Report on strong and weak scalability of an HPC application

Nicole Orzan

January 17, 2018

The aim of this exercise was to compute strong and weak scalability of an application which provide a Monte-Carlo integration of a quarter of a unit circle to compute the number PI, which is obtained multiplying 4 times the computed area. The serial and the MPI version of the code were given.

Code Analysis

Serial Code

At first I had to determine the CPU time required to calculate PI with the serial code using 1000 iterations, that is 1000 random point generated inside the quarter of unit circle. Noticing that this number was too small to obtain a significative CPU-used time (the elapsed time resulting was always zero), I increased it to 10^8 iterations. I used the command /usr/bin/time to obtain the CPU-time required. I decided to perform in total 12 runs of which I took the time, and then to compute the mean time and the variance of this sample.
The obtained values are $x_{mean} = 1.97$s and $\sigma = 3.88$s

Parallel Code

In this exercise we had to study the parallel code establishing his scalability, which is intended as the capability of a system or a process to handle a growing amount of work. In the context of high performace computing there are two types of scaling: weak and strong. Strong scaling shows how the performance increases computing a fixed-size problem increasing the number of processors used to compute it. The speedup S is computed as:

$$S = \frac{T_1}{T_P} \tag{1}$$

where T_1 is the time needed to compute the problem using one processor, and T_P is the time needed to compute the problem using P parallel processors.

Weak scaling, instead, is computed defining how the computing time varies with the number of processors for a problem size fixed per processor. A program shows weak scalability if, as problem size and processors number grow, the speed of the operations per processors stays constant.

Strong scaling

To compute strong scalability, at first I runned the MPI code once fixing the size of the problem to the serial one (10^8 iterations) to observe the overhead given by the MPI implementation. I noticed a big overhead of nearly 1s.

At this point I started the strong scaling test. To calculate the speedup, I decided to fix the size of the problem to three different values, namely 10^7, 10^8 and 10^9 random numbers generated, in order to compare the results in the end; the number of processors used to compute the application was increased: I ran the MPI code for 2, 4, 8, 16 and 20 processors for each of the three fixed sizes. I performed 10 runs for each combination and I computed the mean values, which are shown in

N procs	S (10^7)	S (10^8)	S (10^9)
1	1.0	1.0	1.0
2	1.0464	1.3722	1.8001
4	1.0577	1.6753	3.1976
8	1.0076	1.8439	4.8694
16	0.9338	0.8279	6.5759
20	0.9130	0.7993	7.1024

Figure 1: Strong Scaling. Number of iterations, from the left to the right: 10^7, 10^8 10^9. You can find the tables of the computed errors in appendix A.

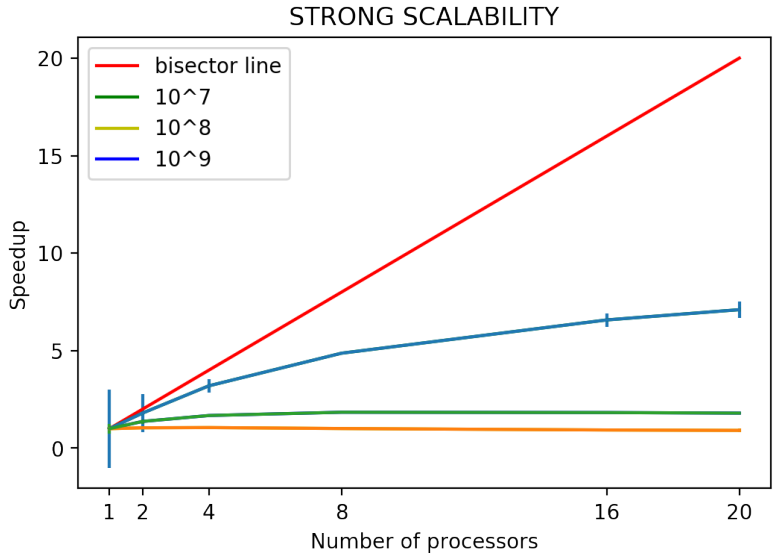

Figure 2: Strong scaling graph, showing the speedup versus the number of processors (with error bars, some are too small to be seen).

the table below (1).
From the graph in figure 2 we can see that increasing the number of iterations in the application, the speedup in closer and closer to the bisector line, which represent its ideal behavior.

In figure 3 it is represented the efficiency calculated form the speedup showed in figure 2.

Weak Scalability

To test the weak scalability of the application I had to increase the problem size together with the number of processors used. As before, I decided to use different sizes of the problem, namely 10^7, 10^8 and 10^9 random numbers generated.
We know that, increasing both the problem size and the number of processors, in an ideal case the value of the scalability has to remain the same. So plotting the elapsed time spent to compute the problem versus the number of processors, we expect to see a line parallel to the x axis. In the table 4 reported the values of the mean times obtained for 10^7, 10^8 and 10^9 random numbers generated for each number of processors used.

Weak or strong scaling?

Is this problem more appropriate for weak or strong scaling? To answer to this question we have to remember Amdhal's law and Gustafon's law. The first one says that the Speedup of our problem is limited by the serial fraction of it, and it is relevant only when the fraction of serial work is independent from the size of the problem (rare case). We have also to remember that theoretical Speedup isn't reachable in real problems.

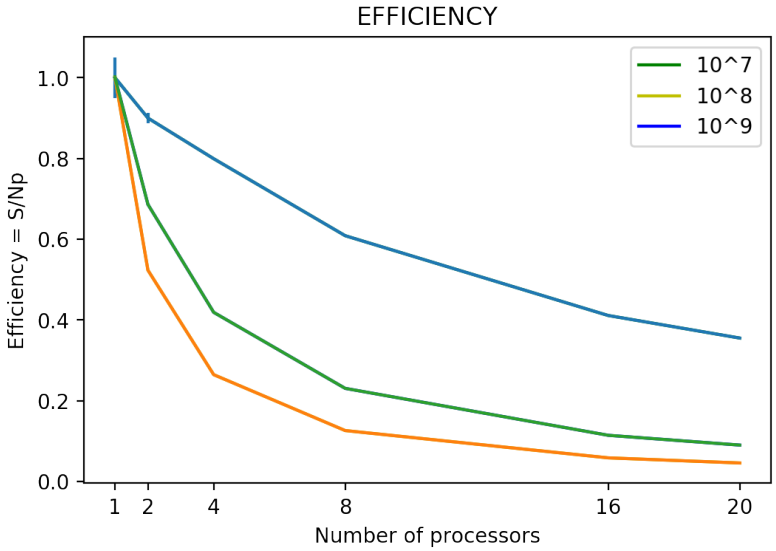

Figure 3: Strong scaling efficiency graph, with error bars (some are too small to be seen).

N procs	Elap (10^7)	Elap (10^8)	Elap (10^9)
1	1.6918	3.4436	21.0654
2	1.7027	3.5345	21.8818
4	1.7254	3.5854	21.9118
8	1.8100	3.7618	23.8981
16	0.9754	4.0827	26.1971
20	2.0445	4.1763	26.3681

Figure 4: Weak scaling. Number of iterations, from the left to the right: 10^7, 10^8 10^9. You can find the tables of the computed errors in appendix A.

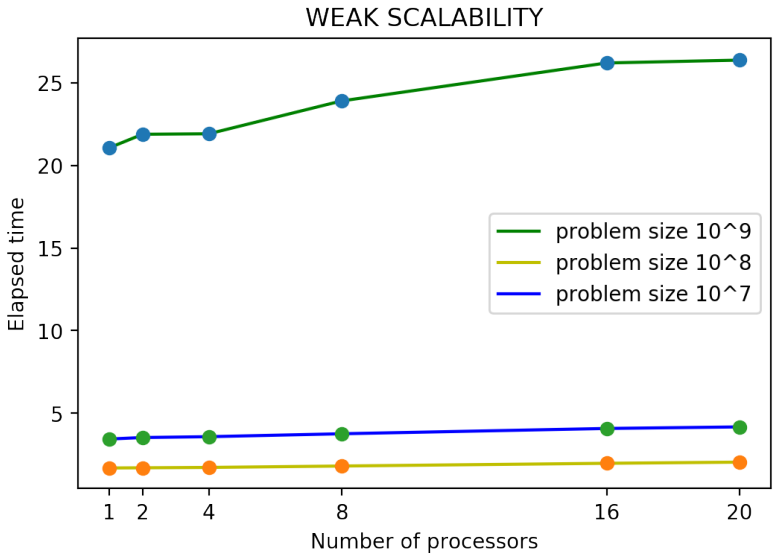

Figure 5: Weak scaling graph, showing the elapsed time versus the number of processors, with error bars (inside the points).

The second one, Gustafon's law, says instead that the proportion of the computations that aren't parallel decreases as the problem size increases.
We can see that our application shows a good strong scaling - above all for a big number of iterations (maybe also bigger that 10^9) - but also a quite good weak one, nevertheless we can clearly see from figure 5 that our problem doesn't have a really good weak scaling for a big number of trials.

Appendice

A Tables of errors

N procs	Elap (10^7)	Elap (10^8)	Elap (10^9)
1	0.0731	0.2500	2.0140
2	0.0615	0.1179	0.9900
4	0.0583	0.0410	0.3422
8	0.0704	0.0103	0.0401
16	0.0898	0.0125	0.3456
20	0.0998	0.0192	0.4172

Figure 6: Strong scaling, table of the errors computed as the standard deviation.

N procs	Elap (10^7)	Elap (10^8)	Elap (10^9)
1	0.0100	0.0099	0.0104
2	0.0100	0.0104	0.0312
4	0.0100	0.0135	0.0834
8	0.0098	0.0129	0.0876
16	0.0102	0.0111	0.1447
20	20.0101	0.0145	0.0688

Figure 7: Weak scaling, table of the errors computed as the standard deviation.