Performance of a Massively Parallel Method of Moment Solver and Its Application

Yan Chen, Zhongchao Lin*, Daniel Garcia-Donoro, Xunwang Zhao, and Yu Zhang

School of Electronic Engineering
Xidian University, Xi'an, Shaanxi 710071, China
zclin@xidian.edu.cn

Abstract ─ A massively parallel Method of Moment (MoM) solver able to run on 200,000 CPU cores and solve matrices larger than 1.3 million unknowns is presented. The solver implements a novel LU decomposition algorithm based on the Communication Avoiding LU (CALU) scheme. By using a new pivoting policy, the communication between processes is improved enhancing the parallel speed up of the algorithm. Solver effectiveness and performance are demonstrated comparing the results with two of the most important math libraries used by direct dense solvers: the commercial MKL and the open source ScaLapack. Results show how simulation time is reduced significantly thanks to this novel LU decomposition algorithm making possible the simulation of incredibility electrically large problems using MoM.

Index Terms ─ Communication avoiding, high performance, LU decomposition, massively parallel, method of moments.

I. INTRODUCTION

Nowadays the analysis of extremely large structures is of crucial interest in military (and also civil) electromagnetic applications. The use of higher working frequencies makes the analysis of these structures, despite the constant enhancement in computer power, a challenge.

Among the pure numerical techniques employed in these analyses such as Finite Element Method (FEM) [1], Finite Difference Time Domain (FDTD) [2] or Method of Moments (MoM) [3], is the latter one that can provide the most accurate results for a wide variety of complex electromagnetic problems. However, the memory requirement and the computing complexity of MoM grow rapidly with $O(N^2)$ and $O(N^3)$, respectively, where N is the number of unknowns [4]. Thus, subject to these constraints, the applications of MoM for the simulation of extremely large structures are seriously limited.

In order to breaks these restriction, researchers have been employing several types of eclectic approaches; hybrid algorithms, as for example, the hybridization of MoM with high frequency methods [5, 6] or fast algorithms such as the fast multiple method (FMM) [7, 8]. These approaches reduce the memory requirement and the computation complexity; however, hybrid algorithms pay the price of losing accuracy, and the fast algorithms may be confronted with slow convergence or even divergence issues in applications involving complex structures and various materials.

It is worth noting that another approach must be considered without the needed of changing MoM as main numerical method keeping, in this way, the accuracy on the results. Along with the latest developments on computer technology, the use of parallel High Performance Computing (HPC) techniques on supercomputer platforms, with thousands of terabytes (TB) of memory available for simulation, can expand massively the application of MoM for extremely large analyses.

Working on the latter approach, authors have been developing during the last years their own numerical tools, being able to solve electromagnetic problem up to 1.0 million unknowns [9] by using a direct dense LU out-of-core solver [10]. Some of the previous work done by authors [4, 11] was focused on the use of LU factorization algorithms provided by the Intel Math Kernel Library (MKL) [12]. However, MKL does not always perform efficiently on some homemade platforms, especially when the communication components are based on Message Passing Interface (MPI) [13] techniques. Contrarily, if OpenMP [14] or other shared-memory parallel technique as the Basic Linear Algebra Subroutines (BLAS) [12] is used, MKL always performs well on Intel's CPU. Thus, one straightforward idea can be the use of open source parallel math libraries based on MPI techniques that help MKL to deal with its problems with MPI. One of the most important open source libraries of this kind is the well-known ScaLapack library [14]. This combination can not only make full use of the high-speed network, but also the full performance of the CPU cores, improving the simulation speed. With this point of view in mind, the already mentioned direct dense LU out-of-core solver was developed and expanded to 4096 CPU

Submitted On: December 29, 2016
Accepted On: August 12, 2017

cores with a parallel efficiency higher than 60% in [16].

However, it is worth to mention that the framework of ScaLapack is not the best scheme for parallelization. The computation and communication model of this framework were analyzed in detail in [16], concluding that the block of columns involved on each step of the factorization process (called panel factorization) is located in a critical position. The factorization process based on classic algorithm as Gaussian elimination with partial pivoting (GEPP) is not able to minimize the number of message exchange because of its pivoting strategy, which requires to permute the element of maximum magnitude to the diagonal position at each step in the panel factorization leading to a poor parallel performance. Then, better LU decomposition algorithms are required to break this poor parallel performance and improve the simulation time on supercomputer platforms with hundred thousand of CPU cores available. Under this scenario, authors have implemented a novel LU decomposition algorithm that drastically enhances the parallel performance of traditional LU decomposition algorithms enabling the possibility of running electrically large simulation on supercomputer efficiently. This novel LU decomposition algorithm takes, as starting point on its development, the Communication Avoiding LU (CALU) algorithm proposed and studied in [17] and [18]. However, mayor changes on the commutation scheme and the pivoting strategy help us to reduce the number and volume of the message exchanged between processes.

The paper presents the performance of a massively parallel MoM solver where the mentioned novel LU decomposition algorithm is included. Thanks to this technique the solver is able to run on more than 200,000 CPU cores and solve problems larger than 1.5 million unknowns. In order to demonstrate its effectiveness, the numerical results of three different benchmarks are compared with two of the most important math libraries used by direct dense solvers: the commercial MKL and the open source library ScaLapack.

II. ELECTROMAGNETIC THEORY

A briefly review about the integral equation form and the basis functions employed by the solver to approximate the solution are given in this Section.

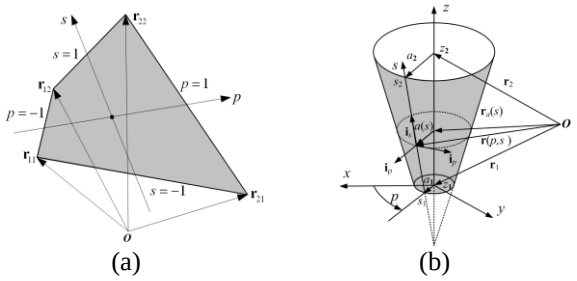
The solver is based on the solution of surface integral equations in frequency domain for the equivalent electric and magnetic currents over the dielectric boundary surfaces and electric currents only over perfect electric conductors (PECs). The set of integral equations obtained are solved by using MoM, and specifically using the Galerkin's method. The solver is able to handle inhomogeneous dielectrics categorized by a combination of various homogeneous dielectrics. Therefore, any composite metallic and dielectric electromagnetic structure can be represented by a finite number of linear, homogeneous and isotropic regions radiating in an unbounded linear, homogeneous and isotropic environment.

The integral equation employed is a general form of the Poggio-Miller-Chang-Harrington-Wu (PMCHW) formulation [4]. When one of the boundary surfaces between the two different regions is PEC, the magnetic currents are equal to zero at the boundary surface and that equation degenerates into the electric field integral equation (EFIE).

Flexible geometric modeling is achieved by using bilinear quadrilateral patches to characterize surfaces, as shown in Fig. 1 (a). Efficient approximation for the unknown currents is obtained by using the higher-order basis functions (HOBs) consisting of combinations of polynomials, which are of the following form:

$$\boldsymbol{F}_{ij}(p,s) = \frac{\boldsymbol{\alpha}_s}{\left|\boldsymbol{\alpha}_p \times \boldsymbol{\alpha}_s\right|} p^i s^j, \quad -1 \le p \le 1, \quad -1 \le s \le 1 \tag{1}$$

where, p and s are the local coordinates, i and j are the orders of basis functions, and $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_s$ are covariant unitary vectors [4]. The polynomials can also be used as the basis functions for wire structures. In this case, truncated cones are used for geometric modeling, as shown in Fig. 1 (b). In the thin-wire model, the circumferential variation of the currents on the wires is neglected, and in addition the length of the wire should be at least 10 times larger than its radius.



(a) (b)

Fig. 1. Geometric modeling for higher-order basis functions: (a) a bilinear quadrilateral patch defined by four vertices with the position vectors of $\boldsymbol{r}_{11}$, $\boldsymbol{r}_{21}$, $\boldsymbol{r}_{12}$ and $\boldsymbol{r}_{22}$, and (b) a truncated cone defined by position vectors and radii of its beginning and end, characterized by $\boldsymbol{r}_1$, a_1, and $\boldsymbol{r}_2$, a_2, respectively.

The orders can be adjusted according to the electrical size of a geometric element. The orders increase as the element becomes larger. The electrical size of a geometric element can be as large as two wavelengths. Typically, the number of unknowns for HOBs is reduced by a factor of 5–10 compared with that for traditional piecewise basis functions, e.g., the RWGs, and thus the use of HOBs drastically reduces the computation and saves memory requirement.

There are also some other advantages in using the polynomial basis functions. For example, the intermediate results obtained in evaluating the elements of the impedance matrix for lower-order can be used in the computation of the elements of the impedance matrix when using higher-order polynomials. In addition, the Green's function for each pair of integration points belonging to two patches is needed to be evaluated only once. These advantages improve the efficiency of the matrix filling of the solver and help when the filling is performed using thousands of CPU cores.

III. PARALLEL LU DECOMPOSITION

As it was commented previously, better LU decomposition algorithms are required to break the poor parallel performance given by traditional LU decomposition schemes. In this section details about the novel LU decomposition algorithm included in our parallel MoM solver are given. In order to provide a better understanding about how this new LU algorithm (refers as NLU from now) enhances the parallel performance of the traditional LU decomposition algorithm, the latter is also detailed in this section. The algorithm described here (refers as ScaLapack LU from now) is the one given in [15].

A. ScaLapack LU algorithm

Let us consider an N×N dense complex matrix divided in blocks of size $n_b \times n_b$ and distributed across a $P_r \times P_c$ process grid. Choosing a value of P_c approximately equal to P_r, the communication volume is minimized, and in consequent, the high parallel efficiency is improved. Also, the communication complexity regarding column panel is higher than the complexity involved in the row panel. Therefore, it is better to set P_c slightly larger than P_r [16]. Figure 2 shows the main steps involved on the ScaLapack LU decomposition and it will be used as additional help when describing the algorithm.

After distributing the matrix on the processes grid, the LU decomposition algorithm starts with k=1 in Fig. 2 (a) where k is the number of the block of columns that are involved on the factorization step. This block of columns is called column panel. The size of this panel is $[N\text{-}(k\text{-}1)\times n_b]$ rows by n_b columns where the row size decreases when the factorization process advances. The panel is marked in green in Fig. 2 (a) and labeled as $\mathbf{Z}_{kk}$ + $\mathbf{L}_{(k)}$. The operation performed during this step can be written as:

$$\mathbf{P}\begin{bmatrix}\mathbf{Z}_{kk}\\ \mathbf{L}_{(k)}\end{bmatrix} = \mathbf{P}\begin{bmatrix}\mathbf{L}_{kk}\\ \mathbf{L}^{(k)}\end{bmatrix}\mathbf{U}_{kk}\,, \tag{2}$$

where $\mathbf{P}$ is the permutation matrix corresponding to the partial pivoting scheme, $\mathbf{Z}_{kk}$ + $\mathbf{L}_{(k)}$ is the column panel previously mentioned and $\mathbf{L}_{kk}$, $\mathbf{U}_{kk}$, and $\mathbf{L}^{(k)}$ are the result of the factorization process. In order to save computational resources, $\mathbf{Z}_{kk}$ is generally overwritten by the lower triangular matrix $\mathbf{L}_{kk}$ and the unit upper triangular matrix $\mathbf{U}_{kk}$. Figure 2 (b) shows the row exchange process where the element of maximum magnitude is permuted to the diagonal position in the panel factorization (this movement is given by matrix $\mathbf{P}$). It is worth to remind that this row exchange process requires a large volume of message communication leading on poor parallel performance. Submatrices $\mathbf{L}_{kk}$, $\mathbf{U}_{kk}$, and $\mathbf{L}^{(k)}$ are also displayed on the figure.

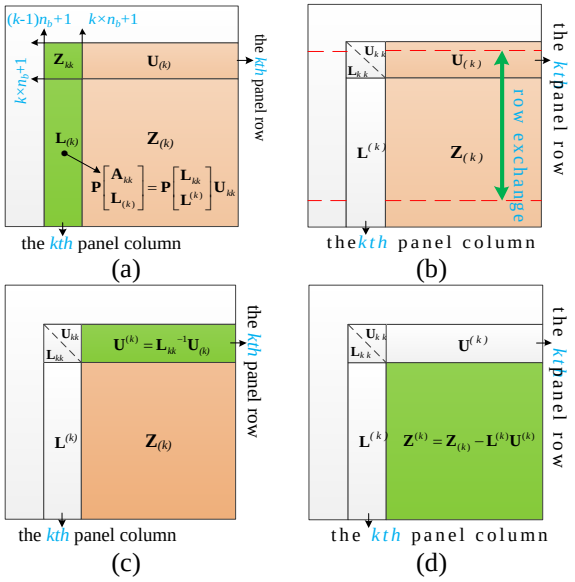


Fig. 2. ScaLapack LU decomposition: (a) panel factorization, (b) row exchange, (c) broadcast L_{kk} and panel row update, and (d) broadcast $L^{(k)}$ and $U^{(k)}$ and trailing update. *Note that the subscript* (*k*) *changing to the superscript* (*k*) *indicates that the current operation is complete.*

The next step corresponds to the row panel update process (see green submatrix marked in Fig. 2(c)) using the submatrices obtained previously. The update is given by:

$$\mathbf{U}^{(k)} = \mathbf{L}_{kk}^{-1}\mathbf{U}_{(k)}\,, \tag{3}$$

where $\mathbf{U}_{(k)}$ is the original row panel and $\mathbf{L}_{kk}$ is the lower triangular matrix of the column panel. Once the unknown $\mathbf{U}^{(k)}$ is obtained, the original row panel $\mathbf{U}_{(k)}$ is overwritten saving computational resources as the previous case.

The last step performs the update of the trailing submatrix $\mathbf{Z}_{(k)}$, marked in orange in Fig. 2 (c). This update uses the results of both panel factorization and row panel update. The resulting $\mathbf{Z}^{(k)}$ submatrix is given by:

$$\mathbf{Z}^{(k)} = \mathbf{Z}_{(k)} - \mathbf{L}^{(k)}\mathbf{U}^{(k)}\,. \tag{4}$$

Once the new $\mathbf{Z}^{(k)}$ submatrix is obtained, k is incremented and a new block of columns is factorized

continuing this process until the original matrix is decomposed completely.

B. Novel LU algorithm

The main different between the ScaLapack LU algorithm and our NLU resides on the way that the column panel is decomposed. This decomposition is located on a critical position on the LU algorithm being, in the case of the ScaLapack, unable to minimize the number of message exchanged due to the partial pivoting scheme. However NLU, adopting a different pivoting strategy, is able to reduce the number and amount of communication during this process enhancing the parallel performance.

It is worth mentioning that there are no differences between the ScaLapack LU algorithm and NLU for the others steps on the factorization process. Once the decomposition of the column panel is done, the algorithms follow the same scheme regarding the row update and the trailing submatrix update. For this reason, and in order to give a better description of our algorithm, only the decomposition of the column panel is detailed here.

Let us consider the same N×N dense complex matrix than the previous case. The column panel shown in Fig. 2 (a) (remember green columns on the left) is extracted and redisplayed in Fig. 3 as **A**. The distribution of the matrix on the process grid is performed as the previous case. However, for simplicity, we assume that the total number of processes is P=24. According to the criteria of choosing P_r and P_c aforementioned, the best choice is P_r = 4 and P_c = 6 obtaining four pieces of **A**:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0^T & \mathbf{A}_1^T & \mathbf{A}_2^T & \mathbf{A}_3^T \end{bmatrix}^T, \tag{5}$$

where the matrix **A** is of size [N-(k-1)×n_b] by n_b and $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{A}_3$ are distributed on processes P_0, P_1, P_2, and P_3, respectively. Without loss of generality, we assume the size of $\mathbf{A}_i$ is m_i×n_b with i=0, 1, 2, 3 and m_i about [N-(k-1)×n_b]/4. Generally, these four submatrices are referenced as *local panels*.

The decomposition of the column panel **A** is divided in three steps. The first step performs the decomposition of each of the submatrices $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{A}_3$ using *partial pivoting*. This decomposition process is done independently in each MPI processes without any interaction between them. Thus, this step is called *local decomposition step* where no communication between processes is required. Equation 6 shows the decomposition of each submatrix of the column panel **A**:

$$\begin{aligned} \mathbf{A_0} &= \mathbf{P}_0^{(1)}\mathbf{L}_0^{(1)}\mathbf{U}_0^{(1)} \\ \mathbf{A_1} &= \mathbf{P}_1^{(1)}\mathbf{L}_1^{(1)}\mathbf{U}_1^{(1)} \\ \mathbf{A_2} &= \mathbf{P}_2^{(1)}\mathbf{L}_2^{(1)}\mathbf{U}_2^{(1)} \\ \mathbf{A_3} &= \mathbf{P}_3^{(1)}\mathbf{L}_3^{(1)}\mathbf{U}_3^{(1)} \end{aligned}, \tag{6}$$

where $\mathbf{P}_i^{(1)}$ is the permutation matrix with the information of the pivoting and $\mathbf{L}_i^{(1)}$ and $\mathbf{U}_i^{(1)}$ are the corresponding submatrices result of the factorization process. Then, a set of *local pivoting rows* is obtained and permuted to the first n_b rows of each submatrix using the permutation matrix $\mathbf{P}_i^{(1)}$. Figure 3 shows this process under the label step 1 where the small matrices on the right contains the rows selected for pivoting during the *local decomposition step*.

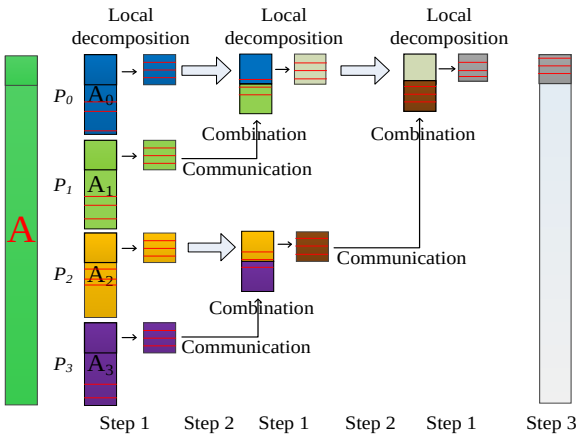


Fig. 3. Illustration of column factorization in our parallel MoM LU decomposition algorithm.

The second step combines the *local pivoting rows* pair to pair using a recursive scheme. For example, the local pivoting rows of $\mathbf{A}_0^{(1)}$ and $\mathbf{A}_1^{(1)}$ are combined in one pair while the rows of $\mathbf{A}_2^{(1)}$ and $\mathbf{A}_3^{(1)}$ are combined to form another pair (see the label step 2 on the left in Fig. 3). These new pairs are called new l*ocal column panels* which are treated as new panels for decomposition. Equation 7 shows the decomposition of this new *local column panels*:

$$\begin{aligned} \begin{bmatrix} \mathbf{A}_0^{(1)} \\ \mathbf{A}_1^{(1)} \end{bmatrix} &= \mathbf{P}_0^{(2)}\mathbf{L}_0^{(2)}\mathbf{U}_0^{(2)} \\ \begin{bmatrix} \mathbf{A}_2^{(1)} \\ \mathbf{A}_3^{(1)} \end{bmatrix} &= \mathbf{P}_1^{(2)}\mathbf{L}_1^{(2)}\mathbf{U}_1^{(2)} \end{aligned}. \tag{7}$$

Once this new local decomposition step is carried out, a new set of *local pivoting rows* is obtained and permuted to the first n_b rows of the column panel (submatrices marked in brown and gray in Fig. 3). Both *local decomposition* and *combination step* are called iteratively until only one block of *pivoting rows* is obtained. The total number of operations needed to obtain the final pivoting block is equal to $\log_2 P_r$. Continuing with the decomposition of **A**, the final local decomposition step is given by:

$$\begin{bmatrix} \mathbf{A}_0^{(2)} \\ \mathbf{A}_1^{(2)} \end{bmatrix} = \mathbf{P}_0^{(3)}\mathbf{L}_0^{(3)}\mathbf{U}_0^{(3)}. \tag{8}$$

Once this local decomposition is done, the final

pivoting rows are obtained. These *pivoting rows* are similar to the one used to form the permutation matrix **P** employed in (2). However, if one permutes these pivoting rows to the first n_b rows of **A**, then the LU decomposition without pivoting could be performed. In practice, even the LU decomposition is unnecessary because the latest *local decomposition* gives the upper triangular matrix **U** which is the same as the ultimately **U** we expect to obtain such as $\mathbf{U}_0^{(3)}$ in (8). So the only operation needed after executing this pivoting technique is a multiplication of an upper triangular matrix $\mathbf{U}_0^{(3)}$ by a block of column of **A** as shown in (9):

$$\mathbf{A}(n_b:m,1:n_b)=\mathbf{A}(n_b:m,1:n_b)\mathbf{U}_0^{(3)\ -1}, \qquad (9)$$

where **A** is the permuted column panel with the *pivoting rows* already located in the first n_b rows of the matrix. It is obviously that the communication appears only in the step of the combination of the *local pivot rows*, as implied by the arrows labeled with "communication" in Fig. 3.

The above description was simplified for the case of $P_r = 4$ but, actually this procedure can be easily extended to a situation with $P_r = 8$, 16, 32 and so on. With some additional but easy work, P_r could be an arbitrary quantity, rather than just power of 2. Of course, the extension from binary tree to quad tree or octree is not a big deal.

It is worth to mention that the *combination step* between *local pivoting rows* is not straightforward. In a first stage, it is necessary to determine which two *pivoting blocks* are combined. Then, as these two associated *pivoting blocks* are located at two different MPI processes, one also needs to define the communication pattern between them. Finally, after the combination, one needs to know which MPI process will perform the new *local decompositions*. In this work, the binary tree reduction method [17] is employed to guide the combination step. More specifically, two pivoting blocks located on adjacent MPI processes are the one chosen to be combined. After a combination, only half of the current MPI processes will perform the new *local decomposition step*. This reduction in the number of the working processes is continuously done until only one process performs the last *local decomposition step*, meanwhile the other processes will be waiting for the final permutation matrix.

Compared with the original CALU algorithm studied in [16, 17] the major difference with NLU resides in the communication pattern involved in the combination step. As shown in Fig. 3 all the processes are paired up to get a new *pivoting block*. In the original CALU algorithm each process exchanges their own *pivoting blocks* with its partner and then both the processes perform the same *local decomposition*, while in NLU only a unidirectional communication is performed and only the receiver-process performs the *local decomposition*. After several tests, we concluded that this variation provides a better implementation and makes the algorithm easier to achieve when using supercomputer platforms. Also, the extension from binary tree to quad tree or octree is much straightforward using the unidirectional communication version than the omnidirectional scheme.

C. Analysis of algorithm complexity

Before going to the numerical results section, the analysis of the computation complexity and the communication pattern of NLU algorithm is carried out. In order to perform a complete analysis, a comparison with the computation complexity of the ScaLapack LU algorithm is also done. As it was commented previously, the main different between the ScaLapack algorithm and NLU resides on the way that the column panel is decomposed. Therefore, the communication during the panel factorization is discussed in detail here.

Let us take, for example, the *k*'th step of LU decomposition shown in Fig. 2. Assuming that the size of $\mathbf{L}^{(k)}$ is $m \times n_b$ and the size of $\mathbf{L}_{kk}$ is $n_b \times n_b$, the amount of multiplication and addition operations can be calculated as it is shown later. We ignore the division operation because there are only few of those during the algorithms. Furthermore, let us assume that the communication latency is α and the communication bandwidth is 1/β. Thus, the communication time *T* taken to send a message of size *L* is:

$$T = \alpha + \beta L. \qquad (10)$$

Now, let us consider the ScaLapack LU algorithm first. For every column in the panel, a binary-exchange of size 2 n_b data items is performed [19], being n_b the number of columns in the panel. The communication complexity of this binary-exchange is $log_2 P_r$ given a total communication time of:

$$\begin{aligned} T_{comm,s} &= n_b \times (\alpha + 2 \times n_b \beta) \times \log_2 P_r \\ &= n_b \alpha \log_2 P_r + 2\beta n_b^2 \log_2 P_r \end{aligned}. \qquad (11)$$

In order to calculate the computation time taken by ScaLapack, we can consider that the panel factorization is a LU decomposition of a non-square matrix. Thus, the computation time can be evaluated according to the standard LU decomposition time that is:

$$T_{fact,s} = (m - \frac{n_b}{3}) n_b^{\ 2} \gamma, \qquad (12)$$

where γ is the time for each multiplication or addition operation, *m* is the number of row in the column panel and n_b is the number of columns.

In the case of NLU, one can see from Fig. 3 that the number of point-to-point communications performed by the busiest MPI process (P_0) is $log_2 P_r$. A final broadcast operation is also necessary at the end of the preprocess step, so the communication complexity of this broadcast operation (also $log_2 P_r$) has to be added to the total communication time. Assuming that the number of element to be broadcasted is $n_b^2 / 2$, the total communication time is given by:

$$T_{comm,c} = (\alpha + \beta n_b^2) \times \log_2 P_r + (\alpha + \beta \times \frac{n_b^2}{2}) \times \log_2 P_r \\ = 2\alpha \log_2 P_r + 1.5\beta n_b^2 \log_2 P_r \quad (13)$$

Generally, m is much greater than n_b, so it is clear that the operations described by (6) and (9) contribute the most amount of calculation in this case. Both operations have almost the same times of multiplication and addition as ScaLapack. So the computation time in NLU is about 2 times more than in ScaLapack:

$$T_{fact,c} = 2(m - \frac{n_b}{3}) n_b{}^2 \gamma \, . \quad (14)$$

Comparing the communication time given by (11) and (13), it can be concluded that NLU requires less communication time than ScaLapack. In the other hand, comparing the computation time given by (12) and (14) one can see how NLU pays a double price during the calculation. However, we have to take into account the order of magnitude of the variables involved on the calculation each time. The γ time employed by the computer to perform a multiplication or addition is much smaller than the latency time α and the inverse of the communication bandwidth β. Furthermore, NLU can utilize many excellent sequential factorization algorithms directly in the *local decomposition step*, such as recursive sequential LU, performing the floating point operation at machine peak performance, which means that the operation time γ in (14) is generally smaller than the time employed in (12). Thus, even employing a larger computational time, the NLU provides a shorter global time for matrix factorization. In order to demonstrate this affirmation, the next section presents the numerical results where the computational time of different benchmarks is compared.

IV. NUMERICAL RESULTS AND DISCUSSION

This section shows the comparison of the computational times for three different benchmarks between NLU and two of the most important math libraries used by direct dense solvers: the commercial MKL and the open source ScaLapack. The benchmarks considered here have consisted of the use of different matrix sizes, the use of different number of CPU cores and a comparison on the parallel efficiency of the algorithms. These benchmarks have been run on three different high performance computing (HPC) platforms that are described next.

A. Description of the computational resources

The first platform is the HPC cluster of Xidian University (XDHPC), which is equipped with 100 compute nodes connected by 56 Gbps InfiniBand network. Each node has two twelve-core Intel Xeon 2690 V2 2.2 GHz CPUs. The second platform is the MilkyWay-2 (Tianhe-2), which has 16,000 compute nodes. Each compute node is equipped with two Intel Xeon E5-2600 processors and three Intel Xeon Phi accelerators. All compute nodes are connected by a homemade 150 Gbps network. The third platform is the Sunway BlueLight MPP, a Chinese petaflop homegrown supercomputer. Sunway BlueLight uses ShenWei SW-3 1600, a 16-core 64-bit MIPS-compatible CPU. The total number of compute nodes on the system is 8704 connected by InfiniBand QDR network.

B. Numerical results for different matrix size

This benchmark has consisted of two tests using different matrix size and checking the factorization time of the algorithms. The first test employs a matrix size of 20,000~100,000 using 240 CPU cores in a process grid of 15x16, while the second test makes use of a matrix size of 80,000~180,000 in 720 CPU cores with a process grid of 24x30. The computing time obtained using the Intel MKL, ScaLapack and NLU are listed in Table 1. The improvement in percentage regarding the computing time using our solver versus MKL and ScaLapack is listed in the last two columns of the table. For example, the percentage corresponding with the improvement against MKL was calculated by using the following expression: $(T_{MKL}-T_{NLU}) / T_{MKL} \times 100\%$. The percentage corresponding to ScaLapack was calculated in a similar way. According to Table 1, the performances of our solver are clearly better than both MKL and ScaLapack for all the cases.

C. Numerical results for different CPU cores

The second benchmark has consisted of the simulation of two real world applications using different number of CPU cores. The first simulation analyzes the radar cross section (RCS) of an airplane. The second test calculates the radiation pattern of an airborne wire array antenna. XDHPC platform was used to run this benchmark.

Testing-I: The RCS of the airplane shown in Fig. 4 is calculated in this test. The size of the airplane is 18.92 m by 13.56 m by 5.05 m and the simulation frequency is 500 MHz, given a total number of unknowns of 203,436. The computing time consumed by NLU and MKL using 240~2048 CPU cores is listed in Table 2. The last row of data indicates the improvement in the computing time with respect to MKL. According to this table, the performance of our solver gets better when the number of CPU cores increases (in consequence more communication message are required) improving between 5.5%-8% the computational time given by MKL. The comparison between the RCS values obtained using our LU decomposition algorithm and the MKL library for the azimuth cut are shown in Fig. 5 where a very good agreement is appreciated. This behavior is expected as both decomposition algorithms are solving the same

matrix although, as aforementioned, NLU improves the computational time around 8% in the best case.

Table 1: Computing time for NLU, MKL and ScaLapack

CPU Cores	Size of Matrix	Time of LU Decomposition (Unit: s)			NLU VS MKL	NLU VS ScaL
		NLU	MKL	ScaL		
240	20,000	12.20	15.35	13.10	20.52%	6.83%
	40,000	61.41	70.86	68.17	13.34%	9.91%
	60,000	175.01	194.71	196.61	10.12%	10.99%
	80,000	397.96	417.31	449.60	4.64%	11.48%
	100,000	714.77	758.18	796.24	5.72%	10.23%
720	80,000	173.04	200.01	192.38	13.49%	11.18%
	100,000	316.15	351.01	346.82	9.93%	9.70%
	120,000	506.72	556.97	578.68	9.02%	14.20%
	140,000	781.27	822.87	851.76	5.06%	9.02%
	160,000	1093.43	1180.17	1264.40	7.35%	15.64%
	180,000	1490.39	1607.81	1693.76	7.30%	13.65%

Table 2: Computing time for NLU and MKL

CPU Cores	240	720	960	1200	2048
NLU Time	20091.5	2127.08	1588.10	1274.39	858.88
MKL Time	19257.7	2131.75	1650.88	1386.33	909.58
Time Saved	-4.33%	0.22%	3.80%	8.07%	5.57%

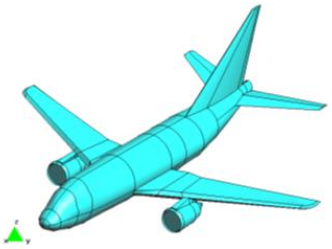


Fig. 4. Airplane model for RCS calculation using RWG.

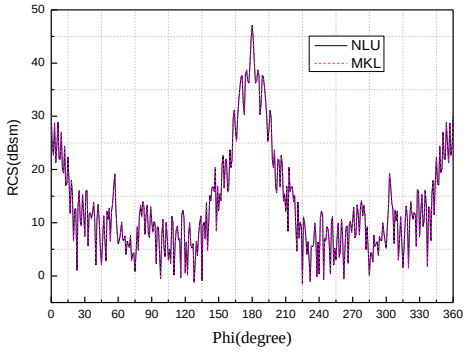


Fig. 5. RCS comparison using NLU and MKL.

Testing-II: The analysis of an airborne wire array antenna with 72 × 14 elements is performed in this test. The simulation model is shown in Fig. 6. The dimensions of the full array are 10 m × 2.5 m × 0.018 m. Each element of the array is fed by a short pin, and the amplitude at the feed of the array is designed by a –35 dB Taylor distribution both along length and width. The operation frequency of the array is 1.0 GHz given a total number of unknowns of 259,128.

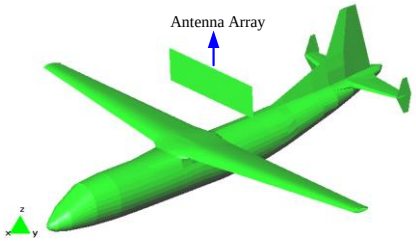


Fig. 6. Simulation model of the wire antenna array.

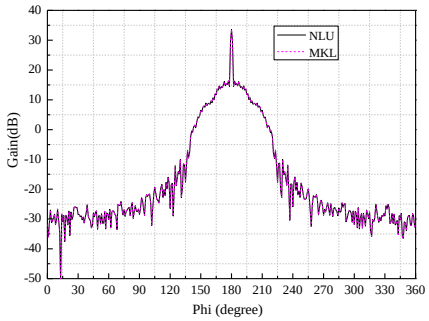


Fig. 7. Gain comparison in dB for NLU and MKL.

Table 3: Computing time for NLU and MKL

CPU Cores	240	720	960	1200	2048
NLU Time	11033.6	4407.73	3295.77	2743.26	1673.46
MKL Time	11174.0	4472.85	3501.06	2909.09	1788.27
Time Saved	1.26%	1.46%	5.86%	5.70%	6.42%

The computing time consumed by our LU decomposition algorithm and MKL using 240~2048 CPU cores is shown in Table 3, where the improvements in the time are listed in the last row as the previous test. The comparison between the 2D gain patterns are given in Fig. 7 where a very good agreement is appreciated. It is worth mention that according to Table 3, as the number of CPU cores increases the improvement in the computing time given by our solver is greater.

D. Numerical results for parallel efficiency

The last benchmark has consisted of the simulation of two massively parallel simulations used to measure the parallel efficiency of the algorithms. The parallel efficiency can be defined as:

$$\eta = T \Big/ pT_p \times 100\% , \tag{15}$$

where T is the total time taken by a single process and T_p is the total time taken by p processes. In practice, if the problem cannot fit into a single computing node, the time for the smallest number of processes is taken as reference.

The calculation of the radiation pattern of a rectangular microstrip patch antenna array formed by 37 × 9 elements mounted on an aircraft is considered first. The simulation model of the array and the airplane are shown in Figs. 8 and 9. The dimensions of each patch element of the array are 205.6 mm × 154.8 mm providing a total dimension for the full array of 10 m × 2.5 m × 0.018 m. The material parameters of the substrate are $\varepsilon_r = 4.2$ and $\mu_r = 1.0$ and the operation frequency of the array is 440 MHz. The dimensions of the airplane were 55 m long by 47.6 m wide and 15.8 m high obtaining a total number of unknowns for the simulation of 308,371.

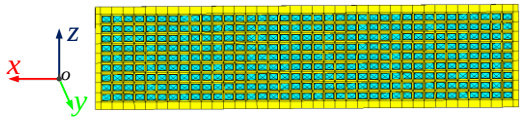


Fig. 8. Model of the microstrip patch array antenna.

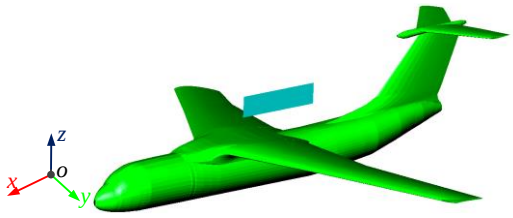


Fig. 9. Computational model with the aircraft and array.

The solving time taken by NLU using 600~12000 CPU cores is shown in Table 4, where the last two columns give the memory required in the form of value and percentage. Milkyway-2 platform was used to run this benchmark. Setting the solving time given by the 600 CPU cores simulation as the reference, the speed up listed in the fourth column is calculated by $S_P = 7504.08/T_P$, and the parallel efficiency listed in fifth column is calculated by $S_P/(P/600)$, where P is the number of the CPU cores and T_P and S_P is the corresponding solving time and speedup. Figure 10 shows the variation of speedup and parallel efficiency on CPU cores. According to the figure, efficiency higher than 65% can still be achieved when the parallel scale extends by 20, from 600 to 12000 CPU cores. Although more CPU cores usually mean shorter simulation time, the efficiency may deteriorate rapidly when the memory ratio is less than 2%. It should be pointed out that the maximum memory ratio should better less than 80% because of the operating system's requirement. Therefore, in order to avoid an extreme degradation on the parallel performance, the number of CPU cores should be choose carefully. Figure 11 shows the simulation results for this benchmark, where a -25 dB side lobe level in both *yoz* and *xoy* planes is appreciated fitting in the design specifications.

Table 4: Parallel efficiency and memory ratio for NLU

0	CPU Cores	Solving Time (s)	Speed-up	Parallel Efficiency (%)	Memory (GB)	Memory Ratio (%)
308,371	600	7504.08	1	100	1417	70.90
	1200	3799.00	1.97	98.76		35.45
	2400	2112.25	3.55	88.81		17.73
	3600	1471.84	5.09	84.97		11.82
	4800	1158.78	6.47	80.94		8.86
	7200	858.42	8.74	72.84		5.91
	9600	650.32	11.53	72.11		4.43
	12000	551.97	13.59	67.97		3.55

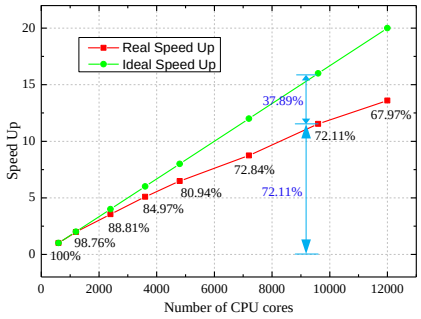
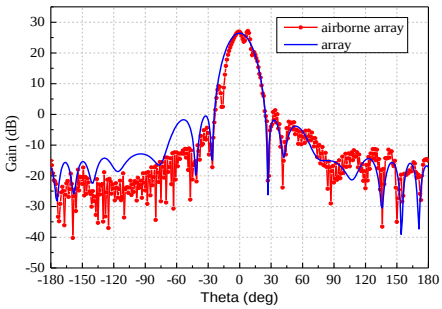
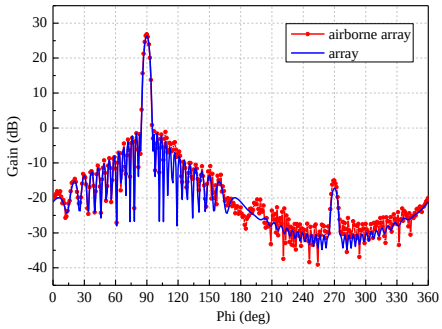


Fig. 10. Parallel efficiency for microstrip.



(a)



(b)

Fig. 11. 2D gain patterns for: (a) yoz plane and (b) xoy plane.

The parallel efficiency of NLU is further tested by the simulation of the bistatic RCS of the airplane shown in Fig. 12 at frequencies of 1.5 GHz and 2.5 GHz. Milkyway-2 platform was also used to run this simulation.

The dimensions of the airplane are 18.92 m × 14.56 m × 5.05 m, given a number of unknowns of 273,808 for the first frequency and 671,777 for the second one. The excitation is a z-axis polarized plane wave propagating along the negative x-axis direction.

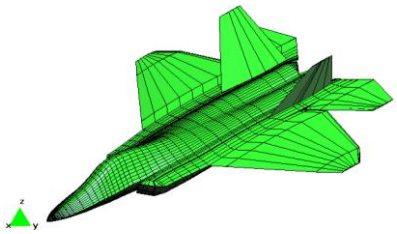


Fig. 12. Airplane model for efficiency test II.

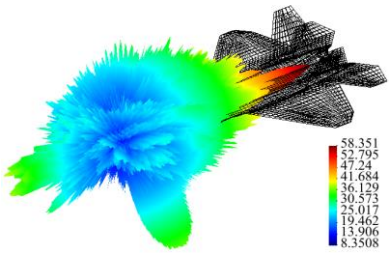
The computing time consumed by NLU in the simulation is shown in Table 5. The number of processes used at 1.5 GHz ranges between 1536 and 25920 CPU cores while this number ranges from 25920~107520 CPU cores at 2.5 GHz. According the table, it can be seen how the efficiency is still higher than 50% even when using more than 100,000 CPU cores, as long as the memory ratio is larger than 2% approximately. The results of the RCS are listed in Fig. 13.

E. Numerical results for practical engineering

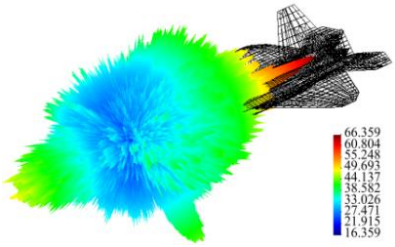
The RCS of the airplane shown in Fig. 12 is now calculated at a higher frequency to demonstrate the ability of the solver to calculate electric super-lager targets. The working frequency in this case is 3.2 GHz given a total number of unknowns of 1,270,200, which requires 23.4 TB of memory. A total number of 201,600 CPU cores are employed to solve the problem. The computing time and simulation result are listed in Table 6 and Fig. 14, respectively.

Table 5: Parallel efficiency for test II

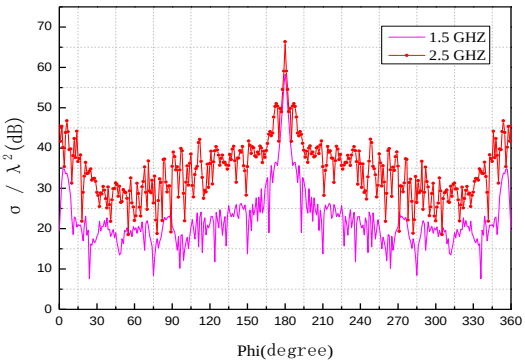
Unknowns	CPU Cores	Solving Time (s)	Parallel Efficiency	Memory (GB)	Memory Ratio
273,808	1,536	2376.71	100%	1117.32	27.28%
	12,960	437.88	80.41%		3.23%
	25,920	246.81	45.51%		1.62%
671,777	25,920	2388.10	100%	6724.66	9.73%
	76,800	1629.09	83.43%		3.28%
	107,520	1264.98	57.06%		2.35%



(a)



(b)

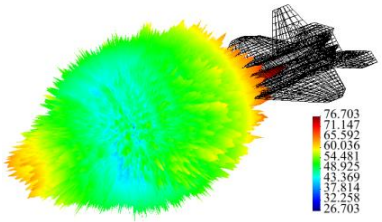


(c)

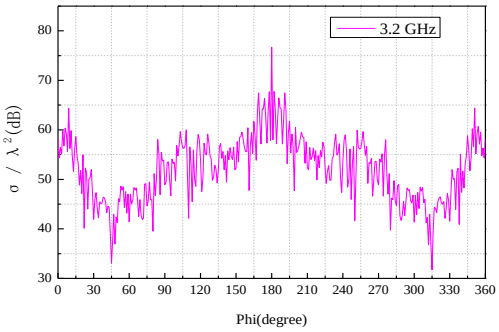
Fig. 13. Bistatic RCS results for: (a) 3D-RCS at 1.5 GHz, (b) 3D-RCS at 2.5 GHz, and (c) 2D-RCS on xoy at 1.5 GHz and 2.5 GHz.

Table 6: Solving matrix equation using 200,000 CPU cores

CPU Cores	Unknowns	Process Grid	Block Size	Filling Time (s)	Solving Time (s)
201,600	1,270,200	400×504	128	23.10	3021.05



(a)



(b)

Fig. 14. Bistatic RCS results at 3.2 GHz for: (a) 3D-RCS and (b) 2D-RCS for azimuth cut.

V. CONCLUSION

A massively parallel MoM solver able to run on 200,000 CPU cores and solve matrices larger than 1.3 million unknowns has been presented. Details about a novel LU decomposition algorithm have been given demonstrating its improvements in the simulation time in comparison with commercial Intel MKL and open source ScaLapack Libraries. The new algorithm is about 10~20 percent faster than the open source ScaLapack framework. Also, compared with the commercial Intel MKL on InfiniBand interconnected platform when thousands of CPU cores are used, it still has 5~10 percent advantage in performance. Furthermore, one can see how the algorithm can still achieve a high parallel efficiency even when 200,000 CPU cores are used presenting a new powerful tool for solving very challenging electromagnetic problems in reasonable time.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB0202102, in part by the National High Technology Research and Development Program of China (863 Program) under Grant 2014AA01A302, in part by the China Postdoctoral Science Foundation funded project under Grant 2017M613068, in part by the Key Research and Development Program of Shandong Province under Grant 2015GGX101028, and in part by the Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase) under Grant No. U1501501.

REFERENCES

[1] J. M. Jin, *The Finite Element Method in Electromagnetics*. John Wiley & Sons, Inc., 1993.

[2] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech House, Norwood, Mass, USA, 2000.

[3] R. F. Harrington, *Field Computation by Moment Methods in IEEE Series on Electromagnetic Waves*. IEEE, New York, NY, USA, 1993.

[4] Y. Zhang and T. K. Sarkar, *Parallel Solution of Integral Equation Based EM Problems in the Frequency Domain*. Hoboken, NJ: John Wiley, 2009.

[5] J. Chen, M. Zhu, M. Wang, S. Li, and X. Li, "A hybrid MoM-PO method combining ACA technique for electromagnetic scattering from target above a rough surface," *ACES Journal*, vol. 29, no. 4, pp. 301-306, 2014.

[6] Y. Kim, H. Kim, K. Bae, J. Park, and N. Myung, "A hybrid UTD-ACGF technique for DOA finding of receiving antenna array on complex environment," *IEEE Trans. Antennas Propag.*, vol. 63, no. 11, pp. 5045-5055, 2015.

[7] J. M. Song, C. C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex object," *IEEE Trans. Antennas Propag.*, vol. 45, no. 10, pp. 1488-1493, 1997.

[8] H. Fangjing, N. Zaiping, and H. Jun, "An efficient parallel multilevel fast multipole algorithm for large-scale scattering problems," *ACES Journal*, vol. 25, no. 4, pp. 381-387, 2010.

[9] Y. Zhang, T. K. Sarkar, M. C. Taylor, and H. Moon, "Solving MoM problems with million level unknowns using a parallel out-of-core solver on a high performance cluster," *in IEEE Antennas and Propagation Soc. Int. Symp*., Charleston, SC, USA, pp. 1-4, 2009.

[10] Y. Zhang, R. A. van ce Geijn, M. C. Taylor, and T. K. Sarkar, "Parallel MoM using higher-order basis functions and PLAPACK in-core and out-of-core solvers for challenging EM simulations," *IEEE Trans. Antennas Propag.*, vol. 51, no. 5, pp. 42-60, 2009.

[11] Y. Zhang, T. K. Sarkar, X. Zhao, D. Garcia-Donoro, W. Zhao, M. Salazar, and S. Ting, *Higher Order Basis Based Integral Equation Solver (HOBBIES)*. Hoboken, NJ: John Wiley, 2012.

[12] Intel Copyright (2015) Intel Math Kernel Library for Linux OS User's Guide, Intel Corporation. available: https://software.intel.com/sites/default/files/managed/df/1e/mkl_11.3_lnx_userguide.pdf.

[13] W. Gropp, T. Hoefler, and R. Thakur, *Using Advanced MPI: Modern Features of the Message-Passing Interface*. Cambridge, MA: The MIT Press, 2014.

[14] B. Chapman, G. Jost, R. van der Pas, and D. J. Kuck, (foreword), *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, MA, 2007.

[15] L. S. Blackford, J. Choi, A. Cleary, et al., "ScaLAPACK: A portable linear algebra library for distributed memory computers - Design issues and performance," *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, IEEE*, pp. 1-20, 1996.

[16] Y. Zhang, Z. Lin, X. Zhao, et al., "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.

[17] L. Grigori, J. W. Demmel, and H. Xiang, "CALU: A communication optimal LU factorization algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1317-1350, 2008.

[18] A. Khabou, J. W. Demmel, L. Grigori, and M. Gu, "LU factorization with panel rank revealing pivoting and its communication avoiding version," *eprint arXiv:1208.2451*, 2012.

[19] http://www.netlib.org/benchmark/hpl/scalability.html