Dijlah Journal of Engineering Science (DJES) Vol. 2, No. 3, September, 2025, pp. 14-25
ISSN: Printed: **3078-9656, Online: 3078-9664**, paper ID: 48

Parallel Program for calculation of the Matrix determinant of the Nth order

Ehab Abdulrazak. Alasadi[1],
[1]Department of Islamic Science, University of Kerbala, Kerbala, Iraq

Article Info

Article history:

Received 13, July, 2025
Revised 12, Aug., 2025
Accepted 20, Aug., 2025

Keywords:

Parallel program,
PVM,
Distribute system.

ABSTRACT

Analyze the possibilities of implementing a parallel algorithm for calculating the determinant of the Nth order (by modifying it to a suitable form). Design and implement (in C/C++) a solution based on sending messages between nodes using the PVM system library. Distribute the load between nodes so that the calculation time is as small as possible. Find out how the execution time and calculation acceleration depend on the number of nodes and the size of the problem (provide a table and graphs). Based on the results, estimate: the communication latency, for what size the task is (well) scalable on a given architecture, and what is the maximum size at which the calculation is still bearable on the available architecture. Discuss the advantages and/or effectiveness of parallel implementation of individual algorithms. If the solution requires it, use files for input and output of matrices where the row of the matrix corresponds to the row of the file and the column values are separated by spaces or tabs. Unless otherwise stated, work with real numbers

Corresponding Author:

Ehab Abdulrazak . AL-ASADI
Department of Islamic Scnience , University of Kerbala
Kerbala, Iraq
Email: ehab.a@uokerbala.edu.iq

1. INTRODUCTION

A distributed system is an application consisting of multiple components
running simultaneously on different computers. These computers must be able to communicate with each other and be able to work independently, Parallel computing is the simultaneous calculation of a single task on multiple processors
in order to speed up the calculation. The processor can be either the CPU in a computer or a single node (computer) in a distributed system. Distributed systems are used for parallel computing in various branches of science. In these systems, a task is divided into multiple subtasks that are independent of each other, which are then calculated simultaneously on individual computers, thus reducing the calculation time. Such systems are either homogeneous or heterogeneous

PVM (Parallel Virtual Machine) is a software package that allows the creation of heterogeneous computer clusters by connecting computers with UNIX or Windows systems, interconnected by a network. This system is freely available, highly portable and easy to use, which has contributed to its widespread use for scientific purposes.
PVM is a so-called message passing system, i.e. a system in which
parallel tasks synchronize and exchange information by sending

messages. The PVM system itself consists of a daemon that runs on each node of the cluster and a library that is compiled into the user program. A program using the library services can be written in various programming languages, most commonly,**[3],[4].**

The determinant is a function dependent on the dimensions of a matrix that assigns a scalar value to each square matrix. This value generally expresses the size of the matrix. In the following text, we will understand the determinant to mean this scalar value.Using the determinant of a matrix, its singularity can be determined. The determinant is also used to solve systems of linear equations using Kramer's rule, determine the eigenvalues of a matrix, the orientation of the coordinate system, and solve other problems.Determining the determinant of a matrix, especially for matrices of larger dimensions, is a computationally demanding task for which it makes sense to design a parallel algorithm

2. METHOD

The calculation of the determinant generally consists of 2 steps:

1. Modifying the matrix to a suitable equivalent form
2. Calculating the determinant from this form in a suitable way

In the first step, suitable equivalent modifications are applied to the matrix in turn. Its output is a matrix of the desired form, from which it is possible to calculate the determinant relatively easily apply one of the known methods of calculating the determinant from a matrix to the resulting matrix. The first step is not necessary and serves only to speed up the calculation.

Modifying a matrix to a suitable form

When modifying a matrix from matrix A to matrix B, there are several rules regarding its determinant:

If B was created from A

• by swapping two rows or columns, then det(B) = -det(A)

• by multiplying one row or column by a constant c, then det(B) = c.det(A)

• by adding the product of one row to another row, or by adding one column to another column, then det(B) = det(A)

The determinant calculated from the modified matrix must therefore be subjected to the appropriate operations, depending on the rules used in the modification.

Calculating the determinant from the modified matrix

The determinant can be calculated directly from the matrix in several ways:

Calculating the determinant by definition

Let the matrix A = (Ai,j), (where Ai,j is the matrix element in the i-th column and j-th row) be square. Then

• for matrix dimension A = 1, the determinant is det(A) = A1,1

• for matrix dimension A = 2, the determinant is det(A) = A1,1 A1,2 – A2,1 A1,2

• for matrices of dimension n, the Leibnitz formula can be used

det(A) = sum(sign(p).prod(Ai,p))

where

• sum is the function of the sum of the terms over all their permutations

• sign is a function whose output is 1 if an even permutation p follows and -1 if an odd permutation p follows (which was created by an even or odd number of exchanges of elements in it)

• prod is the function of the product of terms for i = 1 to n

Example of the method of calculating the determinant according to Leibnitz's formula for n = 3:

det(A) = A1,1 A2,2 A3,3 + A1,3 A2,1 A3,2 + A1,2 A2,3 A3,1 +

+ A1,3 A2,2 A3,1 + A1,1 A2,3 A3,2 + A1,2 A2,1 A3,3

This results in n! sum terms, so the method is not suitable for large matrices.

Laplace Decomposition

Using the Laplace decomposition, the determinant of a matrix can be calculated by decomposition along any row or column. The calculation by decomposition of row i of a matrix A of dimension n can be written as follows:
det(A) = sum(Ai,j (-1)i+j Mi,j)
where
• sum is the function of the sum over all j = 1 to n
• Mi,j is the determinant of the submatrix of matrix A, which was created by removing row i and column j from matrix A.

Additional rules for calculating determinants

• The definition of the determinant implies that the determinant of a triangular matrix is equal to the product of the members along the main diagonal, i.e. A1,1 to An,n for a matrix of dimension n
• For square matrices A and B,
det(AB) = det(A)det(B).
• The determinant for a matrix with a row or column of zeros is equal to 0.
• The determinant of a matrix with 2 identical rows or columns is equal to 0.

Similar methods for solving by modifying a matrix

Gaussian elimination method

Gaussian elimination method is an algorithm whose input is an arbitrary matrix and the output is a matrix containing rows defined as follows:
The first non-zero term from the left is equal to 1, the other terms are zero.

This result is achieved by the operations of exchanging rows, adding a row and the multiple of another row, and dividing rows.

The algorithm has a complexity of O(n3) and is numerically quite unstable when working with real numbers. To increase stability, so-called pivoting is used, which is the rearrangement of rows so that the largest term of the column is always on the main diagonal (this is partial pivoting).

After completing the Gaussian elimination algorithm, it is easy to calculate the determinant from the matrix thus obtained as the product of the terms on the main diagonal.

3. RESULTS AND DISCUSSION

Rough algorithm design

Solutions based on the definition of the determinant or Laplace's decomposition formula are too complex to be used effectively to calculate matrices of larger dimensions (for n = tens to hundreds).

The complexity of the elimination algorithm (matrix modification) is approximately O(n3), which is acceptable even for larger n. It is therefore advantageous to base the solution on this algorithm.

Basic steps of the algorithm

1. Finding a pivot for each row and rearranging the matrix so that the pivots are on the main diagonal (partial pivoting).
2. Modifying the matrix to the form of a lower (or upper) triangular matrix.
3. Calculating the determinant as the product of the terms on the main diagonal of the triangular matrix.

It will be desirable to implement parts 1 and 2 of the algorithm distributed.

Detailed algorithm design - partial pivoting

Description

In each column of the matrix, the largest number is found from the numbers below the main diagonal of the matrix. The row containing this number is swapped with the row containing the element of the main diagonal of this column. The change in sign of the resulting determinant is noted.

Pseudocode

(A[i][j] is the element at the i-th row and j-th column)

```
det_sign = 1 //1 = +, 0 = -
for (i = 0; i < size; i++)
    max_val = A[i][i]
    max_index = 0
    for (j = i+1; j < size; j++)
        if (A[j][i] > max_val)
            max_val = A[j][i]
            max_index = j
        end_if
    end_for
    if max_index != 0
        swap rows with indices i and max_index
        det_sign = 1-det_sign
```

```
end_if
end_for
```

Detailed algorithm design – modification to lower triangular matrix

Description

In each column i, for each row j, below the main diagonal, calculate the quotient of the member in column i and row j with the member in column i and row with the main diagonal – row i*. Then, it multiplies row i by this value (this is an intermediate result and is not stored in the matrix) and then subtracts this row from row j.

at this point, it is necessary to check whether the member A[i][i] is not equal to zero. If so, it is necessary to find the pivot in column i in one of the rows below the diagonal and swap these two rows.

Pseudocode

```
for(i = 0; i< dimension; i++)
if A[i][i] == 0
//pivoting for this column and rows below //diagonal
end_if
if A[i][i] == 0
return det = 0
end_if
for(j = i+1; j < dimension; j++)
share = A[j][i] / A[i][i]
row j = row j – (row i * share)
end_for
```

Parallelization of the algorithm

Point 1 and especially point 2 of the proposed algorithms are computationally demanding.

Unfortunately, in point 1, the results of finding pivots in the following columns depend on the modifications that were performed on the previous columns (since entire rows are exchanged). This fact makes parallelization of this part of the algorithm very difficult.

Point 2 – modifying the matrix to lower triangular is easier to adapt to parallel calculation. Although it would be difficult to decompose the calculation of individual columns, a relatively simple parallel calculation of new row values is possible, since these are independent of each other.

There is little point in modifying point 3 of the algorithm into a parallel form, since this is only a computationally fairly undemanding part**.[6]**

Modified pseudocode of point 2.

```
for(i = 0; i< dimension; i++) //serial part
if A[i][i] == 0
//pivoting for this column and rows below //diagonal
end_if
if A[i][i] == 0
return det = 0
end_if
for(j = i+1; j < dimension; j++) //parallel part
//distribution of rows to nodes
share = A[j][i] / A[i][i]
row j = row j – (row i * dimension)
//collection of changed rows from nodes
end_for
```

Description of modified point 2 of the algorithm

The central computer sequentially passes through all columns of the matrix (total dimension of cycles). After handling the condition when the term on the diagonal is zero, in each cycle it is necessary to process (dimension – index_of_current_cycle) rows. These are distributed as evenly as possible among all nodes (the central computer may be among them). After determining the value of all rows, the next column follows – the next cycle.

Data transferred between computers and synchronization

At the beginning of the cycle, it is necessary to calculate new values for (size – index_of_current_cycle) rows. In addition to the row being processed, each node also needs a row in which the given column contains a member on the main diagonal (in pseudocode, this is row i). So, at the beginning of the cycle, each node receives ((size – index_of_current_cycle)/number_of_nodes + 1) rows. After processing, it sends ((size – index_of_current_cycle)/number_of_nodes) rows back to the central computer. Sending rows also solves synchronization - after receiving all rows from this column, the central computer moves to the next column and starts the next cycle.

Input and output requirements

The input must be a square matrix with elements from the set of real numbers.

The proposed algorithm has a relatively high complexity; therefore, it is advisable to check the size of the input matrix and for very large n not to run the algorithm at all (the limit will be determined experimentally).

The output will be the determinant of the matrix if the calculation is successful. The alternative output will be a value signaling the non-existence of a determinant for the given matrix and in other cases an error output.

PROGRA M IMPLEMENTATIONS

The application was implemented in C++ using an object-oriented approach.

The core of the application are the following classes:

• Matrix class – is used to store a matrix and work with the values it contains.

• slaves class – used to create and terminate subordinate processes intended for calculation (slaves) and communicate with them

The following methods are important:

• Matrix::load – loads a matrix from a file on disk

• Matrix::pPivot – finds the pivot (the member with the largest absolute value) in one of the columns of the matrix and swaps the rows so that the pivot is on the main diagonal

• Matrix::dTrojuholnik – adjusts the matrix to lower triangular

• Matrix::det – calculates the determinant of a lower triangular matrix

• slaves::slaves – constructor, creates a specified number of slaves

• slaves::~slaves – destructor, sends termination messages to all slaves

• slaves::isend, slaves::dsend – wrapper functions for sending int and double values to slaves[**1],[3].**

• number of nodes on which slaves are running (n [nodes])*

* it is enough to monitor the number on which slaves are running, because only these perform the computationally/time-intensive part

Table (1) Dependence of total Time on the number of nodes and matrix size

n / size	**sekv.**	**1**	**2**	**4**	**6**	**8**	**10**
100 x 100	**0,0707**	**3,1628**	**2,328**	**1,878**	**1,791**	**1,896**	**1,8045**
200 x 200	**0,5964**	**23,205**	**13,5**	**11,57**	**10,8917**	**10,2314**	**11,2783**
400 x 400	**4,755**	**111,451**	**86,160**	**71,435**	**63,605**	**62,900**	

masterTime[s] = userTime + systemTime

Table(2) Dependence of master Time on the number of nodes and matrix size

size \ n	**sekv.**	**1**	**2**	**4**	**6**	**8**	**10**
100 x 100	0,072	0,4933	0,585	0,604	0,58	0,6147	0,6227
200 x 200	0,5918	3,2233	3,3983	3,6211	3,6717	3,9429	3,8717
400 x 400	4,7525	20,930	21,33	23,77	24,44	24,75	

Tables description
The time values in the tables are the average of several measurements. The number of measurements depended on the size of the problem (length of calculation) and time options. For a 100x100 matrix, each measurement was performed at least 10 times. For a 200x200 matrix, at least 5 times. For a 400x400 matrix, at least 2 times. For a 400x400 matrix, the results are therefore more of an indicative nature; this matrix was used to better estimate the maximum size of the task. The value for a 400x400 matrix and 10 nodes was not measured for time reasons.
The value of masterTime[s] was created by the sum of the values of userTime and systemTime. It is not used to answer important questions, but only to show the length (amount) of the master calculation compared to the total calculation time.

The values in the column marked “seq.” correspond to the measured values when using the sequential version of the algorithm. A graphical representation of the measured values can be found in Appendix A of this document.

Speedup calculation

S(s,n) = T(s,1)/T(s,n)

Where
• T(s,1) is the computation time of the sequential algorithm
• T(s,n) is the computation time of the parallel algorithm

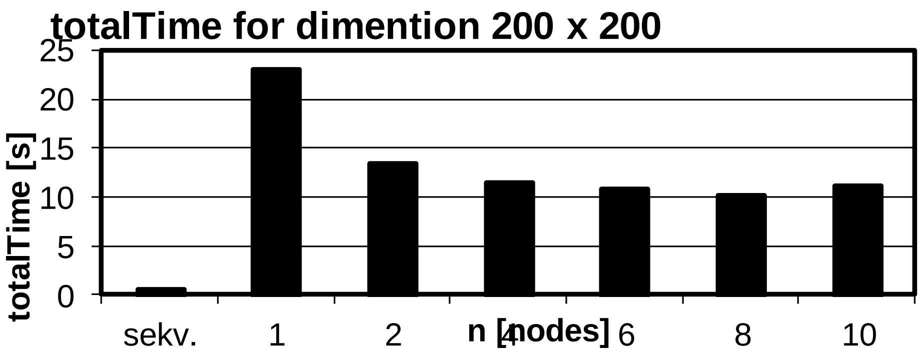
Table(3) Dependence of acceleration on the number of nodes and matrix size

size \ n	1	2	4	6	8	10
100 x 100	0,0224	0,0304	0,0376	0,0395	0,0373	0,0392
200 x 200	0,0257	0,0442	0,0515	0,0548	0,0583	0,0529
400 x 400	0,0427	0,0552	0,0666	0,0748	0,0756	

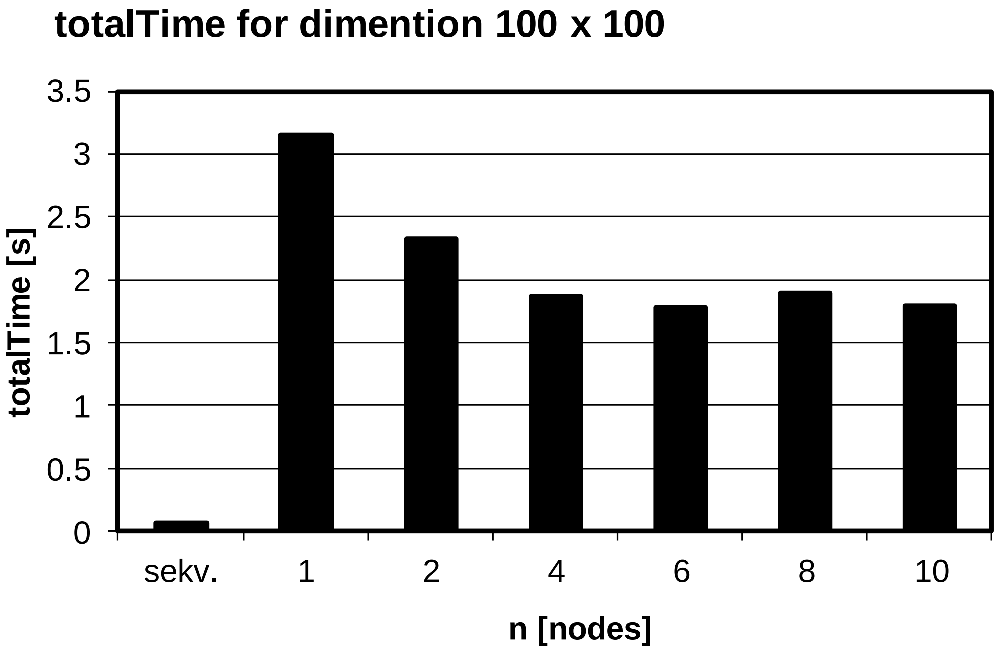
4. CONCLUSION

The result is speedup values from 0.02 to 0.08, which are very bad values. They are much smaller than 1, which means that the sequential algorithm was significantly faster than the parallel version of the algorithm. This is due to the significant additional overhead in the parallel algorithm and the load on the bus through which the master communicates with the slaves (and the subsequent communication delay). The overhead of the parallel algorithm is due to the rather fine granularity of the algorithm and consequently the large amount of data that the slave needs to work.

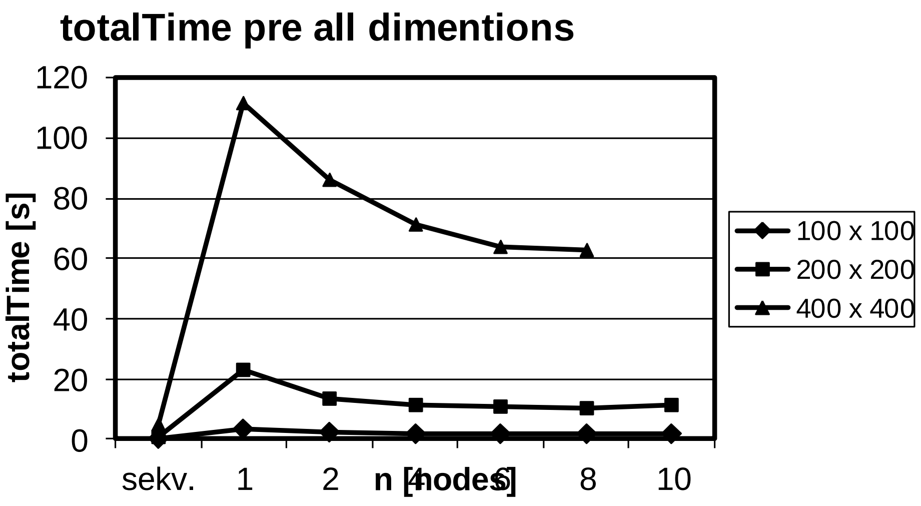
A graphical representation of the dependence of the speedup on the number of nodes and on the size of the task can be found below.



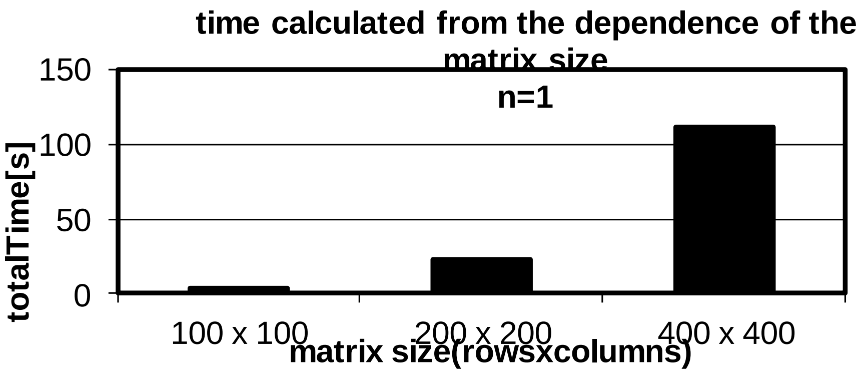
Fig(1) total time vs no of nodes



Fig(2) total time vs no of nodes



Fig(3) Total time vs no of nodes



Fig(4) time vs matrix size

REFERENCES

[1] YoLinux: YoLinux Tutorial“POSIX thread (pthread) libraries“, http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html,accessed 15.Dec.2024

[2] Eric W. Weisstein“Geometric Mean“ http://mathworld.wolfram.com/GeometricMean.html, accessed 15.Dec.2024 .

[3] Dick. B,et al," PThreads Programming: A POSIX Standard for Better Multiprocessing , O'Reilly Media,1996.

[4] David.B," Programming with POSIX Threads, Addison-Wesley Professional,1997.

[5] W. S, Stephen .R," Advanced Programming in the UNIX Environment, Addison-Wesley Professional,2013.

[6] Hesham El-Rewini , "Advanced Computer Architecture and Parallel Processing", Wiley-Interscience,2008.

[7] Alasadi, E. (2024)." Parallel algorithm for testing the singularity of an N-th order matrix", Al-Bahir Journal for Engineering and Pure Sciences, 4(2), 1.

[8] RAl-Asadi, E. A. (2015). "Finding N prime numbers using distrusted computing PVM (parallel virtual machine)". Int. J. Eng. Technol, 5(11), 578-588.

BIOGRAPHIES OF AUTHORS

The recommended number of authors is at least 2. One of them as a corresponding author.
Please attach clear photo (3x4 cm) and vita. Example of biographies of authors:

Author 1 picture	**Lecturer. Ehab A. Alasadi**, Received His MSc. degree in the computer engineering from University of Technology In Slovakia –in 2009 and Doctor of. He has been a full-time lecturer in Islamic Science collage -, Kerbala, Iraq, since March 2010. He, ongoing He can be contacted at email: ehab.a@uokerbala.edu.iq