Journal of Computational Information Systems 9: 14 (2013) 5561–5567
Available at http://www.Jofcis.com

A Parallel Method for Matrix Inversion Based on Gauss-jordan Algorithm

Kaiqi YANG*, Yubai LI, Yijia XIA

School of Communication and Information Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

Abstract

The inversion of matrix is widely used in many scientific applications and engineering calculations and this operation requires a large number of computational efforts and storage space. In order to reduce the consumed time and to increase efficiency, a parallel algorithm for matrix inversion based on classic Gauss-Jordan elimination with pivoting is proposed. Upon the multi-core DSPs platform, the parallel algorithm is implemented and evaluated. The experimental results show the efficiency and scalability of this algorithm.

Keywords: Matrix Inversion; Gauss-Jordan Elimination; Pivoting; Parallel Algorithm

1 Introduction

The inversion of matrix is often required in many engineering practices and scientific applications. And matrix inversion is believed to be an intricate computation process, because it requires a lot of computational efforts. For general matrices, there are a variety of existing matrix inversion algorithms (e.g. methods based on QR decomposition [1], methods based on LU decomposition [2] and methods based on singular value decomposition and so on), and most of them cost $O(n^3)$ operations which could not meet some current applications' demands (where n denotes the order of the matrix). There are several algorithms of matrix inversion with high performance for special matrices (e.g. tridiagonal matrix [3], adjacent pentadiagonal matrix [4], triangular matrix [5]), but they cannot be directly used in computing the inversion of general matrices. Therefore, the parallel algorithms for general matrices inversion are needed to be given.

In recent years, researchers have proposed several feasible parallel matrix inversion algorithms. For examples, a parallel method for matrix inversion with systolic arrays [6] which used pipelines to increase the systematic throughput is proposed as a feasible solution. However, this method is based on systolic array, and the circuit structure is difficult to implement. A block LU decomposition matrix inversion method based on a Network-on-Chip architecture [7] solved matrix

*Corresponding author.
Email address: 413827897@qq.com (Kaiqi YANG).

1553–9105 / Copyright © 2013 Binary Information Press
DOI: 10.12733/jcis6364
July 15, 2013

inversion through LU decomposition with a high degree of parallelism. But when the order of matrix grows bigger in this method, whether the block size selection is the smaller one or the bigger one will both cut off the performance, because the former choice needs more iterations and the latter one requires more time to inverse sub-matrices. Moreover, several block-based algorithms for matrix inversion via Gauss-Jordan elimination [8, 9, 10, 11] are proposed. However, a number of matrix-matrix products are required in these algorithms, which could cut off the performance, and the lacking of pivoting operation in these algorithms may cause some stability issues.

In this paper, a parallel method for matrix inversion based on Gauss-Jordan algorithm is proposed. In order to cut off the communication overhead, efficient communication mechanisms of multi-core DSP are used.

2 Review of Gauss-jordan Algorithm for Matrix Inversion

Gauss-Jordan algorithm for matrix inversion is based on Gauss-Jordan elimination, and it's a reordering of the computations performed by row operations [12]. The inversion of matrix can be finally written out from the augmented matrix which consists of the original matrix and an identity matrix.

In the period of row operations, some parts of the augmented matrix are transformed to be unit matrices and zero matrices, which are unnecessary to store. So those redundant parts of augmented matrix are discarded to get a new square matrix with the same order as the original matrix. By observing the process of row operations, there is no difficult to derive the following updating equations:

$$\alpha_{kk} = 1/\alpha^*_{kk} \tag{1}$$

$$\alpha_{kj} = \alpha^*_{kj} \times \alpha_{kk}, j = 0, 1, ..., n-1; j \neq k \tag{2}$$

$$\alpha_{ij} = \alpha^*_{ij} - \alpha^*_{ik} \times \alpha_{kj}, i, j = 0, 1, ..., n-1; i, j \neq k \tag{3}$$

$$\alpha_{ik} = -\alpha^*_{ik} \times \alpha_{kk}, i = 0, 1, ..., n-1; i \neq k \tag{4}$$

Where notation with $*$ represents the element before updating. Here α_{kk} is known as the main element, and the k-th row and the k-th column are called the main row and the main column respectively.

There is one more thing worth noting. Eq. (1) is a division operation, so if the absolute value of α_{kk} is too small, the computation process may cause a rounding error which could lead to a very wrong inversion result. In order to avoid the rounding error diffusion and to improve stability of the algorithm, a pivoting operation is needed to guarantee α_{kk} to be the element with the maximum absolute value. And the operation should be performed before every updating.

Fig. 1 shows the pivoting process. The main idea of pivoting is to swap the element with the maximum absolute value with the main element. So the first thing to do is to find the element with the maximum absolute value among those elements in the bottom-left corner of α_{kk}. Without loss of generality, here the element with the maximum absolute value is assumed to be $\alpha_{i,j}$, then swap the i-th row with the k-th row and swap the j-th column with the k-th column, respectively. For the matrix recovery later, the row number and column number of those two swapped elements should to be recorded.

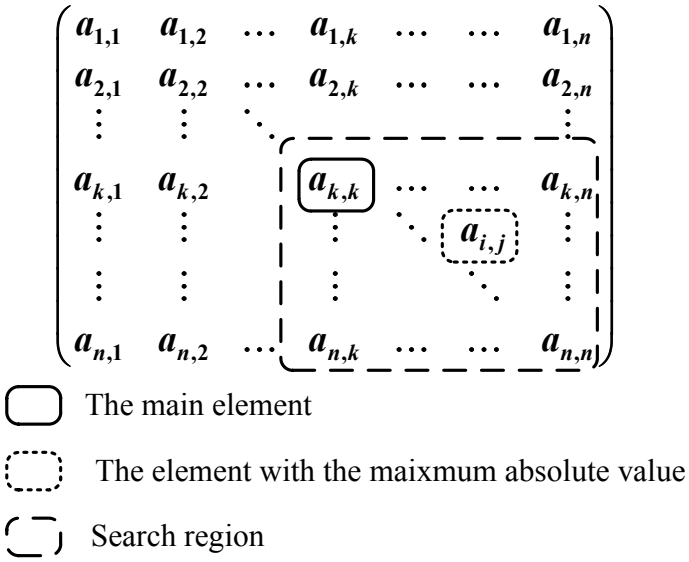

Fig. 1: Schematic diagram of pivoting

3 A Parallel Method for Gauss-jordan Algorithm

It's not difficult to see that the updating of a certain element in the matrix only relies on the main row as well as the row it locates, that is to say, the updating process among different non-main rows has nothing to do with each other. Once those elements in the main row have been updated, other rows can be updated simultaneously by Eq. (3) and Eq. (4), which provides convenience for algorithm parallelism.

To make use of the parallelism hidden in Gauss-Jordan algorithm, the original matrix is divided into several smaller sub-matrices which should be assigned to several cores respectively later, where the amount of those sub-matrices is in accordance with the number of those cores. Assuming p cores are available and the order of matrix A is n, the distribution of sub-matrices follows the following rules: distributing the jth row of the original matrix A to the core numbered (j mod p+1), and with all other rows distributed to the same core, those rows form one sub-matrix, and the original matrix A is divided into p sub-matrices finally. Here the number of rows of matrix A is not necessary divisible by p, so the last several sub-matrices could own one less row than other sub-matrices.

Fig. 2 shows the matrix division when 4 cores are available. Here, the order of matrix A is assumed to be divisible by 4. For the convenience of explanation, the core owns the main row is called the main core in this study. Apparently, once the main core updates the main row by Eq. (1) and Eq. (2), all other cores could update their sub-matrices simultaneously by Eq. (3) and Eq. (4) afterwards. The aforementioned is an update process, and the final result will reveal after n times of such updating. As mentioned in section 2, a pivoting process is also needed before updating.

When matrix A is already divided into p sub-matrices, the updating process of sub-matrices can be done in parallel, and the pivoting operation is handled by a single core. Here are the main steps of the parallel algorithm.

Step 1 Core 1 performs a pivoting operation on matrix A, and records the swapped row number and column number.

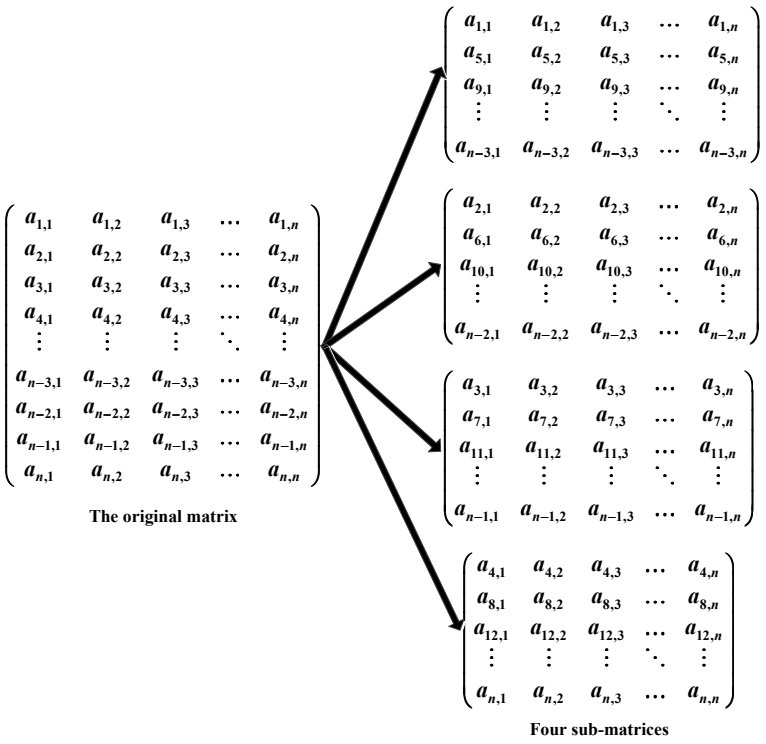

Fig. 2: Matrix division

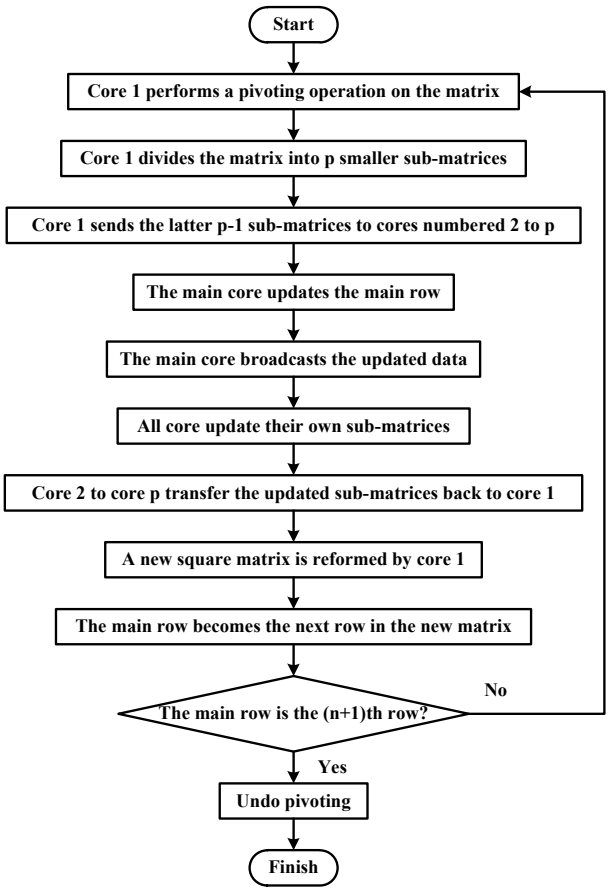

Fig. 3: Flow diagram of the algorithm

Step 2 Core 1 divides the matrix A into p smaller sub-matrices, and assigns those sub-matrices to the number of p cores.

Step 3 After the matrix division is finished, core 1 sends the latter p-1 sub-matrices to other p-1 cores, respectively.

Step 4 The main core updates the main row by Eq. (1) and Eq. (2), and broadcasts the updated data.

Step 5 While the main core updates other elements in its sub-matrix (the main row has been updated), all other cores perform an updating operation on their own sub-matrices based on the received data.

Step 6 Core numbered 2 to core numbered p transfer their own updated sub-matrices back to core 1.

Step 7 A new square matrix A^* is reformed from all of the updated sub-matrices by core 1.

Step 8 The main row becomes the next row in the new square matrix. Therewith, the main element becomes the next diagonal entry, and the main core goes to the next core. Then repeat step 1 to 7.

Step 9 When the nth update operation is finished, undo pivoting in accordance with those swapped row numbers and column numbers recorded in step 1. Then Matrix A is overwritten by A^{-1} finally.

In summary, this implementation enables overlapping the update of sub-matrix in the main core with the update of sub-matrices in all others cores, which makes the most computational part in the algorithm a parallel operation, with the price of $3n(p\text{-}1)$ times of data communication among several cores.

4 Experimental Results

The performance of Gauss-Jordan algorithm for matrix inversion with pivoting is tested on TMS320C6474 EVM, which consists of two DSPs with six cores at 1GHz each. The inter-DSP communication technique SRIO with the data rate up to 3.125Gbps and inter-core communication technique EDMA3 with the data rate up to 16.840Gbps on TMS3206474EVM are used as the data communication methods.

In order to compare the performance variation, the algorithm is tested on single core, double cores and four cores, respectively. Moreover, both the single core mode and double cores mode use the candidate cores all from DSP1, while the four cores mode use three cores from DSP1 and one core from DSP2.

Besides the evaluation of algorithm performance on different experimental modes, the correctness of the algorithm is also verified to assure that the final matrix A^* is the inversion of the original matrix. (e.g. by multiplying the final A^* by the original matrix A, a new matrix G is got and if the matrix G is equal to an identity matrix I, the final matrix A^* is for sure to be A^{-1}.). The verification work is done by MATLAB, and the result is proofed to be correct as expected.

In this platform, every core keeps its independent memory, and the advantage in storage allocation of the algorithm for a single core is revealed. When p cores are used to inverse an $n \times n$ matrix, core 1 needs to store all sub-matrices for the pivoting operation, and other cores only need to keep their own sub-matrices, each of which consists of n^2/p elements. The more of the running cores, the more memory space a single core could save.

The performance attained by the implementation of Gauss-Jordan algorithm running on single core, double cores and four cores for inversion of different size of matrices are shown in Table 1.

Table 1: The CPU clock cycles consumed on different implementation platform (1GHz CPU)

Order of matrix / **Number of cores**	**12×12**	**24×24**	**36×36**	**48×48**
Single core	1224192	9454199	31486992	74354646
Double cores	1242591	9640287	32343931	77696788
Four cores	1098168	8089545	26910182	66073290

From table 1, the experiment results demonstrate that the algorithm running on dual cores costs more time compared with algorithm on single core. The result could be explained by the tedious communication process concealed in the parallel algorithm. During each updating, $3(p\text{-}1)$ times of data exchanging are required, and n times of updating make the total number of times for data exchanging to $3n(p\text{-}1)$ when p cores are used to inverse an $n \times n$ matrix, which makes the communication overhead a high value. However, the comparison between algorithms on single core with that on four cores shows that the saved time of parallel algorithm can balance out the consumed time of communication process with the increasing number of processing cores. In consequence, the performance of Gauss-Jordan algorithm is promoted with the increasing number of parallel cores.

When compared with some mature and previously published algorithms, the shortage of this parallel method is exposed. For example, benefited from less communication overhead and fast method for matrix multiply, those block-based algorithms [8, 9, 10, 11] show a better performance than the algorithm discussed in this study, and the performance improvement is more obvious when the test platform changed from dual cores to four cores. However, the parallel algorithm proposed in this study shows good transportability, flexibility and scalability which many algorithms lack of. And as a novel method for matrix inversion which makes use of those efficient communication methods of multi-core DSPs, it has a reference significance in engineering applications.

5 Conclusion

In this study, a parallel method for matrix inversion based on Gauss-Jordan algorithm is proposed and evaluated. The operation of pivoting is used in this method to increase stability. Experimental results have demonstrated the performance improvement of this parallel method compared with the Gauss-Jordan algorithm. And further optimization is needed to decrease the communication overhead in this parallel method.

Acknowledgement

I would like to express my gratitude to the support of the National Major Projects [2011ZX03003-003-04] and the National Natural Science Funds [61201005].

References

[1] Chitranjan K. Singh, Sushma Honnavara Prasad, Poras T. Balsara, VLSI architecture for matrix inversion using modified gram-schmidt based QR decomposition, In Proceedings of the 20th International Conference on VLSI Design Held Jointly with 6th International Conference on Embedded Systems, January 6-10, 2007, Bangalore, India, pp. 836-841.

[2] Vijay Sahota, Richard Bayford, Jinv: A parallel method for distributed matrix inversion, In Proceedings of the Developments in E-systems Engineering (DESE), September 6-8, 2010, London, UK, pp. 163-167.

[3] Y. Huang, W.F. McColl, Analytical inversion of general tridiagonal matrices, J. Phys. A., 30(1997) 7919-7933.

[4] M. E. Kanal, Parallel algorithm on inversion for adjacent pentadiagonal matrices with MPI, J. Supercomput., 59(2012) 1071-1078.

[5] Florian Ries, Tommaso De Marco, Roberto Guerrieri, Triangular matrix inversion on heterogeneous multicore systems, IEEE Transac. Parall. Distrib. Syst. TPDS, 23(2012), pp. 177-184.

[6] Quan Sun, Ming Zhao, Xiujun Zhang, Implementation matrix inversion of LU algorithm with systolic array, Microelectr. Comp., 23(2007) 138-141.

[7] Yingying He, Yukun Song, Gaoming Du, Duoli Zhang, Research of matrix inversion acceleration method, In Proceedings of the International Conference on Computational Intelligence and Software Engineering (CiSE 2009), December 11-13, 2009, Wuhan, China, pp. 1-4.

[8] Pablo Ezzatti, Enrique S. Quintana-Orti, Alfredo Remon, High performance matrix inversion on a multi-core platform with several GPUs, In Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, February 9-11, 2009, Ayia Napa, China, pp. 87-93.

[9] N. Melab, E-G. Talbi, S. Petiton, A parallel adaptive version of the block-based Gauss-Jordan algorithm, In Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing, April 12-16, 1999, San Juan, China, pp. 350-354.

[10] Lamine M. Aouad, Serge G. Petiton, Parallel basic matrix algebra on the Grid'5000 large scale distributed platform, In Proceedings of the IEEE International Conference on Cluster Computing, September 25-28, 2006, Barcelona, pp. 1-8.

[11] Ling Shang, Serge Petiton, Maxime Hugues, A new parallel paradigm for block-based Gauss-Jordan algorithm, In Proceedings of the 8th International Conference on Grid and Cooperative Computing, August 27-29, 2009, Lanzhou, Gansu, China, pp. 193-200.

[12] E.S. Quintana, G. Quintana, X. Sun, A. Van De Geijn, A note on parallel matrix inversion, SIAM J. Sci. Comput., 22(2001) 1762-1771.

[13] Guoyong Mao, Xiaobin Zhang, Yun Li, Yujie Li, Laizhi Wei, Partition-Based Optimizing Algorithm for Dense Matrix Computation under Multicore Environment, Journal of Computational Information Systems, 2010, 6(14), pp. 4923-4931.