Journal of Engineering and Applied Sciences 13 (23): 9889-9891, 2018
ISSN: 1816-949X


# High Performance Matrix Inversion for Solving Linear Equations System

Hussein A. Lafta and Farah Abdul-Hassan
Department of Computer Science, College of Science for Women,
University of Babylon, Hillah, Iraq

**Abstract:** The linear algebra for solving a system of linear equations has an important role in many fields of science like Engineering, Physics, Statistics and Computer Science, etc. Matrix inversion play a primary role in solving these equations. A main challenge task is to invert a large-scale matrix with several thousands or millions of rows and columns about $2n^3$ floating-point operations. We aimed to solve the system of linear equations based on matrix inversion in a way that achieves a high performance and reduces the time-consuming. In our research, we implement a matrix inversion on programmable Graphics Processors (GPUs) using MATLAB language for programming and Nvidia video card. The results of our method gain a higher performance than the sequential methods with simplicity of coding.

**Key words:** Matrix inversion, linear algebra, parallel computing, graphical processing units, coding, sequential

## INTRODUCTION

Solving the linear equations system has an important role in many fields of science like Engineering, Physics, Statistics and Computer Science, etc. The general formula of the linear equation system is: Ax = b, where A is a non-singular matrix, x is unknown values that satisfy every equation in the linear system for the given matrix A. The b is a vector of right hand side values. The essential problem in solving linear equations is to find the unknown values of solution vector x (Murthy *et al*., 1998).

**Linear equation systems can be solved using two methods:** Direct methods such as (matrix inverse method, LU-decomposition by Gaussian elimination, Gaussian elimination with partial pivoting, cholesky algorithm and Iterative methods such as (Jacobi method, Gauss-Seidel method) as a comparison, the direct methods research best with dense matrices whereas iterative methods are working with sparse matrices.

Matrix inversion is one of the direct methods for solving linear equation systems, it plays a significant role in solving these systems. The inversion of matrices appears in many fields of scientific applications such as model reduction, image processing, social network, recommendation systems and optimal control. The fundamental challenge of linear equation systems is to find the inversion of a large-scale matrix with thousands or millions rows and columns about $2n^3$ floating-point operations where n is the number of rows and columns. In practice, the analytical solution for this systems of equations are quite computationally complex and taking a lot of time, especially as the dimensions of the system matrices increase (Jamil, 2012).

The matrix inversion can be used to reduce the computational efforts required to solve the systems of equations and acquire very accurate solutions in a short time. Our main purpose is to implement the matrix computations on GPUs (Graphics Processing Units), the performance is the major reason of using GPUs in matrix computations. The use of GPUs has increased considerably over the last few years because they improve the speed of graphics-related computations due to the dozens or even hundreds of cores they made of (Davies *et al*., 2007). The results of our method gain a higher performance than the sequential methods with simplicity of coding and efficient utilization of memory.

**Literature review:** This study presents some previous researches in implementing matrix inversion method for solving linear equation systems.

Ares *et al*. (2013), present a GPU-based matrix inversion algorithm for distributed memory contexts they used Gauss-Jordan elimination algorithm.

Sharma *et al*. (2013), redesign for matrix inversion by the Gauss Jordan algorithm on a CUDA platform they take advantage of the characteristic of large scale parallelization of a massively multithreaded GPU.

Ibearugbulem *et al*. (2014), present an iterative method for computing determinants and for solving eigenvalue which is one of the linear algebra problems they use a MATLAB iterative code and use a trial eigenvalue is to compute the determinant in their program.

**Corresponding Author:** Hussein A. Lafta, Department of Computer Science, College of Science for Women, University of Babylon, Hillah, Iraq

Abbas (2014) suggests a new parallel algorithm that is finding the determinants of block matrices in the order of (W×W), using Gauss elimination method.

Shirazi *et al*. (2015) propose a new version of the Gauss-Jordan matrix inversion method accelerated by Graphics Processing Units (GPUs). The results show that the proposed method is faster than baseline method. Gray (2016), uses multiple processors in parallel to find the inversion of a dense symmetric positive definite matrix.

## MATERIALS AND METHODS

**Parallel matrix inversion using Laplace expansion:** Our method depends on using determinants and Laplace expansions to find the matrix inversion to solve the linear equation system. Laplace expansion is a method for finding the determinant by expanding each matrix recursively to (sub-matrices) until reaching a base case of 2×2 matrix. Laplace expansion expresses a determinant |A| of the matrix A that has a n×n dimensions, it is representing the summation of all determinants of its -1×n-1 sub-matrices. Given an n×n matrix A such that:

$$A = \begin{bmatrix} a_{ij} \cdots a_{jn} \\ \vdots \ \ddots \ \vdots \\ a_{m} \cdots a_{n.n} \end{bmatrix} \quad (1)$$

where, $a_{ij}$ is the value of the pivot element that is lying in the ith row and jth column. Laplace can be calculated by the following Eq. 2:

$$C_{ij} = (-1)^{i+j} * M_{ij} \quad (2)$$

where, $M_{ij}$ is the minor of $a_{ij}$ in matrix A and c is the cofactors. And the determinant is computed by the following Eq. 3:

$$|A| = \sum_{j=1}^{n} a_{ij}.C_{ij} \quad (3)$$

Algorithm 1 shows the steps of calculate |A|, depending on Laplace expansion method and then compute the matrix inversion.

**Algorithm 1; Laplace expansion method for computing |A|:**

Step 1: find $M_{i,j}$, Create a sub-matrix recursively by eliminating the elements of each row and column in the $a_{i,j}$

Step 2: find the cofactor, $c_{i,j} = (-1)^{i+j}.M_{i,j}$

Step 3: if n = 2, $D_{ij} = (a_{11}*a_{22})-(a_{21}*a_{12})$

Step 4: $a_{i-1,j-1} = a_{ij}.D_{ij}$

Step 5: $D = \sum_{j-1}^{*} a_{i,j}, C_{i,j}$

Compute matrix inversion:

Step 6: Adj = transpose(c) where Adj is the Adjoint of $a_{i,j}$

Step 7: $A^{-1} = Adj/D$

Inverse of a matrix can be calculated only if the matrix is nonsingular and the determinant should not be zero. The linear equation system can be solved by multiply the matrix inverse by the absolute values vector to infer unknown values such that:

$$A^{-1}b = x \quad (4)$$

In our research, we execute the iterative code using MATLAB language to evaluate the inverse of a square and nonsingular matrix by calculating its determinant through Laplace expansion method, we use a three dimensional dynamic array to compute the determinant of the matrix and the final result of our code is the solution of the linear equation system. Then we applied (gpuArray) and (gather) build-in classes to run our code on the GPU which is supported by Nvidia video card.

## RESULTS AND DISCUSSION

**The implementation:** We use Parallel Computing in MATLAB Language which support the Nvidia video card (GPU) Intel HD Graphics GT 620. We use a two dimensional matrix of integers and a vector of the absolute values that represents (b) vector of the linear equation system Ax = b. The experimental results are as follow:

The original matrix is:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 9 | 8 | 3 | 5 | 8 | 7 | 2 |
| 1 | 9 | 3 | 3 | 7 | 4 | 1 | 1 | 10 |
| 6 | 6 | 9 | 6 | 7 | 8 | 3 | 9 | 10 |
| 8 | 6 | 3 | 7 | 2 | 8 | 1 | 10 | 5 |
| 10 | 10 | 10 | 9 | 2 | 2 | 1 | 7 | 9 |
| 2 | 3 | 4 | 10 | 5 | 5 | 9 | 8 | 2 |
| 6 | 8 | 2 | 6 | 10 | 5 | 7 | 8 | 5 |
| 5 | 8 | 3 | 2 | 4 | 7 | 4 | 4 | 10 |
| 1 | 4 | 7 | 2 | 6 | 8 | 10 | 7 | 8 |
| 4 | 6 | 5 | 3 | 3 | 8 | 1 | 2 | 10 |

The vector of absolute values:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | -1 | 0 | 3 | 5 | 6 | 2 | 3 |

The inverse of the matrix and the value determinant is: -93336855

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0.078039 | -0.116 | 0.019654 | -0.00964 | 0.031894 | 0.037287 | -0.06912 | 0.020453 | -0.038 |
| 0.150279 | -0.05778 | -0.12769 | 0.153558 | 0.085834 | 0.010618 | -0.01433 | 0.074317 | -0.23279 |
| 0.075895 | 0.049811 | -0.11623 | -0.00101 | -0.00829 | 0.131579 | 0.074269 | 0.098009 | -0.03805 |
| 0.070713 | 0.046697 | 0.027475 | 0.053212 | 0.041671 | -0.05798 | -0.09159 | -0.06845 | -0.02942 |
| -0.15768 | 0.132315 | 0.03034 | -0.05699 | -0.13875 | -0.00602 | 0.06462 | -0.07692 | 0.159701 |
| 0.02933 | -0.20145 | 0.146502 | -0.04009 | -0.00989 | -0.03679 | 0.023168 | 0.0089 | -0.03743 |
| -0.02909 | -0.09442 | -0.03667 | 0.157237 | 0.031013 | -0.00122 | -0.03759 | 0.038061 | -0.06689 |
| -0.04468 | -0.00509 | 0.029523 | -0.10826 | -0.00088 | -0.06805 | 0.075313 | -0.05148 | 0.137254 |
| -0.20369 | 0.327893 | -0.01303 | -0.12248 | -0.0852 | -0.00237 | 0.044824 | 0.052563 | 0.252922 |
| 0.050145 | -0.13034 | 0.16092 | -0.02743 | 0.089953 | 0.04189 | -0.08541 | -0.1647 | -0.08874 |

The solution vector which represents the value of unknown values (x):

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -0.54856 | -0.437 | -0.7099 | 0.4247 | 0.9599 | -0.1047 | -0.1681 | 0.1249 | 1.0101 | 0.0773 |

- Elapsed time in the sequential mode is: 0.007170 sec
- Elapsed time using GPU is: 0.006114 sec

## CONCLUSION

We implemented an efficient solution for a large system of linear equations which would provide a solution for many fields of science, i.e., Physics, Statistics, Computer Science etc. We focused on the implementation of matrix inversion method by Laplace expansion which solves scaled systems in multiple levels. Graphics Processing Unit (GPU) was use to implement our equation. Results prove that our method gain faster time from the traditional method.

## REFERENCES


Abbas, S.H., 2014. Computing the determinants in the multiprocessor computer. Intl. J. Innovative Res. Sci. Eng. Technol., 3: 15523-15530.

Ares, G., P. Ezzatti and E.S. Quintana-Orti, 2013. Towards a distributed GPU-accelerated matrix inversion. High Perform. Comput. Latam, 2013: 80-88.

Davies, R., P. Shi and R. Wiltshire, 2007. New lower solution bounds of the continuous algebraic Riccati matrix equation. Linear Algebra Appl., 427: 242-255.

Gray, A., 2016. Invertastic: Large-scale dense matrix inversion. ARCHER Whitepaper, 2016: 1-7.

Ibearugbulem, O.M., E.S. Osasona and U.J. Maduh, 2014. Iterative determinant method for solving eigenvalue problems. Intl. J. Comput. Eng. Res., 9: 28-31.

Jamil, N., 2012. A comparison of direct and indirect solvers for linear systems of equations. Intl. J. Emerging Sci., 2: 310-322.

Murthy, K.B., K. Bhuvaneswari and C.R. Murthy, 1998. A new algorithm based on Givens rotations for solving linear equations on fault-tolerant mesh-connected processors. IEEE. Trans. Parallel Distrib. Syst., 9: 825-832.

Sharma, G., A. Agarwala and B. Bhattacharya, 2013. A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA. Comput. Struct., 128: 31-37.

Shirazi, M., M. Kargahi and F. Khunjush, 2015. Gauss-Jordan matrix inversion speed-up using GPUs with the consideration of power consumption. Proceedings of the 5th International Conference on Advanced Communications and Computation (INFOCOMP), June 21-26, 2015, IARIA, Brussels, Belgium, ISBN:978-1-61208-416-9, pp: 20-25.