Assignment 1

Mattia Mencagli (mmencagl@sissa.it)

November 4, 2020

Question 1.

For which values of N do you see the algorithm scaling?
At N=2000 the maximum value of the scalability is for one processor, so the algorithm does not scale. For N≥3000 the algorithm scales well until a certain value of P, in Fig.1 I report an example of this behavior at a fixed value of N. The best value of P increases with N as I show in the following question.

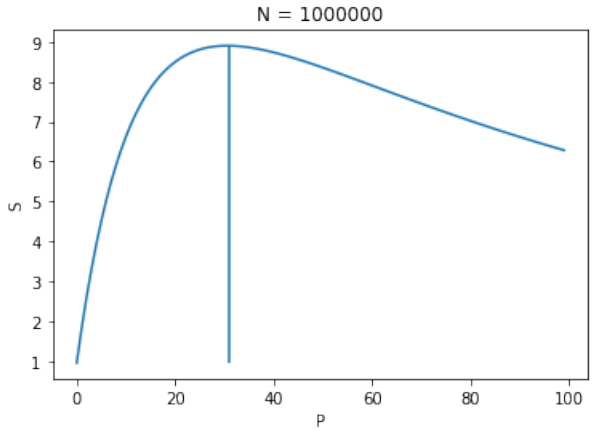

Figure 1: S versus P, at the fixed value of $N = 10^6$.

For which values of P does the algorithm produce the best results?
The best value of P increase with N as it is shown in Fig.2.

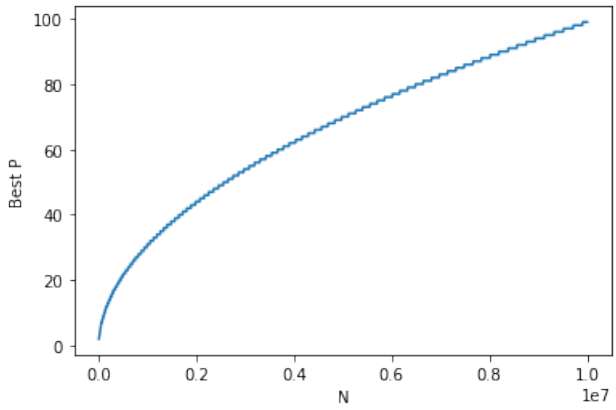

Figure 2: Best value of P in function of N that is in range $(10^4, 10^7)$.

Can you try to modify the algorithm sketched above to increase its scalability?
I try, but I did not found anything.

Question 2.

2.1 **Strong scalability**

I verify that for the for serial algorithm (MOVES=10^8) the walltime measured by the C clock and the /usr/bin/time coincide exactly, while for the mpi algorithm sent on a single core there is a little difference as is reported in Table.1.

	Walltime	/usr/bin/time
serial	2.62	2.62
mpi(s.c.)	2.63	2.72

Table 1: Table of measured time for the two run. Everything is in seconds. mpi(s.c.) stay for mpi on a single core.

Moroever the mpi(s.c) is slightly slower with respect to the serial probably because of the use of the mpi routine that implies the use of a lot of mpi functions with their computational costs.

For the parallel run I am going to consider as the parallel time the bigger walltime measured by cores, because it represents the total time that the computer takes to finish the run.
In Fig.3 and Fig.4 I present the time and the scalability versus the number of processors for four different value of the maximum iteration ($10^8, 10^9, 10^{10}, 10^{11}$). The time measured by /usr/bin/time is always a little longer with respect to the time measured in code. However, this is relevant for the runs with 10^8 and 10^9 iterations, and it is clear that increasing the iterations the two measured times going to coincide. It is interesting to notice that the linear behaviour (the perfect scalability) is almost achieved everywhere the time was measured with the code clock. However the linear behavior is retrieve only for the runs with a big number of iterations whether the time was measured with /usr/bin/time. This is due to the fact that the /usr/bin/time measures also the starting and the final part of the code, that is serial and where the clock is not present. Increasing the number of iteration we also increase the work to do in the parallel part, this decrease the total percentage of serial code and so the difference in the two measurements tends to disappear. Another interesting aspect to notice is the fact that all the run follow the perfect scalability until $P \simeq 24$, then at higher P they make worst. In Fig.5 I report the scalability for the four different nombuer of iterations compared. Here, looking at the code clock, we can notice that the run with 10^8 iterations reach his peak performance around $P = 32$, while the other run hasn't reach it yet.

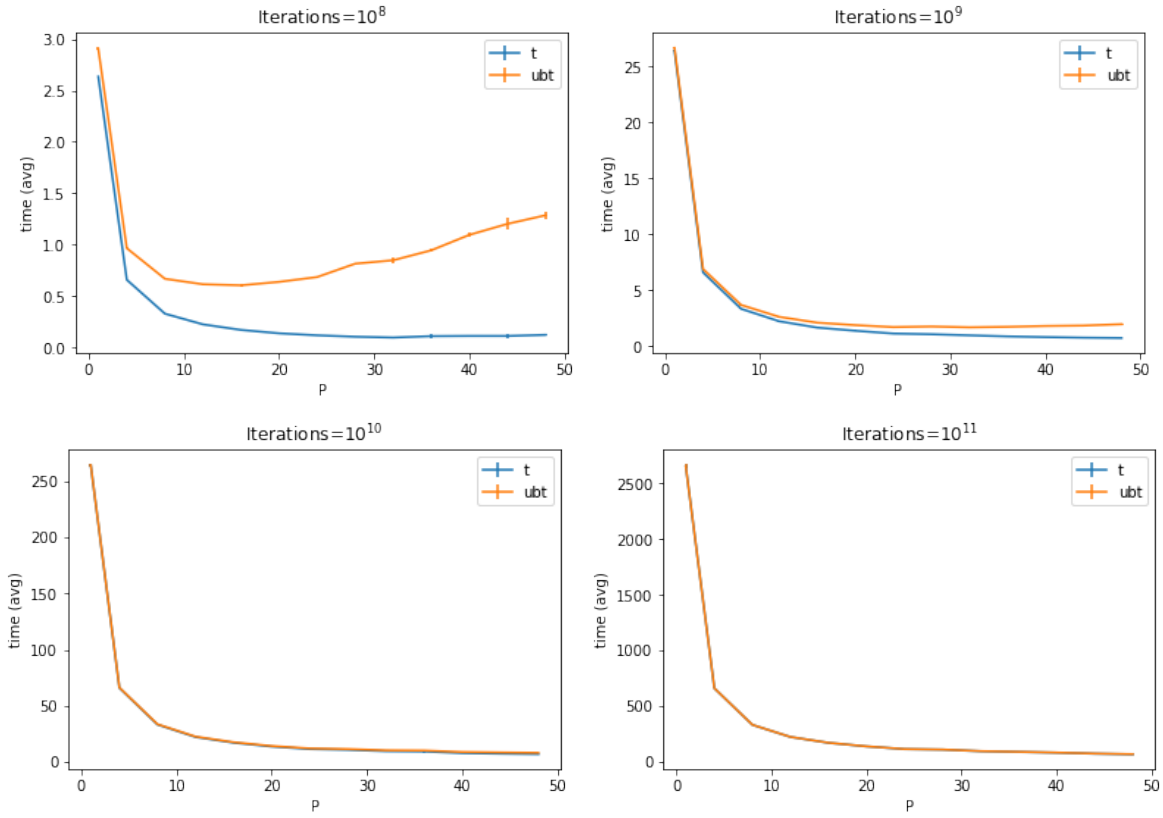

Figure 3: Plots of time versus processors. "t" is the time measured in code while "ubt" is measured by /usr/bin/time.

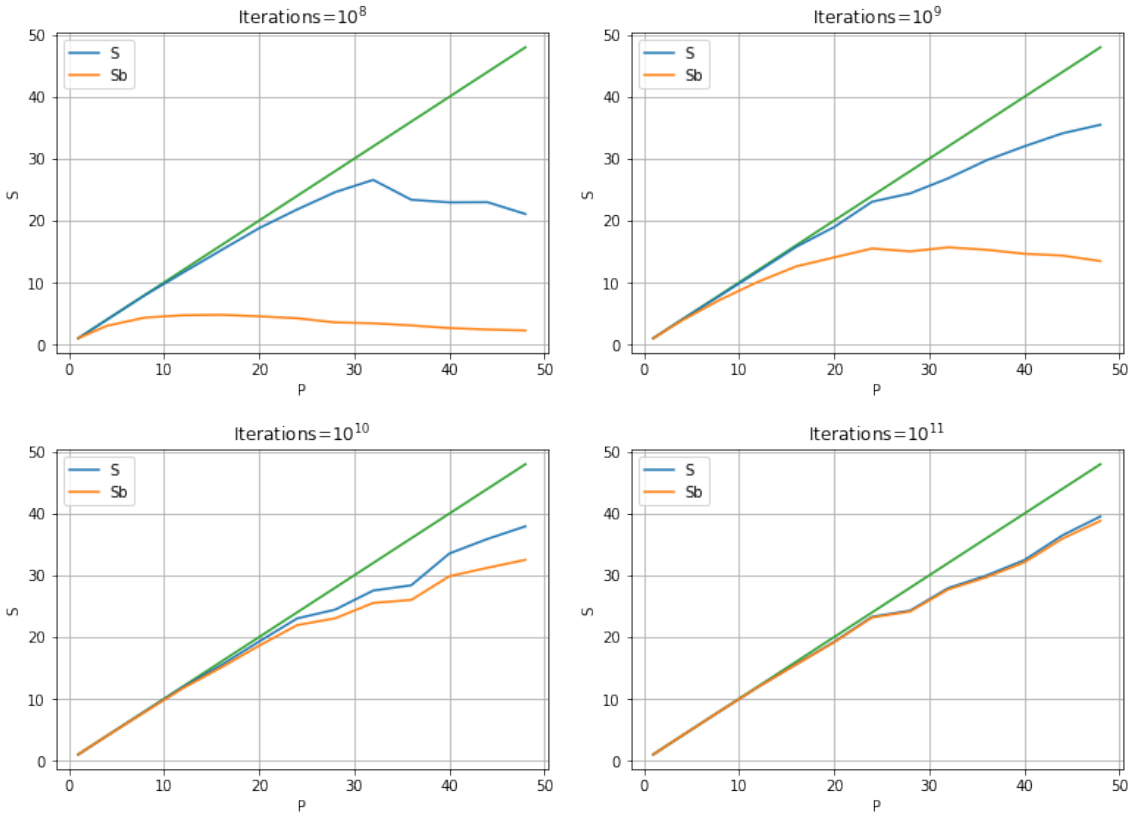

Figure 4: Plots of scalability versus processors. "S" is the scalability computed from the time measured in code while "Sb" from the time measured by /usr/bin/time.

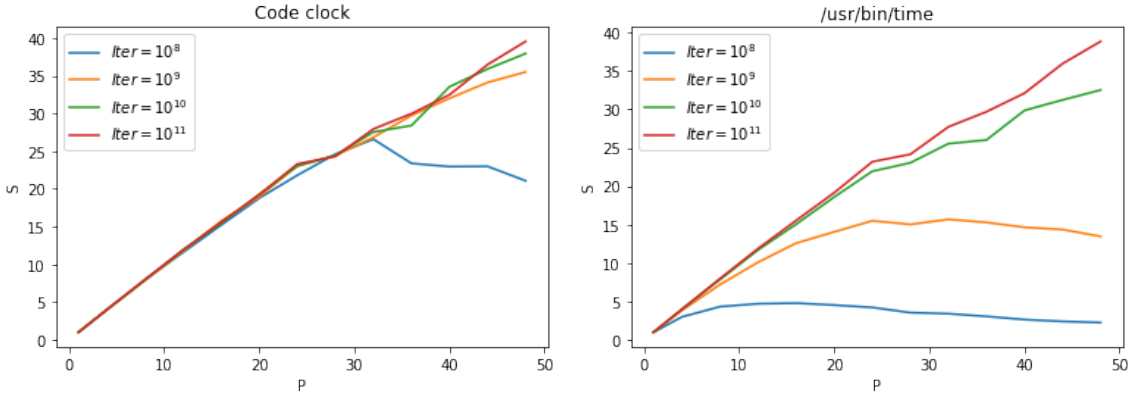

Figure 5: Plots of scalability versus processors for the four number of iterations, in the case of code clock and /usr/bin/time.

2.2 **Parallel overhead**

To estimate the parallel overhead of the mpi program in measuring of the strong scaling I compute for each number of processor used:

$$Parallel\,overhead = T(P) * P - T_{ser}$$

Here I consider $T_{ser} = T(P = 1)$. Then I repeat it for each number of iterations $(10^8, 10^9, 10^{10}, 10^{11})$. In Fig.6 I show the resulting parallel overhead for both the measured times. As expected the parallel overhead increase with the increasing of the iterations number, since in these cases the parallel part is bigger. The usr/bin/time have a bigger parallel overhead for the low iterations number while is almost the same for high iterations number. This is due to the fact that the usr/bin/time measure also the serial part of the code, this extra time is included in the parallel overhead in my very simple model to compute it.

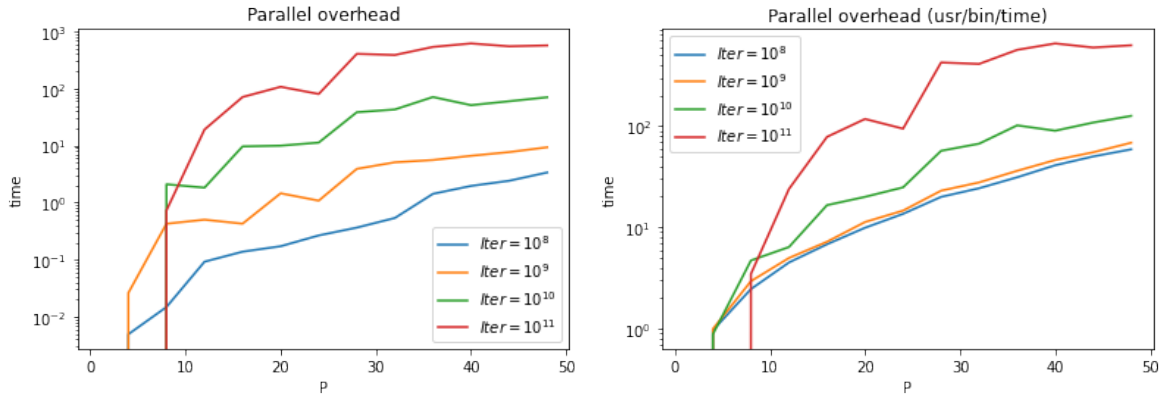

Figure 6: Plots of the parallel overhead time versus processors for the four number of iterations, in the case of code clock and /usr/bin/time.

2.3 **Weak Scalability**

Here I report the same plot done in the strong scalability tests. In Fig.7 the time versus the number of processor is shown for each number of iterations. If we had perfect weak scalability we would see the same time for each P, so a constant function. Since the computer is not perfect the time slightly increase with the number of processors. Here again for higher number of iterations the code clock and the usr/bin/time are going to coincide. In Fig.8 scalability versus the number of processor is shown and the perfect weak scalability is $S = 1$ for each P, as before increasing P we have a decreasing of S. Fig.9 show that the weak scaling remain the same whatever number of iterations is used (except for the usr/bin/time with $P \cdot 10^8$ iterations, that have in its measurement a significant serial part, as it is explained above).

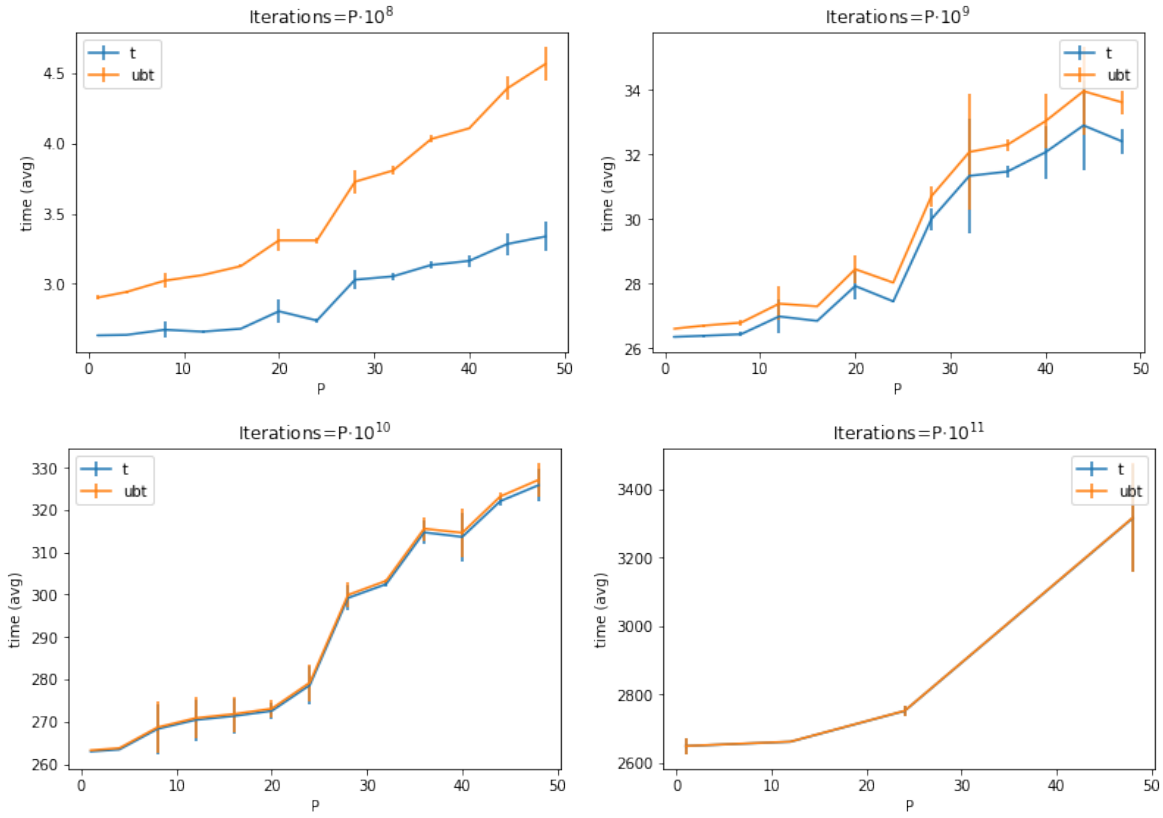

Figure 7: Plots of time versus processors. "t" is the time measured in code while "ubt" is measured by /usr/bin/time.

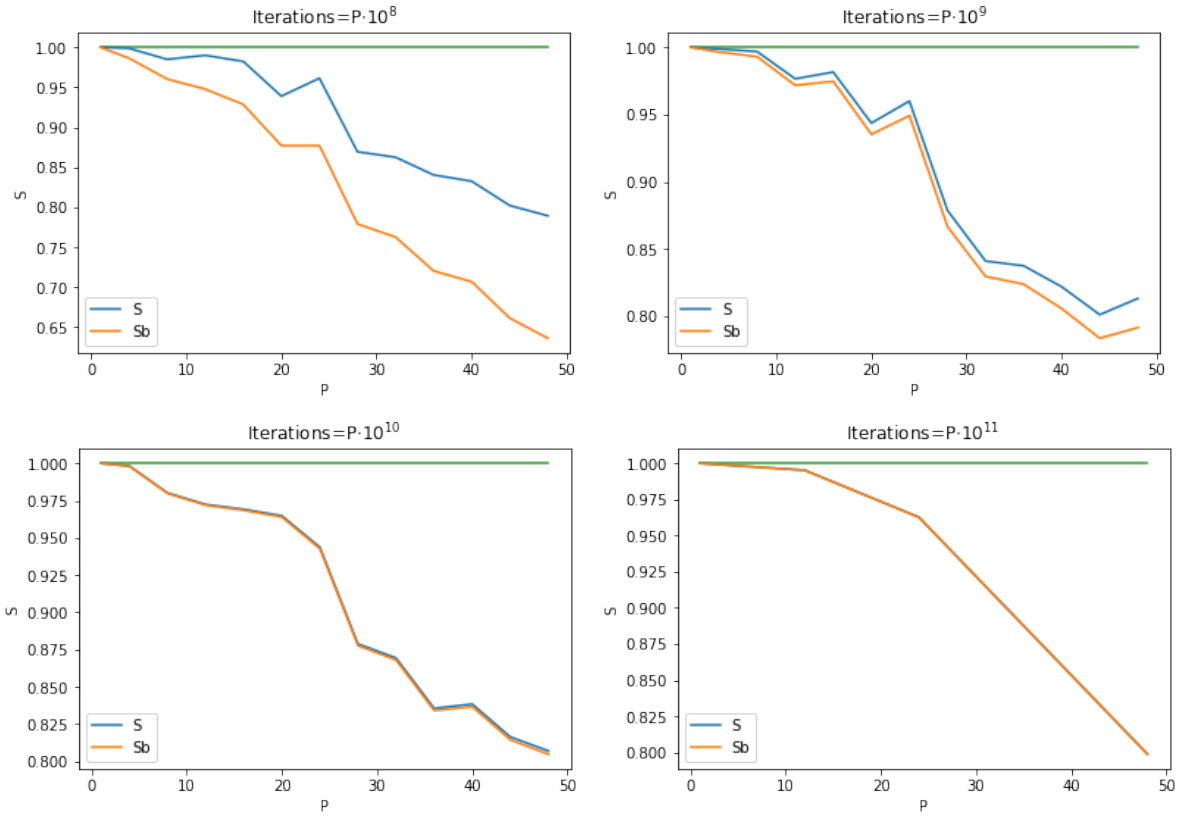

Figure 8: Plots of scalability versus processors. "S" is the scalability computed from the time measured in code while "Sb" from the time measured by /usr/bin/time.

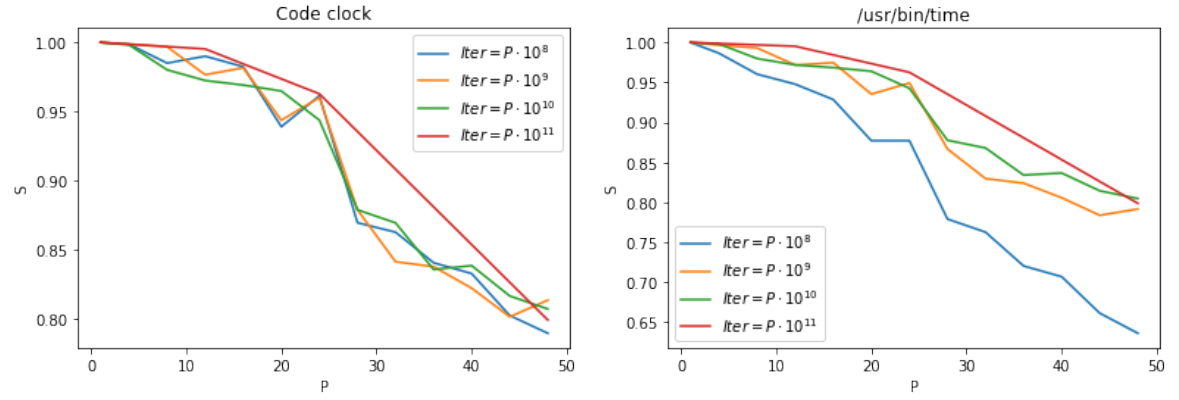

Figure 9: Plots of scalability versus processors for the four number of iterations, in the case of code clock and /usr/bin/time.