INVITED PAPER *Special Section on Recent Advances in Simulation Techniques and Their Applications for Electronics*

Hybrid MIC/CPU Parallel Implementation of MoM on MIC Cluster for Electromagnetic Problems

Yan CHEN[†], **Yu ZHANG**[†a)], **Guanghui ZHANG**[†], **Xunwang ZHAO**[†], **ShaoHua WU**[††], **Qing ZHANG**[††], ***Nonmembers***, ***and*** **XiaoPeng YANG**[†††], ***Member***

SUMMARY In this paper, a Many Integrated Core Architecture (MIC) accelerated parallel method of moment (MoM) algorithm is proposed to solve electromagnetic problems in practical applications, where MIC means a kind of coprocessor or accelerator in computer systems which is used to accelerate the computation performed by Central Processing Unit (CPU). Three critical points are introduced in this paper in detail. The first one is the design of the parallel framework, which ensures that the algorithm can run on distributed memory platform with multiple nodes. The hybrid Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) programming model is designed to achieve the purposes. The second one is the out-of-core algorithm, which greatly breaks the restriction of MIC memory. The third one is the pipeline algorithm which overlaps the data movement with MIC computation. The pipeline algorithm successfully hides the communication and thus greatly enhances the performance of hybrid MIC/CPU MoM. Numerical result indicates that the proposed algorithm has good parallel efficiency and scalability, and twice faster performance when compared with the corresponding CPU algorithm.

key words: *MIC accelerating MoM, MPI and OpenMP parallel programming mode, multiple nodes, out-of-core, pipeline*

1. Introduction

In the field of computational electromagnetics (CEM), the method of moments (MoM) is widely used for solving electromagnetic radiation and scattering problems [1]–[3]. It is well known that the lower/upper (LU) decomposition solver and the iterative solver are the most common ways to solve the matrix equations of MoM. To avoid slow convergence or divergence issue of the iterative solver, the LU decomposition is utilized as the matrix equation solver here. However, with the electrical size of problems increasing, the solution time of LU solver increases rapidly due to the computational complexity of $O(N^3)$, where N is the number of unknowns [4]. With the rapid development in computer hardware capabilities, parallel computing technique has become an efficient approach to solve extremely complicated engineering problems. In the field of scientific computing, the most frequently used parallel programing model is the Message Passing Interface (MPI) [5], [6].

Heterogeneous computing is another important direction of scientific computing. The heterogeneous computing refers to computing platforms that use more than one kind of processor, for example, a system equipped with Central Processing Unit (CPU) and Graphics Processing Unit (GPU) [7]. Generally, the CPU is referred to as the main processor or host, while other kinds of processors are referred to as the coprocessor or device. The GPU accelerated CEM algorithm has received rapid development [8], [9].

Significantly, another coprocessor named Intel Many Integrated Core architecture (MIC) [10] has also become a prevalent commodity recently in parallel computing due to its powerful computational capability. It has similar function to GPU. Intel MIC is targeted for highly parallel, High Performance Computing (HPC). The MIC will contribute more performance in future super computer platform. For example the Milkyway-2 super computer which has been claiming the lead in Top 500 list for 5 times and is still the fastest super computer now all over the world. The MIC coprocessors provide more than 70 percent of the total performance of the Milkyway-2 [11]. So the MIC must be the course for the future in CEM.

When the MIC coprocessor is taken into account, several problems greatly restrict the application of the hybrid MIC/CPU MoM [12]. Firstly, the heterogeneous cooperative computing technique must be combined with large scale parallel technique. Secondly, the memory of MIC coprocessor is usually much less than the main memory, which greatly restricts the scale of the problems that hybrid MIC/CPU MoM can solve. Thirdly, the data MIC coprocessor handles must be transferred to the MIC memory before computing starts.

In this paper, a hybrid MIC/CPU parallel LU decomposition algorithm, which can run on a distributed memory system, is implemented to solve the matrix equations generated from MoM. The implementation is complete in the sense that: 1) A hybrid MPI and Open Multi-Processing (OpenMP) [13] parallel framework is designed to ensure that the program can run on multiple nodes. 2) Under the framework, an out-of-core algorithm is utilized to break the limitation of MIC memory. 3) A pipeline technique is proposed to overlap the data movement with the MIC computation.

The remaining part of this paper is organized as follows. In Sect. 2, we outline the basic principle of LU decomposition and the hybrid MIC/CPU platform. In Sect. 3, the Implementation of Parallel Framework for Hybrid MIC/CPU is presented. The numerical results in Sect. 4 are followed by the conclusion.

Manuscript received October 20, 2015.
Manuscript revised January 9, 2016.
[†]The authors are with Xidian University, Xi'an, China.
[††]The authors are with Inspur, Beijing, China.
[†††]The author is with Beijing Institute of Technology, China.
a) E-mail: yuzhang@mail.xidian.edu.cn
DOI: 10.1587/transele.E99.C.735

2. Preliminaries

The basic principle of LU decomposition is briefly reviewed firstly in this section, the readers are referred to [3], [4] for an in-depth discussion.

2.1 The Basic Principle of LU Decomposition

By following the MoM procedure to solve an integral equation, a matrix equation can be obtained as

$$\mathbf{AI} = \mathbf{V} \tag{1}$$

where $\mathbf{A}$ is a $N \times N$ dense complex matrix in which N unknowns are used to represent the continuous electric and/or the magnetic current over the surface of interest. $\mathbf{I}$ is an $N \times 1$ coefficient vector of the unknown current coefficients to be solved for, and $\mathbf{V}$ is an $N \times 1$ excitation vector.

The numerical solution of Eq. (1) generally proceeds by factoring the impedance matrix $\mathbf{A}$ into a lower and an upper triangular matrix, which is called LU factorization or decomposition. The right-looking blocked parallel LU factorization algorithm is utilized to decompose the matrix $\mathbf{A}$ in this paper. The basic process of the right-looking blocked parallel LU factorization algorithm is indicated in Fig. 1. It computes one panel column (the yellow submatrix in Fig. 1 (a)) and one panel row (the blue submatrix in Fig. 1 (c)) at each time and then use them to update the trailing matrix (the green submatrix in Fig. 1 (d)). The shaded parts in Fig. 1 indicate the portions of matrix $\mathbf{L}$ and matrix $\mathbf{U}$, which has been finished in previous computation. The three steps are implemented repeatedly until the matrix is factorized. Generally, of all the steps, the update operations contribute more than 80% of the computation time for a large scale dense complex matrix.

2.2 Hybrid MIC/CPU Platform

The MIC coprocessor is physically connected to the CPU through a PCI Express (PCI-E) bus. Multiple MIC coprocessors can be installed in a single host node and many nodes can be connected through a network such as InfiniBand or Ethernet, as shown in Fig. 2.

It must be pointed out that the number of MIC coprocessors is usually less than the number of CPU cores in one node and the memory of MIC is far less than the system memory, or RAM as we referred to.

The MIC coprocessor features more than 50 scalar processing cores with vector processing units, and each core has four-way hyper-threading support to help hide memory and multi-cycle instruction latency. So there are up to 200 hyper threads in a MIC coprocessor.

The most common way to use MIC coprocessor is to regard it as an acceleration device of the host, which is similar to GPU.

up to 200 hyper threads in a MIC coprocessor.

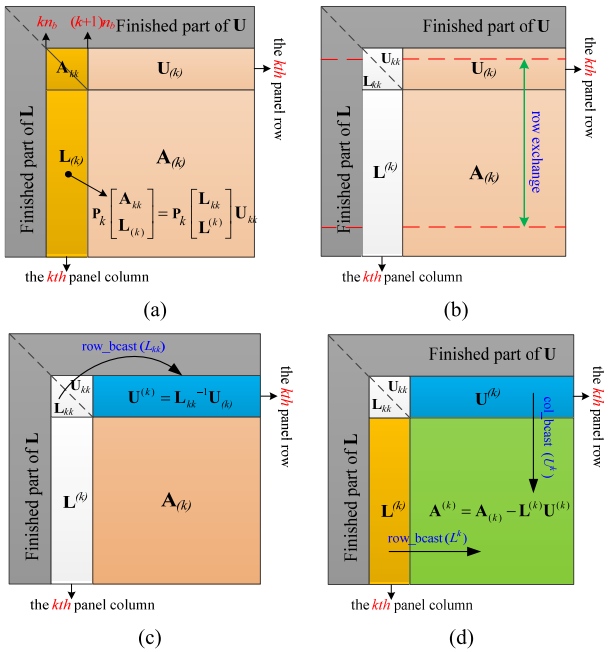


Fig. 1 The parallel LU decomposition. (a) panel factorization. (b) row exchange. (c) broadcast $\mathbf{L}_{kk}$ and panel row update. (d) broadcast $\mathbf{L}^{(k)}$ and $\mathbf{U}^{(k)}$ and trailing update. Note that the subscript (k) changing to the superscript (k) indicates that the current operation is complete.

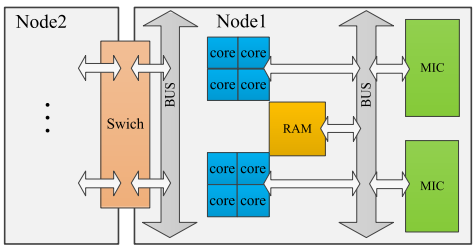


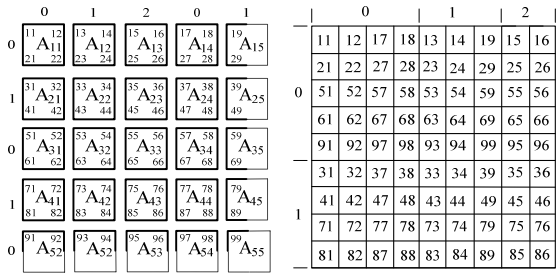
Fig. 2 Hardware architecture of one node in hybrid MIC/CPU platform.

3. Implementation of Parallel Hybrid MIC/CPU MoM

3.1 Parallel Framework of Hybrid MIC/CPU MoM

We are more concerned with load balance in this subsection. Load balance involves allocating data and tasks to MPI processors so as to ensure the most efficient use of resource, which means that all the MPI processors could finish the computation simultaneously.

Generally, two aspects are closely tied to the load balance. One is the amount of data and tasks allocated on these processors. The other is the ability of each MPI process to handle the data, or computer power as we call. To illustrate the effect of load balance, consider P MPI processor represented by P_i ($i = 1, 2, \ldots, P$), and the corresponding computer power of the processor is recorded as F_i. Suppose there are T_i unit of task assigned to processor P_i. Clearly, for a good load balance strategy, the equation followed must



(a) (b)

Fig. 3 Block-cyclic distribution of a matrix: (*a*) a matrix consisting of 5×5 blocks; (*b*) rank and coordinates of each process owning the corresponding blocks in (*a*).

be satisfied

$$\frac{T_1}{F_1} = \frac{T_2}{F_2} = \cdots = \frac{T_P}{F_P} \tag{2}$$

Specifically, a block-cyclic matrix distribution is used in blocked parallel LU factorization algorithm. As an example, consider matrix **A** shown in Fig. 3 (a), which is distributed to 6 processes in the 2×3 process grid.

Figure 3 (b) shows the result of the data distribution using the block-cyclic distribution methodology. In Fig. 3 (a), the outermost numbers denote the row and column indices of the MPI process coordinates. By varying the dimensions of the blocks of **A** and the process grid, different mappings can be obtained.

From the block-cyclic distribution methodology, one can see that the data and corresponding computation are assigned on different MPI processors roughly and uniformly. In other words,

$$T_1 \approx T_2 \approx \cdots \approx T_P \tag{3}$$

In order to achieve a good load balance, there must be

$$F_1 \approx F_2 \approx \cdots \approx F_P \tag{4}$$

It means that all MPI processors must provide almost appropriate storage and computer power. For example, each MPI processor is binding to one CPU core and allocated by 1 GB $\sim$ 2 GB memory. However, the number of coprocessors is usually less than CPU cores internal one node. So if one assign one CPU core and one MIC coprocessor to each MPI process, it would lead to an unmatched situation between MPI process and MIC coprocessor and cannot satisfy the load balance condition of Eq. (4). A straight-forward method to solve the problem is to reduce the number of MPI processes so that it equals to the number of MIC coprocessors. In order to make full use of all CPU cores, the multi-threads technique must be used in each MPI process.

However the number of MPI processes is determined by the number of MIC coprocessors, which is too inflexible in many practical applications. Furthermore, if the number of CPU cores cannot be divided by the number of MPI processes, this scheme will fail to work. The relationship between CPU cores and MPI processes can be changed comparatively and discretionarily, while one MIC coprocessor is always allocated to one MPI process. Consider assigning one MIC coprocessor to several MPI processes. It is equivalent to partition one MIC into several virtual MICs. This will greatly increase the flexibility of the hybrid MIC/CPU program without causing load unbalance.

Unfortunately, when many MPI processes are accessing to one MIC coprocessor at the same time, the program will inevitably lead the resource contention of MIC coprocessors. In order to make full use of MIC cores and avoid resource contention, one must segment the MIC cores for different MPI process. As we all known, there is a micro Linux Operating System (OS) in MIC coprocessor, and the concept of Environment Variable (ENV) is available on the OS of MIC. So one can separate the resource of MIC by setting different ENV's for different MPI processes. The pseudo-code used to set MIC ENV's is given as follows.

```
Algorithm: void mic_work_init_()
{
1       mic_env_pre(mic_id, nthreads, mic_affinity_env)
2       #pragma omp parallel num_threads(num_omp)
3       {
4           int omp_id = omp_get_thread_num()
5           if(omp_id == 0)
6           Transfer (nthreads, mic_affinity_ev) to device
7           {
8               omp_set_num_threads(nthreads);
9               kmp_set_defaults(mic_affinity_env);
10          }
11      }
12      return;
}
```

The function *omp_set_num_threads* sets the number of threads of MIC used by the MPI process that perform this function. The function *kmp_set_defaults* [10] binds threads MIC to MPI process. The function *mic_env_pre()* returns the information used by *omp_set_num_threads* and *kmp_set_defaults*.

After this, each MPI process will complete its own calculation task using CPU cores and MIC coprocessors assigned to it.

3.2 Out-of-Core Algorithm

It is clear that the most efficient scheme to accelerate the parallel MoM using hybrid MIC/CPU platform is to accelerate the hotspot of traditional parallel MoM primarily. Hotspot means the function or subroutines that executes for the longest time. Lots of applications indicate that a carefully designed matrix filling subroutine takes only 2 percent of total time of MoM, or less. So for the scope of this article, we will focus on the acceleration of LU factorization using MIC/CPU collaborative calculation.

The Intel VTune Amplifier XE [14] is used here to test the parallel LU factorization automatically. One can see from testing result Fig. 4 that the trailing update or subroutine ZGEMM [15] as we call, occupies about 81 percent of the total time of the LU decomposition.

Actually, subroutine ZGEMM performs the update operation $\mathbf{A}^{(k)} = \mathbf{A}_{(k)} - \mathbf{L}^{(k)}\mathbf{U}^{(k)}$ as shown in Fig. 1 (d). In order

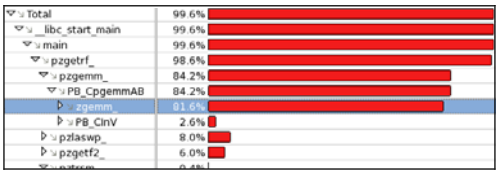


Fig. 4 Hotspot testing in one MPI processor.

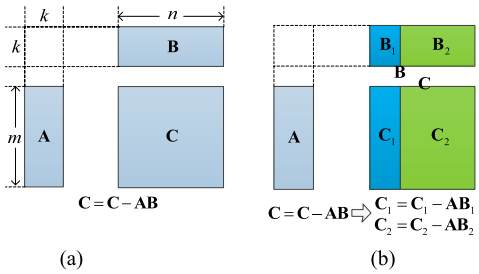


Fig. 5 Basic implementation of hybrid MIC/CPU MoM. (a) The data and computation assigned to one MPI process. (b) Basic implementation of hybrid MIC/CPU MoM

to simplify the description, the ZGEMM is rewritten as

$$\mathbf{C} = \mathbf{C} - \mathbf{AB} \tag{5}$$

where the sizes of matrices **A**, **B**, and **C** are labeled in Fig. 5 (a). Note that two '**C**'s on the left and right hand side of Eq. (5) are different in value, while they are stored in identical memory space of the computer. In other words they are stored in same array of FORTRAN, C or other programming languages with different values at different time. The right one that implies $\mathbf{A}_{(k)}$ possesses the old values and the left one that implies $\mathbf{A}^{(k)}$ possesses the new values. Equations (6)–(8) below have the same meaning. The basic hybrid MIC/CPU MoM algorithm can be implemented straightly, that partition the ZGEMM into two parts

$$[\mathbf{C}_1, \mathbf{C}_2] = [\mathbf{C}_1, \mathbf{C}_2] - \mathbf{A}\,[\mathbf{B}_1, \mathbf{B}_2] \tag{6}$$

as shown in Fig. 5 (b). The green part of data must be transferred to the memory of MIC and the corresponding calculation must be performed by MIC cores. Likewise, the blue ones stay on main memory and handled by CPU. It is worth noting that the ratio of the segmentation must be determined by the performance of CPU and MIC.

Once the required storage of matrices **A**, $\mathbf{B}_2$, $\mathbf{C}_2$ expands the MIC memory available, the scheme above will fail to work. In order to break the restriction of MIC memory, the matrices are split up into many small tiles, as shown in Fig. 6. So the second expression in Eq. (6) can be rewritten as

$$\mathbf{C}_{ij} = \mathbf{C}_{ij} - \mathbf{A}_i\mathbf{B}_j,\ i \in [1, y]\ and\ j \in [1, x] \tag{7}$$

where x and y are the numbers of columns and rows of the tiles. The size of each tile is labeled as M, N, and K.

And then the tiles are loaded to MIC memory, calculated, and uploaded to main memory one by one. The tile is so small that it is completely enough for MIC to store

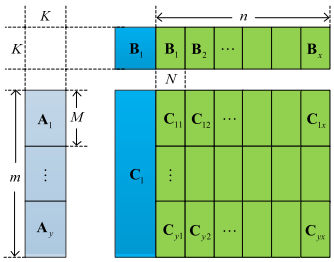


Fig. 6 The out-of-core algorithm

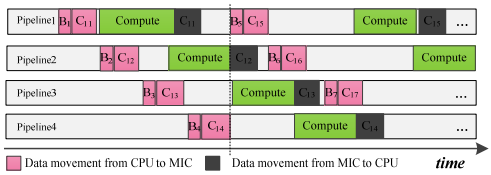


Fig. 7 The pipeline design

it. This algorithm is referred as the out-of-core algorithm, where the "core" means the MIC memory.

3.3 Pipeline Algorithm

The out-of-core algorithm greatly expands the scale of problems that hybrid MIC/CPU MoM can solve. However, one can see that the data movement is completely synchronously. Considering the huge storage required by MoM, the time consumed on data movement is a considerable expense. So the data movement must be optimized to enhance the efficiency of the hybrid MIC/CPU MoM program.

The data movement is integrant that it cannot be avoided, but one can overlap it with the calculation performed by MIC. In this section, a pipeline method is achieved. Asynchronous data movement is involved in the pipeline design besides asynchronous kernel computation. Figure 7 shows the pipeline design when $n = 4$. The red panes indicate the data movement from main memory to MIC memory and the black ones mean the opposite direction, of which the relatively tiny red panes represent the transformation of matrices $\mathbf{B}_j$ and the big red and black panes represent the transformation of matrices $\mathbf{C}_{ij}$. The data movement of matrices $\mathbf{A}_i$ is neglected because each matrix $\mathbf{A}_i$ will stay in memory for a very long time until all $\mathbf{B}_j$ and $\mathbf{C}_{ij}$ tiles in the same row finish.

3.4 Optimization of Parameters in Hybrid Parallel MIC/CPU MoM

When designing the pipeline for overlapping the data movement with computation, it was pointed out that the size of tile or the values of M, N and K in Fig. 6. The corresponding computation performed is

$$\mathbf{C}_{ij} = \mathbf{C}_{ij} - \mathbf{A}_i\mathbf{B}_j \tag{8}$$

Assuming the communication rate between host and

device through PCI-E is $v_{PCI\text{-}E}$, the performance of MIC is $v_{comp\text{-}M}$, and the performance of CPU is $v_{comp\text{-}C}$. In actual, the matrix $\mathbf{A}_i$ is firstly transferred to the device before the computation starts. The time it takes for the data movement of matrices $\mathbf{B}_i$ and $\mathbf{C}_{ij}$ through PCI-E is $16 \times (2MN+KN)/v_{PCI\text{-}E}$, where the factor 16 is the bytes that one complex double precision value occupies and 2 in front of MN is due to the fact that the matrix $\mathbf{C}_{ij}$ needs to be uploaded and downloaded. The calculation time of Eq. (8) by MIC coprocessor is $8MNK/v_{comp\text{-}C}$. To completely hide the communications, the calculation time should be greater than the communications time. So in order to overlap the data movement with the computation, one can get that $M > 2v_{comp\text{-}M}/v_{PCI\text{-}E}$ and $K > 4v_{comp\text{-}M}/(v_{PCI\text{-}E} - 2v_{comp\text{-}M}/M)$. For example, assuming $v_{PCI\text{-}E} = 6.0\,\text{GB/s}$ and $v_{comp\text{-}M} = 1.0\,\text{Tflops}$, which is actually the real parameters of our computing platform, substitute these values into both the equations above, and one can get $M > 310$ and $K > 628$. Note that M has little influence on K when M is large enough and N is never restricted.

If M and K do not satisfy the overlapping condition, or in other words, the time it takes for data movement is longer than the time for computation, the speedup will be figured out by the expression as follows

$$\frac{t_{comp\text{-}C}}{t_{comm\text{-}M}} = \frac{8MNK}{v_{COMP\text{-}C}} \div \frac{32MN + 16MK + 16KN}{v_{PCI\text{-}E}} \approx \frac{v_{PCI\text{-}E}}{4v_{comp\text{-}C}}K \tag{9}$$

where $t_{comm\text{-}M}$ means the time that it takes to transfer data between CPU and MIC. It can be seen that K plays a critical role in Eq. (9). Assuming that $v_{comp\text{-}C} = 422\,\text{Gflops}$, the peak performance of 2 Intel Xeon CPUs we used in this paper, the speedup will be $0.003817\,K$.

Now consider a testing of ZGEMM to validate the theoretic presented above and to indicate the efficiency of the out-of-core and pipeline algorithm. Because the size of tile is entirely unrelated to the number of nodes, only one node of XDHPC-MIC is used. The size of matrix $\mathbf{C}$ is set as $m = 20,000$ and $n = 13,000$, and a set of K's, ranging from 128 to 1024, are tested in this case. The ZGEMM is performed twice, one is by the CPU alone and the other is by the MIC with the out-of-core and the pipeline algorithms. The bandwidth of PCI-E and the real performance of CPU and MIC are also measured in the testing, which can be used to determine the best K and the theoretical speedup of MIC to CPU.

Figure 8 gives the testing result for different K, in which the real speedup is directly figured out by the practical testing times, and the theoretical speedup is determined by the measured $v_{PCI\text{-}E}$, $v_{comp\text{-}M}$ and $v_{comp\text{-}C}$. It is also known through calculation that the overlapping condition is satisfied when the value of K is equal to or greater than 576. And the same tendency of real speedup as theoretical speedup with only tiny difference indicates that the out-of-core and pipeline algorithm implemented in this paper operates efficiently.

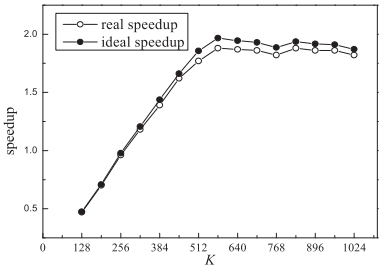


Fig. 8 The curves of real and ideal speedup with different K

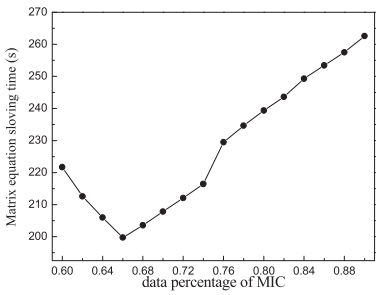


Fig. 9 The time of ZGEMM on one node using different percentage.

3.5 Distribution Ratio between MIC and CPU

Most of the time the ZGEMM will not be performed only by MIC alone, the CPU and MIC are all employed to do the work synergistically. The distribution ratio of data and calculation between CPU and MIC will greatly affect the efficiency of the hybrid parallel MIC/CPU implementation. Only when the CPU and MIC finish their own computation tasks at the same time, the best performance will be obtained. It was pointed out in Sect. 3.2 that the ratio of the segmentation must be determined according to the comparison of CPU to MIC in computing power. Considering $v_{comp\text{-}M} = 1.0\,\text{Tflops}$ and $v_{comp\text{-}C} = 422\,\text{Gflops}$ which has been mentioned above, the ratio should be calculated as 1,000/(1,000+422) ≈ 0.70. However, the real performance of MIC and CPU is generally not equal to the peak performance. So the actual ratio must be determined by testing go further.

Consider a ZGEMM operation with $m = 40,000$, $n = 40,000$ and $k = 640$, the percentage ranging from 0.6 to 0.9 is tested to select the best value. In this testing, four MPI processes are launched on each node and thus each process controls six cores through using OpenMP threads technique. The time it takes for calculating the ZGEMM at different percentage is plotted in Fig. 9.

From comparison, the percentage value of 0.66, which is close to the theoretical one 0.70, performs the better than other values, and it can save 2% to 20% of the matrix solving time. Throughout the rest of this paper, the percentage value is set to be 0.66.

Table 1 Comparison of computation parameters.

Algorithm	K	Nodes	CPU cores	MPI	OpenMP threads	Percentage	Pipelines	Buffer size
Intel MKL LU	128	4	96	96	-	-	-	-
Hybrid MIC/CPU	576	4	96	16	6	0.66	4	128

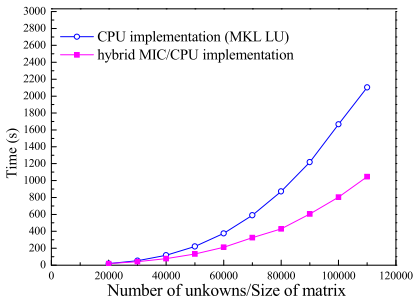


Fig. 10 Testing times for different sizes of matrices

Fig. 11 The model of an airplane

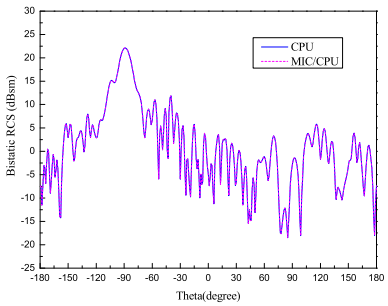


Fig. 12 2D RCS of the airplane: *xoz* plane

4. Numerical Results

The platform in our work is composed of 6 MIC/CPU hybrid architecture nodes or XDHPC_MIC as we call. Each node is equipped with 2 12-core Xeon E5-2692 v2 CPUs and 1 Xeon Phi 7110P MIC coprocessor. There are 61 cores and thus 244 hyper threads in the MIC coprocessor. And the sizes of main memory and MIC memory are 64 GB and 8 GB, respectively.

The parallel code is developed based on MPI and OpenMP hybrid programming model. Two basis functions, RWG [16] basis function and high order [3], [4] basis function, are used to construct the impedance matrix. We compare the proposed hybrid MIC/CPU parallel MoM implementation with the CPU implementation in performance. The LU decomposition of Intel MKL, which is the fastest parallel mathematics library on Intel platform, is utilized as the CPU implementation. So the comparison is truly a significant work for evaluating the performance of the hybrid MIC/CPU implementation.

4.1 LU Decomposition of Different Size of Matrices

In this section, the proposed algorithm is used to decompose stochastic non-singular matrices. Specifically speaking, 4 nodes of XDHPC-MIC are used. In hybrid MIC/CPU parallel LU algorithm, 4 MPI run on each node, which implies one MIC coprocessor is partitioned to 4 parts and assigned to 4 MPI processes. In order to make full use of all CPU cores, 6 OpenMP threads are dispatched, and one is used to control MIC. In parallel CPU implementation 96 MPI processes are launched and no OpenMP is used. The parameters are listed in Table 1 in detail. Figure 10 shows the benchmarking results for the matrices ranging from 20,000 $\times$ 20,000 to 110,000 $\times$ 110,000 elements in size.

The results show that the MIC accelerated LU decomposition scheme presented in this paper can save at least 50% of computation time than the Intel MKL, so long as the size of matrix exceeds 60,000.

It was pointed out that if K is selected as 576, the real speedup of MIC alone ZGEMM to CPU alone ZGEMM is 1.88 (Fig. 8), so the hybrid MIC/CPU synergistic implementation should have the speedup of 2.88. Considering that 16 CPU cores are consumed to control MIC rather without computation, the theoretical speedup should be 2.7, which is obtained by $1.88 + (96 - 16)/96$. Assuming the hotspot ZGEMM holds 80% time of the LU, the ultimate speedup of LU is theoretically 2.16, figured out by $2.7 \times 80\%$. The performance loss in real speedup 2.00 is contributed by the inefficiency of load unbalance when K is equal to 576.

4.2 RWG Basis Function

Here are the scattering results of an airplane presented to demonstrate the performance of the hybrid MIC/CPU parallel MoM implementation introduced above. The airplane is illustrated in Fig. 11 with the dimensions 11.6 m $\times$ 7.0 m $\times$ 2.93 m. MoM with RWB basis functions is used to generate the impedance matrix. The bistatic radar cross section (RCS) of the airplane is simulated at the frequency of 500 MHz. The total number of unknowns is 34,824. The two-dimensional (2D) RCS results are given for comparison in Fig. 12.

Two nodes in platform XDHPC-MIC are used. The computation parameters and the matrix equation solving time are listed in Table 2. Compared with the fastest parallel CPU implementation, the speedup of the hybrid MIC/CPU parallel implementation is about 2 times.

4.3 Higher-Order Basis Function

In this section, we present the radiation patterns and per-

Table 2 Comparison of computation parameters and time.

Memory (unit:GB)	Algorithm	K	CPU cores	MPI	OpenMP threads	LU time (unit:s)	Speedup
18.07	Intel MKL LU	128	48	48	-	217.381	-
	Hybrid MIC/CPU	576	48	8	6	108.377	**2.00**

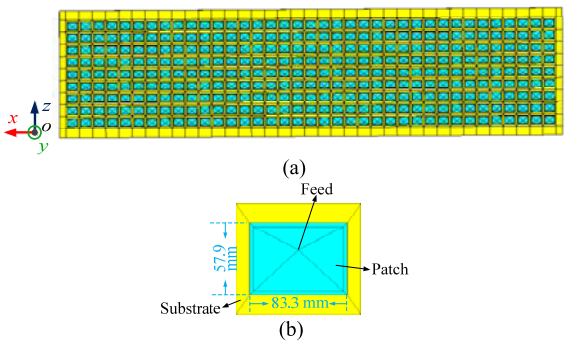


Fig. 13 Model of the microstrip patch array antenna. (a) Full array. (b) Array element.

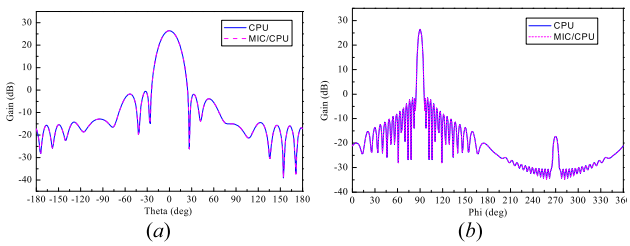


Fig. 14 2D gain patterns of the microstrip patch array antenna: (a) *yoz* plane and (b) *xoy* plane. Note that θ coordinate is measured from xoy plane to z axis and φ coordinate is measured from +x axis to y axis in this paper.

formance the hybrid MIC/CPU parallel Higher-Order MoM algorithm for an airborne phased array. Consider a rectangular microstrip patch array antenna with 37 × 9 elements, as shown in Fig. 13.

The dimensions of the full array are 10 m × 2.5 m × 0.018 m and the sizes of each patch are 57.9 mm × 83.3 mm. Each element of the array is fed by a short pin. The operation frequency of the array is 440 MHz and the number of unknowns is 83,996. We use 4 nodes (96 CPU cores) to solve this problem. The two-dimensional (2D) gain patterns are given in Fig. 14.

The running parameters and detail information are listed in Table 3. One can see from Table 3 that the speedup is also up to 1.99, which has only little difference compared to the theoretical one.

Another example is also presented to demonstrate the scalability and performance of the hybrid MIC/CPU parallel Higher-Order MoM algorithm. In this case, the RCS of the airplane model which is shown in Fig. 15 with the dimension of 18.92 m × 14.56 m × 5.05 m is simulated. The operating frequency is 1.0 GHz. The excitation is a z-axis polarized plane wave propagating along the negative x-axis direction.

Table 3 Comparison of computation parameters and time.

Memory (unit:GB)	Algorithm	K	CPU cores	MPI	OMP threads	LU time (unit:s)	Speedup
105.13	Intel MKL LU	128	96	96	-	1,011.20	-
	Hybrid MIC/CPU	576	96	16	6	508.24	**1.99**

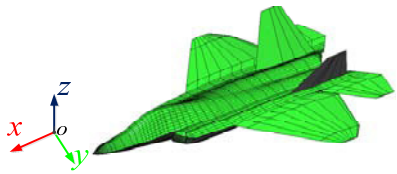


Fig. 15 The model of an airplane

Table 4 Comparison of computation parameters and time.

Memory (unit:GB)	Algorithm	K	CPU cores	MPI	OpenMP threads	LU time (unit:s)	Speedup
281.05	Intel MKL LU	128	144	144	-	3,063.79	-
	Hybrid MIC/CPU	576	144	24	6	1,531.35	**2.00**

The number of unknowns generated by this problem is 137,335, which requires about 281.05 GB in storage. In total, 6 nodes or 144 CPU are used to solve the full dense complex matrix equation. The running parameters and simulating time are listed in Table 4.

It is concluded that a desired speedup can also be obtained even when the number of nodes increases to 6. This indicates that the algorithm proposed in this paper has the same scalability as the Intel MKL LU decomposition algorithm.

5. Conclusion

In this paper, the hybrid MIC/CPU parallel LU decomposition algorithm is used to solve the matrix equation generated by the RWG or Higher-Order MoM. Three critical points are introduced in this paper in detail. Based on these three points, we successfully add the MIC support to the parallel MoM algorithm and obtain a good speedup. The numerical results indicate the accelerated algorithm proposed always works at a high efficiency, that is, the speedup of 2.0 compared with the commercial Intel MKL library. The hybrid MIC/CPU parallel LU decomposition also shows a good scalability when the number of CPU cores increases from 48 to 144, or the number of nodes increases from 2 to 6.

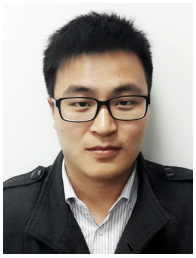
Acknowledgments

This work is supported by the National High Technology Research and Development Program of China (863 Program) (2012AA01A308), the NSFC (61301069, 61072019),

the Program for New Century Excellent Talents in University of China (NCET-13-0949) and the project with contract No.2013KJXX-67.

References

[1] R.F. Harrington, Field Computation by Moment Methods, in IEEE Series on Electromagnetic Waves, New York: IEEE, 1993.
[2] Y. Yan, Y. Zhang, C.-H. Liang, H. Zhao, and D. Gracia-Doñoro, "RCS computation by parallel MoM using higher-order basis functions," Int. J. Antennas Propag., vol.2012, pp.1–8, 2012.
[3] Y. Zhang and T.K. Sarkar, Parallel Solution of Integral Equation Based EM Problems in the Frequency Domain, John Wiley, Hoboken, NJ, 2009.
[4] Y. Zhang, Z. Lin, X. Zhao, and T.K. Sarkar, "Performance of a Massively Parallel Higher-Order Method of Moments Code Using Thousands of CPUs and Its Applications," IEEE Trans. Antennas Propag., vol.62, no.12, pp.6317–6324, 2014.
[5] S. Jiang, Y. Zhang, Z. Lin, and X. Zhao, "An Optimized Parallel FDTD Topology for Challenging Electromagnetic Simulations on Supercomputers," International Journal of Antennas and Propagation, vol.2015, pp.1–10, 2015.
[6] J. Mo, L. Shen, B. Wei, W. Fang, and Y. Yan, "RCS computation of engine by parallel higher-order MoM with out-of-core technique," IET International Radar Conference, Xi'an, The China, pp.1–3, April 2013.
[7] R. Farber, CUDA Application Design and Development, Elsevier, Singapore, 2013.
[8] X. Mu, H.-X. Zhou, K. Chen, and W. Hong, "Higher Order Method of Moments With a Parallel Out-of-Core LU Solver on GPU/CPU Platform," IEEE Trans. Antennas Propag., vol.62, no.11, pp.5634–5646, 2014.
[9] C. Jia, L. Guo, and P. Yang, "EM Scattering From a Target Above a 1-D Randomly Rough Sea Surface Using GPU-Based Parallel FDTD," IEEE Antennas Wireless Propag. Lett., vol.14, pp.217–220, 2015.
[10] Jim Jeffers and James Reinders, Intel Xeon Phi Coprocessor High Performance Programming, Morgan Kaufmann Press, San Francisco, Feb. 2013.
[11] Copyright 1993-2015 TOP500.org, "Tianhe-2 (MilkyWay-2)," Top500, http://www.top500.org/system/177999, accessed Sept. 19. 2015.
[12] G. Zhang, Y. Chen, Y. Zhang, S. Jiang, and X. Zhao, "MIC Accelerated LU Decomposition for Method of Moments," AP-S/URSI 2015, Vancouver, BC, Canada, pp.756–757, 2015.
[13] B. Chapman, G. Jost, R. van der Pas, and D.J. Kuck, Using OpenMP: Portable Shared Memory Parallel Programming, The MIT Press, Cambridge, MA, 2007.
[14] Intel Copyright, "Intel VTune Amplifier 2016," Intel Corporation, https://software.intel.com/en-us/intel-vtune-amplifier-xe/, accessed Sept. 19. 2015.
[15] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst, Numerical Linear Algebra on High-Performance (photocopy edition), Beijing: Tsinghua University Press, Feb. 2011.
[16] S.M. Rao, D.R. Wilton, and A.W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," IEEE Trans. Antennas & Propagat., vol.30, no.3, pp.409–418, May 1982.



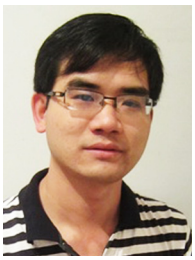
Yan Chen received the B.S. degree from ShanDong Normal University, JiNan, China, in 2012, and is currently working toward the Ph.D. degree at Xidian University. His current research interests is computational electromagnetic.

Yu Zhang received the B.S., M.S., and Ph.D. degrees from Xidian University, Xi'an, China, in 1999, 2002, and 2004, respectively. He joined Xidian University as a faculty member in 2004. He was a visiting scholar and Adjunct Professor at Syracuse University from 2006 to 2009. As principal investigator, he is doing or has completed some projects including project of NSFC.

Guanghui Zhang received the B.S. degree from Hunan University, ChangSha, China, in 2012, and is currently working toward the M.S. degree at Xidian University. His current research interests is computational electromagnetic.

Xunwang Zhao received the B.S., and Ph.D. degrees from Xidian University, Xi'an, China, in 2004, and 2008, respectively. He joined Xidian University as a faculty member in 2008. As principal investigator, he is doing or has completed some projects including project of NSFC.

ShaoHua Wu received the Ph.D. degree from Tsinghua University, Beijing, China, in 2013. He has joined the Inspur (Beijing) Electronic Information Industry Co., Ltd. since 2013. His current work focuses on the high performance computing.



Qing Zhang received the M.S. degree from Huazhong University of Science and Technology, WuHan, China, in 2009. He joined Inspur as a faculty member in 2009. He is currently the HPC Application R&D Manager at Inspur Group. He is also Inspur-Intel China Parallel Computing Joint Lab and Inspur-Nvidia GPU Joint Lab Chief Architect.

XiaoPeng Yang received the B.E. and M.E. degrees from Xidian University, Xi'an, China, in 1999 and 2002, and the Ph.D degree from Tohoku University, Sendai, Japan, in 2007, respectively. He joined School of Information and Electronics, Beijing Institute of Technology, Beijing, China.