Middle East Technical University
Department of Computer Engineering
CENG 478
Introduction to Parallel Computing
Spring 2019-2020
Assignment 1

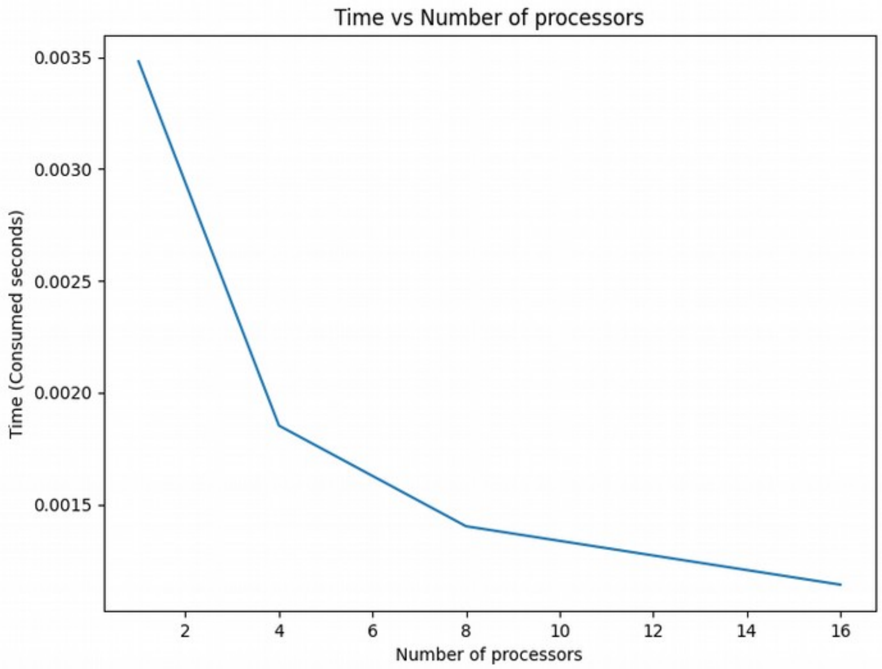
Furkan TAŞBAŞI
2041853

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

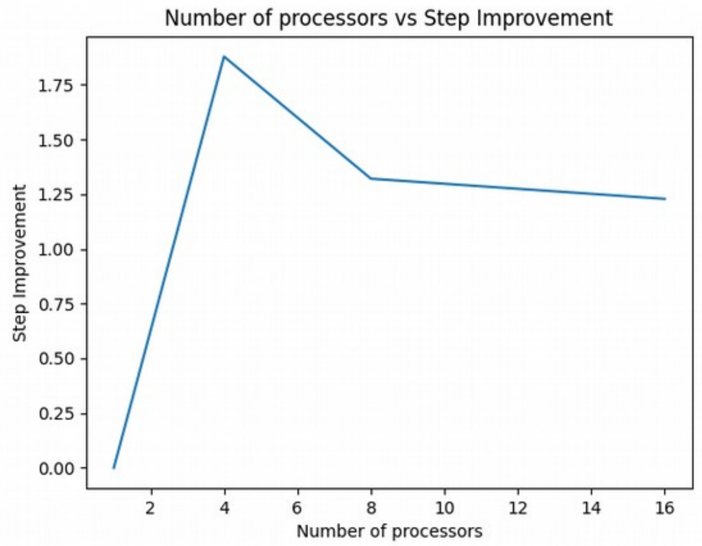
(b) Firstly, I read input from file in *Process 0*. I wanted to send inputs of other processes as blocks (Blocking approach) which is float arrays that's why i created a MPI datatype named as *rowtype*. I used *MPI_Type_contiguous(partialSIZE, MPI_FLOAT, &rowtype)* structure for creating a *MPI_Float* array type sized as *partialsize* and named as *rowtype*, *MPI_Type_commit(&rowtype)* structure for committing this datatype. Then I put all data into temporary 2D float array. After that, I placed all rows of 2D temporary array into *rowtype* *MPI_Float* row structure then I sent this data type to corresponding processes sequentially. While sending corresponding data to processes; I used *MPI_Send* and *MPI_Recv* functions with tags because I wanted to stay away from possible deadlocks while doing synchronous operations between processes (Blocking Approach: processes waits until exact input/output size) and *MPI_Send* works after sending data which is useful because processes are becoming Sender or Receiver up to task in this program. To becoming sender or receiver process is calculated via "($process\_rank \pmod{2^{round}}$)", round count to continue is calculated via $\log_2(\textit{number of processes})$ and finally partner process's rank is calculated via $partner = rank - 2^{(round-1)}$. If a process is a sender, it would send data 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 because sometimes a receiver may have more than one partner process. After sending data to partner 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 computations are completed and consumed time is calculated by *Process 0*, and I freed the MPI datatype *rowtype* with *MPI_Type_free* function then I finished program with *MPI_Finalize* function.

(c) My outputs table is given below.

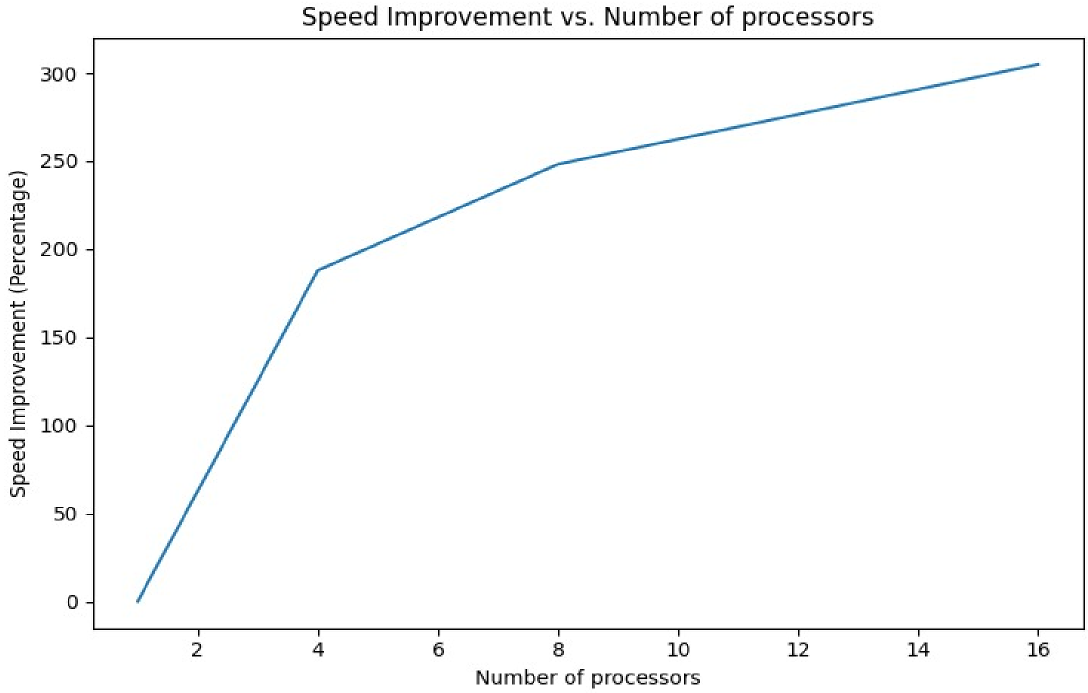
Number of processors	Max_Value	Total_Time
1	4.8	0.003481
4	4.8	0.001853
8	4.8	0.001403
16	4.8	0.001142



(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 dramatically for 4 processors case which is approximately 2 times faster in comparison with 1 processor case. Moreover, the time consumption decreases more in other cases such as 8 and 16 processors cases but those are not as dramatic as in the 4 processors case and it continues in a gradual way that it converges to a limiting point. 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 but its' increase ratio is getting lower. It can 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 0.5 times faster than 1 processors case). After that point, the speedup increase continues gradually. From the Amdhal’s Law it 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, we can increase data size and processors count 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.