Vol.2010-HPC-124 No.5
2010/2/22

# キャッシュメモリを考慮した FDTD カーネルの性能改善

南　武志[†1]　岩下武史[†2]
高橋康人[†1]　中島浩[†2]

本論文では，高周波電磁場解析の一手法である 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．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小領域に分割し，これらの小領域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，FDTD カーネルを高速に実行する方法を提案する．その結果，4 コアの AMD Opteron プロセッサによる数値実験において，各コアで FDTD カーネルを並行に実行した場合，通常の FDTD 法の実装と比較して，約 3 割の速度向上を得た．

# Cache-Aware Performance Improvement of FDTD Kernel

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

This paper deals with performance improvement of 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 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 FDTD method attains up to 30 percent speedup compared with an ordinary implementation of the FDTD method.

## 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科学技術計算における重要な問題の一つであるが，解析対象の大規模化・複雑化に伴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一つとして，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，高周波の電磁場解析において主要な解析手法の一つである FDTD(Finite Difference Time Domain) 法[1)2)] 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及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，一般に解析領域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．同手法を通常の FDTD 法による実装と比較し，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. FDTD 法

FDTD 法は主に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毎に陽的に解いていく電磁場数値解析手法の一つである．本論文では離散電磁場変数の配置に Yee のメッシュを用い，電磁場計算においては Leap-flog アルゴリズムを用いる．以下にその詳細について述べる．

電磁場のふるまいを表す Maxwell の方程式は次のように表される．

†1 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
†2 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

Vol.2010-HPC-124 No.5
2010/2/22

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

$$\nabla \times \boldsymbol{H} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{j}, \tag{2}$$

$$\nabla \cdot \boldsymbol{B} = 0, \tag{3}$$

$$\nabla \cdot \boldsymbol{D} = \rho. \tag{4}$$

ここで，諸変数は $\boldsymbol{E}$:電場，$\boldsymbol{H}$:磁場，$\boldsymbol{D}$:電束密度，$\boldsymbol{B}$:磁束密度，$\boldsymbol{j}$:電流密度，$\rho$:電荷密度である．また，誘電率 $\epsilon$，透磁率 $\mu$，導電率 $\sigma$ を用いると，構成方程式は

$$\boldsymbol{D} = \epsilon \boldsymbol{E}, \tag{5}$$

$$\boldsymbol{B} = \mu \boldsymbol{H}, \tag{6}$$

$$\boldsymbol{j} = \sigma \boldsymbol{E}, \tag{7}$$

と与えられるため，Maxwell 方程式から，$\boldsymbol{D}$，$\boldsymbol{B}$ を消去して支配方程式

$$\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t}, \tag{8}$$

$$\nabla \times \boldsymbol{H} = \epsilon \frac{\partial \boldsymbol{E}}{\partial t} + \sigma \boldsymbol{E}, \tag{9}$$

が得られる．

式 (8)，(9) に対して，Leap-flog アルゴリズムを用いた時間の離散化を行う．このアルゴリズムは，時間的に半ステップずつずらして電場と磁場を配置することで，時間微分項に関する差分近似において中心差分を行い，電場から磁場を，磁場から電場を，交互に求めていくアルゴリズムである．時間ステップ $\Delta t$ 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

$$\frac{\boldsymbol{H}^{n+1/2} - \boldsymbol{H}^{n-1/2}}{\Delta t} = -\frac{1}{\mu}\left(\nabla \times \boldsymbol{E}^n\right), \tag{10}$$

$$\frac{\boldsymbol{E}^n - \boldsymbol{E}^{n-1}}{\Delta t} = \frac{1}{\epsilon}\left(\nabla \times \boldsymbol{H}^{n-1/2}\right) - \frac{\sigma}{\epsilon}\boldsymbol{E}^{n-1/2}, \tag{11}$$

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

$$\boldsymbol{E}^n = \frac{1 - (\sigma\Delta t/2\epsilon)}{1 + (\sigma\Delta t/2\epsilon)}\boldsymbol{E}^{n-1} + \frac{\Delta t/\epsilon}{1 + (\sigma\Delta t/2\epsilon)}\left(\nabla \times \boldsymbol{H}^{n-1/2}\right), \tag{12}$$

$$\boldsymbol{H}^{n+1/2} = \boldsymbol{H}^{n-1/2} - \frac{\Delta t}{\mu}\nabla \times \boldsymbol{E}^n, \tag{13}$$

が求められる．

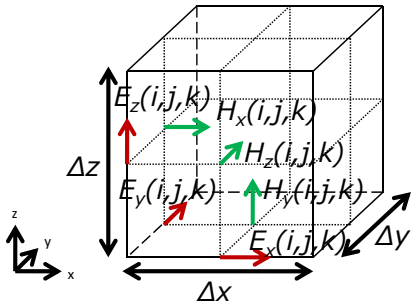


**図 1** Yee のメッシュ
Fig. 1 Yee-Cell

解析空間上の $x$ 軸，$y$ 軸，$z$ 軸での座標を $(x, y, z)$ と表記する．**図 1** に示すように，空間を離散化しセルの辺上に電場 $\boldsymbol{E}_x$，$\boldsymbol{E}_y$，$\boldsymbol{E}_z$，磁場 $\boldsymbol{H}_x$，$\boldsymbol{H}_y$，$\boldsymbol{H}_z$ を交互に配置し，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．これにより式 (12)，(13) に含まれる $\nabla \times \boldsymbol{E}$，$\nabla \times \boldsymbol{H}$ を中心差分公式を用いて以下の差分式で表現することが可能となる．

$$\begin{aligned}\boldsymbol{E}_x^n(i,j,k) = {} & C_e(i,j,k)\boldsymbol{E}_x^{n-1}(i,j,k) \\ & + \frac{C_{er}(i,j,k)}{\Delta y}\left\{\boldsymbol{H}_z^{n-1/2}(i,j,k) - \boldsymbol{H}_z^{n-1/2}(i,j-1,k)\right\} \\ & + \frac{C_{er}(i,j,k)}{\Delta z}\left\{\boldsymbol{H}_y^{n-1/2}(i,j,k) - \boldsymbol{H}_y^{n-1/2}(i,j,k-1)\right\},\end{aligned} \tag{14}$$

$$\begin{aligned}\boldsymbol{E}_y^n(i,j,k) = {} & C_e(i,j,k)\boldsymbol{E}_y^{n-1}(i,j,k) \\ & + \frac{C_{er}(i,j,k)}{\Delta z}\left\{\boldsymbol{H}_x^{n-1/2}(i,j,k) - \boldsymbol{H}_x^{n-1/2}(i,j,k-1)\right\} \\ & + \frac{C_{er}(i,j,k)}{\Delta x}\left\{\boldsymbol{H}_z^{n-1/2}(i,j,k) - \boldsymbol{H}_z^{n-1/2}(i-1,j,k)\right\},\end{aligned} \tag{15}$$

$$\begin{aligned}\boldsymbol{E}_z^n(i,j,k) = {} & C_e(i,j,k)\boldsymbol{E}_z^{n-1}(i,j,k) \\ & + \frac{C_{er}(i,j,k)}{\Delta x}\left\{\boldsymbol{H}_y^{n-1/2}(i,j,k) - \boldsymbol{H}_y^{n-1/2}(i-1,j,k)\right\} \\ & + \frac{C_{er}(i,j,k)}{\Delta y}\left\{\boldsymbol{H}_x^{n-1/2}(i,j,k) - \boldsymbol{H}_x^{n-1/2}(i,j-1,k)\right\},\end{aligned} \tag{16}$$

Vol.2010-HPC-124 No.5
2010/2/22

$$\boldsymbol{H}_x^{n+1/2}(i,j,k) = \boldsymbol{H}_x^{n-1/2}(i,j,k) + \frac{C_{hr}(i,j,k)}{\Delta y}\left\{\boldsymbol{E}_z^n(i,j{+}1,k) - \boldsymbol{E}_z^n(i,j,k)\right\} + \frac{C_{hr}(i,j,k)}{\Delta z}\left\{\boldsymbol{E}_y^n(i,j,k{+}1) - \boldsymbol{E}_y^n(i,j,k)\right\}, \quad (17)$$

$$\boldsymbol{H}_y^{n+1/2}(i,j,k) = \boldsymbol{H}_y^{n-1/2}(i,j,k) + \frac{C_{hr}(i,j,k)}{\Delta z}\left\{\boldsymbol{E}_x^n(i,j,k{+}1) - \boldsymbol{E}_x^n(i,j,k)\right\} + \frac{C_{hr}(i,j,k)}{\Delta x}\left\{\boldsymbol{E}_z^n(i{+}1,j,k) - \boldsymbol{E}_z^n(i,j,k)\right\}, \quad (18)$$

$$\boldsymbol{H}_z^{n+1/2}(i,j,k) = \boldsymbol{H}_z^{n-1/2}(i,j,k) + \frac{C_{hr}(i,j,k)}{\Delta x}\left\{\boldsymbol{E}_y^n(i{+}1,j,k) - \boldsymbol{E}_y^n(i,j,k)\right\} + \frac{C_{hr}(i,j,k)}{\Delta y}\left\{\boldsymbol{E}_x^n(i,j{+}1,k) - \boldsymbol{E}_x^n(i,j,k)\right\}. \quad (19)$$

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 $\epsilon$，透磁率 $\mu$，導電率 $\sigma$ によって，

$$C_e(i,j,k) = \frac{1-(\sigma(i,j,k)\Delta t/2\epsilon(i,j,k))}{1+(\sigma(i,j,k)\Delta t/2\epsilon(i,j,k))}, \quad (20)$$

$$C_{er}(i,j,k) = \frac{\Delta t/\epsilon(i,j,k)}{1+(\sigma(i,j,k)\Delta t/2\epsilon(i,j,k))}, \quad (21)$$

$$C_{hr}(i,j,k) = \frac{\Delta t}{\mu(i,j,k)}, \quad (22)$$

と定められる．

本論文で扱う FDTD 法では，式 (14) から (19) を用いて電場 $\boldsymbol{E}$ と磁場 $\boldsymbol{H}$ を更新する．なお，以降において一般的な FDTD 法の実装とは式 (14) から (19) をそのまま電場，磁場それぞれについて格子点座標に関する 3 重ループにより実装する方法を指すものとする．**図 2** に一般的な実装法による FTDT 法の電場の更新に関する計算の擬似コードを示す．FDTD 法では，解析空間上の電場 $\boldsymbol{E}(i,j,k)$ に含まれる 3 つのベクトルを更新するためには空間座標に依存する変数が 17 個必要であり，磁場 $\boldsymbol{H}(i,j,k)$ に含まれる 3 つのベクトルを更新するためには 16 個必要となる．これらの多くは解析空間上で $(i,j,k)$ に隣接する格子点上のベクトル変数であるが，解析領域は一般にキャッシュメモリ容量よりもはるかに大きいため，キャッシュメモリ上での再利用は困難で，計算時にメインメモリへのアクセスが必要となる．

## 3. 性能改善手法

本稿では，キャッシュメモリを利用してメモリ帯域による性能の低下を軽減する性能改善手法（以下，Tiling と呼ぶ）を提案する．Tiling は解析領域を格子状に分割し，分割された個々の領域をキャッシュメモリ上で計算することによりメインメモリへのアクセスの頻度を減らし高速化を行う手法である．

Tiling を行う場合，電磁場の更新において **図 3** に示されるように，解析領域を格子状に分割し，分割された小領域内で複数ステップの電磁場の更新を行い，更新された値を元の領域に戻す操作を行う．分割された小領域を以下タイルと呼ぶ．以上の操作を分割されたタイルすべてにおいて行うことにより，タイルがキャッシュメモリに収まるサイズの場合，キャッシュメモリ上で複数ステップの計算をまとめて行うことでキャッシュヒット率の向上が得られる．

3 次元の解析領域で Tiling を行う場合，**図 4** のように一つのタイルの一辺に含まれる要素（格子点）の数を $n_t$ とし，大きさ $n_t \times n_t \times n_t$ の立方体に分割する．また，タイル内で一度に行う電磁場更新のステップ数を $s_t$ とする．この時電場 $\boldsymbol{E}$ と磁場 $\boldsymbol{H}$ は交互にそれぞれ $s_t$ 回更新される．それぞれのタイルごとに電磁場の更新を行うが，式 (14) から (19) より，電場 $\boldsymbol{E}$，磁場 $\boldsymbol{H}$ の更新のためにはそれぞれ解析領域上で隣に位置する格子点の $\boldsymbol{H}$，$\boldsymbol{E}$ 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 $\boldsymbol{E}(i+1,j,k)$，$\boldsymbol{E}(i,j+1,k)$，$\boldsymbol{E}(i,j,k+1)$ が必要となり，電場と磁場を合わせた 1 ステップの計算のためにはタイルのサイズを各面の外側に一つの格子点分拡大した大きさ $(n_t+2)\times(n_t+2)\times(n_t+2)$ の立方体の解析領域が必要となる．タイル内での計算が $s_t$ ステップの場合，最初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．また，この時タイル外で必要となる領域は相互に他のタイルに含まれているため，タイル内の計算結果の $\boldsymbol{E}$，$\boldsymbol{H}$ を更新すると，タイルの計算の順序にかかわらず，後に計算することになる他のタイルの計算で必要な格子点上の値が失われてしまう．そのため計算結果は元の領域とは別に保持しなければいけない．

以上を踏まえて，具体的には以下のような実装を行う．

Step 1. 解析領域全体と同じ大きさ $n$ の 2 つの解析領域 $S$ と $R$ と 一片のサイズが $n_t+2s_t$ の領域 $T$ を用意し，$S$ に初期値を入力する

```
dx=1/Δx
dy=1/Δy
dz=1/Δz
for(i=1;i<nx;i++)
  for(j=1;j<ny;j++)
    for(k=1;k<nz;k++)
      Ex(i,j,k) = Ce(i,j,k) * Ex(i,j,k) + Cer(i,j,k) * dy(j) * (Hz(i,j,k) - Hz(i,j-1,k)) + Cer(i,j,k) * dz(k) * (Hy(i,j,k) - Hy(i,j,k-1))
      Ey(i,j,k) = Ce(i,j,k) * Ey(i,j,k) + Cer(i,j,k) * dz(k) * (Hx(i,j,k) - Hx(i,j,k-1)) + Cer(i,j,k) * dx(i) * (Hz(i,j,k) - Hz(i-1,j,k))
      Ez(i,j,k) = Ce(i,j,k) * Ez(i,j,k) + Cer(i,j,k) * dx(i) * (Hy(i,j,k) - Hy(i-1,j,k)) + Cer(i,j,k) * dy(j) * (Hx(i,j,k) - Hx(i,j-1,k))
    }
  }
}
```

**図 2** 一般的な FDTD 法における電場 $\boldsymbol{E}$ の更新
Fig. 2 Update of electric field in general FDTD method

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．

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

Step 5. 領域 $T$ から領域 $R$ へコピーする．この時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．

上記の手順を繰り返すにより Tiling を用いた FDTD 法の実行が可能となる．

実装の手順によりわかるように，本手法では一般的な FDTD 法の実装と比べて約 2 倍のメモリ領域が必要となる．また，Tiling を行うと通常の手法と比べて計算量が増加する．これは複数タイムステップをまとめて処理する際に，$s_t$ タイムステップ後におけるタイル内の電磁場が正しく計算されるためにはタイルの大きさ以上の計算が必要となるためである．そこで，一般的な実装法と比較して，Tiling によりどの程度計算量が増加するか検討する．ここでは簡単のために，解析領域境界の影響は考慮しないものとする．通常の FDTD 法で $s_t$ タイムステップあたりに更新する格子点数を $F_n$，Tiling を用いた FDTD 法で $s_t$ タイムステップあたりに計算する格子点数を $F_t$ とし，両者を比較すると

$$F_n = 2s_t n_t^3, \tag{23}$$

$$F_t = \sum_{k=n_t}^{n_t+2s_t-1} k^3, \tag{24}$$

と表せる．この両者の比率 $(F_t/F_n)$ は**図 5** のようになり，タイルサイズが比較的小さな場合計算量は一般的な実装法と比べて非常に多くなるが，タイルサイズが大きくなるに従い減少し 1 に漸近することがわかる．

## 4. 数 値 実 験

### 4.1 使用計算環境と解析モデル

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 オープンスーパーコンピュータのノードである富士通 HX600[3] である．HX600 は 4 個の AMD 社製クアッドコア Opteron プロセッサと 32GB(DDR2-667) のメモリを有している．プログラミング言語として C 言語を使用し，最適化オプションには-Kfast,noprefetch,restp=all を指定した．Opteron プロセッサの性能は，理論ピーク演算

Vol.2010-HPC-124 No.5
2010/2/22

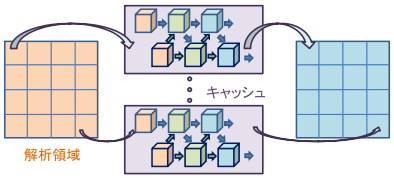


**図 3** Tiling の概要
Fig. 3 Outline of Tiling

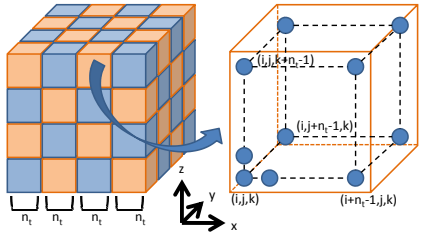


**図 4** 解析領域の分割
Fig. 4 Division of analytical domain

性能が 36.8GFlops，コアあたり 9.2GFlops，キャッシュメモリ容量は L2 キャッシュがコアあたり 512KB，L3 キャッシュがプロセッサあたり 2MB となっている．また，メインメモリの帯域はプロセッサあたり 10.6GB/s である．

本実験で使用する解析領域は，立方体形状の解析領域とする．解析領域内で電場 $\boldsymbol{E}$，磁場 $\boldsymbol{H}$ を交互に更新する一般的な実装法と Tiling を用いた手法について，領域のサイズを変えながら 120 タイムステップの計算を行い，両者の 1 格子点の 1000 タイムステップあたりの計算時間を求め比較する．

本数値実験では逐次的な FDTD 法のプログラムを用いるが，FDTD 法は一般に並列化が容易な手法であるため，実応用上はプロセッサ内の全てのコアが使用される状況が想定される．そこで以下の二つの場合について実験を行う．

Case 1. プロセッサあたり 1 コアを使用する場合．

Case 2. プロセッサあたり 4 コアを使用する場合．

ただし，Case 2 では各コアにおいて逐次的な FDTD プログラムを並行に実行する．プロセッサあたり 1 コアが動作する場合，計算を実行するコアはプロセッサの L3 キャッシュとメモリ帯域のすべてを使用することが可能である．一方，プロセッサあたり 4 コアが動作する場合，4 つのコアが L3 キャッシュとメモリ帯域を共有する．FDTD 法は常に大量のメモリアクセスを必要とするため，Case 2 の場合それぞれのコアが利用できるメモリ帯域は平均 2.65GB/s と見積もられる．

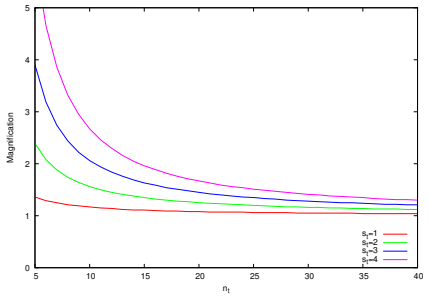


**図 5** Tiling により増加する計算量
Fig. 5 Calculated amount of the Tiling

### 4.2 数値実験結果

一般的な実装法による FDTD カーネルをプロセッサあたり 1 コアおよび 4 コアで実行した結果が **図 6** である．これを見ると，解析領域のサイズは，1 コアの場合は $30^3$，4 コアの場合は $20^3$ を越えたあたりで計算時間が急激に増加していることがわかる．これは解析領域のデータ量の増加によりキャッシュメモリ上での計算が不可能になり，メインメモリへのアクセスが発生したためである．実際 2 次キャッシュとコアが利用可能な 3 次キャッシュ上に解析領域の全データを保持できるおよそのサイズはプロセッサあたり 1 コアの場合 $32^3$，4 コアの場合 $24^3$ となるため，上記のサイズとほぼ一致する．キャッシュメモリ上で計算可能な場合とメインメモリへのアクセスが必要な場合とでは，1 コアの場合は約 3 倍，4 コアの場合は約 7 倍，計算時間に差が出ることが確認できる．

**図 7**，**図 8** に，Tiling のタイムステップ数 $s_t$ が 1 から 4 の時の数値実験結果を示す．図 7 はプロセッサあたり 1 コアの場合，図 8 はプロセッサあたり 4 コアの場合である．ここで，図への描写は，タイルの一辺のサイズ $n_t$ がプロセッサあたり 1 コアの場合 12，18，24，プロセッサあたり 4 コアの場合は 6，12，18 について行った．

図 7 を見ると，Tiling のステップ数 $s_t$ やタイルのサイズ $n_t$ にかかわらず一般的な FDTD 法の方が提案手法よりも計算時間が短く，高速に動作している．これは，プロセッサあたり

1 コアしか使用しない場合メモリの帯域は十分に確保されており，提案手法において 図 5 により表される計算量の増加による影響が大きかったためと考えられる．

一方，プロセッサあたり 4 コアを使用した場合の 図 8 をみると，Tiling のステップ数 $s_t$ が 1，2，3 の場合，タイルのサイズ $n_t$ を適切に選択すると，解析領域のサイズが十分に大きいとき，提案手法は一般的な FDTD 法より高速に動作することが分かる．ここで，$s_t$ が 4 の場合に性能が発揮されないのは，1 コアを使用した場合と同様に，計算量の増加による影響が実行演算性能の改善を上回ったからであると考えられる．

図 7，図 8 をタイルのサイズ $n_t$ に関してまとめたものが **図 9**，**図 10** である．ここで計算時間は，プロセッサあたり 1 コアを使用した場合は解析領域のサイズ 220 から 280 の平均値，4 コアを使用した場合は解析領域のサイズが 170 から 230 の平均値とする．プロセッサあたり 1 コアを使用した場合が 図 9，4 コアを使用した場合が 図 10 となる．これらより，図 7，図 8 と同様にプロセッサあたり 1 コアしか使用しない場合はタイルサイズ $n_t$，タイル内でのタイムステップ数 $s_t$ をどのような値としても提案手法は一般的な FDTD 法以上の計算時間がかかる．一方，4 コアを使用する場合は $n_t$ と $s_t$ の値次第で提案手法は一般的な FDTD 法と比べ最大で約 3 割高速となることが確認できる．

また，図 9，図 10 において，Tiling のステップ数 $s_t$ に関わらず，タイルサイズ $n_t$ に対して計算時間は極小値を持つ 2 次曲線状になることが確認できる．計算時間が極小となる大きさより $n_t$ が小さい場合，図 5 とほぼ同様に $n_t$ の増加に従い計算時間は減少する．これは，タイルサイズが十分に小さく領域 $T$ の全データがキャッシュメモリ上に格納可能で，領域 $S$，$R$ とのやり取り以外にメインメモリへのアクセスが発生しないため，性能が 図 5 に示すタイル内での計算量に依存するためである．しかし，$n_t$ が大きくなると計算時間はやがて増加に転じる．これは，図 6 において一般的な FDTD 法の計算で解析領域全体がキャッシュメモリのサイズを超えたときに計算時間が増加するのと同様のことが，領域 $T$ がキャッシュメモリサイズを超えることによって起きているからであると考えられる．実際，図 10 を見ると，計算時間が最小となるとき $n_t = 12$，$s_t = 2$ で，この時タイルの計算を行う領域 $T$ のサイズは $(n_t + 2s_t)^3 = 16^3$ となり，コアの利用できるキャッシュサイズのおよそ 1/3 を占めている．Tiling を用いた計算では $T$ の計算に加えて $S$，$R$ とのデータのやり取りも必要なため，$S$ と $R$ に関してもキャッシュ上に $T$ と同じ大きさの領域を確保しなければ，データをコピーする際に $T$ の一部がキャッシュメモリ上から消えてしまい，タイルの計算においてメインメモリへのアクセスが発生し遅延が起きる．以上より，Tiling を行う場合，$T$ がコアの利用可能なキャッシュサイズのおよそ 1/3 を占る時，最もキャッシュを有効に利用しているといえる．これは，図 10 において $s_t$ が異なる場合でも計算時間を極小とする領域 $T$ のサイズはほぼ同じとなることや，図 9 においても計算時間が最も短くなる場合の $T$ のサイズはコアの利用可能なキャッシュサイズのおよそ 1/3 であることからも確認できる．

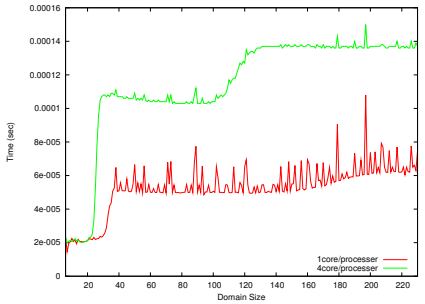


**図 6** 一般的な FDTD 法の計算時間
Fig. 6 Computation time of normal FDTD method

## 5. おわりに

本論文では，FDTD 法に関して，キャッシュメモリを有効に利用することにより，計算量の増加と引き換えにメモリ帯域による性能の劣化を軽減する手法として，Tiling による性能改善手法を提案した．提案手法を数値実験で一般の実装法による FDTD 法と比較した結果，プロセッサあたり 1 コアを使用した場合には提案手法の優位性は得られなかった．しかし，プロセッサあたり 4 コアを使用した場合，コアあたりのメモリ帯域が小さくなるため，提案手法は一般の実装法と比べて約 3 割程度の高速化を実現した．

## 参考文献

1) K. S. Yee, “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) 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).
3) Nakashima, H.: T2K Open Supercomputer: Inter University and Inter-Disciplinary: Collaboration on the New Generation Supercomputer, in Proc. Intl. Conf. Informatics Education and Research for Knowledge-Circulating Society, pp.137-142, (2008).

Vol.2010-HPC-124 No.5
2010/2/22

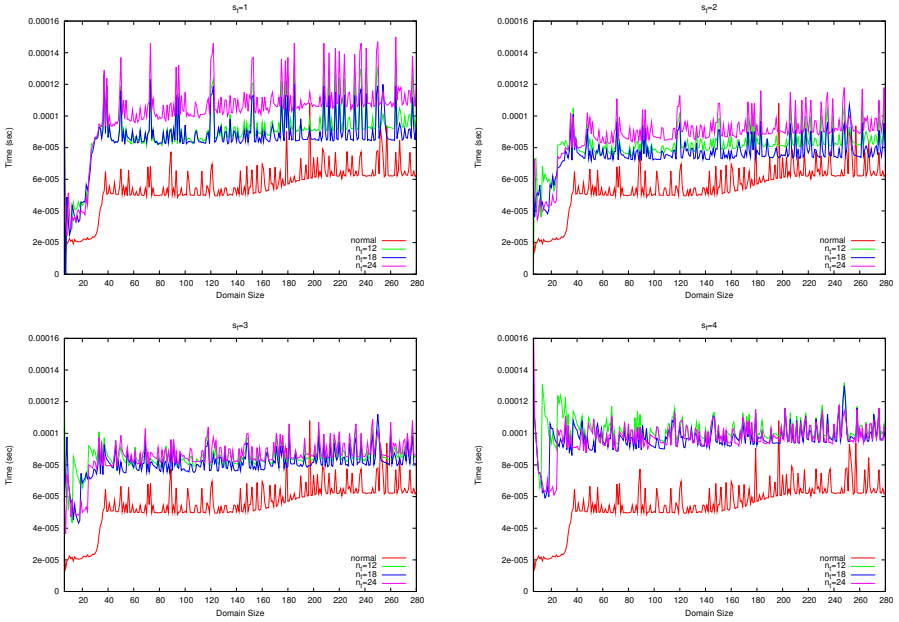


**図 7** 1core/processor での計算時間
Fig. 7 Computation time (1core/processor)

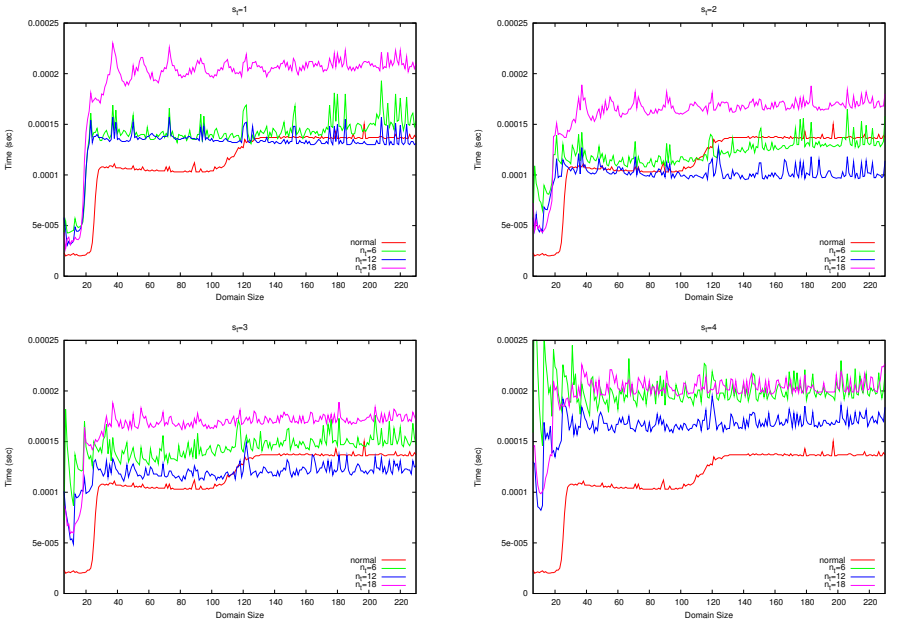


**図 8** 4core/processor での計算時間
Fig. 8 Computation time (4core/processor)

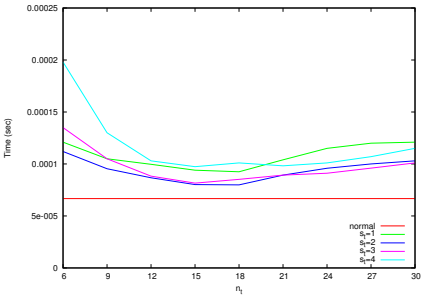


**図 9** 1core/processor での Tiling 性能
Fig. 9 Performance of Tiling (1core/processor)

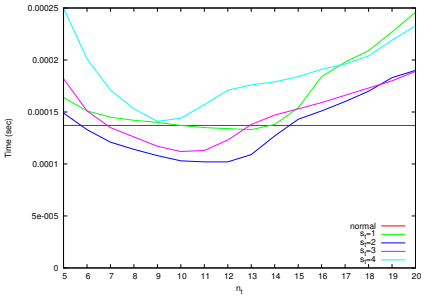


**図 10** 4core/processor での Tiling 性能
Fig. 10 Performance of Tiling (4core/processor)