Middle East Technical University
Department of Computer Engineering
CENG 478
Introduction to Parallel Computing
Spring 2019-2020
Assignment 2

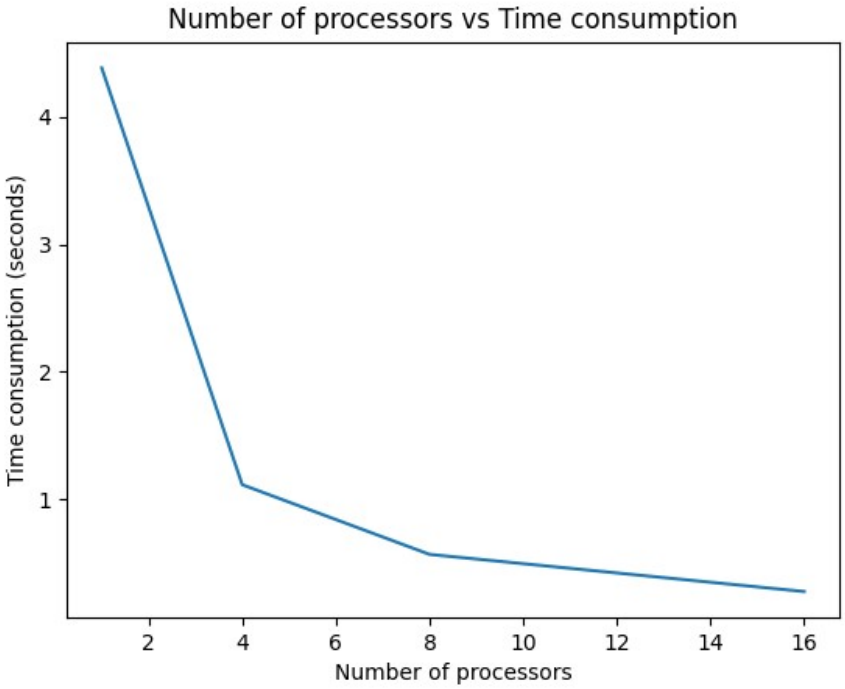
Furkan TAŞBAŞI
2041853

(a) Finding winner of 2^n teams is done sequentially with 1 processor with complexity of O(2^{n+1}). The method which uses parallel processors simultaneously used in homework provides O($\frac{2^n}{p}$) + log(p) complexity that is more efficient then O(2^{n+1}).

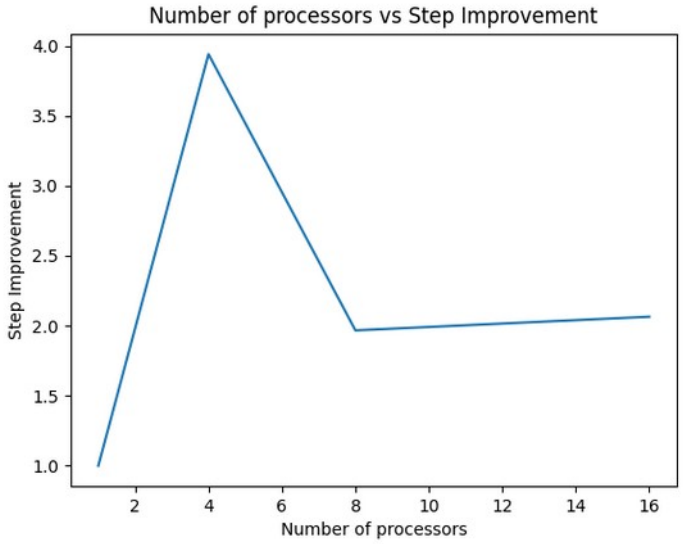
(b) Firstly, the parameter n is read from arg[1] input. Then, I have created a empty array with the length of 2^n in *Process 0.* I send teams(inputs) of other processes using *MPI_Scatter* function which takes main array and sends equal sized portions of it to other processes including itself. While sending corresponding data to processes using *MPI_Scatter*; I decided Process 0 to be a root process. After scattering operation, processes start simulating sub tournaments themselves. For the first round steps are 2 team operations that is one simulation is done for each 2 team. If a team is winner, it's id is placed into itself. Before second round the losers are trashed and only winners are placed into new upgraded array. For the rest of the rounds, we have arrays filled with ids and for each simulation one id which is loser of that match is freed from array. After all simulation is done; all processes will have only 1 team as a sub tournament winner. We get these winners together for quarter final. For quarter final all odd numbered processes send their winners to its' partner process and then this sending process ends. The regarding partnership is calculated via following formulas : To becoming sender or receiver process is calculated via "($process\_rank$ (mod 2^{round})", round count to continue is calculated via $\log_2(number\ of\ processes)$ and finally partner process's rank is calculated via $partner = rank - 2^{(round-1)}$. If a process is a sender which is odd numbered process, it would send it's winner (only one integer) with a tag which is equal to receiver process's rank. So target process's rank is used as both partnership between two processes and a session flag structure that keeps them away from deadlocks since I used blocking approach. After scattering teams to processes, *Process 0* gets starting time with *MPI_Wtime* function. I used *MPI_Wtime* function instead of operating system's time functions because *MPI_Wtime* provides a higher resolution timer as wall-clock time in a cheaper way in comparison with system time calls. Finally, all simulations are completed and consumed time is calculated by *Process 0.*
Finally, I finished program with *MPI_Finalize* function.

(c) My outputs table is given below.

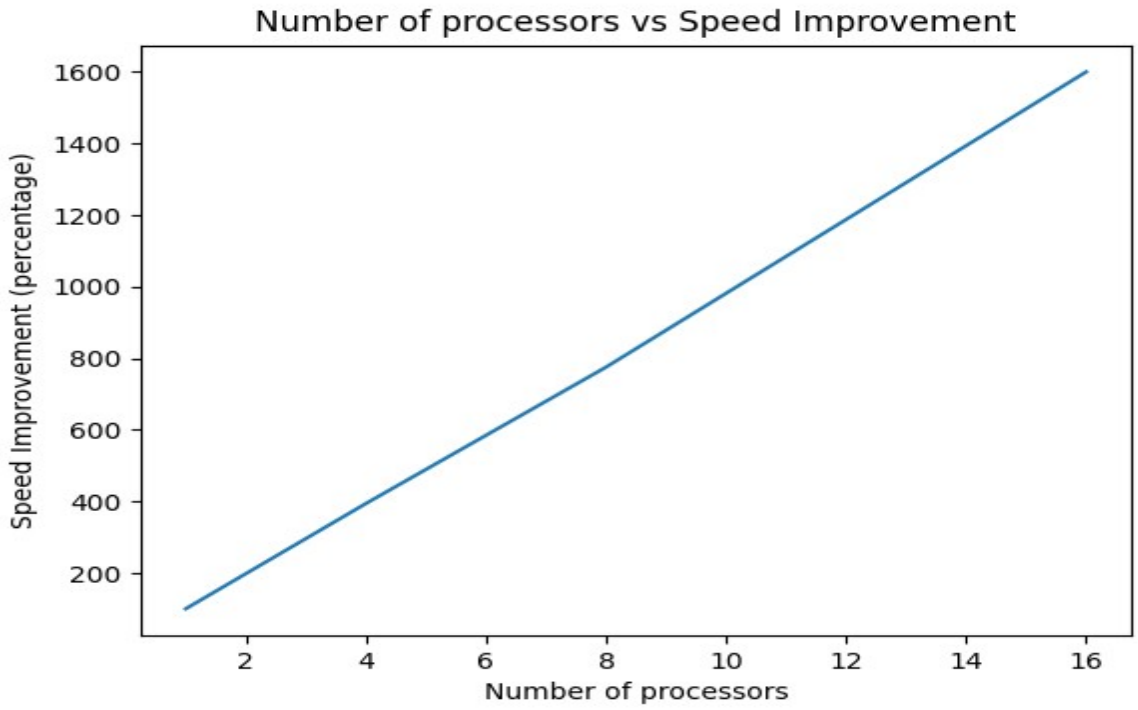
Number of processors	Total_Time
1	4.3846866
4	1.1127734
8	0.5656638
16	0.273978



(d) I've taken average time consumption of all tasks (each task was run 10 times) with respect to number of processors used to plot this graph. As given in the plot above; time consumption decreases proportionally to processor count for 4 processors case which is approximately 4 times faster in comparison with 1 processor case. Moreover, the time consumption decreases proportional to processor count in other cases such as 8 and 16 processors cases . If we look at the graph below which is named as "Step Improvement vs. Number of processors", we can conclude that step improvement in comparison with 1 processor case increasing suddenly for 4 processors case, then it continues increasing through 16 processors case and its' increase ratio is keeping the proportionality with processor count that is comparison with 1 processor case; 4 processor case is 4 times faster and 16 processor case is 16 times faster than first case. Since the overall task is so large we can still can take better performance with providing more processor to it. Moreover, we can increase processor count up to the case where one processor can have at most 2 teams to match up then it will be the peak maximum performance point that is be explained by Amdhal's Law that the speedup is not unlimited for parallel processes, it converges to a max point in limit and does change a lot, the max point is calculated theoretically by Amdhal's Law.



all steps are compared with previous steps speed



As given in the “Speed Improvement vs. Number of processors” graph above, we see that the speedup ratio is increasing dramatically up to 4 processors case (which is 4 times faster than 1 processors case). After that point, the speedup increase continues in a same manner. Since the tournament task we do here is so suitable for parallelization we can get max efficiency with max performance here by adding more processors. We can only trouble with performance loss because of inter-process communication delays for this specific task. Finally, from the Amdhal’s Law performance max point converges to a theoretical point in future cases (more processors used), because the speedup is not dependant only up to processors count, it is also dependent both to task properties such as problem size (especially sequential and unparallizable part) and other hardware issues. If we want to keep performance increase positive after the max(overhead) point that no more performance gain is provided though we increase processor count, we can both try to make the unparallel sub-tasks parallel as much as possible and after that we can increase data size and processors count and with a ratio which should be same with the max point’s ratio, this principle is called as isoefficiency that is stated by Gustaffson’s Law.