Canepa Andrea - s4076249
High Performance Computing
Parallel Computing - Mandelbrot Set

Mandelbrot Set

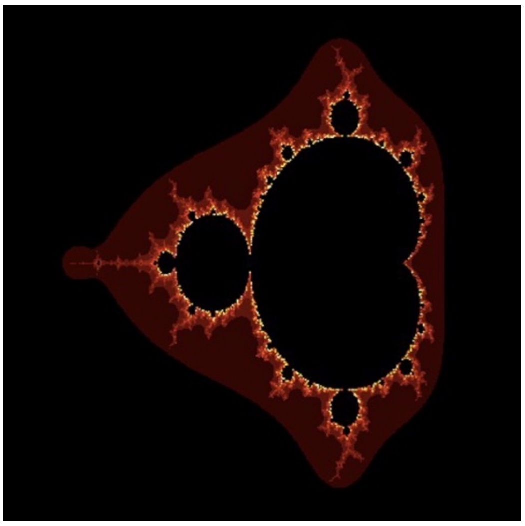
The Mandelbrot Set is the set of complex numbers c for which the function

$$f_c(z) = z^2 + c$$

does not diverge when iterated from z = 0, i.e., for which the sequence

$$f_c(0),\ f_c(f_c(0)),\ \text{etc}$$

remains bounded in absolute value.



Hardware & Tools

All the code is written in C++ and all the experiments and measurements are performed via ssh on **Cheetah**, a cluster of 18 dual-core nodes.

OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C++. It consists of a set of library routines, compiler directives and environment variables that influence run-time behavior.
The programmers have a simple and flexible for developing parallel applications.

MPI (Message Passing Interface) a message-passing application programmer interface.
which is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes.
MPI's goals are high performance, scalability, and portability.

The problem

We’re going to compute the Mandelbrot Set in both sequential and parallel way , doing a comparison between performances and computing the **speedup**.

As the execution time is an inverse index (lower value means better performance), the speedup of the parallel version respect to the sequential one, is computed in the following way:

$$Speedup = \frac{execution\ time_{sequential\_version}}{execution\ time_{parallel\_version}}$$

The speedup is computed taking the best execution times (each problem is executed **7 times**, and the **minimum** value is taken) for each task size (the numbers of rows to compute given to each worker) because our goal is to find the task size which causes the best speedup.

All the measurements are repeated for three different output matrices (which indicate the ”problem size”):

- 1450 x 1350 : small size
- 2000 x 1700 : medium size
- 4500 x 3000 : large size

We distinguish (and obviously compare) two different implementations of the parallel version:

- MPI and OpenMP running on 18 nodes (dual-core)
- MPI running on 36 nodes (cores)

Having n nodes (in the parallel version), one will be the dispatcher and the other n-1 nodes the workers.

All the times are expressed in seconds.

Sequential version

For each output matrix size the program is executed 7 times, in order to take the best performance value (the minimum – highlighted in green in the following table).
These "green-values" will be used to compute the speedup.

Small Size (1450x1350)	Medium Size (2000x1700)	Large Size (4500x300)
58.4564	129.828	567.697
58.4355	129.952	568.174
58.4376	129.91	567.549
58.4461	130.111	567.807
58.4319	130.081	567.201
58.4232	129.874	567.485
58.3892	129.778	567.5

Looking at the table you can see how the computational time increases, increasing the size of the problem.

Files and instructions

Compilation and execution of the program is done by the "***script_sequential.sh***" script.

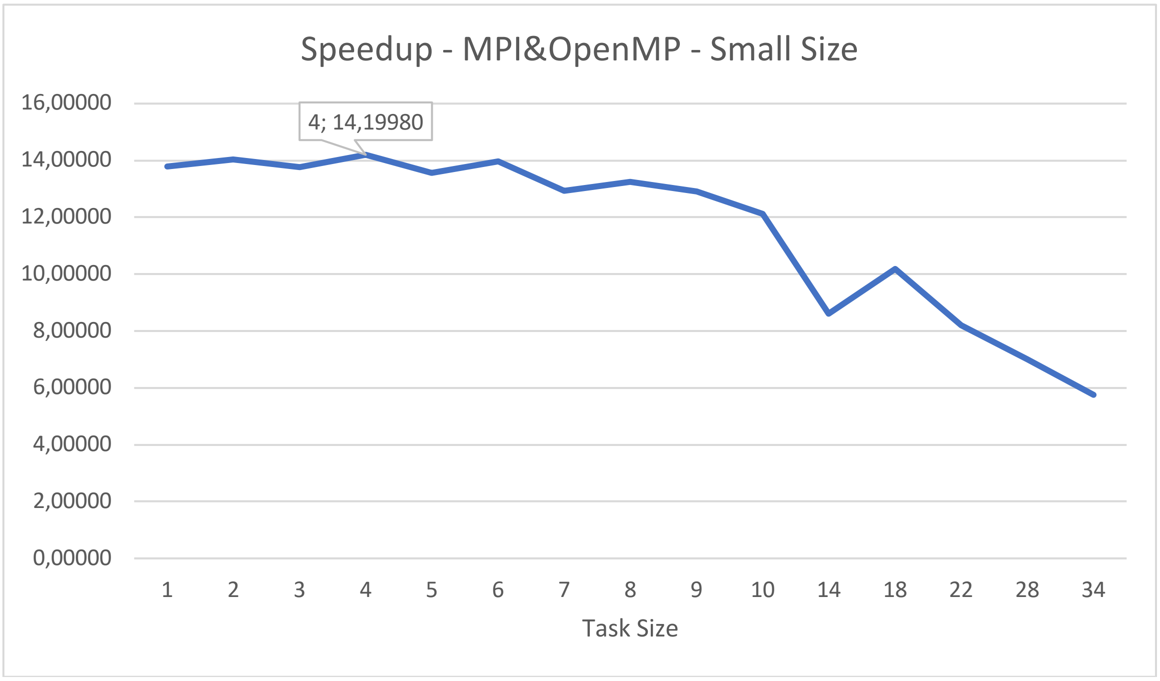
The executable takes the following parameters:
- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

The file "**sequential.dat**" (generated by the script) contains the values shown in the above table.
"**mandelbrot_seq_plot.gp**" is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 92 of the code contained in "**mandelbrot_seq.cpp**").

MPI + OpenMP version

Small Size (1450x1350)	
Task Size	**Execution Time**
1	4.23726
2	4.15787
3	4.24457
4	4.11197
5	4.30335
6	4.17918
7	4.51148
8	4.40812
9	4.40812
10	4.81409
14	6.78013
18	5.73665
22	7.11183
28	8.34425
34	10.148

Small Size (1450x1350)	
Task Size	**Speedup**
1	13.7799
2	14.0431
3	13.7562
4	14.1998
5	13.5683
6	13.9715
7	12.9423
8	13.2458
9	12.9134
10	12.1288
14	8.61181
18	10.1783
22	8.21015
28	6.99754
34	5.75378



Files and instructions

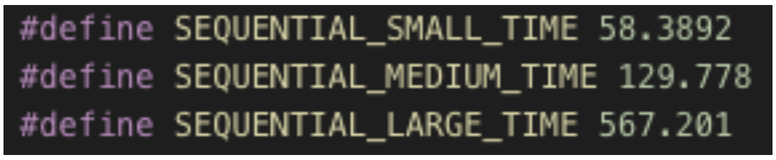
Compilation and execution of the program is done by the "***script_parallel_s.sh***" script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

The file "**parallel_s.dat**" (generated by the script) contains the values shown in the above tables. "**mandelbrot_par_plot.gp**" is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 150 of the code contained in "**mandelbrot_par.cpp**") and need to be modified writing which problem size we want to print.

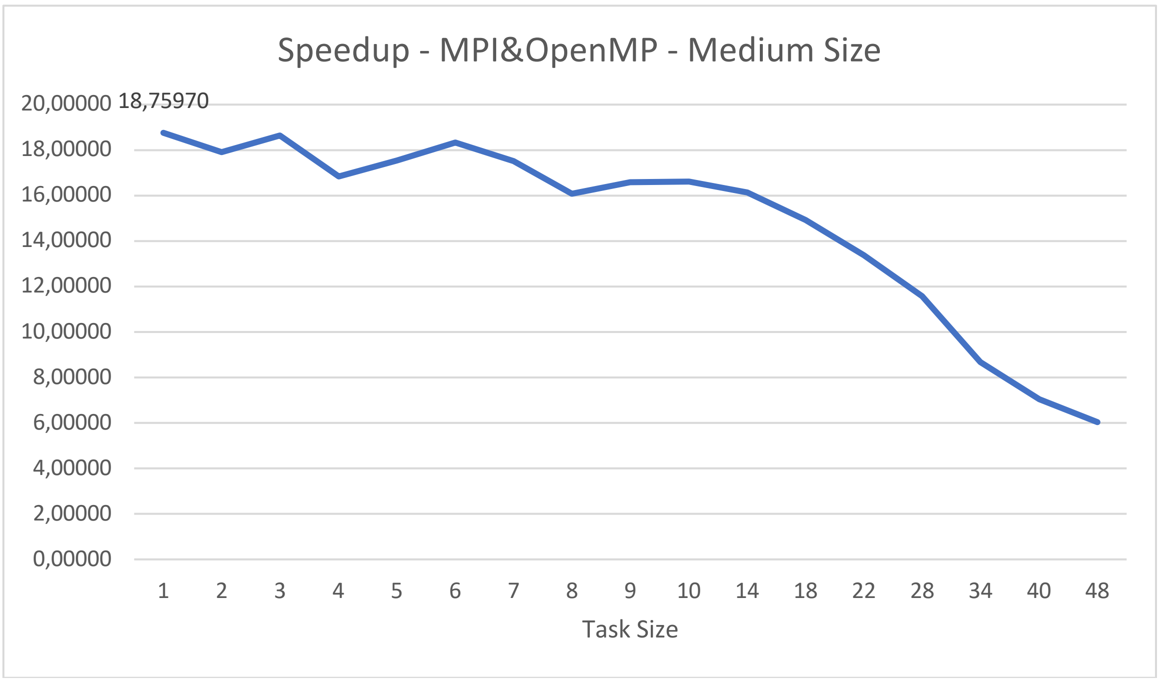
Before launching the script, the line 148 of the mandelbrot_par.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).



```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

Medium Size (2000x1700)	
Task Size	**Execution Time**
1	6.91793
2	7.25028
3	6.96561
4	7.70836
5	7.39684
6	7.07826
7	7.41324
8	8.06715
9	7.81918
10	7.81543
14	8.04463
18	8.69684
22	9.70362
28	11.2035
34	14.953
40	18.4166
48	21.5028

Medium Size (2000x1700)	
Task Size	**Speedup**
1	18.7597
2	17.8997
3	18.6312
4	16.836
5	17.5451
6	18.3347
7	17.5062
8	16.0872
9	16.5974
10	16.6054
14	16.1323
18	14.9224
22	13.3742
28	11.5837
34	8.67905
40	7.0468
48	6.03541



Files and instructions

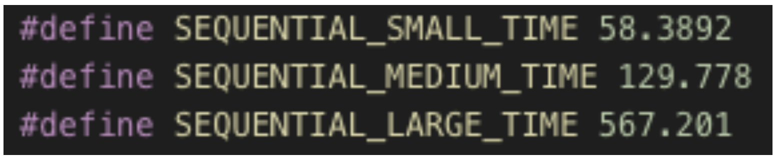
Compilation and execution of the program is done by the "***script_parallel_m.sh***" script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

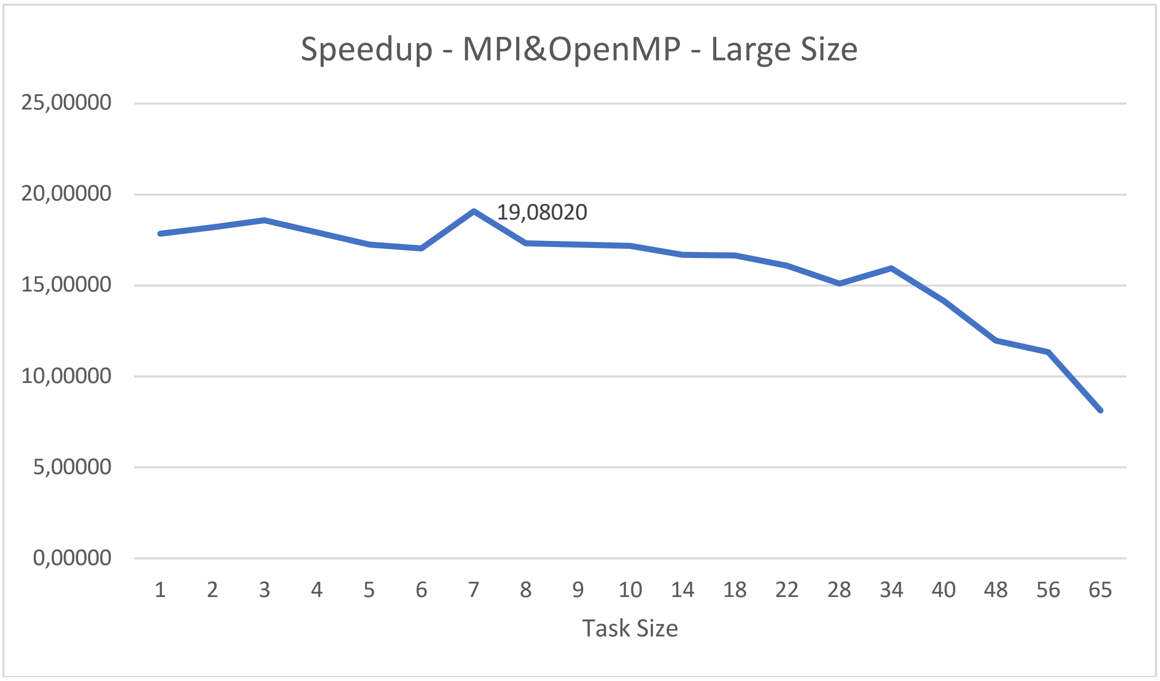
The file "**parallel_m.dat**" (generated by the script) contains the values shown in the above tables. "**mandelbrot_par_plot.gp**" is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 150 of the code contained in "**mandelbrot_par.cpp**") and need to be modified writing which problem size we want to print.

Before launching the script, the line 148 of the mandelbrot_par.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).


```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

Large Size (4500x3300)	
Task Size	**Execution Time**
1	31.7734
2	31.1444
3	30.504
4	31.6672
5	32.8619
6	33.318
7	29.7273
8	32.7528
9	32.9051
10	32.9889
14	33.998
18	34.0822
22	35.2918
28	37.5541
34	35.5776
40	40.1235
48	47.4084
56	50.1029
65	69.7642

Large Size (4500x3300)	
Task Size	**Speedup**
1	17.8514
2	18.212
3	18.5943
4	17.9113
5	17.2601
6	17.0238
7	19.0802
8	17.3176
9	17.2375
10	17.1937
14	16.6834
18	16.6421
22	16.0717
28	15.1036
34	15.9426
40	14.1364
48	11.9641
56	11.3207
65	8.13026



Files and instructions

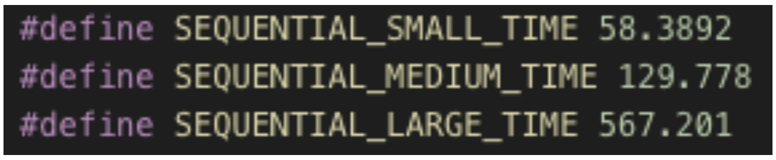
Compilation and execution of the program is done by the "***script_parallel_l.sh***" script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

The file "**parallel_l.dat**" (generated by the script) contains the values shown in the above tables. "**mandelbrot_par_plot.gp**" is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 150 of the code contained in "**mandelbrot_par.cpp**") and need to be modified writing which problem size we want to print.

Before launching the script, the line 148 of the mandelbrot_par.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).

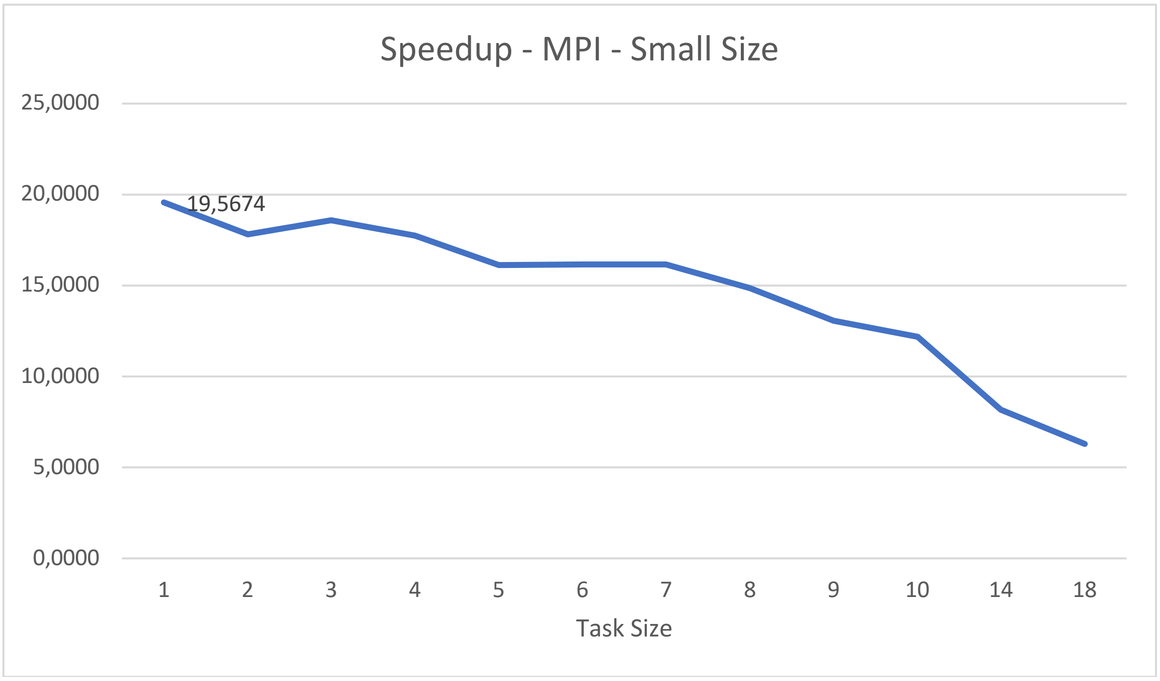


```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

MPI version

Small Size (1450x1350)	
Task Size	**Execution Time**
1	2.98401
2	3.27633
3	3.13932
4	3.29299
5	3.61935
6	3.61326
7	3.61237
8	3.92653
9	4.46825
10	4.79844
14	7.1314
18	9.28139

Small Size (1450x1350)	
Task Size	**Speedup**
1	19.5674
2	17.8215
3	18.5993
4	17.7314
5	16.1325
6	16.1597
7	16.1637
8	14.8704
9	13.0676
10	12.1684
14	8.18762
18	6.29099



Files and instructions

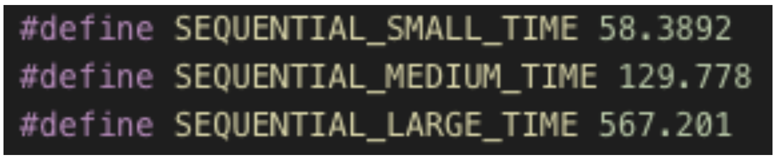
Compilation and execution of the program is done by the “***script_mpi_s.sh***” script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

The file “**mpi_s.dat**” (generated by the script) contains the values shown in the above tables. “**mandelbrot_par_plot.gp**” is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 149 of the code contained in “**mandelbrot_mpi.cpp**”) and need to be modified writing which problem size we want to print.

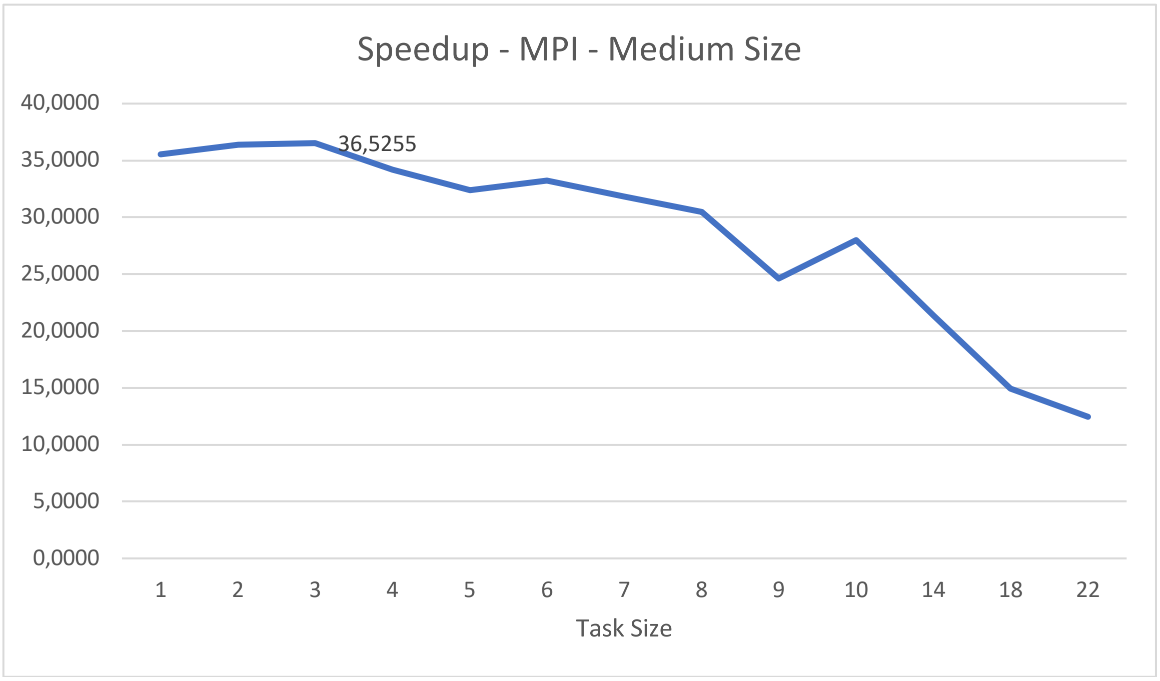
Before launching the script, the line 147 of the mandelbrot_mpi.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).



```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

Medium Size (2000x1700)	
Task Size	**Execution Time**
1	3.65083
2	3.56901
3	3.55308
4	3.79555
5	4.00774
6	3.90234
7	4.0756
8	4.2553
9	5.26667
10	4.6315
14	6.07715
18	8.68052
22	10.4211

Medium Size (2000x1700)	
Task Size	**Speedup**
1	35.5475
2	36.3625
3	36.5255
4	34.1922
5	32.3818
6	33.2565
7	31.8426
8	30.498
9	24.6414
10	28.0207
14	21.3551
18	14.9505
22	12.4533



Files and instructions

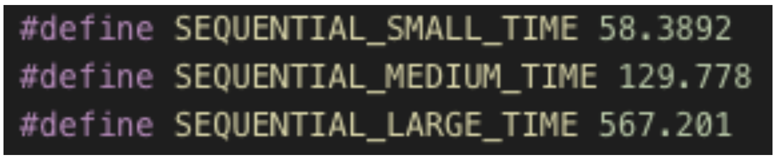
Compilation and execution of the program is done by the “***script_mpi_m.sh***” script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

The file “**mpi_m.dat**” (generated by the script) contains the values shown in the above tables. “**mandelbrot_par_plot.gp**” is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 149 of the code contained in “**mandelbrot_mpi.cpp**”) and need to be modified writing which problem size we want to print.

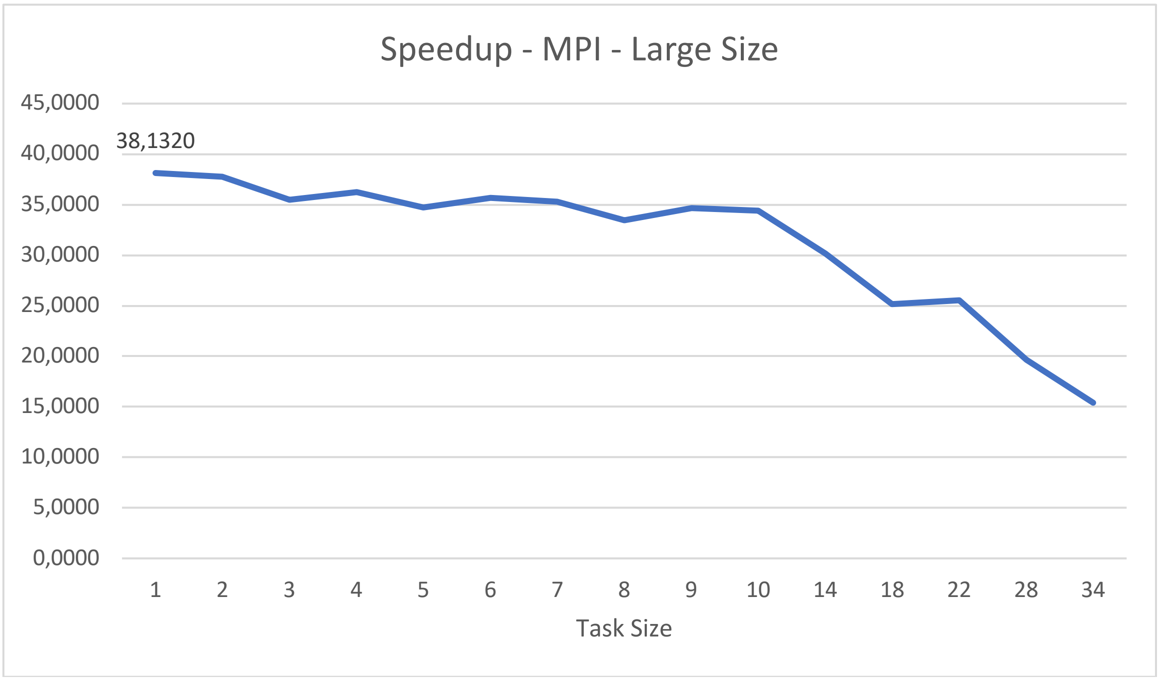
Before launching the script, the line 147 of the mandelbrot_mpi.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).



```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

Large Size (4500x3300)	
Task Size	**Execution Time**
1	14.8747
2	15.0266
3	15.9945
4	15.65
5	16.3267
6	15.9104
7	16.0664
8	16.9443
9	16.361
10	16.485
14	18.8008
18	22.561
22	22.2167
28	28.8584
34	36.8271

Large Size (4500x3300)	
Task Size	**Speedup**
1	38.132
2	37.7465
3	35.4623
4	36.2428
5	34.7407
6	35.6497
7	35.3035
8	33.4744
9	34.6679
10	34.4071
14	30.1689
18	25.1408
22	25.5304
28	19.6546
34	15.4017



Files and instructions

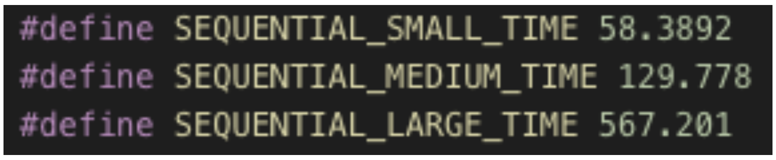
Compilation and execution of the program is done by the “***script_mpi_l.sh***” script.

The executable takes the following parameters (a part from the number of nodes and the nodelist):

- The length of the x-axys
- The length of the y-axys
- The real-part of the starting complex number
- The imaginary-part of the starting complex number
- The length of the region we want to compute
- The task size
- The .dat file where the output matrix will be printed (in order to call *gnuplot* on it).

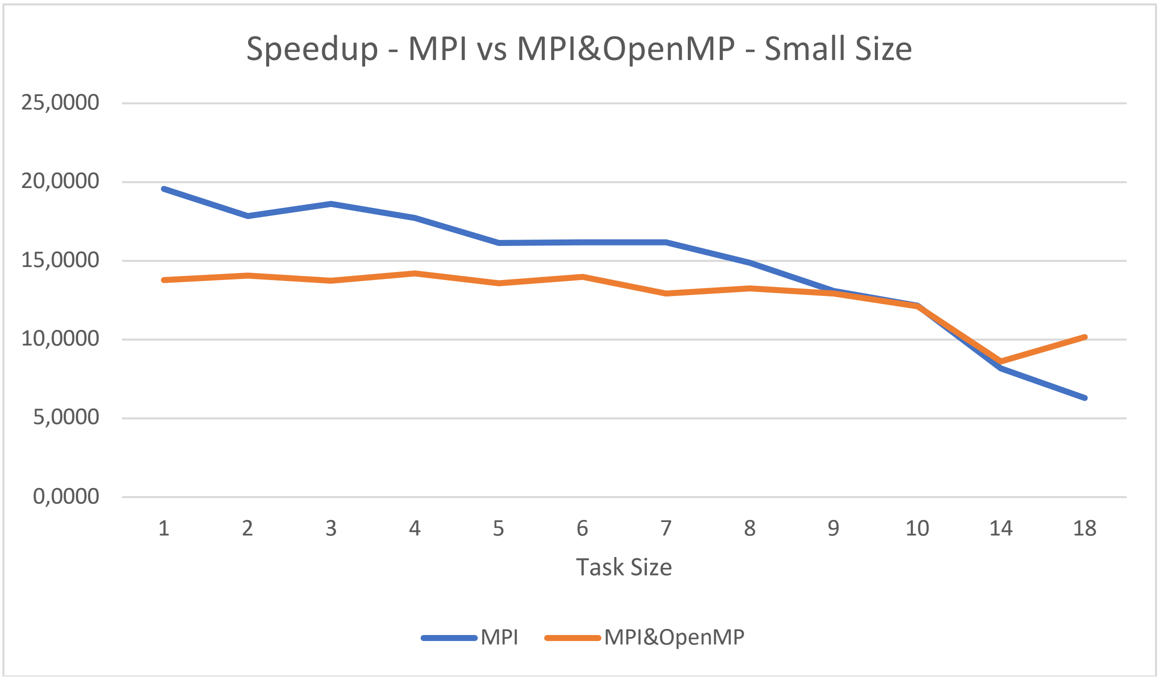
The file “**mpi_l.dat**” (generated by the script) contains the values shown in the above tables. “**mandelbrot_par_plot.gp**” is responsible for the matrix plot (which has been disabled by commenting the *printMatrixToFile* function call at line 149 of the code contained in “**mandelbrot_mpi.cpp**”) and need to be modified writing which problem size we want to print.

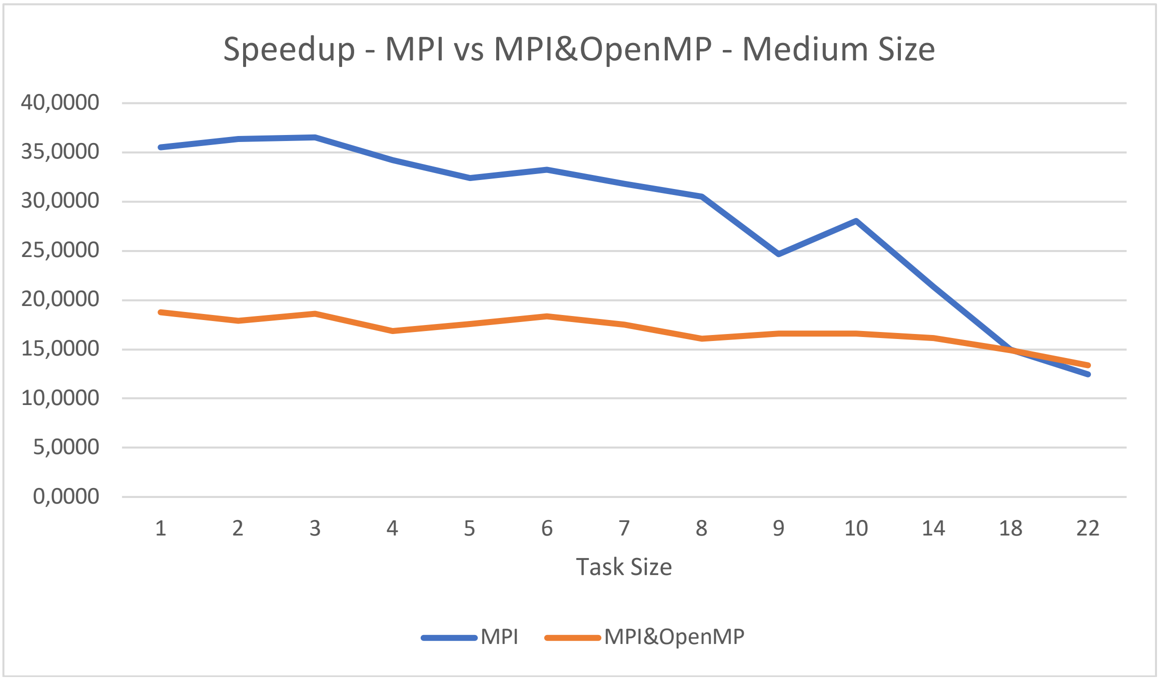
Before launching the script, the line 147 of the mandelbrot_mpi.cpp code need to be modified writing which sequential value must be used to compute the speedup (depending on the problem size).

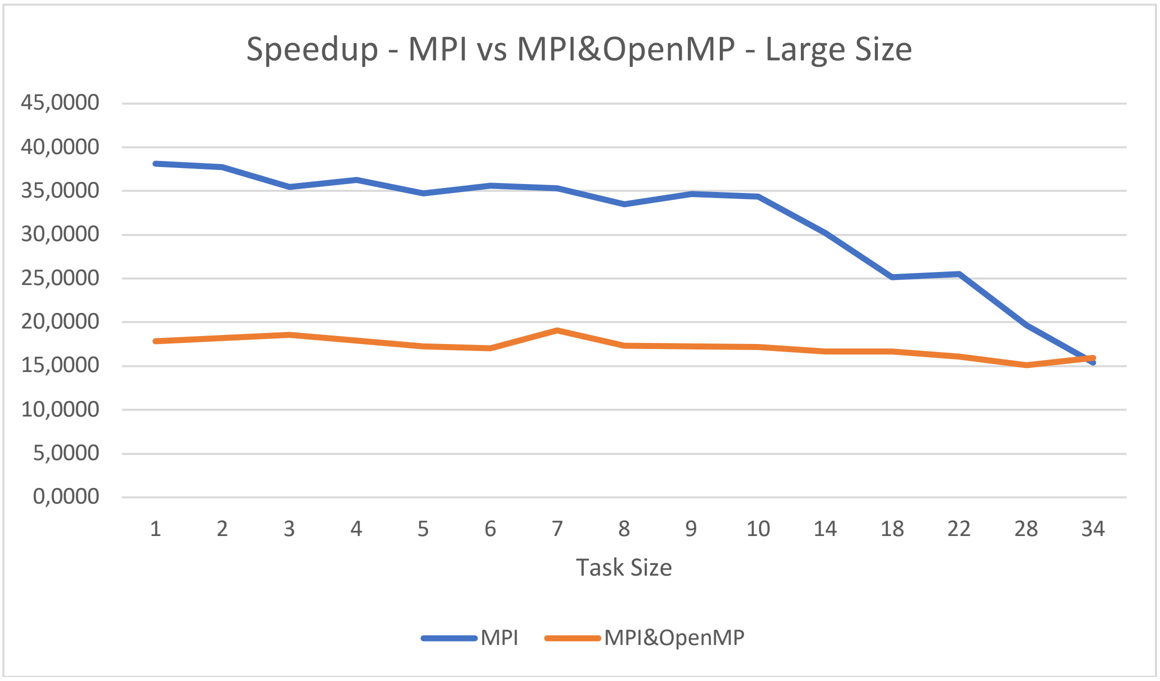


```
#define SEQUENTIAL_SMALL_TIME 58.3892
#define SEQUENTIAL_MEDIUM_TIME 129.778
#define SEQUENTIAL_LARGE_TIME 567.201
```

Comparison between MPI and MPI&OpenMP







Conclusions

Analyzing the results obtained, it is easy to understand how parallelization can significantly reduce computational time.

From the experiments, it turned out that a small task in parallel versions implies better performance and consequently a greater speedup.

It is important to underline that every task in this problem can have a computational cost totally different from the others and therefore, probably assigning so many to each worker you risk that someone has to deal with many expensive tasks, slowing down the termination of the program.

The parallel version that uses only MPI has been found to perform better than the one that also uses OpenMP. This is probably due to overhead reasons.