Vol.2018-HPC-164 No.6
2018/5/7

Knights Landingにおける Tilied 3D FDTDカーネルの性能評価

深谷 猛[1,a)]　岩下 武史[1]

概要：3 次元 FDTD 法は高周波電磁場解析において頻繁に用いられる数値計算手法であり，その計算パターンは反復型ステンシル計算に分類される．そのため，計算機のメモリアバンド幅に性能が律速し，それに対して，時空間タイリングによりメモリアクセスコストを軽減し，性能向上を図る試みが研究されている．これまで，著者らは，3 次元 FDTD 法に対して，タイルレベルの並列処理を有する時空間タイリング手法を研究しており，最新のマルチコア CPU 環境で，その効果を確認している．本稿では，代表的なメニーコア CPU である，Knights Landing 世代の Intel Xeon Phi プロセッサ（KNL）上で，3 次元 FDTD 法に対する時空間タイリングの効果を検証した結果を報告する．KNL は MCDRAM と呼ばれる高速メモリを有しているなど，汎用 Xeon とは異なった特徴を持っている．そのため，これまでの時空間タイリング手法をそのまま適用しても，十分な効果が得られるとは限らない．今回の性能評価では，汎用 Xeon 向けに開発した，時空間タイリングを用いたプログラムコードをそのまま KNL に移植し，タイルサイズのチューニングのみを行った．そのため，MCDRAM 上にデータを配置した素朴な実装に対して，性能向上を確認することができなかったが，適切なタイルサイズの選択について，汎用 Xeon の場合とは異なる傾向を確認することができるなど，今後のプログラム改良に有益な知見を得ることができた．

1. はじめに

3 次元 FDTD（Finite Difference Time Domain）法 [1] は，アンテナ設計等に代表される，高周波電磁場解析の主要な数値計算手法である [2], [3]．3 次元 FDTD 法の計算パターンは反復型ステンシル計算に分類されるため，一般的にその性能は計算機の（実効）メモリバンド幅律速となる．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させ，プログラムの性能を向上させる手法として，時空間タイリングが幅広く研究されている [4], [5]．著者らは，これまでに，3 次元 FDTD 法に対する時空間タイリングを研究しており，最近，タイルレベルの並列性を有する時空間タイリングを 3 次元 FDTD 法に適用し，最新のマルチコア CPU 環境で，その効果を検証した [6], [7].

今回，3 次元 FDTD 法に対する時空間タイリング手法の研究の一環として実施した，代表的なメニーコアプロセッサである，Knights Landing 世代の Intel Xeon Phi プロセッサ（以降，KNL）を用いた性能評価の結果について報告する．KNL は，汎用 Xeon の数倍の演算コアを有するため，タイルレベルの並列性を持つ時空間タイリング手法と相性が良いことが期待される．一方で，MCDRAM と呼ばれる高速メモリを搭載するなど，汎用 Xeon と異なる特徴を持つため，汎用 Xeon 向けのタイリング手法（やその実装）が適しているとは限らない．そのため，時空間タイリングを施した 3 次元 FDTD 法のプログラム（Tiled 3D FDTD カーネル）に関して，KNL 上での性能の挙動を明らかにすることは重要な課題である．

今回の性能評価では，KNL 向けのプログラム最適化の第一段階として，まず，汎用 Xeon 向けに開発した Tiled 3D FDTD カーネルをそのまま移植し，タイルサイズのチューニングのみを行うこととした．その上で，KNL の（MCDRAM の）メモリモードや SIMD 化の有無など，実行時の条件を変えて，プログラムの性能を測定するとともに，適切なタイルサイズの傾向について調査した．性能評価の結果，MCDRAM を利用しない（Flat モードでデータをメインメモリ上に配置する）場合には，汎用 Xeon の場合と同様に，時空間タイリングにより性能が向上することが確認でき，適切なタイルサイズについても類似の傾向を持つことが分かった．しかし，MCDRAM を利用する（Flat モードでデータを MCDRAM 上に配置する，あるいは Cache モードの）場合には，汎用 Xeon 向けのプログラムコードで，タイルサイズをチューニングしただけでは，素朴な実装に対して性能向上が得られなかった．また，タイルサイズと性能の関係についても，汎用 Xeon の場合とは異なる傾向が

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

あることが確認された．これらの結果から，MCDRAM に関するデータのアクセスコストを削減するためには，（汎用 Xeon 等の）メインメモリに対する時空間タイリングとは異なる手法（や実装方法）が必要であることが明らかとなった．

以下，2 節で 3 次元 FDTD 法，3 節で 3 次元 FDTD 法に対する時空間タイリング，4 節で時空間タイリングを用いたプログラムの実装，について概説する．その後，5 節で KNL 上での性能評価の結果を報告する．6 節で関連研究を紹介し，7 節でまとめと今後の課題を述べる．

2. 3 次元 FDTD 法

3 次元 FDTD 法では，Maxwell 方程式に基づく偏微分方程式

$$\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t},$$
$$\nabla \times \boldsymbol{H} = \epsilon \frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E},$$

を，空間方向に Yee メッシュ，時間方向に Leap-flog 法を用いて離散化する [1]．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 法の詳細については，文献 [8], [9] 等を参照されたい．

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．図 1 から分かるように，電場（配列 `Ex`, `Ey`, `Ez`）と磁場（配列 `Hx`, `Hy`, `Hz`）の値を交互に繰り返し計算する構造となっている．そして，空間の各点の電場や磁場の値は，周囲の点の値を用いて，一定のパターン（ステンシル）に基づいて更新されるため，この計算は反復型ステンシル計算に分類される．そのため，一般的に，素朴に実装した 3 次元 FDTD 法のプログラムの性能は，計算機の（実効）メモリバンド幅律速となることが知られている．

3. 3 次元 FDTD 法に対する時空間タイリング

本節では，3 次元 FDTD 法に対する時空間タイリング手法の概要を述べる．なお，詳細については，文献 [6], [7] を参照されたい．

時空間タイリングは，ループタイリング（ループブロッキング）の一種である．時間と空間のネストしたループの反復空間を小領域（タイル）に分割し，タイル単位で計算を行うことで，データ参照の局所性を向上させ，メインメモリへのアクセスコストの削減を図る．反復型ステンシル計算では，時間ステップ間で計算順序の依存関係が存在するため，依存関係を考慮した上で反復空間をタイルに分割することが求められる．例えば，1 次元 3 点ステンシルの場合，冗長計算を必要としない時空間タイリングとして，平行四辺形型とダイヤモンド型の 2 種類のタイリング手法がよく知られている．平行四辺形型の場合，タイルの形状は 1 種類であるが，タイル間に処理の依存関係が存在し，タイルレベルの並列処理は困難である．一方，ダイヤモンド型の場合，タイルの形状が 2 種類（山形と谷型）となり，山（谷）型のタイル同士を同時に処理することが可能である（タイルレベルの並列性がある）．

表 1 タイリング手法の組み合わせ方（P : 平行四辺形型，D : ダイヤモンド型）．

表記	T-X	T-Y	T-Z	タイルレベルの並列性
`pxpypz`	P	P	P	（超平面上）
`dxpypz`	D	P	P	X 軸
`dxdypz`	D	D	P	X, Y 軸
`dxdydz`	D	D	D	X, Y, Z 軸

表 2 各タイリング手法で必要なタイルの形状（`p` : 平行四辺形型，`m` : ダイヤモンド 山型，`v` : ダイヤモンド 谷型）．なお，括弧内は左から X，Y，Z 軸のタイル形状を示す．

ステップ	`pxpypz`	`dxpypz`	`dxdypz`	`dxdydz`
1	`(p,p,p)`	`(m,p,p)`	`(m,m,p)`	`(m,m,m)`
2	–	`(v,p,p)`	`(v,m,p)`	`(v,m,m)`
3	–	–	`(m,v,p)`	`(m,v,m)`
4	–	–	`(v,v,p)`	`(m,m,v)`
5	–	–	–	`(v,v,m)`
6	–	–	–	`(v,m,v)`
7	–	–	–	`(m,v,v)`
8	–	–	–	`(v,v,v)`

3 次元 FDTD 法では，時間 1 次元と空間 3 次元の，合計 4 次元の反復空間を持つ．我々は，時間 1 次元と空間 1 次元の 2 次元空間に対する時空間タイリング手法を組み合わせることで，3 次元 FDTD 法に対して時空間タイリングを適用するアプローチを採用する．まず，電場と磁場のそれぞれについて，`x`, `y`, `z` の 3 つの要素をまとめて，格子点間の依存関係を考える．その上で，依存関係を踏まえて，時間・空間の 2 次元空間におけるタイルの形状を決定する．具体的なタイルの形状は，**図 2**(a) および (b) に示した通りである．なお，図における `BLX`, `BLT` はタイルの形状を決定するパラメータである．

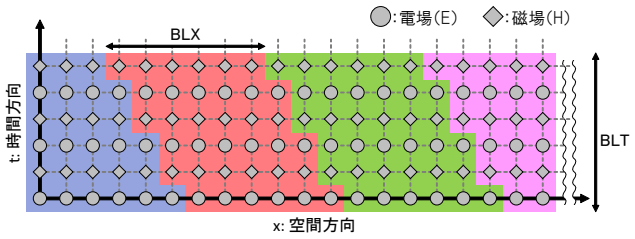
上述の時間・空間の 2 次元空間に対するタイリング手法を，X，Y，Z 軸のそれぞれで選択することで，3 次元 FDTD 法に対する時空間タイリングを実現する．今回の性能評価では，**表 1** に示す，4 種類の組み合わせを評価の対象とする．なお，`pxpypz` は，超平面法によりタイルレベルの並列処理が可能ではあるが，今回の評価ではこれを用いた実装を行わず，タイル内で並列処理を行う．表 1 に示した各組み合わせ方に応じて，必要なタイルの形状数（処理のステップ数）が異なる．具体的には，**表 2** に挙げた通りとなる．

4. 時空間タイリングを用いた実装の概要

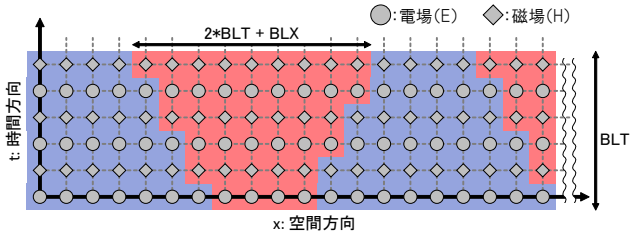
本節では，時空間タイリングを用いた 3 次元 FDTD 法のプログラムの実装の概要を述べる．なお，プログラムは C

```
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 法の計算の主要部（素朴な実装）.



(a) 平行四辺形型



(b) ダイヤモンド型（の拡張）

図 2　3 次元 FDTD 法に対する，時間・空間 2 次元におけるタイルの形状.

言語で作成し，スレッド並列化は OpenMP により行う.

まず，素朴な実装を含めた共通事項を以下に挙げる.

- 電場（磁場）の各要素ごとに配列を確保する.
- 境界を含めた格子点の形状を NX × NY × NZ としたとき，格子点 (x, y, z) に対して，配列のインデックスを (x*NY+y)*NZ+z と対応させる.

素朴な実装は，図 1 に記載した内容に基づいている．一方，時空間タイリングを用いた実装は全て図 **3** に示した構造に基づいている．以下に，図 3 に関する補足説明を記載する．なお，以下では，X 軸を例として記載している.

- 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：タイルの形状の数（TileX[] の要素数）
- TileX[]：タイルの形状に関する配列（表 2 を参照）
- 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）

また，スレッド並列化と SIMD 化については，以下の通りである.

- 最も外側の時間ステップのループの前に omp parallel を置く.
- 最も内側の Z 軸のループの前に omp simd を置く.
- 各実装における処理のスレッド並列化は以下の通り.
  - 素朴な実装：変数 x に関するループの前に omp for collapse(2)
  - pxpypz：変数 x に関するループの前に omp for collapse(2)
  - dxpypz：変数 xx に関するループの前に omp for
  - dxdypz：変数 xx に関するループの前に omp for collapse(2)
  - dxdydz：変数 xx に関するループの前に omp for collapse(3)

なお，配列の初期化についてもスレッド並列化を行うが，ファーストタッチについては特に考慮していない.

5. 性能評価

本節では，KNL を用いて実施した性能評価の結果を報告

```
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 3 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

する．3D FDTD カーネルに関して，まず，素朴な実装の性能を評価し，その次に，時空間タイリングを用いた実装の性能を評価する．最後に，タイルサイズの設定に関して考察を述べる．

5.1 評価環境・評価条件

今回の性能評価は，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 A（Camphor 2）の 1 ノードを用いて行った．システムの緒元は**表 3** に示した通りである．KNL に搭載されている MCDRAM のメモリモードは，以下の 3 種類を試した．

- flat(0) : Flat モードでメインメモリに配列を配置（“`numactl -m 0`” を指定して実行）
- flat(1) : Flat モードで MCDRAM に配列を配置（“`numactl -m 1`” を指定して実行）
- cache : Cache モード

プログラムは Intel コンパイラ（`icc`, ver. 17.0.2）を用いてコンパイルした．コンパイル時のオプションは，`-mcmodel=medium`, `-shared-intel`, `-qopenmp`, `-O3`, `-ipo` を指定し，さらに，

- SIMD 化あり：`-xMIC-AVX512` を加える
- SIMD 化なし：`-no-vec` を加える（同時に，ソースコードから “`#pragma omp simd`” を削除する）

とした．

問題設定は，先行研究 [6], [7], [10], [11] と同じで，金属壁に囲まれた立方体形状の領域の解析を想定した設定とした．計算対象の格子点数は N^3 個（空間の各次元方向に N

表 3 性能評価で用いた計算機の緒元

項目		諸元
CPU	プロセッサ	Intel Xeon Phi 7250
	アーキテクチャ	Knights Landing
	動作周波数	1.40GHz
	コア数	68
	L1 data cache	32KB / core
	L2 cache	1MB / 2cores
	MCDRAM	16GB / CPU
ノード	CPU 数	1
	メモリ	96GB
	Cluster mode	Quadrant

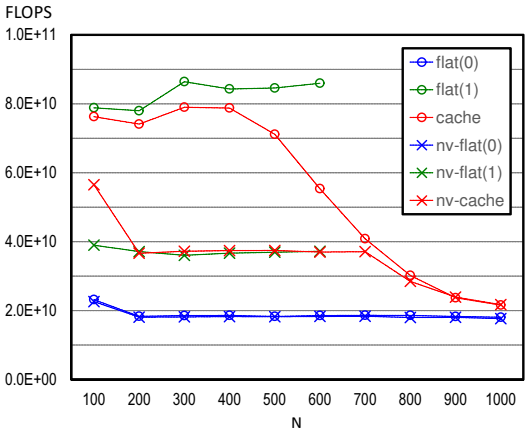


図 4 素朴な実装の性能（68 スレッド）: “nv-” は「SIMD 化なし」の場合（記載なしは「SIMD 化あり」の場合）.

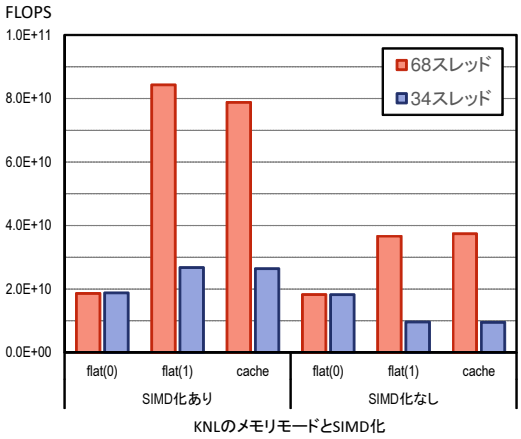


図 5 34 スレッドと 68 スレッドでの素朴な実装の性能の比較（$N = 400$）.

個）とし，境界（金属壁）上の格子点を含めて，各配列を $(N+2)^3$ の大きさで確保した．

プログラムは 1 ノードを占有した状態で，特に記載がない限り，68 スレッドで実行した．なお，KNL の 0 番目のコアから順番に，各コアに 1 スレッドを配置した．時間ステップ数を 512 として，実行時間を測定した．ただし，配列の初期化の時間は除いている．また，性能値（FLOPS 値）は，

$$\frac{39 \times N^3 \times 512}{(\text{実行時間})}$$

で算出した．

5.2 素朴な実装の評価結果

3D FDTD カーネルの素朴な実装の性能について報告する．N を 100 から 1000 まで，100 刻みで変化させて，素朴な実装の性能を測定した結果を**図 4** に示す．なお，メモリモード flat(0) に関しては，MCDRAM の容量を考慮し，$N = 600$ までとした．

図 4 ついて，まず，「SIMD 化あり」の場合（flat(0), flat(1), cache）の結果を考察する．グラフから分かるように，メモリモードが Flat の場合，問題サイズに関わらず，性能がほぼ一定である．今回の KNL のメインメモリの理論ピーク性能は 115.2GB/sec である．よって，3D FDTD カーネルの byte/flop 値（= 192/32）から，メインメモリに律速する場合の FLOPS 値の上限は 23.4GFLOPS となり，flat(0) の結果はメインメモリのバンド幅律速であることが確認できる（上限の 80%程度）．同様に，MCDRAM の実効メモリバンド幅は 490GB/sec 程度であることが報告されているため，予想される FLOPS 値は 100GFLOP 弱となり，flat(1) の結果は MCDRAM のメモリバンド幅律速であると判断できる．

一方，メモリモードが Cache の場合，問題サイズが十分に小さいのであれば，キャッシュメモリとして動作している MCDRAM 上にデータが載ったまま計算を行うことが可能となる．そのため，N が小さい場合に cache の性能が flat(1) の性能の近いのは妥当である．また，問題サイズが大きくなるにつれて，データが MCDRAM から溢れ，メインメモリへのアクセスが必要となる．そのため，N が大きくなるにつれて，cache の性能が flat(1) の性能に漸近していると思われる．なお，$N = 500$（や 600）の場合のデータサイズは MCDRAM の容量よりは小さいが，グラフが示すように，$N = 500$ から cache の性能は段々と低下している．これは，MCDRAM がキャッシュメモリとして動作する際のデータの管理方法に由来すると推測される．

次に，「SIMD 化なし」の場合（nv-flat(0), nv-flat(1), nv-cache）の結果について，考察を述べる．flat(0) と nv-flat(0), あるいは N が十分に大きい場合の cache と nv-cache の性能の差から，性能がメインメモリのバンド幅律速の場合には，SIMD 化の有無が性能に有意な影響を与えないことが読み取れる．逆に，性能が MCDRAM のバンド幅律速の場合，SIMD 化をなしとすることで，性能が 1/2 以下に低下することが，flat(1) と nv-flat(1)（あるいは N が小さい場合の cache と nv-cache）の差から確認できる．MCDRAM と（キャッシュメモリを経由して）データをやり取るする場合に SIMD 化の有無でコストが変わることが原因であると推測されるが，詳細は不明である．

スレッド数と性能の関係についても，簡単な評価を行う．$N = 400$ の場合について，34 スレッドで実行した場合の結果と 68 スレッドの場合の結果とを比較した様子を**図 5** に示す．なお，34 スレッドで実行する際は，コア 0, コア 2, ... と，1 コアおきに 1 スレッドを配置した [*1]．

図 5 から分かるように，flat(0) については，34 スレッドと 68 スレッドの間で性能に差がない．このことから，34 スレッドの段階でメインメモリのバンド幅を使い切っていると予想される．一方，性能が MCDRAM のバンド幅律

[*1] 使用した環境では aprun コマンドでプログラムを実行するため，“-cc none” のオプションを指定した上で，KMP_AFFINITY で陽的にコアを（リストとして）指定して，実行した．

速となる，flat(1) や cache では，68 スレッドを 34 スレッドとすることで，1/2 以下に性能が低下していることが確認できる．スレッド数の減り方以上に性能が低下している点については，KNL ではコア 0 とコア 1 といったように，隣り合う 2 つのコアで L2 cache を共有していることが関係していると考えている．ただし，この点については，コアへのスレッドの割り当て方を変えるなどして，より詳しく調査する必要がある．また，flat(1) と cache に関しては，「SIMD 化なし」の方が，68 スレッドと 34 スレッドの間の性能差が大きい（flat(1) は 1.21 倍，cache は 1.32 倍）が，原因は分かっていない．

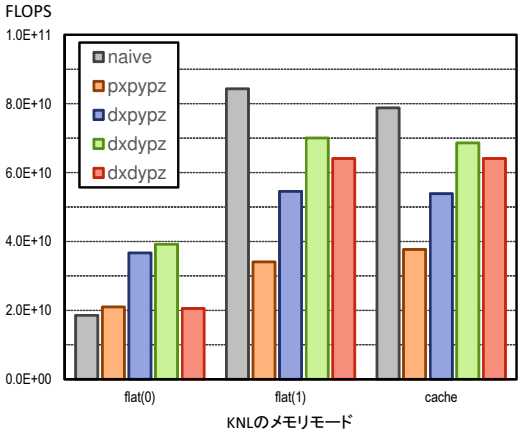
5.3 時空間タイリングを用いた実装の評価結果

時空間タイリングを用いた 3D FDTD カーネルの性能を評価した結果について報告する．$N = 400$ の場合について，4 種類の時空間タイリングの手法（pxpypz, dxpypz, dxdypz, dxdydz）をそれぞれ用いた実装の性能を評価した結果を図 **6**(a) と (b) に示す．なお，ここでは，経験的に選択したタイルサイズの候補の中で最も性能が高かった結果を示しており，タイルサイズの情報は表 **4**(a) と (b) に記載の通りである．また，表中の「データ量」とは，各ケースにおけるタイル 1 個分に対応した計算において参照するメモリ量の概算値である [*2]．

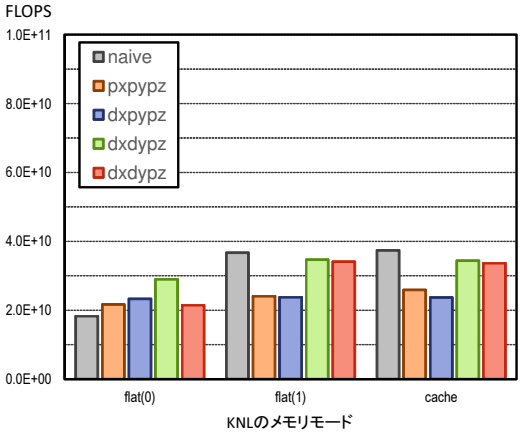
図 6(a) と (b) から明らかなように，SIMD 化の有無に関わらず，flat(0) の場合には時空間タイリングによる（素朴な実装に対する）性能向上が確認できるが，一方で，flat(1) と cache の場合には，時空間タイリングを用いることで逆に性能が低下している．このことから，時空間タイリングにより，メインメモリへのアクセスコストは削減できているが，MCDRAM へのアクセスコストを削減することはできておらず，逆に何らかのオーバーヘッド等が生じてしまっていると考えられる．

表 4(a) と (b) に記載したタイルサイズの情報から，flat(0) の場合には，pxpypz を除いて，タイル内の計算に伴うデータ量がコア当たりの L2 cache の容量（1MB/2）以下となっており，L2 cache を活用することで，素朴な実装よりも性能が向上したと考えるのが妥当である．一方，flat(1) や cache の場合，データ量がコア当たりの L2 cache の容量を超えており，L2 cache を十分に生かしきれていないと予想される．表から分かるように，flat(1) や cache の場合，pxpypz を除いたタイリング手法では，SIMD 化の有無に関わらず，BLZ が 256 となっている．このことから，MCDRAM を利用する場合，メモリの連続方向に一定の長さ以上の連続アクセスをする必要がある可能性が考えられる．SIMD 化を行う場合には，この点は自然であるが，SIMD 化を行わない場合にも影響がある可能性が高いので，より詳細な調査等を行う必要がある．

[*2] 空間の各方向について，平行四辺形型の場合は BLX + BLT，ダイヤモンド型の場合は 2 × BLT + BLX として，それらの積を計算し，さらに 6.5 × 8(byte) を乗じて算出（0.5 は Id の配列に対応）．



(a) SIMD 化あり



(b) SIMD 化なし

図 6 時空間タイリングを用いた実装の性能（68 スレッド，$N = 400$）．

4 種類の時空間タイリングの手法の比較すると，メモリモードと SIMD 化の有無に関わらず，dxdypz の性能が最も高く，flat(1) と cache では dxdydz，flat(0) では dxpypz が次点となった．dxpypz については，タイルサイズの情報から，タイルレベルの並列性がスレッド数よりも少ない（BLX が 0 で，BLT が 4 の場合，タイルレベルの並列性は高々 50 程度）ことが確認できる．一方，dxdypz や dxdydz の場合，2 つ以上の空間方向にタイルレベルの並列性があり，スレッド数に対して十分な並列性が存在する．そのため，dxdypz や dxdydz の方が，dxpypz よりも性能が高くなったと思われる．また，dxdypz の段階で十分な並列性が存在するため，dxdydz とすると，逆にタイルの種類の数が増加することに伴うオーバヘッド等が生じて，性能が dxdypz よりも低くなったと考えられる．なお，タイル内部でスレッド並列化を行う pxpypz では，並列性を確保するために十分大きなタイルサイズとなっており，そのため，タイルレベルの並列性が存在する場合と比べて，L2 cache の効率が低下したためか，性能があまり高くなっていない．

表 4　図 6(a) と (b) の結果におけるタイルサイズの情報

(a) SIMD 化あり

メモリモード	タイリング手法	タイルサイズパラメータ BLX	BLY	BLZ	BLT	データ量 (KiB)
flat(0)	pxpypz	32	32	128	16	16,848
	dxpypz	0	3	32	4	102
	dxdypz	0	0	16	4	65
	dxdydz	0	0	8	4	52
flat(1)	pxpypz	64	32	128	8	19,890
	dxpypz	0	2	256	4	634
	dxdypz	0	0	256	4	845
	dxdydz	0	0	256	4	858
cache	pxpypz	128	128	128	64	359,424
	dxpypz	0	2	256	4	634
	dxdypz	0	0	256	4	845
	dxdydz	0	0	256	4	858

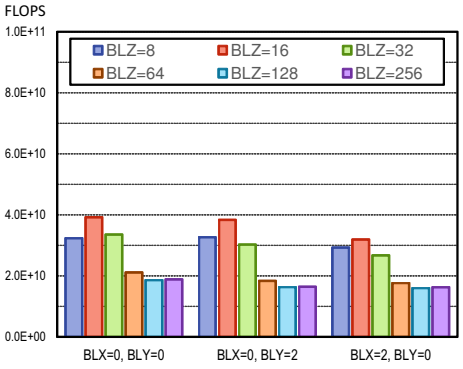
(b) SIMD 化なし

メモリモード	タイリング手法	タイルサイズパラメータ BLX	BLY	BLZ	BLT	データ量 (KB)
flat(0)	pxpypz	32	32	128	16	16,848
	dxpypz	0	1	64	4	138
	dxdypz	0	0	16	4	65
	dxdydz	0	0	8	4	52
flat(1)	pxpypz	32	128	128	64	179,712
	dxpypz	0	2	256	4	634
	dxdypz	0	0	256	4	845
	dxdydz	0	0	256	4	858
cache	pxpypz	64	128	128	32	124,800
	dxpypz	0	3	256	4	739
	dxdypz	0	0	256	4	845
	dxdydz	0	0	256	4	858

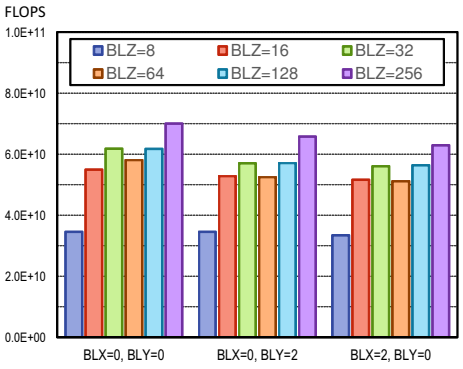
5.4　タイルサイズの設定に関する考察

前節で述べたように，タイルサイズの設定において，BLZ を 256 とすることに意味がある可能性が高い．そこで，dzdypz について，BLZ の大きさと性能の関係に着目して調査した結果を述べる．図 7(a) から (d) は，dzdypz（SIMD 化あり/なし，flat(0) および flat(1)，$N = 400$）に対して，BLZ を変えて性能を測定した結果を示している．なお，BLT は 4 に固定している．

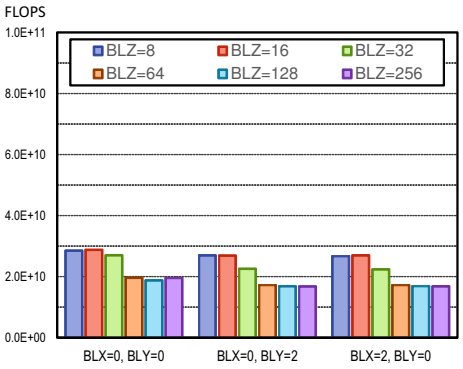
SIMD 化を行った場合において，図 7(a) から分かるように，flat(0) の場合には，BLZ をあまり大きくせず，タイル内の計算に伴うデータ量が，L2 cache の容量（の 1/2）よりも十分に小さくなるように，タイルサイズを設定することが重要である．一方，図 7(b) が示すように，flat(1) では，BLZ を大きくするほど性能が高くなることが確認できる．このように，flat(0) と flat(1) では，BLZ の設定の指針が大きく異なる．同様の傾向は，SIMD 化を行わない場合においても観察することができる．ただし，SIMD 化を行う場合よりも，SIMD 化を行わない場合の方が，flat(1) におい



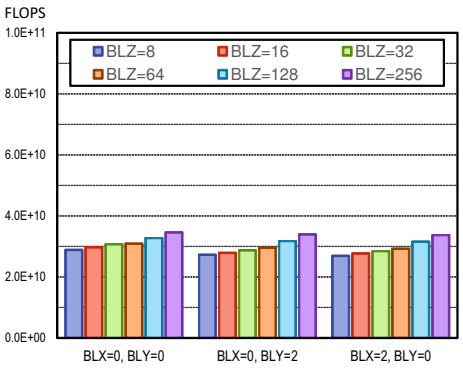
(a) SIMD 化あり：flat(0)



(b) SIMD 化あり：flat(1)



(c) SIMD 化なし：flat(0)



(d) SIMD 化なし：flat(1)

図 7　タイルサイズと性能の関係（dxdypz，BLT=4，68 スレッド，$N = 400$）．

て，BLZ の違いによる性能差が小さくなっている．この結果は，MCDRAM 上にデータを配置した場合における，時空間タイリングのタイルサイズの設定（あるいは，時空間タイリングの実装自体）に関して，通常のマルチコア CPU 環境の場合（flat(0) に近い）とは別の考え方が必要であることを示唆している．また，SIMD 化の有無とは関係のないところで，上記の差に影響を与える要因があると想像できる．

6. 関連研究

文献 [4], [5] 等で概観されているように，反復型ステンシル計算に対する時空間タイリングの研究は幅広く行われている．特に，近年は，並列処理に着目した時空間タイリング手法の研究が活発に行われている [12], [13], [14]．また，タイリングを施したコードの自動生成やコンパイラ・フレームワークに関する研究としては，PLUTO [15]，Pochir [16]，Physis [17] などが著名である．

メニーコア CPU 上での反復型ステンシル計算に対する時空間タイリングの研究としては，松田らが Knights Corner 世代の Xeon Phi を用いた性能評価を実施しているが，時間方向を含めたタイリングをした結果，性能低下が生じたことが報告されている [18]．Knights Landing 世代の Xeon Phi を用いた反復型ステンシル計算に関する研究としては，Cebrián らによる報告があるが，タイリングについては，空間のみ（各時間ステップ内の計算のタイリングのみ）を取り扱っている [19]．時空間タイリングを扱った研究としては，Yount らが SIMD 化から時空間タイリングまで，様々なチューニングを KNL 上での反復型ステンシル計算に対して行った結果を報告している [20]．ただし，時空間タイリングについては，MCDRAM を超えるサイズの問題に対して，MCDRAM を活用することを目的に適用しており，MCDRAM の実効メモリバンド幅を超える性能は得られていない様子である．また，Levchenko らは，Xeon Phi の AVX-512 により適した時空間タイリング手法を提示している [21]．ただし，手法の評価については，Roofline モデルに基づく予測のみであり，実機を用いた結果は報告されていないため，MCDRAM の実効メモリバンド幅以上の性能が得られるかどうかは未知である．

FDTD 法に対する時空間タイリングとしては，多くの研究において，2D-FDTD 等と呼ばれるプログラムがベンチマークの一つとして採用されている．しかし，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，残りの変数を消去して，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ことに注意が必要である．実際，3 次元 FDTD 法のような，複数の要素間に複雑な依存関係がある反復型ステンシル計算に対して，現状では，コード自動生成の適用が不可能であることが言及されている [22]．

3 次元 FDTD 法に対する時空間タイリングとしては，南らが，冗長計算を必要としない時空間タイリング手法を提示し [10]，タイルサイズの自動チューニングも研究している [11]．ここで提示されたタイリング手法は，本稿における pxpypz 型のタイリングに相当するため，スレッド並列化はタイル内部の処理ごととなっていた．これを踏まえて，最近，我々は，3 次元 FDTD 法に対して，タイルレベルの並列性を有する時空間タイリング手法を提示し，その効果を検証した [6], [7]．また，Zakirov らは，マルチ GPU 環境において，3 次元 FDTD 法に時空間タイリングを適用し，性能評価を行っている [23]．論文からは，時空間タイリング手法の詳細について，ダイヤモンド型のタイリングに基づいていることは確認できるが，それ以上に詳しい内容を把握することは困難である [*3]．ただし，PML と呼ばれる境界条件を含めた実装を行っており，より，実際のシミュレーションに近い状況となっている．その他，FDTD 法ではないが，同様に複雑な反復型ステンシル計算に対する時空間タイリングについて，Malas らが Wave-front 型の並列化に基づいた手法を提案している [24]．

7. おわりに

本稿では，時空間タイリングを用いた 3 次元 FDTD 法のプログラムの性能を，KNL 上で評価した結果を報告した．異なるタイルレベルの並列性を持つ時空間タイリング手法について，KNL のメモリモードや SIMD 化の有無を変えて，プログラムの性能評価を行った．今回の性能評価を通して得られた結果としては，メインメモリ上にデータを配置した場合には，時空間タイリングによる性能向上が確認でき，タイルパラメータと性能の関係も汎用 Xeon 上と類似の傾向であることが分かった．一方で，MCDRAM 上にデータを配置した場合（や MCDRAM を Cache として利用した場合）には，時空間タイリングにより性能が低下してしまった．また，タイルサイズと性能の関係についても，汎用 Xeon 上とは異なる傾向が確認された．さらに，この傾向は SIMD 化の有無に関わらず確認されたため，MCDRAM の利用と関係している可能性が高い．時空間タイリングの手法間の比較としては，タイル内部をスレッド並列化するよりも，タイルレベルで処理を並列化し，更に，複数の空間方向においてタイルレベルの並列性が存在する場合の方が，総じて性能が高くなることを確認した．

今後の課題として，今回の性能評価で確認された事象について，プロファイラによる調査等を行うなどして，その原因をより詳しく分析する必要がある．また，単純な構造を持つ反復型ステンシル計算等において，同様の傾向が確認できるか検証を行うことも有益である．そして，これらの調査により，KNL 上での挙動をより詳しく把握した上

[*3] 記述が限られており，さらに，引用している時空間タイリングに関する原著論文がロシア語であるため．

で，それを踏まえて，時空間タイリング手法とその実装方法を改良し，評価を行うことが重要である．

謝辞 本研究は JSPS 科研費（課題番号：JP15H02709）およ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] Lu, J., Thiel, D. and Saario, S.: FDTD analysis of dielectric-embedded electronically switched multiple-beam (DE-ESMB) antenna array, *IEEE Trans. Magn.*, Vol. 38, No. 2, pp. 701–704 (2002).

[3] Yang, F. and Rahmat-Samii, Y.: Microstrip antennas integrated with electromagnetic band-gap (EBG) structures: a low mutual coupling design for array applications, *IEEE Trans. Antennas Propag.*, Vol. 51, No. 10, pp. 2936–2946 (2003).

[4] 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).

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

[6] 深谷 猛，岩下武史：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，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. 2017-HPC-160, No. 35, pp. 1–11 (2017).

[7] Fukaya, T. and Iwashita, T.: Time-space Tiling with Tile-level Parallelism for the 3D FDTD Method, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pp. 116–126 (2018).

[8] 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).

[9] Sullivan, D. M.: *Electromagnetic simulation using the FDTD method*, Wiley-IEEE Press, 2nd edition (2013).

[10] 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).

[11] 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).

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

[13] Yuan, L., Zhang, Y., Guo, P. and Huang, S.: Tessellating Stencils, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pp. 49:1–49:13 (2017).

[14] Malas, T. M., Hager, G., Ltaief, H. and Keyes, D. E.: Multidimensional Intratile Parallelization for Memory-Starved Stencil Computations, *ACM Trans. Parallel Comput.*, Vol. 4, No. 3, pp. 12:1–12:32 (2017).

[15] 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).

[16] 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).

[17] 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).

[18] 松田元彦，丸山直也，滝澤真一朗：Xeon Phi（Knights Corner）の性能特性とステンシル計算の評価，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014-HPC-143, pp. 1–7 (2014).

[19] Cebrián, J. M., Cecilia, J. M., Hernández, M. and García, J. M.: Code modernization strategies to 3-D Stencil-based applications on Intel Xeon Phi: KNC and KNL, *Computers & Mathematics with Applications*, Vol. 74, No. 10.

[20] Yount, C., Duran, A. and Tobin, J.: Multi-level spatial and temporal tiling for efficient HPC stencil computation on many-core processors with large shared caches, *Future Generation Computer Systems*, (online), available from ⟨http://www.sciencedirect.com/science/article/pii/S0167739X17304648⟩ (2017).

[21] Levchenko, V. and Perepelkina, A.: The DiamondTetris Algorithm for Maximum Performance Vectorized Stencil Computation, *Parallel Computing Technologies*, pp. 124–135 (2017).

[22] Henretty, T., Veras, R., Franchetti, F., Pouchet, L.-N., Ramanujam, J. and Sadayappan, P.: A Stencil Compiler for Short-vector SIMD Architectures, *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, ACM, pp. 13–24 (2013).

[23] 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).

[24] 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).