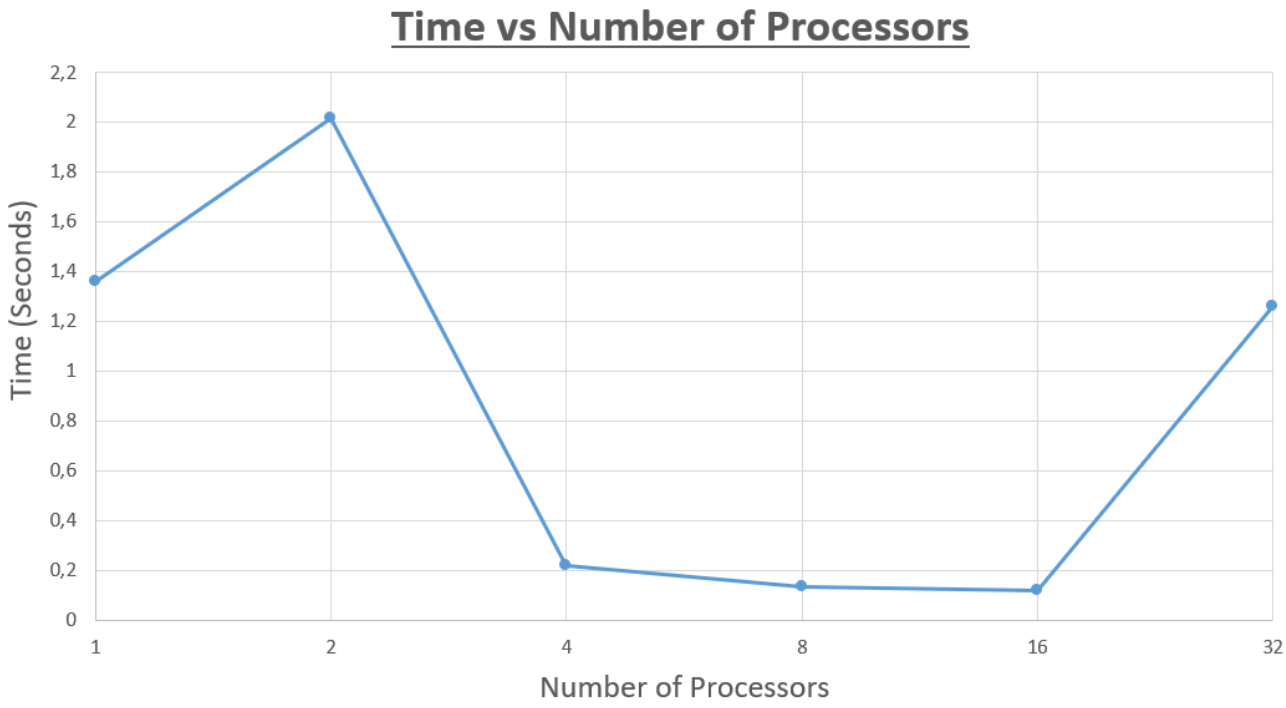
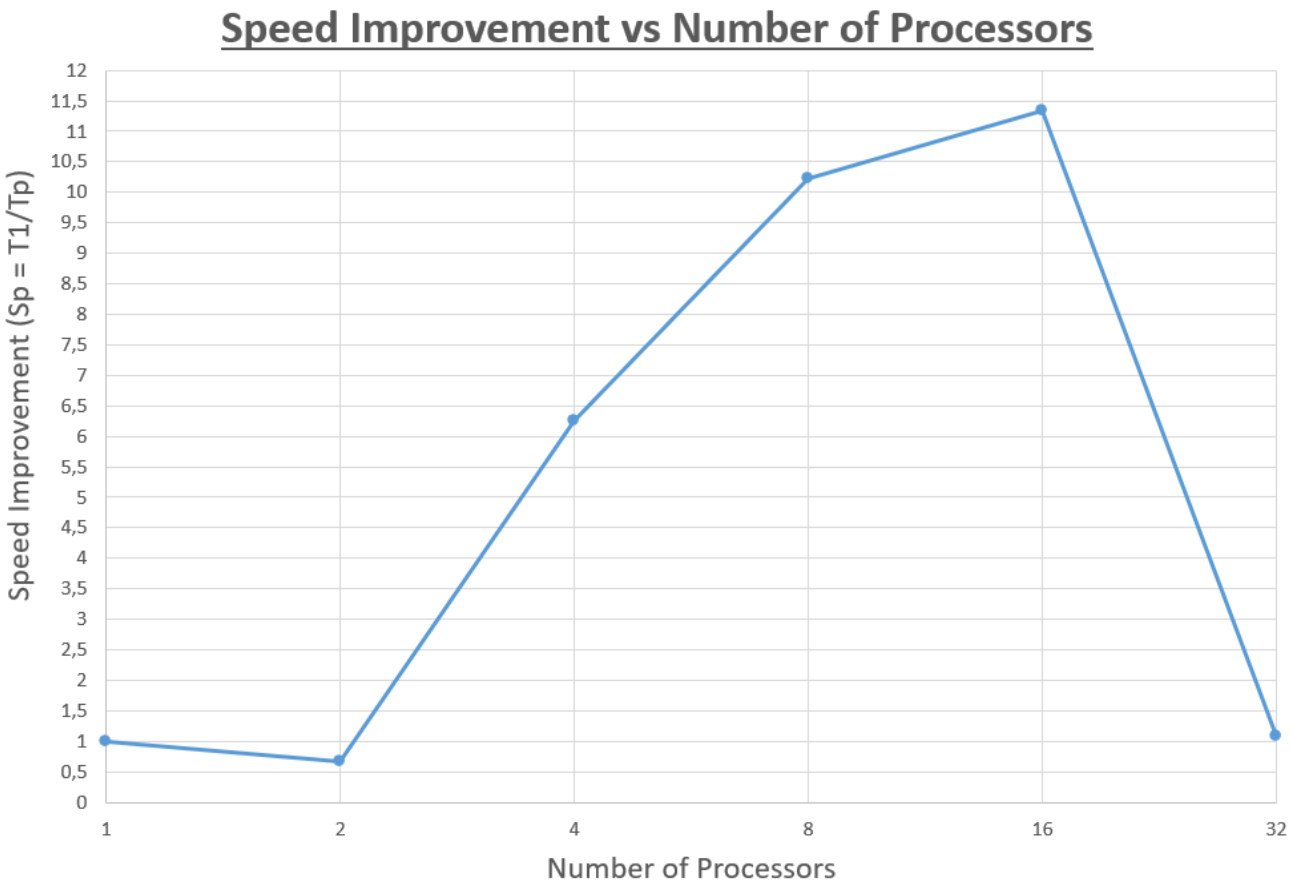
CENG478 - HW2 - Tests and Report

Full Name : Orçun Başşimşek
Id Number : 2098804

a)



Consumed time data for the above graph is taken from my experimental results (from #_out.txt files, # = number of processors). Time axis shows the consumed time from the process that finishes its task latest for each different execution(I used MPI_Reduce for taking maximum elapsed time by the latest processor). As seen in the above graph, increasing the number of processors do not always decrease the consumed time properly. For example, while working with 2 processor, much more time is consumed even if it is compared with 1 processor case. I think the cause of this increase is parallel overhead of course. However, after 2 processor case, when I increased the number of processors, consumed time decreased so much up to 16 processors case. I understand that computation time was smaller and smaller for these cases, and it effects parallel running time positively for us. After that, with 32 processors, again I saw so much time. It may occur due to relatively bigger communication and idle times for 32 processors. As a result, for my implementation, best parallel running time is acquired when working with 16 processors.



I calculated speed improvement(speedup) for each execution as following:

$$S_p = \frac{T_1}{T_n}$$

S_p = Speed Improvement(Speedup)
T_1 = Consumed time with 1 processor
T_n = Consumed time with n processors

As seen in the above graph, execution with 16 parallel processors has the most speedup. If we think that the parallel efficiency is equal to:

$$Parallel\ Efficiency = \frac{S_p}{n} = \frac{Speedup}{Number\ of\ processors}$$

Execution with 4 processors, and then execution with 8 processors gives better parallel efficiency compared to other executions. About speculation of performance improving, I could not easily say that increasing number of processors may improve the performance because while increasing number of processors, each processor may handle less data, but complexity of my algorithm tends

to increase (there will be much more phase at odd-even transposition implementation part of my code). This would cause additional parallel communication overhead. Thus, we need to find optimal point that we gain from parallelization much more.

b)

Quicksort code provided by you consumes 1.334573 seconds at my tests. My parallel algorithm that runs with only 1 processor also consumes 1.362189 seconds. They are very similar because in my implementation, I used your quickSort code to sort my processors' array parts. Thus, actually they do almost same thing for 1 processor case. However, while working with 4,8 and 16 processors, power of parallel computing was very clear and substantial amount of time was gained.