# CENG 478

Assignment 2 Report
Deadline: April 9, 23:59

Full Name: Yasin Fatih ALPUL
Id Number: 2098739

## Complexity

Assume there are $n$ teams in a tournament. The number of matches in each round is demonstrated as follows:

$$\begin{aligned}
\text{Round } 1 &\rightarrow \frac{n}{2} \\
\text{Round } 2 &\rightarrow \frac{n}{4} \\
\text{Round } 3 &\rightarrow \frac{n}{8} \\
&\vdots \\
\text{Round } k-2 &\rightarrow 4 \\
\text{Round } k-1 &\rightarrow 2 \\
\text{Round } k &\rightarrow 1
\end{aligned}$$

Using mathematical calculations, if we sum the number of matches in each of those rounds, we get $n-1$ matches in total.

In the assignment, we are given a number $n$ and that makes $2^n$ teams in total. My implementation divides the teams to $p$ processes and conducts small tournaments in each of those processes. Then, the winners of those tournaments match with each other and the final winner is determined. Thus, the overall complexity becomes:

$$O\left(\frac{2^n}{p} + log_2 p\right)$$

A serial approach would simply yield a $O(2^n)$ complexity. This can also be seen if we give $p = 1$ to the previous expression.

## Design Choices

In the assignment, `MPI_Send`, `MPI_Recv` and `MPI_Scatter` functions are used for message passing. Process 0 creates the $2^n$ teams and then using `MPI_Scatter`, those teams are distributed to separate processes. Each process conducts a small tournament and determines a winner. In the second part, each of those winners match with each other and the final winner is determined. This second part is also made parallel. $p$ teams match with each other at the same time and then $\frac{p}{2}$ etc. until only one team is remaining. `MPI_Send` and `MPI_Recv` functions are used to determine the winner in this stage. The program has been written this way so that all of the processes are utilized as much as possible.

## Experiments

Figure 1 shows the time versus number of processors graph. As seen from the figure, the time decreases as the number of processors increase. Notice that any match between two teams is independent from the other matches and hence the problem in the assignment is embarrassingly parallel. Therefore, we can easily decrease the time by increasing the number of processes. The figure also supports this argument.

Figure 2 shows the speed improvement versus the number of processes. The speed improvement is calculated as,

$$\text{Speedup} = \frac{T_s}{T_p}$$

where $T_s$ is the running time of the sequential algorithm and $T_p$ is the running time of the parallel algorithm. We see a linear relation in the speedup graph.

The efficiency is defined as speedup divided by the number of processes. As can be seen from Figure 3, the efficiency stays roughly the same around 1.0. Of course, it does not go beyond 1.0 but stays near that point. This is possibly due to the nature of the problem an the algorithm that is used.

We can improve the performance by increasing the number of processes.

A note about the values in submitted `.txt`s and the numbers in figures is that I have run the experiments 10 times each and taken the average value of the values. I have also managed to find a timeslot where no one is working on `slurm` machine to minimize the interference.

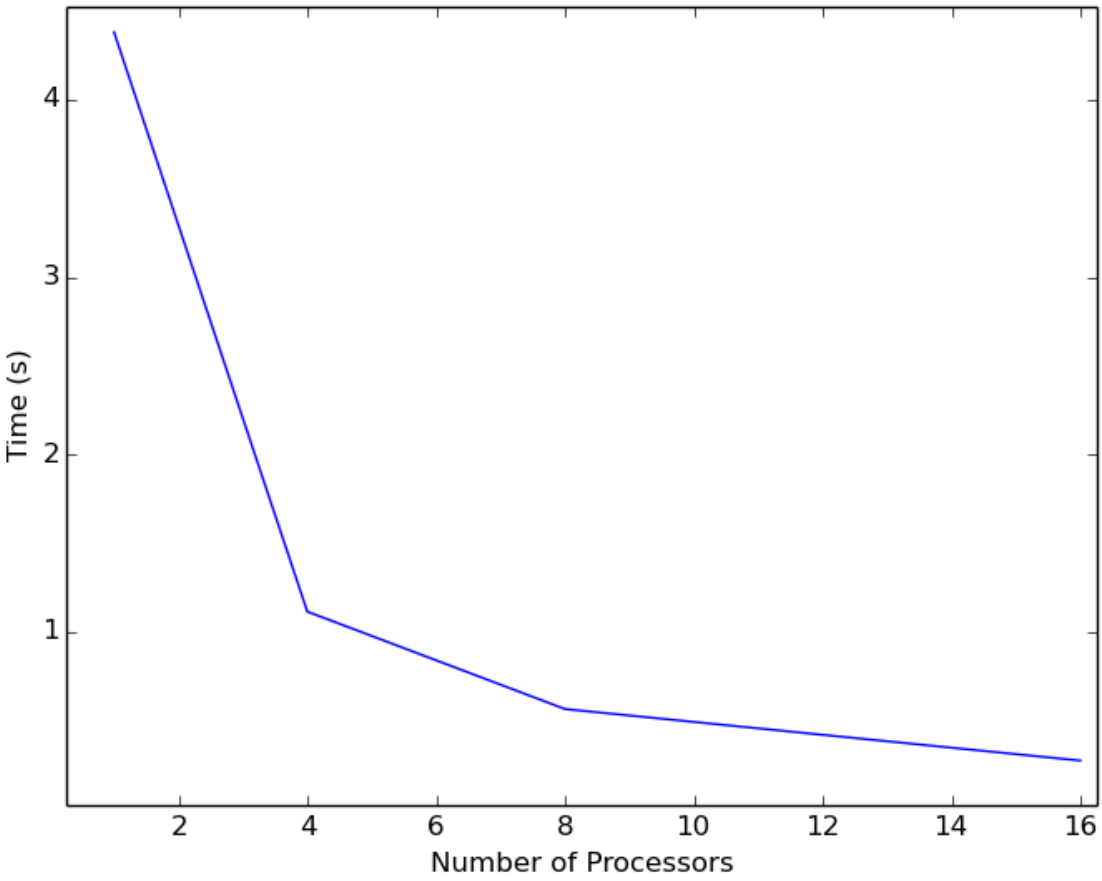


Figure 1: Time vs Number of Processors

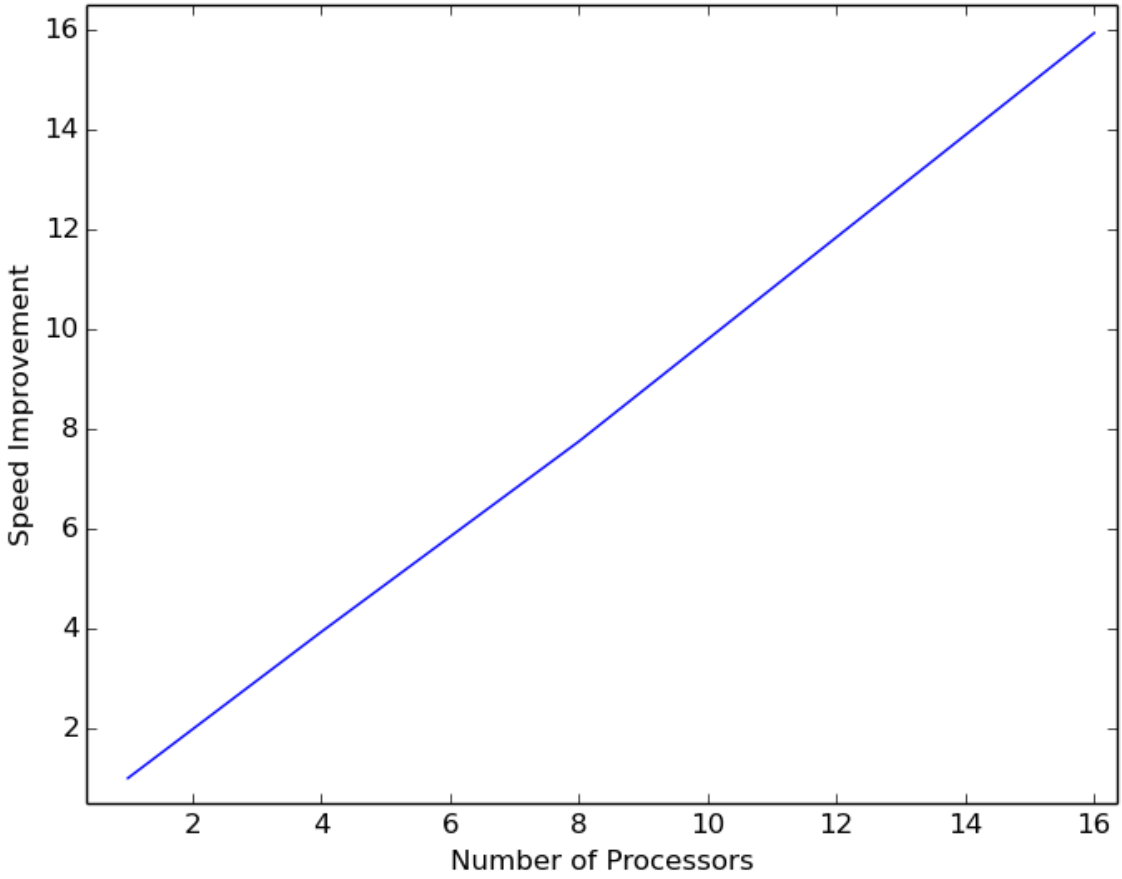


Figure 2: Speedup vs Number of Processors

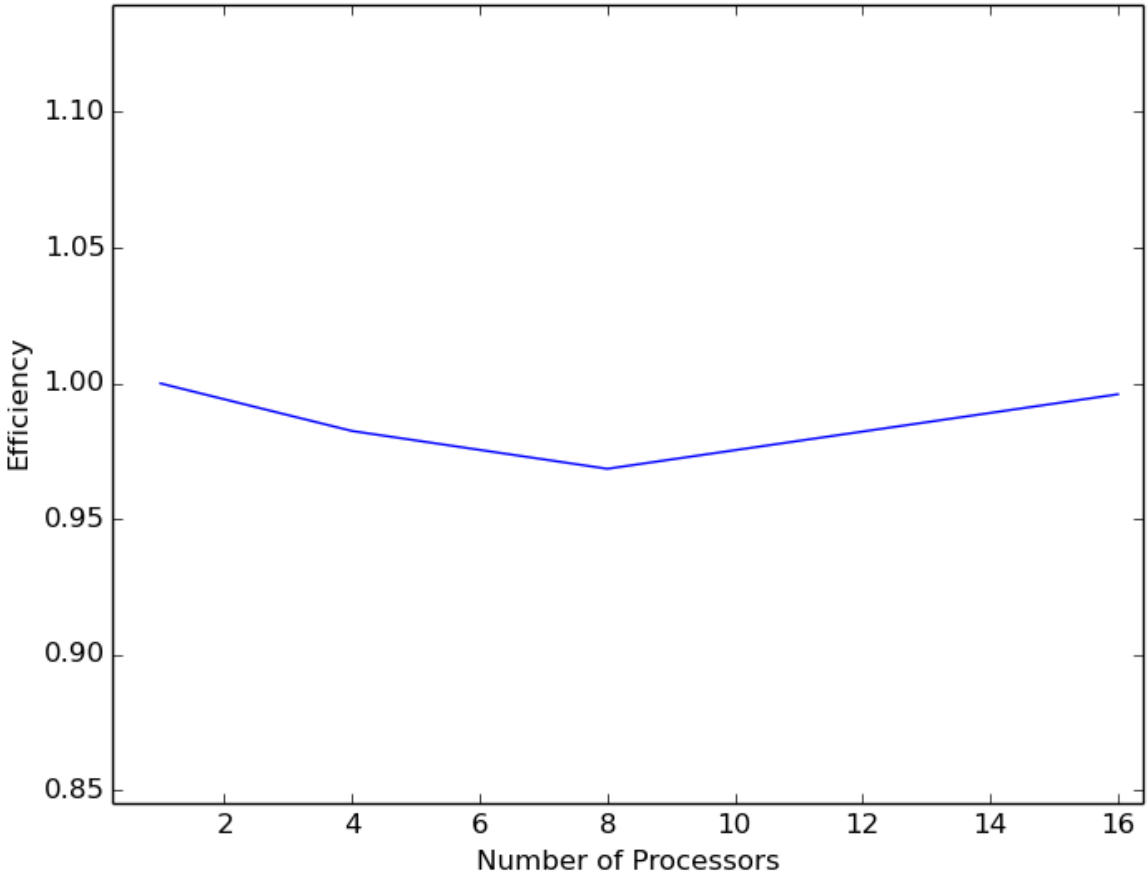


Figure 3: Efficiency vs Number of Processors