# CSE 6220 - High Performance Computing
# Programming Assignment 3

Azlan Shah Bin Abdul Jalil
GTID : 903108282

April 16, 2019

## Problem Description

$Jacobi's method$ is an iterative, numerical method for solving a system of linear equations. Formally, given a full rank $n \times n$ matrix $A \in \Re^{n \times n}$ and a vector $b \in \Re^n$, Jacobi's method iteratively approximates $x \in \Re^n$ for:

$$\begin{aligned} Ax &= b \\ D_{x_{n+1}} &= -(L+U)_{x_n} + b \end{aligned}$$

One of the advantage of $Jacobi's Method$ is that the algorithm can be done in parallel computing. To parallel this process, we distribute matrix $A$ and $L+U$ among the $q \times q$ processor mesh which is a cartessian topology. We can then compute the $(L+U)x$ by using the matrix-vector multiplication. We will then have the vector $x$, $b$ and the diagonal array $D$ of $A$ to be distributed in the first column of the processor cartessian grid. From this, we compute the $x_{n+1}$ using this formula:

$$x_{n+1}^i = \frac{1}{a_{ii}}(b_i - (Rx)_i)$$

The termination criteria is obtain by computin the residual $||Ax_n - b||_2$ and also counting the number of iterations. We have a max iteration as our stopping criteria as well the $l2\_norm$ which corresponds to the residual. When either of these two conditions meet, the calculation will terminate.

## Algorithm - Sequential

The sequential Jacobi will follow the standard known algorithm to get the new approximation of $x_{n+1}$ which is based on the present approximation for every iteration. The $x_0$ will be initialized to 0 and new approximation will be calculated. Iterations will continue untile either of the two stopping criteria mention above is met.

Input : Initialize $x_0 = 0$
$while(||Ax_n - b||_2 > l2\_norm \quad \&\& \quad iter < max\_iter) : do$
    Compute $x_{i+1}$ based on $x_i$
    $iter++$
$end$

## Algorithm - Parallel

There are many ways to do parallel algorithm, however we always want our algorithm to be as efficient as possible. The efficiency has to scale to the number of processor used. To simplify our algorithm, we only allow a perfect square number of processors,$p$ so that we can form a square $\sqrt{p} \times \sqrt{p}$ cartessian mesh. For this algorithm, the matrix will be distributed to each processor in a column-wise fashion. Another ways

would either be distributing by row-wise or by block-wise.
The processors will be embeded into a mesh topology where they have North, South, East and West neighbour. The processor $(0, 0)$ is treated as the root and it will store the initial matrix $A$ and the right hand side of vector $b$. The parallel algorithm will follow these steps:

1. Processors are embedded into a cartessian topology. Using the $MPI\_Cart\_sub$ function to create a new local communicator. Then using $MPI\_Scatterv$ function, the matrix is distributed based on the rows to the processors. The process is repeated to create a submatrix that will be distributed among each processor on the same row.

2. The root processor $(0, 0)$ will distribute the vector to the processors in the first column of the mesh grid.

3. The vector $x$ will be transposed such that processor with index $(i, j)$ will have elements that were in processor $(j, 0)$.

4. The local matrix is multiplied with the local vector.

5. Using parallel reduction, we can sum the resulting vectors along the rows and the result be stored in each corresponding processors in the first column.

6. Compute the new value of $x_{i+1}$ and the residual is also computed.

7. Steps $3 - 6$ will be repeated until either of the stopping criterias are met.

# Runtime

We will first test our parallel algorithm with the sequential algorithm using the same difficulty, $d = 0.5$,. We want to see how well and efficient our parallel algorithm compared to the sequential. The number of elements $N$ will be varied for to see how the runtime increases as we increase $N$. We limit the number of elements up to $N = 12000$ as the sequential solver will take too long or fail due to segmentation fault if we go any higher. For the parallel algorithm, we will vary the number of processor, $p$ used to see the speed and the efficiency. Like mention earlier, the number of processor used has to be a perfect square and the upper limit of number of processor that can be used is limited by the number of available cores in the cluster. For students, we are only allowed to use up to $p = 81$ in the $coc - ice - multi$ cluster.

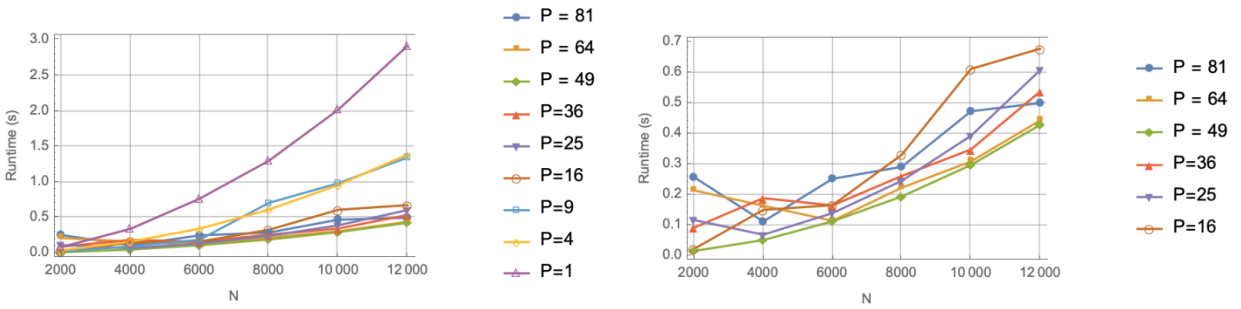


Figure 1: Runtime for various number of processors ($d = 0.5$)

Figure 1 above shows the runtime for various number of processors including the sequential algorithm runtime. The figure on the left shows the higher runtime of the sequential solver. Its runtime increases very significantly for increasing $N$ compared to the parallel solver. The figure on the right shows a close-up plot for higher number of processors. In parallel programming, there's always a tradeoff when we use more processors as it requires more communication. Generally, communication will cost order of magnitudes higher than data computation. Thus for lower $N$, it is not efficient to use more processors as the communication cost will outweight the computation cost. As can be seen on the right figure, for $p = 81$ it has the highest runtime for lowest $N$. However, as $N$ increases and consequently computation cost, the $p = 81$ curve will begin to have better runtime than the lower $p$. It can be seen its gradien at the $N = 12000$ is significantly

less steep than the rest. This shows that for higher $N$, $p = 81$ will have the best runtime.

The higher communication cost is due to the fact that our Parallel Jacobi method has to distribute the matrix $A$ to all processors in the grid. Also in each iteration, the vector $x_n$ has to be *Scatter* to all the processors in the grid from the first column. $MPI\_Reduce$ is also used at every iteration to sum up the vectorm component in each row of processors. Finally, the termination criteria requires the summation of residual in the first column of processors which require $MPI\_Reduce$ again. Thus, all these Communication Primitive Functions contribute to the communication cost.

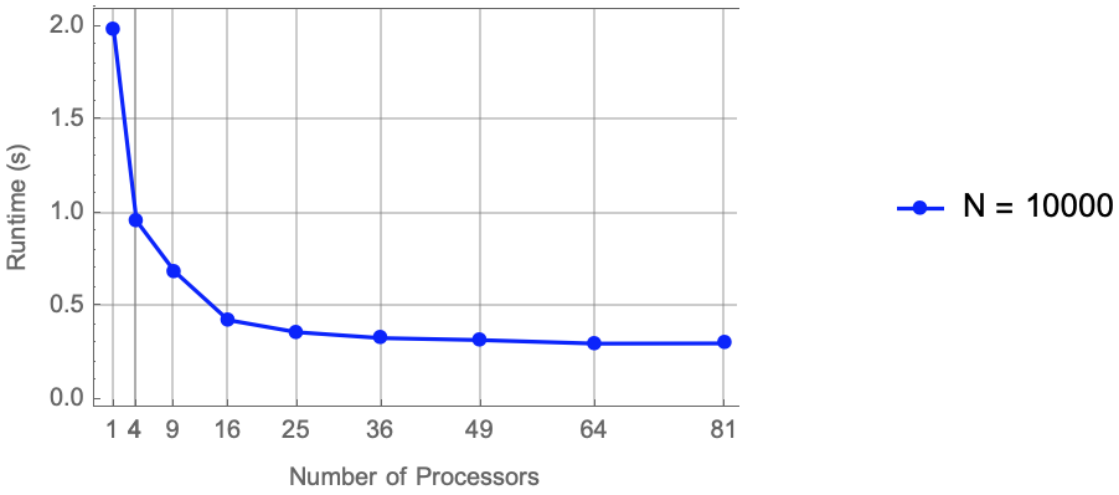


Figure 2: Runtime for various number of processors (Constant $N = 10000$)

Figure 2 above shows the runtime of various number of processor but for constant $N = 10000$. From this plot, it can easily be seen that the runtime decreases exponentially as we increase the $p$. However, we will also get diminishing return as we use higher $p$ for constant $N$. Thus it is not always better to use the highest number of processor when lower number can similarly finish the computation with the same runtime but at significantly lower cost. Thus we need to be aware of our $N$ values and choose our $p$ accordingly. For the case above, $N = 10000$, choosing $p = 16$ or $p = 25$ results in similar runtime as with $p = 81$ but at a quarter of the cost. Also note that as mention before, we design our algorithm to work with $p$ being perfect square only.

## Efficiency

In parallel algorithm, the important criterias we want to improve is the speed up and efficiency. In an ideal case, we would like a speed up in our algorithm everytime we increase the number of processors, $p$ used. The efficiency should also be conserved if we use less processors for our algorithm, meaning the runtime should be proportional to the $p$.
To clearly show speed up and efficiency we have to use even higher $N$. Figure 3 shows the runtime of $N$ up to 15000 for the 3 highest parallel configuration we could have which are $p = 49, 64, 81$. From this plot, we can see that as the computation cost increases, our parallel algorithm is efficient in a way that it's runtime will be faster with more processors. This shows that the computation cost has gone much higher than communication cost that it is possible to use more processors to achieve speedup.

Finally, we can also choose to have better accuracy with our numerical results by increasing the difficulty, $d$. The difficulty corresponds to the number of iterations the solver will do until it reaches it maximum stopping criteria. The more iterations we do, the more accurate the vector $x_n$ will be, however at the cost of higher runtime as more communication will be done. As mention above, each iteration will call the Communication Primitive Functions which cost signification communication time. This result can be observed in Figure 4 below. It can clearly be seen the runtime scales as we increase the difficulty. The plot

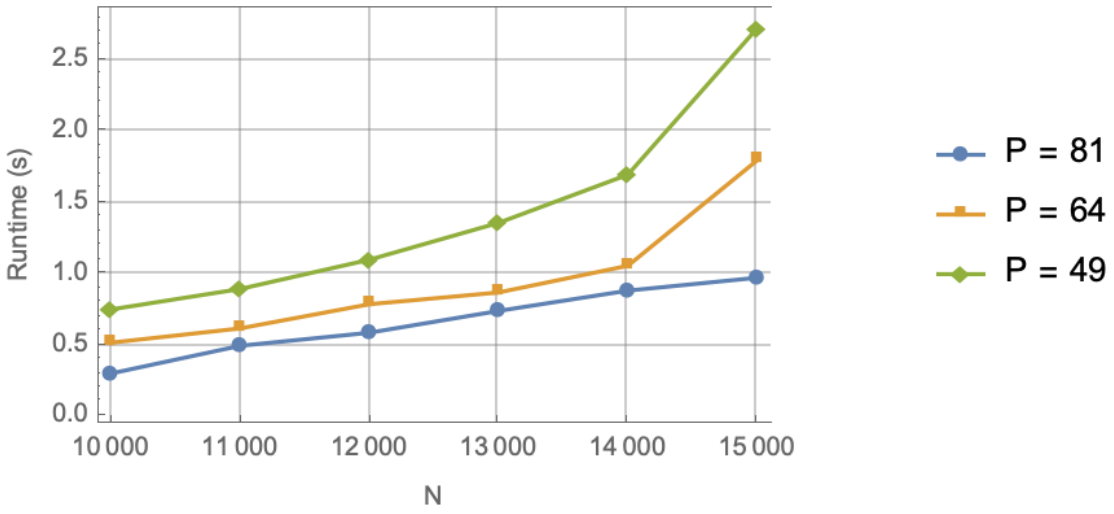


Figure 3: Runtime for various number of processors ($d = 0.5$

has the parameter of $N = 10000$ and $p = 25$. A lower $p$ was chosen so that the communication cost will not be too great and we can observe the relationship between the runtime and the accuracy.The default value is $d = 0.5$ where it is the middle ground between accuracy and speed which was the value used for all the previous results.

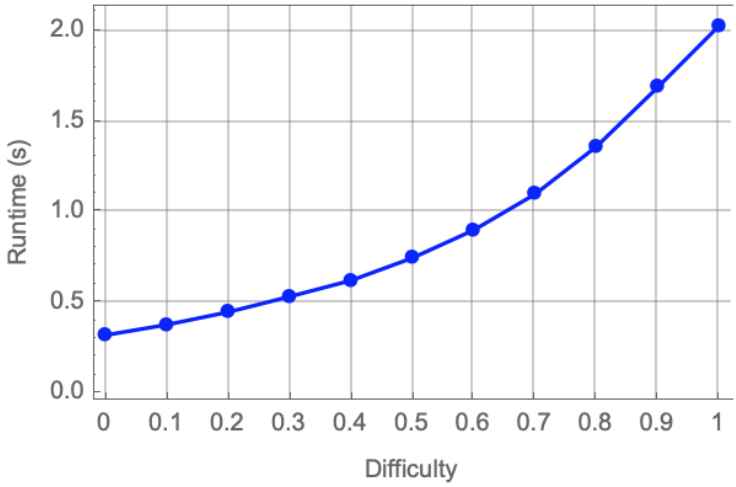


Figure 4: Runtime for various difficulties ($N = 10000, p = 25$)