Vol.2017-HPC-160 No.35
2017/7/28

タイルレベルの並列処理を可能とする時空間タイリング手法を用いた3次元FDTDカーネルの実装と性能評価

深谷 猛[1,a)]　岩下 武史[1]

概要：高周波電磁場解析の代表的な数値計算手法である3次元FDTD法は，反復型ステンシル計算であり，一般的に，その性能は計算機のメモリバンド幅に律速される．反復型ステンシル計算に対しては，メモリアクセスコストを軽減し，性能向上を図る手法として，時空間タイリングと呼ばれる手法が知られている．著者らはこれまでに，3次元FDTD法に対して冗長計算を伴わない時空間タイリング手法を提案したが，スレッド並列化の観点からは，タイルレベルの処理の並列性に関して，改善の余地が大きく残っていた．そこで，本研究では，タイルレベルの並列処理において，より有望と期待される3種類の時空間タイリング手法を3次元FDTD法に実装し，その性能を評価する．性能評価の結果，今回新たに採用した時空間タイリング手法を用いることで，先行研究で提案した手法よりも，3次元FDTD法の性能が向上することが確認できた．加えて，問題サイズにより，適切な時空間タイリング手法が異なることも明らかになった．

1.　はじめに

高周波電磁場解析のための主要な数値計算手法として，3次元FDTD法[1]が広く知られている．3次元FDTD法は，アンテナ解析などをはじめ，工学的に重要な応用が多数あり，その高速化が強く求められている．特に，実際の設計の現場等では，手元に置ける規模の共有メモリ型計算機を使用することも多く，スレッド並列環境での高速化は重要な課題である．

3次元FDTD法は反復型ステンシル計算と呼ばれる計算に分類される．一般的に，反復型ステンシル計算を素朴に実装した場合，メインメモリへのアクセスコストが計算時間において支配的となり，結果として，プログラムの性能は計算機のメモリバンド幅に律速される．これは，近年の計算機において，メモリアクセス性能が演算性能に対して相対的に低下していることを踏まえると，大きな問題である．

反復型ステンシル計算におけるメモリアクセスコストを軽減するための手法として，時空間タイリングと呼ばれる手法[2]が知られている．時空間タイリングを用いた反復型ステンシル計算では，計算の依存関係を保った上で，ある範囲内のみを一定時間ステップ分計算する．それにより，キャッシュメモリの利用効率を向上させ，メインメモリへのアクセスコストを軽減する．

これまでに著者らは，3次元FDTD法に対して，冗長計算を伴わない時空間タイリング手法を提案し，その有効性を確認している[3]．しかし，文献[3]で提案した時空間タイリングの手法は，タイルレベルでの処理の並列性に関する制約が強く，スレッド並列化の観点からは改善の余地が大きく残っていた．近年，CPUのマルチコア・メニーコア化が進んでいることを考慮すると，多数のコアの活用を念頭においた，スレッド並列化部分の改良は重要な課題である．

このような状況を踏まえて，本研究では，タイルレベルでの処理のスレッド並列化において有望と認識されている，ダイヤモンド型の時空間タイリングを採用した手法に着目する．具体的には，3次元FDTD法に対して，ダイヤモンド型に基づく3種類の時空間タイリング手法を検討し，それぞれプログラムを実装する．そして，素朴な実装，先行研究[3]で提案した手法とともに，性能評価を行い，本研究で新たに採用した時空間タイリング手法の有効性を検証する．

以下，まず，第2節では3次元FDTD法のカーネルの概要を述べる．次に，第3節で，3次元FDTD法に対する時空間タイリング手法について，タイルレベルの並列性を含めて検討する．第4節では，実装の概要を述べ，第5節で性能評価の結果を示す．その後，第6節で関連研究を紹介し，第7節でまとめを述べる．

[1] 北海道大学 情報基盤センター
[a)] fukaya@iic.hokudai.ac.jp

ⓒ 2017 Information Processing Society of Japan

2. 3次元FDTD法の概要

3次元FDTD（Finite Difference Time Domain）法 [1] は高周波電磁場解析における主要な数値計算手法の一つである．本節では，3次元FDTD法の素朴な実装を中心に，その概要を述べる．

3次元FDTD法では，電磁場の振る舞いをMaxwell方程式を基に，

$$\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t}, \quad (1)$$

$$\nabla \times \boldsymbol{H} = \epsilon \frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E}, \quad (2)$$

により記述する．ここで，$\boldsymbol{E}$ は電場，$\boldsymbol{H}$ は磁場である．また，ϵ，μ，σ はそれぞれ誘電率，透磁率，導電率である．

3次元FDTD法では，Yeeのメッシュを用いて空間方向の離散化を行い，Leap-flogアルゴリズムを用いて時間方向の離散化を行う．数値計算スキームの具体的な形やその導出過程については紙面の都合により割愛するが（詳細は文献 [4] 等を参照のこと），数値計算スキームは陽解法であり，離散化された電場 $\boldsymbol{E}$ と磁場 $\boldsymbol{H}$ の値を交互に計算することで時間発展計算を進める．

3次元FDTD法を素朴に実装した場合の計算の主要部を**図1**に示す．なお，図1は著者らの先行研究 [3] で示した実装と本質的に同じ構造である．図1中の配列 `Id` は各格子点の媒質の種類を格納しており，媒質の種類に応じて電場・磁場の更新式中のスカラ係数部分の値が異なるものとなる．

図1から分かるように，多少複雑な構造をしているが，3次元FDTD法は反復型のステンシル計算に分類される計算である．3次元FDTD法の素朴な実装では，解析対象の領域が十分に大きく，電場や磁場に関する配列のサイズがキャッシュメモリのサイズを超える場合，各時間ステップ（の電場および磁場）の計算時に，上記の配列のデータをメインメモリから参照することになる．一方，最内ループ内の処理におけるbyte/flop値は，電場と磁場に関する配列だけを考えたとしても，電場の更新部分が96byte/21flop，磁場の更新部分が96byte/18flopとなり，近年の計算機システムのB/F値（メモリ性能/演算性能）よりもはるかに大きい．そのため，一般的に，図1に示した3次元FDTD法の素朴な実装の性能はメモリバンド幅に律速される．

3. 3次元FDTD法に対する時空間タイリング

3.1 時空間タイリングの基本的な考え方

反復型ステンシル計算に対して，メモリアクセスコストを軽減し，性能向上を図るための手法として，時空間タイリングと呼ばれる手法 [2] が知られている．時空間タイリングは，計算対象の領域を小領域（タイル）に分割し，タイル内の要素についてのみ，一定の時間ステップ分の計算を行う手法である．このとき，タイル内の要素がキャッシュメモリに収まるようにすることで，タイル内の計算部分でメインメモリへのアクセスコストが軽減され，結果として，計算全体の性能向上が期待される．

反復型ステンシル計算では，ある格子点の値を更新する際に隣接格子点の値を使用するため，この依存関係を考慮する必要がある．時空間タイリングの手法は，主に，冗長計算を伴う手法と伴わない手法の2つに分けられるが，今回は，後者を考える．冗長計算を伴わない時空間タイリング手法の基本となるのは，平行四辺形型およびダイヤモンド型と呼ばれる2種類のタイリング手法である．1次元3点ステンシル（`u[t+1][i]` の値を `u[t][i-1]`，`u[t][i]`，`u[t][i+1]` から更新）の場合を例にして，それぞれのタイリング手法の様子を**図2**(a) および (b) に示す．平行四辺形型の場合は左側のタイルから順番に，ダイヤモンド型の場合は山型のタイル，谷型のタイルの順番に，タイル内のみを一定時間ステップ分計算することが可能となっている．

空間が2次元以上となる場合も，基本的に，上述の空間1次元の場合の手法の組み合わせにより，時空間タイリングを行うことが可能である．具体的に，2次元5点ステンシルに対して，ダイヤモンド型と平行四辺形型を組み合わせた例を，**図3**(a), (b) および (c) に示す．

上記の3つの例の中で，ダイヤモンド型とダイヤモンド型を組み合わせた場合（図3(c)）の計算手順を具体的に挙げると，まずは，図3(c) に示した，ピラミッド型（山型 × 山型）のタイルを全て計算する．次に，**図4**(a) に示した形状（谷型 × 山型）のタイルを全て処理する．その後，図4(b)（山型 × 谷型），図4(c)（谷型 × 谷型）と順番に処理を進めることで，一定時間ステップ分，全空間の格子点の計算が完了する．このように，空間の各軸で選択するタイリング手法に応じて，必要となるタイルの形状の種類が変化し（上記の例では4種類），形状に関する依存関係を満たす [*1] 順序で，各形状のタイルを処理することになる．

空間が3次元以上の場合についても，同様のアプローチにより，時空間タイリングが可能である．例えば，空間3次元の場合において，ダイヤモンド型 × 平行四辺形型 × 平行四辺形型という組み合わせを選択すると，タイルの形状は2種類 [*2] となる．

平行四辺形型とダイヤモンド型のタイリング手法では，タイルレベルの処理の並列性において大きな違いがある．平行四辺形型の場合，軸方向のタイル処理について依存関係が発生するが，ダイヤモンド型では同じ形状のタイル（例えば，山型のタイル同士）を並列に処理することができる．上述の2次元5点ステンシルの例では，図3(a) に示した組み合わせだと，並列化のためには超平面法が必須と

*1 谷型を山型よりも先に処理することはできない．
*2 （山型 × 平行四辺形型 × 平行四辺形型），（谷型 × 平行四辺形型 × 平行四辺形型）の順番．

ⓒ 2017 Information Processing Society of Japan

```
for(t = 0; t < steps; t++){
  for(x = x_min; x <= x_max; x++){
    for(y = y_min; y <= y_max; y++){
      for(z = z_min; z <= z_max; z++){
       m = Id[x][y][z];
       Ex[x][y][z] = Ce[m]*Ex[x][y][z] + Cery[m]*(Hz[x][y][z]-Hz[x][y-1][z]) + Cerz[m]*(Hy[x][y][z]-Hy[x][y][z-1]);
       Ey[x][y][z] = Ce[m]*Ey[x][y][z] + Cerz[m]*(Hx[x][y][z]-Hx[x][y][z-1]) + Cerx[m]*(Hz[x][y][z]-Hy[x-1][y][z]);
       Ez[x][y][z] = Ce[m]*Ez[x][y][z] + Cerx[m]*(Hy[x][y][z]-Hy[x-1][y][z]) + Cerx[m]*(Hx[x][y][z]-Hx[x][y-1][z]);
  }}}
  for(x = x_min; x <= x_max; x++){
    for(y = y_min; y <= y_max; y++){
      for(z = z_min; z <= z_max; z++){
       m = Id[x][y][z];
       Hx[x][y][z] = Hx[x][y][z] + Chry[m]*(Ez[x][y+1][z]-Ez[x][y][z]) + Chrz[m]*(Ey[x][y][z+1]-Ey[x][y][z]);
       Hy[x][y][z] = Hy[x][y][z] + Chrz[m]*(Ex[x][y][z+1]-Ex[x][y][z]) + Chrx[m]*(Ez[x+1][y][z]-Ez[x][y][z]);
       Hz[x][y][z] = Hz[x][y][z] + Chrx[m]*(Ey[x+1][y][z]-Ey[x][y][z]) + Chry[m]*(Ex[x][y+1][z]-Ex[x][y][z]);
  }}}
}
```

図 1　3 次元 FDTD 法の素朴な実装の主要部.

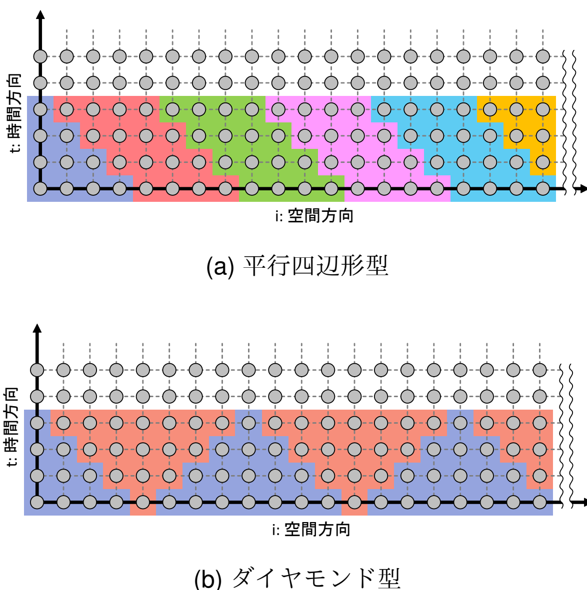


図 2　1 次元 3 点ステンシルに対する，冗長計算を必要としない時空間タイリング手法の例.

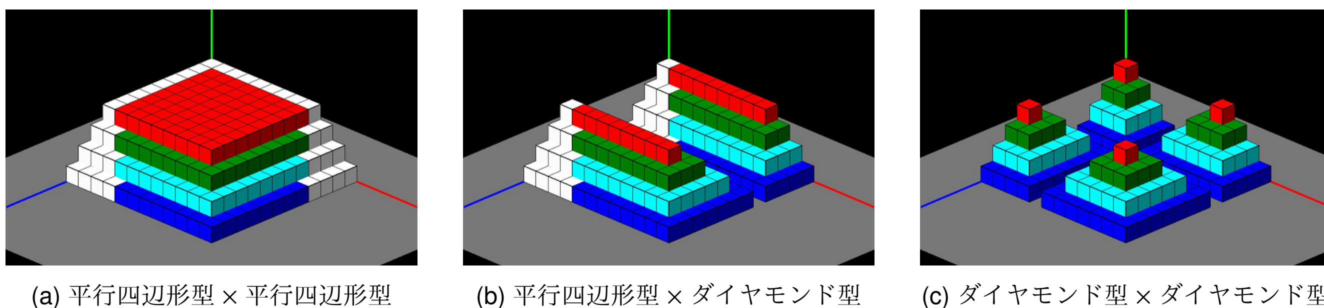
なる．一方，図 3(b) に示した組み合わせの場合，ダイヤモンド型を採用した軸に関して，同じ形状のタイルの間に自明な並列性が存在する．具体的には，図中の（白色ではなく）カラーになっている 2 つの領域について，並列処理が可能である．同様に，図 3(c) に示した組み合わせの場合，2 次元空間全域において，同じ形状のタイル同士は並列処理が可能（例えば，図中の 4 つのピラミッド形状の領域は並列処理可能）となり，並列性が図 3(b) よりもはるかに大きくなる．

3.2　3 次元 FDTD 法の時空間タイリングとタイルレベルの処理の並列性

3 次元 FDTD 法に対する時空間タイリング手法を議論するために，まずは，計算における格子点間の依存関係を整理する．電場と磁場の更新時における格子点（インデックス）間の依存関係は，ベクトルの成分をまとめて考えると，図 **5**(a) および (b) のようになる．これらの図が示すように，電場と磁場のどちらの場合でも，依存関係は各軸に関して同じ構造となっている．したがって，空間 1 次元の場合についてタイルの形状を考え，その組み合わせを考えることで，3 次元空間に対する時空間タイリングを議論できる．

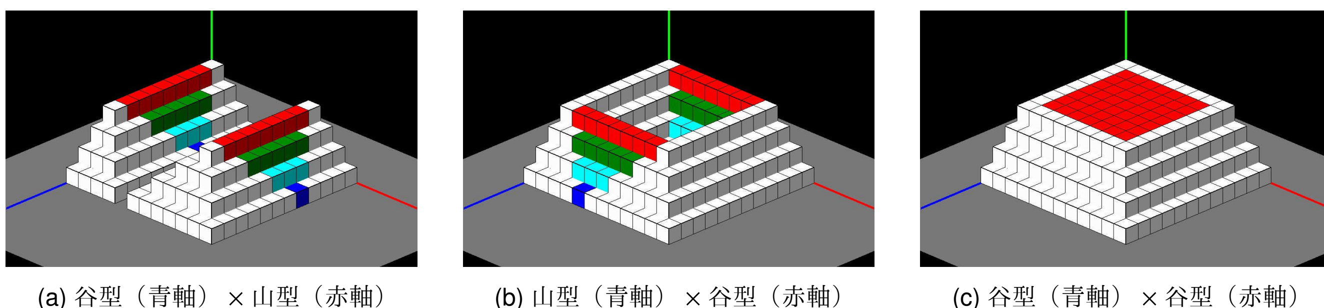
以下，x 軸を例として，空間 1 次元に対するタイルの形状を考える．図 5(a) および (b) から明らかなように，x 軸方向については，電場の場合は対象の点 x とその 1 つ前の点 x-1，磁場の場合は点 x と点 x+1 の間に依存関係がある．この点を踏まえると，今回の FDTD 法に対する 1 次元のタイリング手法は，天下り的ではあるが，平行四辺形型が図 **6**(a)，ダイヤモンド型（を拡張したもの）が図 6(b) に示した形となる．ここで，BLT, BLX はタイルサイズを指定するパラメータである．なお，ダイヤモンド型は，山の頂点（谷の底）に一定の長さを許容する形で拡張している．また，BLT は電場と磁場の更新を併せて 1 とする．

先に説明したように，x,y,z 軸方向のそれぞれについて，平行四辺形型もしくはダイヤモンド型のタイリングを独立して選択し，その組み合わせにより，3 次元 FDTD 法に対して時空間タイリングを行うことができる．以下では簡単のため，時空間タイリング手法を ***x*y*z** という形で表記し，*の部分には，その後ろの軸において，平行四辺形型



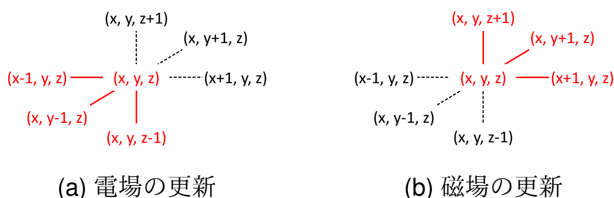
(a) 平行四辺形型 × 平行四辺形型　(b) 平行四辺形型 × ダイヤモンド型　(c) ダイヤモンド型 × ダイヤモンド型

図 3　2 次元 5 点ステンシルに対する，冗長計算を必要としない時空間タイリング手法の例：青色と赤色の軸が空間方向，緑色の軸が時間方向．



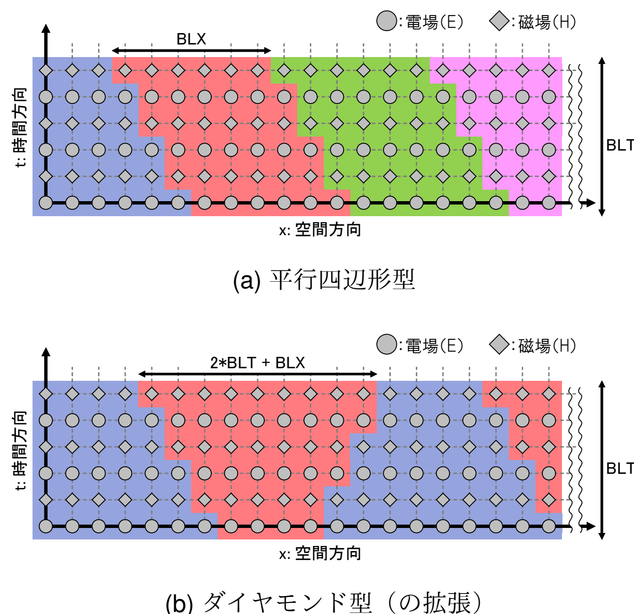
(a) 谷型（青軸） × 山型（赤軸）　(b) 山型（青軸） × 谷型（赤軸）　(c) 谷型（青軸） × 谷型（赤軸）

図 4　2 次元 5 点ステンシルに対する，ダイヤモンド型 × ダイヤモンド型の時空間タイリングの場合に必要となるタイルの形状：青色と赤色の軸が空間方向，緑色の軸が時間方向．図 3(c) で示したもの（山型（青軸） × 山型（赤軸））を除く．



(a) 電場の更新　(b) 磁場の更新

図 5　3 次元 FDTD 法の電場・磁場の更新時における格子点の依存関係：（赤色の）実線部分に依存関係がある．



(a) 平行四辺形型

(b) ダイヤモンド型（の拡張）

図 6　3 次元 FDTD 法に対する，1 次元方向の時空間タイリングの形状．

を採用した場合は **p**，ダイヤモンド型を採用した場合は **d** と記すことにする．例えば，dxpypz は，x 軸方向はダイヤモンド型，y 軸と z 軸方向は平行四辺形型のタイルを採用した手法を意味する．今回の性能評価では，著者らが先行研究 [3] で提案した手法（pxpypz 型）と，ダイヤモンド型を採用した 3 種類の手法（dxpypz 型，dxdypz 型，dxdydz 型）の合計 4 種類の時空間タイリング手法を実装し，その性能を比較することにする．

4.　実装の概要

本節では，時空間タイリング手法を施した 3 次元 FDTD カーネルの実装の概要を述べる．なお，実装は C 言語で行い，OpenMP を用いてスレッド並列化を施した．

まず，素朴な実装を含めた全ての実装において共通する事項を以下の挙げる．

- 電場と磁場に関する配列は，それぞれの各成分ごとに 1 次元配列を動的に確保する．
- 境界を含めた格子点の総数を $NX \times NY \times NZ$ とすると，格子点 (`(x,y,z)` に対して，インデックスを `x*NY + y*NZ + z` と対応させる．（`x`，`y`，`z` は 0 から開始．）
- 配列の初期化の処理もスレッド並列化するが，実際の計算部分におけるスレッドの割り当ては特に考慮しない（first touch は特に行わない）．

なお，素朴な実装では，x に関するループに対して，omp for collapse(2) を指定してスレッド並列化を施す．また，全ての実装において,OpenMP のパラレルリージョンは最も外側の t に関するループで指示する．

次に，時空間タイリングを施した実装の概要を説明する．今回は，**図 7** に示した共通の構造をベースにプログラムを実装した．以下，図 7 に関する補足事項を列挙する．なお，空間方向で共通する事項については，x 軸を例として記載する．

- x_tile_type：タイリング方法（平行四辺形 / ダイヤモンド）
- x_min，x_max：計算対象範囲
- GetNumOfTiles()：タイル数を計算する関数
- k_max：タイルの形状の種類数（TileX[] の要素数）
- TileX[]：タイルの形状に関する配列（後述）
- SetRange()：タイル内のインデックスの範囲を計算する関数（E：電場，M：磁場）

先に述べたように，タイリング手法によって，必要となるタイルの形状の種類が変化するが，この違いについては，配列 TileX，TileY，TileZ を変えることで対応する．具体的には，以下のような設定となる．なお，p は平行四辺形型，dm はダイヤモンド型（山型），dv はダイヤモンド型（谷型）を意味する．

- pxpypz 型：
  - TileX = {p}
  - TileY = {p}
  - TileZ = {p}
- dxpypz 型：
  - TileX = {dm，dv}
  - TileY = {p ，p }
  - TileZ = {p ，p }
- dxdypz 型：
  - TileX = {dm，dv，dm，dv}
  - TileY = {dm，dm，dv，dv}
  - TileZ = {p ，p ，p ，p }
- dxdydz 型：
  - TileX = {dm，dv，dm，dm，dv，dv，dm，dv}
  - TileY = {dm，dm，dv，dm，dv，dm，dv，dv}
  - TileZ = {dm，dm，dm，dv，dm，dv，dv，dv}

選択したタイリング手法に応じて，タイル内の各時間ステップの計算範囲が異なる．そのため，図 7 に示した実装では，計算範囲を設定する関数 SetRange を時間ステップごとに呼び出し，タイルの形状 TileX[k], 場の種類（電場/磁場），タイル形状に関するパラメータ BLT，BLX，タイル内での現在の時間ステップ t，タイルの位置の情報 xx から計算範囲（x_head，x_tail）を計算している [*3]．

[*3] x_min と x_max は境界に関する例外処理のための引数．

また，前節で述べたように，タイリング手法ごとにタイルレベルの並列性が異なる．それを踏まえて，それぞれの実装に関して，以下のように OpenMP の並列化指示文を挿入した．

- pxpypz 型：変数 x に関する 2 つのループそれぞれに対して omp for collapse(2)
- dxpypz 型：変数 xx に関するループに対して omp for
- dxdypz 型：変数 xx に関するループに対して omp for collapse(2)
- dxdydz 型：変数 xx に関するループに対して omp for collapse(3)

5. 性能評価

本節では，3 次元 FDTD 法に関して，素朴な実装，および前節で述べた時空間タイリングを施した 4 種類の実装の性能を評価した結果を報告する．

5.1 評価環境・評価方法等

今回は，京都大学学術情報メディアセンターのスーパーコンピュータ（システム B：Laurel 2）の 1 ノードを使用して性能評価を行った．システムの構成等は**表 1** に記載した通りである．コンパイラは icc（ver. 17.0.2）を使用し，コンパイル時のオプションは，-mcmodel=medium, -shared-intel, -qopenmp -O3, -ipo, -xHost を指定した．また，プログラムを実行する際は，同じノードで他のジョブが実行されないようにしている．

テスト問題としては，先行研究 [3] と同じ問題で，金属壁に囲まれた立方体形状の解析を想定した設定を行った．計算対象の格子点は，3 つの空間軸方向それぞれについて，N 個（全体で N^3 個）とし，境界（金属壁）上の格子点を含めて，$(N+2)^3$ の大きさで配列を確保した．

時間ステップ数を 512 として，計算時間を測定した．ただし，初期化等の時間は除いており，電場と磁場の更新に要する時間のみを測定した．また，性能値（FLOPS 値）は，$39 \times N^3 \times 512$ / (計算時間) で算出している．

5.2 素朴な実装の性能

時空間タイリングの効果を評価する上で基準となる，素朴な実装（naive 実装）の性能を報告する。まず，スレッド数を 36 に固定し，計算格子点数を変えて naive 実装の性能を評価した結果を**図 8** に示す．なお，スレッドのコアへの割り当て方については，

- KMP_AFFINITY=granularity=fine,compact,1,0 を指定（compact と表記）：片方の CPU のコアから順番にスレッドを割り当てる．
- KMP_AFFINITY=granularity=fine,scatter を指定（scatter と表記）：両方の CPU のコアに出来るだけ均等にスレッドを割り当てる．

```
x_ntiles = GetNumOfTiles(x_tile_type, BLT, BLX, x_min, x_max);//各軸方向のタイル数を計算
y_ntiles = GetNumOfTiles(y_tile_type, BLT, BLY, y_min, y_max);
z_ntiles = GetNumOfTiles(z_tile_type, BLT, BLZ, z_min, z_max);
#pragma omp parallel shared(...), private(...)
for(tt = 0; tt < steps; tt += BLT){
  for(k = 0; k < k_max; k++){//タイルの形状に関するループ
    for(xx = 0; xx < x_ntiles; xx++){
      for(yy = 0; yy < y_ntiles; yy++){
        for(zz = 0; zz < z_ntiles; zz++){
          for(t = 0; t < BLT; t++){//タイル内の時間ステップのループ
            SetRange(&x_head, &x_tail, TileX[k], "E", BLT, BLX, t, xx, x_min, x_max);//タイル内の要素の範囲を計算
            SetRange(&y_head, &y_tail, TileY[k], "E", BLT, BLY, t, yy, y_min, y_max);
            SetRange(&z_head, &z_tail, TileZ[k], "E", BLT, BLZ, t, zz, z_min, z_max);
            for(x = x_head; x <= x_tail; x++){
              for(y = y_head; y <= y_tail; y++){
                for(z = z_head; z <= z_tail; z++){
                  //Ex, Ey, Ez の更新（素朴な実装と同じ）
                  m = Id[x][y][z]; Ex[x][y][z] = ...; Ey[x][y][z] = ...; Ez[x][y][z] = ...;
            }}}
            SetRange(&x_head, &x_tail, TileX[k], "M", BLT, BLX, t, xx, x_min, x_max);//タイル内の要素の範囲を計算
            SetRange(&y_head, &y_tail, TileY[k], "M", BLT, BLY, t, yy, y_min, y_max);
            SetRange(&z_head, &z_tail, TileZ[k], "M", BLT, BLZ, t, zz, z_min, z_max);
            for(x = x_head; x <= x_tail; x++){
              for(y = y_head; y <= y_tail; y++){
                for(z = z_head; z <= z_tail; z++){
                  //Hx, Hy, Hz の更新（素朴な実装と同じ）
                  m = Id[x][y][z]; Hx[x][y][z] = ...; Hy[x][y][z] = ...; Hz[x][y][z] = ...;
            }}}
}}}}}
```

図 7　時空間タイリングを施した 3 次元 FDTD カーネルの実装の全体像

表 1　京大スパコンシステム B（Laurel 2）のノード構成

項目		諸元
CPU	プロセッサ	Intel Xeon E5-2695 v4
	アーキテクチャ	Broadwell
	動作周波数	2.10GHz
	コア数	18
	L1 cache (data)	32KB / core
	L2 cache	256KB / core
	L3 cache	45MB / CPU
	理論演算性能	605 GFLOPS
	Hyper threading	有効
ノード	CPU 数	2
	メモリ	128GB
	理論演算性能	1.21 TFLOPS

の 2 種類を試した．

図 8 から，affinity の設定に関わらず，$N = 120$ を境に性能が大きく低下していることが確認できる．$N = 120$ の場合に必要となるメモリサイズが 84MB 程度で，今回使用した CPU2 台分の L3 キャッシュのサイズ（45MB×2）と同程度である．したがって，この点を境に性能が低下した主要因は，データが L3 キャッシュに収まらなくなったことであると推測できる．また，N が小さい場合において，affinity を compact にした方が scatter にするよりも性能が良かったが，詳細については調査中である．

次に $N = 200$ と $N = 300$ の場合について，スレッド数を変化させて性能を測定した結果を**図 9** に示す．図 9 が示すように，36 スレッド時の性能は大差がないが，それまでの性能の挙動が affinity の設定によって異なっている．compact の場合，18 スレッドまでは 1 つの CPU のみを利用するので，一時的に性能が飽和していると推測される．一方，scatter の場合，2 つの CPU を均等に利用するので，最終的な性能限界に達するまでは順調に性能が向上しているようである．

得られた naive 実装の性能を検証するために，stream ベンチマーク [5] の Triad により実効メモリバンド幅を測定し，その結果と FDTD の naive 実装の byte/flop 値から，予想される性能を算出し，実際に測定された結果と比較した．この結果を**表 2** と**表 3** に記載する．なお，FDTD カーネルの byte/flop 値は 192byte/39flop とした．表 2 や表 3 を見る限

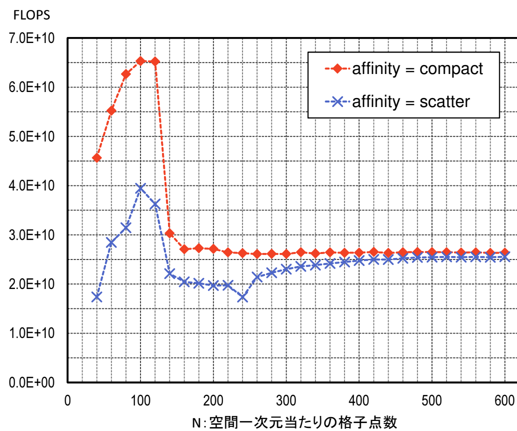


図 8　FDTD naive 実装の性能と問題サイズの関係（36 スレッド）.

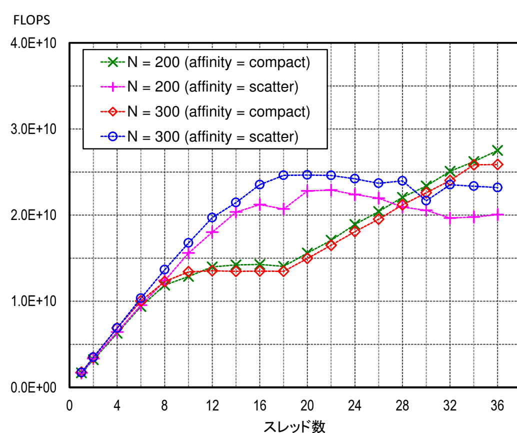


図 9　FDTD naive 実装の性能とスレッド数の関係（$N = 200, 300$）.

表 2　stream ベンチマーク Triad の結果と FDTD naive 実装の性能の比較（$N = 200$）: stream ベンチマークの要素数は N^3 個（double 型）.

affinity	compact		scatter	
スレッド数	18	36	18	36
stream Triad (GB/sec)	64.7	134.1	100.6	89.2
FDTD 予測性能 (GFLOPS)	13.1	27.2	20.4	18.1
FDTD 実測性能 (GFLOPS)	14.1	27.5	20.7	20.1
実測性能 / 予測性能	1.07	1.01	1.01	1.11

表 3　stream ベンチマーク Triad の結果と FDTD naive 実装の性能の比較（$N = 300$）: stream ベンチマークの要素数は N^3 個（double 型）.

affinity	compact		scatter	
スレッド数	18	36	18	36
stream Triad (GB/sec)	64.9	127.1	115.8	108.4
FDTD 予測性能 (GFLOPS)	13.2	25.8	23.5	22.0
FDTD 実測性能 (GFLOPS)	13.5	25.9	24.6	23.2
実測性能 / 予測性能	1.02	1.00	1.05	1.05

り，得られた naive 実装の性能は妥当であると判断できる.

5.3　時空間タイリングを施した実装の性能

時空間タイリングを施した 4 種類の実装（pxpypz 型，dxpypz 型，dxdypz 型，dxdydz 型）の性能を報告する．なお，性能評価はスレッド数は 36，affinity は compact として行った．また，時空間タイリングに関する各種パラメータについては，経験的に好ましいと思われる候補を中心に設定した．

まず，$N = 200$ における各実装の，naive 実装に対する性能向上を**図 10**(a) から (d) に示す．また，各実装の中で最も性能が高かったケースとその際のパラメータ設定を**表 4** にまとめた．表 4 から分かるように，今回の $N = 200$ に関する評価では，dxdypz 型の性能向上 1.88 が最高であった．

次に，$N = 300$ に対する同様の結果を**図 11**(a) から (d) および**表 5** に示す．表 5 より，$N = 300$ の場合には，dxpypz 型の性能向上 2.22 が最高であった．

5.4　考察

今回の性能評価で得られた結果について考察を述べる．

まず，図 10(a) や図 11(a) から，先行研究 [3] が示すように，タイルレベルの並列性を用いない時空間タイリング（pxpypz 型）でも性能が向上することが確認できた．ただし，図 10(a) や図 11(a) から分かるように，タイル内の処理の並列性を十分に確保するために，タイルサイズを十分大きくする必要がある．

次に，図 10(b) および図 11(b) が示すように，dxpypz 型の時空間タイリングを用いた場合，$N = 200$ ではあまり性能が向上しないが，$N = 300$ では高い性能向上が確認されている．dxpypz 型では，x 軸方向のみタイルレベルの並列性が存在するが，$N = 200$ で $\mathrm{BLX} = 0, \mathrm{BLT} = 4$ の場合，x 軸方向のタイル数が高々 22 個で，スレッド数に対して少ない．一方，$N = 300$ で同じ設定だと，タイル数が 34 個まで増加する．この点が，$N = 200$ と $N = 300$ で効果が大きく異なったことの主要因だと推測できる．また，どちらの場合でも，BLX や BLT を大きくすると，x 軸方向のタイルサイズが大きくなり，結果として，タイル数の減少（つまり，並列性の低下）に伴う性能低下が生じていることが，図 10(b) や図 11(b) から確認できる．

一方，図 10(c) および図 11(c) から分かるように，dxpypz 型と異なり，dxdypz 型は $N = 200$ と $N = 300$ の両方の場合で，一定の性能向上を示した．これは，dxdypz 型では 2 次元（x 軸方向と y 軸方向）のタイルレベルの並列性があり，問題サイズが小さい場合でも，スレッド数に対して十分な並列性が存在するからであると考えられる．例えば，$N = 200$ で $\mathrm{BLX} = \mathrm{BLY} = 0, \mathrm{BLT} = 4$ の場合，$484 (= 22^2)$ 個のタイルが独立に処理できる．そのため，タイルサイズを大きくすることも可能で，例えば，BLT を 4 から 6 とした方が性能が良くなる事例も確認できる．

今回の問題サイズとスレッド数では，2 次元のタイルレベルの並列性（dxdypz 型）で十分であったため，dxdydz 型を用いて，タイルレベルの並列性を 3 次元に増やしても，タイルの種類の増加に伴うスレッド間の同期回数の増加

ⓒ 2017 Information Processing Society of Japan

表 4 各実装の比較（$N = 200$）：各実装の最高性能とその際のパラメータの設定.

実装	GFLOPS	性能向上	BLX	BLY	BLZ	BLT
naive	27.5	–	–	–	–	–
pxpypz	42.4	1.54	128	64	128	32
dxpypz	34.9	1.27	0	4	128	4
dxdypz	51.7	1.88	4	4	256	4
dydydz	44.3	1.61	0	0	128	6

表 5 各実装の比較（$N = 300$）：各実装の最高性能とその際のパラメータの設定.

実装	GFLOPS	性能向上	BLX	BLY	BLZ	BLT
naive	25.9	–	–	–	–	–
pxpypz	39.7	1.53	128	32	128	64
dxpypz	57.5	2.22	0	1	128	4
dxdypz	53.8	2.08	0	0	256	6
dydydz	46.7	1.80	0	0	128	6

等の影響により，naive 実装よりは性能が向上するものの，dxpypz 型や dxdypz 型の性能向上には及ばなかった.

6. 関連研究

反復型ステンシル計算の時空間タイリングに関する研究は，文献 [2] に端を発し，現在まで非常に活発に行われている（文献 [6], [7] 等で良く概観されている）. 例えば，ダイヤモンド型と平行四辺形型のタイルを組み合わせる手法は，文献 [8], [9] 等で提案・評価されている. また，文献 [10] では分散並列実装時の性能評価も行われている. これらの文献における性能評価では，2 次元 5 点型や 3 次元 7 点型のステンシルがテスト問題として用いられており，今回の 3 次元 FDTD 法のステンシルのような複雑な形状のステンシルに関する性能評価は行われていない. また，2 つ以上の次元にダイヤモンド型のタイリングを採用できることは上記の文献等でも指摘されているが，実際にその性能を評価した事例については，著者の知る範囲では見当たらない.

これらのような, 時空間タイリング手法そのものの研究に加えて，タイリングコードの自動生成・変換等に関する研究も活発に行われている. 例えば，PLUTO [11]，Pochir [12]，Physis [13] などが良く知られており，利便性の向上等を目指した研究（[14]）が最近でも行われている. また，メモリアクセスコストだけでなく，大規模問題を扱う場合のディスクからのデータ転送コストを軽減することを目的とした研究 [15] もなされている.

FDTD 法に関しては，現象の 2 次元性を仮定した（例えば，$\boldsymbol{E}_z = 0, \boldsymbol{H}_x = \boldsymbol{H}_y = 0$ の）場合，残った変数を消去して，計算式を変更することで，空間 2 次元の比較的シンプルな反復型ステンシル計算に帰着できる. そのため，このケースについては上記のツール等の適用も比較的容易であることから，過去の研究の性能評価における代表的なベンチマークの一つ（2D-FDTD 等と呼ばれる）となっており，多くの報告がある.

一方，実応用分野において，2 次元 FDTD 法と比べてその利用事例が多い 3 次元 FDTD 法については，著者らの知る限り，報告例は 2 次元解析と比して極めて少ない. 著者らはこれまでに，文献 [16][3][17] 等において，3 次元 FDTD 法に対する時空間タイリングの適用可能性やタイルサイズ等の性能パラメータの自動チューニングに関する報告を行ってきたが，これらは先駆的な研究とみなすことができる. 但し，近年では 3 次元 FDTD 法においても時空間タイリング手法の利用が広がってきており，例えば，文献 [18] では複数 GPU を対象とした分散並列実装が行われ，より高精度な離散化手法に関する言及もなされている. ただし，上記の報告においても，本研究で用いたダイヤモンド型のタイリングを空間 2 次元以上で採用する手法については述べられておらず，性能評価時の計算領域についても，導波管のような一方向に著しく長い直方体型領域を用い，空間 1 次元のダイヤモンド型タイリングで十分な並列性が得られるモデルが対象となっている.

その他，具体的なアプリケーションプログラムにおける反復型ステンシル計算に対して時空間タイリングを適用した最近の研究事例としては，周波数領域での電磁場解析を対象とした研究 [19] や, 非逐次型データ同化手法であるアジョイント法に関する研究 [20] が挙げられる.

7. おわりに

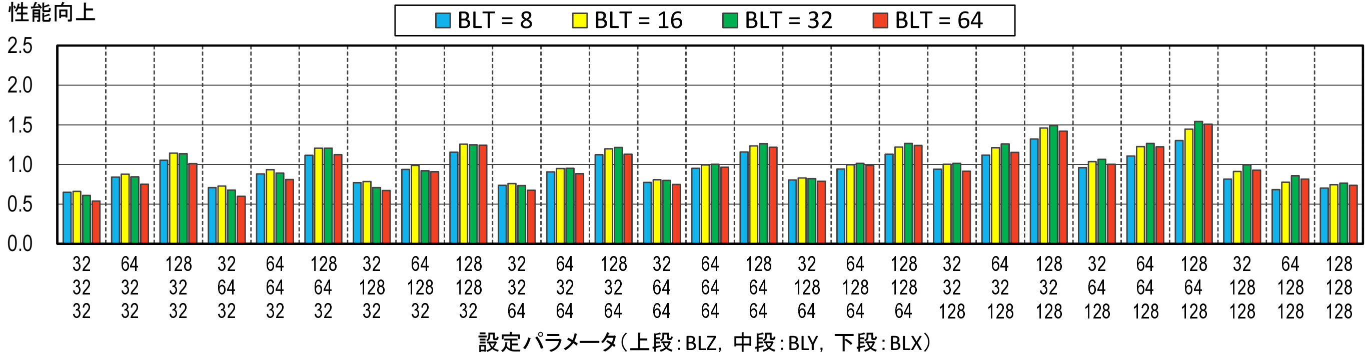
本論文では，高周波電磁場解析の代表的数値計算手法である 3 次元 FDTD 法に対する時空間タイリングに関して，タイルレベルでの並列処理の観点で有望な特徴を持つ，ダイヤモンド型のタイリングを採用した手法の効果を検証した. 実際にダイヤモンド型を採用した時空間タイリング手法を適用した 3 次元 FDTD 法のプログラムを実装し，最新の Xeon 環境（2CPU，36 スレッド）で性能評価を行った結果，素朴な実装に対して，200^3 のサイズの問題で dxdypz 型（空間 2 次元でダイヤモンド型を採用）の時空間タイリングにより 1.88 倍，300^3 のサイズの問題で dxpypz 型（空間 1 次元でダイヤモンド型を採用）の時空間タイリングにより 2.22 倍の性能向上を達成した.

また，今回の性能評価を通して得られた主な知見としては，

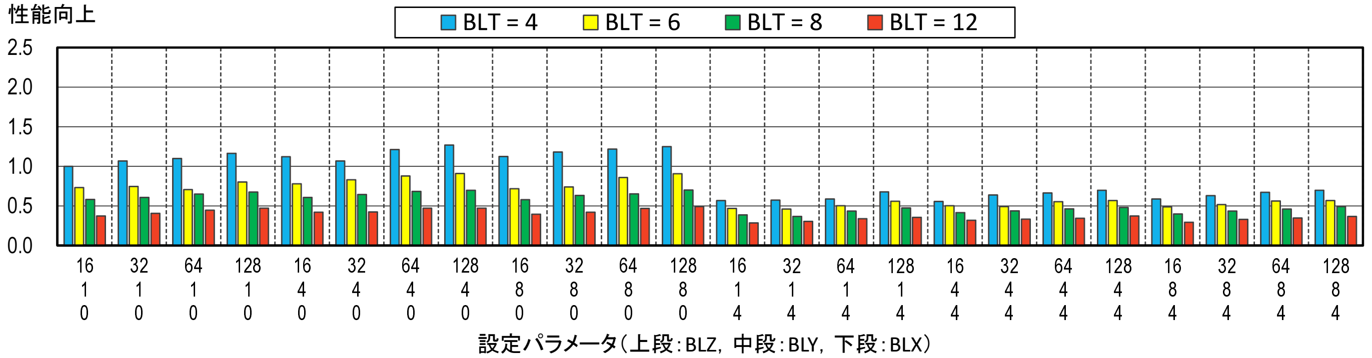
- タイルレベルの並列性が乏しい先行研究 [3] の手法（pdpypz 型）より，タイルレベルの並列性が十分に存在する手法（dxpypz 型や dxdypz 型）の方が，良い性能を得られる可能性が高い.
- 使用するスレッド数に対して，相対的に問題サイズが小さい場合，複数次元に関してタイルレベルの並列性が存在する手法（dxdypz 型）が有望となる.

が挙げられる.

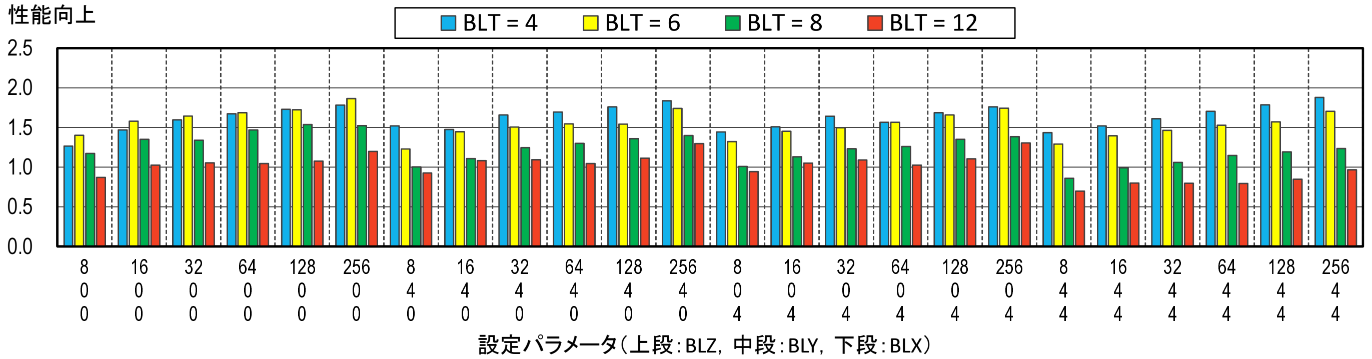
一方，主な今後の課題は以下の通りである.



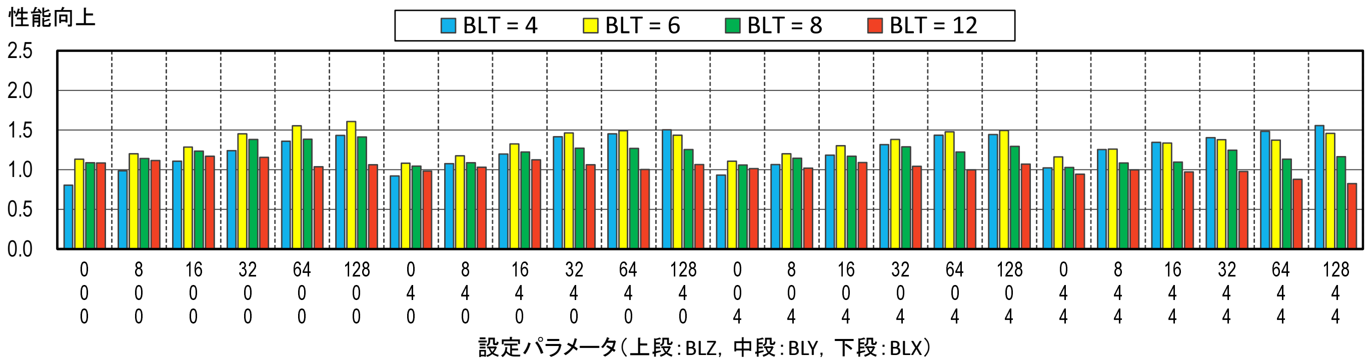
(a) pxpypz 型



(b) dxpypz 型



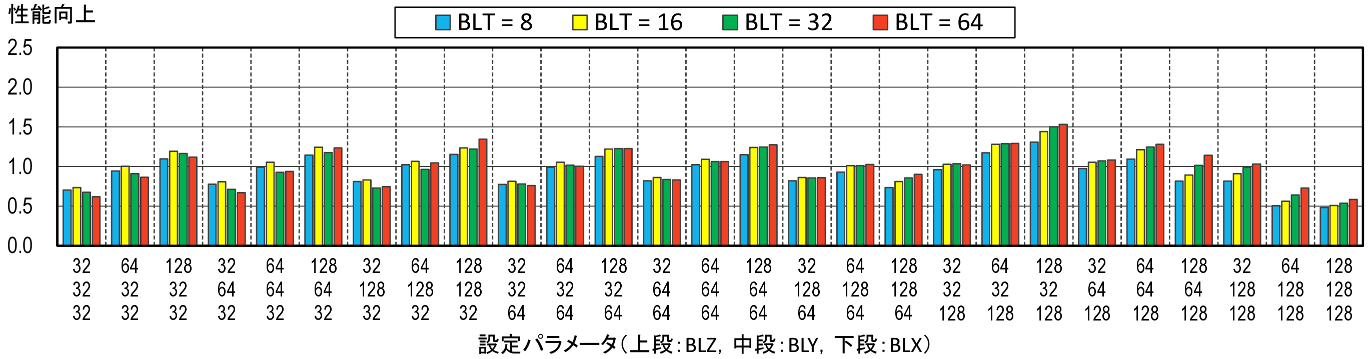
(c) dxdypz 型



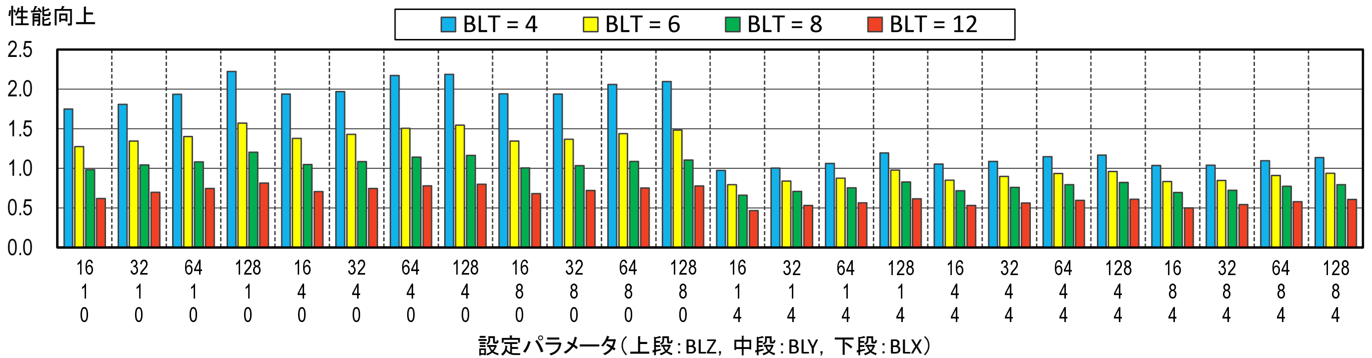
(d) dxdydz 型

図 10 naive 実装に対する性能向上（$N = 200$，36 スレッド, affinity=compact）.

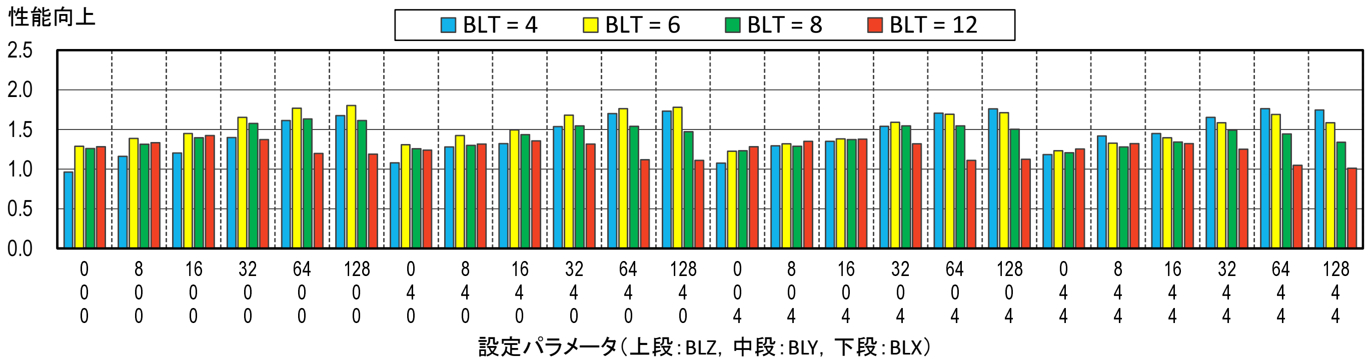
ⓒ 2017 Information Processing Society of Japan



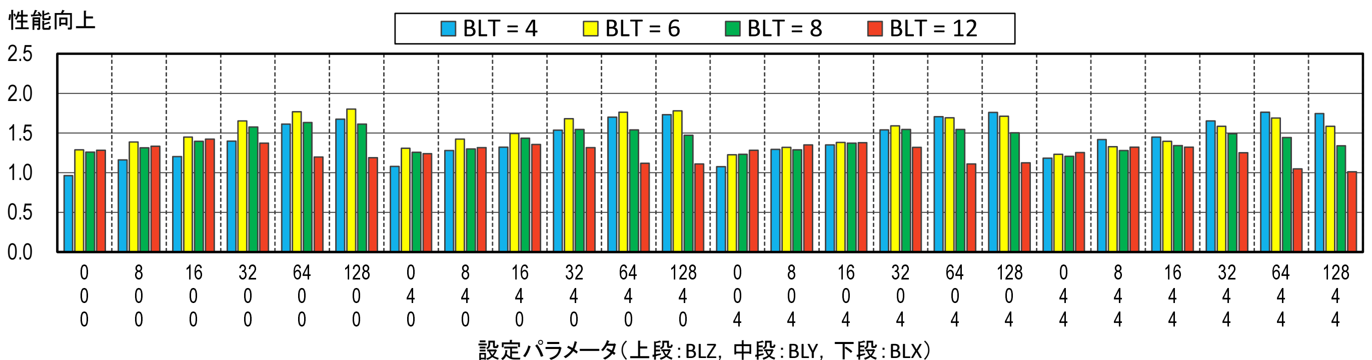
(a) pxpypz 型



(b) dxpypz 型



(c) dxdypz 型



(d) dxdydz 型

図 11 naive 実装に対する性能向上（$N = 300$，36 スレッド, affinity=compact）.

ⓒ 2017 Information Processing Society of Japan

- プロファイラ等を用いた，キャッシュメモリのヒット率の違いなどの調査
- タイル形状に関するパラメータと性能の関係の詳細な調査，およびパラメータの（自動）チューニング手法の検討
- メニーコア環境での性能評価
- 実際のアプリケーションにおける境界条件（PML 等）を含めた実装と性能評価
- 3 次元 FDTD 法以外の反復型ステンシル計算への展開．

謝辞 本研究は JSPS 科研費（課題番号：JP15H02709）及び学際大規模情報基盤共同利用・共同研究拠点（課題番号：jh160039-NAJ）の援助を受けている。

参考文献

[1] Yee, K. S.: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antennas and Propagation*, pp. 302–307 (1966).

[2] Wolfe, M.: More Iteration Space Tiling, *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, ACM, pp. 655–664 (1989).

[3] 南　武志，岩下武史，中島　浩：冗長計算を伴わない 3 次元 FDTD 法の時空間タイリング，情報処理学会論文誌：コンピューティングシステム，Vol. 6, No. 1, pp. 56–65 (2013).

[4] 宇野　亨，何　一偉，有馬卓司：数値電磁界解析のための FDTD 法：基礎と実践，コロナ社 (2016).

[5] McCalpin, J. D.: Memory Bandwidth and Machine Balance in Current High Performance Computers, *IEEE TCCA Newsletter*, pp. 19–25 (1995).

[6] Orozco, D., Garcia, E. and Gao, G.: Locality Optimization of Stencil Applications Using Data Dependency Graphs, *Proceedings of the 23rd International Conference on Languages and Compilers for Parallel Computing*, LCPC'10, Springer-Verlag, pp. 77–91 (2011).

[7] Zhou, X.: Tiling Optimizations for Stencil Computations, PhD Thesis, University of Illinois at Urbana-Champaign (2013).

[8] : Cache Accurate Time Skewing in Iterative Stencil Computations, *2011 International Conference on Parallel Processing*, pp. 571–581 (2011).

[9] Grosser, T., Cohen, A., Holewinski, J., Sadayappan, P. and Verdoolaege, S.: Hybrid Hexagonal/Classical Tiling for GPUs, *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '14, ACM, pp. 66:66–66:75 (2014).

[10] Malas, T., Hager, G., Ltaief, H., Stengel, H., Wellein, G. and Keyes, D.: Multicore-optimized wavefront diamond blocking for optimizing stencil updates, *sisc*, Vol. 37, No. 4, pp. C439–C464 (2015).

[11] Bondhugula, U., Hartono, A., Ramanujam, J. and Sadayappan, P.: A practical automatic polyhedral parallelizer and locality optimizer, *ACS SIGPLAN Notices*, Vol. 43, No. 6, pp. 101–113 (2008).

[12] Tang, Y., Chowdhury, R. A., Kuszmaul, B. C., Luk, C.-K. and Leiserson, C. E.: The Pochoir Stencil Compiler, *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, ACM, pp. 117–128 (2011).

[13] Maruyama, N., Nomura, T., Sato, K. and Matsuoka, S.: Physis: An Implicitly Parallel Programming Model for Stencil Computations on Large-scale GPU-accelerated Supercomputers, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, ACM, pp. 11:1–11:12 (2011).

[14] 黒田勝汰，遠藤敏夫，松岡　聡：ディレクティブによる時空間ブロッキングの自動適用，情報処理学会研究報告：ハイパフォーマンスコンピューティング（HPC），Vol. 2016-HPC-157, No. 18, pp. 1–9 (2016).

[15] 緑川博子，丹　英之：Flash を用いた Out-of-core ステンシル計算のための最適ブロッキングパラメタ自動チューニングシステム，情報処理学会研究報告：ハイパフォーマンスコンピューティング（HPC），Vol. 2016-HPC-155, No. 36, pp. 1–9 (2016).

[16] 南　武志，高橋康人，岩下武史，中島　浩：キャッシュメモリを考慮した 3 次元 FTDT カーネルの性能改善，情報処理学会論文誌：コンピューティングシステム，Vol. 4, No. 2, pp. 70–83 (2011).

[17] Minami, T., Hibino, M., Hiraishi, T., Iwashita, T. and Nakashima, H.: *Automatic Parameter Tuning of Three-Dimensional Tiled FDTD Kernel*, pp. 284–297, Springer International Publishing (2015).

[18] Zakirov, A., Levchenko, V., Perepelkina, A. and Zempo, Y.: High performance FDTD algorithm for GPGPU supercomputers, *J. Phys: Conference Series*, Vol. 759, No. 1, p. 012100 (2016).

[19] Malas, T. M., Hornich, J., Hager, G., Ltaief, H., Pflaum, C. and Keyes, D. E.: Optimization of an electromagnetics code with multicore wavefront diamond blocking and multi-dimensional intra-tile parallelization, *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 142–151 (2016).

[20] 池田朋哉，伊藤伸一，長尾大道，片桐孝洋，永井　亨，荻野正雄：アジョイント法における Forward model への階層ブロッキング適用による高性能化，情報処理学会研究報告：ハイパフォーマンスコンピューティング（HPC），Vol. 2016-HPC-157, No. 17, pp. 1–8 (2016).

ⓒ 2017 Information Processing Society of Japan

正誤表

下記の箇所に誤りがございました．お詫びして訂正いたします．

訂正箇所	誤	正
10 ページ 図 11(c)		

ⓒ2017 Information Processing Society of Japan