Phi Coprocessor Acceleration Techniques for Computational Electromagnetics Methods

Xiaoling Yang and Wenhua Yu

2COMU, Inc.
4031 University Dr., Suite 100, Fairfax, VA 22030, USA
ybob@2comu.com, wenyu@2comu.com

Abstract ─ In this paper, we introduce the architecture of Intel Xeon Phi coprocessor, programming and acceleration techniques in the computational electromagnetic methods. This paper describes how to develop the acceleration codes based on the Intel Xeon® Phi coprocessors in the parallel format for general computational electromagnetics methods. We also present management of the cores and threads in the Intel Xeon Phi coprocessors to accelerate computational electromagnetics simulations. The Intel Phi coprocessor can be used as a regular CPU and shares the same source codes for Intel Xeon E3 and E5 CPUs with a different compilation option. The examples shown here are for acceleration of the parallel FDTD methods. The Intel Xeon Phi coprocessor is not a GPU and is a general hardware acceleration simulation platform.

I. INTRODUCTION

Intel Xeon Phi coprocessor is a general numerical acceleration platform, which can be used to accelerate the computational electromagnetics methods such as FDTD [1-4], FEM [5], MoM [6], and so on. The Intel Xeon Phi coprocessor (price from $1,600 to $4,500) includes 60 compute cores, four hardware threads per core, two pipelines, 512-bit SIMD instructions, 32 512-bit wide vector registers which hold 16 singles or 8 doubles, up to 16 GB 16-channel GDDR5 RAM, and 320 GB/s memory bandwidth. One motherboard can hold maximum four Intel Xeon Phi coprocessors, as shown in Fig. 1.

A Phi coprocessor card can be handled as a Linux node, and each one has an on-board flash device that loads the coprocessor OS (Operating System) on boot and can be monitored by an optional cluster monitoring software Ganglia (http://ganglia.info/) [7,8], as shown in Fig. 2. The Phi coprocessor programming uses the Intel MIC (Many Integrated Core) instructions. One code can be executed on the host CPU, on both the host CPU and Phi coprocessor at the same time, or on the Phi coprocessor only that is totally different from GPU, as shown in Fig. 3.

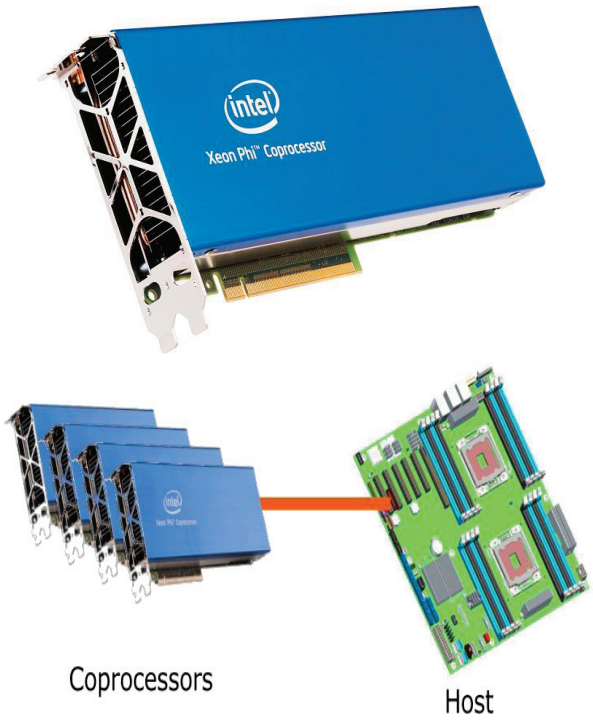


Fig. 1. Intel Xeon Phi coprocessor and cluster artwork (www.intel.com) Copyright©Intel.

Submitted On: March 10, 2014
Accepted On: August 8, 2014

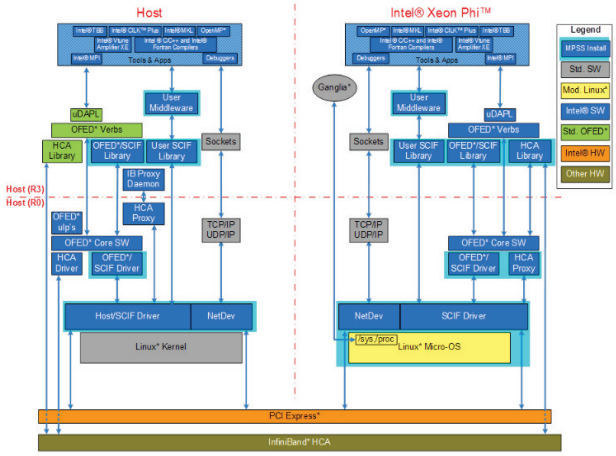


Fig. 2. Intel Xeon Phi coprocessor architecture (www.intel.com) Copyright©Intel.

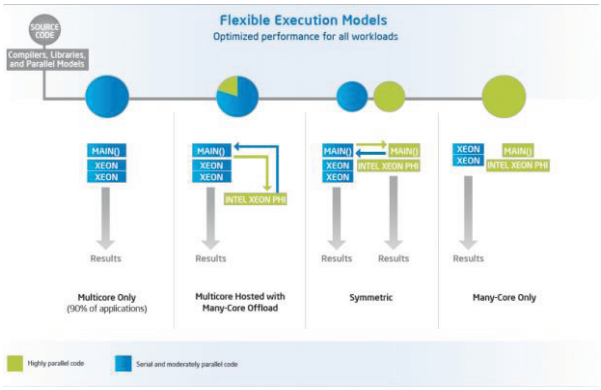


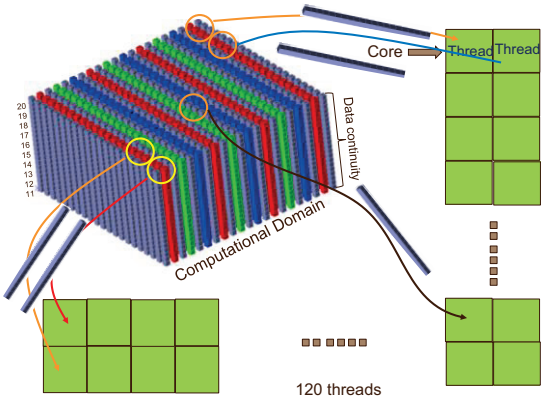
Fig. 3. Code processing with the host CPU and Phi coprocessor (www.intel.com) Copyright©Intel.

II. CODE DEVELOPMENT TECHNIQUES

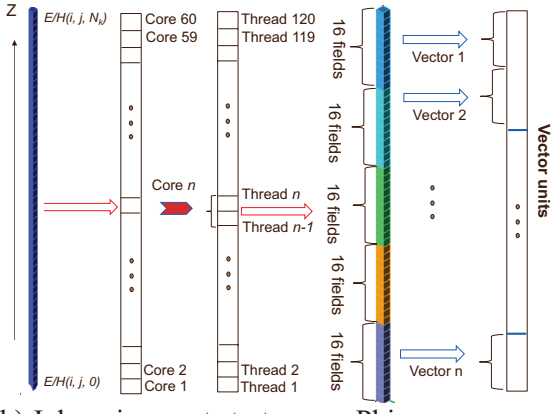
The code processing on the Phi coprocessor has four levels; namely, card level, core level, thread level and vector unit level. The code excitation can be assigned to different cards, cores, threads and vector units, as shown in Fig. 4. We use the parallel FDTD method to demonstrate how to develop the parallel code on the Phi coprocessor. 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 job assignment procedure is shown in Fig. 5.

```
#pragma omp target device(0)                    Cards
#pragma omp teams num_teams (60) num_threads (4)
{
    # pragma omp distribute                     Cores
    for (int i = 0; i < 2048; i++)
    {
       #pragma omp parallel for                 Threads
       for (int j = 0; j < 512; j++)
       {
          # pragma omp simd                     Vector units
          for (int k=0; k<32; k++)
          {
             foo(i,j,k);
          }
       }
    }
}
```

Fig. 4. A sample core processing on the Intel Xeon Phi coprocessor.



(a) Job assignment for threads on Phi coprocessor



(b) Job assignment strategy on Phi coprocessor

Fig. 5. Job assignment method for the parallel FDTD method.

In both the Finite Element Method (FEM) and the Method of Moments (MoM), solving matrix equations is most time consuming. Now, we investigate how to use the Phi coprocessor to calculate the multiplication of two matrixes. We begin with the following equation:

$$C_{nl} = A_{nm} B_{ml}\,, \tag{1}$$

where C_{nl}, A_{nm} and B_{ml} are matrixes and the subscripts indicate the number of rows and columns of the corresponding matrixes. If we define a 1-D array and map the 1-D array to a 2-D array that is used to allocate memory for the matrixes in (1), one matrix with 5×3 elements can be mapped from a 1-D array with 15 elements, as shown in Fig. 6. It is obvious from Fig. 6, that the data of the matrix C is continuous along its row index m.

If we follow the idea described above to allocate the matrixes A and B, the column of matrix B will not be continuous, in turn, the multiplication operation of the matrixes A and B is very low efficient. It is a well-known fact that we need to make a transpose of B to speed up the matrix multiplication. If we calculate the matrix multiplication on the Phi coprocessor, it is observed that the calculation for the smaller A and B is much faster than the larger A and B. This happens because the elements of smaller A and B can be held in the cache to increase the cache hit rate. The cache hit rate becomes lower when the matrixes A and B become larger.

Fig. 6. Mapping relationship from a 1-D array to a 2-D array.

Now we use the following pseudo-code to explain the programming techniques on the Phi coprocessor. A 5110P Phi coprocessor includes 60 cores and 240 hardware threads. We need to use OpenMp to split the task to smaller pieces based on the number of threads (nThreads=240 for 5110P Phi model). We split the matrix A to be the $nn \times mm$ blocks and the matrix B to be the $mm \times ll$ blocks so that the matrix C to be the $nn \times ll$ blocks. The block size of the matrix C is 16×16; the block size of the matrix A is 16×64 and the block size of the matrix B is 64×16. It ensures that the data is continuous in memory. The index k (the column of matrix A and row of matrix B) is normalized by 16 based on the 512-bit SIMD instruction and the constant nBlockSizeBySIMD is 4 in this case for the better code performance.

Code segment; use the standalone format and run the code on Phi:

```
#pragma omp parallel for
//Use OpenMP on Phi coprocessor
for (iThread=0; iThread<nThreads;
iThread ++) {
//240 hardware threads
for (t=iThread; t<nn * ll; t+=nThreads)
{
//t is block index
for (k=0; k<mm * nBlockSizeBySIMD;
k+=nBlockSizeBySIMD) {
//k is the column index of matrix A and the row index of
//matrix B counted by SIMD width
for (i=i0; i<i1; i ++) {
//i0(t) and i1(t) are index in matrix  for the t^th block
for (j=j0; j<j1; j ++) {
//j0(t) and j1(t) are index in matrix  for the t^th block
v=_mm512_set1_ps(0.0);
//initialize the variable v
v=_mm512_fmadd_ps(vA[i][k],vB[j][k],v);
//calculate A(i,k)*B(j,k) + v
v=_mm512_fmadd_ps(vA[i][k+1],
vB[j][k+1], v);
//calculate A(i,k+1)*B(j,k+1) + v
v=_mm512_fmadd_ps(vA[i][k+2],
vB[j][k+2], v);
//calculate A(i,k+2)*B(j,k+2) + v
v=_mm512_fmadd_ps(vA[i][k+3],
vB[j][k+3], v);
//calculate A(i,k+3)*B(j,k+3) + v
C[i][j] +=_mm512_reduce_add_ps(v);
// Summate elements of the vector v and add to C[i][j]
      }
    }
   }
  }
}
```

For two matrixes A and B with 2048×2048 elements, the perfromance of the solving matrix without the domain decomposition technique on an Intel Xeon E5 2640 v2 Ivy-Bridge CPU is 0.63 seconds, but the perfromance with the domain

decomposition technique on the same CPU is 0.4 seconds. The perfromance of the solving matrix with the domain decomposition technique on the Intel Xeon Phi Coprocessor 5100P is 0.15 seconds.

III. NUMERICAL RESULTS

In this section, we use the parallel FDTD code based on the Phi coprocessor to demonstrate the performance of the Phi coprocessor. The host computer incudes two Intel Xeon E5-2640 v2 CPUs with 32 GB DDR3 RAM and one 5110P Phi coprocessor is mounted on the host through a PCI-16 slot. 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 the Phi coprocessor. The problem size we first test is 1.29 GB 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 to 1,350 million cells per second when the problem size increases to 7.2 GB, as shown in Fig. 7.

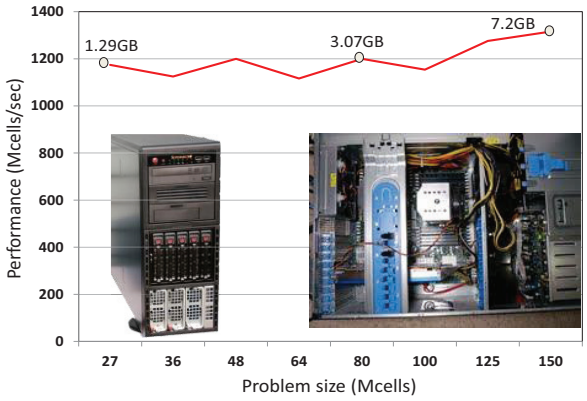


Fig. 7. Performance of the 5110P Phi coprocessor for the parallel FDTD code with the regular size of problems.

In the MIC instruction set, a single instruction allows us to perform a multiplication and one addition; for example, we have three variables A, B and C, the result of multiplication A and B and then addition with C can be reached by on operation in the MIC instruction:

$$D = A \times B + C. \quad (2)$$

The feature in (2) is called FMA (Fused Multiply-Add) in the MIC instruction. We demonstrate the performance of Phi coprocessor for the small problems such as 0.2 million cells and check the performance of the FMA feature on the FDTD code. The result is plotted in Fig. 8, and we cannot observe the performance down significantly for the small problems and the performance improvement from the FMA feature neither from Fig. 8.

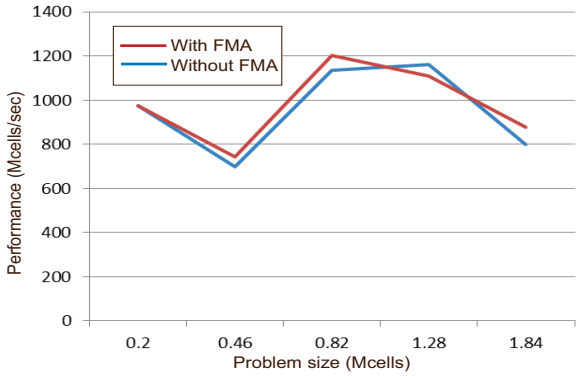


Fig. 8. Performance of the 5110P Phi coprocessor for the parallel FDTD code with the small size of problems.

IV. CONCLUSIONS

Intel Xeon Phi coprocessor is used to accelerate the electromagnetic simulations using its many core architecture and 512-bit vector units. Unlike the GPU acceleration, the Intel Xeon Phi coprocessor acceleration is more general and supports OpenMp. If a source code is compiled on an Intel Xeon E3 or E5 CPU, it can be run directly on a Phi coprocessor with a compilation option "-mmic".

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] J. M. Jin, "The finite element method in electromagnetics," (2nd edition), *New York: John Wiley & Sons*, 2002.
[6] A. Peterson and R. Mittra, "Computational methods for electromagnetics," *Wiley-IEEE Press*, 1997.
[7] "Intel® Xeon Phi™ coprocessor: system software developers guide," http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-system-software-developers-guide.html.
[8] "Intel® Manycore platform software stack," http://registrationcenter.intel.com/irc_nas/3988/MPSS_Users_Guide.pdf.

Xiaoling Yang graduated from Tianjin University, China with B.S. in Applied Mathematics and B.E. in Electric Engineering in 2001 and M.S. in Applied Mathematics in 2004. After that, he joined the Electromagnetic Communication Lab of Pennsylvania State University for several years as Research Associate. He has published over twenty conference and journal papers and co-authored four books in computational electromagnetics field. He also served as Reviewer for multiple conferences and journals. He was elevated as IEEE senior member in 2010. His research interests includes FDTD and FEM methods, parallel computing, hardware acceleration (GPU and Phi), 3-D modeling and visualization.

Wenhua Yu is Tepin Professor of Jiangsu Normal University, the President of 2COMU, Inc. and a Visiting Professor of Harbin Engineering University. He is the Director of Big-Data Analysis and Processing Key Lab of Jiangsu Province. He was a Visiting Professor/Research Associate of Pennsylvania State University from 1996 to 2010. He has worked on the topics related to FDTD methods, software development techniques, high performance computing techniques, and engineering applications for many years, first time applied the vector units and Phi coprocessors to solve electromagnetic problems, and has published more than 150 technical papers on the parallel FDTD methods and simulation techniques. He also authored Conformal Finite Difference Time Domain Maxwell's Equations Solver Software and User's Guide (Artech House, 2003), Parallel Finite Difference Time Domain Methods (Artech House, 2006), Electromagnetic Simulation Techniques Based FDTD Methods (John Wiley & Sons, 2009), Advanced FDTD Method: Acceleration, Parallelization, and Engineering Applications (Artech House, 2011), and VALU Acceleration Techniques for Parallel FDTD Methods (IET/SciTech Publisher, 2013), Advanced Computational Electromagnetics Methods and Applications (Editor, Artech House, 2014), and three books in Chinese (2005, 2010, 2012). He also translated one book (from English to Chinese) Understanding the Finite Difference Time Domain Method (John B. Schneider, Washington State University) (Tsinghua University Press, 2014). He is the General Co-Chair of several international conferences on the topic of computational electromagnetics, antennas and microwave circuits. He is a Lead Guest Editor of the Journal of International Antennas and Propagation on special issue "Small Antennas: Miniaturization Techniques and Applications." He is the member of technical committee of several international journals and a Guest Editor of special issue of Harbin University Workshop of Journal of Applied Computational Electromagnetics Society. He is a TPC Co-Chair of 2014 International Conference on Wireless Communications and Signal Processing. He is a senior member of IEEE and a primary developer of the GEMS software package. He is also the founder of Global Chinese Electromagnetic Network (www.globalchineseEM.org).