GPUクラスタにおける大規模都市気流シミュレーションの最適化と性能モデル

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，約 15,000 行のコード規模を持つ 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について述べる．プログラミングコストを抑えるために，自動で GPU メモリとホストメモリ間でメモリスワップを行うランタイムライブラリである HHRT を用いる．その条件下で性能劣化を防ぐための最適化を行うことにより，元プログラムの最大 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．

Optimization and Performance Model for Large-scale Wind Simulation in Metropolitan on GPU Clusters

Yuki Takasaki[1]　Toshio Endo[1]　Satoshi Matsuoka[1]

Abstract: The scale of stencil computation in GPU is usually limited by the memory size of GPU. Temporal blocking technique is used to break this limitation without performance degradation. Urban turbulence simulation code for GPU cluster, which consists of 15,000 lines, are modified to exploit the scale, keeping its performance. To reduce programming cost, HHRT runtime library is used to swap data between host memory and GPU memory automatically. With optimization to prevent performance degradation, we achieved up to 85% of the performance of the original program. We also revealed parameters affecting the performance by constructing performance model on memory swapping of HHRT.

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 に代表されるメニーコアプロセッサの進歩により，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．

本論文では，GPGPU 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，シミュレート対象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 上の実行に向いており，多くの成功例をもつ [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た．シミュレーションの分野において求められる，大

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

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．

本論文では，GPU 搭載スーパーコンピュータ向けに実装された既存の流体アプリケーションを対象とし，MPI と CUDA を用いて記述された都市気流シミュレーションソフトウェア [2] を取り上げ，大規模性・高性能性・高生産性を実現する。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でき，大規模性を達成する．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] 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（今回のアプリケーションは約 15,000 行から成る）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 搭載スーパーコンピュータである TSUBAME2.5 上での性能評価を通して，実際のステンシルアプリケーションに対して，大規模化，高性能化，高生産性の 3 つを達成することを示す．

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 アクセラレータや Intel Xeon Phi の利用が広がっており，本論文では 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である傾向にある．デバイスメモリ容量を超えるためには、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．シミュレートされ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実装例を図 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．

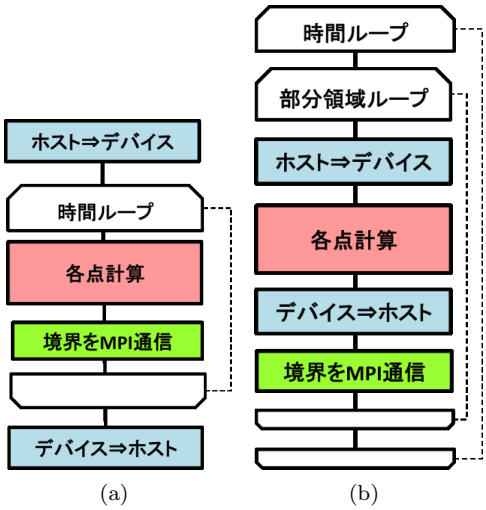

図 1 (a) 通常の MPI+CUDA、(b) 単純な大容量化に対応したアルゴリズム．

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 PCI-Express 通信量もほぼ $1/k$ にできる．

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．

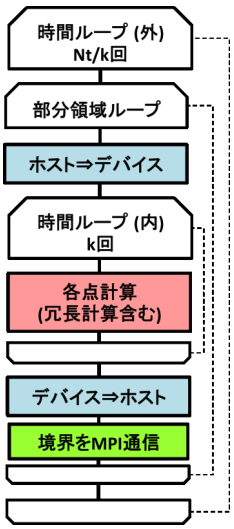

図 2 大規模対応テンポラルブロッキングのアルゴリズム

2.4 HHRT

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

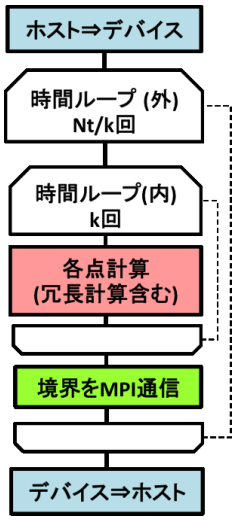

図 3 HHRT とテンポラルブロッキングを組み合わせたアルゴリズム

3. 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また16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6つのカーネルを順次計算する．
- 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，コードの行数も約15,000行となる大規模なものである．

4. 大規模・高性能・高生産性の実アプリケーションにおける実現

都市気流シミュレーションの提案手法であるHHRTとテンポラルブロッキングの導入について説明する．本アプリケーションの既存実装は，前章に述べたようにベンチマークより複雑であるものの，基本的には図1(a)のような構成を持っている．これを，図3のようにソフトウェアを改造し，HHRT上で実行を行う．

4.1 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2つの小さなコード変更を行う．

- hhrt.hのインクルードの追加
- HH_initHeap()関数による各プロセスの最大利用メモリ量の宣言

上記の変更を行い，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．

4.2 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った．
- 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・終了点の調整を行った．
- 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した．
- 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った．

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，変更は不要であった．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た．その結果，テンポラルブロッキング導入におけるコード変更量は，全体コード量約15,000行に対して，約300行に抑えられた（元プログラムソースとのdiffに基づく計量）．

4.3 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．詳細は評価の章に譲るが，上記の基本的な実装をGPU搭載スーパーコンピュータで実行したところ，予想よりもHHRT内部のスワップ処理回数が3倍程度多くなってしまい，そのため性能が期待よりも大きく低下していた．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ることがわかった．

HHRTのスワップ処理は、ブロッキング通信命令が呼ばれたときに発生するため，複数回ブロッキング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 回のブロッキング通信を行っており，MPI 通信のためだけに 5 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した．つまり，1 度スワップアウトすれば十分なところ，3 回のスワップアウトと 2 回のスワップインが起こり合計 5 回のスワップ処理となり，PCI-Express 通信量を 5 倍に押し上げてしまっている．

これを改善するために，MPI_Waitall1 回にまとめるような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 1 回の MPI_Waitall で待つように変更を行った．この最適化にともなうコード変更量は約 800 行であった．前節のテンポラルブロッキング自身の組み込みよりも大きくなったが，これは三次元の隣接通信を行う元プログラムの通信部分がもともとコードがやや複雑だったためである．

4.4 HHRT のスワップデータ量削減による最適化

HHRT のスワップ処理は，スワップ処理時に GPU 上に確保したデータを保護するために，ホストメモリへデータを退避する．しかし，ステンシル計算において，次ステップで全てのデータを保持する必要がないことが調査の結果わかった．このため，スワップ処理では，無駄なデータ転送を行っているため，無駄なデータ転送を削減することで，通信時間を減らすことができる．

そこで，HHRT には，HH_madvise という API があり，スワップイン・スワップアウト時のデータ転送において，転送しないデータを選択することで，そのデータを転送しないようにすることができる．

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．このようにして，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せず，転送時間を削減することができる．

転送不要なデータとして，ステンシル計算の場合，ダブル・バッファリングの一方は，値の保証の必要がないため，一方の転送を無視することができる．都市気流シミュレーションなどの実アプリケーションの場合，配列数が多くバッファの一方だけでは性能向上率は少ない．しかし，次ステップで使われる値を格納しない配列があるアプリケーションの場合，計算ステップ終了後は、値を保持する必要がないため，スワップイン・スワップアウト時にデータ転送の必要がなくなるため，転送時間の削減ができる．

都市気流シミュレーションでは，このような配列が複数あったため，HH_madvise による最適化を行うことができる．１つの配列データに対するこの最適化に伴うコード変更は最低 2 行である．

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あらかじめ算出しておく必要がある．cudaMemcopy は，HostToDevice と DeviceToHost のそれぞれの実行バンド幅を測定し，通信するデータサイズにより実行時間を算出する．同様に，カーネル計算も実行 Flops を測定し，その flop 数から算出している．ただし，MPI 通信のためのバッファコピーは，実行バンド幅とデータサイズから算出する．

提案手法と通信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 から図 7 に示す．各図は，1GPU を共有している MPI プロセス数 n，1GPU が同時にメモリ資源を共有出来るプロセス数 m，ブロッキング段数 k としたときの各処理実行時間を表している，また，各図の長方形が，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 MPI プロセスの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ため、競合が起こった場合に実行待ちとなっている．ただし，スワップインとスワップアウト処理は，同時に実行することが出来る．

図 4 から図 7 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，各図の黒矢印間の時間を表している．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カーネル計算時間を表しており，スワップイン，スワップアウトの転送時間には，MPI 通信に必要なバッファ領域の CPU-GPU 間の転送時間を，カーネル計算時間には，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．

各式はそれぞれ 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 つに場合分けされる．$m = 2$ の時に，各実行時間の内に $T_a \geq T_b + T_c$ となるような 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．ここで，T_a，T_b，T_c には，T_{INk}，T_{OUTk}，T_{CALk} が入る．

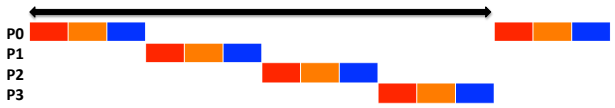

図 4 (1)$m = 1$ の実行モデル

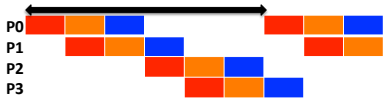

図 5 (2)$m = 2$ の実行モデル

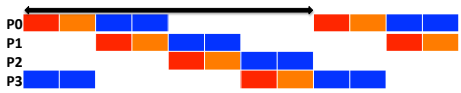

図 6 (3)$m = 2$ の実行モデル

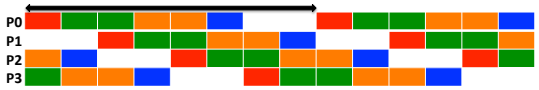

図 7 (4)$m = 3$ の実行モデル

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 HH_madvise の最適化したプログラム

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 8 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 NORMAL の 5%程度まで性能が低下している．これは、1 ステップ毎にスワップ処理が発生してしまうためである．

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，NORMAL の最大 27%の性能となった．しかし，HH_TB のスワップ回数を調査したところ，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 のコードは、1 ステップあたり 3 回の MPI_Wait 命令がよばれており，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った．

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 PCI-Express 通信が実行時間の大半を占める．そこで，PCI-Express 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 1.4 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ことが原因である．

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，MPI 通信の最適化と Hint 機能による最適化によって，性能劣化を最大 15%程度にまで防ぐことが可能であることがわかる．

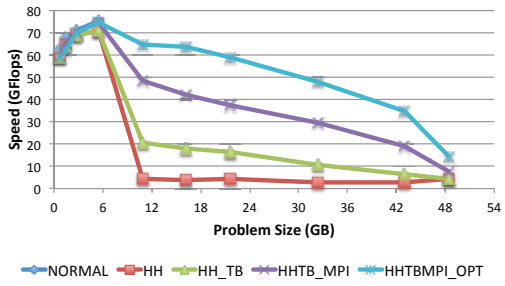

図 8　1GPU での実行結果

6.3　ブロッキング段数の評価

図 9 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 HH_madvise によってスワップ処理時の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．

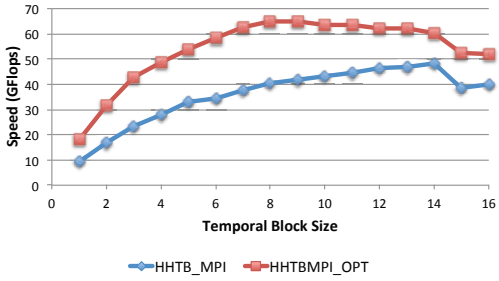

図 9　ブロッキング段数による影響

6.4　複数ノードでの評価

図 10 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，OPT_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．

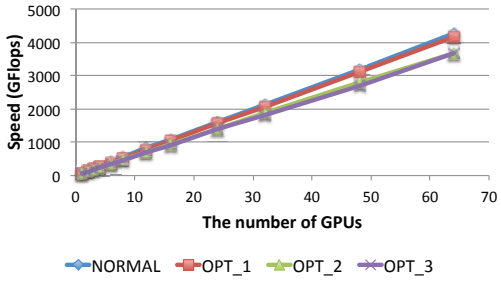

図 10　複数ノードでの実行結果

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_HINT 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_HINT の実測値を比較し，その結果を図 11 から図 14 に示す．各図はそれぞれプロセス

数 4，8，16，32 の時の 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 1 ステップあたりの予測値は，$T_{step}(m,n,k) = T_{Iter}(m,n,k)/k$ として算出した．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．ただし，4 プロセスにおいて，ブロッキング段数 1 と 2 で，大きくずれているが，これは，MPI 通信時間を考慮していないことが，原因であると考えられる．

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．

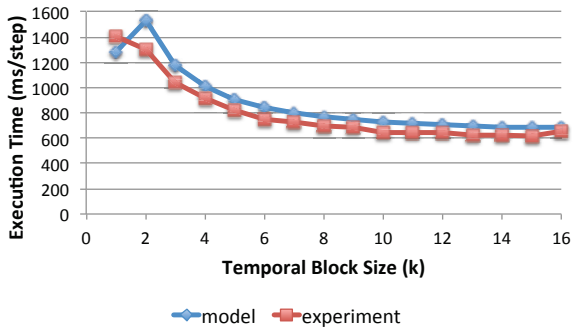

図 11　4 プロセスの場合

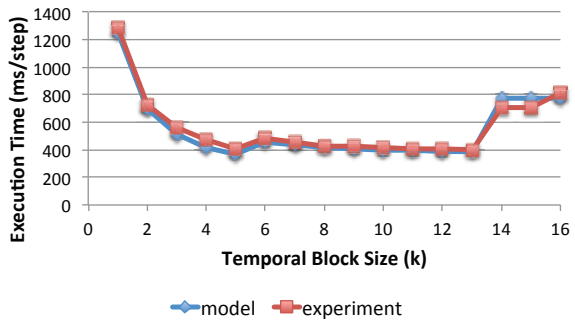

図 12　8 プロセスの場合

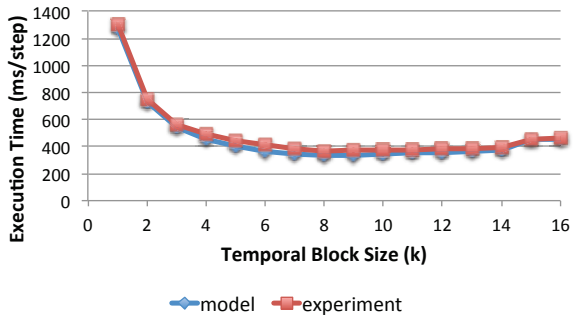

図 13　16 プロセスの場合

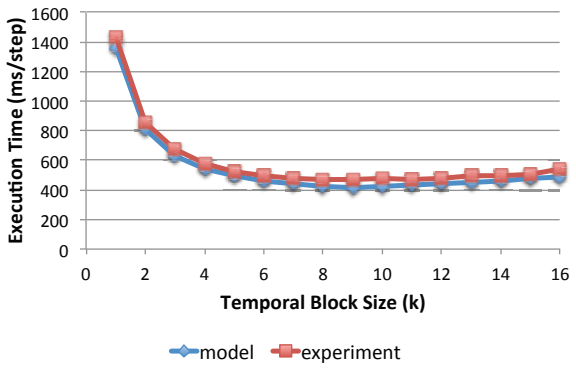

図 14　32 プロセスの場合

7.　関連研究

ステンシル計算は連続体のシミュレーションにおいて重要なカーネルであり，多数のアプリケーションがスパコン上で実装されてきた [1], [2], [3].

ステンシル計算の最適化技法についても，空間ブロッキング，テンポラル・ブロッキングなど数多く提案・評価されてきたが，以降では後者に注目する．時間方向にまたがった計算順序の変更により，メモリアクセス局所性を改良するテンポラルブロッキング手法は，当初は主に CPU のキャッシュ利用効率の向上のために提案された [6], [7], [8], [9]. また GPU においてデバイスメモリとオンチップ共有メモリ間のデータ移動を削減する研究もなされている [10]．本研究と同様に，GPU デバイスメモリの容量制限を超えるために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は Mattes らの FDTD の実装 [11] が挙げられる．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可能であることを実証した [12], [13].

しかしながら上記の研究においては，プログラミングコストの軽減についての議論が不足していると考えられる．プログラマが最適化を行う際には，時間ループ・空間ループ双方の大幅な書き変えが必要であり，それは特に複雑な構造を持つ実アプリケーションにおいて困難となると考える．それに対して我々は，MPI および CUDA で記述された実アプリケーションがすでに存在するという前提のもと，原則的に時間ループ部分の書き換えと，HHRT ライブラリ上での実行により，デバイスメモリ容量超えが可能であることを実証した．

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．このアプローチでは，アプリケーションが DSL で記述されていることを前提にしており，すでに並列化されたアプリケーションを仮定する本研究とは異なる．一方で，DSL によるアプローチでは、テン

ポラルブロッキングを含めた最適化をシステム側で行い，ユーザから隠ぺい可能であると期待される．

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，高性能，大容量，低プログラミングコストの３つを実現するための提案手法として，テンポラルブロッキングと HHRT を適用すること提案し，また，その最適化を行った．提案手法の導入により，GPU のメモリ容量を超えるデータを処理することを可能にし，元プログラムの最大 85%の実行を可能にした．これにより，本提案手法は，GPU 対象の実ステンシルアプリケーションの大規模化のための最適手法の一つであることを示した．また，性能モデルの構築により，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可能にした．

今後の課題として，問題サイズ増加に伴う性能劣化を防ぐために，HHRT の MPI プロセスの管理方法の改善と，更なる大規模化のための SSD，HDD 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] Ryoji Takaki, Kazuomi Yamamoto, Takashi Yamane, Shunji Enomoto, Junichi Mukai: The Development of the UPACS CFD Environment. In proceedings of ISHPC 2003, pp. 307-319 (2003).

[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).