Solving Linear Systems And Finding Determinants Using MPI

Michael Spagon
Tiffany Tillett

Introduction

- We implemented algorithms to calculate determinants and solve linear systems of equations
- We applied Gaussian Elimination to both of these problems, but each problem uses a different version of the algorithm
- For each application, we will discuss:
  - The mathematical concepts
  - The algorithm
  - The results of the experiment

Determinant

The determinant is a mathematical property that can be used to calculate the area of a triangle, the equation of a line and other useful items.

For a 2x2 matrix, it is defined as:

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

In the 3x3 case, the determinant is defined as:

$$\begin{aligned} |A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= aei + bfg + cdh - ceg - bdi - afh. \end{aligned}$$

Naive Recursive Implementation

Due to the definition, it makes sense to implement a recursive solution with the 2x2 matrix as the base case.

Pseudocode:

```
● Det = 0
● For each element in the first row of the matrix
    ○ Create a sub-matrix excluding the row and column of the current element
    ○ Recursively calculate the determinant on that submatrix and apply the
      correct +/- sign
    ○ Det += element * sub_det
```

Analysis of Recursive Implementation

Creating the sub-matrix will take O(n^2) steps

We make O(n) recursive calls

We have n iterations of the loop

So both time and work are = O(n^4)!

Parallel Recursive Implementation

If we have n processors, we can do the calculation for each column in the matrix simultaneously.

This helps us a bit but we are still O(n^3). This is not ideal.

We did implement this solution. However, due to the poor performance, it times out for matrices larger than 10x10.

Gaussian Method

Has two phases:

- Gaussian elimination
- Reduce the elements of the diagonal to get determinant

Elimination

Goal: Reduce the matrix A to upper triangular form U:

$$\begin{bmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-1} \\ & & \dots & \\ 0 & 0 & \dots & u_{n-1,n-1} \end{bmatrix}$$

Elimination

By applying a series of ***equivalent*** transformations, which do not change properties of the matrix:

- **Multiplication** of matrix row by a nonzero constant.
- **Permutation** of matrix rows.
- **Addition** of any matrix row to any other matrix row.

Elimination Example

Subtract 2 times the first row from the second

$$\begin{bmatrix} 1 & 3 & 2 \\ 2 & 7 & 5 \\ 1 & 4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 18 \\ 26 \end{bmatrix}$$

Elimination Example

Subtract the first row from the third row.

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 1 & 1 \\ 1 & 4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 16 \\ 26 \end{bmatrix}$$

Elimination Example

Subtract the second row from the third row

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 16 \\ 25 \end{bmatrix}$$

Elimination Example

Upper Triangular Matrix Found!

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 16 \\ 9 \end{bmatrix}$$

Elimination Pseudocode

```
for $k = 1, n - 1$
    for $i = k + 1, n$

    $\ell_{ik} = \frac{a_{ik}}{a_{kk}}$ (assuming $a_{kk} \neq 0$)

    for $j = k, n$
        $a_{ij} = a_{ij} - \ell_{ik} a_{kj}$
```

O(n^3)

Parallel Gaussian Elimination

When using MPI to do Gaussian Elimination in parallel, there are some additional steps:

- All processors simultaneously compute their own scaling factor and update their rows based on which row is currently the reference row.
- As each row in the matrix is updated, it must be broadcasted to all other processors so that they can correctly update subsequent rows

Using Gaussian Elimination to Calculate Determinant

Reminder: The resulting matrix after performing Gaussian Elimination

$$\begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & u_{0,n-1} \\ 0 & u_{1,1} & \cdots & u_{1,n-1} \\ & & \cdots & \\ 0 & 0 & \cdots & u_{n-1,n-1} \end{bmatrix}$$

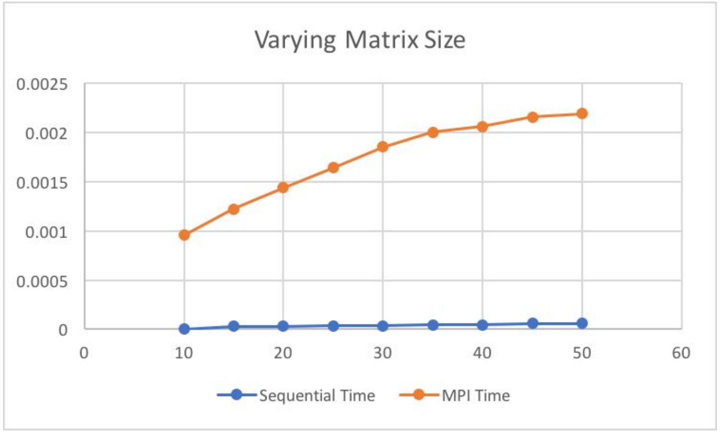
After we perform Gaussian Elimination, it is a straightforward calculation to get the determinant.

Determinant = $u_{0,0} \times u_{1,1} \times \ldots \times u_{n-1,,n-1}$

Implementation Details

- We must use doubles instead of ints so that our scaling will not give us a factor of 0.
- Simply implementing the straightforward method of Gaussian Elimination can result in very large numbers
- Multiplying these large numbers together to calculate the determinant causes a couple of different issues
    - Large run-times due to the size of the floating point numbers
        - Long calculation time
        - Long communication time between processes
    - Data overflow issues as matrix size increases

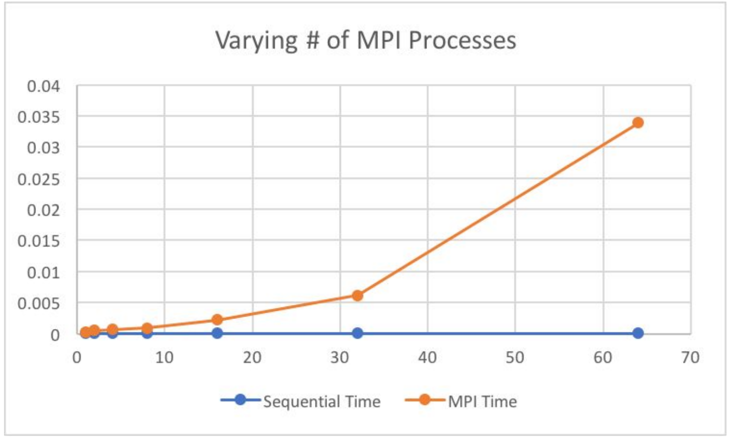
Varying Matrix Size



As expected, the time increases as the size of the matrix increases.

However, what was not expected is that the parallel version performs worse than the sequential version and that the gap widens as the size increases.

Varying # of MPI Processes



We can see that the MPI performance degrades as the number of processes decreases

How can we make it Better?

- There is a concept called pivoting which helps to keep the numbers in the matrix at a reasonable size.
- Pivoting will vastly improve the performance of Gaussian Elimination
- The version of Gaussian Elimination with pivoting is the version which was applied to solving linear systems
- Stay tuned for more information on pivoting!

Solving Linear Systems

- Two phases:
    - Gaussian elimination
    - Back substitution
- Why pivoting is necessary
- Sequential algorithm
- Expose parallelism
- Partition the rows of the matrix
- Runtime results

Linear Equations

A *linear equation* with n unknowns:

$$\boldsymbol{a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1} = b}$$

Where $a_0, a_1, \dots, a_{n-1}$ and b are constants.

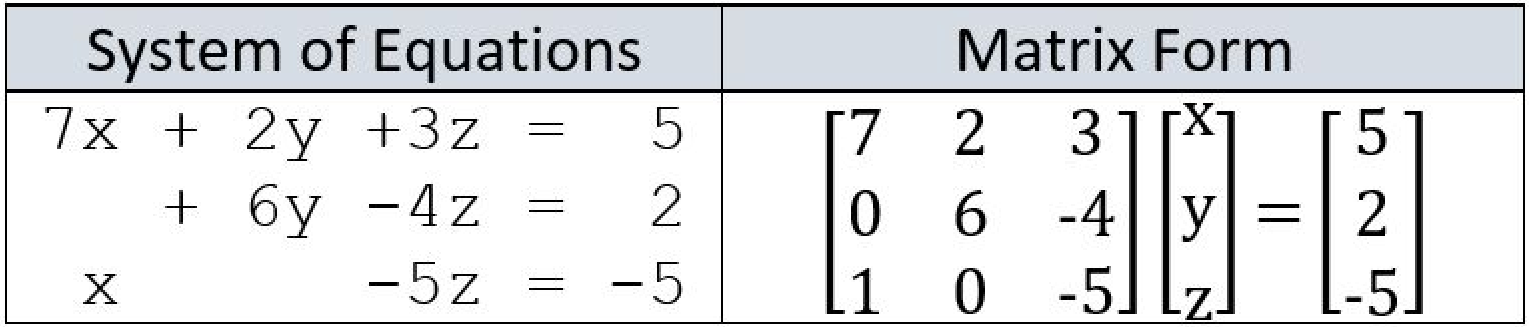
System of Linear Equations

A set of n linear equations can be written as:

$$\begin{aligned}
a_{0,0}x_0 &+ a_{0,1}x_1 &+ \cdots + a_{0,n-1}x_{n-1} &= b \\
a_{1,0}x_0 &+ a_{1,1}x_1 &+ \cdots + a_{1,n-1}x_{n-1} &= b \\
&\dots \\
a_{n-1,0}x_0 &+ a_{n-1,1}x_1 &+ \cdots + a_{n-1,n-1}x_{n-1} &= b
\end{aligned}$$

Matrix Form

A system can be converted into form Ax = b., where A is a coefficient matrix.



System of Equations	Matrix Form
$\begin{array}{rrrrr} 7x & +\ 2y & +3z & = & 5 \\ & +\ 6y & -4z & = & 2 \\ x & & -5z & = & -5 \end{array}$	$\begin{bmatrix} 7 & 2 & 3 \\ 0 & 6 & -4 \\ 1 & 0 & -5 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ -5 \end{bmatrix}$

Solving Linear Systems of Equations

Two steps:

1. Gaussian Elimination
2. Back Substitution

Gaussian Elimination

Goal: Upper triangular form

$$\begin{bmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-1} \\ & & \dots & \\ 0 & 0 & \dots & u_{n-1,n-1} \end{bmatrix}$$

What is a pivot?

A **pivot**, is an element of a matrix which is selected by an algorithm to do certain calculations.

Gaussian Elimination requires the pivot element to NOT be zero.

$$\left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 0 & -1 & -11 \\ 0 & 2 & -1 & -3 \end{array}\right]$$

On second iteration, swap 2nd, and 3rd row.

No Pivot, No Numerical Stability

Example,

$$\left[\begin{array}{cc|c} 0.00300 & 59.14 & 59.17 \\ 5.291 & -6.130 & 46.78 \end{array}\right]$$

Exact solution: $x_1 = 10.00$ $x_2 = 1.000$

4-digit arithmetic leads to: $x_1 \approx 9873.3$ $x_2 \approx 4$

How to Partial Pivot?

Simple:

Select largest element from the current pivot column and use it as the pivot element.

Pivoting Example

$$\left[\begin{array}{cccc|c} 0.02 & 0.01 & 0 & 0 & 0.02 \\ \textcircled{1} & 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} \textcircled{1} & 2 & 1 & 0 & 1 \\ 0.02 & 0.01 & 0 & 0 & 0.02 \\ 0 & 1 & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

Pivoting Example

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & -0.03 & -0.02 & 0 & 0 \\ 0 & \boxed{1} & 2 & 1 & 4 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccc|c} 1 & 2 & 1 & 0 & 1 \\ 0 & \boxed{1} & 2 & 1 & 4 \\ 0 & -0.03 & -0.02 & 0 & 0 \\ 0 & 0 & 100 & 200 & 800 \end{array}\right]$$

Parallel Gaussian Elimination

```
struct {
        double  maxValue;
        int     taskID;
} localPivot, pivot;
```

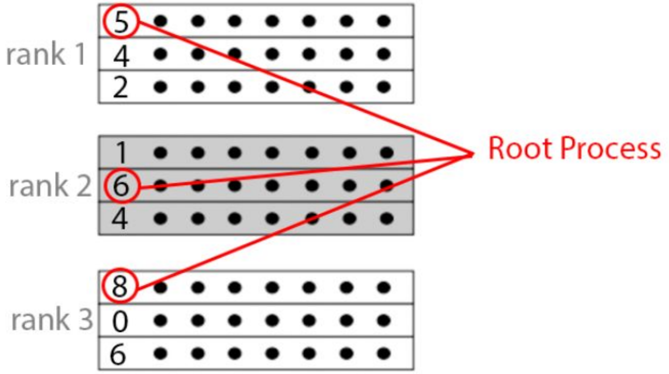
Every iteration *i* of Gaussian elimination consists of:

- **Choosing the pivot row** - The root process collects the max value from all of the other processes to determine what processor has the pivot.

```
MPI_Allreduce(&localPivot, &pivot, 1, MPI_DOUBLE_INT, MPI_MAXLOC, MPI_COMM_WORLD);
```

- **Send** the pivot row to all subtasks.

- **Subtract** the pivot row from the rows.



Gaussian Elimination

Achieved!

$$\begin{bmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-1} \\ & & \dots & \\ 0 & 0 & \dots & u_{n-1,n-1} \end{bmatrix}$$

Proceed with Back-Substitution

Back Substitution

$$\begin{aligned} x_0+3x_1+2x_2 &= 1 \\ x_1+x_2 &= 16 \\ 3x_2 &= 9 \end{aligned}$$

From the third equation, it is clear that x_2 is 3.

$$\begin{aligned} x_0+3x_1+2x_2 &= 1 \\ x_1 \qquad &= 13 \\ x_2 &= 3 \end{aligned}$$

The value of x_0 can be determined from the last iteration of **back substitution.**

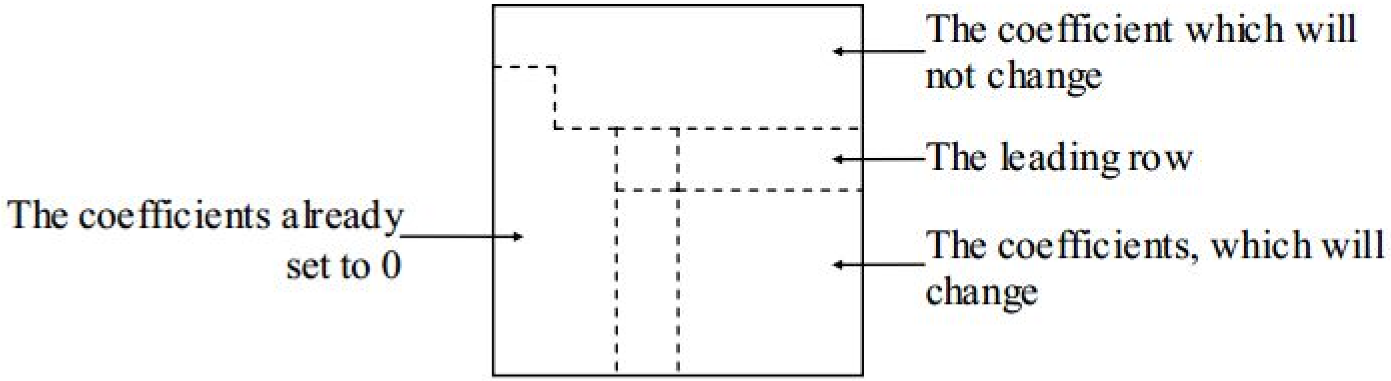
Parallel Back Substitution

Use a “known” value in parallel. For example, x_2 may be applied to the first and second equations in parallel.

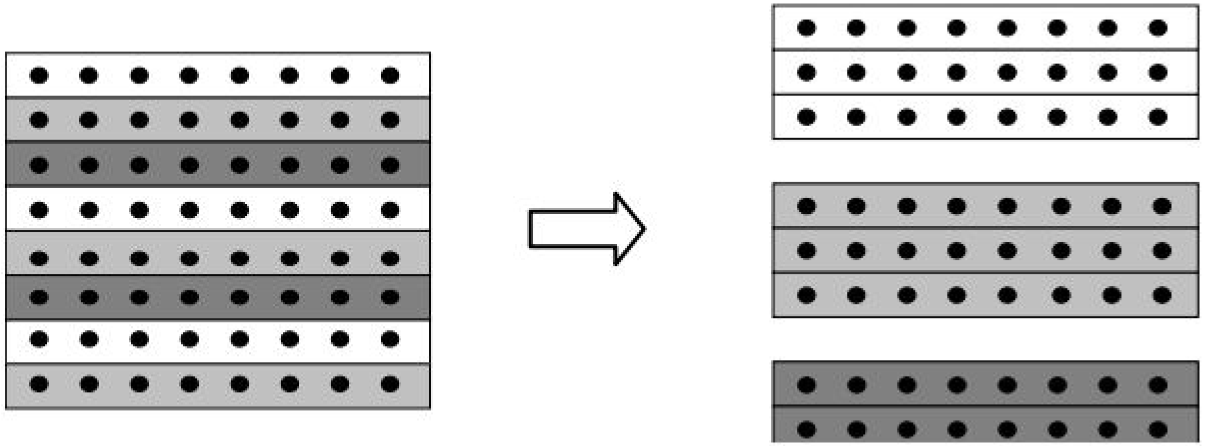
$$\begin{array}{rrr} x_0+3x_1 & & =-5 \\ x_1 & & =13 \\ & x_2 & =\ 3 \end{array}$$

Information Dependency

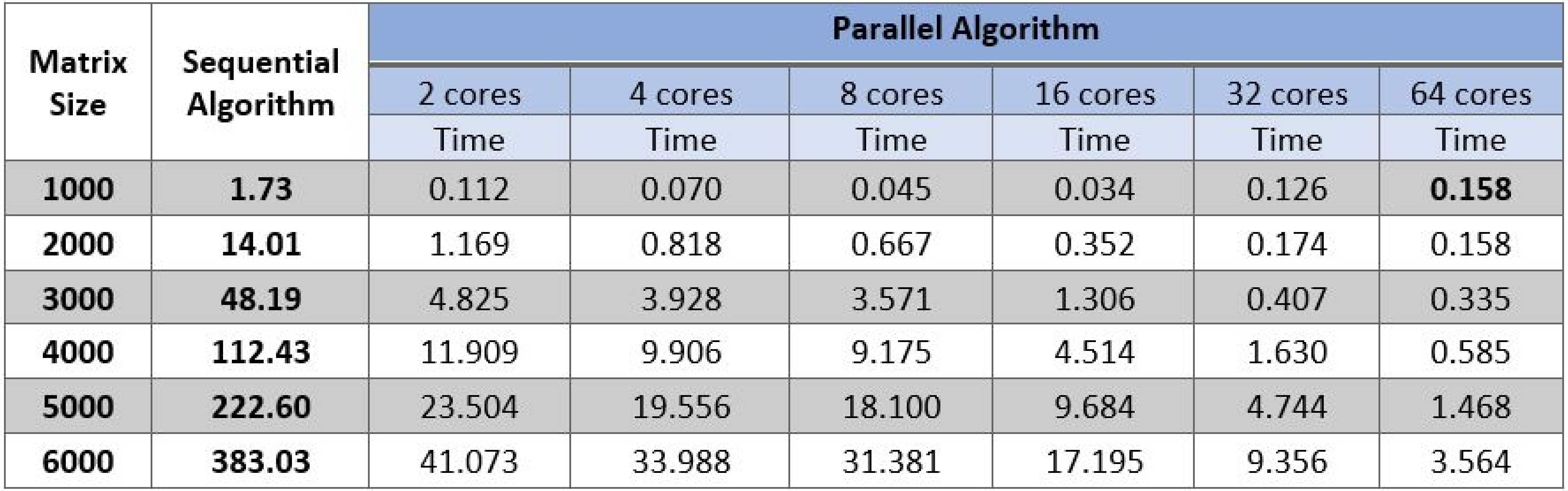
Processes remain idle during both elimination and back-substitution.


The coefficient which will not change
The leading row
The coefficients already set to 0
The coefficients, which will change

Cyclical Row-Striping

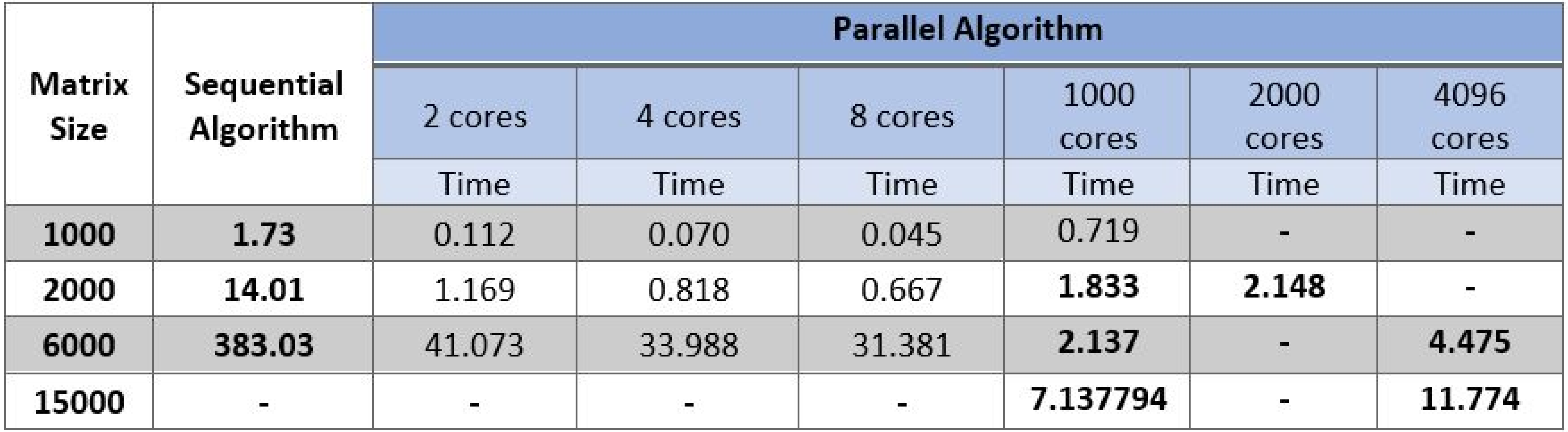


Results



Matrix Size	Sequential Algorithm	Parallel Algorithm					
		2 cores	4 cores	8 cores	16 cores	32 cores	64 cores
		Time	Time	Time	Time	Time	Time
1000	**1.73**	0.112	0.070	0.045	0.034	0.126	**0.158**
2000	**14.01**	1.169	0.818	0.667	0.352	0.174	0.158
3000	**48.19**	4.825	3.928	3.571	1.306	0.407	0.335
4000	**112.43**	11.909	9.906	9.175	4.514	1.630	0.585
5000	**222.60**	23.504	19.556	18.100	9.684	4.744	1.468
6000	**383.03**	41.073	33.988	31.381	17.195	9.356	3.564

Communication Overhead



Matrix Size	**Sequential Algorithm**	**Parallel Algorithm**					
		2 cores	4 cores	8 cores	1000 cores	2000 cores	4096 cores
		Time	Time	Time	Time	Time	Time
1000	**1.73**	0.112	0.070	0.045	0.719	-	-
2000	**14.01**	1.169	0.818	0.667	**1.833**	**2.148**	-
6000	**383.03**	41.073	33.988	31.381	**2.137**	-	**4.475**
15000	-	-	-	-	**7.137794**	-	**11.774**

Future Work

Explore different data distribution schemes

Analyze message complexity to find the optimum number of cores for a given problem size.

Test against direct solvers from ScaLAPACK (Scalable Linear Algebra PACKage).