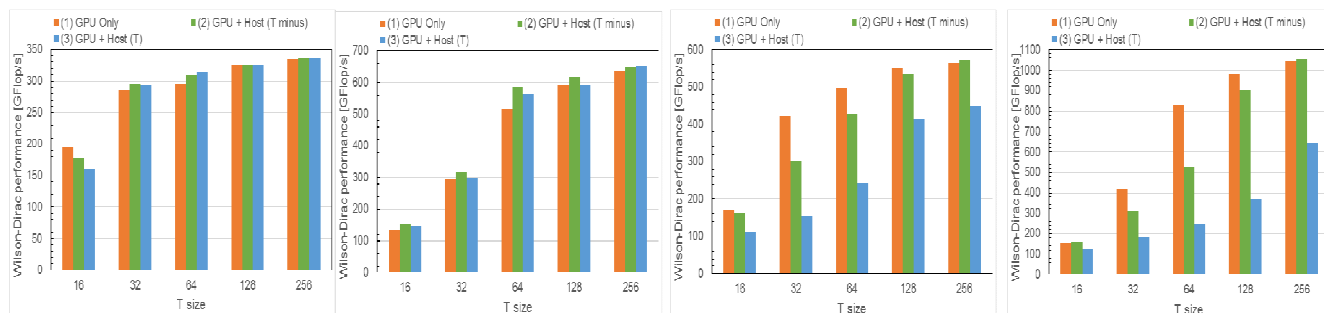
表 2 に，本性能評価環境における CPU と GPU の性能比較をまとめる．単純にピーク性能値で比較することはできないが，CPU の性能は GPU の 10 分の 1 程度はあり，CPU と GPU の協調動作を行うことで，数パーセントの性能向上が望める．

表 2 実行環境における CPU と GPU の性能比較

	CPU(Xeon E5-2665)	GPU(Tesla K20X)
コア数	8	2,688
ピーク性能	倍精度 153.6 Gflops 単精度 307.2 Gflops	1311.74 Gflops 3935.23 Gflops
メモリバンド幅	51.2 GB/s	249.6 GB/s

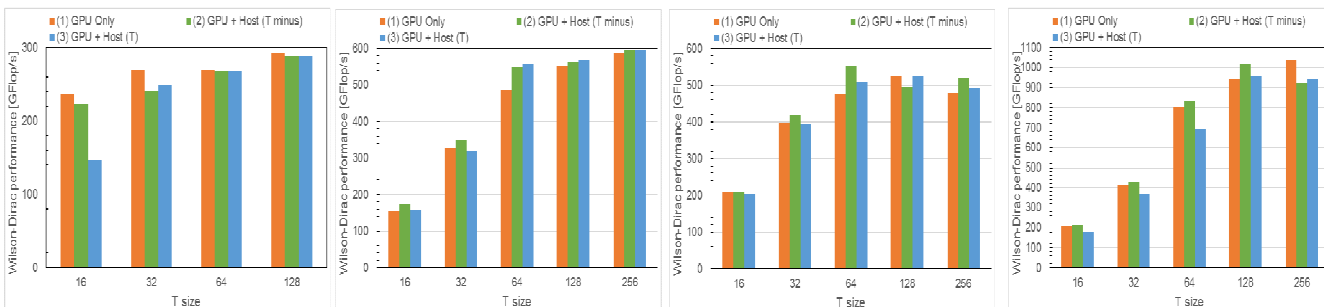
5.2 T 軸方向に協調動作を行う場合の性能評価

Wilson-Dirac 演算子について，倍精度，単精度それぞれを，T 軸方向の境界部分の処理について，(1)すべて GPU で処理する場合，(2)負方向のゲージ行列積のみをホストで実行する場合，(3)正負両方向のゲージ行列積をホストで実行する場合について比較した．このとき，2 種類の格子サイズ， $16 \times 16 \times 16 \times N_t$ および， $32 \times 32 \times 32 \times N_t$ について， N_t の値を 16 から 256 まで変化させたときの性能を測定した．また，2 ノードまたは 4 ノードを使用し，T 軸方向にブロック分割を行った．ノードあたり使用する GPU の数も 1 または 2 とし，2GPU を使用する場合は Y 軸方向にブロック分割を行った．倍精度の結果を図 6 および図 7 に，単精度の結果を **Error! Reference source not found.** および **Error! Reference source not found.** に示す．



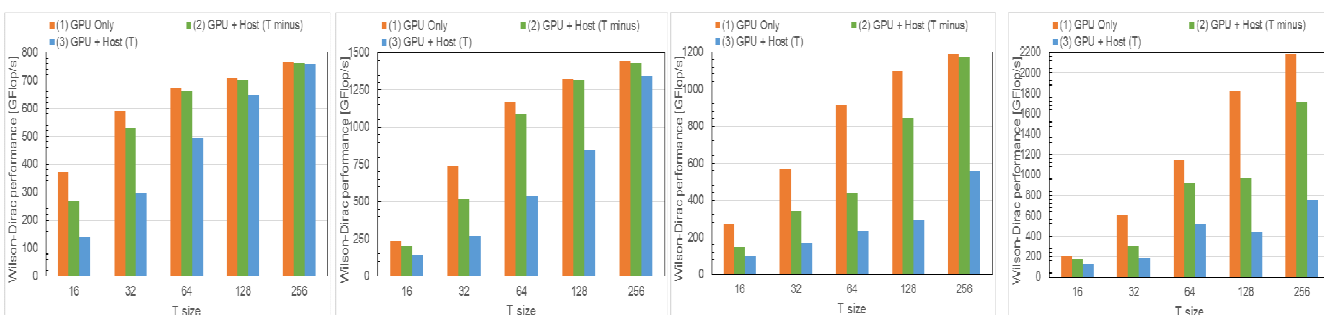
(a) 2 ノード, ノードあたり 1GPU (b) 4 ノード, ノードあたり 1GPU (c) 2 ノード, ノードあたり 2GPU (d) 4 ノード, ノードあたり 2GPU

図 6 倍精度, 格子サイズ 16x16x16xNt のときの Wilson-Dirac 演算子の実効性能の測定値



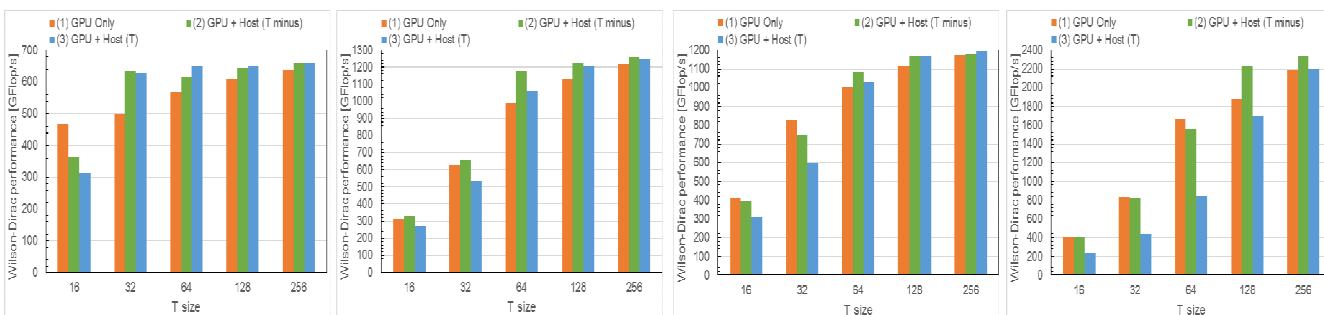
(a) 2 ノード, ノードあたり 1GPU (b) 4 ノード, ノードあたり 1GPU (c) 2 ノード, ノードあたり 2GPU (d) 4 ノード, ノードあたり 2GPU

図 7 倍精度, 格子サイズ 32x32x32xNt のときの Wilson-Dirac 演算子の実効性能の測定値



(a) 2 ノード, ノードあたり 1GPU (b) 4 ノード, ノードあたり 1GPU (c) 2 ノード, ノードあたり 2GPU (d) 4 ノード, ノードあたり 2GPU

図 8 単精度, 格子サイズ 16x16x16xNt のときの Wilson-Dirac 演算子の実効性能の測定値



(a) 2 ノード, ノードあたり 1GPU (b) 4 ノード, ノードあたり 1GPU (c) 2 ノード, ノードあたり 2GPU (d) 4 ノード, ノードあたり 2GPU

図 9 単精度, 格子サイズ 32x32x32xNt のときの Wilson-Dirac 演算子の実効性能の測定値

この測定においては, T 軸方向をブロック分割し, T 軸方向の境界部分についての処理を GPU とホストで処理を行うため, 単純に T 軸方向のサイズが大きいほど, ホスト側の処理量の割合が相対的に小さくなる. したがって, ホスト側と GPU 側の処理性能の比よりも, ホスト側の処理量の割合が小さくならないと, GPU とホストで協調処理を行っても性能が向上しない. 例えば, 図 6(a)では, Nt=16 の

ときは性能比よりも処理量の割合が大きいため, GPU のみで処理した方が性能が良いが, Nt=32 以上になるとホスト側でも処理をした方が性能が良くなっているのが分かる.

また, データサイズが小さいとき, 負方向のみをホスト側で処理した方が性能が出やすいが, データサイズが大きくなると, 正負両方向を処理しても良い性能を得られる場合があることが分かる.

単精度と倍精度を比べると、単精度の場合は GPU の処理性能が CPU の性能よりも比較的大きくなるため、協調処理によって得られる性能向上が得られるのは倍精度よりも大きなデータサイズのときになってしまう。

さらに、ノードあたり 2 つの GPU を用いると、ノードあたりの GPU による計算能力は 2 倍になるがホスト側の計算能力は変わらないため、相対的な性能差も 2 倍と大きくなるため、協調動作による性能向上が得られる機会も比較的小さくなっている。

5.3 T 軸および Z 軸方向に協調動作を行う場合の性能評価

次に、Wilson-Dirac 演算子について、4 ノードを用いて、T 軸および Z 軸方向にそれぞれ 2 ノードを用いてブロック分割する場合について、T 軸および Z 軸方向の境界部分の処理について、(1)すべて GPU で処理する場合、(2)T 軸のみ負方向のゲージ行列積のみをホストで実行する場合、(3)T 軸のみ正負両方向のゲージ行列積をホストで実行する場合、(4)T 軸および Z 軸の負方向のゲージ行列積のみをホストで実行する場合、(5)T 軸および Z 軸の正負両方向のゲージ行列積をホストで実行する場合、について比較した。倍精度の場合の結果を図 10 および図 11 に、単精度の場合の結果を図 12 および図 13 に示す。

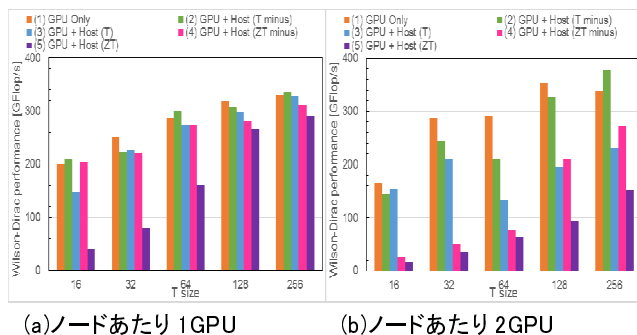


図 10 倍精度、格子サイズ 16x16x16xNt のとき 4 ノードを使用して ZT 方向に分割した場合の Wilson-Dirac 演算子の実効性能比較

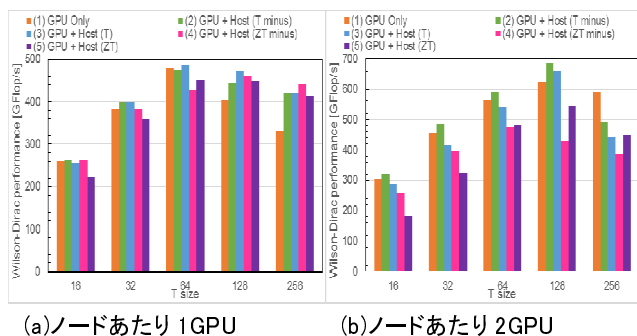


図 11 倍精度、格子サイズ 32x32x32xNt のとき 4 ノードを使用して ZT 方向に分割した場合の Wilson-Dirac 演算子の実効性能比較

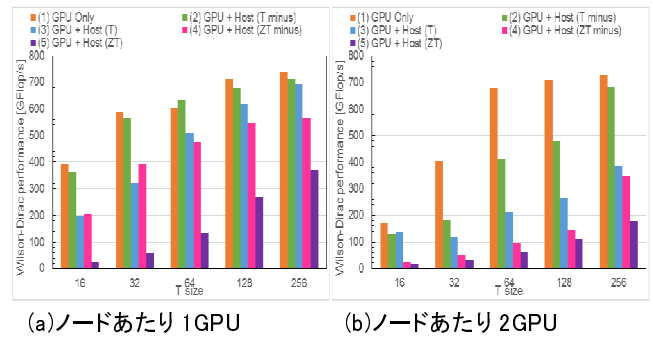


図 12 単精度、格子サイズ 16x16x16xNt のとき 4 ノードを使用して ZT 方向に分割した場合の Wilson-Dirac 演算子の実効性能比較

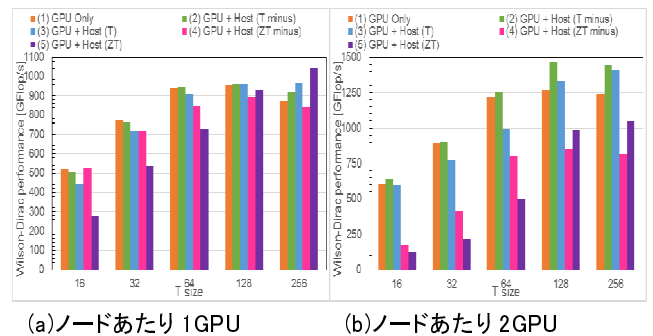


図 13 単精度、格子サイズ 32x32x32xNt のとき 4 ノードを使用して ZT 方向に分割した場合の Wilson-Dirac 演算子の実効性能比較

いずれの場合においても、図 6～図 9 に示す 4 ノードを使用して T 軸を 4 分割した場合に比べて性能が半分程度まで落ちている。これは、GPU とホスト間のデータ転送および MPI による通信が 2 軸分必要となったため、それぞれのバンド幅の取り合いが生じているためと思われる、最適化の余地がまだある可能性があるが、今後の検討項目とする。

T 軸方向のみについて協調動作を行った場合、T 軸のみをブロック分割した場合とほぼ同じような傾向が見られた。Z 軸方向についても協調動作を行った場合、良好な結果が得られる場合もあるが、ほとんどの場合、大きく性能を落とす結果になってしまった。やはり、T 軸のようにハーフスピノルの生成を省略できるような特別な軸を利用するのが性能向上に寄与しやすいと考えられる。

6. おわりに

格子 QCD の Wilson-Dirac 演算子について、GPU を搭載したクラスタ上で分散メモリ並列化を行うとき、GPU とホストの間で転送されるデータを利用して、ホスト上でも計算を行う協調動作を行う方法を検証した。T 軸方向について、境界部分のハーフスピノルを GPU 上で生成せずに直接転送してからホスト上で処理を行うことで、条件が合えば数パーセントの性能向上が見込めることが分かった。しかしながら、ホスト上で処理できている部分は小さく、それでも性能向上できる条件はまだ厳しく、更なる工夫が

必要であると考えられる。

また、今後 NVLINK[11]が実装され、GPU 間、GPU とホスト間がより高速に接続され、GPU Direct により、MPI による通信がホストのメモリを経由せずに高速に実行できるようになる場合、また違った協調動作を考える必要があると思われる。NVLINK を考慮した協調動作を検討していきたい。

参考文献

- 1) T. Shirakawa et al. QCDPAX—an MIMD array vector processors for the numerical simulation of quantum chromodynamics, Proceedings of the 1989 ACM/IEEE conference on Supercomputing, 1989.
- 2) R. D. Mawhinney, The 1 Teraflops QCDSP Computer, Parallel Computer 25, No. 10/11, pp.1281-1296, September, 1999.
- 3) P. A. Boyle et al. QCDOC: A 10-Teraflops Computer for Tightly-Coupled Calculations, Proceedings of the ACM/IEEE conference on Supercomputing SC04, 2004.
- 4) A. Gara et al. Overview of the Blue Gene/L System Architecture, IBM Journal of Research and Development Vol. 49, No. 2/3, pp. 195-212, 2005.
- 5) IBM Blue Gene Team, Overview of the IBM BlueGene/P project, IBM Journal of Research and Development, vol. 52, no. 1/2, pp. 199-220, 2008.
- 6) The Blue Gene Team, Blue Gene/Q: by co-design, Computer Science - Research and Development, Volume 28, Issue 2-3, pp. 127-135, May 2013.
- 7) H. Fukaya et al. [JLQCD collaboration], Two-flavor lattice QCD simulation in the epsilon-regime with exact chiral symmetry, Physical Review Letters 98, 172001, 2007.
- 8) N. Ishii et al. Nuclear force from lattice QCD, Physical Review Letters, June, 2007.
- 9) CORAL Collaboration, <https://asc.llnl.gov/CORAL/>
- 10) M. A. Clark et al. Solving Lattice QCD systems of equations using mixed precision solvers on GPUs, Comput. Phys. Commun. 181, 1517, 2010.
- 11) NVIDIA NVLINK HIGH-SPEED INTERCONNECT, <http://www.nvidia.com/object/nvlink.html>