First Assignment Report

Section 0

Theoretical Peak performance for laptop (Hp-Pavillion):

- CPU: Intel ® Core™ i7-7500U.
- Base frequency 2.70 GHz.
- 2 cores
- Floating point operations per cycle: 16 (Intel Kaby Lake architecture, https://en.wikipedia.org/wiki/FLOPS)

	Model	CPU	Frequency	Cores	Peak performance
Laptop	i7-7500U	1	2.70 GHz	2	86.4 GFLOPS

Sustained and theoretical peak performance for smartphone (Xiaomi Mi A1):

- CPU: Octa core Qualcomm Snapdragon 625.
- Frequency: 2 GHz.
- 2 FLOPS

	Model	Sustained performance	Matrix size	Peak performance	Memory
Smartphone	Qualcomm Snapdragon 625	1209 MFLOPS	2500	32 GFLOPS	4 GB

Top 500:

	Model	Performance	Top 500 year	Number 1 HPC system	Number of processors (TOP500)
Smartphone	Qualcomm Snapdragon 625	1209 MFLOPS	Until November 1994	Until 1985 (Cray-2, 1.9 GFLOPS)	4 (Cray 2)
Laptop	i7-7500U	86.4 GFLOPS	Until November 2001	Until November 1993 (Numerical Wind Tunnel Japan)	140 (Numerical Wind Tunnel Japan)

Section 1

Theoretical model for parallel sum of N numbers.

T_{comp} = Time to compute a floating point operation.

T_{read} = Time to read from file.

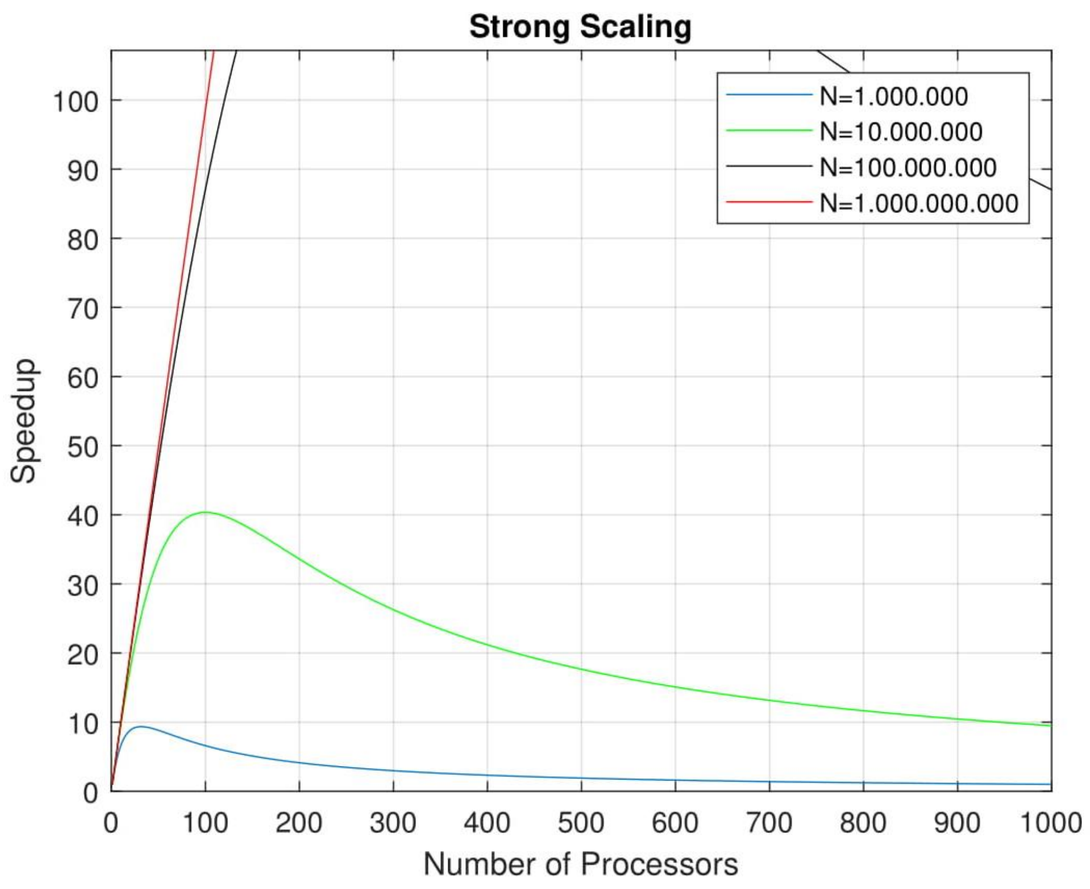
T_{comm} = Time for each processor to communicate a message.

Parallel algorithm (master-slave):

- Master processor reads N from input file $\rightarrow T_{read}$
- Master processor distributes N to each slave $\rightarrow (P-1)*T_{comm}$
- N/P sums over each processor (including master) $\rightarrow T_{comp}*N/P$
- Slaves send partial sums $\rightarrow (P-1)*T_{comm}$
- Master performs one final sum $\rightarrow (P-1)*T_{comp}$

Final model: $T_p = T_{read} + T_{comp}*(P-1+N/P) + 2*(P-1)*T_{comm}$

Assumptions: $T_{comp} = 2*10^{-9}$, $T_{read} = 10^{-4}$, $T_{comm} = 10^{-6}$



In this plot we see how the model scales when increasing the number of processors. For all values of N the algorithm doesn't scale well, as after an initial increase in performance the speedup actually diminishes. Of course the bigger N is the later the performance decreases, for instance when N=10.000.000 there is a strong increase in performance for P lower than 90. The decrease in performance is due to the communication time, that for large P is bigger than the benefit from having more processors. Therefore a way to improve the scalability of the algorithm is to reduce the communication time by implementing collective operations between processors.

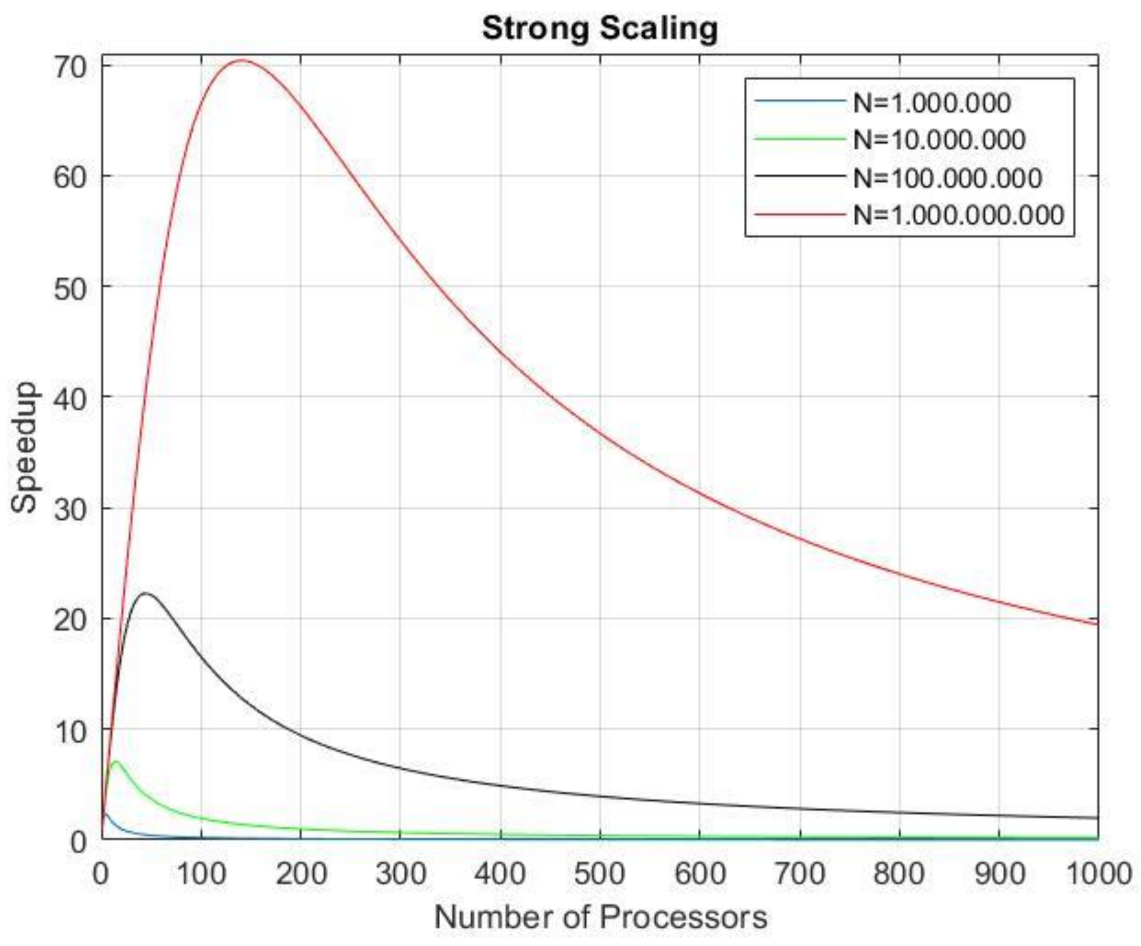
The model I implemented in section 3 is slightly different:

Parallel algorithm (master-slave):

- Each processor reads N from input file $\rightarrow$ $P^* T_{read}$
- N/P sums over each processor (including master) $\rightarrow$ $T_{comp}*N/P$
- Slaves send partial sums $\rightarrow$ $(P-1)*T_{comm}$
- Master performs one final sum $\rightarrow$ $(P-1)*T_{comp}$

Final model: $T_p= P^* T_{read} + T_{comp}*(P-1+N/P) + (P-1)*T_{comm}$

Assumptions: $T_{comp} = 2*10^{-9}$, $T_{read} = 10^{-4}$, $T_{comm} = 10^{-6}$

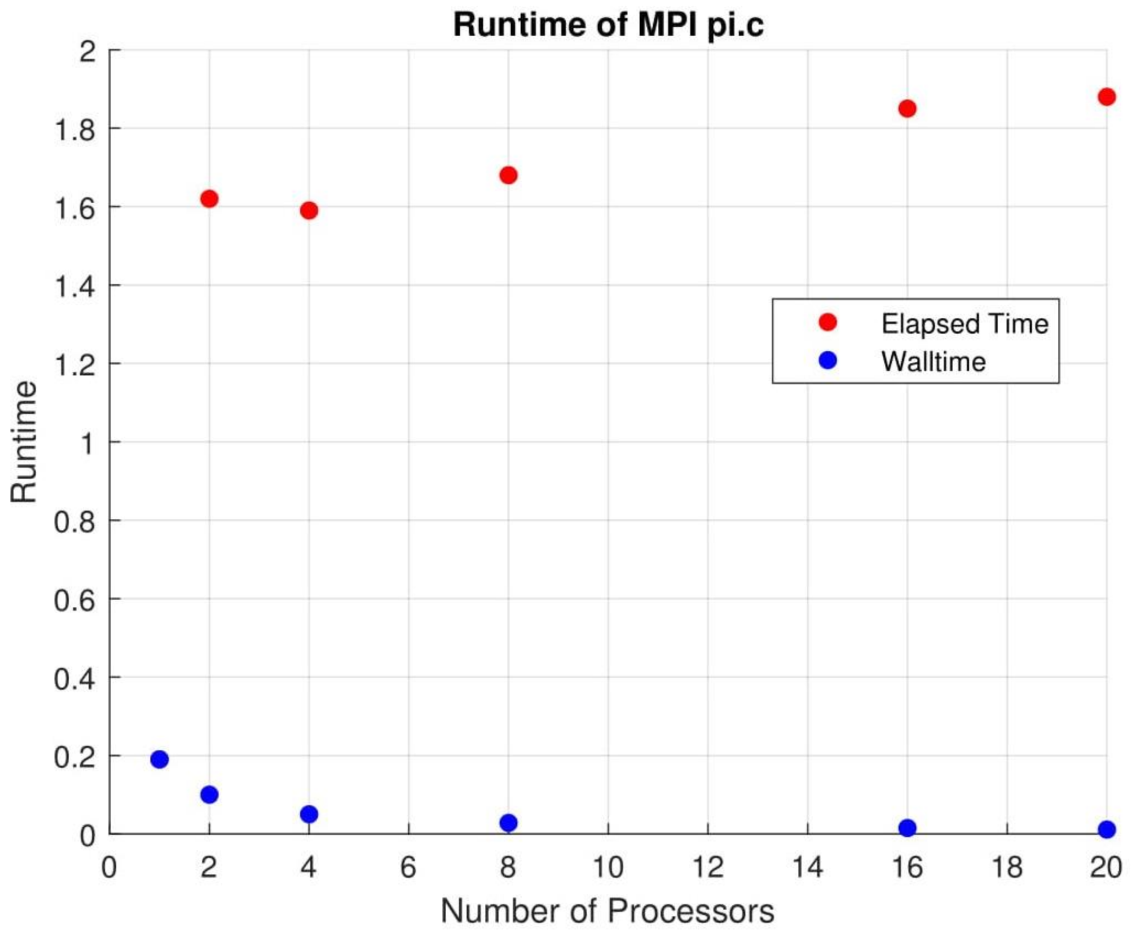


What changes from the previous model is that rather than sending the input to each slave all processors read from the file. Under the assumption we have made this model is not as good as the previous one, as the speedup is quite lower. I chose this model as during testing I noticed that the communication time between processors was as high as $2*10^{-3}$, way larger than our assumptions. As before a way to improve it is to use collective operations, thereby reducing the communication time between processors.

Section 2.1

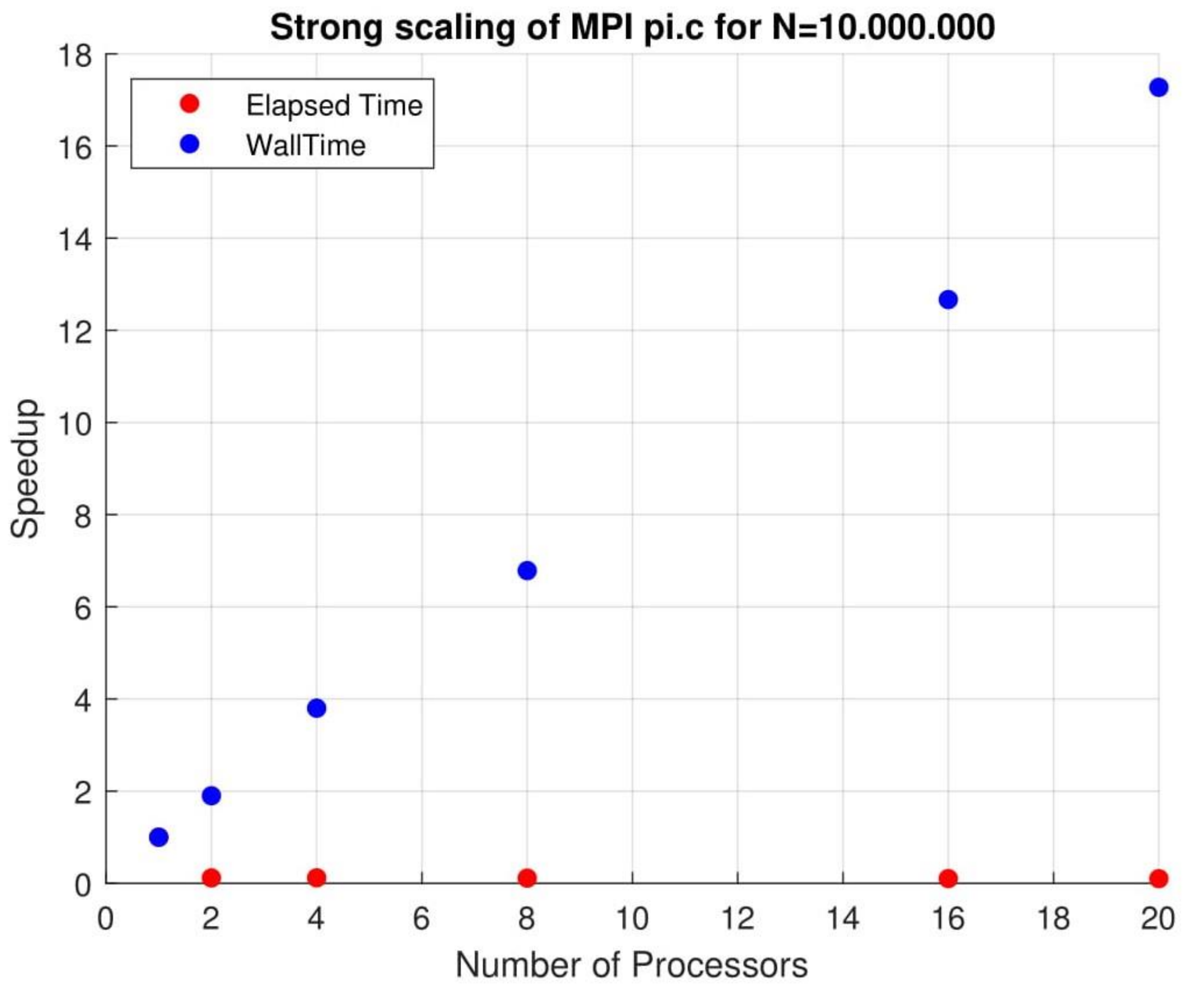
N=10.000.000

- Serial time of execution is 0.19 s.
- Parallel time of execution with 1 processor is 1.7 s, therefore the overhead caused by MPI machinery is approximately 1.5 s.



In this plot we compare runtime over number of processors, considering both the walltime of the processor which took the longest to finish and the elapsed time given by /usr/bin/time. In this case the overall runtime increases as P increases, meaning that there is no advantage in doing the computations faster with more processors.

We can then consider strong scalability.

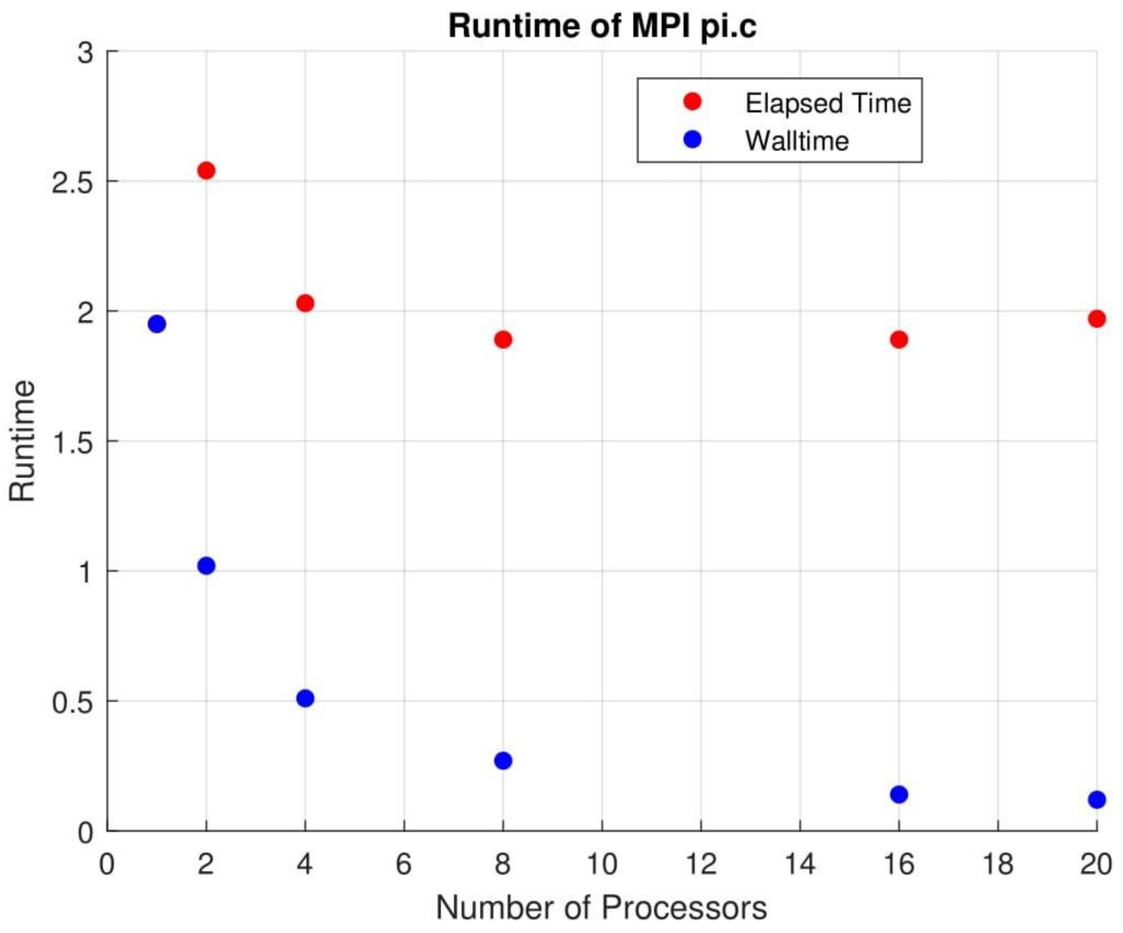


The speedup is calculated as T_S/T_P, where T_S is the serial time of execution and T_P the parallel time with P processors. The walltimes scale almost linearly, as expected, whereas the elapsed time indicate that the program doesn't scale at all for this problem size. This is due to the overhead caused by MPI, which with such small N is predominant.

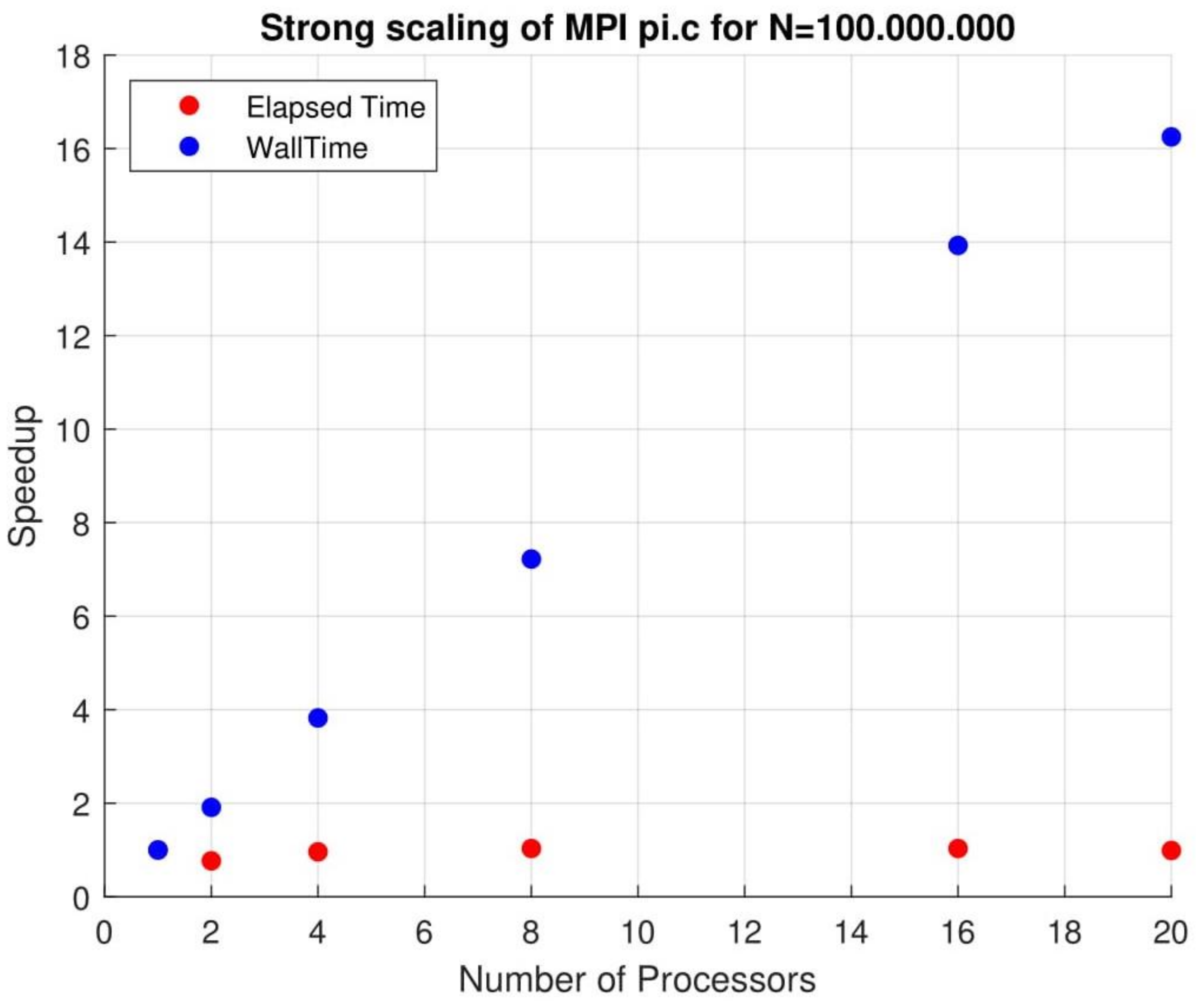
N=100.000.000

Serial time of execution is 1.9 seconds.

As before, we consider the runtime versus processor plot and the strong scaling of the algorithm.



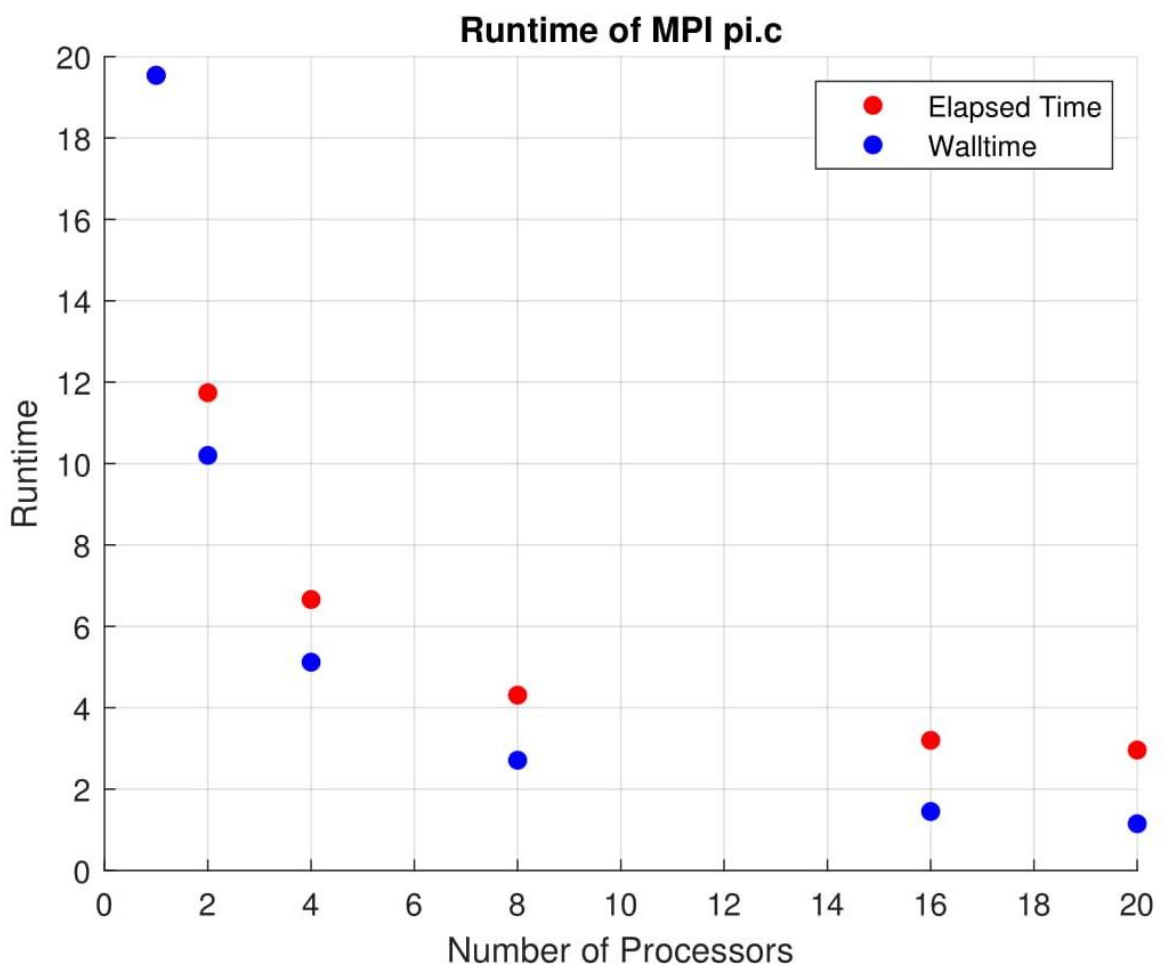
In this case both the elapsed runtimes and the walltimes diminish as P increases, at least in the beginning.



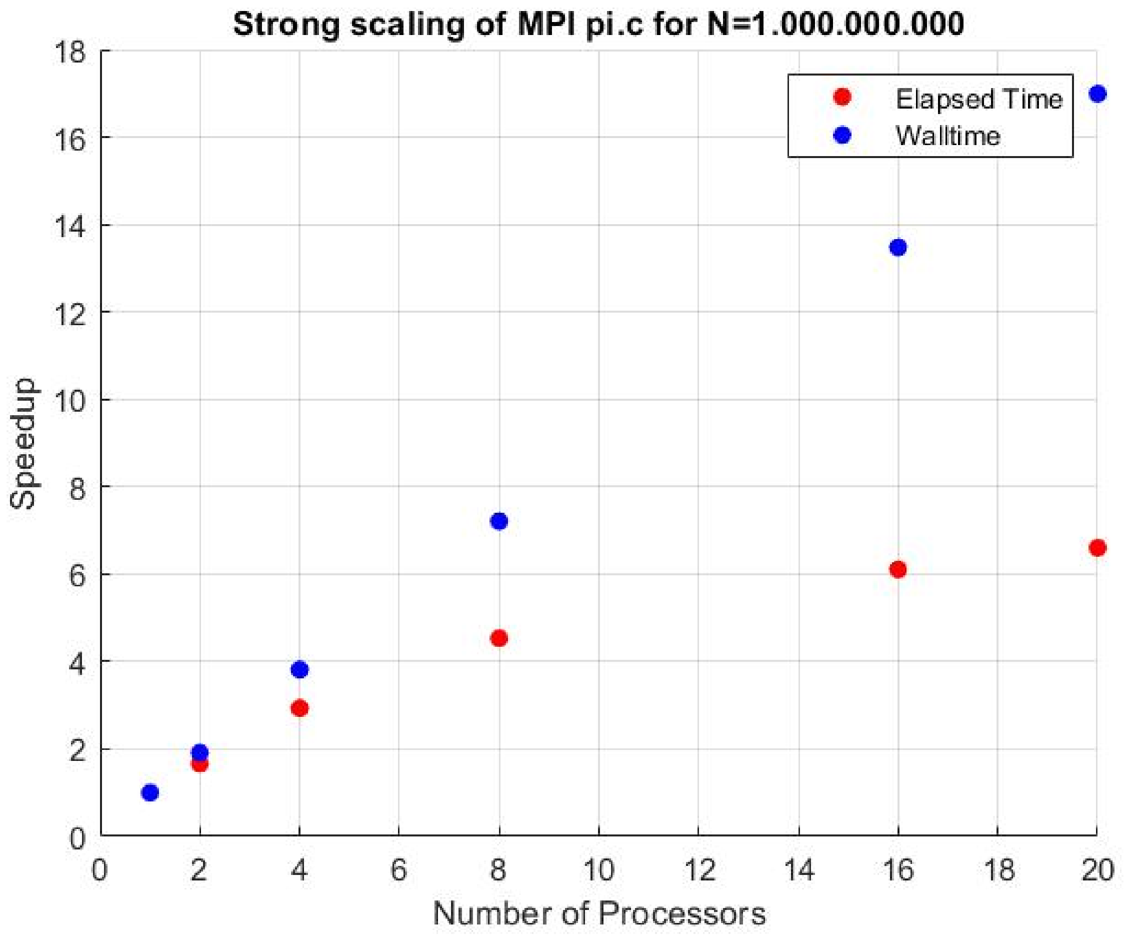
Yet again the walltimes scale almost linearly, whereas the overall speedup is almost constant, meaning that the number of iterations is sufficiently big to partially compensate for the MPI overhead.

N=1.000.000.000

Serial time is 19.54 s.

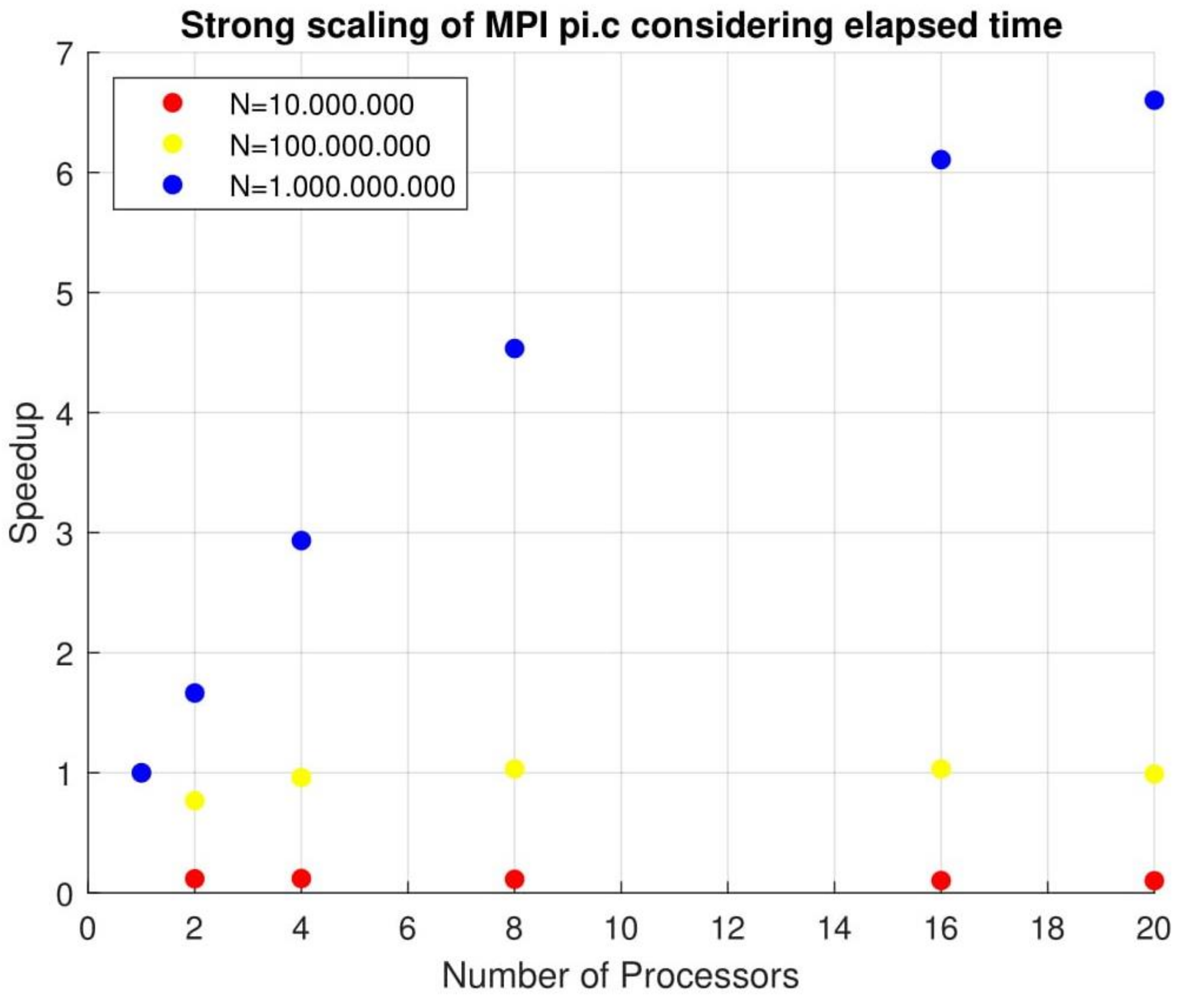


Now both walltimes and elapsed times follow almost the same curve. Considering scaling we get:



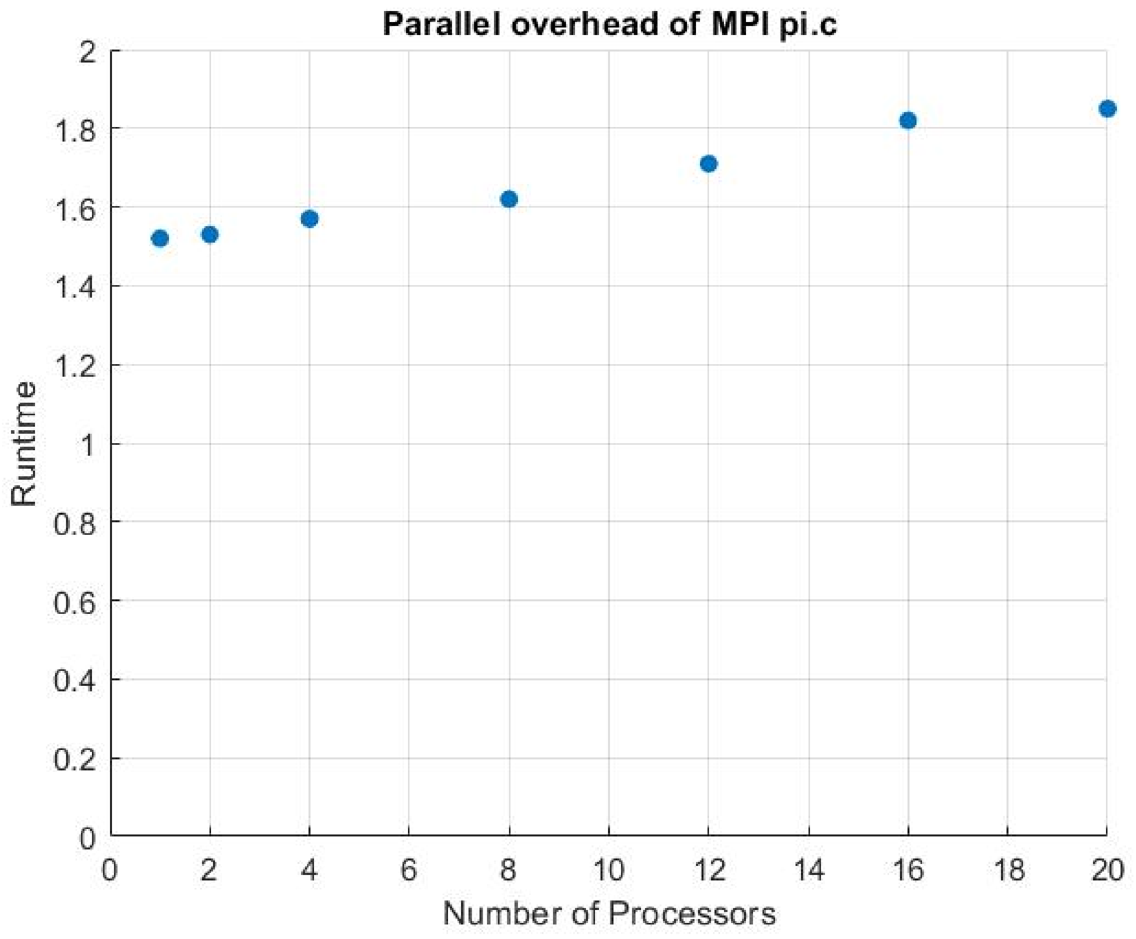
As before the walltimes scale perfectly. The elapsed time scale as well, and the curve they follow is reminiscent of Amdahl's law.

In the last plot we compare strong scaling for the three different values of N. I decided to consider only the elapsed times, as they represent the actual time it takes for the application to end.



Section 2.2

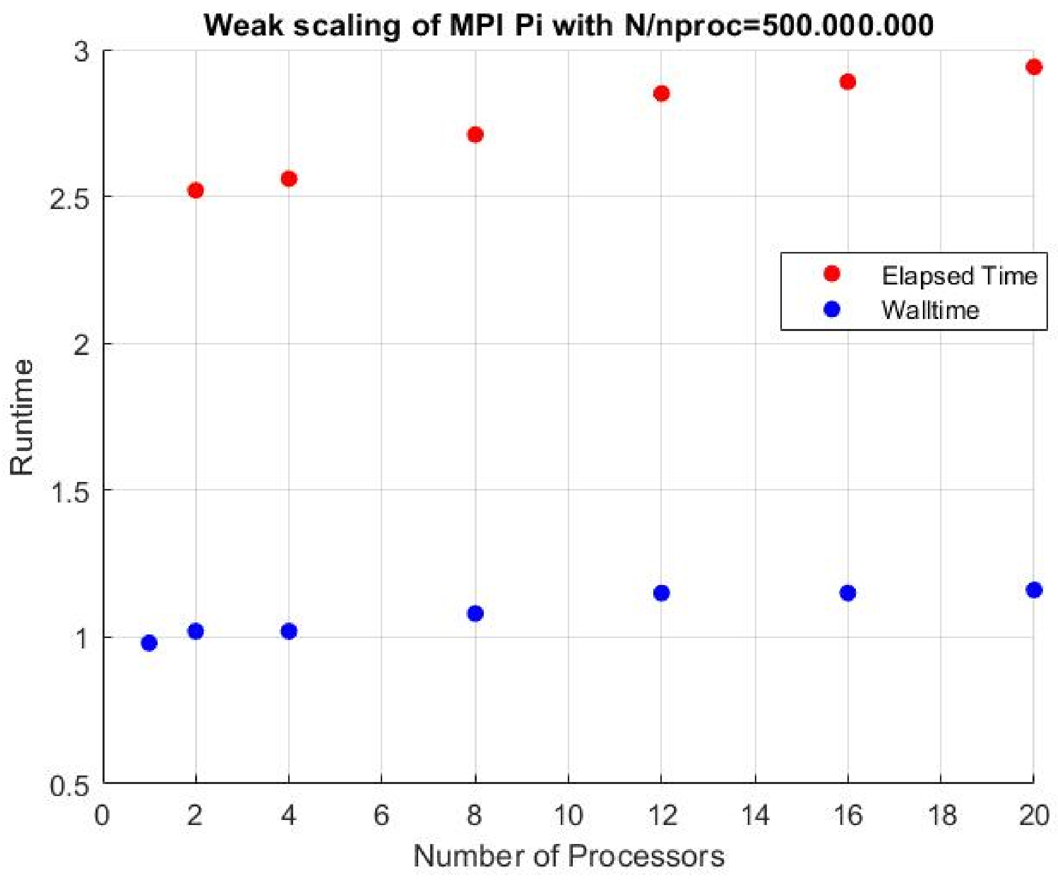
The parallel overhead is made up of different factors: the MPI overhead, the communication times between processors and synchronization times. To estimate it, it is sufficient to run the code without any actual computation. Not considering the constant 1.5 seconds given by MPI inizialization, the overhead increases with the number of processors, as expected, albeit not significantly.



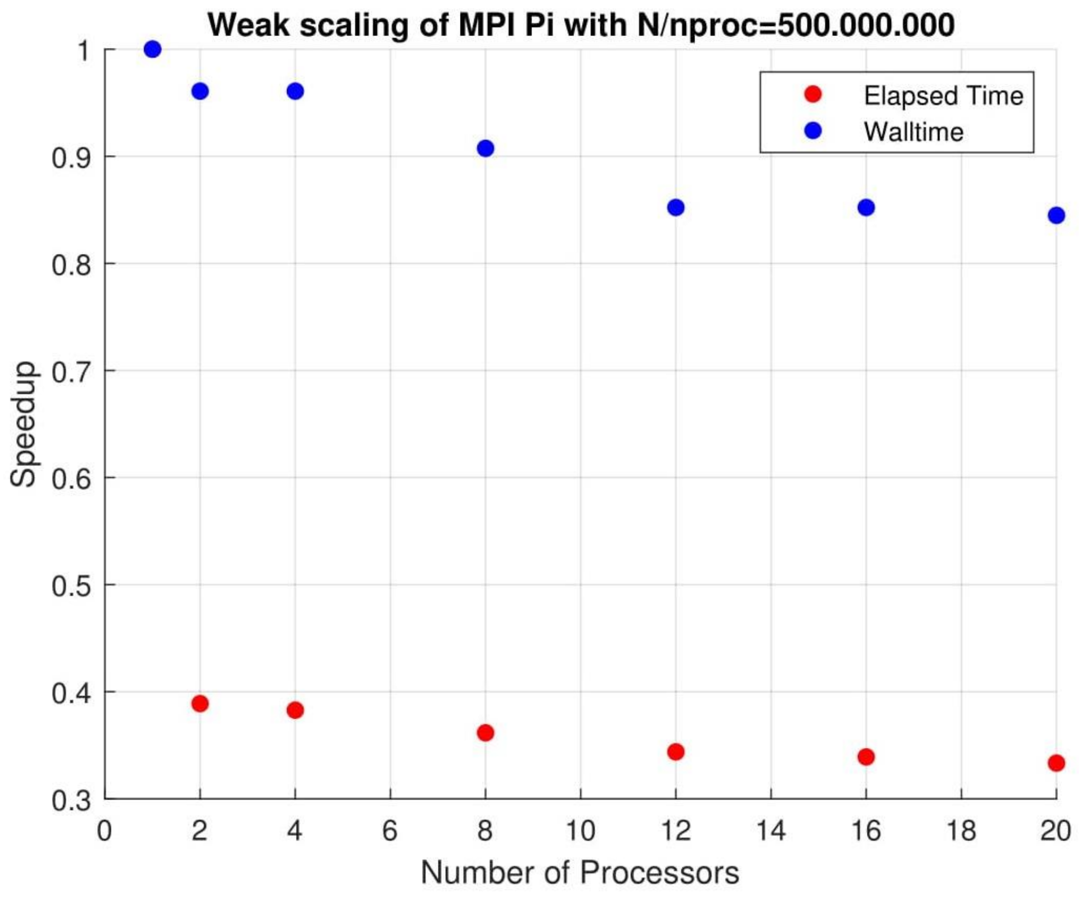
Section 2.3

In the weak scaling tests I fixed a constant of N/P = 500.000.000

The runtime plot is the following, where we can see that the runtime doesn't remain constant as the number of processors and the problem size increase simultaneously, but it slightly increases.



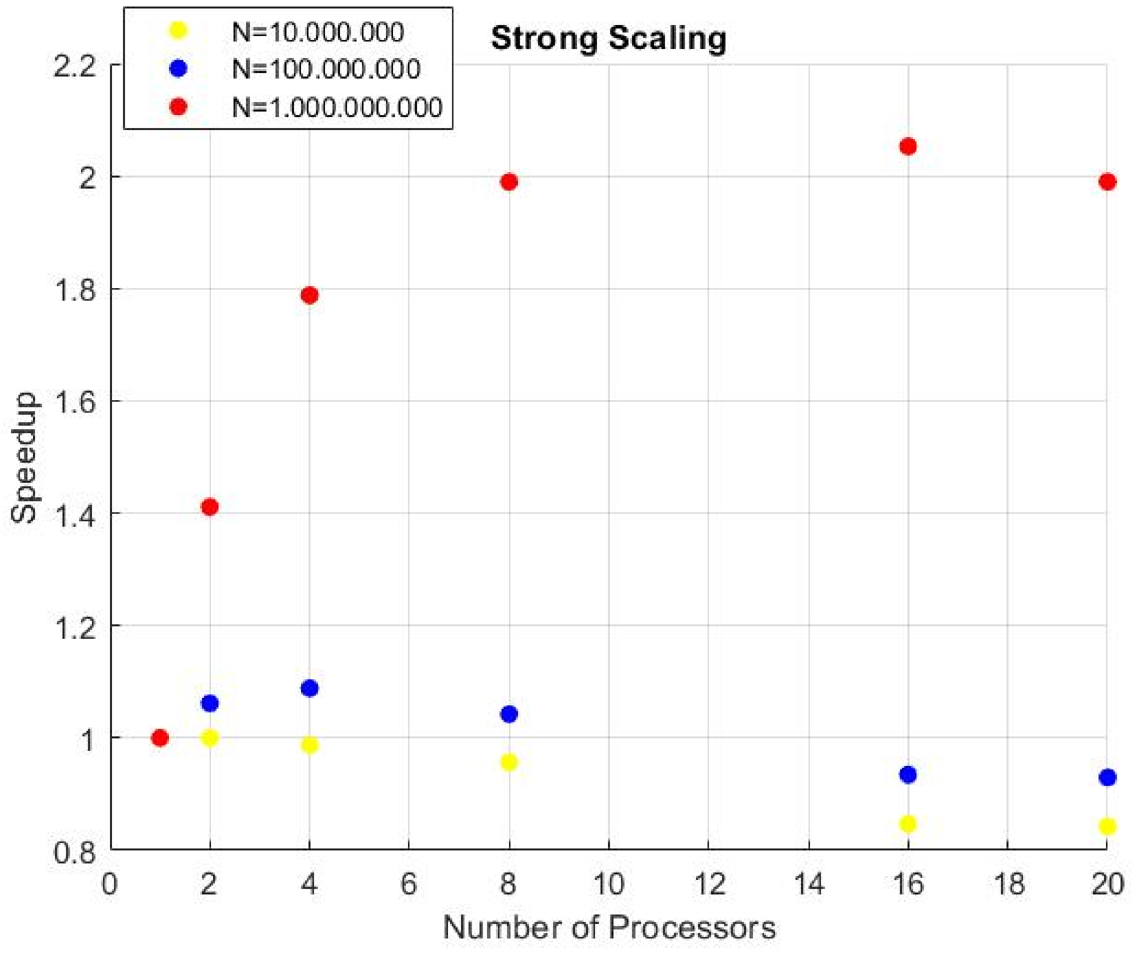
Plotting the efficiency T(1)/T(P), we get a weak scalability plot.



Therefore we can conclude that the application doesn't scale perfectly, as evidently when increasing the number of processors there is a non negligible amount of time spent in communications.

Section 4

Scalability of the parallel program to sum N numbers.



Speedup was obtained by plotting T(1)/T(P). The program scales a bit, for N sufficiently large, as P increases, although the speedup is way less than the one obtained in the corresponding theoretical model in section 1. This is due to the fact that, as already mentioned, the communication time was of the order of 10^{-4} , with a peak of $2*10^{-3}$ obtained when running with 16 processors, rather than 10^{-6} as assumed in section 1. Instead both the reading time and the computing time were approximately as assumed in section 1.