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　70–83 (Mar. 2011)

キャッシュメモリを考慮した 3 次元 FDTD カーネルの性能改善

南　武志[†1]　高橋康人[†2]
岩下武史[†1]　中島浩[†1]

本論文では，高周波電磁場解析の一手法である 3 次元 FDTD 法におけるキャッシュメモリを考慮した性能改善手法に関して述べる．3 次元 FDTD 法の計算カーネルは時間発展に関するループにより与えられ，各タイムステップにおいて電場と磁場の値が交互に更新される．FDTD カーネルは演算あたりのロード/ストア量が大きく，一般にメモリ帯域の影響を受けやすい計算である．そこで，本論文では解析領域をタイルと呼ばれる小領域に分割し，これらのタイルでの電磁場計算を複数タイムステップ分まとめて計算することにより，キャッシュメモリを有効活用し，3 次元 FDTD カーネルを高速に実行する方法を提案する．提案手法を評価した結果，AMD 製クアッドコア Opteron プロセッサよる数値実験において 4 スレッドによる並列処理を行った場合，一般的な 3 次元 FDTD 法の実装と比較して約 50%の性能向上を得た．また，タイルサイズがキャッシュメモリの約 25%程度の場合に最適な性能を発揮することを確認した．

Cache-aware Performance Improvement of Three-Dimensional FDTD Kernel

Takeshi Minami,[†1] Yasuhito Takahashi,[†2]
Takeshi Iwashita[†1] and Hiroshi Nakashima[†1]

This paper deals with performance improvement of three dimensional FDTD kernel for high frequency electromagnetic field analyses. The FDTD method is one of explicit time stepping methods. The electric and magnetic fields are updated alternately in each time step. Since the calculation of the FDTD method has a large byte/flop ratio, its performance is strongly affected by memory throughput. In this paper, we propose a cache-aware three dimensional FDTD method, in which the analyzed domain is divided into multiple small domains. By updating electrical and magnetic fields in each small domain in multiple time steps, we can utilize cache data efficiently. Numerical tests on a quad-core AMD Opteron processor show that the proposed three dimensional FDTD method attains up to 50 percent speedup compared with an ordinary implementation of the three dimensional FDTD method. Furthermore, the proposed three dimensional FDTD method demonstrated the best performance when the tile size was about 25 percent of the size of the cache memory.

1. はじめに

計算機による電磁場解析シミュレーションは重要な科学技術計算の 1 つであるが，解析対象の大規模化・複雑化にともない計算時間が増大しており，シミュレーションの高速化が要望されている．計算機シミュレーションの高速化のための手法の 1 つとして，シミュレーションを行う計算機システムに合わせたチューニングを行うことによる性能改善があげられる．

本論文では，高周波電磁場解析において主要な解析手法の 1 つである FDTD（Finite Difference Time Domain）法[1)] の性能改善手法について述べる．FDTD 法は解析空間内の電場および磁場を与えられた初期値から時間ステップごとに陽的に解いていく手法であり，時間ステップを進めるごとに電磁場の値を更新する．しかしながら，通常のシミュレーションにおいて解析に必要なデータサイズは使用プロセッサのキャッシュ容量よりもはるかに大きく，1 演算あたりのメモリデータ参照・更新回数も大きいため，大量のメモリアクセスが必要となる．そのため，メモリの帯域が計算速度のボトルネックとなりプロセッサの演算性能と比較して十分な性能が得られない場合が多い．さらに，マルチコアプロセッサによる並列処理を行った場合，複数のコアが同じメモリ帯域を利用するため，コアあたりの帯域はさらに制限され，本問題はより顕著となる．

そこで，本論文では 3 次元 FDTD 法においてプロセッサの演算能力を効果的に利用するため，キャッシュメモリの利用効率を上げ，メモリ帯域による性能の低下を軽減する手法を提案する．本手法は一般に Iterative stencil computation と呼ばれる時間に関するループの中に空間に関するステンシル演算のループを持つ計算手順において，複数のタイムステップを小領域で行うことによりキャッシュのヒット率を向上させる手法を 3 次元 FDTD 法に

†1 京都大学
Kyoto University
†2 同志社大学
Doshisha University

適用したものである[2]．同手法を一般的な 3 次元 FDTD 法による実装（Naive な実装）と比較し，マルチコアプロセッサにおける並列処理においてメインメモリの帯域が十分でない場合キャッシュヒット率の改善により性能の向上がみられることを示す．

2. 3 次元 FDTD 法による電磁場シミュレーション

本章では，3 次元 FDTD 法の計算手順と一般的に用いられる実装法について述べる．

2.1 3 次元 FDTD 法の概要

3 次元 FDTD（Finite Difference Time Domain）法は，直方体メッシュに区切られた空間内に離散的に配置された未知変数である電場 $\boldsymbol{E}$，磁場 $\boldsymbol{H}$ を，与えられた初期値から時間ステップごとに陽的に解いていく電磁場数値解析手法の 1 つである．本研究では離散電磁場変数の配置に Yee のメッシュを用い，電磁場計算では Leap-flog アルゴリズムを用いる[3]．

電磁場の振舞いを表す支配方程式は，Maxwell の方程式より次のように表される．

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

$$\nabla \times \boldsymbol{H} = \epsilon \frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E}. \tag{2}$$

ここで，諸変数は $\boldsymbol{E}$：電場，$\boldsymbol{H}$：磁場，誘電率 ϵ，透磁率 μ，導電率 σ である．

FDTD 法の電磁場計算においては，以下の Leap-flog アルゴリズムが用いられる．このアルゴリズムでは，離散的な時間発展計算を電場と磁場とで半ステップずらし，また時間微分項の差分近似に中心差分を用いることで，あるタイムステップの電場から半ステップ後の磁場を求め，またその磁場からさらに半ステップ後の電場を求めるという繰返しにより，電磁場計算を行う．

式 (1)，式 (2) に対して，Leap-flog アルゴリズムを用いた時間方向の離散化は以下のように行われる．時間ステップ Δt に対して，時間方向に関して時刻 $n\Delta t$ の電場 $\boldsymbol{E}^n$，時刻 $(n+(1/2))\Delta t$ の磁場 $\boldsymbol{H}^{n+1/2}$ は

$$\frac{\boldsymbol{E}^n - \boldsymbol{E}^{n-1}}{\Delta t} = \frac{1}{\epsilon}\left(\nabla \times \boldsymbol{H}^{n-1/2}\right) - \frac{\sigma}{\epsilon}\boldsymbol{E}^{n-1/2}, \tag{3}$$

$$\frac{\boldsymbol{H}^{n+1/2} - \boldsymbol{H}^{n-1/2}}{\Delta t} = -\frac{1}{\mu}\left(\nabla \times \boldsymbol{E}^n\right), \tag{4}$$

となる．ここで $\boldsymbol{E}^{n-1/2}$ を $\left(\boldsymbol{E}^n + \boldsymbol{E}^{n-1}\right)/2$ で近似すると，離散化されたマクスウェルの方程式

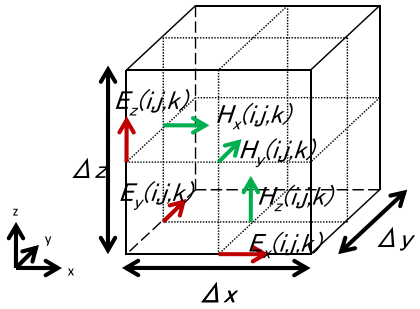


図 1　Yee のメッシュ
Fig. 1　Yee-Cell.

が得られる．

$$\boldsymbol{E}^n = \frac{1-(\sigma\Delta t/(2\epsilon))}{1+(\sigma\Delta t/(2\epsilon))}\boldsymbol{E}^{n-1} + \frac{\Delta t/\epsilon}{1+(\sigma\Delta t/(2\epsilon))}\left(\nabla \times \boldsymbol{H}^{n-1/2}\right), \tag{5}$$

$$\boldsymbol{H}^{n+1/2} = \boldsymbol{H}^{n-1/2} - \frac{\Delta t}{\mu}\left(\nabla \times \boldsymbol{E}^n\right), \tag{6}$$

ここで，$\nabla \times \boldsymbol{E}$，$\nabla \times \boldsymbol{H}$ は x 軸，y 軸，z 軸を持つ直交座標系では電場 $\boldsymbol{E}$，磁場 $\boldsymbol{H}$ の各成分 E_x，E_y，E_z，H_x，H_y，H_z を用いて以下のように表すことができる．

$$\nabla \times \boldsymbol{E} = \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}, \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}, \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y}\right), \tag{7}$$

$$\nabla \times \boldsymbol{H} = \left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z}, \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x}, \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right). \tag{8}$$

解析空間上の x 軸，y 軸，z 軸での座標を (x,y,z) と表記し，格子座標 (i,j,k) での電場を $\boldsymbol{E}(i,j,k)$，磁場を $\boldsymbol{H}(i,j,k)$ と表す．**図 1** に示すように，解析空間を離散化しセルの辺上に電場 E_x，E_y，E_z，磁場 H_x，H_y，H_z を交互に配置することより，式 (7)，式 (8) に対して中心差分公式を用いることができ，式 (5)，式 (6) は以下の差分式で表現することが可能となる．

$$
\begin{aligned}
E_x^n(i,j,k) = {} & C_e(i,j,k)E_x^{n-1}(i,j,k) \\
& + \frac{C_{er}(i,j,k)}{\Delta y}\left\{H_z^{n-1/2}(i,j,k) - H_z^{n-1/2}(i,j-1,k)\right\} \\
& + \frac{C_{er}(i,j,k)}{\Delta z}\left\{H_y^{n-1/2}(i,j,k) - H_y^{n-1/2}(i,j,k-1)\right\}, \qquad (9)
\end{aligned}
$$

$$
\begin{aligned}
E_y^n(i,j,k) = {} & C_e(i,j,k)E_y^{n-1}(i,j,k) \\
& + \frac{C_{er}(i,j,k)}{\Delta z}\left\{H_x^{n-1/2}(i,j,k) - H_x^{n-1/2}(i,j,k-1)\right\} \\
& + \frac{C_{er}(i,j,k)}{\Delta x}\left\{H_z^{n-1/2}(i,j,k) - H_z^{n-1/2}(i-1,j,k)\right\}, \qquad (10)
\end{aligned}
$$

$$
\begin{aligned}
E_z^n(i,j,k) = {} & C_e(i,j,k)E_z^{n-1}(i,j,k) \\
& + \frac{C_{er}(i,j,k)}{\Delta x}\left\{H_y^{n-1/2}(i,j,k) - H_y^{n-1/2}(i-1,j,k)\right\} \\
& + \frac{C_{er}(i,j,k)}{\Delta y}\left\{H_x^{n-1/2}(i,j,k) - H_x^{n-1/2}(i,j-1,k)\right\}, \qquad (11)
\end{aligned}
$$

$$
\begin{aligned}
H_x^{n+1/2}(i,j,k) = {} & H_x^{n-1/2}(i,j,k) \\
& + \frac{C_{hr}(i,j,k)}{\Delta y}\left\{E_z^n(i,j+1,k) - E_z^n(i,j,k)\right\} \\
& + \frac{C_{hr}(i,j,k)}{\Delta z}\left\{E_y^n(i,j,k+1) - E_y^n(i,j,k)\right\}, \qquad (12)
\end{aligned}
$$

$$
\begin{aligned}
H_y^{n+1/2}(i,j,k) = {} & H_y^{n-1/2}(i,j,k) \\
& + \frac{C_{hr}(i,j,k)}{\Delta z}\left\{E_x^n(i,j,k+1) - E_x^n(i,j,k)\right\} \\
& + \frac{C_{hr}(i,j,k)}{\Delta x}\left\{E_z^n(i+1,j,k) - E_z^n(i,j,k)\right\}, \qquad (13)
\end{aligned}
$$

$$
\begin{aligned}
H_z^{n+1/2}(i,j,k) = {} & H_z^{n-1/2}(i,j,k) \\
& + \frac{C_{hr}(i,j,k)}{\Delta x}\left\{E_y^n(i+1,j,k) - E_y^n(i,j,k)\right\} \\
& + \frac{C_{hr}(i,j,k)}{\Delta y}\left\{E_x^n(i,j+1,k) - E_x^n(i,j,k)\right\}. \qquad (14)
\end{aligned}
$$

ここで C_e，C_{er}，C_{hr} は空間座標によって決まるスカラ量であり，各格子点で定義される誘電率 ϵ，透磁率 μ，導電率 σ によって，

$$C_e(i,j,k) = \frac{1-(\sigma(i,j,k)\Delta t/(2\epsilon(i,j,k)))}{1+(\sigma(i,j,k)\Delta t/(2\epsilon(i,j,k)))}, \qquad (15)$$

$$C_{er}(i,j,k) = \frac{\Delta t/\epsilon(i,j,k)}{1+(\sigma(i,j,k)\Delta t/(2\epsilon(i,j,k)))}, \qquad (16)$$

$$C_{hr}(i,j,k) = \frac{\Delta t}{\mu(i,j,k)}, \qquad (17)$$

と定められる．

```
for(t=0;t<nt;t++){
  for(i=1;i<nx;i++){
    for(j=1;j<ny;j++){
      for(k=1;k<nz;k++){
        m=id[i][j][k];
        Ex[i][j][k] = Ce[m] * Ex[i][j][k]
                    + Cery[m] * (Hz[i][j][k] - Hz[i][j-1][k])
                    + Cerz[m] * (Hy[i][j][k] - Hy[i][j][k-1]);
        Ey[i][j][k] = Ce[m] * Ey[i][j][k]
                    + Cerz[m] * (Hx[i][j][k] - Hx[i][j][k-1])
                    + Cerx[m] * (Hz[i][j][k] - Hz[i-1][j][k]);
        Ez[i][j][k] = Ce[m] * Ez[i][j][k]
                    + Cerx[m] * (Hy[i][j][k] - Hy[i-1][j][k])
                    + Cery[m] * (Hx[i][j][k] - Hx[i][j-1][k]);
  }}}
  for(i=1;i<nx;i++){
    for(j=1;j<ny;j++){
      for(k=1;k<nz;k++){
        m=id[i][j][k];
        Hx[i][j][k] = Hx[i][j][k]
                    + Chry[m] * (Ez[i][j+1][k] - Ez[i][j][k])
                    + Chrz[m] * (Ey[i][j][k+1] - Ey[i][j][k]);
        Hy[i][j][k] = Hy[i][j][k]
                    + Chrz[m] * (Ex[i][j][k+1] - Ex[i][j][k])
                    + Chrx[m] * (Ez[i+1][j][k] - Ez[i][j][k]);
        Hz[i][j][k] = Hz[i][j][k]
                    + Chrx[m] * (Ey[i+1][j][k] - Ey[i][j][k])
                    + Chry[m] * (Ex[i][j+1][k] - Ex[i][j][k]);
  }}}
}
```

図 2　3 次元 FDTD 法の Naive な実装法におけるループ構造と電磁場計算
Fig. 2　Loop structure of three-dimensional FDTD kernel based on naive implementation method.

2.2　一般的な 3 次元 FDTD 法の実装

前節で述べたように，3 次元 FDTD 法では，空間上で離散化された電場と磁場を交互に時間発展的に更新する．そこで，一般的には以降で Naive な実装法と呼ぶ以下のような方法で実装される．すなわち，電場と磁場の 3 方向成分をそれぞれ 3 次元配列により表し，時

間発展に関するループを最外ループとして，その内側に電場計算に関する 3 重ループ，磁場計算に関する 3 重ループを記述する．また本論文では，解析空間は複数の媒質により構成されるものとし，式 (9)〜(14) のスカラ係数を媒質ごとにあらかじめ計算して配列に格納しておき，それを格子点の媒質の種類を定める整数配列 $id(i,j,k)$ を介して参照するものとしている．すなわちある格子点 (i,j,k) の媒質 m は $m = id(i,j,k)$ で与えられ，その格子に関するスカラ係数は

$$C_e(i,j,k) = C_e(m),$$
$$C_{er}(i,j,k) = C_{er}(m),$$
$$C_{hr}(i,j,k) = C_{hr}(m),$$
$$C_{erx}(m) = C_{er}(m)/\Delta x, \quad C_{ery}(m) = C_{er}(m)/\Delta y, \quad C_{erz}(m) = C_{er}(m)/\Delta z,$$
$$C_{hrx}(m) = C_{hr}(m)/\Delta x, \quad C_{hry}(m) = C_{hr}(m)/\Delta y, \quad C_{hrz}(m) = C_{hr}(m)/\Delta z,$$

と与えられるものとしている．

図 2 に 3 次元 FDTD 法の Naive な実装法におけるループ構造と電磁場計算の例を示す．図 2 に示されるように，Naive な実装法では，1 格子点・タイムステップあたりの演算数は 39 であるのに対し，データのロード/ストア回数は媒質に関するデータを除いても 30 となる．したがって，解析領域が十分に大きい場合には，メモリの帯域に計算が律速される．

3. キャッシュメモリの有効利用による 3 次元 FDTD 法の性能改善手法

前節で述べたように，3 次元 FDTD 法の実装においては解析領域が十分に大きい場合メモリの帯域に計算が律速される．

そこで，まず空間に関するループである電場・磁場の更新を解析領域を分割した小領域（タイル）を単位として行うことで，配列要素の参照間隔を短くすることで時間的局所性を向上させ，要求メモリ帯域を低減させる手法が考えられる．本手法は空間ループのみに関するタイリング（Tiling）手法であることから以降で Tiling(S-only) と表記する．Tiling(S-only) はループ内の未知変数の更新順序を変更するのみで適用可能であるため大きなオーバヘッドを生じないという長所がある．しかしながら，3 次元 FDTD 法では空間に関するループの外側に時間に関するループが存在するため，タイル上での計算を複数タイムステップ行うことでさらに性能改善を図れる可能性がある．そこで，本論文では以降で Tiling(ST) と表記する複数タイムステップの計算をタイル上で行い，タイムステップ間でタイルを再利用することで，キャッシュメモリのヒット率の向上を図る性能改善手法を提案する．

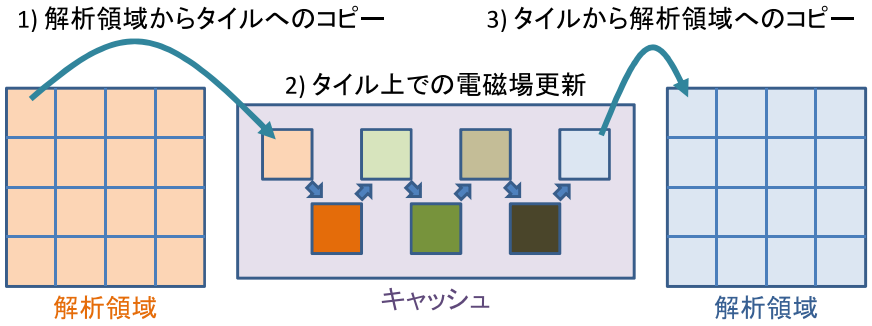


図 3　Tiling(ST) の概要
Fig. 3　Outline of Tiling(ST).

3.1 提 案 手 法

Tiling(ST) を適用した 3 次元 FDTD 法では，解析領域をタイルと呼ぶ小領域に分割し，1) 解析領域からタイルへのコピー，2) タイル上での複数ステップの電磁場の更新，3) 更新後の値の元の全体領域へのコピーという手順を各タイルに対して行う．**図 3** に提案手法の概要を示す．本手法において，タイルがキャッシュメモリ（主に 2，3 次キャッシュ）に収まるサイズの場合，上記 2) のタイル上での電磁場更新操作においてすべての配列要素の参照・ストアがキャッシュメモリにヒットするため，キャッシュヒット率の向上が期待できる．

以下に提案手法の詳細について述べる．3 次元の解析領域に対する Tiling(ST) では，**図 4** のように 1 つのタイルの 1 辺に含まれる要素（格子点）の数を n_t とし，大きさ $n_t \times n_t \times n_t$ の立方体状のタイルを得る．また，タイル内で 1 度に行う電磁場更新のステップ数を s_t とする．このとき電場 $\boldsymbol{E}$ と磁場 $\boldsymbol{H}$ は交互にそれぞれ s_t 回更新される．このタイルに対する 1 回の $\boldsymbol{E}$ および磁場 $\boldsymbol{H}$ の更新には，式 (9) から式 (14) より，それぞれ解析領域上で隣接する格子点の $\boldsymbol{H}$，$\boldsymbol{E}$ が必要であることが分かる．すなわち，電場 $\boldsymbol{E}(i,j,k)$ の更新のためには磁場 $\boldsymbol{H}(i-1,j,k)$，$\boldsymbol{H}(i,j-1,k)$，$\boldsymbol{H}(i,j,k-1)$ が，磁場 $\boldsymbol{H}(i,j,k)$ の更新のためには電場 $\boldsymbol{E}(i+1,j,k)$，$\boldsymbol{E}(i,j+1,k)$，$\boldsymbol{E}(i,j,k+1)$ が必要となる．したがって，タイルに含まれる電場と磁場の更新のためには，1 タイムステップあたりタイルの各面の外側 1 格子分のデータが必要となり，提案手法では，タイル上の操作が s_t ステップ繰り返されることを考慮すると，タイルの更新のために必要となる領域のサイズは $(n_t+2s_t)\times(n_t+2s_t)\times(n_t+2s_t)$ となる．すなわち，上記 1) の解析領域からタイルへのコピー操作の対象は，タイルを 6 方向に s_t 格子点だけ拡大した領域とする必要がある．さらに，あるタイルの計算のために必

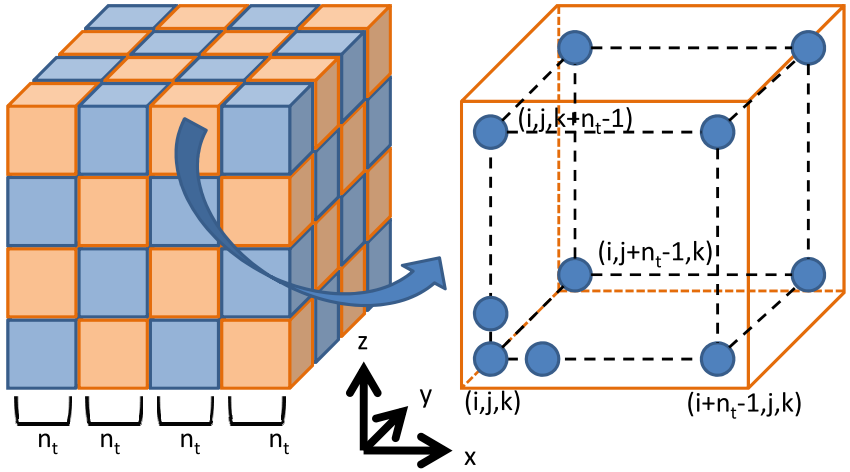


図 4　解析領域の分割
Fig. 4　Division of analytical domain.

要なタイル外の領域は，相互に他のタイルに含まれているため，タイル内の計算で本来の $\boldsymbol{E}$，$\boldsymbol{H}$ を直接更新すると，他のタイル上での計算で必要な格子点上の値が失われてしまう．したがって，上記の手順 1) から 3) で示しているように，タイルを構成する配列を解析領域全体に対する電場・磁場などの配列と別に用意し，タイルの計算に先立って解析領域から値をコピーする必要がある．また，タイル上での計算結果をそのまま解析領域全体の配列に書き戻した場合，他のタイルの辺縁の値を更新してしまい，それらのタイルの計算に必要な値が失われてしまう．そこで，本手法では，解析領域の未知変数である電場・磁場の配列を各々について 2 つ用意し，これを交互に更新することとする．

以上をふまえて，具体的には以下のような実装を行う．なお，以下の記述において可読性のために「解析領域」あるいは「領域」という表現を使用するが，これはプログラム上ではいずれも電場・磁場に対応する配列である．

Step 1.　解析領域全体と同じ大きさ n の 2 つの解析領域 S，R と 1 辺のサイズが n_t+2s_t の領域 T を用意し，S に初期値を入力する．

Step 2.　領域 S を大きさ $n_t \times n_t \times n_t$ のタイルに分割する．

Step 3.　領域 S の未計算のタイルを選び，タイルとその外側の格子点 s_t 個分を合わせた大きさ $(n_t+2s_t)\times(n_t+2s_t)\times(n_t+2s_t)$ の領域を T へコピーする（ただし，解析領域の境界周辺部では解析領域のサイズに合わせてタイルサイズを調整する場合がある）．

Step 4.　タイル内で s_t タイムステップの電磁場更新を行う．ここで k 回目（$1 \le k \le s_t$）の電磁場更新では，タイルの辺縁を $s_t - k$ だけ拡大した領域を更新対象とする．

Step 5.　領域 T から領域 R へコピーする．このときタイルの位置が領域 S と R で同じ位置にくるようにする．

Step 6.　領域 S に未計算のタイルが残っていれば **Step 3** に戻る．

Step 7.　計算を続ける場合，S と R の名称を交換し **Step 2** に戻る．

以上の手順から明らかなように，本手法では Naive な実装と比べて約 2 倍のメモリ領域が必要となる．しかし，s_t タイムステップの計算が行われる間，領域 S が計算開始時点の値を保持することを利用して，Tiling(ST) においても領域分割によるスレッド並列化を容易に実現できるという利点がある．

3.2　提案手法により期待できる効果

Naive な実装では 2 章で述べたように計算時間がメモリ帯域により律速されるのに対し，Tiling(ST) を用いるとほとんどのロード/ストアがキャッシュヒットするためメモリ帯域による制約が大幅に緩和される．一方，Tiling(ST) を行うと通常の手法と比べて計算量およびデータのロード/ストア量が増加する．これは上記の **Step 4** に示すように，s_t タイムステップ後のタイル内の電磁場を更新するためには，タイルの大きさ以上の格子点の計算が必要となるためである．そこで，Tiling(ST) による計算量（格子点更新回数）の増加と計算速度の向上を見積もり，両者のトレードオフについて検討する．

まず，n_t^3 の領域に対して s_t タイムステップの計算を行う際に更新する格子点数を電場と磁場とで個別に求め，それらの和を計算量の指標として Naive な実装と Tiling(ST) の比較を行う．Naive な実装では電場・磁場ともタイムステップあたりの更新格子点数は明らかに n_t^3 であるので，計算量の指標 F_n は下式となる．

$$F_n = 2s_t n_t^3. \tag{18}$$

一方 Tiling(ST) を用いると，1 タイムステップあたりに更新する格子点数は，電場については $(n_t+2s_t-1)^3$, $(n_t+2s_t-3)^3$, …, $(n_t+1)^3$ と減少し，磁場については $(n_t+2s_t-2)^3$, $(n_t+2s_t-4)^3$, …, n_t^3 と減少する．したがって計算量の指標 F_t は

$$F_t = \sum_{k=1}^{2s_t}(n_t+k-1)^3, \tag{19}$$

と表せる．Tiling(ST) を用いた手法における計算する格子点数の増加率 F_t/F_n は **図 5** に

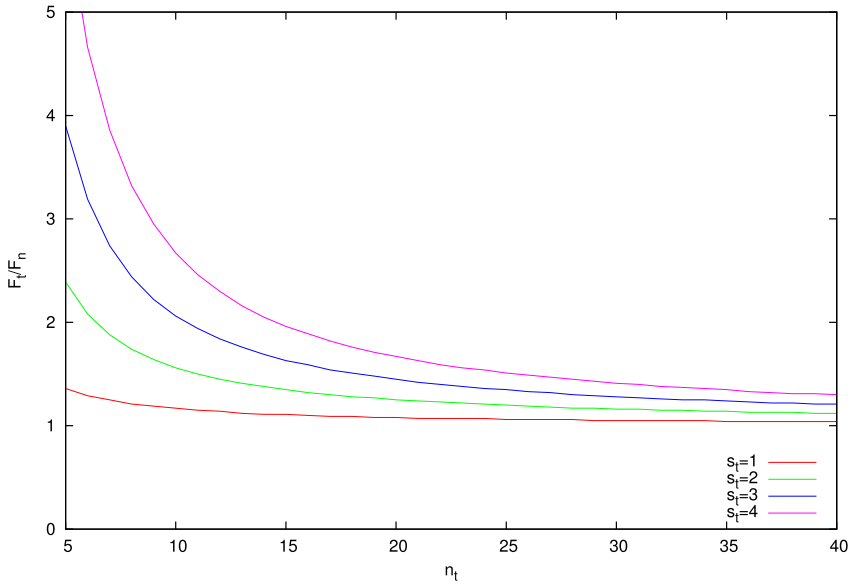


図 5　Naive な実装を基準とした Tiling(ST) の計算量（計算される格子点数）
Fig. 5　Ratio of computational costs of Tiling(ST) compared with Naive implementation.

示すように，タイルサイズが小さくタイムステップが大きい場合には Naive な実装法に対する比率が非常に大きくなるが，タイルサイズが大きくなるに従い減少し 1 に漸近する．

一方，Tiling(ST) を行うと単位時間あたりに更新できる格子点数，すなわち計算性能の向上が見込める．Naive な実装における計算性能の逆数，すなわち 1 格子点を更新するために要する「単位計算時間」を，領域がキャッシュに収まる場合（case-NC）と収まらない場合（case-NN）でそれぞれ τ_c，τ_n とする．一方，Tiling(ST) を用いる場合の s_t ステップの計算では，タイル領域 T の配列アクセスがすべてキャッシュヒットすることが期待できる．ここで 3.1 節の **Step 3** の解析領域 S からタイル領域 T へのコピー，および **Step 5** の T から解析領域 R へのコピーは，電磁場配列の更新により代用することができるため，Tiling(ST) での単位計算時間 τ_t を見積もるための電磁場配列の参照・更新とキャッシュとの関係は，以下のようになる．

(1)　第 1 ステップの電場計算では，S の電場・磁場配列を参照しながら T の電場配列を更新する．電場配列の更新はキャッシュヒットすることが期待できるが，case-NN でも更新アクセスは通常キャッシュヒットするため，単位計算時間は τ_n と近似できる．

(2)　第 1 ステップの磁場計算では，S の磁場配列と T の電場配列を参照しながら T の磁場配列を更新する．ここで S の磁場配列は第 1 ステップの電場計算で参照されているためキャッシュヒットすることが期待でき，単位計算時間は case-NC と同様に τ_c と近似できる．

(3)　第 2 ステップから第 $(s_t - 1)$ ステップまでは，T の配列の参照・更新のみが行われるため，単位計算時間は τ_c と見積もられる．

(4)　第 s_t ステップの電場計算では，T の電場・磁場配列を参照しながら R の電場配列を更新する[⋆1]．ここで，ストアミスのコストはストアバッファなどの効果で完全に隠蔽できるとすると，case-NC と同様に単位計算時間を τ_c と見積もることができる．

(5)　第 s_t ステップの電場計算では，T の磁場配列と R の電場配列を参照しながら R の磁場配列を更新する[⋆2]．ここで R の電場配列は第 s_t ステップの電場計算で更新されているためキャッシュヒットすることが期待でき，単位計算時間は第 s_t ステップの電場計算と同様に τ_c と見積もられる．

上記をまとめると，τ_t は以下のように近似できる．

$$\tau_t = \frac{(\tau_n + \tau_c) + 2(s_t - 2)\tau_c + 2\tau_c}{2s_t} = \frac{\tau_n + (2s_t - 1)\tau_c}{2s_t}. \tag{20}$$

以上から，n_t^3 の領域に対する s_t タイムステップの計算に要する時間を，Naive な実装と Tiling(ST) を用いた実装の各々について t_n，t_t とすると，

$$t_n = F_n\tau_n, \qquad t_t = F_t\tau_t, \tag{21}$$

となり，Tiling(ST) による計算時間の短縮率 $r_t = t_t/t_n$ は

$$r_t = \frac{t_t}{t_n} = \frac{F_t\tau_t}{F_n\tau_n}, \tag{22}$$

と表せる．実際の Tiling(ST) を用いた手法の実装においてはオーバヘッドが存在し，またストアミスのコストを完全に隠蔽できるとは限らないため，式 (22) により与えられる r_t は，Tiling(ST) による計算時間の短縮率の下限を与えると考えることができる．

⋆1 厳密にはタイルの辺縁領域については T の電場配列を更新する．
⋆2 厳密にはタイルの辺縁領域については T の電場配列を参照する．

4. 性能評価

4.1 使用計算機環境と解析モデル

提案手法の有効性の評価のため，数値実験を行った．実験に使用した計算機は，京都大学学術情報メディアセンターの T2K オープンスーパコンピュータのノードである富士通 HX600[4] である．

HX600 は 4 個の AMD 社製クアッドコア Opteron(8356) プロセッサと 32 GB（DDR2-667）のメモリを有している．本プロセッサの性能は，理論ピーク演算性能が 36.8 GFlops，コアあたり 9.2 GFlops，キャッシュメモリ容量は L2 キャッシュがコアあたり 512 KB，L3 キャッシュがプロセッサあたり 2 MB となっている．また，メインメモリの帯域はプロセッサ（ソケット）あたり 10.6 GB/s である．

本数値実験では，金属壁に囲まれた立方体形状の解析領域を使用する．金属壁内では電場 E と磁場 H はともに 0 となるとし，演算は行わない．このため，解析領域中の 1 方向の格子点数を n と表した場合，金属壁を含む領域全体は $(n+2) \times (n+2) \times (n+2)$ の格子に分割され，このうち計算の対象となる格子は $n \times n \times n$ となる．実応用を考慮して，本論文で主に対象とするのは 1 方向の格子点数 n が 200〜250 程度の場合である．

また，本数値実験ではマルチコアプロセッサによるスレッド並列処理を利用する．HX600 の 1 ノード上の 1 つのプロセッサを用い，最大 4 スレッドによる並列実行を行う．2 章で述べたように FDTD 法は陽的な時間発展型の計算であるため，領域分割により容易に並列化可能である．スレッド並列処理を行う場合，本実験では解析領域を x 方向にスレッド数で分割するものとする．なお，本実験では各スレッドを 1 つのプロセッサ（ソケット）上に集中して配置し，解析プログラム中の各種配列は，使用するプロセッサ（ソケット）に接続されたメモリ上に配置するものとする．

プログラムは C 言語を用いて記述し，富士通製 C コンパイラ（ver.3.0）により最適化オプション -KOMP -Kfast,noprefetch,restp=all を指定してコンパイルを行った．

4.2 Naive な実装の性能評価

提案手法の性能評価の予備実験として，Naive な実装による 3 次元 FDTD 法の性能調査を行った．上記の計算機環境で 1 格子点・タイムステップあたりの計算時間を測定した．**図 6** にスレッド並列を用いた Naive な実装の測定結果を示す．横軸が領域の 1 方向の格子点数 n，縦軸が 1 格子点・タイムステップあたりの計算時間を示している．

図 6 より，いずれのスレッド数を用いた場合においても，格子サイズの違いによって 1 格子点・タイムステップあたりの計算時間 t_n が大きく変化することが確認できる．特に，領域のサイズ n がおよそ 40 のあたりで，t_n に大きな変化が表れる．これは，2.2 節で述べたとおり，格子サイズが十分に小さな場合は配列データがすべて 2 次，3 次キャッシュメモリ上に存在し，高速に計算が可能となるのに対し，上記のサイズを超えると配列データはキャッシュメモリだけでは保持できなくなり，メインメモリへのアクセスが生じ，実効演算性能の低下につながったためと考えられる．本実験で使用した Opteron プロセッサがキャッシュメモリ上に解析領域の全格子データを保持できるおよそのサイズ n は，利用できる L2，L3 キャッシュの量より，逐次実行の場合約 36，4 スレッド並列の場合約 42 となり，$n = 40$ 付近での実効演算性能の劣化が配列データのキャッシュあふれによるものであることが分かる．

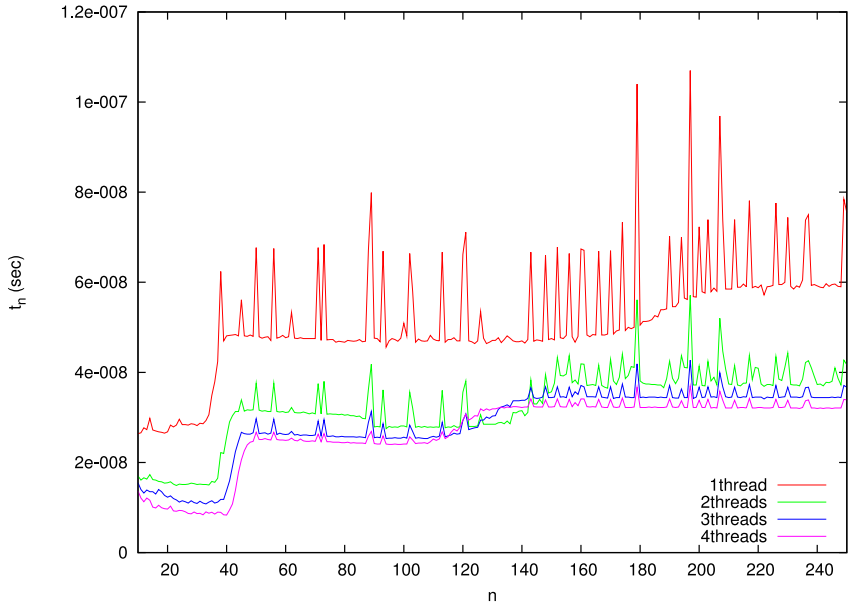


図 6　Naive な実装における 1 格子点・タイムステップあたりの計算時間
Fig. 6　Computation time for one grid point and one time step of Naive FDTD method.

次に，Naive な実装法における台数効果について調べる．図 6 より，実応用上重要な n が 200 以上の場合，スレッド数が 2 の場合には台数効果が得られるが，スレッド数を 3，4 と増やしてもほとんど台数効果は得られないことが分かる．これは，2 スレッドを使用した

計算においてすでにメモリ帯域のほとんどを使いきっており，それ以上スレッド数を増加させても台数効果が得られない状況を示していると考えられる．また，図 6 において，Naive な実装では，特定の格子サイズにおいて大きな実効演算性能の劣化が観測される．これは，キャッシュのスラッシングによるものと考えられ，電場あるいは磁場，もしくはその両方に関する配列を 1 つの構造体としてまとめることで回避できることが著者らの数値実験[5] により確認されている．しかしながら，構造体を利用した場合，計算に使用しないデータがキャッシュ上に読み込まれる影響により，スラッシングを起こさない格子サイズの場合には，実効演算性能は Naive な実装法よりも低下する．そこで，以降の提案手法との比較評価では，構造体を用いた実装法ではなく，すべての配列を個別に持つ Naive な実装法を比較対象とする．

4.3　提案手法［Tiling(ST)］の性能評価

Naive な実装と提案手法［Tiling(ST)］の各々を用いた場合について，格子サイズを変化させ，1 格子サイズあたり 120 タイムステップの計算を行い，両者の 1 格子点・タイムステップあたりの計算時間を求めて比較する．Tiling(ST) を用いた場合では，タイルサイズ n_t を 5〜25 の範囲，タイムステップ s_t を 1〜4 の範囲で変化させ，その性能を評価する．1 格子点・タイムステップあたりの計算時間は，Naive な実装と Tiling(ST) を用いた場合ともに，1 方向の格子サイズ n を 200〜250 の範囲で変化させた場合の平均値とする．

図 7，**図 8**，**図 9**，**図 10** が，逐次実行および 2 から 4 の各スレッド数で，Tiling(ST) の時間ステップ数 s_t の違いによる 1 格子点・タイムステップあたりの計算時間の比較を行ったものである．グラフの横軸はタイルのサイズ n_t である．

4.3.1　Tiling(ST) の有効性

図 7 より，逐次実行の場合はタイルのサイズ n_t やステップ数 s_t にかかわらず Naive な実装のほうが Tiling(ST) を用いた場合よりも計算時間が短い．逐次実行を行った場合，コアあたりのメモリの帯域幅が並列実行時と比べて広く，図 6 にみられるように Tiling(ST) による性能向上の源泉となるキャッシュ上で計算を行うことができる場合とできない場合の 1 格子点あたり計算速度の差が小さい．そのため，図 5 に示した計算量の増加や Tiling(ST) にともなうオーバヘッドによる影響により，Naive な場合と比べて速度向上が得られなかったと考えられる．これに対し，複数のコアを使用したスレッド並列処理の場合，図 8，図 9，図 10 に示すように，適切なタイルサイズ n_t とステップ数 s_t を用いたとき，スレッド数の増加とともに提案手法の有効性が高まり，スレッド数が 3 以上の場合は Naive な実装を上回る性能を発揮している．これは，スレッド数が 2 以上の場合，Tiling(ST) による計算量

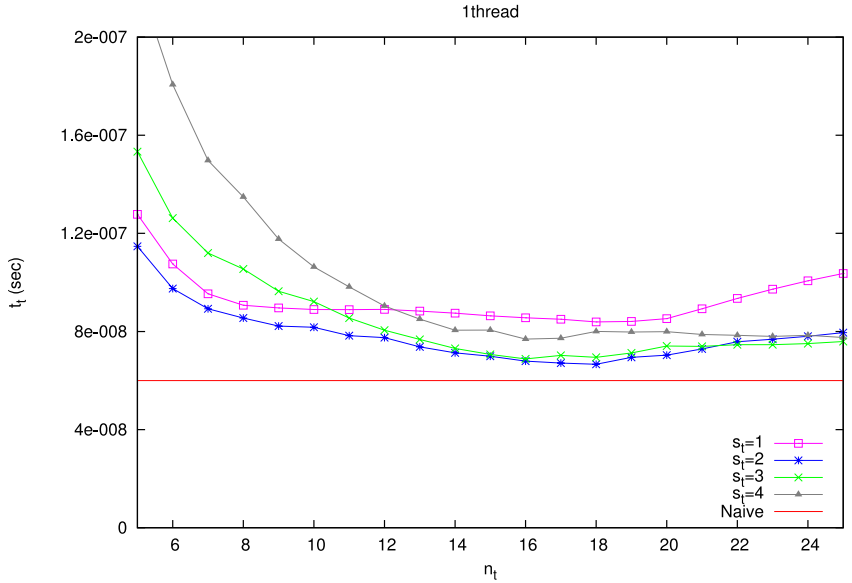


図 7　逐次実行における Tiling(ST) の性能
Fig. 7　Performance of Tiling(ST) (serial).

の増加率は変わらないのに対して，スレッドあたりのメインメモリの帯域は減少していくため，スレッド数の増加にともない Tiling(ST) によるメインメモリへのアクセス量の減少の効果がより効果的に働いたためと考えられる．

図 10 より，Tiling(ST) による方法は 4 スレッド並列で実行された場合，Naive な実装と比べ，実行時間が最大で 33%短縮される（性能は約 1.5 倍向上する）ことが確認できる．また，このとき式 (22) により算出される計算時間の短縮率は，図 6 より $\tau_n = 3.9\tau_c$ と見積もる[⋆1]と $r_t = 0.62$ となるのに対し，実測された短縮率は $n_t = 13$，$s_t = 2$ のとき最小で，0.67 となった．すなわち，式 (22) で与えられる最大期待できる短縮率の約 93%の計算時間の短縮を実現している．

図 11 に空間に関するループのみをキャッシュブロッキングする Tiling(S-only) と

⋆1 τ_n は $n = 200$〜250 の場合の平均値から $\tau_n = 3.25 \times 10^{-8}$（sec）とし，$\tau_c$ は性能が最も良くなる格子サイズ $n = 40$ の場合を用い $\tau_c = 8.33 \times 10^{-9}$（sec）として見積もった．

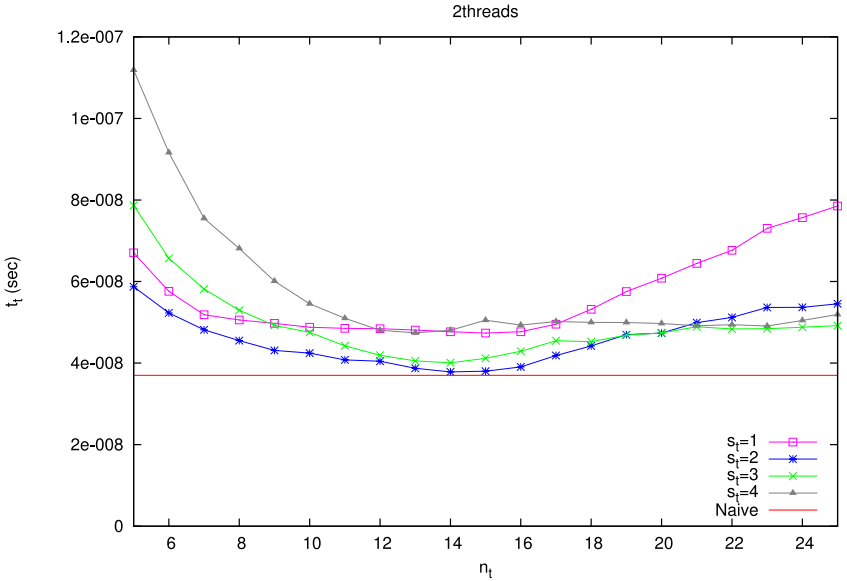


図 8　2 スレッド並列実行における Tiling(ST) の性能
Fig. 8　Performance of Tiling(ST) (2 threads).

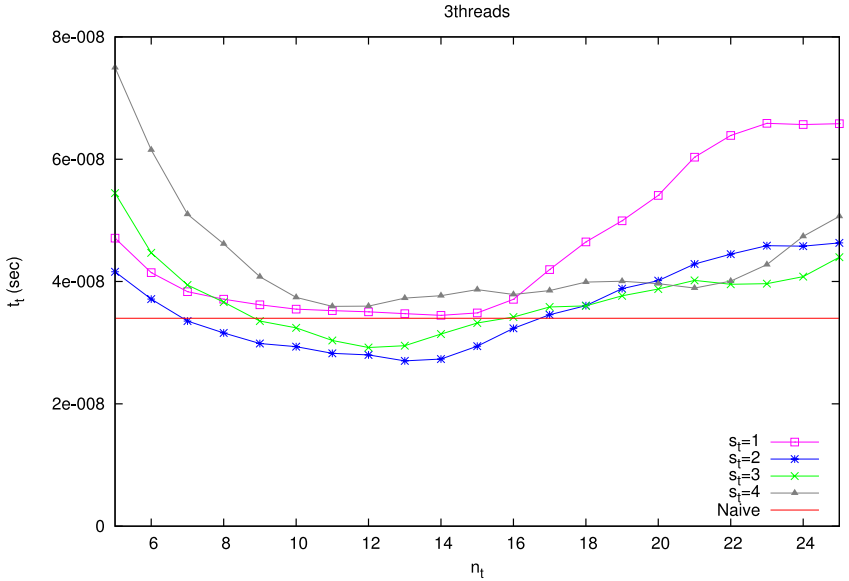


図 9　3 スレッド並列実行における Tiling(ST) の性能
Fig. 9　Performance of Tiling(ST) (3 threads).

Tiling(ST) の性能比較の結果を示す．図 11 は Tiling(S-only) と Tiling(ST) を 4 スレッド並列で実行し，タイルのサイズ n_t に対する 1 格子点・タイムステップあたりの計算時間の比較を行ったものである．ここで，Tiling(ST) の計算時間は，s_t が 1 から 4 の場合の最小値とした．また，Tiling(S-only) は Tiling(ST) と同様に解析領域をサイズ n_t の立法形状の小領域に区切り，この小領域ごとに更新する手法をとった．このとき，Tiling(ST) は，Tiling(S-only) と比べ Naive な実装を超える性能を発揮することができるタイルサイズ n_t の範囲は狭いが，互いに最大の性能を発揮する n_t においては Tiling(S-only) 以上の性能を発揮した．Tiling(ST) が Tiling(S-only) と比べ狭い範囲の n_t でしか性能を発揮できないのは，3 章で述べたように，Tiling(S-only) は大きなオーバヘッドを生じず高速化できるのに対して，Tiling(ST) は計算量の増加とキャッシュヒット率の向上のトレードオフにより，タイルのサイズに関してより大きな制約を受けるためである．一方で，Naive な実装に対する Tiling(S-only) の短縮率は $n_t = 20$ のとき最小で 0.79 であった．Tiling(ST) の短縮率は 0.67 であり，Naive な実装に対する性能改善効果は Tiling(S-only) と比較して約 12%高く，より効果的であることが確認された．

図 12 に Naive な実装における逐次実行を基準とした，Naive な実装と Tiling(ST) を用いた実装の台数効果を示す．ここで，Tiling(ST) による結果は，各スレッド数に対して，最も高い性能が得られた n_t, s_t（値は図中に記す）による場合を示す．図より，Naive な実装においてはスレッド並列処理を行った場合 3 スレッド並列以上では性能の向上が得られないが，Tiling(ST) を用いた実装を行うことにより，3 スレッド並列以上でも線形に近い性能向上が得られることが分かる．

4.3.2　Tiling(ST) におけるタイルサイズとタイムステップの影響

図 7 から図 10 より，いずれのスレッド（コア）数における数値実験においても，1 格子点・タイムステップあたりの計算時間はある n_t および s_t で最小値をとることが分かる．今回の実験で最小の 1 格子点・タイムステップあたり計算時間を与えるのは，スレッド数が 1 のとき $n_t = 18$，スレッド数が 2 のとき $n_t = 15$，スレッド数が 3 のとき $n_t = 13$，スレッド数が 4 のとき $n_t = 13$ で，いずれも Tiling(ST) のステップ数 $s_t = 2$ の場合である．こ

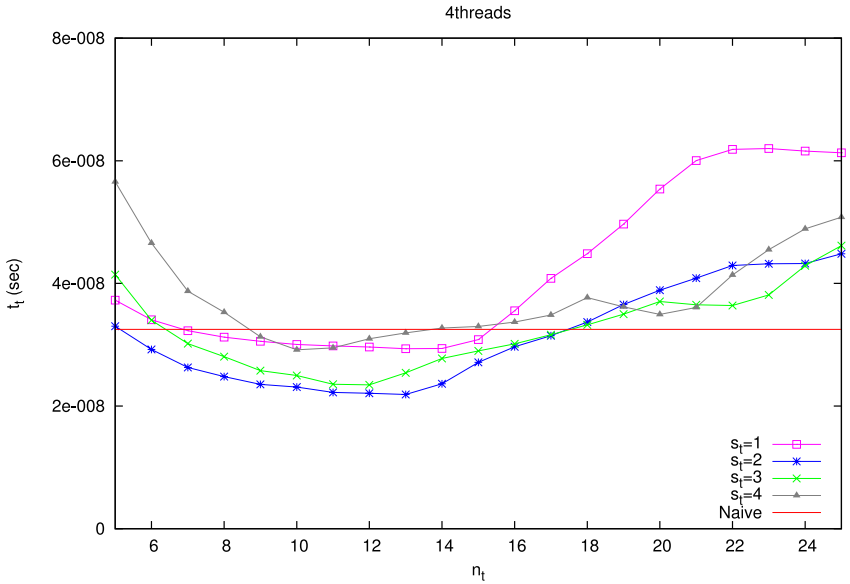


図 10　4 スレッド並列実行における Tiling(ST) の性能
Fig. 10　Performance of Tiling(ST) (4 threads).

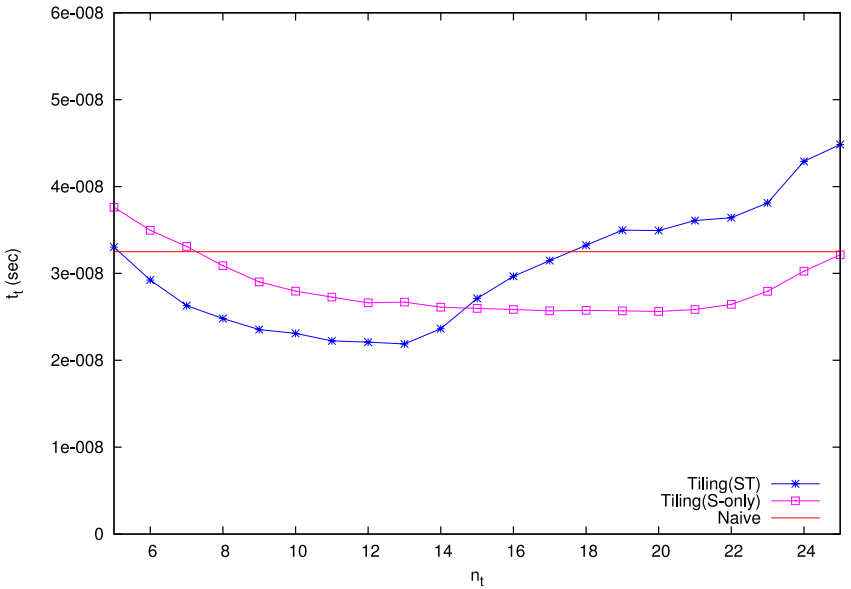


図 11　Tiling(ST) と Tiling(S-only) の性能比較
Fig. 11　Performance comparison of Tiling(ST) and Tiling(S-only).

のとき，タイルサイズとキャッシュ容量の関係は**表 1** のようになる．

表 1 より，Tiling(ST) において最適なタイル全体のサイズ $n_t + 2s_t$ はスレッド数にかかわらず，1 コアが利用できるキャッシュサイズのおよそ 1/4，25%前後に相当することが分かる．

以下，Tiling(ST) による手法におけるタイルサイズとタイムステップ数の影響について考察する．まず，タイムステップ数を固定した場合，図 7 から図 10 より，1 格子点・タイムステップあたりの計算時間はタイルサイズに対して，下に凸形のグラフを描く．

ここで，タイル全体のサイズ $n'_t = n_t + 2s_t$ とし，上記の最適なサイズを $n_t'^{\min}$ とする．まず $n'_t < n_t'^{\min}$ では，タイルサイズは十分小さく完全にキャッシュに収納可能であると仮定する．この場合，計算時間比は式 (18) と式 (19) の比，すなわち計算量の比に比例すると考えられる．一方，図 5 にみられるように，n_t が減少するのに従って，計算量の比はより多くなり，s_t が大きい場合ほどその影響が顕著となる．したがって，タイルサイズが十分に小さく，計算に必要な配列がキャッシュにすべて収納される範囲では，タイルサイズを小さくすることは実効演算性能の改善に寄与することはなく，なるべく大きいタイルを用いる方が有利となる．次に，一方 $n'_t > n_t'^{\min}$ では，Tiling(ST) の有効性の源泉であるキャッシュメモリの活用効果がタイルサイズの増加にともない低下し，計算時間が増加するものと考えられる．すなわち，Tiling(ST) を用いた計算ではタイル領域 T へのアクセスに加えて解析領域 S，R へのアクセスも必要なため，T がキャッシュサイズの 1/3 を超過すると T がキャッシュメモリに常駐する状況を保てなくなる．逆に T がキャッシュサイズの 1/3 以下であれば，T のキャッシュメモリへの常駐状態が保たれる．以上より，T がコアの利用可能なキャッシュサイズのおよそ 1/3 を占めるとき，最もキャッシュを有効に活用できると考えられる．一方，実際の実行においては Tiling(ST) に関連した配列のみがキャッシュ容量のすべてを占めることは困難であり，かつ T や S，R の物理アドレス空間が必ずしも連続とならないために生じるキャッシュブロックの競合のために，実測ではタイルサイズが約 25%を超えると競合性のキャッシュミスが発生し，実効演算性能が低下したと考えられる．

次にタイムステップ数 s_t について考える．上記の議論から，タイル全体のサイズ n'_t は

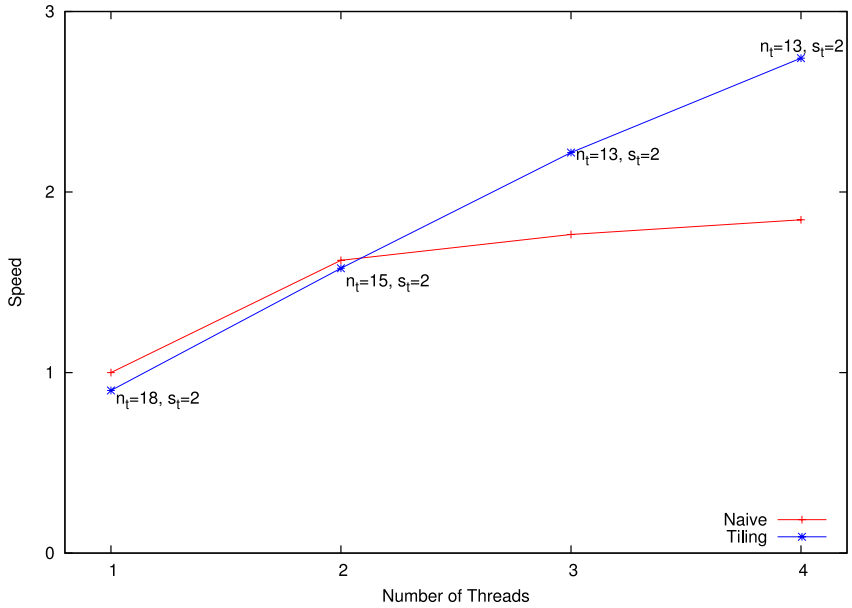


図 12　Naive な実装と Tiling(ST) を用いた実装の台数効果の比較
Fig. 12　Comparison of speed up ratio of Naive and Tiling(ST) implementations.

表 1　タイルがキャッシュに占める割合
Table 1　Proportion of tile to the cash.

threads	cache/thread（kB）	tile size（kB）	%
1thread ($n_t = 18, s_t = 2$)	2,512	596	23.7
2threads ($n_t = 15, s_t = 2$)	1,512	384	25.4
3threads ($n_t = 13, s_t = 2$)	1,195	275	23.0
4threads ($n_t = 13, s_t = 2$)	1,024	275	26.9

S，R へのアクセスを考慮したうえでタイルに関する計算においてメインメモリへのアクセスを起こさない最大のタイルサイズが最も有効であると考えられる．そこで，n'_t を一定とした場合，n_t および s_t の組合せの影響について考える．タイムステップ数 s_t が増加した場合，式 (20) より，S，R と T との間のデータコピーが T 上での計算に比べて相対的に低下するため，タイル上の計算に関する実効演算性能は向上することが期待できる．一方，$n_t + 2s_t$ を一定としているため n_t が減少し，計算量の比が増加する．したがって，これらはトレードオフの関係となり，一定の n'_t に対する最適な s_t は一概にいえない．ただし，現状の一般のプロセッサのキャッシュサイズから適切な n'_t は 10 程度となっており，図 5 によると s_t を増加させることにともなう計算量の比の増加が依然として大きい領域となっている．このような理由から，今回の数値実験では，スレッド数にかかわらず，$s_t = 2$ の場合が最速となった．

5. 関 連 研 究

一般に，大規模な配列を含む演算では，メインメモリへのアクセスが演算性能を低下させる場合がある．そこで，配列をブロック化し，各ブロックをキャッシュ上で局所的に計算することにより，メインメモリへのアクセスを低減させ，計算の高速化を行う手法が知られ，文献 6) や文献 7) では，行列積や LU 分解など，各配列要素の再利用性が高い 2 次元配列を対象としたキャッシュブロッキングについて解説している．これらの行列計算では各行列要素の再利用性が高く，たとえば $n \times n$ の行列積では 1 つの行列要素が n 回利用されることとなる．このような演算では，キャッシュのヒット率の向上による大きな効果が期待できる．

本研究で対象とした 3 次元 FDTD 法の計算カーネルは Iterative stencil computation（反復型ステンシル計算）の一種と考えることができる．Iterative stencil computation は一般に演算量と比べてデータのロード/ストア量が多く，対象とする問題が十分に大きい場合にはメインメモリの帯域に律速される．そこで，上記の行列計算と同様に，解析領域のブロック化によってキャッシュのヒット率を向上させることによる計算の高速化が期待できる．しかしながら，Iterative stencil computation は一般に，外側の時間に関するループと内側の空間に関するループによる二重ループにより構成されるが，空間に関するループ内で配列の各要素の再利用性が低いため，内側のループのみのタイリングによる性能向上は十分ではない場合がある．この場合，冗長に計算を行ったり，バッファ領域を設けたりすることで，外側の時間に関するループの複数ステップの計算をまとめ，配列要素の再利用性を高めることにより，さらなる性能向上について試みることができる．

Iterative stencil computation における計算速度の向上に関する初期的な研究報告として，文献 2) では，解析領域をデータがキャッシュに収まるサイズの小領域（タイル）で分割し，各小領域で複数の（時間に関する）反復を行うことにより，キャッシュのヒット率を向上させることを可能としている．さらに，同様の方法に基づく Iterative stencil computation の高速化に関する最近の研究成果として以下の文献があげられる．文献 8) では，大容量キャッ

シュやハードウェアプリフェッチを持つプロセッサ上での性能モデルを構築し，Power3 や UltraSparc2 を含む数種のプロセッサ上で性能評価を行っている．文献 9) では，階層的な構造を持つキャッシュメモリを対象に，タイルを再帰的に分割することにより，すべてのキャッシュレベルにおいてキャッシュのヒット率を向上させる方法について報告している．文献 10) では，スレッド並列処理を利用する場合に関する研究を行い，負荷分散を考慮した各タイルのスレッド割当て法を提案し，Opteron 2218 および Xeon X5482 による性能評価を行っている．

上記の既存研究は主に外側の時間に関するループの内側に 1 つの空間に関するループを持つ場合を対象としており，時間に関するループの内側に 2 つの空間に関するループを持つ 3 次元 FDTD 法に関する報告はなされていない．ただし，文献 10) では，2 次元 FDTD 法を対象に，電場と磁場に関する 2 つのループを 1 つのループに統合し[11]，Time-Skewing と呼ばれる手法を適用した例が報告されており，本手法を 3 次元 FDTD 法に応用することは可能である．しかし，同手法は冗長な計算を必要としない反面，タイル境界上の未知変数に関する複数タイムステップのデータ保持と特別な処理が必要となる．このタイル境界の処理オーバヘッドは，2 次元 FDTD 法ではタイルサイズに対して境界上の未知変数が占める割合が小さいため，ほとんど問題とはならない．しかし，本研究で対象とする 3 次元 FDTD 法においては，タイル全体に対しての境界面上の未知変数の占める割合が 2 次元 FDTD 法と比べて増大するために，その影響がより顕著となると考えられる．たとえば，現在のプロセッサのキャッシュ容量を考慮して，$4{,}096 = 2^{12}$ 個の格子点によるタイルを考えた場合，2 次元のタイルサイズは 64^2 に対し境界上の格子点数は $64 \times 4 = 256$ 個とタイルサイズの 6%程度であるのに対して，3 次元のタイルサイズは 16^3 となり，境界面上の格子点は $16^2 \times 6 = 1{,}536$ 個とタイルサイズの約 38% に達する．したがって 3 次元 FDTD 法に対しては，本論文の提案手法のように，タイル境界面での特別な処理を必要とせずオーバヘッドが小さい方法が適切であると考えられる．

また，空間に関するループを統合した場合，格子点上の未知変数間に依存関係が生ずるために，単純な領域分割による並列化が不可能となる．一方，提案手法では電場・磁場に関するループを統合する必要はなく，また 3.1 節で述べたように解析領域に対する 2 つの配列を使用することで，Tiling(ST) を用いた場合においても領域分割による並列処理が可能である．

6. ま と め

本研究では，電磁場解析の一手法である 3 次元 FDTD（Finite Difference Time Domain）法を対象とし，その計算性能の改善に関する検討を行い，キャッシュメモリの有効性を改善し計算性能を向上させる方法を提案した．

3 次元 FDTD 法の計算性能はプロセッサの演算性能よりもメモリアクセス性能に影響されやすいことに着目し，キャッシュメモリのヒット率を向上させてメインメモリへのアクセスによる性能の低下を軽減する性能改善手法（Tiling(ST)）を提案した．Tiling(ST) は解析領域をキャッシュメモリのサイズよりも小さな格子状の小領域に分割し，個々の小領域内で複数タイムステップにわたり電磁場の更新計算を連続して行う手法である．小領域全体がキャッシュメモリに格納された状態で計算を行い，計算量の増加と引き換えに，キャッシュミス率とメインメモリへのアクセス頻度の低減による高速化を図ることができる．

Tiling(ST) の評価は，京都大学学術情報メディアセンターの T2K オープンスーパコンピュータのノードである富士通 HX600 で，AMD 社製クアッドコア Opteron(8356) プロセッサを用いて実施した．数値実験の結果，逐次計算を行った場合には Naive な実装法に対して提案手法の優位性はみられなかった．これは，逐次実行の場合プロセッサあたり 1 コアしか使用しないため，並列実行時と比べてメモリの帯域が相対的に広く，キャッシュ上で計算を行うことによる性能改善に対して，提案手法のオーバヘッドである計算量（計算の対象となる格子点数）の増加の影響が大きかったためと考えられる．一方，スレッド並列処理を行った場合，メモリ帯域を各スレッドで共有するため，並列度の上昇とともに提案手法の Naive な実装に対する相対的な性能は上昇した．その結果，4 スレッド並列の場合，提案手法は適切なタイムステップ数やタイルサイズを設定することで，Naive な実装法と比べて最大で約 33%の実行時間の短縮（約 50%の性能向上）を得た．また，本数値実験において提案手法における適切なタイルサイズは実行スレッド数によらず，コアあたりのキャッシュサイズの約 25%前後であることが示された．解析的な分析により，タイルサイズが少なくともコアあたりのキャッシュサイズの 3 分の 1 以下でなければ，メインメモリへのアクセスによる性能劣化を引き起こすことが分かっており，適切なタイルサイズはキャッシュサイズの約 20〜30%を目安に設定すればよいと考えられる．

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 Propagat.*, Vol.AP-14, pp.302-3-7 (Aug. 1966).
2) Wolf, M.: More iteration space tiling, *Proc. Supercomputing '89* (1989).
3) 宇野　亨：FDTD 法による電磁界およびアンテナ解析，コロナ社，東京 (1998).
4) Nakashima, H.: T2K Open Supercomputer: Inter University and Inter-Disciplinary: Collaboration on the New Generation Supercomputer, *Proc. Intl. Conf. Informatics Education and Research for Knowledge-Circulating Society*, pp.137–142 (2008).
5) 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 HPC 研究会 2010-HPC-124 (2010).
6) Dongarra, J.J., Duff, I.S., Sorensen, D.C. and van der Vorst, H.A.: *Numerical Linear Algebra on High-Performance Computers*, Society for Industrial Mathematics (1987).
7) 寒川　光：RISC 超高速化プログラミング技法，共立出版 (1995).
8) Kamil, S., Husbands, P., Oliker, L., et al.: Impact of Modern Memory Subsystems on Cache Optimizations for Stencil Computations, *ACM Workshop on Memory System Performance*, June (2005).
9) Frigo, M. and Strumpen, V.: Cache oblivious stencil computations, *ICS '05: Proc. 19th annual international conference on supercomputing*, pp.361–366, ACM (2005).
10) Strzodka, R., Shaheen, M., Pajak, D. and Seidel, H.P.: Cache oblivious parallelograms in iterative stencil computations, *ICS '10: Proc. 24th ACM international conference on supercomputing*, pp.49–59, ACM (2010).
11) Bondhugula, U., Hartono, A., Ramanujam, J. and Sadayappan, P.: A practical automatic polyhedral parallelizer and locality optimizer, *ACM SIGPLAN Not.*, Vol.43, No.6, pp.101–113 (2008).

(平成 22 年 7 月 26 日受付)
(平成 22 年 11 月 8 日採録)

南　武志（学生会員）

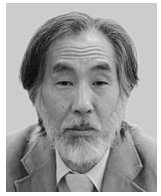
平成 22 年 4 月より京都大学大学院情報学研究科博士後期課程在籍．高性能計算，電磁界解析に関する研究に従事．

高橋　康人

昭和 55 年生．平成 20 年 3 月早稲田大学大学院理工学研究科博士後期課程修了．平成 18 年 4 月早稲田大学理工学術院助手．平成 20 年 4 月京都大学大学院情報学研究科特定助教．平成 22 年 4 月より同志社大学理工学部電気工学科助教．主に電磁現象を対象とした高速大規模数値解析技術に関する研究に従事．平成 17 年，平成 20 年電気学会優秀論文発表賞，平成 22 年電気学会電力・エネルギー部門誌優秀論文賞受賞．博士（工学）．IEEE，電気学会，日本太陽エネルギー学会，日本 AEM 学会各会員．

岩下　武史（正会員）

平成 10 年京都大学大学院工学研究科電気工学専攻博士課程修了．博士（工学）．平成 10 年京都大学大学院工学研究科リサーチ・アソシエイト（日本学術振興会未来開拓学術研究推進事業 PD），平成 12 年同大学大型計算機センター助手を経て，平成 15 年より同大学学術情報メディアセンター助教授（平成 19 年職名変更により同准教授），現在に至る．高性能計算，線形反復法，電磁界解析，並列処理に関する研究に従事．IEEE，SIAM，日本応用数理学会，電気学会，日本 AEM 学会，日本計算工学会各会員．



中島　　浩（正会員）

昭和 31 年生．昭和 56 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．推論マシンの研究開発に従事．平成 4 年より京都大学工学部助教授．平成 9 年より豊橋技術科学大学教授．平成 18 年より京都大学教授．並列計算機のアーキテクチャ，プログラミング言語の実装方式に関する研究に従事．工学博士．昭和 63 年元岡賞，平成 5 年坂井記念特別賞受賞．IEEE-CS，ACM，ALP，TUG 各会員．