A Parallel implementation of Gram-Schmidt Algorithm for Dense Linear System of Equations

S. Roholah Ghodsi
PhD Candidate, Mechanical and Aerospace Eng**.** Dep**.**, Science and Research Branch, Islamic Azad University (IAU), Iran

Bahman Mehri
Professor, Mech. & Aero.Eng**.** Dep**.**, Science and Research Branch, IAU, Iran

Mohammad Taeibi-Rahni
Professor, Mech. & Aero.Eng**.** Dep**.**, Science and Research Branch, IAU, Iran

ABSTRACT
The linear system of equations with dense coefficient matrix is very common in science and engineering. In this paper, a parallel algorithm based on Gram-Schmidt QR factorization method for the exact solution of dense system of linear equations is presented. Although several parallel approaches have been proposed to solve the system of linear equations until now, the aim of this paper is to show the ability and limitation of this parallel algorithm in comparison with the sequential one. The suggested parallel algorithm is executed on MIMD architecture and distributed memory. In order to specify the efficiency of this algorithm, the amounts of speedup and FLOPs in executions with different size of matrix (from 2000 to 12000 equations) on up to 5 processors are compared together. The results show that the achieved speedup is significant, and also the performance of this practical parallel algorithm increases as the number of equations grows.

Keywords
Gram-Schmidt Method, QR Factorization, Parallel Processing, Dense Matrix, Linear system of equations.

2000 MR Subject Classification: 65F05, 65F25, 68W10

1. INTRODUCTION
Numerical linear algebra is an active field of research which provided many algorithms for the treatment of standard problems like the solution of system of linear equations. The problem of solving dense systems of linear equations has been dealt with the history of mathematics. Furthermore, this subject is one of the main problems in numerical computation, computer science, and different field of Engineering.

The numerical solution methods for linear systems of equations are broadly classified into two categories: direct methods, such as QR factorization; and iterative methods.

Direct methods obtain the exact solution in finitely many operations and are often preferred to iterative methods in real applications because of their robustness and predictable behavior. On the other hand, the term "iterative methods" refers to a range of techniques that use approximations to obtain more accurate solutions to a linear system at each step. Beginning with a given approximate solution, these methods modify the components of the approximation, until convergence is achieved.

The most suitable algorithm for a linear algebra problem depends on the properties of the system of equations like: non-zero elements, symmetry, real or complex coefficients, etc. Furthermore the scientist has to decide whether to use a direct solver, leading to a transformation of the original matrix and thus (for large problems) generating a need for enormous main memory and powerful processor, or to use an iterative solver which works with the original matrix.

Some methods for the solution of this problem involve triangularization of a matrix followed by back substitution. The triangularization procedure is computationally more complex than the back substitution. Therefore the scope of this paper is just on a parallel algorithm of orthogonalization and triangularization procedure, in order to verify its efficiency. The other procedures are done sequentially.

In the field of linear algebra, there are some high performance libraries, which are used widely in the scientific codes. The most famous libraries are Basic Linear Algebra Subprograms (BLAS) [**1**], Linear Algebra Package (LAPACK) [**2**], and Scalable Linear Algebra Package (SCALAPACK) [**3**]. The BLAS and LAPACK consist of sequential algorithms, which are used on single processor. On the other side, SCALAPACK is the package of parallel algorithms. From a practical point of view, the important decision is whether the user implements the linear algebra algorithm himself or relies on available libraries. Otherwise the user has to choose a parallelization scheme which best fits his specific application problem and he has to implement the necessary algorithms himself [**4**].

Up to now, excellent numerical methods for solving the system of linear equations on single and parallel systems have been developed, and many reliable and high-quality codes are available for different cases of linear systems [**5**, **6** and **7**]. The first real library of subroutines for linear algebra on dense matrices was developed in Algol by Wilkinson and Reinsch [**8**]. In 2002, Dunn and Meyer described three parallel QR factorization algorithms: distributed memory, Synchronous Message Passing, and Asynchronous Message Passing [**9**].

2. FORMULATION

The algorithm is based on Gram-Schmidt Orthogonalization and QR factorization. The formulation is briefly explained in this section.

Let $\{a_1, a_2, ..., a_n\}$ be a basis for a subspace S of an inner product space V. An orthonormal basis $\{u_1, u_2, ..., u_n\}$ for S can be constructed using the following Gram–Schmidt Orthogonalization process:

$$u_k = \frac{a_k - \sum_{i=1}^{k-1}\langle a_k, u_i\rangle u_i}{\left\| a_k - \sum_{i=1}^{k-1}\langle a_k, u_i\rangle u_i \right\|}, \qquad (1)$$

where $k = 2, \ldots, n$, and $\| \|$ is a norm on the vector space.

A QR factorization of $A \in \mathbb{R}^{n\times n}$ ($m \geq n$) is a factorization $A=QR$, where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $R \in \mathbb{R}^{n\times n}$ is an upper triangular. There are two algorithms to do Gram-Schmidt Orthogonalization process, the first one is based on Classical Gram–Schmidt Orthogonalization, and the second one is based on Modified Gram–Schmidt Orthogonalization [**10**]. The classical Gram–Schmidt is unstable numerically and so is never implemented in practice [**11**]. In this project, the second algorithm is used in order to produce orthogonalized matrix.

Now, consider a system of linear equations, $AX=B$, which A is a dense and non-singular matrix with non-zero elements. If $A=QR$ is a QR Factorization of matrix $A \in \mathbb{R}^{n\times n}$, then this linear system can be solved as follow:

1. Compute Q and R, from $A= \{a_1, a_2, a_3, \ldots, a_n\}$,

$$q_i = a_i + \frac{\overbrace{|a_i \,.\, q_1|}^{r(1,i)}}{|q_1 \,.\, q_1|} q_1 + \ldots + \frac{\overbrace{|a_i \,.\, q_{i-1}|}^{r(i-1,i)}}{|q_{i-1} \,.\, q_{i-1}|} q_{n-1}, \qquad (2)$$

so, $Q = \{q_1, q_2, q_3, \ldots, q_n\}$.

On the other hand, the upper triangular matrix R is computed from equation (2) simultaneously. The matrix R is shown as,

$$R = \begin{bmatrix} 1 & r_{1,2} & r_{1,3} & & r_{1,n} \\ 0 & 1 & r_{2,3} & \cdots & r_{2,n} \\ 0 & 0 & 1 & & r_{3,n} \\ & \vdots & & \ddots & \vdots \\ & & & \cdots & 1 \end{bmatrix}. \qquad (3)$$

2. Multiply the transpose of Q to the both side of system of linear equations,

$$Q^T Q\, R\, X = Q^T B. \qquad (4)$$

Consider $Q^TQ=Q'$, which is a diagonal matrix and then $Q'\, R = R'$. In the other side, $Q^TB=B'$. So,

$$R' X = B'. \qquad (5)$$

In this equation, R is an upper triangular matrix. Therefore the unknown matrix X is computed easily by back substitution.

3. HARDWARE SETUP

This numerical experiment has been done on a parallel system with MIMD (Multiple Data, Multiple Instruction) Structure, and distributed memory [**12**]. The library of parallel processing used in this paper is standard MPI [**13**]. Furthermore, the programming language is FORTRAN. The information of all computers and network, which are used in this experiment, is shown in table 1.

Table 1. The computers and network information

CPU	Pentium Dual-Core 3.00 GHz
Cache	2 MB
RAM	2 GB
FSB	1066
Network	Hub switch 16 port 100 Mbps

4. PARALLEL ALGORITHM

In order to solve a system of linear equations with numerical modified Gram-Schmidt QR factorization method, initially the orthogonal and upper-triangular matrices must be computed. Therefore, according to the Equation (2), the columns of the orthogonal matrix are computed one-by-one. In the computation of each column, the previous computed columns of orthogonal matrix are used. When a column of matrix Q is computed, automatically a column of the upper-triangular matrix is known as,

$$\begin{cases} r(i,j) = 1 & i = j \\ r(i,j) = 0 & i > j \\ r(i,j)\ From\ \ Eq.2 & i < j \end{cases}. \qquad (6)$$

It seems that, the solution is completely arranged in sequence. It should be considered that the denominator of each coefficient, in the computation of a column of orthogonal matrix in equation (2), consists of a term, which has been repeated in the computation of previous column, i.e. $|q_k \,.\, q_k|$. On the other hand, when a column of orthogonal matrix q_i is obtained, those terms in the next unknown columns (i.e. from q_{i+1} to q_n), which are related to current column can be computed simultaneously. Obviously, this is a parallel characteristic.

In order to parallelize this algorithm, first of all, the orthogonal matrix is set equal to the matrix A in all processors, because the first term in equation (2) is a_i. Then, each processor has responsibility for computing some columns of orthogonal matrix; an example is shown in Table 2.

Table 2: Distribution of columns of orthogonal matrix to different processors

Processor	*The Columns of Orthogonal Matrix*		
#1	q_2	q_{k+2}	...

#2	q_3	q_{k+3}	...
#3	q_4	q_{k+4}	...
⋮	⋮	⋮	⋱
#k	q_{k+1}	q_{k+2}	

When a processor completely computes a column, then it broadcasts the column for all other processors and sends its computed coefficients *r(1:i-1, i)* to the main processor. Therefore, all required data for computing the next terms of remaining columns is available for all processors.

This processing continue until, all columns of orthogonal matrix are computed. Afterward, the main processor computes the unknown matrix *X* sequentially, with back substitution procedure from excising orthogonal and triangular matrix. The flowcharts of single and parallel algorithm are shown in the Fig.1 and Fig.2, respectively.

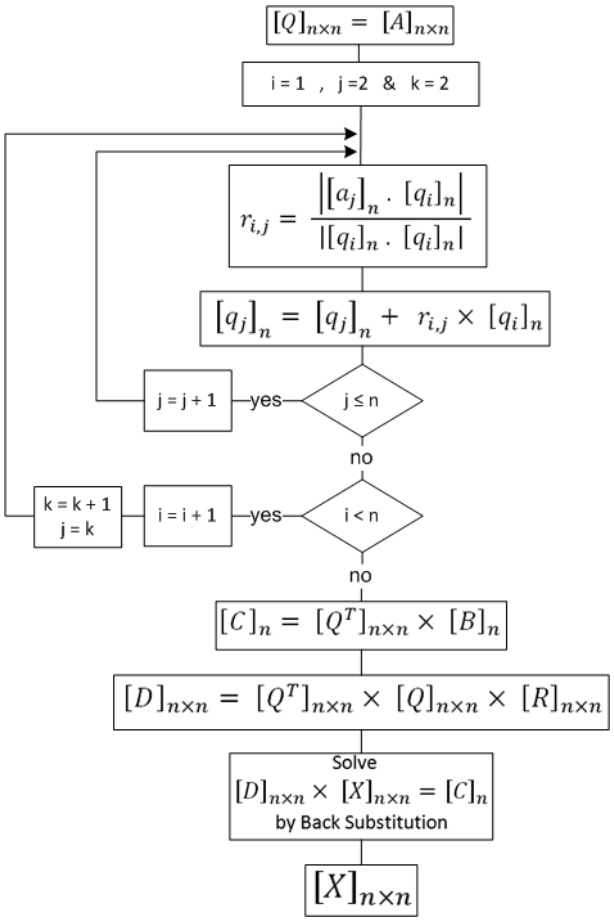

Figure 1: Solution of system of linear equations with Single QR Factorization algorithm

Because the scope of this paper is on the performance of parallelizing the process of generating orthogonal matrix, the back substituting is computed in the main processor lonely, as shown in Fig. 2.

5. EFFICIENCY AND SPEEDUP

The important thing in parallelizing an algorithm is efficiency of the original algorithm. The algorithms discussed here perform mathematical tasks that transform an initial data into a desired result using an ordered list of arithmetic operations, comparisons, and decisions.

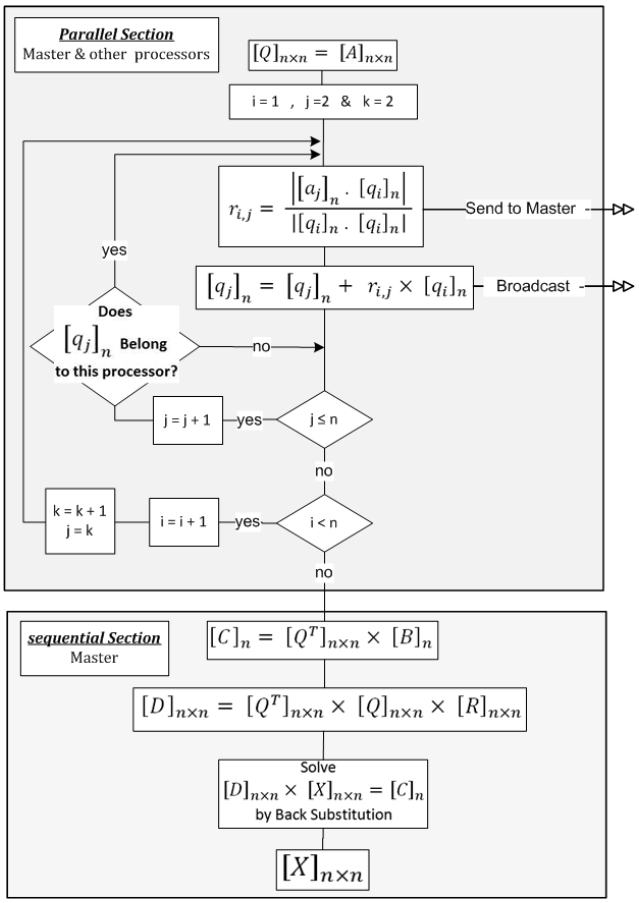

Figure 2: Solution of system of linear equations with Parallel QR Factorization algorithm

So, in general, in a given computational environment, the more efficient algorithm finishes its task sooner with the nearest result to the exact one. On computers, computation time was heavily dominated by evaluating floating point operations. If *n* measures the input matrix $A \in \mathbb{R}^{n\times n}$, then an $O(n^p)$ algorithm is one that, for some positive constant *c*, performs cn^p plus a sum of lower powers of *n* floating point operations [**14**]. In choosing a numerical method, efficiency must be balanced with considerations like robustness against rounding error and likelihood of failure.

In order to compare different algorithms to solve a system of equations *Ax*=b with $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^n$, the $O(n^p)$ of them are

noted in table 3 [**15**]. In detail, the computational work involved in an QR Factorization is of order $2n^3 + O(n^2)$. Furthermore, there are three other choices for orthogonalization [**16**]:

1. Householder: $(4/3)n^3 + O(n^2)$
2. Given: $(8/3)n^3 + O(n^2)$
3. Fast Given: $(4/3)n^3 + O(n^2)$

Obviously, the efficiency of QR factorization is acceptable. In addition, it returns exact result for unknowns of the system of linear equations.

Table 3: Comparison efficiency of different algorithm to solve a system of equations

Triangular back substitution (If A is triangular)	$O(n^2)$
Gaussian elimination with partial pivoting	$O(n^3)$
The QR-factorization	$O(n^3)$
SVD	$O(n^4)$
Gaussian elimination with complete pivoting	$O(n^4)$
Cramer's rule	$O((n+1)n!)$
Cramer's rule (determinants evaluated by Gaussian elimination)	$O((n+1)n!)$

The important thing is the efficiency of parallel algorithm and its ability and limitation. The key issue in the parallel processing of a single application is the speedup, especially its dependence on the number of processors used. Speedup is defined as a factor by which the execution time for the application changes,

$$S = \frac{T_s}{T_p}, \qquad (7)$$

where, p is the number of processor,
T_s = the executing time of the sequential algorithm,
T_P = the executing time of the parallel algorithm with p processors.

The achieved speedup on parallel system is shown in Fig.3.

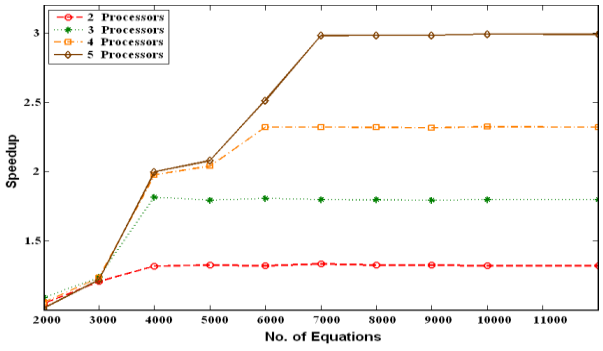

Figure 3: The speedup of different executions

Apparently, when the size of the matrix is not large, increasing the number of processor cannot change the speedup greatly. The reason is existing delay in communication. But when the number of equations grows, adding more processor can improve the speedup.

On the other hand, the efficiency of parallel processing is a performance metric, which is defined as,

$$E_p = \frac{S}{p}. \qquad (8)$$

It is a value, typically between zero and one, estimating how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. Fig. 4 reports the efficiency of the QR factorization on parallel computers. This graph shows a decline in efficiency by increase in the number of processors, which is happened because of communication delay.

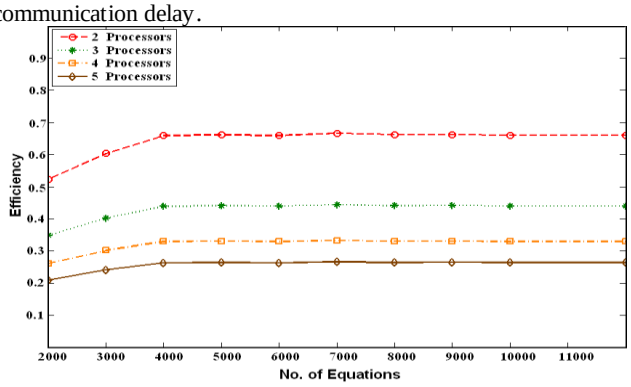

Figure 4: The Efficiency of the different executions with different number of processor

According to the obtained results, this parallel algorithm depends on hardware. If the computational power increases, the efficiency of parallel algorithm decreases. However, the challenges of numerical efforts are the lack of computational power and the size of problems. The progress in computational power has a great impact on many of numerical algorithms. From other point of view, when the size of matrix grows, the efficiency of this parallel algorithm becomes better.

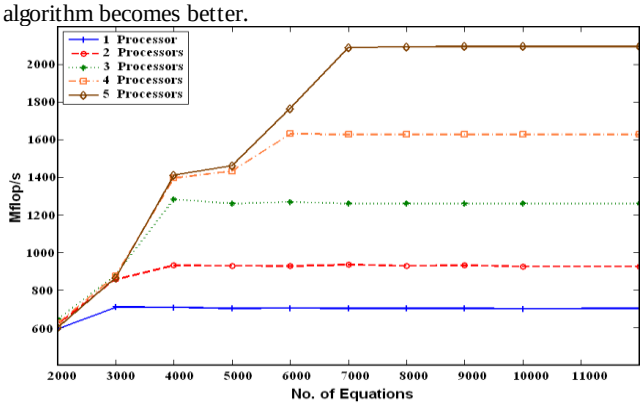

Figure 5: The Mflops of the executions with different number of processor

Fig. 5 presents the FLOPS (floating Point Operations per Second) of different cases. We obtained a peak performance of 2.094

GFLOPS on a 5-processor for 12000 equations. The single processor rate for this size of matrix was measured at 597 MFLOPS.

Fig. 6 shows the executing time of computations with different number of processors. It can be observed that time is reduced with the increase in number of processors, whereas the Communication time rises.

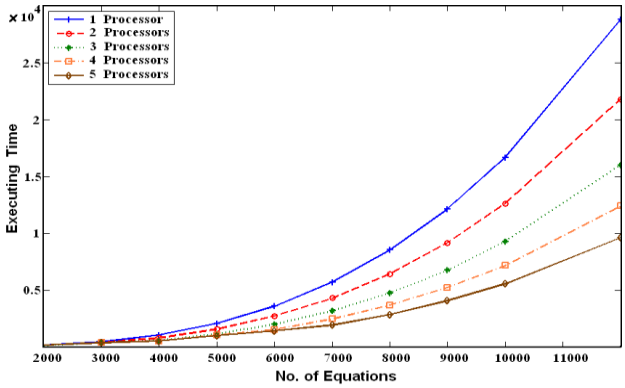

Figure 6: The executing times of different executions

As a result, the Fig. 4 presents that the method has limitation to improve parallel processing. However, the Fig. 3 and 6 shows the ability of this algorithm to increase the speedup and decrease executing time in comparison with single processing.

On the other hand, the efficiency of Gram-Schmidt QR factorization is $O(n^3)$, which is acceptable in comparison with other method. Furthermore, it is shown that, the described parallel algorithm is relatively successful in speedup. Hence, it seems that, the proposed parallel method has remarkable ability to compare with other efficient methods (it is not in scope of this paper).

6. CONCLUSION

In this paper, we demonstrated a parallel algorithm based on the Gram-Schmidt QR factorization, which can be used for direct solution of an arbitrary system of linear equations. We analyzed in detail the implementation of our parallel algorithm using up to 5 processors.

An efficient parallel solver was presented to reduce the computational time. Whatever, the number of equations increases, the performance of the algorithm becomes better and more processors can be used efficiently. Obviously, the number of unknowns and equations in real problems are extra large, so the concept of this algorithm can be helpful to solve them exactly.

The purpose of this paper was not to compare the efficiency of this method with other solution of system of linear equations; it showed just the limitation and ability of described parallel method.

In future, we will optimize this parallel algorithm to solve the system of linear equations with tridiagonal coefficient matrix.

7. REFERENCES

[1] Demmel, J. 1995. LAPACK Users' Guide. Society for Industrial and Applied Mathematic, 2nd Sub edition.

[2] Blackford, L.S., Choi, J., Cleary, A., DAzevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R.C. 1987. ScaLAPACK user's guide. Society for Industrial Mathematics.

[3] Körfgen, B., Gutheil, I. 2006. Parallel Linear Algebra Methods. Comp. Nanoscience: Do It Yourself, NIC Series, Vol.31, 507-522.

[4] Heller, D. 1978. A Survey of Parallel Algorithms in Numerical Linear Algebra. SIAM Review, Vol.20, No.4, 740-777.

[5] Saad, Y. 2003. Iterative Methods for Sparse Linear Systems. SIAM press, Philadelphia, 2nd edition.

[6] Fernando, G., Tinetti, D., Giusti, A. 2004. Parallel Linear Algebra on Clusters. 3rd International workshop on Parallel Matrix Algorithms and Applications (PMAA'04), CIRM, Marseille, France.

[7] Wilkinson, J.H., and Reinsch, C. 1986. Handbook for Automatic Computation. Vol. 2: Linear Algebra. Springer press, Berlin.

[8] Dunn, I.N., and Meyer, G.L. 2002. QR factorization for shared memory and message passing, Parallel Comp. Vol.28, No.11, 1507–1530.

[9] Trefethen, L.N., and Bau, D. 1997. Numerical Linear Algebra. SIAM press, Philadelphia.

[10] Meyer, C.D. 2000. Matrix Analysis and Applied Linear Algebra. SIAM, Philadelphia.

[11] Wittwer, T. 2006. An Introduction to Parallel Programming. VSSD uitgeverij, Netherlands.

[12] Snir, M., and Gropp, W. 1998. MPI: the Complete Reference. The MIT Press, Cambridge, 2nd edition.

[13] Hogben, L. 2006. Handbook of Linear Algebra (Discrete Mathematics and its Application), Chapman & Hall/CRC, Boca Raton, 1st edition.

[14] Golub, G.H., and Van Loan, C.F. 1996. Matrix Computations. The Johns Hopkins University Press, Maryland, 3rd edition.

[15] Lioen, W.M., and Winter, D.T. 1992. Solving Large Dense Systems of Linear Equations on Systems with Virtual Memory and with Cache. Applied Numerical Mathematics, Vol.10, No.1, 73-85.