Comparison of Sequential Approach and Parallel Approach

SOLVING SET OF LINEAR EQUATIONS

YELLANKI SAIRAM AARYAN

ABSTRACT

The solution of systems of linear equations is one of the mathematical concepts that can be applied to numerous real-world problems.
The size and complexity of some systems may also appear to be beyond your ability to solve them alone. Therefore, we use computational tools.

To solve a set of linear equations, we present a technique in this report that is launched on the CPU for a sequential approach and on CUDA-C for a parallel approach. We employ the Doolittle method or LU Factorization method.

In conclusion, the report is summarized by comparing the Sequential and Parallel performances to identify which provides the best solutions in various scenarios.

INTRODUCTION

- **Linear equation: -** If a variable's maximum power is always 1, an equation is said to be linear. It is also called a one-degree equation. The typical form of a linear equation with one variable is Ax + B = 0. Here, the variable is x, the coefficient is A, and the constant is B. The typical form of a linear equation with two variables is Ax + By = C. Here, the variables are x and y, the coefficients are A and B, and C acts as a constant.

- **Linear Equation Solution**: - The intersection points are the locations where two lines converge to create a solution to a linear equation.

- To put it another way, the solution set of a system of linear equations is the set of all possible values for the variables that fulfil the given linear equation.

System of linear equations

$$ax + by = c$$

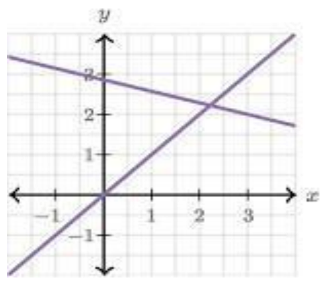
$$a_1x + b_1y = c_1$$

There are 3 possible types of solutions:

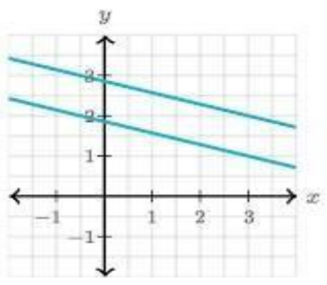
- **Unique Solutions**: - There is only ever one solution to the linear equation with one variable. A linear equation with a unique solution is the only one for which LHS and RHS are equivalent upon substitution. Two simultaneous linear equations in two variables should have a pair of ordered equations as their solution (x, y). In this case, the ordered pair will satisfy the set of equations.

- **No Solution**: - If the graphs of the linear equations are parallel, the system of linear equations cannot be solved. There is no point in this situation when no lines cross one another.

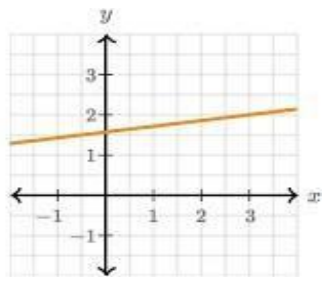
- **Infinite many solutions**: - Numerous solutions exist for a linear equation with two variables. There are an endless number of solutions to the issue set of linear equations when the LHS of each equation becomes the RHS. The system of linear equations with an infinite number of solutions can be represented as a graph of intersecting straight lines.



Unique Solutions



No Solutions



Infinite many solutions

- ✓ The following straightforward matrix system will ultimately represent a huge number of real-world applications that may be transformed into mathematical models.
- ✓ $Ax = b$
- ✓ where A represents a known $n \times n$ matrix and b a known $n \times 1$ vector.
    - Vector x represents the n unknowns

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

LU Factorization

- The algorithm we are using to solve a linear equation is LU Factorization
- LU Factorization: - LU factorization which is short for lower and upper triangular factorization, factors a matrix as the product of a lower triangular matrix and anupper triangular matrix.
- LU decomposition is typically used by computers to solve square systems of linear equations. It is also a crucial step when inverting a matrix or calculating the determinant of a matrix.
- Consider a square matrix A. An LU factorization is the factorization of A into two elements, a lower triangular matrix L and an upper triangular matrix U, with the appropriate row and/or column orderings or permutations.

  $$A = LU$$
- All of the elements above the diagonal in the lower triangular matrix are zero, whereas all of the elements below the diagonal are zero in the upper triangular matrix

For example, for a 3 × 3 matrix *A*, its LU decomposition looks like this:

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} L11 & 0 & 0 \\ L21 & L22 & 0 \\ L31 & L32 & L33 \end{bmatrix} \begin{bmatrix} U11 & U12 & U13 \\ 0 & U22 & U23 \\ 0 & 0 & U33 \end{bmatrix}$$

- To Decompose A into L and U we are using Do-little's Method

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} L11 & 0 & 0 \\ L21 & L22 & 0 \\ L31 & L32 & L33 \end{bmatrix} \begin{bmatrix} U11 & U12 & U13 \\ 0 & U22 & U23 \\ 0 & 0 & U33 \end{bmatrix}$$

Set Lii = 1

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L21 & 1 & 0 \\ L31 & L32 & 1 \end{bmatrix} \begin{bmatrix} U11 & U12 & U13 \\ 0 & U22 & U23 \\ 0 & 0 & U33 \end{bmatrix}$$

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} U11 & U12 & U13 \\ L21U11 & L21U12+U22 & L21U13+U23 \\ L31U11 & L31U12+L32U22 & L31U13+L32U23+U33 \end{bmatrix}$$

In order to determine the unknown values, such as U11, U12, and so on, we will compare LHS with RHS and obtain the Lower triangular matrix and Upper triangular matrix.

Now we have successfully decomposed A into LU

But we need X to get X we solve LY=B

How we got LY=B is our original matrix was AX =B here we decomposed A into LU so LU(X) =B we are considering U(X) = Y so we got LY=B which we'll be using now

By applying a forward solution, we get

YNow we solve UX = Y

We already have U and Y so by back substitution we get X which is our solution matrix.

SEQUENTIALAPPROACH

We already discussed the whole algorithm so right now we need to codeit we are using C to code the sequential part.

To solve AX=B we'll decompose A into L and U and then consequentlythe unknown elements of vector X.

We'll divide the code into 7 parts so it'll be easier for us

1. The first part of the code is a function to allow users to input all theelements of matrix A
2. The second part of the code is a function to input all the elements of vectorB
3. Third part we'll write a function to decompose A into L and U
4. Forth part is to Print all elements of L
5. The fifth part is to Print all elements of U
6. The sixth part is we solve LY=B to get X
7. Seventh part of the code is to print all elements of X that is our solution matrix.

CODE: -

```
#include <stdio.h>
#include <stdlib.h>
#define N 100 /*Constant N refers to the number of
              equationsand unknowns in the system. It is
              manually changeable and is used to define the
              size of 1D and 2D arrays in the program*/

void input_A(double a[N][N]) {int
i, j;
for(i=0; i<N; i++)

{for (j=0; j<N;j++)
{printf("Input element a%d%d of matrix A: \n",i,j);
scanf("%lf", &a[i][j]);}}
}

void input_b(double b[N])
{
```

```
int i; printf("\n");
for(i=0; i<N; i++)
{printf("\nInput element b%d of vector b: ",i);
scanf("%lf", &b[i]);}

}
```

```
void factor_A(double a[N][N], double l[N][N], doubleu[N][N])
{
if (a[0][0]==0)
printf("Impossible factorization\n");

else
{int i, j, k; double sum;
u[0][0]=a[0][0]; l[0][0]=1;
for(i=1;i<N;i++)
{u[0][i]=a[0][i]; l[i][i]=1;
l[i][0]=a[i][0]/u[0][0];
}
for (i=1;i<N-1;i++)
{sum=0;
for (k=0;k<i;k++)
{sum+=l[i][k]*u[k][i];}
u[i][i]=a[i][i]-sum;if
(u[i][i]==0)
{printf("Impossible factorization\n"); break;}else
{for (j=i+1;j<N;j++)
{sum=0;
for (k=0;k<i;k++)
{sum+=l[i][k]*u[k][j];}
u[i][j]=a[i][j]-sum;
sum=0;
for (k=0;k<i;k++)
{sum+=l[j][k]*u[k][i];}
l[j][i]=(a[j][i]-sum)/u[i][i];}
}
}
sum=0;
for (k=0;k<N-1;k++)
{sum+=l[N-1][k]*u[k][N-1];}
u[N-1][N-1]=a[N-1][N-1]-sum;
```

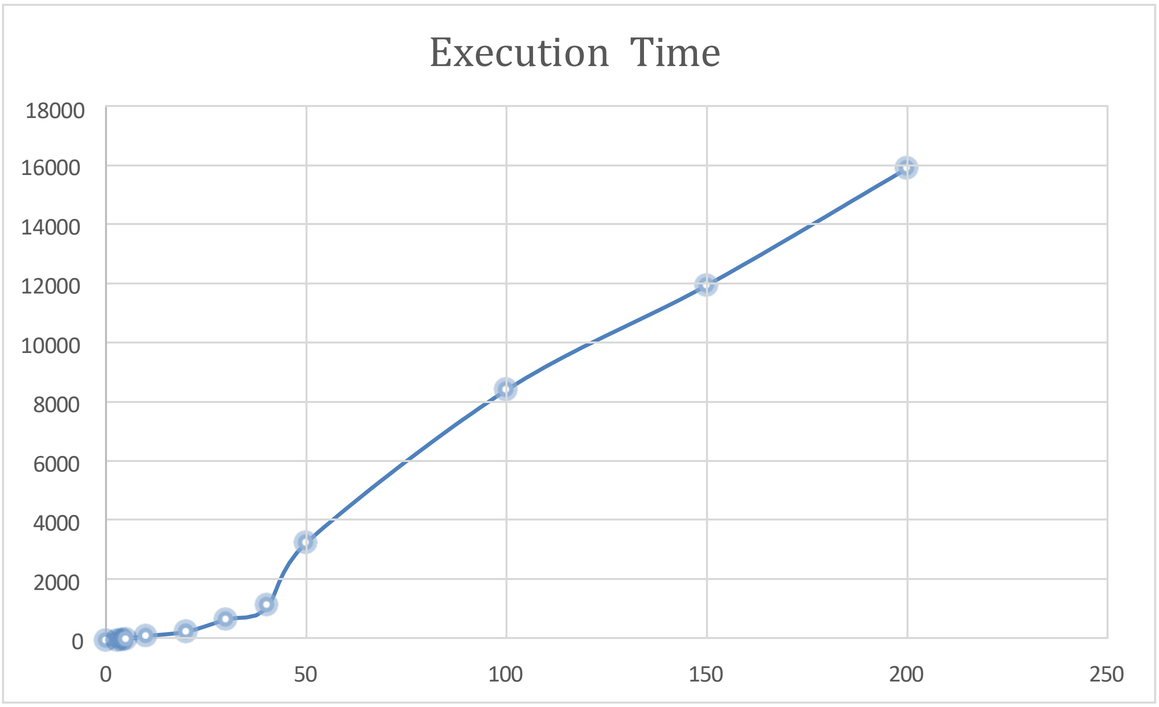
```
for(i=0;i<N-1;i++)
{for(j=i+1;j<N;j++)
{l[i][j]=0;
u[j][i]=0;}}

} }

void solve(double l[N][N], double u[N][N], doubleb[N],
double x[N])
{int i, j; double sum=0;
double y[N];

y[0]=b[0];


for(i=1;i<N;i++)
{for(j=0;j<i;j++)
sum+=l[i][j]*y[j];
y[i]=b[i]-sum;
sum=0;}
x[N-1]=y[N-1]/u[N-1][N-1];
for(i=N-2;i>=0;i--)
{for(j=i+1;j<N;j++)
sum+=u[i][j]*x[j];
x[i]=(y[i]-sum)/u[i][i];
sum=0;

} }

void print_L(double t[N][N])
{
int i, j; printf("\nMatrix L:\n");for
(i=0;i<N;i++)
for (j=0;j<N;j++)
printf("\nThe value of element L%d%d of matrix L is:
%lf",i,j,t[i][j] );
}

void print_U(double t[N][N])
{
int i, j; printf("\nMatrix U:\n");for
(i=0;i<N;i++)
for (j=0;j<N;j++)
printf("\nThe value of element L%d%d of matrix U is:
```

```
%lf",i,j,t[i][j] );
}

void print_x(double x[N])
{
int i;
printf("\nVector x:\n");
for(i=0;i<N;i++)
printf("\nThe value of element x%d of vector x is:
%lf",i,x[i]);

}

int main()
{
double A[N][N], L[N][N], U[N][N], b[N], x[N];
input_A(A); /*Initializing matrix A by the user*/
factor_A(A,L,U); /*Factorizing matrix A*/ print_L(L);
print_U(U); /*Printing matrices L and U*/
input_b(b);                /*Initializing vector b by the user*/
solve(L,U,b,x); /*Finding vector x*/
print_x(x); /*Printing vector x*/
/*free memory*/
free(A);
free(L);
free(U);
free(b);
free(x);

return 0; }
```

<u>Sequential Execution time</u>:-

We observed the following results for each value of n

Number of Unknowns	Execution time (ms)
3	5
4	18
5	43
10	155
20	305
30	715
40	1079
50	3310
100	8477
150	12016
200	15964



PARALLELAPPROACH

In an attempt to parallelize, we are using CUDA for coding.

However, we are unable to parallelize the entire piece of code because a portion of Doolittle's resolution contains necessary sequential operations that cannot be avoided.

We need not change the entire code as we’ll be parallelizing only a part of it

Just as in the sequential part we’ll divide the code into 7 parts to ease the

process

1. The first part of the code is a function to allow users to input all theelements of matrix A (No Changes)

2. The second part of the code is a function to input all the elements of vectorB(No Changes)

3. Third part we’ll write a function to decompose A into L and U (Some changes are made to the code so it can be launched on the GPU)

4. Forth part is to Print all elements of L (Some changes are made to the code so it can be launched on the GPU)

5. The Fifth part is to Print all elements of U(Some changes are made to thecode so it can be launched on the GPU)

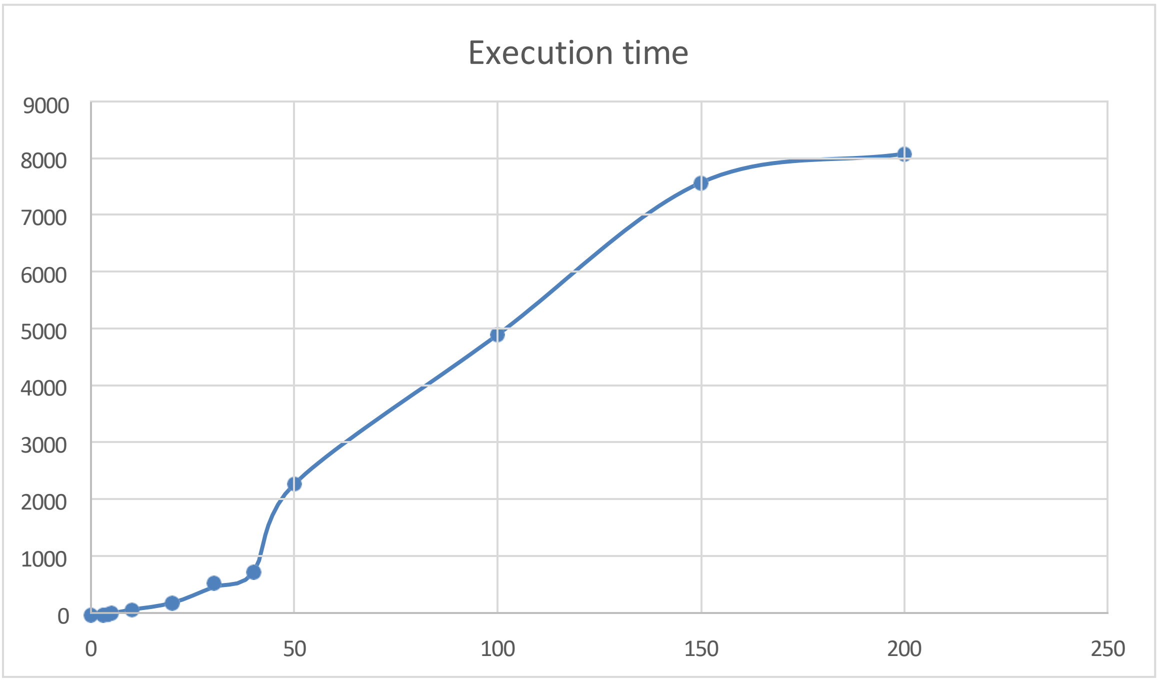
6. The sixth part is we solve LY=B to get X (This part cannot be renderedParallel)

7. The seventh part of the code is to print all elements of X which is our solution matrix (Some changes are made to the code so it can be launched on theGPU)

Parallel Execution Time:-

We have observed the following results for each value of n.

Number of Unknowns	Executed Time (ms)
3	7
4	15
5	32
10	94
20	214
30	488
40	755
50	2307
100	4933
150	7611
200	8117



COMPARATIVEANALYSIS

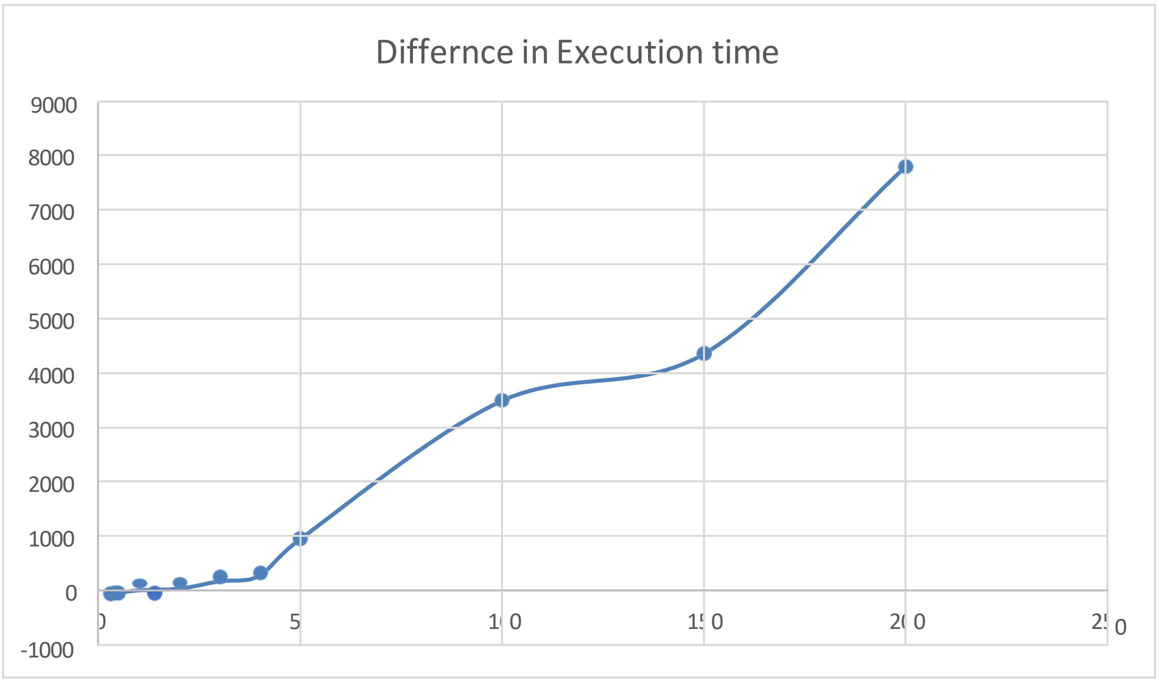
We are comparing the execution times of Series and Parallel and we have tableted the data so it will be easy to compare.

Consider Series Execution time as t(S)
Parallel Execution time is t(P)

Number of Unknowns	Difference t(S)-t(P)	Ratio t(S)/t(P)
3	-2	0.71
4	3	1.20
5	11	1.34
10	61	1.65
20	91	1.43
30	227	1.47
40	324	1.43
50	1003	1.43
100	3544	1.72
150	4405	1.58
200	7847	1.97

Here,
If t(S)-t(P) is positive then the Efficiency of Parallel is better than Sequential and ifit's Negative then the Efficiency of Sequential is better than Parallel.

If t(S)/t(P) is greater than 1 then the Efficiency of Parallel is better than sequentialand if it's less than 1 then the Efficiency of Sequential is better than Parallel.


Differnce in Execution time
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
-1000
0
5
10
15
20
25

CONCLUSION

From the promising results that we got, we were able to interpret that the difference in execution time is negligible for a few equations. However, for large numbers of equations, the difference between serial and parallel execution times appears to be very striking, and the serial elapsed time can occasionally be close to twice as long as the parallel algorithm's elapsed time.

As we can see from the comparisons above, there are very slight differences between sequential and parallel execution times for small data sets, and sometimes sequential execution is faster than parallel. However, as dataset sizes grow, parallel execution times improve and eventually become twice as quick as sequential.

REFERENCES

- Gallivan K. A., Plemmons R. J., and Sameh A. H.,"Parallel algorithms for dense linear algebra Computations" SIAM Rev., vol. 32, March 1990.

- Wilkinson, B., Allen, M., "Parallel Programming", Pearson Education (Singapore), 2002.

- B. Kirk David, W. Hwu Wen-mei, "Programming Massively Parallel Processors", 2nd ed. Elsevier, 2013.

- Cyril Zeller, NVIDIA Corporation, "CUDA C/C++ Basics", Supercomputing 2011 Tutorial https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics (Accessed 2019-09-05.)

- Marzia Rivi, UCL, Department of Physics & Astronomy, "Parallel Computing: a brief discussion", Research Programming Social, Nov 10, 2015 https://www.ucl.ac.uk/research-it-services/sites/research-it-services (Accessed 2019-09-09.)

- Lily Asis, "The difference between a CPU and GPU", contributing writer, may 31, 2018 http://www.upgrademag.com/web/2018/05/31/the-difference-between-a-cpu-and-gpu (Accessed 2019-09-10.)

- Martin Heller, Contributing Editor, InfoWorld, "What is CUDA? Parallel programming for GPU's", August 30, 2018 https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html (Accessed 2019-09-10.)

- Führer Claus and Sopasakis Alexandros, "Solving Systems of Equations" in "FMN050" chapter 2, pp. 20-26, March 2014 http://www.maths.lth.se/na/courses/FMN050/media/material/part4 (Accessed 2019-09-11.)