Vol.2012-ARC-202 No.31
Vol.2012-HPC-137 No.31
2012/12/14

GPUメモリ容量を超える問題規模に対応する高性能ステンシル計算法

金光浩[†1†2]　遠藤敏夫[†1†2]　松岡聡[†1†2†3]

GPU 上でのステンシル計算を行う際，その問題サイズは GPU メモリ容量に制限され，その容量は通常ホストメモリより小さい．本論文では GPU メモリ容量を超えた問題サイズへの対応と高性能を両立する最適化手法を提案，評価する．メモリアクセス局所性を向上させるために提案されてきた時間ブロッキング手法に基づき，時間ブロッキングを複数階層について適用し，かつ冗長な計算量を削減する手法を述べる．三次元領域の七点ステンシル計算を評価した結果，単純な方法に比べ 20 倍以上，既存の時間ブロッキング手法に比べ 1.4 倍以上の高速化を実現した．

A Fast Stencil Computation Method for the Domain to Surpass Memory Capacity of GPU

Guanghao Jin[†1†2]　Toshio Endo[†1†2]
Satoshi Matsuoka[†1†2†3]

The problem size of the stencil computation on GPU is limited by the GPU memory capacity, which is typically smaller than that of host memory. This paper proposes and evaluates optimization techniques to achieve both larger problem size than GPU memory and high performance. They are based on the temporal blocking method, which has been proposed to improve memory access locality of stencil computation. We apply temporal blocking to several layers, and then reduce redundant computation. Performance evaluation with 3D 7-point stencil computation, we achieved >20 times performance of naïve implementation and 1.4 times performance of implementation based on existing temporal blocking.

1. はじめに

ステンシル計算は様々な科学分野のシミュレーションにおいて重要なカーネルの一つである[1][2]. ステンシル計算においてはシミュレーション対象領域を規則的グリッドで表現し，時間ステップごとにグリッドの各点を，前の時間ステップの近傍値を基に計算する．その典型的な実装においては，二つのグリッドを容易し，一方のグリッドの値を読み出し，次ステップを表すもう一方のグリッドの値を更新する. 次のステップでは二つのグリッドの役割を交換し，計算を続ける．

高い演算性能と電力効率のために CPU と GPU を組み合わせた高性能計算が近年注目されている[3]. たとえば東京工業大学 TSUBAME2.0 スパコンに搭載されている NVIDIA Tesla M2050 GPU のピーク性能はチップ当たり 500GFLOPS 以上であり, ハイエンド CPU の約 5~10 倍の性能である．

GPU は一般的に計算機の中で補助デバイスとして設置され, そのメモリ容量は CPU のそれより小さい. たとえば, M2050 GPU は 3 GB のメモリを持つのに対し，TSUBAME2.0 の計算ノードの CPU メモリは 54 GB である．通常 GPU 上で演算を行う際には，計算対象データを GPU 側にコピーして計算するので, GPU のメモリ容量制限はドメインサイズを制限する主要因となる．一般的にシミュレーションを高精度に行うためには，大きなグリッドサイズへの対応が必要となる．そのためには，GPU を多数用いることにより，利用可能な合計容量を増加させて対応するのが一般的である．

しかし問題サイズへの対応のためだけに多数 GPU を用いることは，計算資源を無駄にする場合がある．本論文では, ステンシル計算において GPU のメモリ容量を超えるドメインに対応を高性能に行うことを目的とする．単純な手法としては，GPU メモリを超えるデータを CPU メモリに保持しておき, 計算のたびに一部ずつ GPU メモリにコピーする，つまり概念的にはスワップイン・アウトしながら計算を進めることが考えられる．この手法は密行列演算のように局所性の高い計算では成功する[4]が，単純なステンシル計算においては, 時間ステップごとに全領域を CPU メモリと GPU メモリ間で転送する必要が生じ, 性能は大きく低下してしまう．

ステンシル計算の局所性を向上させる手法として時間ブロッキングが提案されてきた[5][6][7][8]. この方法では，ドメインをより小さいサブドメインに分割し，サブドメインの計算を複数の時間ステップについて進めるものである．ドメイン全体の一ステップの計算が全て終わってから次のステップに移る通常の計算方法よりも，局所性の向上が可能である．

†1 東京工業大学
Tokyo Institute of Technology
†2 JST-CREST
†3 国立情報学研究所
National Institute of Informatics

Vol.2012-ARC-202 No.31
Vol.2012-HPC-137 No.31
2012/12/14

本論文では時間ブロッキング手法を, GPU メモリと CPU メモリ間の通信削減のためにも適用する方法を提案する．さらに，GPU 上のキャッシュの効率利用のために，GPU カーネル内でも時間ブロッキング手法を適用する．このように二階層の時間ブロッキングを組み合わせて利用することにより, 大きなドメインの演算を低コストで可能とする．更に，時間ブロッキングの結果発生する冗長な演算を削減し，さらに演算性能を向上できることを示す．

東工大 TSUBAME2.0 スパコンの 1 ノードで 3D 拡散方程式の 7 点ステンシル計算を用いた実験により，提案手法は単純な(時間ブロッキングを用いない)方法に比べ 20 倍以上, 既存の時間ブロッキング手法の 1.4 倍以上の高速化を実現することを示す．

2. ステンシル計算と最適化手法

7 点ステンシル計算においては，各時間ステップ(n+1 番ステップ)において領域の各点を, 以下のような式のように更新する．各点の更新を行うために前の時間ステップ(n 番ステップ)における近傍の値を必要とする．

$$f^{n+1}_{i,j,k} = c_0 f^{n}_{i,j,k} + c_1 f^{n}_{i+1,j,k} + c_2 f^{n}_{i-1,j,k} + c_3 f^{n}_{i,j+1,k} + c_4 f^{n}_{i,j-1,k} + c_5 f^{n}_{i,j,k+1} + c_6 f^{n}_{i,j,k-1}$$

このような計算においてドメインをサブドメインに分割すると，各サブドメインは他のサブドメインに属している隣接点を必要とする．これらの隣接点の値を格納する領域は袖領域と呼ばれる．

ステンシル計算の局所性を向上させるために，1 節に述べたような時間ブロッキング手法が提案されてきた．ドメインをより小さいサブドメインに分割し，サブドメインの計算を複数の時間ステップについて進める．既存研究では CPU キャッシュの効率的利用[5][6]や，MPI 通信回数の削減[7]を目的に利用されてきた．

2.1 GPU カーネルレベルの時間ブロッキング

CPU だけでなく GPU 上においても，時間ブロッキングの考え方に基づき GPU 内部のキャッシュやオンチップメモリを効率利用する既存研究が存在する[8]．この手法は本論文の提案手法に統合されているため(3 節参照), 概要を示す．

時間ステップにまたがって GPU オンチップメモリ(ここでは NVIDIA GPU のシェアードメモリ)の再利用を行うために, 一つの GPU カーネルが複数時間ステップを一度に計算する．図 1 にそのアルゴリズムを示す．この手法により GPU メモリアクセス削減と同時に TLB ミスのコストも削減されると考えられる．ただし必要とするメモリ領域の制限から，まとめるステップ数を大きくすることは困難であり, 本論文では一 GPU カーネル中でまとめるステップ数は 2 とする．

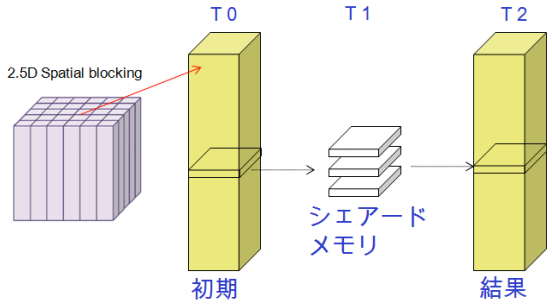

図1: GPUのオンチップメモリを効率利用する時間ブロッキング手法

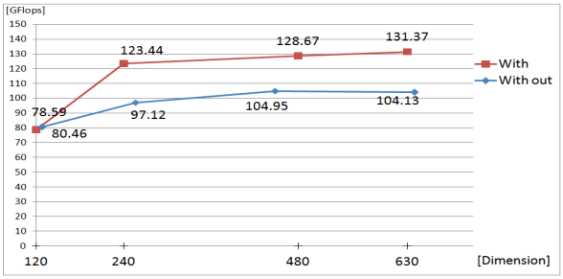

図2: GPUのキャッシュを効率利用する時間ブロッキングの性能

図 2 にその性能を示す．時間ブロッキングを用いることにより，通常の場合と比べ約 20%の性能向上が得られている．

2.2 サブドメインへの分割とその問題

GPU のメモリ容量よりも大きなドメインへの対応として考えられる単純な手法として，以下が考えられる．大きいドメインを複数のサブドメインに分け, 順番に GPU に送って計算する方法である．

三次元拡散方程式を例にとり，ドメイン分割を単純化のために Z 方向の一次元のみとする．図 3 のように，一つの時間ステップ中に，各サブドメインについて以下を行う必要がある：サブドメインと袖領域を GPU に送り，GPU 上でそのサブドメインを計算し, その後結果を CPU へコピーする．この方法を，本論文では通常手法と呼ぶが，各時間ステップにおいて領域全体を PCI-Express 通信するため，多大な性能コストがかかるという問題がある．

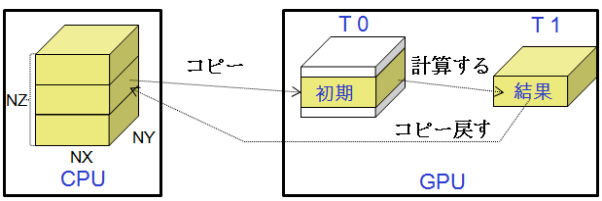

図3: 単純に複数サブドメインに分割した手法

(通常手法)

3. GPU メモリを効率利用する手法の提案

本節では GPU メモリ容量を超えつつ高性能を維持する手法を提案する．基本的なアイデアは，時間ブロッキング手法を CPU-GPU 間通信の削減のためにも用いるということで．以下では，アルゴリズムを順に改良する形で説明する．さらに続く 4 節では冗長演算を削減する手法を述べる．

3.1 マルチドメイン･マルチ時間ステップ法 (MM 法)

通常手法の欠点である大きな CPU-GPU 間通信コストを削減するために，時間ブロッキングの概念を適用する．その手法をマルチドメイン・マルチ時間ステップ法(MM 法)と呼ぶ．この手法では，各サブドメインのデータを GPU メモリに保持したまま，複数時間ステップの計算を行う．動作の概要を図 4 に示す．本図では 2 ステップをまとめているが，より多いステップ(たとえば 64 ステップ)をまとめることも可能である．

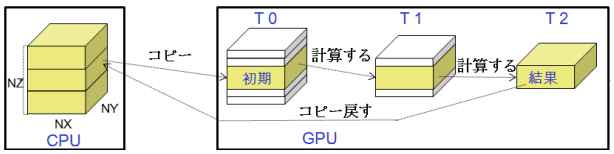

図4: MM法

この手法により複数の時間ステップの計算を GPU メモリローカルで計算でき，CPU と GPU の頻繁な通信を避けられる．しかし，(1) この手法単体では GPU 内のキャッシュの効率利用はされない，(2) 複数ステップ分の袖領域のために冗長な計算や通信が発生する，という問題がある．

3.2 MMT 法

MM 法の問題(1)を解決するために，MMT 法(MM + Temporal blocking kernel)を提案する．MM 法との差異は，2.1 節で述べた GPU 内の時間ブロッキングを統合することにより，GPU メモリアクセスコストを削減し，性能を向上させる点である．

この手法においても，2.2 に述べた問題(2)は依然残るので，以下でその詳細を示す．図 5 では単純化のために，ドメインは Z 方向の一次元とする(一つの XY 平面が一つの丸に相当することになる)．そしてドメインはここでは 3 つのサブドメインに分割され，各サブドメイン(袖領域を含む)は GPU メモリに納まるサイズとする．また図では 4 ステップの計算をまとめた様子を示す．各 Tn の n はタイムステップ n= 1,2,3 を表す．各サブドメインの計算のために CPU から GPU へコピーされる部分は T0 の領域となり，ここには複数の袖領域が含まれる．計算後に CPU にコピーされる部分（結果のコピー）は T4 の列である．MM 法，MMT 法においては，通常の方法よりもコピーの回数は大きく削減される一方，CPU から GPU へのコピー量(T0)，および計算に冗長な部分が含まれている．

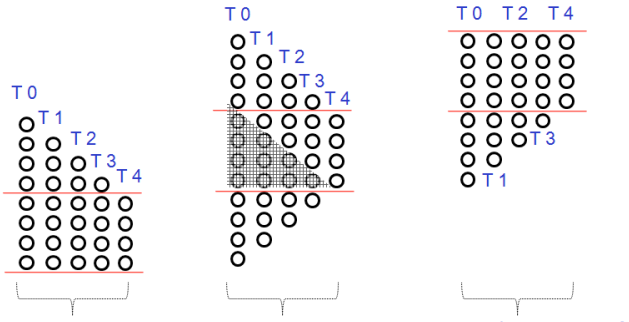

図5: MM法/MMT法のアルゴリズムの流れ

4. 冗長演算と通信を削減する MMTB 法の提案

3 節で述べた MMT 法の問題である，冗長な計算と通信を低減するため，サブドメイン間で情報を再利用する手法である MMTB 法(MMT + Buffer copy)を提案する．

MM/MMT 法の問題は，図 5 のように，第 0 と第 1 計算ステップの一部が重なっており冗長であることである．第 0 計算ステップを計算する時，重なった一部を GPU に保存して次の計算ステップに再利用すると計算時間を短縮できる．

図 6 のように MMTB 法では，次のサブドメインの計算に再利用可能な部分(図の色付きの部分)の計算結果を，2 時間ステップ毎に GPU メモリ上に保存する．2 時間ステップ毎とする理由は，GPU 内の時間ブロッキングにより 2 時間ステップごとに結果のみが GPU メモリ上に格納されているためである．

次の計算ステップでは，2 時間ステップ毎に，先ほど保存した領域を，袖領域に供給しつつ計算する．この方法で計算を続けることにより，最後のタイムステップに正しいサブドメインの結果が得られる．

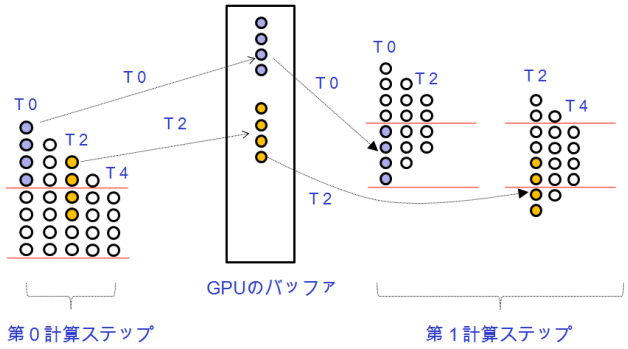

図6: MMTB法のアルゴリズム

MMTB 法の疑似コードを以下に示す：

```
For(i = 0 ; i < TTI ; i += TTS)
  For (j = 0 ; j < TCS; j += 1)
```

Vol.2012-ARC-202 No.31
Vol.2012-HPC-137 No.31
2012/12/14

```
    …
    // If sub-domain is in the middle
    Copy sub-domain j and ghost boundaries
          from CPU to GPU;
    For( k = 0; k < TTS; k += 2)
      Supply 4 XY-planes from Buffer;
      Compute 2 time steps in 1 kernel with
        un-overlapped XY-planes & 4 XY-planes ;
      Store 4 XY-planes to Buffer for
           next computation step;
    End
    Copy sub-domain j from GPU to CPU;
  End
End
```

ここで，TTS は GPU ローカルで計算できる時間ステップ数であり，TCS はサブドメインの数(=図 5 などにおける計算ステップ数)である．TTS ステップの計算を TTI 回繰り返すため，本コードが計算する総時間ステップ数は TTS×TTI となる．

MMT と MMTB の差異を図 7 に示す．MMT の計算はサブドメインと複数の袖領域を含むのに対して，MMTB は非重複部分だけを含む．計算量だけでなく通信量も削減されている．MMT における初期通信量はサブドメインサイズと袖領域のサイズの両方であるのに対して，MMTB は重なっていない部分だけである．

このような計算結果の再利用により冗長演算を減らす手法は既存研究([5]など)と同じ性質を持つ．既存研究ではキャッシュの効率利用を目的とし，データの移動が暗黙的に行われているのに対し，本研究ではデータ移動を明示的に行う必要がある前提で，冗長演算削減が有効であることを示すという違いがある．

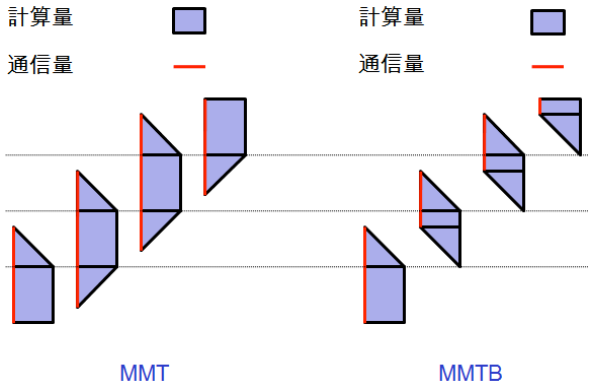

図7: MMT法とMMTB法の計算量・通信量の比較

なお例外的ではあるが，サブドメイン数が 2 のときに，最終サブドメインの結果(図 8 の色つきの T4)を，次のイテレーションの第 0 計算ステップの初期値として再利用可能である．

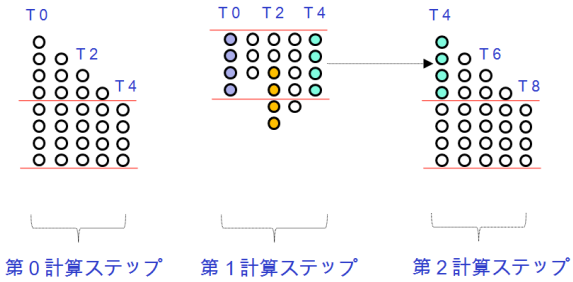

図8: ドメイン数2の場合の最適化

5. 性能評価

東京工業大学学術国際情報センターの TSUBAME2.0 スパコンの 1GPU を用い提案手法を評価した．TSUBAME2.0 のノードは 6 コア Intel Xeon X5670 (2.93GHz)を 2 つと，54GB のメモリを持つ．そして NVIDIA Tesla M2050 GPU を三つ搭載する(本実験では一つのみ利用). M2050 GPU は，14 個の Streaming Multiprocessor と 3GB のメモリを持つ．

評価には，三次元拡散方程式に基づく 7 点ステンシル計算を用いた 5.1 節, 5.2 節ではドメインサイズを 720×720×720 として MMTB と MMT を比較する．その後，ドメインサイズを変化させながら性能評価を行う．最後に性能を制約する原因を分析する．

5.1 ブロッキングする時間ステップ数の性能への影響

MMT 法と MMTB 法において，まとめる時間ステップ数(疑似コードの TTS)を変化させた場合の性能評価を行った．サブドメイン数は 3 とした．図 9 には計算時間と，CPU-GPU 通信時間を示す．

MMT 法においては TTS を変化させた場合にトレードオフがあることが分かる．つまり TTS を大きくすれば通信回数を削減することができる一方，冗長演算のために計算時間が増大していることが分かる．MMTB 法では計算時間の増大は見られず，ほぼ一定である．通信時間についても，MMT 法と比べて通信量の削減ができているために短縮できている．

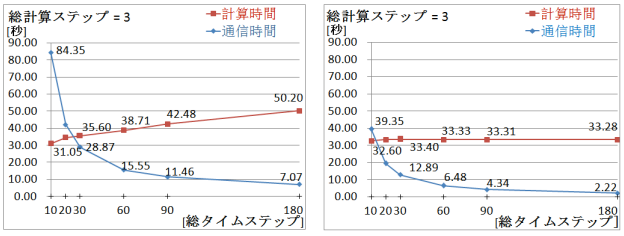

図9: TTSを変化させた場合の計算時間と通信時間．(左) MMT法，(右) MMTB法

5.2 サブドメイン数の性能への影響

次にサブドメイン数を変化させた場合の性能を図 10 に示す．全体ドメインサイズ(=720×720×720)と，TTS(=30)は固定である． MMT においては，サブドメイン数が多い場

合に冗長演算量や通信量が増大し，計算時間は長くなっている．一方，MMTB 法ではほぼ一定に近く，その実行時間は MMT 法より短い．

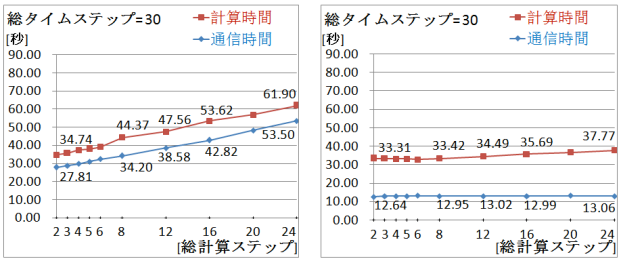

図10: サブドメイン数を変化させた場合の計算時間と通信時間．(左)MMT法，(右)MMTB法

5.3 ドメインサイズを変化させた場合の手法の比較

ここでは，ドメインサイズを変更させた場合の通常手法，MM 法，MMT 法，MMTB 法の性能を比較する．ドメインを直方体とし，一辺のサイズを 240 から 2160 まで変化させたときの速度性能は図 11 のようになる．なお MM，MMT，MMTB における，一度に計算する時間ステップ数(TTS)は，各条件において最も高性能となる値を選んだ．

通常手法においては，一辺が 720 を超えると大幅に性能が下がることが分かる．この理由は領域が GPU メモリサイズを超えたために，時間ステップごとに領域全体を CPU と GPU 間でコピーするコストが大きいことである．領域が小さい場合に比べて，1/20 から 1/30 程度に性能低下している．

MM，MMT，MMTB においてはそのような急激な性能低下は見られない．MM と MMT を比べると，MMT のほうが特にドメインが 1200 以下の場合に高性能となっている．さらに MMTB は，MMT の 1.4 倍以上の性能を実現しており，冗長な演算と通信の削減の効果が表れている．

MMTB はドメインサイズが小さく GPU メモリに納まる場合でも，通常手法とほぼ同等の性能となっている．MMTB はどのドメインサイズについても，四手法の中で最良か最良に近い性能を実現できている．

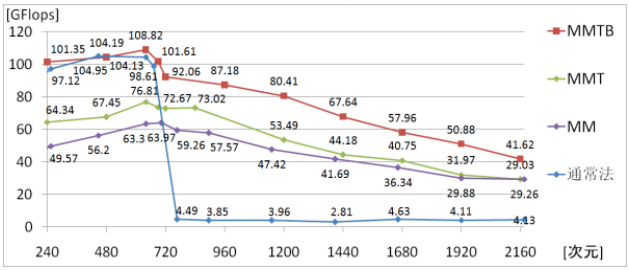

図11: 通常手法，MM法，MMT法，MMTB法の比較

5.4 性能を制限する要因についての議論

MMTB 法により，冗長な通信を削減することが可能であるものの，サブドメインの数が増加すると，バッファコピーの時間も増加してしまうことが分かっている(図 12 左のグラフ)．サブドメイン数は，現在の一次元分割においては，以下のような理由により増加せざるを得ない．

GPU メモリ上には二つのグリッドと袖領域が確保されるため，ドメインの一辺のサイズ D が増加するにしたがって，TTS（ローカルで計算できるタイムステップ）と Ds（サブドメインのサイズ）は減少する．

GPUのメモリ容量≧

$$2\times D\times D\times(Ds+TTS)+D\times D\times TTS\times 4/2$$

図 12 右のグラフのように，サイズ D が大きくなると利用可能な TTS が減り，通信回数が増えて通信時間が増えることとなる．Ds が減るとドメイン数は増えてしまうので，バッファコピーの時間も増える．これによりドメインサイズの増大に従い性能が落ちてしまう．この問題を軽減するために，ドメインを多次元分割することを検討している．

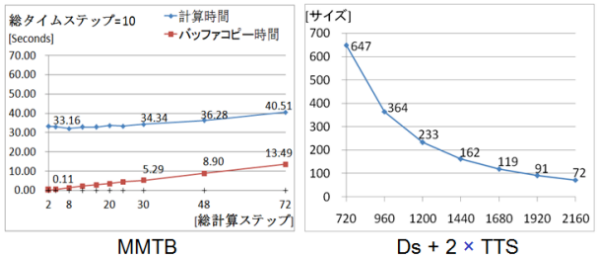

図12: ドメインサイズと性能制約の関係

6. まとめと今後の課題

本論文では，GPU 上のステンシル計算の効率化のために，GPU メモリと GPU キャッシュの双方を効率利用する，二階層の時間ブロッキング手法を組み合わせ，さらに冗長演算・コピーを削減する最適化手法について述べた．これらの統合により，GPU メモリ容量を超える大きなドメインにおいても高速に計算できることを示した．

現在の実装では，分割方向が一次元であることに起因する性能低下が考えられるため，その点の改良を計画している．また今後の課題に含まれる項目は以下の通りである：CPU-GPU 間通信と計算のオーバーラップ，マルチ GPU・マルチノードへの対応，およびその上での最適化手法の提案などである．また，単純な 7 点ステンシルだけではなく，実用的なアプリケーションに提案手法を適用していきたい．

謝辞 本研究の一部は JST-CREST「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」の支援による．

参考文献

[1] L.Renganarayanan, M.Harthikote-Matha, R.Dewri, and S.V.Rajopadhye. Towards optimal multi-level tiling for stencil computations. IEEE International Parallel & Distributed Processing Symposium (IPDPS 2007), pp.1–10, 2007.

[2] F.Shimojo, R.K.Kalia, A.Nakano, and P. Vashishta.

Divide-andconquer density functional theory on hierarchical real-space grids:parallel implementation and applications. Physical Review, vol.B, no. 77, pp.1–12, 2008.

[3] Naoya Maruyama, Tatsuo Nomura, Kento Sato and Satoshi Matsuoka. Physis: An Implicitly-Parallel Programming Model for Stencil Computing on Large-Scale GPU-Accelerated Supercomputers. IEEE SC11, 2011.

[4] Toshio Endo and Satoshi Matsuoka. Massive Supercomputing Coping with Heterogeneity of Modern Accelerators . In Proceedings of IEEE International Parallel & Distributed Processing Symposium (IPDPS 2008), 10pages, April 2008.

[5] 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, april 2010.

[6] 南 武志，岩下 武史，高橋 康人，中島 浩．キャッシュメモリを考慮した FDTD カーネルの性能改善．情報処理学会研究報告， Vol.2010-HPC-124 No.5, 7pages, 2010.

[7] 河村 知輝，丸山 直也，松岡 聡．並列ステンシル計算における通信の自動最適化に向けた性能モデルの評価.情報処理学会研究報告，Vol.2012-HPC-135, 8pages, 2012

[8] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim and Pradeep Dubey. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. IEEE SC10, 2010.