Phi Coprocessor And Acceleration Techniques For Finite Difference Time Domain Methods

Wenhua Yu
2COMU, Inc.
4031 University Dr. Suite 100
Fairfax, VA 220301
E-mail: wenyu@2comu.com

***Abstract*— in this paper, we introduce the architecture of Phi Coprocessor, programming techniques and acceleration techniques in the finite different time domain (FDTD) methods. Phi Coprocessor can be used as a regular CPU and run the EM code optimized for regular CPUs such as Intel Xeon E5 or AMD Opteron 6300 with slight code modifications. The examples will be for the acceleration of the parallel FDTD methods.**

I. INTRODUCTION

Phi coprocessor is a general numerical acceleration platform, which can be used to accelerate the computational electromagnetic methods [1-4] such as FDTD, finite element method (FEM), method of moments (MoM) and so on. Phi Coprocessor includes 60 compute cores, four hardware threads per core, two pipelines, 32 512-bit wide vector registers which hold 16 singles or 8 doubles, up to 16GB 16-channel GDDR5 RAM, and 320GB/s Memory Bandwidth. Similar with GPU, Phi coprocessor is used to enhance the data processing capacity of regular CPU using the faster memory and many built-in cores, as shown in Fig. 1.

(a) Phi 5110P coprocessor (b) NVidia Tesla GPU card

Fig. 1. Typical Intel Phi coprocessor and NVidia GPU card

Different from GPU (www.nvidia.com), Phi Coprocessor (www.intel.com) card can be handled as a Linux cluster node and each one in Phi coprocessor has an on-board flash device that loads the coprocessor operating system (OS) on boot and can be monitored by an optional cluster monitoring software Ganglia (http://ganglia.info/). Unlike vector unit inside Intel and AMD CPUs, which is operated by SSE or AVX, Phi Coprocessor programming uses MIC (Many integrated Core, 512-bit SIMD) instructions. Phi's loop data bus and ring interconnection architecture allows Phi to fast exchange the information between the compute cores and memory [5], as shown in Fig. 2. Same as the NVidia GPU card, Phi coprocessor supports GDDR5 memory and the memory bandwidth between Phi and the host memory is up to 8GB per second.

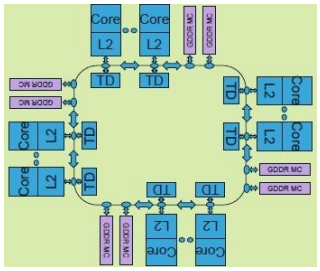

Fig. 2 Ring interconnection architecture includes 8 data MCs (two channels) and 4 bytes/channel, which has total bandwidth of 5.0GT/s (320GB/s).

One code can be executed on the host CPU, or on both the host CPU and Phi at the same time, or on the Phi coprocessor only that is totally different from GPU, as show in Fig. 3.

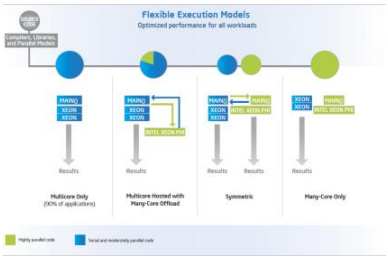

Fig. 3. Code processing with CPU and Phi coprocessor

II. CODE DEPVELOPMENT TECHNIQUES

The code processing on the Phi coprocessor has four levels, namely, card level, core level, thread level and vector unit level. We use the FDTD method to demonstrate how to develop the parallel code on the Phi coprocessor. The FDTD job assignment concept is demonstrated in Fig. 4. If the storage of a 3-D array in the memory is continuous along the z-direction, we divide the data into 60 blocks in the x-y plane, which is equal to the number of cores in the Phi coprocessor. Select one column in an individual block and assign it to two threads, all cores will be coalesced and each

thread will work on one-half column of the selected data. The vector unit will work on 16 adjacent data at the same time and generate 16 results in each cycle. The graphical job assignment procedure is shown in Fig. 5.

```
void FDTDUpdateE()
{
   int n = (nx + 1) * (ny + 1);  //number of blocks
   int nthreads, nsize;

   #pragma omp parallel      //Tell code that for parallel processing
   {
      #pragma omp single     //Thread coalesced
      {
         nthreads = omp_get_num_threads();  //Get the block location
         if (n % nthreads) nsize = n / nthreads;
         else nsize = n / nthreads + 1;
      }
      int idthread = omp_get_thread_num();  //Get the thread ID
      int n1, n2;
      n1 = idthread * nsize;
      n2 = (idthread + 1) * nsize;
      if (n2 > n) n2 = n;
      FDTDUpdateECore512(n1, n2);      //Parallel processing
   }
```

Fig. 4 A parallel FDTD processing on the Phi coprocessor

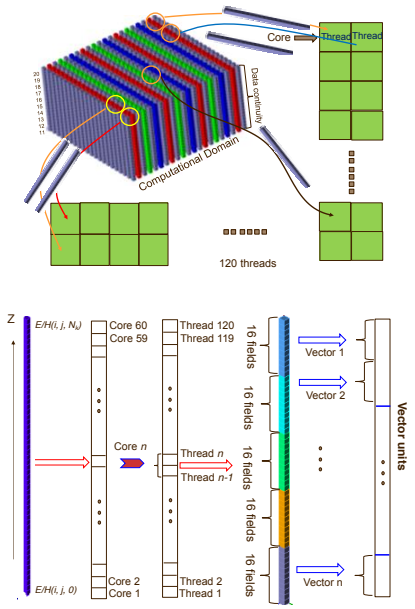

Fig. 5. Graphical explanation of job assignment for the parallel FDTD method

III. NUMERICAL RESULTS

In this section, we use the parallel FDTD code based on the Phi coprocessor to demonstrate its performance. The host computer incudes two Intel Xeon E5-2640 v2 CPUs with 32GB DDR3 RAM and one 5110P Phi coprocessor is mounted on the host through PCI-16. The Phi coprocessor is installed with 8 GB GDDR5 RAM. We use a typical example, an empty box truncated by the PEC boundary, to demonstrate the performance of a Phi coprocessor. The problem size that we first test is 1.29GB and the performance is 1,200 million cells per second. The performance on a single Phi card can be easily achieved to 1,200 million cells per second and the performance increases up to 1350 million cells per second when the problem size increases to 7.2GB, as shown in Fig. 6. We also demonstrate the performance of Phi coprocessor for the small problems such as 0.2 million cells and we cannot observe the performance down slightly but it is still very good compared to the large problems, as shown in Fig. 7.

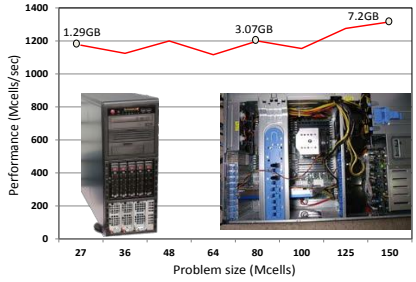

Fig. 6. Performance of 5110P Phi coprocessor for the parallel FDTD code with regular size of problems.

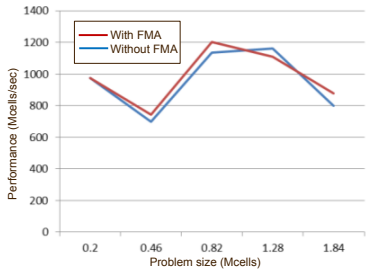

Fig. 6. Performance of 5110P Phi coprocessor for the parallel FDTD code with small size of problems.

IV. CONLUSIONS

Phi coprocessor is used to accelerate the finite difference time domain method using its many core architecture and 512-bit vector units. In contrast to the GPU acceleration, Phi coprocessor acceleration is more general and supports OpenMP. If a source code is compiled on Intel Xeon E3 or E5 CPU, it can be run directly on a Phi coprocessor.

REFERENCES

[1] A. Taflove and S. Hagness, *Computational Electromagnetics: The Finite-Difference Time-Domain Method*, 3rd ed., Artech House, Norwood, MA, 2005.

[2] W. Yu, X. Yang, Y. Liu, et al., *Parallel Finite Difference Time Domain Method*, Artech House, Norwood, MA, 2006.

[3] W. Yu, X. Yang and W. Li, *VALU, AVX, GPU Acceleration Techniques for Parallel Finite Difference Time Domain Methods*, SciTech Publisher Inc., Raleigh, NC, 2013.

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

[5] http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html