INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING, *SERIES B*
Volume 5, Number 1-2, Pages 57–65



# AN OPTIMIZED CALCULATION METHOD FOR 3D-FDTD PARALLELIZATION

JIANG XIE, ZHONGHUA ZHOU, BING HE, HUIRAN ZHANG, AND DONGBO DAI

**Abstract.** The finite-difference time-domain (FDTD) has been commonly used for electromagnetic field simulation. However, parallelization of FDTD usually consumes large amount of memories and very long time, especially when the target domain is in 3D space. And the communication between calculating cores can be the bottleneck of the parallelize performance. In this paper we propose an optimized method named Crossing Calculation Method to accelerate FDTD simulation. Crossing Calculation Method can overlap the communication time which includes data exchanging time and synchronization time between neighborhood calculating nodes. The result shows that Crossing Calculation Method has linear speedup along with the increasing of calculating cores.

**Key words.** Parallel Computing, Finite-Difference Time-Domain, Crossing Calculation Method and Communication Time Overlapping.

## 1. Introduction

The finite-difference time-domain (FDTD) method [1] is a popular numerical computation method in electromagnetic field distribution analysis. Researchers have done a lot of work to complete this method. Although some people have promoted some absorbing boundary conditions theories [2, 3, 4] to reduce the simulation area, the simulation target area is still very large, especially when it comes to a three-dimensional problem. The size of grid grows cubically so that the use of memory is too large to simulate by a single computer and the computing time can be very long. It is very important to make the simulation fast. Many researchers have contributed a lot of useful work [6, 7, 8] on accelerating FDTD method.

Parallelization is usually used to make the simulation faster. The Message Passing Interface (MPI) [5] is a very common library for parallelization in CPU clusters. It is very convenient to use MPI to parallel FDTD simulation on CPU [9]. Recently GPU parallelization has become a hot issue in high performance calculation area, researches have promoted some GPU parallelization method [10, 11, 12] and at the same time some people have combined GPU and CPU [13, 14] to make the FDTD simulation much more faster.

Those methods can better improve the simulation speed of FDTD, however in the MPI based parallelization method [15, 16, 17, 18] when the calculation core number increases the speedup increases slower because the communication time which includes data exchanging time and synchronization time becomes a bottleneck. Sometimes it takes more time to communication than calculation. Figure 1 displays the speedup result of the classic parallelization method. Figure 2 shows the percentage of the communication time.

We promote an optimized calculation method named Crossing Calculation Method (CCM) to accelerate the parallelization. CCM overlaps the data exchanging time by controlling the calculation nodes calculating from different directions. The experiment results of our method in a CUP cluster with 160 Intel(R) Xeon(R) 2.53GHz

Received by the editors January 15, 2014 and, in revised form, March 21, 2014.
2000 *Mathematics Subject Classification.* 35R35, 49J40, 60G40.

cores and 160 GB memory shows that CCM has linear speedup ratio along with the increasing of calculating cores.

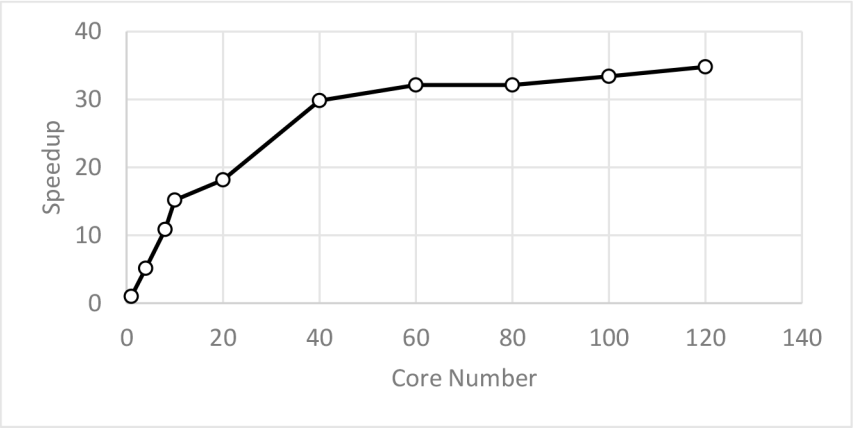


FIGURE 1. Speedup of classic FDTD parallelization method. In this figure, with the increasing of the calculation core number the increase rate of speedup declines. It mainly because the communication between cores becomes the bottleneck of the performance.

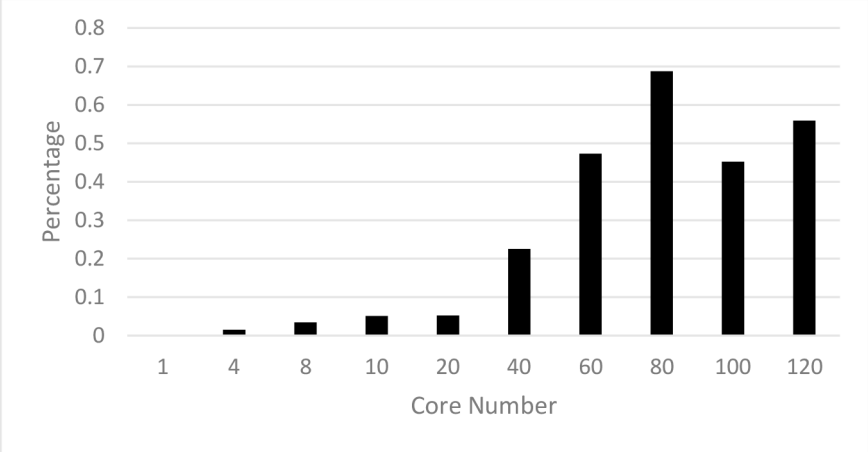


FIGURE 2. The percentage of the communication time of classic FDTD parallelization method. In this figure, the percentage of communication time increases as the core number increases and sometime it even takes more than 50%.

## 2. CCM

**2.1. FDTD Parallelization.** FDTD method is easy to parallelize because the whole calculation space region can be separated into some different small regions. The sub regions are relative independent because only the neighboring regions share some data. Figure 3 displays a basic architecture of the parallelization.