Vol.2015-HPC-148 No.13
2015/3/2

GPU搭載システムにおける都市気流シミュレーションの大規模化と性能モデル

高嵜 祐樹[1]　遠藤 敏夫[1]　松岡 聡[1]

概要：GPU 向けステンシル計算の規模は，通常 GPU のメモリ容量に制限されるが，テンポラル・ブロッキングと呼ばれる手法により性能劣化なく大規模化を実現可能である．本研究では，10,000 行を超えるコード規模を持つ GPU クラスタ向けアプリケーションである都市気流シミュレーションの大規模化・高性能維持を実現する手法として，HHRT をテンポラルブロッキングを組み合わせた手法を導入した結果，大規模化に対して，性能劣化とプログラミングコストを抑えることに成功した．本研究では，更なる性能最適化のために，HHRT のスワップデータサイズを削減する手法を提案する．その結果，性能が約 1.3〜1.9 倍向上し，元プログラムの約 19〜85%の性能を達成した．さらに性能予測モデルの構築により，性能に影響を与えるパラメータの絞り込みを可能にした．

1. はじめに

エクサスケール時代に向けた高性能計算システムにおける課題の一つにメモリウォール問題，つまりプロセッサの計算速度に比べ，メモリのデータ転送速度と容量の増加の進歩が遅いことがあげられる．近年，GPGPU や Xeone Phi に代表されるアクセラレータの進歩と普及により，ますます対応の必要性は高まっており，その解決に向けてメモリ階層を効率的に活用する技術が求められている．

本稿では，GPU を用いたスーパーコンピュータ上におけるステンシル計算を対象に取り上げ，問題規模の大規模化と高性能性を目的とする．ステンシル計算は，流体シミュレーションなどの分野の重要カーネルであり，一般的には，シミュレーションの対象となる領域を規則格子で表し，各時間ステップでは各格子点の値の更新を隣接する格子の値を使って計算する．非常に並列性が高いこと，および必要とするメモリバンド幅性能が高いという性質から，GPU 上の実行において高速化を実現している [1], [2]．しかし，GPU に搭載されているデバイスメモリはホストメモリより小さく，GPU 上のステンシル計算のほとんどにおいて，対象とする問題規模はデバイスメモリサイズ内に限定されている．シミュレーションの分野において求められる，大規模，高精度かつ高性能な計算を実現するには，デバイスメモリだけでなくホストメモリも加えたメモリ階層を効率的に利用する必要がある．

[1] 東京工業大学
Tokyo Institute of Technology

この問題に対し，我々は，GPU スパコン向けに実装された既存の流体アプリケーションの一つであり，MPI と CUDA で記述された都市気流シミュレーションソフトウェア [2] を対象として，大規模性・高性能性・高生産性を実現することを提案した [3]．第一段階として，対象ソフトウェアを，HHRT と呼ばれるメモリスワップを行うランタイムライブラリ [4], [5] 上で動作させる．これにより，デバイスメモリ容量を超えることが可能となり，大規模化を実現できる．次に性能を高く維持するために，第二段階としてアルゴリズムのメモリアクセス局所性を向上させる．そのためにテンポラルブロッキング [6], [7], [8], [9], [10] と呼ばれる手法を用いてソフトウェアを書き換える．このとき，HHRT の実行モデルを基にすることにより，書き換えコストを比較的小さくすることができる．

我々はすでに，以上の手法が単純な 7 点ステンシルベンチマークにおいては適用可能であることを示してきた [5] が，大規模実アプリケーションに適用できるか明らかではなかった．そこで対象アプリケーションに我々の手法が適用可能であるかの予備調査および，アプリケーションの性質を考慮した最適化も重要であることを明らかにした．より具体的には:

- 予備調査において，データ間の依存関係が隣接データ間のみであり，各時間ステップにおけるリダクションのような大域的な処理がないことの確認が必要であった．
- 今回のアプリケーションは方向によって複数の境界条件の計算方法を用いており，それぞれが時間ブロッキ

ングに対応可能か調査が必要であった．

- 今回のアプリケーションにおいて，プロセス間境界のMPI 通信の方法が HHRT 上での低速な動作を引き起こしており，修正が必要であった．

以上の手法により，GPU を搭載したスパコンであるTSUBAME2.5 上での性能評価をしたところ，元プログラムの約 10〜64%の性能達成した．

本研究では，更なる性能向上のために，HHRT のスワップ処理時のデータ転送量を削減することを提案する．これにより，更に約 1.8〜2.9 倍の性能向上を確認し，元プログラムの約 19〜85%の性能を実現した．

また，提案手法を導入することにより現れるパラメータに対して，最適解の探索コストを抑えるための性能モデルを構築した．性能モデルの予測値と実測値を比較したところ，誤差を含むものの最適解の絞り込みに成功した．

2. 背景

2.1 GPU 搭載マシンのメモリ階層

スーパーコンピュータの性能を向上させるために GPU や Intel Xeon Phi が普及しており，本研究では NVIDIA 社の GPU について述べる．このようなシステムのメモリ階層は，従来のキャッシュ，ホストメモリ，HDD からなるメモリ階層より，高階層となっている．ホストメモリは，数十から数百 GB の容量と数十 GB/s のメモリバンド幅であるのに対し，デバイスメモリは、容量 6GB〜12GB とメモリバンド幅 250GB/s と、小容量で高速なメモリとなっている．デバイスメモリ容量を超える問題を扱うためには，ホストメモリのメモリバンド幅より更に小さい 8GB/s という PCI-Express バスを通したデータ転送を効率よく行うことや転送量を削減することが重要となる．

2.2 ステンシル計算

ステンシル計算は、流体シミュレーション分野などで、使われる一般的なカーネルである．シミュレーションの対象となる領域を規則格子で表し、時間経過による各格子点の値の変化を計算する．各格子点の値の更新には、隣接する格子の値を使って計算される．各格子点の計算は、独立に計算することが出来るため、非常に並列性が高い計算となっている．また，ダブルバッファリング手法が良く使われ，本論文でもそれを仮定する．

GPU クラスタ上での 7 点ステンシル計算を例にとり，プロセス間の通信を MPI，GPU のプログラミングを CUDA で記述された実装の例を図 1(a) に示す．これは GPU メモリを超えられない単純な例である．GPU のメモリ容量を超えるデータに対応した例を図 1(b) に示す．大容量のデータを扱うために，領域を部分領域に分割し，部分領域を入れ替えながら計算して行く．MPI 通信に加え，2 回の CPU-GPU 間のデータ転送を全部分領域にわたって毎ステップ行わなければならないので、性能が大幅に低下してしまうという問題がある．

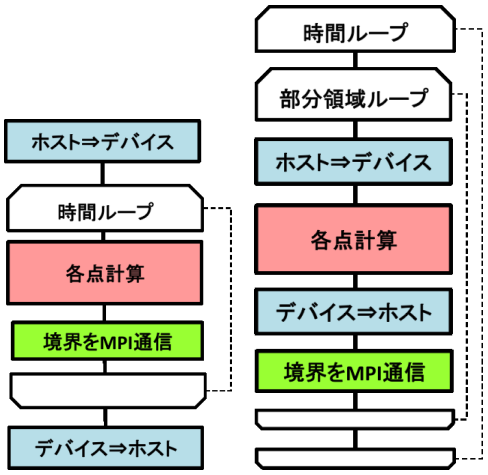

図 1 (a) 通常の MPI+CUDA、(b) 単純な大容量化に対応したアルゴリズム

2.3 テンポラルブロッキング

テンポラルブロッキングは、ステンシル計算のメモリ局所性を向上させるための手法である [6], [9]．ここでは計算対象の配列を小さい部分空間に区切り，その部分空間について，時間ステップ計算を他と独立に複数回進める．これにより，時間ステップ毎に配列全体をスキャンする通常の方法よりも局所性が良好である．もともとはキャッシュヒット率を向上させるために提案された手法であるが，GPU のデバイスメモリを超えるステンシル計算の隣接通信で必要な CPU-GPU 間の通信回数を減らすためにも用いられている [12], [13].

テンポラルブロッキングを用いて図 1(b) を改良すると図 2 のようなアルゴリズムとなる．部分領域ループの中に更に内部時間ループが追加されていることがわかる．時間ブロッキングする時間幅をテンポラルブロッキングサイズ k とし、時間発展する回数を t とおくと、各部分領域が順番に内側の時間ループで k 回計算するので、外側の時間ループは t/k 回の計算となる．図 1(b) と比べると、PCI-Express 通信を 1/k 回に削減している．一回あたりの転送量はほぼ変わらない (どちらも部分空間全体＋袖領域) ので，計算全体の PCIe 通信量もほぼ 1/k とできる．

しかしながら図 1(a) のような既存アプリケーションが存在するときに，図 2 のように書き換えるのはプログラミングコストが高く，これを改良するために，次に述べるメモリスワップランタイム HHRT を用いる．

2.4 HHRT

HHRT(Hybrid Hierarchical Runtime)[4], [5] は、メモリ

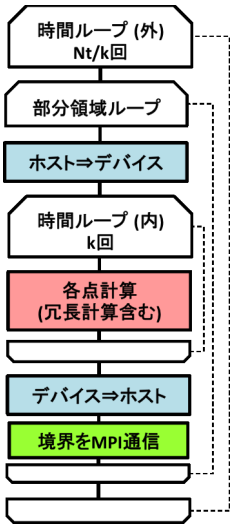

図 2 大規模対応テンポラルブロッキングのアルゴリズム

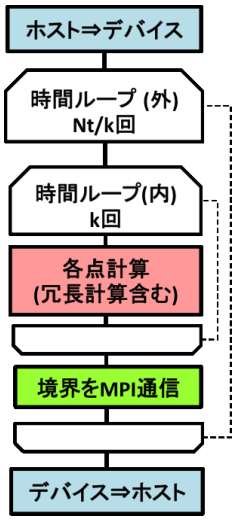

図 3 HHRT とテンポラルブロッキングの組み合せのアルゴリズム

階層間のメモリスワップをサポートするランタイムライブラリで、メモリスワップを必要とするアプリケーションのプログラミングコストを抑えることを目的としている．HHRT は、現在、CUDA と MPI でコーディングされたアプリケーションを対象に、GPU-CPU 間のメモリスワップをサポートするランタイムライブラリである．

HHRT を利用した場合の特徴は複数の MPI プロセスが単一 GPU の計算リソースを共有していることであり，これらのプロセス間合計ではデバイスメモリ容量を超えるデータを扱うことができる．合計利用メモリ量がデバイスメモリ容量を超える場合には，プロセス単位での (OS のようにページ単位ではない) スワッピングを行う．詳細は過去の文献を参照されたい [4], [5].

この HHRT 単体はデータの追い出しを自動化する一方，テンポラルブロッキングのようなアルゴリズムの改良を行うものではない．そのためその組み込みは依然ユーザの責任であるものの，図 2 よりも単純に，図 3 のように記述することができる [5].

2.5 都市気流シミュレーション

今回大規模化に使用する小野寺らが開発した都市気流シミュレーション [2] について説明する．これは、東京の 10Km 四方のエリアを対象に、実際の建造物のデータを使い 1m 間隔の格子解像度で気流シミュレーションを行い、高層ビルの背後に発生する渦によるビル風や幹線道路に沿って流れる風の道などを再現する実ステンシルアプリケーションである．格子ボルツマン法 (LBM:Lattice boltzmann methd) とよばれる手法が使われている．LBM は、連続対として記述された流体に対して、離散化された空間格子状において、並進と衝突をする仮想的な粒子の集合を速度分布関数として仮定し、格子状の粒子の速度分布関数について時間発展を解いて行く手法である．19 方向もしくは 27 方向の速度を，それぞれ格子データで表現するのが特徴の一つである．

7 点ステンシルプログラムと都市気流シミュレーションの違いは、以下のような点が挙げられる．

- 7 点ステンシルは、格子点をダブルバッファリングのための 2 つの 3 次元配列で表すのに対し、LBM は 3 次元配列を 19 ないし 27 個，さらにダブルバッファリングのために 2 倍の個数用いる．
- 都市気流シミュレーションは 12 個の物理量のための三次元配列を必要とする．
- 7 点ステンシルは 1 つのカーネル関数のみ用いるのに対し、都市気流シミュレーションは各ステップあたり 7 つのカーネルを順次用いる．
- 境界条件は、7 点ステンシルは単純なディレクレ条件を用いる (境界値は固定) のに対し、都市気流シミュレーションは、X 軸を周期的境界条件、Y と Z 軸はノイマン境界条件を使って計算する．

以上のように、計算に必要なデータ数、1 ステップあたりの計算、境界条件が主な違いとなっており、コードの行数も 10,000 行を超える大規模なものである．

3. HHRT とテンポラルブロッキングによる大規模化手法

HHRT とテンポラル・ブロッキングの導入による大規模化手法 [3] について，都市気流シミュレーションへの導入を例に説明する．

3.1 HHRT の導入

第一段階として，既存実装をほぼそのまま HHRT 上でコンパイル・実行可能とするために下記の小さなコード変更を行う．

- hhrt.h のインクルードの追加
- main 関数、MPI、CUDA のランタイム API の関数名

に、HH() の付加
- HH_initHeap() 関数による各プロセスの最大利用メモリ量の宣言

なお将来は HHRT の改良により，上記のほとんどが不要となり，透過的に利用可能とする計画である．上記を行い，HHRT ライブラリとリンクすることにより、GPU メモリサイズを超えた問題サイズにおいてもプログラムを実行することがひとまず可能となる．

HHRT を使わない場合には，大規模問題サイズの対応のためには，配列の部分空間への分割および，明示的な CPU-GPU 間の転送の記述などのために，図 1(b) のようなプログラム全体に係るループ構造の変更が必要となってしまう．これに対し，HHRT の利用により，上記のようなコード内の局所的・機械的な変更により，プログラミングコストを抑えつつ，同様の効果を得ることができる．

3.2 テンポラルブロッキングにむけた予備調査と実装

テンポラルブロッキングは、アクセス局所性を向上させるアルゴリズムであるが，どんなステンシル計算にも適用可能なわけではない．テンポラルブロッキングは，ある時刻における点 x の計算が，「直前の時刻の，点 x の近傍の値にのみ依存」するような計算の場合に適用可能である．これに反する例としては，共役勾配法のように，各ステップにおいて領域全体に係る内積が必要な場合が挙げられる．また，各時間ステップにおいて広範囲の平均操作などが必要な場合も障害となる．

今回の都市気流シミュレーションにおいて，上記の条件を満たすアルゴリズムであるか否かについて，カーネル関数を中心としたコード精査により調査した．その結果，各点の計算は近傍にのみ依存することが確認でき，テンポラルブロッキングは適用可能であると判断した．

上記の判断のもと，都市気流シミュレーションについて，図 1(a) の構成から図 3 の構成への変更を手作業で行った．ここで，HHRT を利用することにより，コード変更は比較的局所的なもので済んだ．具体的には，下記のような変更である．

- 時間ループの開始箇所と終了箇所を変更し，時間ループの二重化を行う．
- 内側時間ループの進行にしたがって，計算対象に含める袖領域が一段ずつ狭まっていくような，空間ループ開始・終了点の調整を行う．
- MPI による境界通信を，外側時間ループと内側時間ループの間に移動し，さらに k 段の境界を一気に通信するよう変更する．
- 境界条件については，ノイマン条件の計算は内側時間ループの中にとどめたのに対して，周期境界条件のための MPI 通信を，前項目と同様に，外側時間ループと内側時間ループの間で行う．

この結果，主な変更は各ループの開始箇所と終了箇所に集中し，実アプリケーションにおいてコードを長くする主要因であるループ内の計算ロジックについては，変更は不要となる．さらに，コードの移動も上記 3 番目，4 番目の点に抑えられる..

3.3 HHRT のスワップ動作に合わせた MPI 通信の最適化

上記のステップまでで，基本的には大規模・高性能・高生産性は実現されたが，ここではさらなる性能最適化について述べる．導入アプリケーションの通信アルゴリズムによって，HHRT のスワップ処理が多く行われ，性能向上の妨げとなる場合がある．都市気流シミュレーションでは，調査の結果，元プログラムの実装の性質と，HHRT の性質の相性が悪く，以下のような問題が起こっていた．

HHRT のスワップ処理は、ブロッキング通信命令や MPI_Wait 命令等が呼ばれたときに発生するため、複数回の通信が呼ばれるアプリケーションでは、この複数回のスワップ処理が性能低下の原因となってしまう．都市気流シミュレーションでは、全体空間は 3 次元にプロセス間分割されており，19 方向ないし 27 方向の近傍値を用いるためには，各プロセスは隣接する 26 プロセスとの境界通信を必要とする．元プログラムでは、X、Y、Z 方向の通信を順番に行う 3 回の MPI_Waitall 命令を呼んでおり、MPI 通信のためだけに 3 回のスワップ処理を行うような実装となっていた．通常の (HHRT を使わない) 実行ではこれは大きな問題とはならなかったが，HHRT の「MPI 通信がブロックする箇所では，プロセスがスワップアウトしうる」という性質により性能上の問題が発生する．つまり，一度スワップアウトすれば十分なところ，3 回スワップアウトが起こり，PCI 通信量を 3 倍に押し上げてしまっている．

これを改善するために，MPI_Waitall を 1 回の通信にまとめるコード変更を行った．つまり，26 個の全ての隣接プロセスに対して同時にノンブロッキング通信を起動することとした．各隣接プロセスに対して MPI_Isend と MPI_Irecv が起こるため，52 個の通信となる．その後，すべての通信を一回の MPI_Waitall で待つように変更を行うことで改善する．

4. HHRT のスワップデータサイズの削減による性能最適化

本研究では，HHRT とテンポラルブロッキングを導入したアプリケーションに対して，更なる性能向上のために，HHRT のスワップデータサイズの削減による性能最適化について提案する．

4.1 スワップ処理時におけるステンシル計算の不要データ

HHRT のスワップ処理は，計算で扱うすべてのデータを

転送するため，通信コストがとても高い．しかし，全てのデータが，計算再開時に必要である訳ではないため，無駄が多い．このため，スワップ処理時のデータ転送量を減らすことで，性能向上が期待できる．この節では，ステンシルアプリケーションにおける，削減されうる配列について説明する．

ステンシル計算において，次ステップを計算するのに必要なデータは，値の更新に必要な，前ステップの値である．ステンシル計算は，一般的に，ダブルバッファリングを使った計算を行っている．ダブルバッファリングは，一方のバッファから値を読み込み，計算した値をもう一方のバッファに書き込む計算となっている．そして，次の計算には，２つのバッファの役割を入れ替えることによって，同様の計算が可能となる．このため，１ステップの計算を終えた，読み込みバッファは，次のステップでは，書き込みバッファとなるため，読み込みバッファ内のデータは，どのような値であっても構わない．

ステンシル計算の境界通信は，一般的に，値の更新後に発生するため，HHRT のスワップ処理時に格子点の更新は完了している．このため，スワップ処理時に，２つのバッファの内，読み込みバッファは，値の保持の必要がないため不要なデータなので，削減が可能となる．

7 点ステンシルのような，３次元配列が２つのみであれば，削減率は半分となるが，大規模なステンシルアプリケーションとなると，様々なデータを使った計算を行うために，配列数が多くなり，１つのバッファの削減だけでは，削減率が低くなってしまう．

大規模なステンシルアプリケーションでは，ダブルバッファリングと同様に，次ステップで，値が参照されない配列が存在する可能性がある．複数のカーネル計算存在するようなアプリケーションがあり，あるカーネルの計算結果を使うような計算がある場合には，計算結果の格納に使われる配列は，次ステップで値が参照されることがなく，1 ステップの計算終了時には，値の保持が必要ない不要なデータとなるため，スワップ処理時に削減可能となる．

都市気流シミュレーションでは, 格子ボルツマン法の D3Q19 モデルによる配列（3 次元配列 19 個）を含め，３次元配列が 52 個ある．この中で，削減可能な配列は，27 配列あり，約半分のデータを削減できる．

4.2 HH_madvise 関数による実装

HHRT には、HH_madvise という API があり、スワップイン・スワップアウトの時のデータ転送をするデータを選択することができる．HH_madvise 関数は、配列の先頭アドレス、データサイズ、データの状態を引数として渡すことで、そのデータをスワップイン・スワップアウトするか選択することができる．データの状態は HHMADV_NORMAL と HHMADV_CANDISCARD の２状態があり、それぞれスワップイン・スワップアウト時に、配列データの値を保証するかしないか、つまり、スワップイン・スワップアウト時にデータ転送をするかしないかを表している．これにより、値の保証が必要ないデータをスワップイン・スワップアウト時に転送する必要がないため、通信時間を削減することができる．この最適化の実装にかかるプログラミングコストは，1 配列あたり，2 行の HH_madvise 関数の追加のみで可能となるため，ほぼコストがかからず最適化可能となっている．

5. 性能モデルの構築

提案手法を導入したアプリケーションを実行するためには，1GPU に処理させる問題サイズに対して，領域の分割数，つまり 1GPU を共有する MPI プロセス数と最適なブロッキング段数を指定する必要がある．このため，最適な MPI プロセス数とブロッキング段数を推定するための性能モデルを構築する．

MPI+CUDA のアプリケーションは，主に cudaMemcpy の HostToDevice と DeviceToHost，そして Kernel 計算と MPI 通信の 4 つから構成されている．今回のモデル構築では，MPI 通信は，HHRT のメモリスワップ処理の実行時間で隠蔽できると仮定して構築する．そのため，HostToDevice，DeviceToHost，Kernel 計算の３つからなる性能モデルを構築する．

モデル構築にあたり各実行時間は次のように算出した．cudaMemcopy は，HostToDevice と DeviceToHost のそれぞれの実行バンド幅を測定し，実効バンド幅とデータサイズにより，実行時間を算出している．同様に，Kernel 計算も実行 Flops を測定し，実行 Flops と flop 数から算出している．ただし，MPI 通信のためのバッファへのコピーは，実行バンド幅を測定し，実行バンド幅とデータサイズから算出している．

提案手法とスワップデータ削減の最適化を行った都市気流シミュレーションの動作モデルを図 4 に示す．各図は，1GPU を共有する MPI プロセス数 n，1GPU のメモリ資源を同時共有出来るプロセス数 m，ブロッキング段数 k としたときの，m と各実行時間の関係により場合分けしており，図は $n = 4$ の時を表している．図の長方形は，各時間を表しており，それぞれの色はスワップイン (赤)，スワップアウト (青)，計算 (オレンジ)，実行待ち (緑) を表し，P0〜P3 はランクを表している．また，メモリ以外の各ハードウェア資源は複数プロセスで同時に占有することは出来ない (スワップインとスワップアウトは同時実行可能) ため、その場合，実行待ちとなっている．

各図の場合分けに対応したモデル式を以下に示す．

$$T_{Iter}(1, n, k) = n(T_{INk} + T_{CALk} + T_{OUTk}) \quad (1)$$

$$T_{Iter}(2, n, k) = \frac{n}{2}(T_{INk} + T_{CALk} + T_{OUTk}) \quad (2)$$

$$T_{Iter}(2,n,k) = n\max\{T_{INk}, T_{CALk}, T_{OUTk}\} \quad (3)$$

$$T_{Iter}(m,n,k) = n\max\{T_{INk}, T_{CALk}, T_{OUTk}\} \quad (4)$$

式 $T_{Iter}(m,n,k)$ は、各図の矢印の時間を表している．各式はそれぞれ同時共有メモリ数 m によって場合分けされており，それぞれ (1)$m=1$，(2)(3)$m=2$，(4)$m \geq 3$ の 3 つからなり，m=2 の時，$T_a \geq T_b + T_c$ となる T_a が (2) 存在しない，(3) 存在するという条件によって式が異なる．また，T_{INk}、T_{OUTk}、T_{CALk} は，それぞれブロッキング段数 k の時のスワップイン，スワップアウトの転送時間と計算時間を表しており，スワップイン，スワップアウトは，MPI 通信に必要なバッファ領域の転送時間を含み，計算時間は，MPI 通信に必要なバッファと配列間のコピー時間を含んでいる．

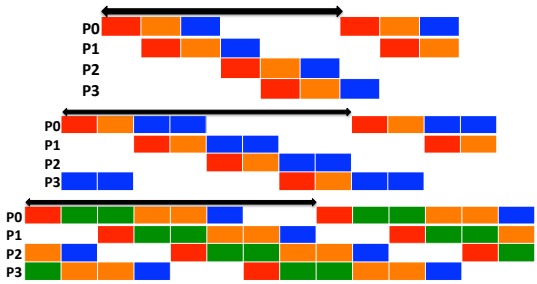

図 4 m=2 および 3 のときの実行モデル

6. 性能評価

6.1 評価環境

今回の性能評価のために、東京工業大学学術国際情報センターのスーパーコンピュータ TSUBAME2.5 の Thin ノードを使って測定した．Thin ノードの構成は、Intel Xeon 5670 2.93GHz(6 cores) プロセッサを 2 ソケット、NVIDIA Tesla K20X を 3 基搭載しており、ホストメモリの容量 54GB(一部 96GB) となっている．各計算ノード間は、Infiniband QDRx4 によって接続されている．今回の評価実験では、1 ノードあたり 1GPU を使用し、1 ノードでの実験はホストメモリ 96GB のノードで実験し、複数ノードを使用した実験はホストメモリ 54GB のノードで実験を行った．また、開発環境は、OS が SUSE Linux 11 sp1、コンパイラが gcc 4.3.4、MPI が OpenMPI 1.6.5、そして CUDA は 6.0 を使用した．

6.2 1GPU での評価

GPU のデバイスメモリを超えるデータ処理の性能を評価するために、ホストメモリ容量 96GB の Thin ノードを使用した．比較対象のプログラムの詳細を以下に示す．

- NORMAL:提案手法を一切使わない元プログラム
- HH:NORMAL に HHRT を導入したプログラム
- HH_TB:HH にテンポラル・ブロッキングを導入したプログラム
- HHTB_MPI:HH_TB を MPI 通信の最適化したプログラム
- HHTBMPI_OPT:HHTB_MPI にスワップ処理の最適化を行ったプログラム
- CPU:Open MP で並列化した CPU 版のプログラム

測定対象となる各プログラムを，問題サイズを変化させた時の性能の推移を図 5 に示す．図にプロットされた性能は，最適なブロッキング段数と分割数の時の性能を使用した．

実験で利用した GPU K20X は，デバイスメモリ容量が 6GB であるため，NORMAL は，6GB 以上の問題サイズを実行することができないことがわかる．

NORMAL に HHRT を導入した HH は，デバイスメモリ容量である 6GB の制限を超え，約 48GB の問題サイズまでの実行を可能にしている．しかし，6GB を超えると NOMAL の 5%程度まで性能が低下しており，これはスワップ転送が原因である．

HH のスワップコストを削減するためにテンポラル・ブロッキングを導入した HH_TB は，HH に比べ最大 4.9 倍の性能向上し，NOMAL の最大 27%の性能となっている．

ここで，HH_TB のスワップ回数を調べると，1 ステップあたり 3 回のスワップを行っていることがわかった．HH_TB は、1 ステップあたり 3 回の MPI のブロッキング通信が行われていた．このため，HHRT のスワップ条件によって，3 回のスワップを行い性能が低下していた．そこで、HH_TB の MPI のブロッキング通信を 1 回にまとめる最適化した HHTB_MPI は，HH_TB に比べ最大 2.4 倍の性能向上し，NORMAL の最大 64%を達成した．この結果から，複数回の MPI 通信を行うアプリケーションに対して，MPI 通信をまとめる最適化が必要であることがわかる．

HHRT の実行モデルでは，MPI プロセスのスワップ処理を行う必要があるため，計算より速度の遅い PCIe 通信が実行時間の大半を占める．そこで，PCIe の通信時間を減らすために，通信データ量を削減する最適化をした HHTBMPI_OPT は，HHTB_MPI に比べ最大 1.9 倍性能向上し，NORMAL に対して最大 85%の性能を達成した．

容量の増加と伴に性能が低下している原因は，HHRT の MPI プロセスの管理コストとホストメモリ容量である．HHRT の MPI プロセスの管理コストは，複数の MPI プロセスが 1GPU を共有するのに必要なコストで，1MPI プロセスあたり約 72MB の管理領域を GPU メモリ上に確保するため，分割数が多くなると GPU メモリを圧迫してしまう．ホストメモリ容量による制限は，HHRT は，ホストメモリの半分の容量までの問題サイズしか扱えないためである．どちらの場合も，ブロッキング段数を多くするとメモリ容量が大きくなるため，ブロッキング段数を増やすことによる性能向上ができない．

しかし，HHTBMPI_OPT と CPU を比較すると，HHTBMPI_MPI は，問題サイズ 48GB の時，ブロッキング段数を挙げられないが，スワップデータの削減によって，CPU の約 1.7 倍の性能を達成を達成しており，提案手法を導入した方が優位であることがわかる．
以上の結果から，提案手法を導入したアプリケーションは，スワップデータサイズの削減により，より有用な手法であることが示せた．

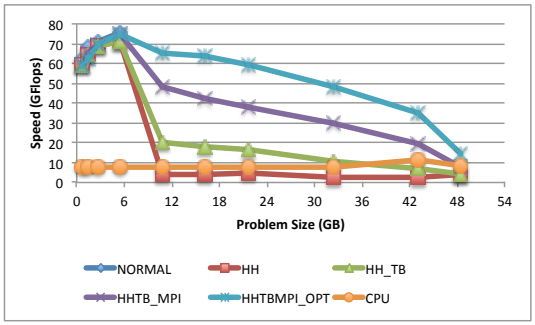

図 5 1GPU での実行結果

6.3 ブロッキング段数の評価

図 6 は，ブロッキング段数を変化させたときの HHTB_MPI と HHTBMPI_OPT の性能の変化を表したグラフである．HHTBMPI_OPT はスワップデータサイズが削減されたことで，最適なブロッキング段数が HHTB_MPI と異なり，また，HHTB_MPI より少ないブロッキング段数で，最適値となっていることがわかる．また，最適値以降性能が低下している理由は，ブロッキング段数を増やすことによる冗長計算の増加によって，計算コストが，スワップ時間などの通信コストを上回ったためである．

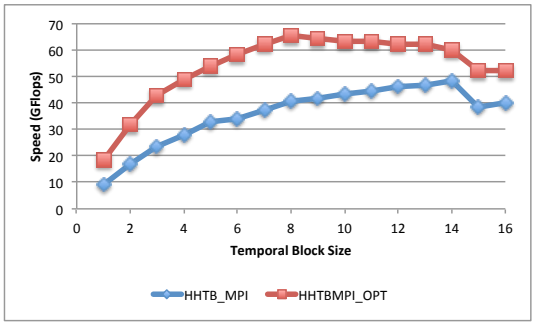

図 6 ブロッキング段数による影響

6.4 複数ノードでの評価

図 7 は、1 ノードあたり 1GPU を使用して，ノード数を変化させていった場合のウィークスケーリング性能を表している．問題設定は，NORMAL が 5.5GB で，OPT は HHTBMPI_OPT と同じプログラムで，OPT_1,2,3 は問題サイズの違いを表しており，それぞれ 5.5GB, 10.8GB, 16.2GB である．NORMAL と OPT_1 を比較すると，HHRT や TB の 2 重ループのコストによって OPT_1 は，数%の性能劣化しているが，ほぼ同等の性能で，スケーラビリティの劣化が無いことがわかる．
全体で比較すると，OPT_2,3 は，ほぼ同等の性能で，スケーラビリティも高く，NORMAL の約 80%程度の性能を維持していることから，提案手法は，複数ノードを使用した大規模環境にも対応していることがわかる．

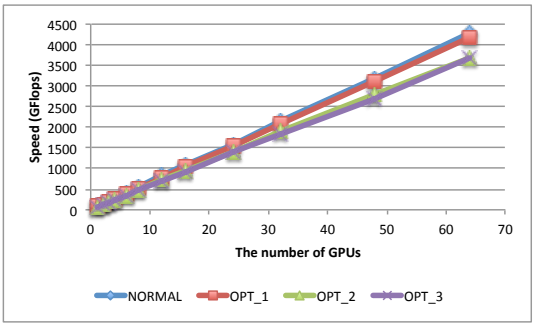

図 7 複数ノードでの実行結果

6.5 性能モデルの評価

提案手法を適応したアプリケーションの最適な性能を予測する性能モデルの評価を行った．対象プログラムは、HHTBMPI_OPT で，問題サイズは，10.8GB に設定し，分割数とブロッキング段数を変化させたときの性能モデルの予測値と HHTBMPI_OPT の実測値を比較し，その結果を図 8 に示す．図 8 は，分割数 8 と 16 のときの 1 ステップあたりの実行時間を表しており，　横軸はブロッキング段数、縦軸に実行時間をとっている．性能モデルの実行時間は，$T_{step}(m,n,k) = T_{Iter}(m,n,k)/k$ で算出している．予測値と実測値の差はあるものの，ブロッキング段数の変化に伴う実行時間の変化をかなり表現できていると考えられる．実測値の最適解は，$(n,k) = (16,8)$ のときで，これは予測値の最適解に一致している．以上の結果から，構築した性能モデルは，パラメータの絞り込みが可能であると考えられる．

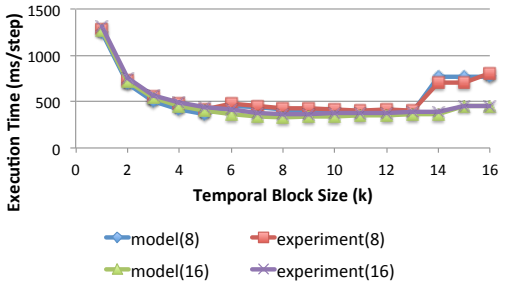

図 8 8 プロセスと 16 プロセスの性能モデルの評価

7. 関連研究

テンポラルブロッキングを使った GPU デバイスメモリの容量制限を超えるためのデバイスメモリとホストメモリ間のデータ移動を削減する研究として，Mattes らの FDTD の実装 [11] が挙げられる．この研究は，手動で，テンポラルブロッキングと，部分領域ループを追加しており，10%程度の性能低下で大規模かを実現している．さらに我々の研究グループでは，多数 GPU・多数ホスト環境におけるテンポラルブロッキングの採用により，大規模・高性能ステンシル計算が性能劣化なく可能であることを実証している [12], [13].

ステンシル計算の記述を容易にすることを目指したアプローチとして，Physis[14] などのドメイン固有言語 (DSL) を用いることが考えられる．Physis を使った GPU での大規模なステンシル計算の研究として，河村らはテンポラルブロッキングコードの自動生成 [15] を行っており，手動コードより少ないコード数での記述を実現している．

8. まとめと今後の課題

本研究は，ステンシル計算を対象に，提案手法である HHRT とテンポラルブロッキングの導入に対し，更なる性能最適化のために，HHRT のスワップデータを削減させる手法を提案した．これにより，性能が約 1.3〜1.9 倍の性能向上を実現し，プログラムの約 19〜85%の性能を達成した．また，提案手法の導入によって，現れるパラメータの最適解の絞り込みを行うために，性能モデルの構築を行った．性能モデルによる予測値と実測値の比較を行ったところ，誤差は見られたものの，最適解の絞り込みが可能であることを確認した．

今後の課題として，まず，更なる性能向上のために，通信と計算のオーバーラップへの対応が必要である．また，問題サイズ増加に伴う性能劣化に対応するために，HHRT の MPI プロセスの管理方法の改善と，SSD，HDD などへの対応が必要である．

謝辞 本研究は JST-CREST の研究課題「ポストペタスケール時代のメモリ階層の深化に対応するソフトウェア技術の深化に対応するソフトウェア技術」および科学研究費助成事業（基盤研究（S）23220003）の支援によります．

参考文献

[1] Takashi Shimokawabe, Takayuki Aoki, Tomohiro Takaki, Akinori Yamanaka, Akira Nukada, Toshio Endo, Naoya Maruyama, Satoshi Matsuoka: Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer. In Proceedings of IEEE/ACM Supercomputing (SC11), 11pages (2011).

[2] 小野寺直幸, 青木尊之, 下川辺隆史, 小林宏充: 格子ボルツマン法による 1m 格子を用いた都市部 10km 四方の大規模 LES 気流シミュレーション. 情報処理学会ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS2013), pp. 123–131 (2013).

[3] 高嵜祐樹，遠藤敏夫，松岡聡: GPU クラスタ上の実ステンシルアプリケーションの大規模化に向けた局所性向上の評価. 情報処理学会研究報告 2014-HPC-146 No 23, 8pages (2014).

[4] 遠藤敏夫: 並列プログラムをメモリ階層利用可能とするランタイム．情報処理学会研究報告 2013-HPC-140 No 43, 8pages (2013).

[5] T. Endo and G. Jin: Software Technologies Coping with Memory Hierarchy of GPGPU Clusters for Stencil Computations. In Proceedings of IEEE Cluster Computing (CLUSTER2014), 8pages (2014).

[6] Michael E. Wolf and Monica S. Lam: A Data Locality Optimizing Algorithm. In proceedings of ACM PLDI 91, pp. 30–44 (1991).

[7] David Wonnacott: Using Time Skewing to Eliminate Idle Time due to Memory Bandwidth and Network Limitations. In proceedings of IEEE IPDPS 2000, pp. 171–180 (2000).

[8] M. Wittmann, G. Hager, and G. Wellein: Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory. Workshop on Large-Scale Parallel Processing (LSPP10), in conjunction with IEEE IPDPS2010, 7pages (2010).

[9] T. Minami, M. Hibino, T. Hiraishi, T. Iwashita and H. Nakashima: Automatic Parameter Tuning of Three-Dimensional Tiled FDTD Kernel. In Proceedings of The Ninth International Workshop on Automatic Performance Tuning (iWAPT2014), 8pages (2014).

[10] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In Proceedings of IEEE/IEEE Supercomputing (SC10), 13pages (2010).

[11] L. Mattes and S. Kofuji: Overcoming the GPU memory limitation on FDTD through the use of overlapping subgrids. International Conference on Microwave and Millimeter Wave Technology (ICMMT), pp.1536–1539 (2010).

[12] G. Jin, T. Endo and S. Matsuoka: A Multi-level Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPU. AsHES workshop, held with IEEE IPDPS2013, 8pages (2013).

[13] G. Jin, T. Endo and S. Matsuoka: A Parallel Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPUs. In Proceedings of IEEE Cluster Computing (CLUSTER2013), 8pages (2013).

[14] N. Maruyama, T. Nomura, K. Sato, and S. Matsuoka: Physis: An Implicitly Parallel Programming Model for Stencil Computations on Large-Scale GPU-Accelerated Supercomputers, IEEE/ACM Supercomputing (SC11), 12pages, Seattle (2011).

[15] 河村知輝，丸山直也，松岡聡: 自動テンポラルブロッキングによる大規模ステンシル計算の実現. 情報処理学会研究報告 2014-HPC-143 No 32, 6pages (2014).