Exercise 1 - Scalability

Michela Venturini

November 20, 2018

Abstract

The target of the exercise is to compute strong and weak scalability of an HPC application; the application used in this trial is a Monte-Carlo integration that compute the number PI, provided in two different ways: one is a basic implementation of the algorithm while the second one has a parallel MPI implementation.

1 Serial efficiency

The first part of the exercice consists of measuring the time taken by both the serial program and the parallel one (using only one processor) to run; To compare them and estimate the overhead of the parallel execution with respect to the serial one we use the Serial efficiency given by:

$$E(N) = \frac{T_{best}}{T(N,1)} \tag{1}$$

where N is the size of the problem, T_{best} is the running time achieved using serial implementation and $T(N,1)$ is the the running time achieved running the parallel application using only one MPI process.

Serial execution We compile the serial implementation of the algorithm by using gcc compiler and we execute it using the /usr/bin/time command to time the application. Fort the purpose of the exercise we consider only the user time which is the time CPU actually spent running the program .

```
> gcc pi.c -o serial_pi
> time ./serial_pi 1000000000
```

Parallel execution We compile and run the parallel implementation of the program using OpenMPI by means `mpicc` and `mpirun -np 1` that specifiy the number of MPI processes (again using time command. The table 1 shows the two results by using $N = 10^9$ where N is the number of iterations of the algorithm and the size of the problem.

```
> mpicc pi.c -o parallel_pi
> time mpirun -np 1 parallel_pi 1000000000
```

Application	User time(s)
Serial	19.786
Parallel	20.787

Table 1: Serial and Parallel application

Using equation 1 I obtained $E(N) = 0.952 < 1$. This value is lower than one thus $T(N,1) > T_{best}$. This is due to the overhead of OpenMPI, but the impact of this factor should decrease as the number of MPI processes increase.

2 Scalability

Scalability indicates how efficient an application is when using increasing numbers of parallel processing elements. In HPC systems scalability has two different flavours:

- Strong scalability that measure the performance of a system to handle a fixed amount of work with growing number of processors; The goal in this case is to find a tradeoff that allows the computation to complete in a reasonable amount of time, yet does not waste too many cycles due to parallel overhead. In strong scaling, a program is considered to scale linearly if the speedup (in terms of work units completed per unit time) is equal to the number of processing elements used. In general, it is harder to achieve good strong-scaling at larger process counts since the communication overhead for many/most algorithms increases in proportion to the number of processes used.
- Weak scalability that measure the performance of a system to handle a growing amount of work with a growing number of processors. This type of measurement is generally used for programs that take a lot of memory or other system resources (es. memory-bound). In the case of weak scaling, linear scaling is achieved if the run time stays constant while the workload is increased in direct proportion to the number of processors.

2.1 Strong scalability We compute strong scalability of the parallel application on Ulysses as follow:

1. compile the application using mpicc
2. request interactively two nodes that allows us to use a greater number of processors taking into account the higher overhead into communication
   ```
   > qsub -l nodes=2:ppn=20,walltime=00:30:00 -I
   ```

3. run a bash script to obtain measures of the running time for different number of processors.
   The core part of my script is composed by a for cycle running the program keeping the size problem constant and changing the number of MPI processes as follow:

```
for procs in 1 2 4 8 16 32; do
 time mpirun -np $procs pi_parallel $((1000000000/$procs))
done
```

From the output of the executions we consider the *walltime* that indicates the time taken by the masternode to execute the entire application in parallel. The figure 1 shows walltime versus number of nodes.
It can be seen how the execution time decreases increasing the number of MPI processes

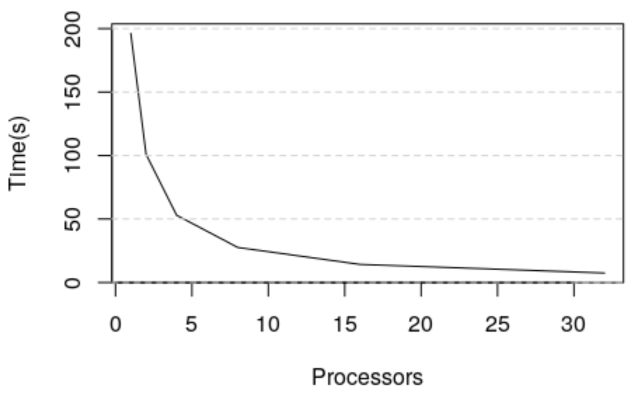

Figure 1: Strong Scalability

and keeping the size of the problem constant. Moreover in the figure 2 can be noticed that the total amount of time to run the program using different number of MPI processes is not constant but slightly increase up to 22% using 32 MPI processes. This fact is mainly due to:

1. serial sections running time that not scales with MPI processes and it is summed up for every one of them
2. communication overhead that arise increasing number of MPI processes
3. load balance that means the processes to ensure that resources are used effectively

Speed-Up and Amdahl's Law To evaluate the strong scalability we use the Amdahl's Law which defines the Speed-Up as the time taken to run the program in serial over the

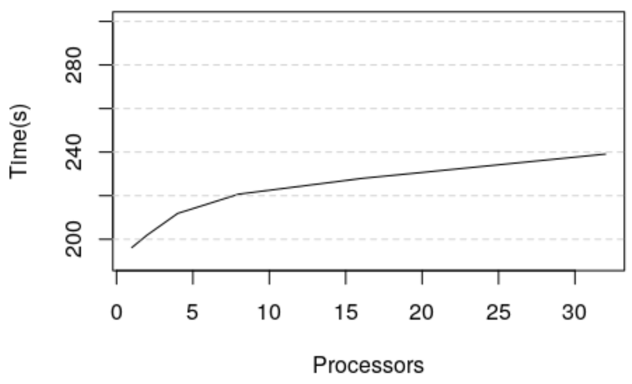

Figure 2: T(N,P)*P vs Processors

time to run the program in parallel for a fixed size problem.

$$S(N) = \frac{T(N,1)}{T(N,P)} \tag{2}$$

According to the Amdahl's Law the maximum speedup for $\lim_{P \to \infty}$ is bounded by the tame taken to execute the serial part of the problem. In particular:

$$S(N) = \frac{T(N,1)}{T(N,P)} = \frac{1}{s + \frac{1-s}{P}} \tag{3}$$

$$\lim_{P \to \infty} \frac{T(N,1)}{T(N,P)} = \frac{1}{s + \frac{1-s}{P}} = \frac{1}{s} \tag{4}$$

From the figure 3 we can notice that the actual Speed-up increases in a sublinear way (due to Amdahl's law) but it scales well.

2.2 Weak scalability In order to compute weak scalability we increase the problem size simultaneously with the number of MPI processes being used. In the ideal case this would imply that running time remains constant through all the executions.Also in this case we a bash script whose core part is:

```
for procs in 1 2 4 8 16 32; do
 time mpirun -np $procs pi_parallel 1000000000
done
```

Can be deduced that at every iteration the size of the problem double as the number of MPI processes. The figure 4 shows the walltime versus the number of MPI processes.

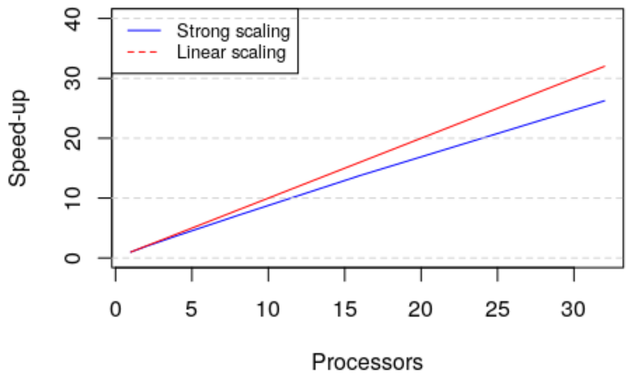

Figure 3: Speed-Up

The ideal case should attempt a constant run, while the real case shows a maximum increment in time of 16%. This is due to serial sections that, instead of the former case, is not constant and increases with the size of the problem, communication overhead and load balance, as for the case of strong scalabiliy.

3 Comparison between Strong and weak scaling

To compare Strong and weak scaling we use the Parallel efficiency that measure how effectively a given resource, as CPU computational power, can be used in a parallel program. In the figure 5 is shown the Parallel efficiency, defined for strong scaling:

$$P(N) = \frac{T(N,1)}{P * T(N,P)} \tag{5}$$

and for weak scaling:

$$P(N) = \frac{T(N,1)}{T(N,P)} \tag{6}$$

What we can deduce is that for this application the weak scaling works slightly better than weak scaling: the parallel efficiency in the former case is nearer to the ideal case and decreases more slowly increasing number of processors.

Another important consideration is that the Monte-Carlo approach to calculate the value of PI is statistical and the precision of the output strictly depends on the size of the problem (that is number of iterations - N). I reckon this fact let this problem be more appropriate for weak scaling that allows to exploit resources by increasing workload with resources in the same amount of time.

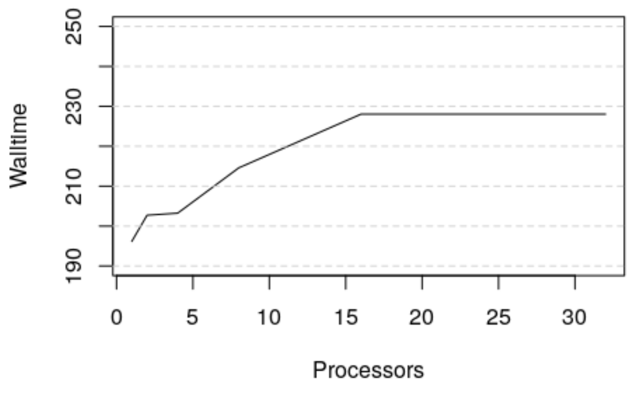

Figure 4: Weak Scalability

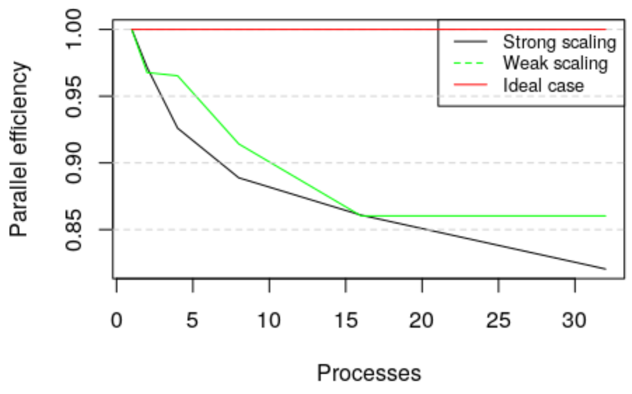

Figure 5: Parallel Efficiency