Automatic Parameter Tuning of Three-Dimensional Tiled FDTD Kernel

Takeshi Minami[1], Motoharu Hibino[1], Tasuku Hiraishi[2], Takeshi Iwashita[2] and Hiroshi Nakashima[2]

[1] Graduate School of Informatics, Kyoto University, Japan
[2] Academic Center for Computing and Media Studies, Kyoto University, Japan

Abstract. This paper introduces an automatic tuning method for the tiling parameters required in an implementation of the three-dimensional FDTD method based on time-space tiling. In this tuning process, an appropriate range for the tile size is first determined by trial experiments using cubic tiles. The tile shape is then optimized by using the Monte Carlo method. The tiled FDTD kernel was multi-threaded and its performance with the tuned parameters was evaluated on multi-core processors. When compared with a naively implemented kernel, the performance of the tuned FDTD kernel was improved by more than a factor of two.

1 Introduction

The three-dimensional (3D) finite-difference time-domain (FDTD) method is widely used in high-frequency electromagnetic field analysis for the design of electrical devices [1, 2]. The method is based on iterative stencil computations composed of nested outer temporal and inner spatial loops. In iterative stencil computations, the number of floating point operations is often relatively small when compared with the total amount of data transferred between the CPUs and the main memory. Consequently, the computation time of the 3D FDTD kernel is often determined by the (effective) memory bandwidth rather than by the processing core performance. In particular, on computational nodes based on multi-core processors, the speed-up obtained is seldom proportional to the number of cores used, despite the fact that the computational kernel of the 3D FDTD method can be parallelized in a straightforward manner [3]. This is due to the well-known "memory-wall problem". In iterative stencil computations, "time-space" tiling is one possible solution to this problem [4].

In time-space tiling, the domain to be analyzed is divided into a number of smaller regions, i.e. *tiles*, each of which is so small that it is accommodated by the cache, and the electromagnetic field variables in each tile are updated for a certain number of time steps successively. Because most calculations on the tile are performed in the cache, this technique increases the cache hit ratio and thus produces better performance. The application of this technique to a two-dimensional FDTD kernel was reported in [5]. However, because the

3D FDTD kernel has a more complex stencil shape and loop structure than the two-dimensional case, there are few reports on time-space tiling for the 3D FDTD method. Under these circumstances, we previously proposed a time-space tiling method for the 3D FDTD kernel without redundant calculations (3D tiled FDTD), and confirmed its effectiveness when using a multi-core processor system [6].

In this paper, to achieve further improvements in performance, we describe a parameter tuning method for the 3D tiled FDTD kernel. The parameters to be tuned are the shapes of the tiles and the number of time steps of the successive updates of a tile. We tune these parameters based on the experimental results from relatively small-sized jobs in which the kernel performs for a small number of time steps. Numerical tests on multi-core processors of the latest generation exhibit that the tuned 3D FDTD kernel attains a more than two-fold performance improvement when compared to the naive implementation of the kernel.

2 Parameter Tuning for 3D Tiled FDTD Kernel

2.1 Time-space Tiling for 3D FDTD Method

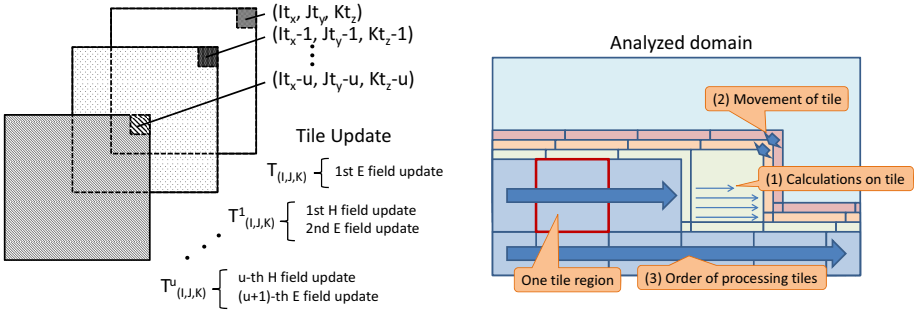
In this subsection, we introduce time-space tiling for the 3D FDTD method. Here, we consider a grid of $n_x \times n_y \times n_z$ for the analysis, where the subscripts represent the direction of each axis in the 3D coordinate system. In the tiling method, the grid is divided into tiles, each of which consists of $t_x \times t_y \times t_z$ grid points. For the purposes of the discussions that follow, we introduce 3D coordinates for the grid points and the tiles, which are written as (i, j, k), (i=1, ..., n_x, j=1, ..., n_y, z=1, ..., n_z) and (I, J, K)=($\lceil i/t_x \rceil$, $\lceil j/t_y \rceil$, $\lceil k/t_z \rceil$), respectively. Then, the tile located at (I, J, K), which is denoted here by $T_{I,J,K}$, is given by

$$T_{I,J,K} = \{(i,j,k) | (I-1)t_x + 1 \le i \le I \cdot t_x,\ (J-1)t_y + 1 \le j \le J \cdot t_y,\\ (K-1)t_z + 1 \le k \le K \cdot t_z\}. \quad (1)$$

The time-space tiling involves updating the electric and magnetic field variables for multiple time steps for each tile. The method is expected to reduce the number of main memory accesses and increase both the cache hit ratio and the kernel performance. However, the simple tiling method, in which fixed tiles are used, requires redundant calculations. Therefore, in our technique, the tile is moved by one grid point in all negative x-, y-, and z-directions at each time step to avoid these redundant calculations. For a more precise explanation, we introduce the notation $T^u_{I,J,K}$, which represents the moved tile for the u-th step update and is defined as follows:

$$T^u_{I,J,K} = \{(i,j,k) | (i+u, j+u, k+u) \in T_{I,J,K},\\ 1 \le i \le n_x,\ 1 \le j \le n_y,\ 1 \le k \le n_z\}, \quad (2)$$

where $1 \le u \le t_s$ and t_s is the number of successive updates of the tile. In this technique, the electric field in $T^{u-1}_{I,J,K}$ and the magnetic field in $T^u_{I,J,K}$ are



(a) Movement of tile for time steps (b) Order of calculations of field variables

Fig. 1. Time-space tiling for FDTD kernel

updated in the u-th time step for $T_{I,J,K}$, as shown in Fig. 1 (a). It should be noted that the tiles are processed in lexicographical order. Fig. 1 (b) shows the order in which the calculation is performed for each grid point.

In the present work, the calculations on each tile are parallelized. Because most calculations on the tile are executed on the cache, sufficient speedup is expected when using a multi-core processor. The spatial loop for the field variable on each tile is parallelized by using OpenMP in the 3D tiled FDTD program.

2.2 Design of Automatic Tuning Method

In the 3D tiled FDTD kernel, the tuned parameters are given by the number of grid points of each tile in the x-, y-, and z-directions, which are t_x, t_y, and t_z, respectively, and the number of time steps of the successive updates of a tile, t_s. Before determining the tuning strategy, we first consider the general characteristics of the parameters in terms of their effects on the kernel performance. Since the performance improvement owing to the tiling technique is derived from the increased cache hit ratio, the tile size must be smaller than the last level cache size. In addition, as the effect of increasing t_s gradually declines, the effects of setting too many time steps in a tile need not be examined. Therefore, we set the maximum number for t_s at 30.

Next, we conducted preliminary numerical experiments for a better understanding of the kernel characteristics, where the experimental condition is the same as described in Section 3.1. Fig. 2 shows the computation time for the tiled FDTD program with cubic tiles, where t_c is the number of grid points in a single tile in one direction. This figure indicates that tiles that are either too small or too large are ineffective. This is because the use of tiles that are too small causes increased management costs for the tiles. On the other hand, since the cache is not usually occupied only by the data for the tile, the appropriate tile size range spreads below the size perfectly matching the cache size. Consequently, we find the appropriate tile size range, which depends on the cache architecture

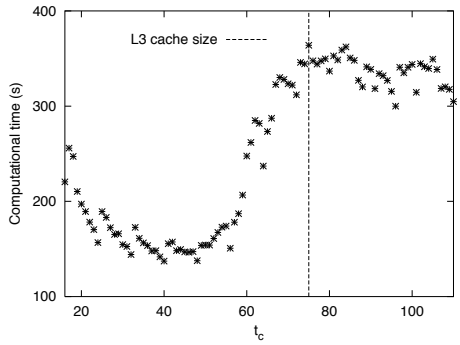


Fig. 2. Computation times for various cubic tiles

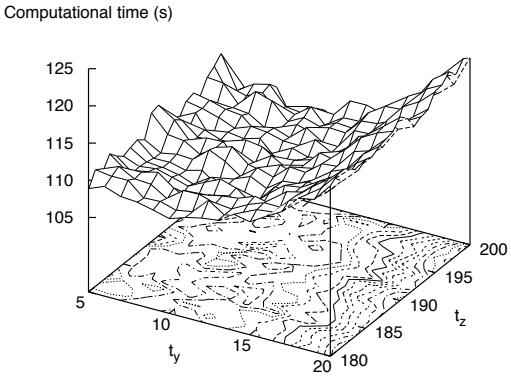


Fig. 3. Effects of t_y and t_z on the computation time

and the size of the problem, by means of trial experiments using cubic tiles on the computer that is to be used for the FDTD calculations.

The preliminary experimental results have also revealed some characteristics of the tiled FDTD program. In our initial research, although we tried using a local optimization technique such as the steepest descent method to tune the parameters, we could not obtain acceptable results. Fig. 3 shows the computation times when t_y and t_z are varied with t_x=40 and t_s=10. In the figure, many local minimum points can be observed, which implies that the local optimization techniques have not worked well. Consequently, we chose to use either a brute force approach or a Monte Carlo method in the tuning step, while the search space for the tile shape was limited to reduce the tuning cost.

In our program, the elements of the arrays for the field variables are contiguous on the main memory in the z-direction. Therefore, a tile having relatively long edges along the z-direction has an advantage for contiguous memory access, which is expected to increase the cache hit ratio. Another requirement is given

by the x-direction spacial loop to be multi-threaded. That is, t_x is required to be larger than the number of threads, and if it is sufficiently large, the load imbalance among the threads is kept small. Accordingly, we can narrow down the search space for the tile shape to tiles that satisfy $t_x > t_y$ and $t_z > t_y$.

The numerical results also imply that the number of time steps t_s can be tuned separately from the tile shape parameters. It was noted that the effect of t_s on the calculation time is smaller than that of the tile shape. The numerical results also indicate that the optimized value of t_s for a specific tile also works well for tiles that are similar to the specified tile. Considering the aspects discussed above, we finally propose the following tuning strategy:

STEP 1. (Cubic tile optimization) Fix t_s at 10. Optimize the tile size, t_c, subject to $t_c = t_x = t_y = t_z$. Let t'_c denote the optimized value obtained for t_c.

STEP 2. Optimize $t_s \in \{1, 2, \ldots, 30\}$ using the tile size $(t'_c)^3$. Here, let t'_s denote the optimized value of t_s.

STEP 3. (Tile shape optimization) Fix t_s at t'_s. Perform a Monte Carlo search in the tile-size space satisfying the following criteria:

(a) $0.95 \times (t'_c)^3 \leq t_x \times t_y \times t_z \leq 1.35 \times (t'_c)^3$

(b) $t_y < t_x$, $t_y < t_z$

STEP 4. Optimize t_s by using the tile size that was obtained in STEP 3.

STEP 5. (Optional) Perform a local search around the parameters obtained in Step 4.

Condition (a) in STEP 3 is derived from our previous experiences of the tuning process. When the tile shape is optimized, the tile tends to be larger than an optimized cubic tile. This is because the optimized tile uses the cache more efficiently, and thus the tile size ratio with respect to the cache size increases when compared with the optimized cubic tile.

Although we believe that STEP 5 increases the robustness of the tuning process, the performance improvement obtained from STEP 5 is often too small to compensate the computational cost required for the step. Therefore, in the following numerical experiments, we did not conduct STEP 5 and used the results of STEP 4 for the final tuned parameters.

3 Numerical Results

3.1 Computer and Program

To examine the performance of the tuned 3D FDTD kernel, we conducted numerical tests using an 8-core Intel Xeon E5-2670 processor. The Intel C compiler version 12.1.6 was used with the option of "-restrict -O3 -xHost -static_intel -openmp". The domain to be analyzed is an cubic grid space of 800^3 and the total number of time steps is 90. The number of trial tasks in STEP 3 of the tuning process is 500.

Fig. 4 lists the sample program code based on a naive implementation for our analysis. The spatial parameters (coefficients) for each grid point, which are determined based on the permittivity, the magnetic permeability, and the conductivity, are given through the array **id**, which designates the kind of medium

```
for (t=0;t<total_t;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]=Cex[m]*Ex[i][j][k]+Cexry[m]*(Hz[i][j][k]-Hz[i][j-1][k])+
        Cexrz[m]*(Hy[i][j][k]-Hy[i][j][k-1]);
     Ey[i][j][k]=Cey[m]*Ey[i][j][k]+...;
     Ez[i][j][k]=Cez[m]*Ez[i][j][k]+...;}}}
  H-field update (three nested loops)}
```

Fig. 4. Loop structure of 3D FDTD kernel based on naive implementation

```
for (t=0;t<total_t;t+=ts){
  for (l=1;l<=ntile;l++){
    for (tt=0;tt=ts-1;tt++){
      for (i=is[l]-tt;i<=ie[l]-tt;i++){
        for (j=js[l]-tt;j<=je[l]-tt;j++){
          for (k=ks[l]-tt;k<=ke[l]-tt;k++){
            m=id[i][j][k];
            E-field update}}}
      H-field update (three nested loops) }}}
```

Fig. 5. Loop structure of 3D tiled FDTD kernel

(i.e. material) involved at each grid point. In the analysis, the model is assumed to be formed by only a few materials. However, the FDTD program can deal with a model that contains more materials, and can even accommodate a model with coefficients that vary at each grid point. Fig. 5 shows the simplified sample program of the tiled 3D FDTD method, where the process for the domain boundaries is excluded for simplicity. In the numerical tests, both programs are multi-threaded by the use of OpenMP. The outer-most spatial loop with respect to i, which corresponds to the spatial loop in the x-direction, is parallelized.

In the proposed tuning method, a number of trial jobs should be managed. For this purpose, we used Xcrypt [7], which is a Perl-based script language for job-level parallel processing on batch-queuing parallel computers. The Xcrypt (Perl) program automatically generates the set of parameters for each tuning step, to be swept by many jobs in parallel, from the results of the previous step.

3.2 Numerical Results

Table 1 provides a performance comparison between the tuned tiled 3D FDTD program and the program based on the naive implementation. In both sequential and parallel computation, the tuned version of the program is approximately 2.3 times as fast as the naive implementation. The number of trial jobs is about 600 and the tuned parameter is expected to be effective for all analyses with the same problem size and the same number of threads. Consequently, the proposed

Table 1. Performance evaluation of 3D tuned FDTD kernel

(a) Sequential computation using one thread

	# trial jobs	t_x, t_y, t_z, t_s	Elapsed time (s)
Naive implementation	-	-	1586
Tuned kernel by Step 2 (cubic tile)	103	48, 48, 48, 23	875
Tuned kernel	633	35, 8, 414, 29	686

(b) Parallel computation using eight threads

	# trial jobs	t_x, t_y, t_z, t_s	Elapsed time (s)
Naive implementation	-	-	237
Tuned kernel by Step 2 (cubic tile)	96	40, 40, 40, 17	135
Tuned kernel	626	24, 10, 320, 15	104

tuning method is expected to be useful for practical simulations in which many time steps are solved.

The tuned FDTD program is about 1.3 times as fast as the tiled FDTD program with the optimized cubic tile. The final tuned tile has a relatively long edges in the z-direction, which corresponds to the contiguous memory access. The final tuned tile is larger than the optimized cubic tile obtained in STEP 2. In parallel computation, the sizes of the optimized cubic and tuned tiles are 16.3% and 19.5% of the cache size, respectively. This result indicates that the tile shape optimization promotes efficient use of the cache.

Fig. 6 shows the relative (parallel) speedup when compared with the sequential FDTD kernel based on the naive implementation. Fig. 6 confirms that the tuned kernel attains good parallel performance achieving 8-thread speedup of 6.6-fold and 15.2-fold over the sequential executions of its own and of the naive kernel.

For further confirmation of the effectiveness of the proposed tuning method, we conducted additional numerical tests using different problem sizes and another computational node. One of the computational nodes of the Cray XE6 system was used for these tests. The XE6 node consists of two 16-core AMD Opteron processors, and one of these processors was used in the analysis. The Cray C compiler version 5.26 was used with the option of "-h omp -O3 -h pic -dynamic". Table 2 shows a comparison between the results from the tuned 3D FDTD program and those from the naive implementation. It was confirmed that the tuned kernel outperformed the naive kernel in all cases. The tuned kernel was mostly more than twice as fast as the naive implementation. The numerical results indicate that the use of space-time tiling with the tuned parameters is effective in obtaining better performance on multi-core processors.

4 Conclusion

In this paper, we summarized a 3D tiled FDTD method and proposed an auto-tuning technique for the tiling parameters to enhance the performance of the method. In the tuning step, we first find the appropriate tile size range by performing trial jobs with cubic tiles. Next, the tile shape is optimized by using the

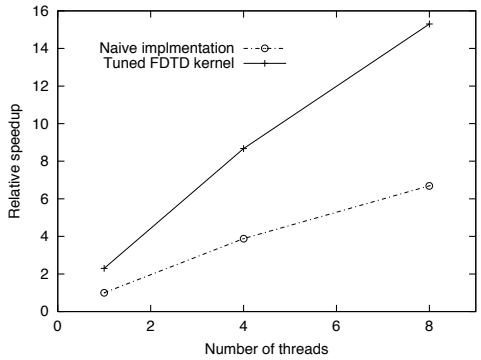


Fig. 6. Comparison of the tuned kernel with naive sequential implementation

Table 2. Evaluation results of the tuned kernel on various computational conditions

Machine	# Threads	Problem size	Naive implementation (s)	Tuned version (s)
XE6	16	600^3	124	73.3
XE6	16	800^3	292	140
XE6	16	1000^3	632	268
Xeon E5	8	600^3	97.5	44.5
Xeon E5	8	1000^3	503	213

Monte Carlo method. To reduce the total number of trial jobs, we enforce limitations with respect to the tile size and shape in this tuning step. The resulting tuned FDTD kernel is about twice (up to 2.36 times) as fast as the naive kernel in both serial and parallel computations on two types of multi-core processors of the latest generation.

References

1. J. Lu, D. Thiel and S. Saario, "FDTD analysis of dielectric-embedded electronically switched multiple-beam (DE-ESMB) antenna array", IEEE Trans. Magn., vol. 38, pp. 701-704, (2002).
2. G. Ala, M.C. Di Piazza, G. Tine, F. Viola and G. Vitale, "Numerical simulation of radiated EMI in 42 V electrical automotive architectures", IEEE Trans. Magn., vol. 42, pp. 879-882, (2006).
3. K.C. Chew and V.F. Fusco, "A parallel implementation of the finite difference time-domain algorithm", Int. J. Numer. Model., vol. 8, pp. 293-299, (1995).
4. M. Wolf, "More iteration space tiling", Proc. Supercomputing '89, pp. 655-664, (1989).
5. R. Strzodka et al., "Cache oblivious parallelograms in iterative stencil computations", Proc. ICS '10, pp. 49-59, (2010).
6. T. Minami et al., "Temporal and Spatial Tiling Method without Redundant Calculations for Three-Dimensional FDTD Method", IPSJ Tran. Advanced Computing Systems, to appear, (In Japanese).
7. T. Hiraishi et al., "Xcrypt: A Perl Extension for Job Level Parallel Programming", Proc. WHIST2012, (2012).