# Highly efficient parallel direct solver for solving dense complex matrix equations from method of moments

Yan Chen, Zhongchao Lin, Yu Zhang, Daniel García Doñoro

School of Electronic Engineering, Xidian University, No. 2 South Taibai Road, Xi'an, Shaanxi 710071, People's Republic of China
E-mail: zclin@xidian.edu.cn

Published in *The Journal of Engineering*; Received on 1st March 2017; Accepted on 3rd March 2017

**Abstract:** Based on the vectorised and cache optimised kernel, a parallel lower upper decomposition with a novel communication avoiding pivoting scheme is developed to solve dense complex matrix equations generated by the method of moments. The fine-grain data rearrangement and assembler instructions are adopted to reduce memory accessing times and improve CPU cache utilisation, which also facilitate vectorisation of the code. Through grouping processes in a binary tree, a parallel pivoting scheme is designed to optimise the communication pattern and thus reduces the solving time of the proposed solver. Two large electromagnetic radiation problems are solved on two supercomputers, respectively, and the numerical results demonstrate that the proposed method outperforms those in open source and commercial libraries.

## 1 Introduction

Nowadays more complex structures and higher working frequencies make the analysis of electromagnetic characteristics using pure numerical techniques, such as method of moments (MoM), a challenge. Parallel computing is a useful tool to extend the problem scale and reduce the computation time. Working on this approach, authors have been developing during the last years their own MoM being able to solve electromagnetic problem up to about 1.0 million unknowns (dense matrix) by using a direct lower upper (LU) solver [1], and the parallel scale was expanded to 4096 CPU cores with parallel efficiency higher than 60% [2]. LU decomposition provided by Scalable Linear Algebra Package [3] (ScaLAPACK) is employed to solve the linear system.

The efficiency of the ScaLAPACK depends on efficient implementations of the BLAS [4] (routines performing basic vector and matrix operations, such as matrix–matrix multiply) and the BLACS [4] (routines create a linear algebra oriented message passing interface). Although Intel Math Kernel Library (MKL) [5] provides a high-performance BLAS library, it does not always perform efficiently, especially on non-Intel CPU platform. To obtain a better performance for large-scale parallel MoM simulations, one needs to improve calculating speed of BLAS and reduce the communication time in LU decomposition. Therefore, a novel LU decomposition scheme based on communication avoiding LU [6] is proposed in this study to reduce the number and amount of message exchange.

On the other hand, modern microprocessors include multiple levels of cache and vector units that can operate on multiple data with a single instruction, or single instruction multiple data units as we know. It is thus important to use these vector instructions in order to achieve optimal hardware usage efficiency. Therefore, a vectorisation and cache optimised BLAS (VCOBLAS) is proposed in this study. By adopting some micro-optimisation methods, the VCOBLAS greatly improve the rate of vectorisation (MMX/SSE/AVX Instruction) and cache (L1/L2 Level) usage. In practical, the VCOBLAS determines the real utilisation of each CPU's peak computing capability, while the novel LU decomposition scheme determines the scalability when many CPU cores are used. Numerical results indicate that the proposed solver can achieve a higher performance compared with both the open source ScaLAPACK and commercial Intel MKL.

## 2 Novel LU decomposition scheme

Parallel LU solver is used in the factorisation of a dense matrix of MoM by repeatedly factorising block columns. The main difference between the ScaLAPACK LU algorithm and our novel LU algorithm resides on the way that the column panel is decomposed, so only the panel factorisation is considered. The panel column matrix, represented as $\boldsymbol{A}$, according to the two-dimensional (2D) block cyclic data distribution scheme [3], is distributed on $P_r$ processes. For simplicity, we assume that $P_r = 4$, as shown in Fig. 1. The novel panel factorisation scheme can be described by three steps in detail. The first step performs the decomposition of each of the submatrices $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ and $\boldsymbol{A}_3$ using partial pivoting and four pivoting rows are obtained. This decomposition process is done independently in each message passing interface (MPI) processes without any interaction between them. Thus, this step is called *local decomposition*. Then the second step combines the local pivoting rows pair to pair using a recursive scheme. Both *local decomposition* and *combination* steps are called iteratively until only one block of pivoting rows is obtained. Once the *local decomposition* is done, the final pivoting rows are obtained and a factorisation without pivoting is performed. The main innovation of parallel communication avoiding algorithm for the LU factorisation (CALU) is the fact that it adopts a new pivoting strategy that is similar to the reduction algorithm or tournament pivoting, and thus leads to the reduction of the number and amount of the inter-message.

## 3 Vectorisation and cache optimised BLAS

The kernel function or hotspot in LU solver is the matrix–matrix multiplication operation, which can be performed by the subroutine ZGEMM in BLAS. MKL provides an optimised BLAS library which can exploit the advantage of Intel CPU. We certainly develop our own ZGEMM codes running on multiple CPU platforms. The original open source ZGEMM routine are tested on a computer equipped with 2 Intel Xeon E5-2690-v2 CPUs (2.2–2.6 GHz) and 64 GB memory. The running characteristics of original ZGEMM, obtained by Intel Vtune Amplifier tool, are listed in Table 1. In the table, 'Peak Perf.' indicates the peak performance of the machine. 'Real Perf.' indicates the real performance the codes could exploit. 'AVX Vect.' is the ratio of AVX-256 Instruction vectorisation of the code. The parameters clock cycle per instruction (CPI), last level cache miss (LLCM) and memory read/write rate

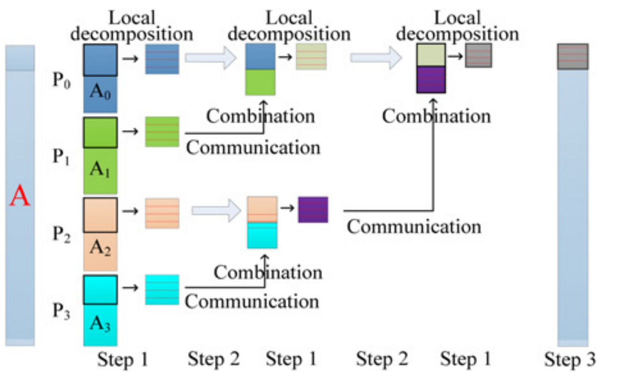

**Fig. 1** *Column factorisation in the novel LU decomposition scheme*

**Table 1** Running characteristics of original ZGEMM

| Real Perf. | Peak Perf. | AVX Vect. | CPI | LLCM | Mem. read | Mem. write |
|---|---|---|---|---|---|---|
| 44Gflops | 498Gflops | 0.0% | 0.78 | 97.88% | 45 GB/s | 1 GB/s |

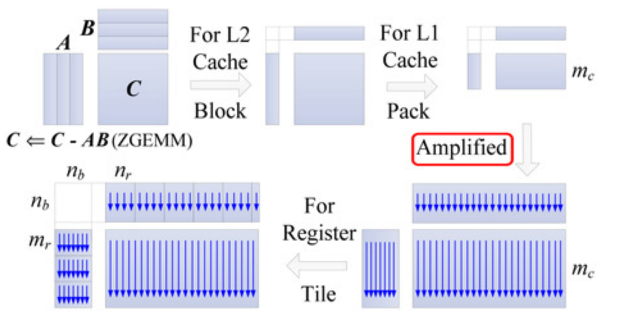

**Fig. 2** *Micro-optimisation of ZGEMM*

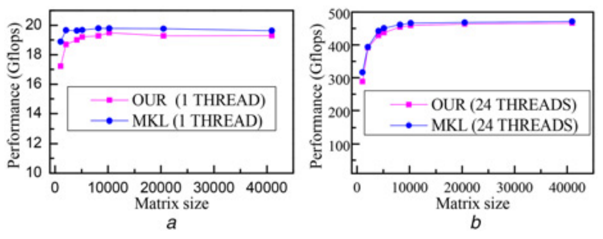

**Fig. 3** *Performance of ZGEMM on one computer*

**Table 2** Running characteristics of MKL and our own ZGEMM

| | Real Perf. | AVX Vect. | CPI | LLCM | Mem. read | Mem. write |
|---|---|---|---|---|---|---|
| MKL | 471Gflops | 67% | 0.36 | 87% | 5.1 GB/s | 3.7 GB/s |
| our | 465Gflops | 63% | 0.41 | 81% | 5.3 GB/s | 3.6 GB/s |

(Mem. Read/Write) are also listed. By using some tricks (i.e. block, pack) rearrangement data as tile storage [7], the number of times for CPU to access memory is greatly reduced and thus the cache utilisation increases. By usage of CPU registers and assembly instruction [7], the AVX-256 (on Intel platform) vectorisation is achieved manually. A brief diagram of all these 'tricks' and the typical values

**Table 3** Comparison of ScaLAPACK and our own LU factorisation

| CPU cores | 2400 | 7200 | 9600 | 12,000 | 20,480 | 30,720 |
|---|---|---|---|---|---|---|
| our LU | 12438.6 | 4642.2 | 3594.1 | 2926.1 | 2025.5 | 1443.3 |
| ScaLAPACK | 13160.1 | 5250.4 | 4086.4 | 3435.3 | 2475.3 | 1756.5 |
| speed-up | 1.124 | 1.153 | 1.176 | 1.208 | 1.222 | 1.217 |

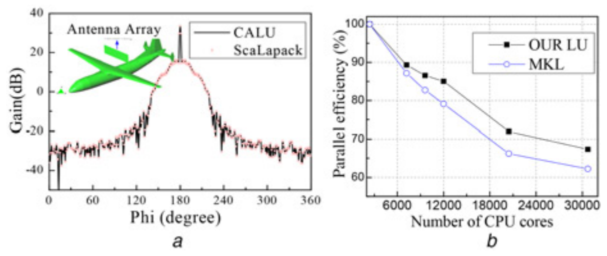

**Fig. 4** *2D radiation pattern and parallel efficiency*
*a* 2D radiation pattern
*b* Parallel efficiency

of the blocks are shown in Fig. 2. The optimising results are listed in Fig. 3 and Table 2. One can see that the optimised code is about 11 times faster than the original one. Compared with MKL, our code has a slight disadvantage, but it can achieve a very high performance on non-Intel CPU platform.

## 4 Results

Two computational platforms are used here. The first one is the Sunway BlueLight MPP, a Chinese petaflop homegrown supercomputer. The total number of compute nodes on the system is 8704 connected by a 40 Gbps InfiniBand QDR network. Each compute node is equipped with a home-grown processor (SW1600 16 Cores 975 MH, non-Intel CPU). The second one 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.

A phased array with main beam pointing to the tail of an airplane is simulated using MoM with higher order basis functions on Sunway BlueLight MPP. The operation frequency of the array is 1.0 GHz given a total number of unknowns of 259,128. The simulation model is shown in Fig 4*a*. As the Intel MKL is unavailable on the homegrown platform, ScaLAPACK is selected as a reasonable alternative. The computing time consumed by the proposed LU solver and open source ScaLAPACK using 2400–30,720 CPU cores is shown in Table 3. According to the table, one can see that the proposed LU solver is about 10–20% faster than the ScaLAPACK. Parallel efficiency determines that the scalability with the number of CPU cores increase, and is calculated by $\eta = \left(T_r/T_p\right)/\left(P/P_r\right) \times 100\%$, where $T_r$ is the reference time when $P_r$ CPU cores are used, and $T_p$ is the computation time when $P$ CPU cores are used. Parallel efficiency evaluated according to the simulation time is shown in Fig. 4*b*. The parallel scale overcomes 30,000 CPU cores with a parallel efficiency higher than 65% (reference: 2400 cores). The comparison between gains results given by proposed LU solver and the ScaLAPACK for the azimuth cut is shown in Fig. 4*a* where a very good agreement is appreciated.

Furthermore, the radiation pattern of the microstrip array with 1984 elements shown in Fig. 5*a* is calculated to present the performance improvement and the parallel efficiency of the proposed LU solver. The dimensions of the full array are $5.898 \times 0.549$ m and the operation frequency is 3.1 GHz, given a number of

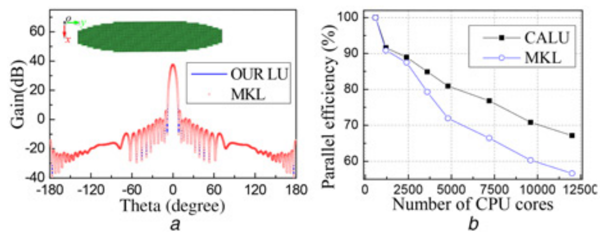

**Fig. 5** *Radiation pattern of the array and parallel efficiency*
*a* Radiation pattern of the array
*b* Parallel efficiency

**Table 4** Simulation time (unit: s) and parallel efficiency

| CPU cores | 600 | 1200 | 2400 | 3600 | 4800 | 7200 | 9600 | 12,000 |
|---|---|---|---|---|---|---|---|---|
| our LU | 5841.5 | 3187.4 | 1641.4 | 1146.6 | 902.1 | 633.6 | 515.4 | 434.9 |
| MKL | 6473.7 | 3562.3 | 1849.1 | 1360.7 | 1124.7 | 812.1 | 671.2 | 571.9 |
| speed-up | 1.108 | 1.118 | 1.127 | 1.187 | 1.247 | 1.282 | 1.302 | 1.315 |

unknowns 275,918. The amplitude at the feed of the array is designed by a −30 dB Taylor distribution both along length and width. The 2D gain patterns are also given in Fig. 5. The computing time listed in Table 4 indicates the improvement of the proposed LU solver by 10–30% against the Intel MKL and that the more the CPU cores are used, the better performance is improved. The parallel efficiency is evaluated according to Table 4 and shown in Fig. 5*b* (reference: 600 cores). It is worth noting that the Intel MKL obtain a much worse scalability when more CPU cores are used on MilkyWay-2 supercomputer; we will determine the cause in a future article.

## 5 Conclusion

In this study, a highly efficient LU solver is presented to help MoM for solving large electromagnetic problems. The own ZGEMM is about 11 times faster than the original one. The proposed LU solver is about 10% faster than the traditional LU solver. Both on the two platforms, a high parallel efficiency can be achieved.

## 6 Acknowledgments

This work was supported by the National High Technology Research and Development Program of China (863 Program) (2014AA01A302), the program for New Century Excellent Talents in University of China (NCET-13-0949).

## 7 References

[1] Zhang Y., Sarkar T.K., Taylor M.C., ET AL.: ‘Solving MoM problems with million level unknowns using a parallel out-of-core solver on a high performance cluster’. IEEE Antennas and Propagation Society Int. Symp., 2009
[2] Zhang Y., Lin Z., Zhao X., ET AL.: ‘Performance of a massively parallel higher-order method of moments code using thousands of CPUs and its applications’, *IEEE Trans. Antennas Propag.*, 2014, **62**, (12), pp. 6317–6324
[3] Blackford L.S., Choi J., Cleary A., ET AL.: ‘ScaLAPACK: a portable linear algebra library for distributed memory computers – design issues and performance’. Proc. of the 1996 ACM/IEEE Conf. on Supercomputing, 1996
[4] Dongarra J.J., Duff I.S., Sorensen D.C., ET AL.: ‘Numerical linear algebra on high-performance’ (Tsinghua University Press, Beijing, 2011)
[5] Intel Copyright, Developer Reference for Intel Math Kernel Library 2017-C, Intel Corporation. Available at https://software.intel.com/en-us/mkl-reference-manual-for-c, 2016
[6] Grigori L., Demmel J.W., Xiang H.: ‘Communication avoiding Gaussian elimination’. Proc. of the 2008 ACM/IEEE Conf. on Supercomputing, 2008
[7] Goto K., Geijn R.: ‘Anatomy of high-performance matrix multiplication’, *ACM Trans. Math. Softw.*, 2008, **34**, (3), pp. 1–25