Vol.2014-HPC-146 No.23
2014/10/3

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

高嵜 祐樹[1]　遠藤 敏夫[1]　松岡 聡[1]

概要：GPU の登場により、CFD で使われているステンシル計算は性能向上している．しかし、問題サイズの大きさはホストメモリより小さい GPU メモリの容量に制限されてしまっている．この問題に対して、ステンシル計算に対応したメモリアクセスの局所性を向上させる手法による解決策が提案されているが、プログラミングコストの増加が問題となっており、大規模なステンシルアプリケーションへの適応は難しいと考えられる．本研究の目的は、ステンシルアプリケーションにおける高性能化、大規模化、低プログラミングコストの３つを実現することである．その実現のために、実局所性向上アルゴリズムと CPU-GPU 間のデータ転送を自動化するメモリスワップランタイムを組み合わせたプログラミングモデルを提案する．本研究では、実ステンシルアプリケーションである都市気流シミュレーションに提案手法を適用し、その性能評価を行った．

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．これは古典的な問題であるが、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いる．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、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として、問題規模の大規模化と高性能性の両立を目的とする．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、以下の特徴を持つ．一般的にステンシル計算では、シミュレート対象領域を規則格子で表し、時間発展ループを繰り返す．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．各格子点の計算は独立に行うことができ、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．現在の GPU にはシステムソフトウェアによるスワップなどのサポートがないため、デバイスメモリを超えるデータを扱うためには、プログラマが明示的にデータ転送を記述しなければならず、また、プログラミングコストも増加してしまう．さらに、CPU-GPU 間の通信でデータが通過する PCI-Express のバンド幅は、デバイスメモリのバンド幅より数十分の一と小さいため、PCI-Express 通信量が大きいと性能低下の要因となってしまう．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、GPU 上のキャッシュ、デバイスメモリに加えて、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．さらには上記の目標を、低いプログラミングコストで実現することが求められる．

本論文では、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である都市気流シミュレーションソフトウェア [2] を対象として取り上げる。そして GPU デバイスメモリを超えつつ、高性能を保つための手法について述べる．

第一段階として、対象ソフトウェアを HHRT と呼ばれるランタイムライブラリ [4], [5] 上で動作させることにより、デバイスメモリ容量を超えた実行を可能とする。HHRT は、プロセス間の通信に MPI を、GPU のプログラミングに CUDA を使用したアプリケーションに対して、CPU-GPU 間のメモリスワップ機構を提供するものである．

しかしこれだけでは上記の PCI-Express 通信が大きなボ

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

© 2014 Information Processing Society of Japan

トルネックとなり、デバイスメモリ容量内の計算に比べて、性能は数十分の一にまで低下してしまう．この問題の改善のために、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．ステンシル計算においては、テンポラルブロッキング [6], [7], [8], [9] と呼ばれる手法が知られており、主にキャッシュヒット率を上げるために活用されてきた．本研究ではこれをデバイスメモリとホストメモリ間の PCI-Express 通信を削減するために活用する．テンポラルブロッキング手法の適用のためには、ループ構造の変更など、ソフトウェアへの変更が必要となるが、その変更が少なくて済むことが望ましい．

我々はこれまでに、テンポラルブロッキング手法を HHRT ライブラリの実行モデル上で実装することにより、プログラミングコストが抑えられることを、7 点ステンシルなどの単純なカーネルについて示してきた [5]．しかし、実アプリケーションにおいてこれが可能であるかは明らかでなかった．そのため本研究では、対象ソフトウェアである都市気流シミュレーションの詳細な調査により、我々の手法が適用可能であるかの判断が必要であった．具体的には、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を確認した．また、時間ステップループの構造変更を行う際に、境界条件処理のタイミングに変更の必要がないことを確認した．

以上の手法により、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、低プログラミングコストの 3 つを達成する．GPU を搭載したスパコンである TSUBAME2.5 上での性能評価を通して、提案手法の有効性を示す．

2. 背景

2.1 メモリ階層

GPU や Xeon Phi に代表されるメニーコアプロセッサ、SSD などの不揮発性メモリの登場により、メモリ階層が深化が進んでいる．これにより、様々な性質をもつ新しいメモリの活用によるメモリウォール問題の解決が期待されている．

現在のスーパーコンピュータで普及している GPU アクセラレータを搭載したシステムのメモリ階層は図 1 のようになる．一般的な、キャッシュ、メモリ、ハードディスクからなるメモリ階層より、高階層となっていることがわかる．ホストメモリは、容量 52GB(or 96GB) とメモリバンド幅 32GB/s(3 channel) であるのに対し、デバイスメモリは、容量 6GB とメモリバンド幅 250GB/s と、小容量な高速なメモリであることが分かる．しかし、デバイスメモリを効率よく使うためには、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如何に効率よく行うかが重要となっている．

以上のように、メモリ階層を効率よく利用するためには、メモリ階層間の様々な問題に対処する必要があるため、プログラミングコストは高くなることが予想される．

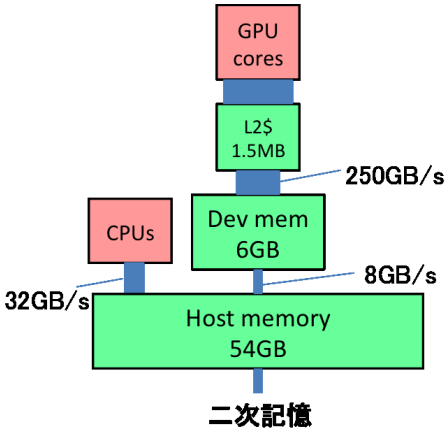


図 1 TSUBAME2.5 のメモリ階層

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．流体を図 2 のような格子で表し、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、ダブルバッファリングを用いることで、各格子点を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．

このステンシル計算の高い並列性から、GPGPU などに代表されるメニーコアプロセッサと非常に相性が良く、ホストメモリより高いデバイスメモリのメモリバンド幅によって、高い性能向上を実現している．

GPU クラスタ上での、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 7 点ステンシル計算を例に、そのプログラミングモデルを考える．図 3(a) が示すように、各時間ステップに値の更新と境界の部分の隣接データを格納する袖領域の通信をする．しかし、現在、GPU のプログラミングモデルでは、CPU-GPU 間のメモリ転送はユーザーが記述する必要があり、メモリ容量を超えるデータを対応するためには、プログラマがデータスワップ処理を記述しなければならない．

GPU のメモリ容量を超えるデータに対応したプログラミングモデルを図 3(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．

© 2014 Information Processing Society of Japan

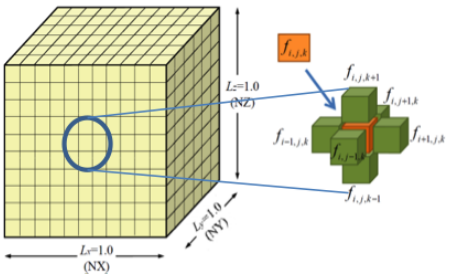


図 2 ステンシル計算

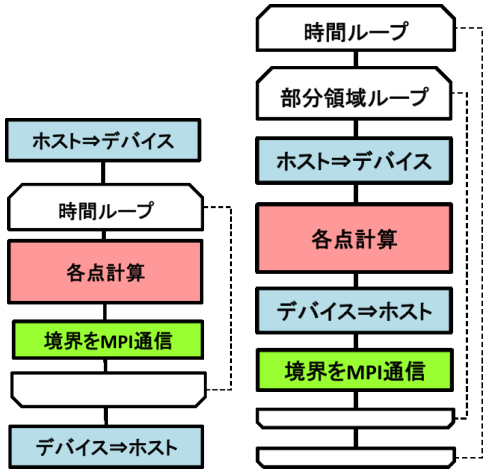


図 3 (a) 通常の MPI+CUDA、(b) 単純な大容量化に対応したプログラミングモデル.

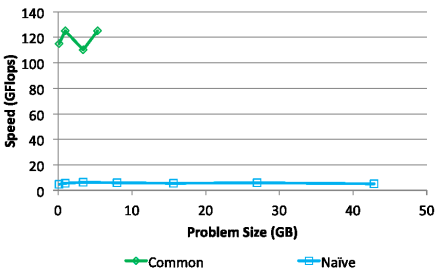


図 4 図 3(a) と (b) の性能. Common=(a)、Naive=(b)

2.3 都市気流シミュレーション

今回に評価に使用する小野寺らが開発した実ステンシルプリケーションである都市気流シミュレーション [2] について説明する.

都市気流シミュレーションは、東京都深部の 10Km 四方のエリアを対象とし、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って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.

都市気流アプリケーションは格子ボルツマン法 (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を離散化された空間格子上で、並進と衝突をする粒子の集合で表され、空間格子上の粒子は速度分布関数を表し、この関数の時間発展方程式を解く計算手法である.

一般的に、LBM による気流計算には、ラージエディ・シミュレーション (LES:Large-eddy simulation) と呼ばれる乱流モデルを導入が必要で、現在、動的スマゴリンスキー・モデルが採用されているが、モデル定数の決定に各格子点で、広領域の平均操作が必要となっており、大規模並列計算や GPU 計算には不向きであった.

都市気流シミュレーションでは、LES にコヒーレント構造スマゴリンスキー・モデルを採用することにより、モデル定数の決定を局所的に行えるようにしたことで、一般的なステンシル計算と同じような計算が行えるようになり、大規模並列計算を可能にしている.

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.

- ステンシル計算は、格子点を 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 個もつ 4 次元配列で表される.
- 都市気流シミュレーションは更に 16 個の配列を必要とする.
- 計算するカーネル数は、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、7 点ステンシルは 1 つであるのに対し、都市気流シミュレーションは 7 つカーネルを計算する.
- 境界条件は、7 点ステンシルは境界部分の値を使って計算するのに対し、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ため、7 点ステンシルプログラムより最適化のプログラミングコストは高くなる傾向にある. 実アプリケーションを採用した理由は、7 点ステンシルプログラムより最適化コストを減らす対象として相応しいと考えたからである.

3. 実ステンシルアプリケーションの大規模性と高性能性の両立

本研究の目的は、ステンシル計算において、高性能で、大容量かつ低プログラミングコストを実現することである. 目的実現のために、局書生向上アルゴリズムとメモリスワップ対応ランタイムを組み合わせることを提案し、その評価を行う.

具体的な手法としては、局所性向上アルゴリズムの一つであるテンポラルブロッキングとメモリスワップ対応ラン

© 2014 Information Processing Society of Japan

タイムである HHRT を採用する．HHRT とテンポラルブロッキングについての詳細は後述するが、プログラマは、MPI と CUDA で記述されたステンシルアプリケーションを GPU の容量を超える大規模化に対応するために、テンポラルブロッキングを実装し、HHRT ライブラリを呼び出すよう記述する．

通常であれば、テンポラルブロッキングの実装に加え、CPU-GPU 間のデータ転送をプログラマが記述する必要があるが、HHRT の導入によりその記述をする必要がなくなり、プログラミングコストを抑えることができる．

3.1 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に、時間ステップ計算を複数回行う時間ブロッキングアルゴリズムである [6], [9]．ステンシル計算を 1 ステップ計算に、隣接格子の値を用いるため、時間ブロッキングを行うことにより、キャッシュヒット率が向上させる．また、時間ブロッキングは、分散メモリ環境におけるステンシル計算の隣接通信の回数 [7], [8] や、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の手法 [12], [13] としても使われている．

本研究では CPU-GPU 間の通信回数を減らすテンポラルブロッキングを実装するため、これについて説明する．CPU-GPU 間の PCI-Express 通信のテンポラルブロッキングのプログラミングモデルを図 5 に示す．図 3(b) と比較すると、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 回の計算するので、外側の時間ループは t/k 回の計算となる．テンポラルブロッキングで最適化してないものと比べると、MPI 通信と PCI-Express 通信を 1/k 回に減っていおり、性能が向上していることが図 6 からわかる．

また、更なる最適化 [12], [13] をすることによって性能が向上に成功しているが、プログラミングコストが非常に高くなってしまうという問題がある．

3.2 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．GPU メモリのスワップ機構は OS などのソフトウェアでサポートされていないため、メモリスワップ処理をプログラマが記述しなければならなず、コードが複雑になり、プログラミングコストが増加してしまう傾向にある．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．

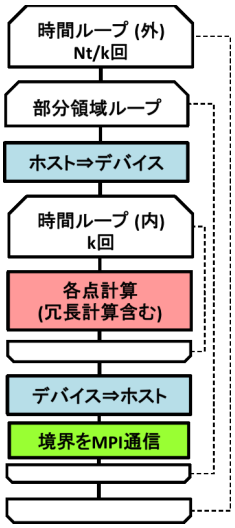


図 5 テンポラルブロッキングのプログラミングモデル

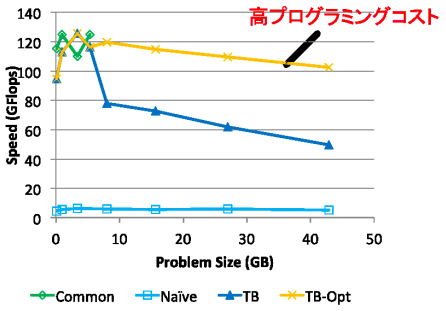


図 6 テンポラルブロッキング化による性能

想定アプリケーションは、複数の GPU アクセラレータを用い、複数計算ノードまたは複数プロセス間の連携に MPI によるメッセージ通信を行い、GPU の制御には CUDA を用いるものとする．このアプリケーションの通常の実行モデルを図 7、HHRT を適用したアプリケーションの実行モデルを図 8 に示す．通常のものと HHRT を適用したものモデルの大きな違いは、HHRT を適応したアプリケーションは、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る．通常の実行モデルでは、1 つの MPI プロセスは GPU のデバイスメモリ容量までのデータサイズしか扱えないが、HHRT を適用した実行モデルでは、複数の MPI プロセスが単一 GPU のデバイスメモリを共有しており、OS の仮想記憶機構のようにデータのスワップをすることによって、擬似的にデバイスメモリ容量を超えるデータを扱うことを可能にする．

プロセスのスワップ処理は、GPU 上で実行中のプロセスが MPI_Send()、MPI_Wait()、MPI_Barrier() などの MPI のブロッキング通信命令と cudaMalloc() などの CUDA のメモリ確保命令を呼んだとき、実行可能であるスリーププ

© 2014 Information Processing Society of Japan

ロセスがいる場合、ブロックしたプロセスをスリープさせ、スワップアウトさせる．その後、実行可能なプロセスを起こし、スワップインさせ、計算を再開させる．

プログラミングコストは、MPI 関数と CUDA 関数と main 関数の関数名を HH(MPI_Send)()、HH(cudaMalloc)()、HH(main)() のように HH() で囲むことと、1MPI プロセスが GPU 上に確保するメモリ容量を HH_initHeap() 関数でライブラリに引数として渡すだけである．

テンポラルブロッキングを実装したステンシルプログラムに HHRT を導入したプログラミングモデルを図 9 に示す．図 5 と比べると、部分領域ループと CPU-GPU 間のデータ転送処理を記述する必要がなくなるため、プログラマはテンポラルブロッキングの実装に注力することができる．

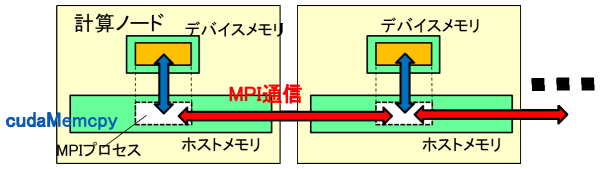


図 7 通常の MPI+CUDA の実行モデル

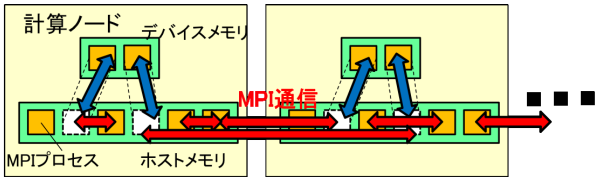


図 8 HHRT の実行モデル

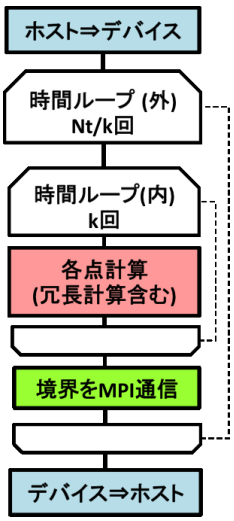


図 9 テンポラルブロッキングと HHRT のプログラミングモデル

3.3 都市気流シミュレーションへの適用

提案手法を都市気流シミュレーションへの適用過程について説明する．適応する手順は、テンポラルブロッキングを実装し、その後、HHRT を導入するためにプログラムを変更した．

まず、テンポラルブロッキングの導入について説明する．テンポラルブロッキングは、局所性を向上させるアルゴリズムであるため、領域全体の値を使う計算がないことが条件となるので、ソースコードを確認する必要がある．都市気流シミュレーションのテンポラルブロッキングに関わる部分である main 関数、MPI 通信、カーネル計算だけで 4300 行程度あるため、プログラムの把握だけで 7 点ステンシルプログラムよりコストがかかることがわかる．コードの確認が終わるとテンポラルブロッキングの実装に入るが、注意したい点がある．基本的には、7 点ステンシルと同じ方針で実装するので、1 カーネル関数自体の変更コストはそれほど大きくないが、冗長領域を増やすことによって変化する値がある可能性があるからである．7 点ステンシルであれば、冗長領域の計算にだけ注目しておけばよかったが、都市気流都市気流シミュレーションのの場合は、境界条件への対応と冗長領域を含むサブドメインの大きさが変数の値に利用されていたので、コード数の多い実ステンシルアプリケーションではステンシルカーネル以外の計算にも注意しなければならない．また、境界条件は、アプリケーションによって、毎ステップ呼び出す必要があるのか、k 回に 1 回呼び出すだけで良いのかにも注意が必要である．

最後に、HHRT の導入について説明する．HHRT は、MPI、CUDA のランタイム API と main 関数に HH() を加えるだけで良いので、少ない変更でメモリスワップ機構を実装できる．しかし、注意したい点は、HHRT のスワップ処理がブロッキング通信命令が呼ばれたときに発生するので、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ので、変更が必要となる．都市気流シミュレーションでは、19 ないし 27 点の値を使った計算を行うので、斜め方向の通信を必要とするために、X、Y、Z 方向の通信をそれぞれ順番に行うことによって実現している．そのため、MPI 通信のために 3 回のスワップ処理を行わなければならないので、大きなオーバーヘッドとなってしまうため性能が低下してしまう．このようなアプリケーションは、ブロッキング命令を 1 回だけ呼ぶようにするために、都市気流シミュレーションの場合では、26 方向の MPI 通信をするように記述する必要がある．

以上の手順で、アプリケーションの大容量化に対応した実装を行うことが出来るので、プログラミングコストを抑えることが出来る．MPI 通信の変更を行う必要がある場合があるものの、特に、メモリスワップ機構とその処理のタ

© 2014 Information Processing Society of Japan

イミングをプログラマが記述する必要がないことがこの提案の最大の利点である．

4. 性能評価

4.1 実行環境

今回の性能評価は、東京工業大学のスパーコンピュータ 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、ホストメモリ容量は 54GB(一部 96GB) となっている．

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 1.6.3、そして CUDA は 5.5 を使用した．

4.2 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．測定対象は、Normal、Normal/HHRT、TB/HHRT、TB-X/HHRT の 4 つとし、各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
- Normal/HHRT:nomal に hhrt のみを適用したプログラム
- TB/HHRT:本研究の提案手法を適用したプログラム
- TB-X/HHRT:TB/HHRT を HHRT の動作に合うように MPI 通信を変更したプログラムのプロトタイプ

測定対象となる各プログラムを、問題サイズを変更して性能測定した図 10 に示す．テンポラルブロッキングを実装したものデータは、最適な性能を使った．

本実験で利用した GPU K20X は、デバイスメモリ容量が 6GB であるため、Normal では、6GB 以上の問題サイズを扱えないことがわかる．

Normal/HHRT は、6GB 以上の問題サイズを実行できるが、性能がかなり低いことがわかる．それに対し、HHRT/TB では、6GB 以上の問題サイズで最大で約 5 倍の性能向上している．

また、HHRT/TB は、アプリケーションが斜めの MPI 通信をするために、6 方向の通信を 3 回の MPI 通信を順番に行っているので、余分なスワップ処理を起こしてしまっている．TB-X/HHRT では、3 回の MPI 通信を 1 回の MPI 通信にまとめる (ただし、プロトタイプであるため実際の実装では、多少性能が低下することが予想される) ことによって、TB-X/HHRT に比べ、更に最大で約 10 倍の性能向上を実現している．

上記の結果から、提案手法は、本研究の目的を達成するための手法の一つであることがわかる．しかし、目標の性能に対して、最大で 4 割程度しか出ていないため、更なる調査が必要である．

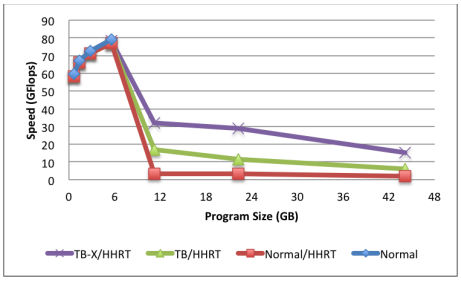


図 10 1GPU での実行結果

4.3 テンポラルブロキングサイズの評価

図 11 は、テンポラルブロッキングサイズ k を変化させたときの TB-X/HHRT(問題サイズ:11.2GB) の性能の変化と計算量の増加率を表したグラフである．テンポラルブロッキングサイズが増加するに伴って、性能が向上していることがわかる．テンポラルブロッキングサイズ 11 と 12 の間で性能が低下しているのは、サイズ 12 以上でデバイスを共有しているプロセスが 1 つになってしまっており、通信と計算のオーバーラップが出来ていないため、性能が低下していると考えられる．また、サイズ 16 から性能が低下してしまっているは、通信回数の削減率より、冗長計算のコストが上回ってしまっていることが理由であると考えられる．

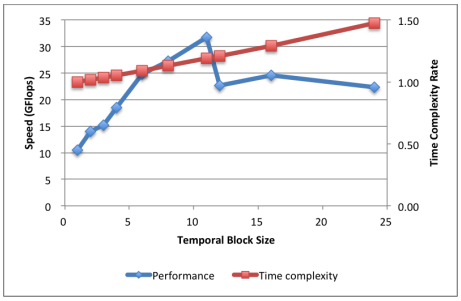


図 11 TB-X/HHRT のテンポラルブロッキングサイズの変化に伴う性能と計算量の増加率

4.4 ウィークスケーリングの評価

図 12 は、1GPU あたりの問題サイズを固定し、GPU の数を変化させていった場合の性能を表している．TB-X/HHRT_S と TB-X/HHRT_L は、それぞれ 1GPU に収まる問題サイズ (5.6GB) と 1GPU に収まらない問題サイズ (11.2GB) の場合の性能を表している．

もとプログラムである Normal と TB-X/HHRT_S を比較すると、HHRT のオーバーヘッドによる若干の性能低下が見られるが、ほぼ同等にスケールしていることがわかる．また、TB-X/HHRT_L については、他と比べ性能は低いが、ウィークスケーラビリティは高いことがわかる．

このことから、HHRT とテンポラルブロッキングの組み

© 2014 Information Processing Society of Japan

合せはウィークスケーラビリティを低下させることなく、大容量化に対応することができることがわかる．

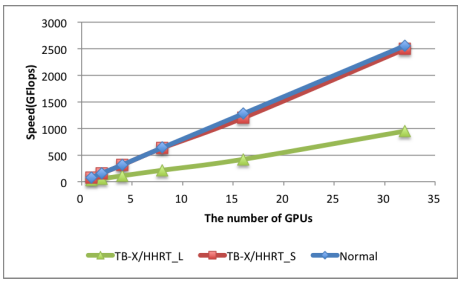


図 12 ウィークスケーリング性能

5. 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．

6. 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、その評価を行った．その結果、HHRT により、少ないプログラムの変更で、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た．さらに、テンポラルブロッキングの導入により、HHRT のみに比べ、最大で約 10 倍の性能向上を実現した．これにより、本提案手法は、目標達成のための手法の一つであることが確かめることができた．

今後の課題として、本研究の目標性能の約 4 割しか出せていないため、性能低下の要因の調査が挙げられる．現在、性能低下の要因として、HHRT のスワップ処理のスケジューリングの最適化と、スワップ処理時のデータ転送と MPI 転送のためのデータ転送の衝突の可能性が考えられるため、詳細な調査を行う必要がある．また、より多くの実アプリケーションにこの手法を適用し、HHRT の更なる改善を進めて行く必要がある．

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 (accepted) (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 Limita-

© 2014 Information Processing Society of Japan

tions. 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).

© 2014 Information Processing Society of Japan