Hindawi Publishing Corporation
International Journal of Antennas and Propagation
Volume 2012, Article ID 851465, 10 pages
doi:10.1155/2012/851465

*Application Article*

# A High-Performance Parallel FDTD Method Enhanced by Using SSE Instruction Set

**Dau-Chyrh Chang,[1] Lihong Zhang,[2] Xiaoling Yang,[3] Shao-Hsiang Yen,[4] and Wenhua Yu[5]**

[1] *Oriental Institute of Technology, Taipei 22061, Taiwan*
[2] *Communication University of China, Beijing 100024, China*
[3] *Pennsylvania State University, University Park, PA 16803, USA*
[4] *Oriental Institute of Technology, Taipei22061, Taiwan*
[5] *2COMU, State College, PA 16803, USA*

Correspondence should be addressed to Dau-Chyrh Chang, dcchang@mail.oit.edu.tw

Received 20 July 2011; Accepted 30 November 2011

Academic Editor: Joshua Le-Wei Li



We introduce a hardware acceleration technique for the parallel finite difference time domain (FDTD) method using the SSE (streaming (single instruction multiple data) SIMD extensions) instruction set. The implementation of SSE instruction set to parallel FDTD method has achieved the significant improvement on the simulation performance. The benchmarks of the SSE acceleration on both the multi-CPU workstation and computer cluster have demonstrated the advantages of (vector arithmetic logic unit) VALU acceleration over GPU acceleration. Several engineering applications are employed to demonstrate the performance of parallel FDTD method enhanced by SSE instruction set.

## 1. Introduction

Since the FDTD method is firstly proposed by Yee in 1966 [1] for solving Maxwell's equations as a type of difference algorithm, it has grown into a popular computational electromagnetic technique after decades of development and become a major electromagnetic simulation tool today. Compared to other computational methods, the FDTD method becomes more and more popular for the practical and complex problems because of its simplicity and flexibility. In addition, the broadband characteristics of the FDTD method allow us to achieve the wideband frequency response in a single simulation. For the electrically large and multiscale problems, it may take too long time for the FDTD simulation to reach the convergent solution. In order to overcome this disadvantage of the FDTD method, there are a large number of publications on the related topics such as conformal technique [2] to enlarge the cell size, subgridding scheme [3] to use local fine mesh, and ADI algorithm [4] to increase the size of time step. On other hand, to use the multicore PC, multi-CPU workstation, and computer cluster to speed up the electromagnetic simulation, the parallel processing based on the (message passing interface) MPI library [5] and (open multiprocessing) OpenMP [6].

With development of the multicore processors, (graphics processing unit) GPU [7] has drawn a lot of attentions in the past ten years since an advanced GPU contains up to several hundreds of computing cores. Since the field update formula in the FDTD method is similar to the format that GPU is used to display acceleration, the FDTD method is introduced to the GPU acceleration in the earlier time. However, to use GPU acceleration, the simulation format has to be in the certain requirement. For example, the data inside memory must be continuous; otherwise, the performance will be degraded dramatically. In the practical problems, the data structure inside the memory varies from one case to another. In addition, the GPU can only use the high-performance memory installed on the GPU card. Any communication between the GPU memory and main memory will cause reducing the simulation performance. Unlike the GPU acceleration, VALU acceleration will fully use the features built inside the regular CPU. Though the VALU unit has existed

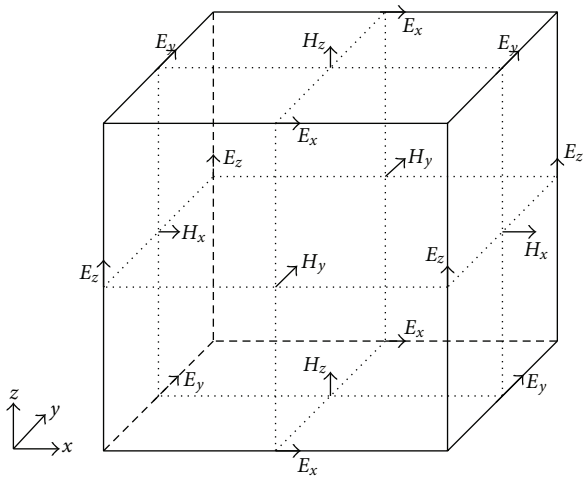



Figure 1: Position of E- and H-fields in the FDTD cell.

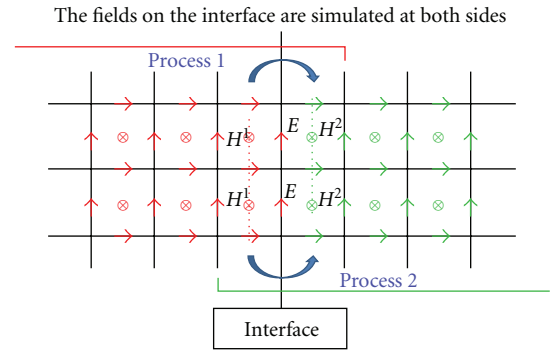



Figure 2: Concept of field exchange between the subdomains in the parallel processing.

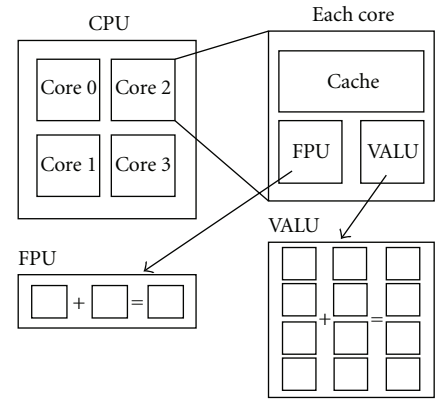



Figure 3: CPU architecture including FPU and VALU.

inside the CPU for many years since the Intel Pentium III. However, it has not used for the acceleration of engineering simulations. Compared to the GPU acceleration, VALU acceleration is more powerful and flexible, and it is the most important that the VALU acceleration is a general platform, and it does not require any extra hardware devices.

The VALU acceleration has been illustrated through several typical examples on both Intel and AMD processors. Since the AMD processor includes more physical cores today than Intel processor. The FDTD simulation accelerated by using VALU units can get more benefit from the AMD than Intel processors. A waveguide filter and patch antenna array are used to validate the VALU acceleration technique. In this paper, we first briefly introduce the parallel FDTD method in Section 2, and then describe the basic architecture of a regular CPU in the Section 3. Section 4 concentrates on the implementation of SSE instruction set in the parallel FDTD code. Section 5 uses several examples to demonstrate the performance improvement of SSE instruction set for the FDTD simulation. Section 6 uses a waveguide filter and patch array to demonstrate the performance improvement of SSE instruction set for the practical problem solving. Finally, we summarize the works in this paper and point out the future works.

## 2. Parallel FDTD Acceleration Method

In FDTD method, Maxwell's equations are discretized into a set of difference equations through the central difference formula. The electromagnetic wave propagation and the interaction of electromagnetic wave with the environment are analyzed through update of the electric and magnetic fields in space and time. The electric/magnetic fields in spatial and time domains can be calculated by the explicit way through its previous value at the same location and the four magnetic/electric fields around it at the half time step earlier as illustrated in Figure 1.

It is observed from Figure 1 that the FDTD method is parallel in nature due to its localized characteristics and suitable for the SSE implementation.

Using this feature, we can achieve an excellent parallel performance when we use the FDTD code on the multiple core processor and computer cluster because it only requires the exchange of information on the interface between the subdomains. In the several popular data exchanging methods [1], the robust one is described in Figure 2.

The MPI library is designed for the communication of distributed processors. The exchanging data in the FDTD simulation is located on the interface between the adjacent processors, as we can see from Figure 2. The parallel efficiency depends not only on the parallel code structure but also the network system. On other hand, the parallel computing on the multicore processors and multi-CPU workstation is based on the OpenMP, which is a shared memory parallel algorithm. OpenMp requires that all the cores or CPUs have the same hostname. In a practical cluster or multi-CPU workstation, MPI and OpenMp are combined to achieve a better simulation performance.

Vector unit has been existed in a regular CPU for many years; however, it was never used in the engineering simulations. In this paper, we describe the implementation approach to apply the vector unit into the FDTD simulation though the SSE instruction set. Compared to the GPU acceleration, the VALU acceleration technique is general platform and does not require any extra hardware devices. Now, we briefly introduce the architecture of regular CPU and then describe how to implement the VALU unit to speed up the FDTD simulation. Each core in the multicore

```
for(i = 0; i <= nx; i ++) {
   vHi_Coeff = _mm_load1_ps(&Hi_Coeff);
   //load single float value to vector
   for(j = 0; j <= ny; j++) {
     vHj_Coeff = _mm_load1_ps(&Hj_Coeff);
      //load single float value to vector
   vEz= ( __m128 *) Ez[i][j];
   vHy= ( __m128 *) Hy[i][j];
   vHx= ( __m128 *) Hx[i][j];
   vHy_minus = ( __m128 *) Hy[i-1][j];
   vHx_minus = ( __m128 *) Hx[i][j-1];
   for(k = 0, vk = 0; k < nz; k += 4, vk ++) {
      vEk_Coeff = _mm_load1_ps(&Ek_Coeff);
      xmm0 = _mm_sub_ps(vHx[vk], vHx_minus [vk]);
      xmm0 = _mm_mul_ps( vHj_Coeff, xmm0);
      xmm1 = _mm_sub_ps(vHy[vk], vHy_minus [vk] );
      xmm1 = _mm_mul_ps(vHi_Coeff, xmm1 );
      xmm0 = _mm_sub_ps(xmm1, xmm0 );
      xmm1 = _mm_mul_ps(vEk [vk], vEk_Coeff);
      vEk [vk] = _mm_add_ps(xmm1, xmm0);
} }}
```

Algorithm 1

processor has its own cache, FPU and VALU, as shown in Figure 3. Unlike the FPU, the VALU allows us to operate on four data at the same time. VALU unit is a unit that allows us to arrange for 32-bit floating numbers and operate the four numbers at the same time. By the way described, we use the VALU through the SSE instruction set to accelerate the parallel conformal FDTD code.

Next, we use the electric field component update as an example to explain how the FDTD code is accelerated by using VALU unit. For example, the electric field $E_z$ can be expressed as follows [8, 9]:

$$\begin{aligned} E_z^{n+1}\left(i,j,k+\frac{1}{2}\right) &= E_z\text{\_Coeff}\cdot E_z^{n}\left(i,j,k+\frac{1}{2}\right) \\ &\quad + H_y\text{\_Coeff} \\ &\quad \cdot\left[H_y^{n+1/2}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)\right. \\ &\qquad \left. -H_y^{n+1/2}\left(i-\frac{1}{2},j,k+\frac{1}{2}\right)\right] \\ &\quad - H_x\text{\_Coeff} \\ &\quad \cdot\left[H_x^{n+1/2}\left(i,j+\frac{1}{2},k+\frac{1}{2}\right)\right. \\ &\qquad \left. -H_x^{n+1/2}\left(i,j-\frac{1}{2},k+\frac{1}{2}\right)\right], \end{aligned} \tag{1}$$

where $E_z$_Coeff, $H_x$_Coeff, and $H_y$_Coeff are coefficients in the field update equations [4]. Using the following procedure VALU can be used to compute four $E_z$ at the same time to accelerate the simulation, which is described as follows.

(i) Load the coefficient of the magnetic field $H_y$ into the SSE registers.

(ii) Load the coefficient of the magnetic field $H_x$ into the SSE registers.

(iii) Convert the float pointer to the SSE 128 bit pointer.

(iv) Calculate the difference of magnetic fields.

(v) Multiply the difference of magnetic fields and their coefficients.

(vi) Calculate the contribution of magnetic fields to the electric fields.

(vii) Multiply the previous electric fields and their coefficients.

(viii) Calculate the electric fields and write them to memory (Algorithm 1).

A 3-D array in the FDTD code is allocated using the *malloc* function in C language, and *_aligned_malloc* function in SSE in this paper. For example, if we need a 3D array *array_name* [*x_size*, *y_size*, *z_size*], we can first define a 1D array *array_name_tmp* [*N*] whose size is $N = x\_size * y\_size * z\_size$, and then map the 1D memory address to 3D array *array_name*. The pseudocode segment is demonstrated as below (see Algorithm 2).

In the C programming language, the data inside the memory is contiguous in *y*-*z* plane. Suppose that the *y_size* and *z_size* are equal to 8, and the data structure of the array *array_name* in the *y*-*z* plane is shown in Figure 3. The memory addresses of the data elements (0,0,0), (0,0,1)…(0,0,7) are continuous likewise, the addresses of (0,1,0), (0,1,1)…(0,1,7) are continuous too. The address of (0,0,7) is continuous and is followed by the element (0,1,0). When we calculate the electric and magnetic fields in the *y*-*z* plane, we only need to know the address of the first element and the total number of elements, as shown in Figure 4.

```
//allocate the 1D memory
array_name_tmp = (float* )_aligned_malloc(
    sizeof (float) * x_size * y_size * z_size, 16);
array_name = (float *** )_aligned_malloc (
    sizeof (float** ) * x_size, 16);
    for(i = 0; i < x_size; i++) {
        array_name [i] = (float ** )_aligned_malloc
            (sizeof (float* ) * y_size, 16);
        for(j = 0; j < y_size; j++) {
            //map the 1D memory address to 3D array
            map_address = i * y_size * z_size + j * z_size;
            array_name [i][j] = &array_name_tmp [map_address];
}}
```

Algorithm 2

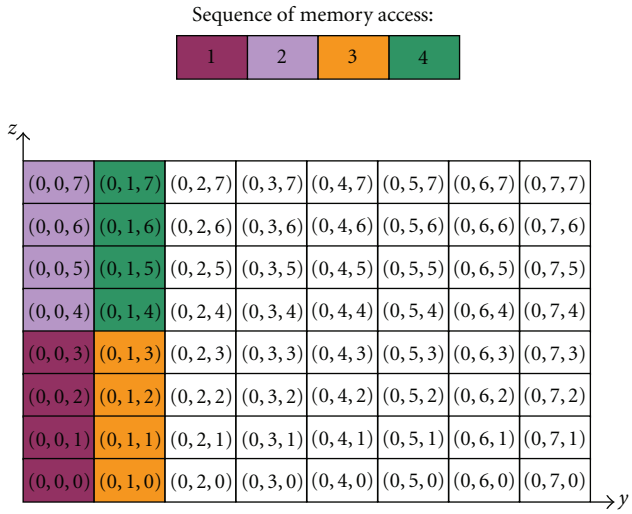



Figure 4: Data structure in the $y$-$z$ plane.

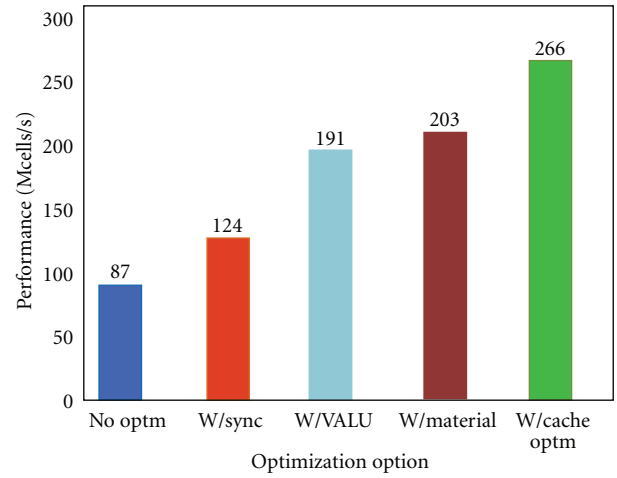



Figure 5: Performance improvement of the parallel conformal FDTD code using VALU acceleration technique.

Using the VALU and the procedure mentioned above can significantly accelerate the multiplication operation compared to the traditional FDTD code. In any case, the data continuity in the FDTD code and memory bandwidth is the most important factors for the VALU acceleration. To evaluate the performance of the FDTD code, we define the performance as follows:

$$\text{Performance} = \frac{\left(N_x \times N_y \times N_z\right) \times \text{Number\_of\_timesteps}}{\text{Simulation\_time(second)}}. \quad (2)$$

For example, the performance of a regular parallel conformal FDTD code on an Intel Core i7-965 3.2 GHz is 87 Mcells/sec, as shown in Figure 4. If we apply the optimized synchronization technique on the parallel FDTD code, the performance increases to 124 Mcells/sec. When we implement the VALU acceleration technique on the parallel conformal FDTD code, the performance increases to 191 Mcells/sec. Besides the field update module, further optimization on the material list table, cache usage, dispersive medium, and near-to-far field transformation will also significantly improve the FDTD performance, as shown in Figure 5.

Finally, we use the parallel conformal FDTD code to simulate an ideal test case that is a hollow box with the simplest excitation and output, and its domain is truncated by using the perfect electric conductor (PEC). We ran the problem with the different sizes using the regular FDTD code (Intel Core i7-965), and the FDTD code with the VALU (Intel Core i7-965 with 4 VALUs, the total memory bandwidth is 32 GB). The simulation summary is plotted in Figure 6. It is observed from Figure 5 that the peak performance of the VALU FDTD code is four times faster than the FDTD code on a regular CPU for the ideal test case.

## 3. FDTD Performance Investigation

In this part, we employ a typical example to demonstrate the performance improvement by using the SSE instruction set on the multi-CPU workstations. The acceleration factor here reflects the simulation acceleration gained by using the SSE implementation. A typical test case includes only voltage

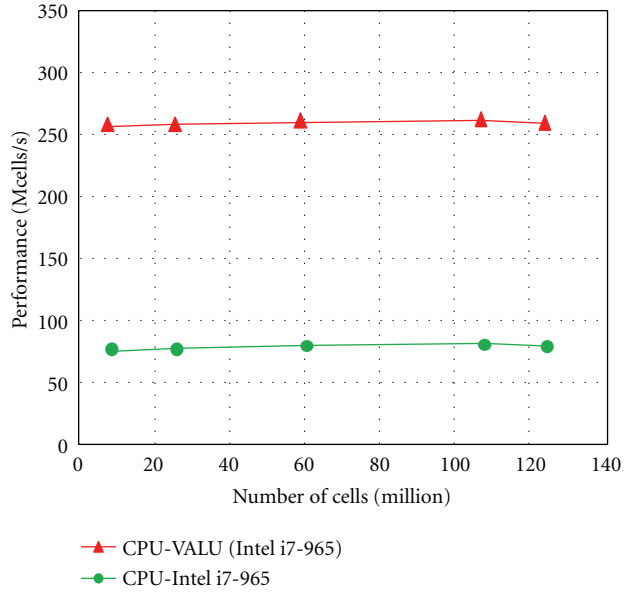



Figure 6: FDTD performance on CPU and CPU value for the ideal test case.

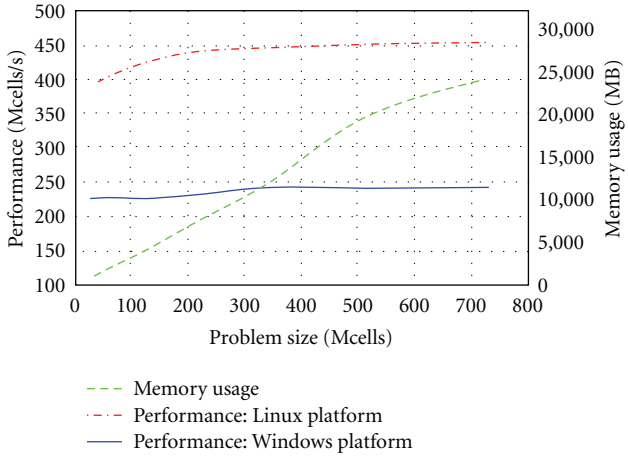



Figure 7: Performance of parallel FDTD enhanced by using SSE instruction set on a 2-CPU workstation platform for the different operating systems.

source and field output at a specific point. The empty box truncated by the PEC boundary is discretized into from $300 \times 300 \times 300$ (27 Mcells), $400 \times 400 \times 400$ (64 Mcells), $500 \times 500 \times 500$ (125 Mcells), $600 \times 600 \times 600$ (216 Mcells), $700 \times 700 \times 700$ (343 Mcells), $800 \times 800 \times 800$ (512 Mcells), and $900 \times 900 \times 900$ (729 Mcells) uniform cells, respectively. The numerical experiments were carried out on a 2-CPU workstation that is installed with two AMD Opteron 6128 2.0 GHz processors. The simulation performance is [summarized in Figure 7].

We compare the FDTD simulation performance on one 18-node high-performance cluster and a popular nVidia Tesla C1060 GPU. The cluster with 36 Intel Xeon X5550 CPUs is installed in 18 nodes, which are connected through

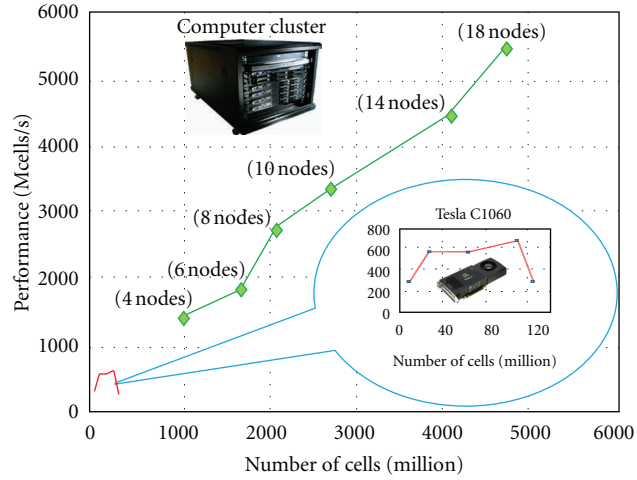



Figure 8: Performance of parallel FDTD enhanced by using SSE instruction set on a cluster platform and comparison to the GPU acceleration.

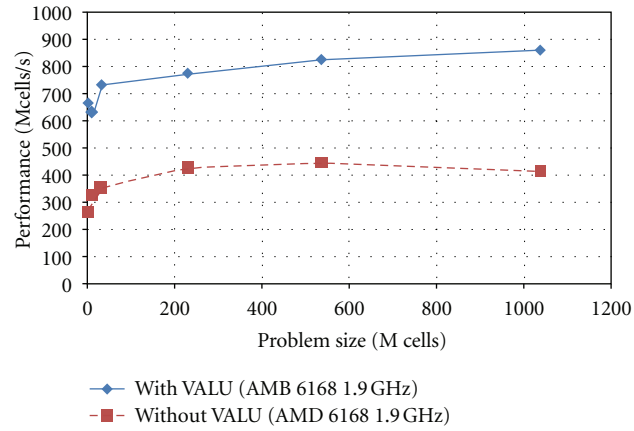



Figure 9: FDTD performance of the VALU acceleration on one 4-CPU workstation.

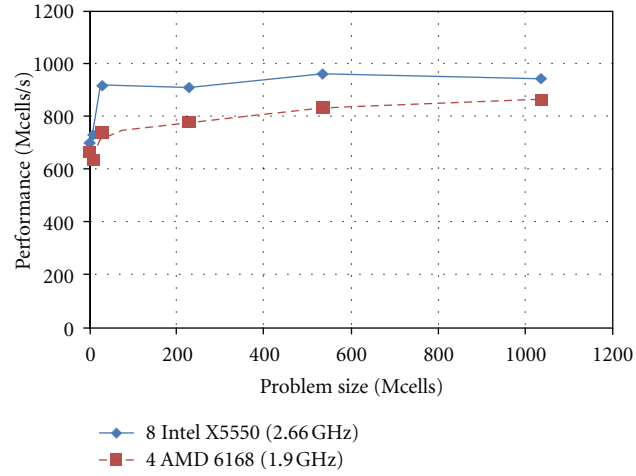



Figure 10: Performance comparison of parallel FDTD code on a workstation with 4 AMD CPUs and a cluster with 4 nodes (8 Intel CPUs).

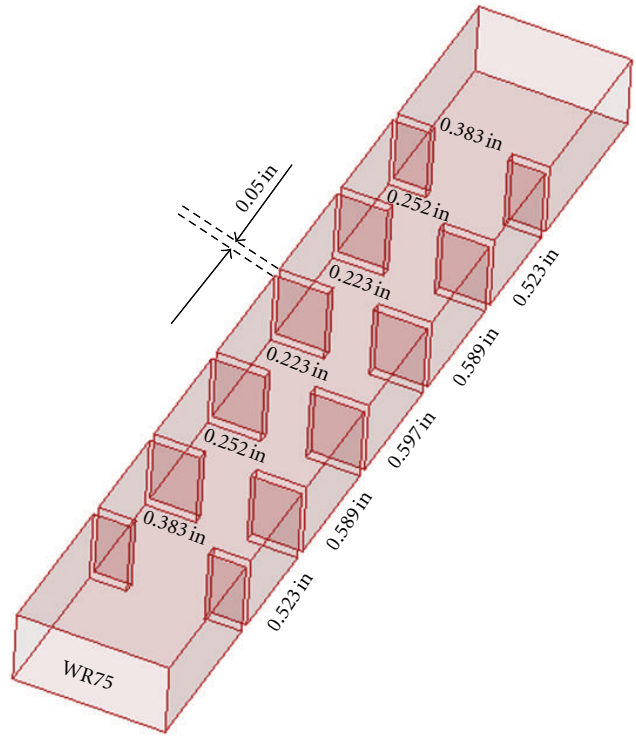



Figure 11: Configuration of waveguide filter with 5 open cavities.

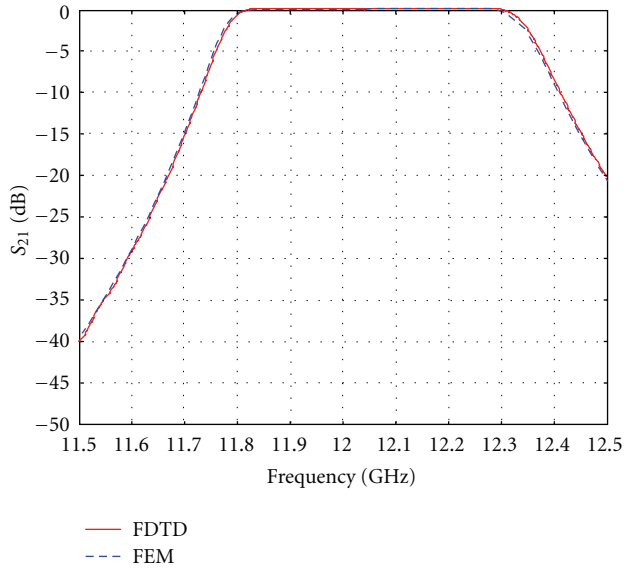



Figure 12: Transmission coefficient of waveguide filter generated by using enhanced FDTD method and FEM.

the 10G Ethernet system. The performance comparison between two platforms is summarized in Figure 8. For the ideal test case, one nVidia Tesla C1060 GPU has the similar performance to two Intel X5550 CPUs.

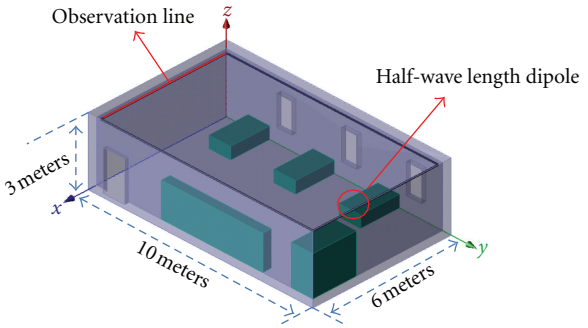
Next, we investigate the simulation performance of parallel FDTD code on a workstation installed with 4 AMD Opteron 6168 1.9 GHz CPUs. The 6-layer PML in this case is used to truncate the computational domain. The performance of VALU acceleration is shown in Figure 9.

We can observe from Figure 8 that the VALU can improve the simulation performance 200% for the practical problem with 6-layer PML truncation. Next, we compare the performance of the workstation with 4 AMD Opteron 6168 CPUs and a cluster with 8 Intel Xeon X5550 CPUs for simulation of the same problem. The 4-node cluster with 8 Intel CPUs is connected with each other using the 10G Ethernet. The performance comparison is plotted in Figure 10, which shows that 4 AMD CPUs have the similar performance tot 8 Intel CPUs. Since each AMD CPU includes 12 cores and 12 VALUs, the AMD workstation has total 48 VALUs; however, each Intel X5550 CPU only has 4 cores and 4 VALUs.

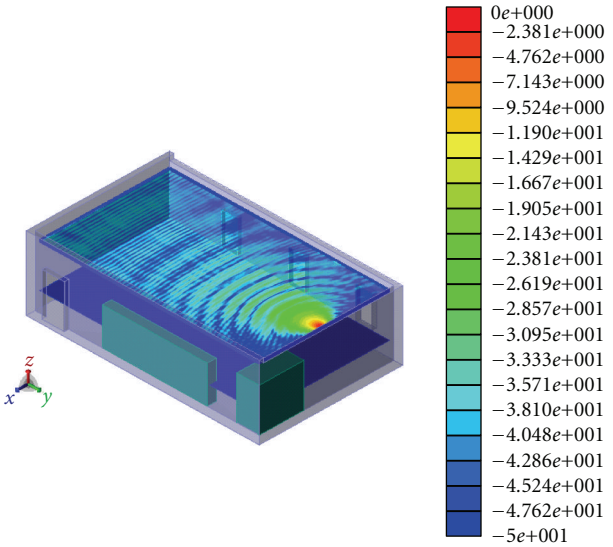
In the practical problems, the simulation factors such as output types, dispersive media, and near-to-far field transformation will influence the SSE performance due to the discontinuous data structure inside memory. However, it can be improved by optimizing the cache hit ratio.

## 4. Engineering Applications

In this part, we use the parallel FDTD code accelerated by using the SSE instruction set to simulate a waveguide (WR75) filter problem [8], as shown in Figure 11. The filter includes five cavities and is excited by $TE_{10}$ mode at one end. The output parameter transmission coefficient for the $TE_{10}$ mode is measured at another end. The purpose is to

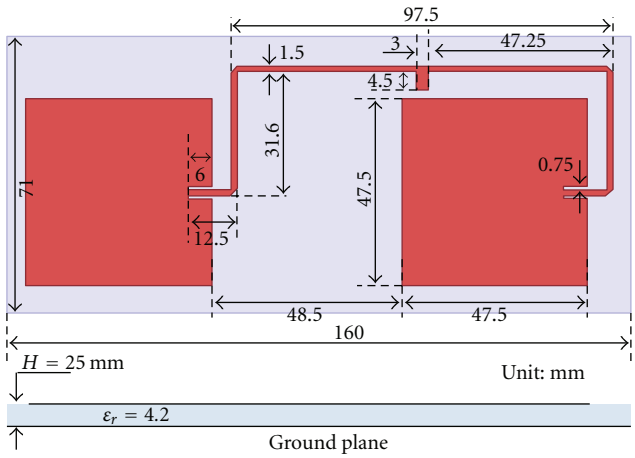



(a) Problem dimension

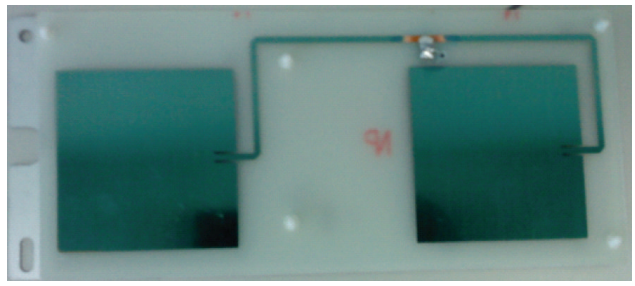



(b) Field distribution at 3 GHz inside the patient room.

Figure 13: A typical patient room as an example to demonstrate the FDTD performance enhanced with SSE instruction set.




(a) Top and side views of the patch array



(b) Prototype of the patch array

Figure 14: Configuration of a patch array and the prototype. (a) Configuration of the patch array; (b) prototype of the patch array.

Table 1: Parallel FDTD performance with the SEE acceleration.

| | FDTD with SSE acceleration | FDTD without SSE acceleration |
|---|---|---|
| Workstation | 2 × AMD Opteron 6128 2.0 GHz | |
| Memory usage | 37 MB | 37 MB |
| Simulation time | 145 sec | 345 sec |

Table 2: Performance comparison of parallel FDTD method on 4-CPU workstation and high-performance cluster.

| Platform | CPU type | Network | Simulation time |
|---|---|---|---|
| Workstation (4 CPUs, 64 GB RAM) | AMP Opteron 6168 1.9 GHz($795 each) | No network | 319 min. (with hardware acceleration and NUMA) |
| 2 CPUs (24 GB RAM) | Intel Xeon X5570 2.93 GHz($1465 each) | No network | 1720 min. (with hardware acceleration, no NUMA) |
| 128 CPUs (1536 GB RAM) | Intel Xeon X5570 2.93 GHz($1465 each) | Infiniband | 29 min.37 sec. (with hardware acceleration no NUMA) |

investigate the performance of SSE acceleration on the 2-CPU workstation for the practical problem.

For the sake of comparison, we use the FEM method [9] to simulate the same problem and plot the results in Figure 11. It is evident from Figure 12 to observe the good agreement. It is worthwhile to mention that the results are same with and without the SSE acceleration.

The parallel FDTD performance with the SSE acceleration is summarized in Table 1. It is observed from Table 1 that the SSE can accelerate the FDTD code 2.37 times for this practical problem.

Next, we use the parallel FDTD performance with the SSE acceleration to simulate a patient room, as shown in Figure 13, in which the field distribution is generated by a dipole antenna, as shown in Figure 13. The room dimension is $10 \times 5 \times 3$ miters, which is discretized into $3333 \times 1666 \times 1000$ uniform cells. The simulation performance is summarized in Table 2. It is observed from Table 2 that a workstation with 4 AMD Opteron 6168 1.9 GHz CPUs has the similar performance as the 10 Intel Xeon X5570 2.93 GHz CPUs. It is worthwhile to mention that, without the SSE acceleration, one AMD AMD Opteron 6168 1.9 GHz CPU has the same performance as one Intel Xeon X5570 2.93 GHz CPU.

The simulation on the cluster platform was provided by an independent third party. The cluster does not support the NUMA architecture. Otherwise, the cluster performance should be better.

Finally, we use the parallel FDTD code enhanced by SSE instruction set to simulate a patch array [10] for IEEE802.11b. The array includes 2 patch antenna elements and arrayed in E-plane. The simulation model and hardware implementation are shown in Figure 14. The substrate is FR4 with relative dielectric constant 4.2. The size of array is 71 mm (width) by 160 mm (length) and by 0.8 mm (thickness). The substrate is 2.5 mm above ground plane. The array is measured at Communication Research Center of Oriental Institute of Technology by

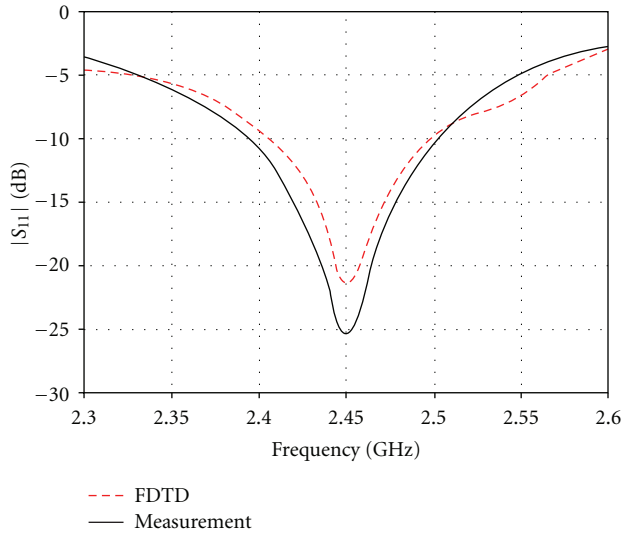
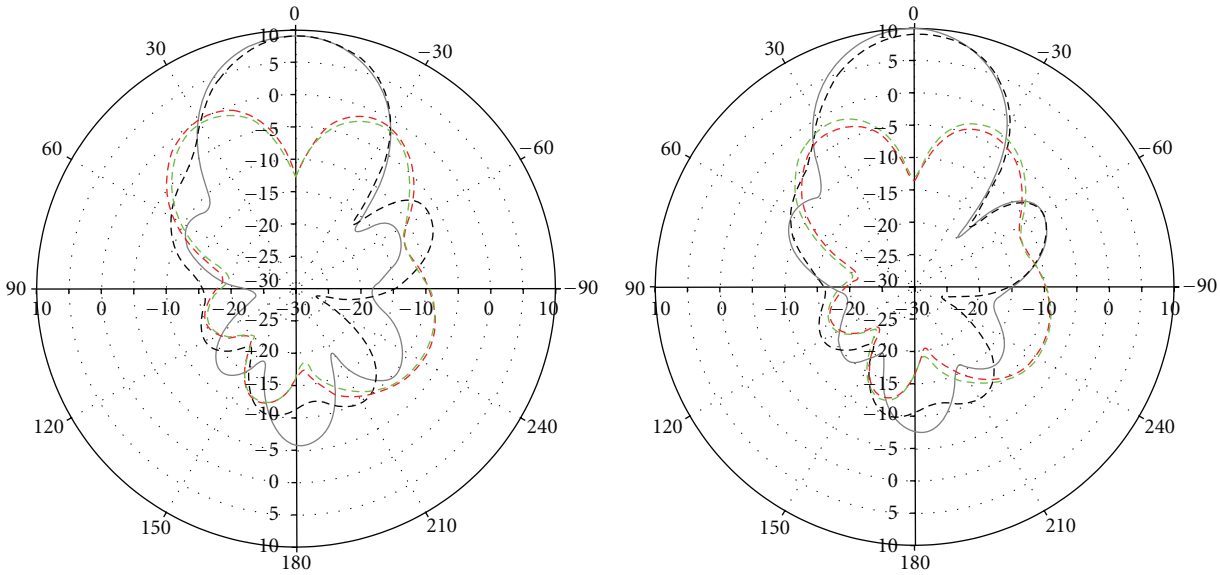



Figure 15: Return loss of the antenna array.

vector network analyzer for return loss and near field range for power pattern. It is observed from Figures 15 and 16 for the return loss and power patterns of the array the good agreement between the FDTD simulation and measurement. The E-plane power patterns at frequencies 2.41 GHz, 2.44 GHz, and 2.46 GHz are shown in Figure 16. The copolar patterns are in agreement for both FDTD simulation and measurement. The minor difference of x-polar patterns for both FDTD simulation and measurement may be caused by the improper hardware implementation or measurement.

## 5. Conclusions

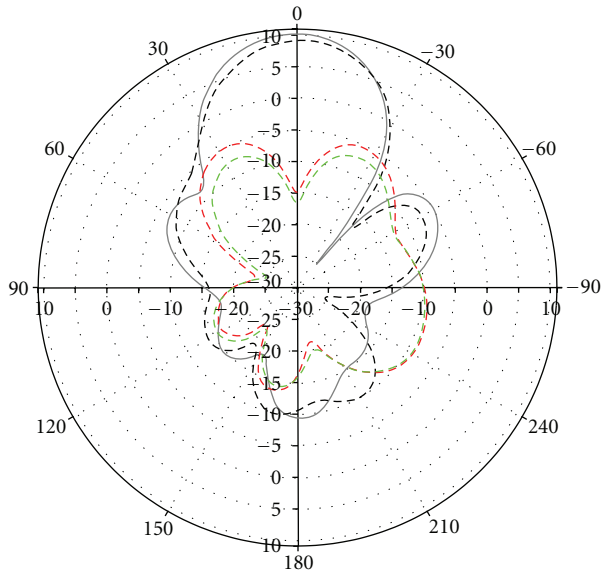
In this paper, we propose a new hardware acceleration technique based on the SSE instruction set and give an implementation technique. The result shows that this technique



(a) Far field power pattern at 2.41 GHz.



(b) Far field power pattern at 2.44 GHz.




(c) Far field power pattern at 2.47 GHz.

FIGURE 16: E-plane far field power pattern of the patch antenna array at 2.41 GHz, 2.44 GHz, and 2.47 GHz.

dramatically improves the computing efficiency without any extra hardware investment and provides an efficient and economical technique for the electromagnetic simulations.

## References

[1] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, no. 5, pp. 302–307, 1966.

[2] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, Mass, USA, 2nd edition, 1999.

[3] https://computing.llnl.gov/tutorials/openMP/.

[4] A. Elsherbeni and V. Demir, *The Finite Difference Time Domain Method for Electromagnetics: With MATLAB Simulations*, SciTech Publisher, Raleigh, NC, USA, 2009.

[5] Intel corporation, Intel Architecture Optimization Reference Manual, http://www.intel.com/design/pentiumii/manuals/245127.htm.
[6] A. Taflove and S. Hagness, *Computational Electrodynamics the Finite-Difference Time Domain Method*, Artech House, Norwood, Mass, USA, 2005.
[7] W. Yu, X. Yang, Y. Liu, R. Mittra, and A. Muto, *Advanced FDTD Method: Acceleration, Parallelization and Engineering Applications*, Artech House, Norwood, Mass, USA, 2011.
[8] M. Yu, "Power-handling capability for RF filters," *IEEE Microwave Magazine*, vol. 8, no. 5, pp. 88–97, 2007.
[9] J. M. Jin, *The Finite Element Method in Electromagnetics*, John Wiley & Sons, New York, NY, USA, 2002.
[10] C. Balanis, *Antenna Theory*, John Wiley & Sons, New York, NY, USA, 1987.