Practical Assignment #3 - Parallel Computing 2022/2023

Simão Cunha
Informatics Department
University of Minho
Vila Nova de Famalicão, Braga, Portugal
a93262@alunos.uminho.pt

Tiago Silva
Informatics Department
University of Minho
Vila Nova de Famalicão, Braga, Portugal
a93277@alunos.uminho.pt

***Abstract*—This report serves to document the third and the last phase of Parallel Computing course at University of Minho project. It begins with a small introduction of the subject of this assignment. Then we explain some theoretical differences between distributed and shared memory, followed by the implementation in MPI and in a mixture with MPI and OpenMP of the k-means algorithm developed in the previous phases. After that we will show the results obtained in the SEARCH and in the PC's of the authors of this project, as well as some critical analysis. We will finalize with the conclusions got from this assignment.**

***Index Terms*—K-Means, MPI, OpenMP, shared memory, distributed memory, SeARCH**

I. Introduction

The purpose of this assignment is to develop a version of the k-means algorithm from the code written in previous phases that must explore parallelism efficiently, but this time we should use alternative programming environment to reduce the execution time.

II. Distributed memory and shared memory

A. Distributed memory

Distributed memory is a type of memory architecture where the memory is shared between several computation nodes in a network. It allows the nodes to access memory from the other nodes - as well as its own -, increasing the total capacity of the available memory and allowing all the nodes to work together to solve bigger problems. In this course, we learned to program with distributed memory using MPI (Message Passing Interface).

B. Shared memory

In this architecture, all the processors share the same space of the physical memory. It allows any processor to access anywhere in the memory, but requires synchronization mechanisms to guarantee the consistence of the data. The communication between the processors is usually done through synchronization barriers and shared variables. In this course, we learned to program with shared memory using OpenMP.

III. Implementation

In the previous phase, we used OpenMP to develop a paralleled version of the algorithm k-means in C, which makes use of the shared memory betweem different cores. In this phase, we decided to use MPI, which gaves an interface to the process communication, such as sending and receiving messages, synchonization operations and processes group management.

A. MPI

Our start point is getting the best sequential version of the algorithm done in phase one. As seen in the report of that phase, our best version was the one using loop unrolling with 4 instructions per cycle. However, we believed that the implementation in MPI of that version would be very difficult to read and, because of that, we decided to use the best version without loop unrolling, which was the one without structs.

The next step is deciding the distribution of the workload to all the processes. Since all the processes must do computation in all arrays, they must have all the same in the beginning of the program because they will work in the same sample data. So, the filling of the arrays with the samples will be in charge of the main process and he must communicate those arrays to the children processes - we used the function `MPI_Bcast` - consequently, the way that the children processes will receive that information is with the use of the same function.

At this point, all the children processes will do something with those arrays and, according with the k-means algorithm, they will update the clusters until they converge. So, each process will operate in equal portions of the sample data and calculate the `sumX`, `sumY` and `clusters_npoints` arrays and the `converged` variable - use of the function `MPI_Recv` in the reception of the messages and `MPI_Send` in the sending by each child process. After that, they will communicate them to the main process which will join all the calculus from all the processes and compute the new centroids.

Last but not the least, the main process will print in the *stdout* the output with the format desired by the teachers.

B. MPI e OMP

After the MPI implementation, we decided to do a version that supported the implementations both from the previous

version and from this version. So, we did a version which combined MPI and OpenMP.

IV. Especificações das Máquinas

During this phase, we used 3 different machines to do the tests and make the conclusions. We used our personal machines as well as provided by the teachers.

A. SeARCH

We want to use this part of the report to correct what was said about SeARCH in the previous report. The information that we gave was wrong. So, the SeARCH is divided in 2 chips with 20 cores each and we believed that each core can use 2 threads as the majority of Intel processors.

Cores	40 (divided in 2 chips)
Threads	We believe that it's 80 Threads

B. Tiago Silva's PC

These are the machine specifications of Tiago Silva:

Processador	AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx
Clockspeed	2.3 GHz
Turbo Speed	4.0 GHz
Cores	4
Threads	8
RAM	16GB
Cache Size	L1: 768KB, L2: 4MB, L3: 4MB

C. Simão Cunha's PC

These are the machine specifications of Simão Cunha:

Processador	Intel(R) Core(TM) i7-8750H
Clockspeed	2.2 GHz
Turbo Speed	4.1 GHz
Cores	6
Threads	12
RAM	16GB
Cache Size	L1: 384K, L2: 1.5MB, L3: 9MB

V. Results and analysis

A. SeARCH

Similar to the previous phase, we ran scripts with several tests to allow us to analyse them. In the previous phase, we use fix values of 10000000 points and 4 and 32 clusters. But in this phase we will increase the number of tests and verify the results for 64 and 128 clusters to allow us to see the behaviour of OpenMP and MPI with bigger workloads.

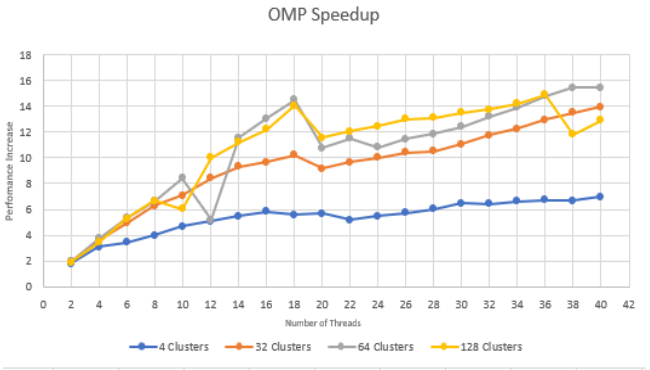
We can see below the base values with the execution times from sequential program - they will be useful to compare with the parallel versions.

TABLE I
Execution times in the sequential version - restricted to 20 iterations

4 Clusters	32 Clusters	64 Clusters	128 Clusters
2,398s	13,505s	27,151s	51,288s

1) OpenMP: In this section, we will show the results obtained from the OpenMP version - this is only and add to the previous table already shown in the previous report. This table shows the different execution times from the OpenMP version with different threads (from 2 until 40, from 2 in 2).

In order to compare the performance gain, we decided to show the obtained results (see table IV) in the following diagram. The speedup was calculated in the same way as the previous phase: $\frac{sequential\_base\_execution\_time}{parallel\_execution\_time}$



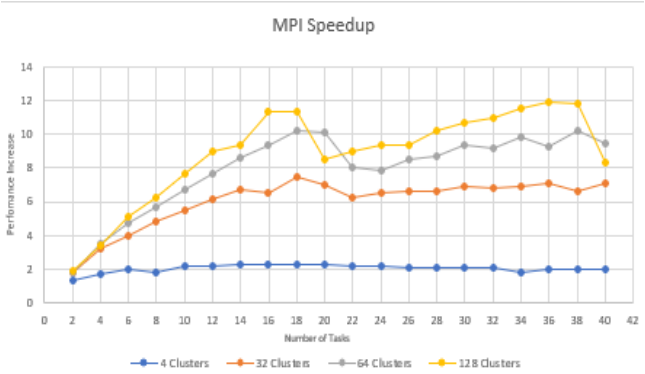
From this diagram, we can extract some information:

- The increase of performance is more significant in the beginning threads and stable from 18-20 threads. This is explained by the existence of the critical section and the following treatment to them. It is hard to evaluate after a certain number of threads if the increase of this number means the increase of the program performance, because, even though the creation of the region is faster, it also makes longer in the critical section;
- The more clusters we have (more charge), the more is the performance gain. This is explained by the fact that, in the costly region, the increase of a cluster means that 10 000 000 points will calculate the distance until that cluster and compare it with the other points, while in the critical region the increase of the cluster only increases in one loop in the execution. However, that only happens until 64 clusters, since in the 128 cluster test, the performance gain was basically the same. We believe that we should test with a greater quantity to see a diference - for example- 256 clusters;
- The performance gain peak, for different number of clusters, was always while using 40 threads, except when the number of the clusters was 128.

2) MPI: In this section, we show the obtained results in MPI version. The table is similar to the OMP version, but instead of "Threads", we write "Tasks".

Only by looking to these results (see table V), without any help from diagrams, we can already see that, even though the execution times were way faster that the results obtained from the sequential version, these are way slower than the OpenMP version, which we already could expected because the overhead in the process communication is bigger than the one obtained in threads use.

As in the previous section, we show the results in the following diagram to compare the performance gain:

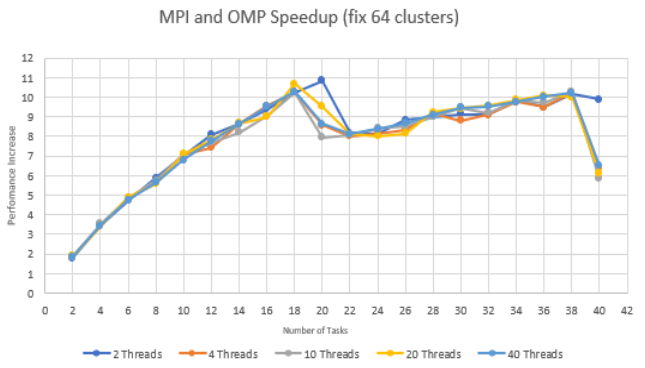


From this diagram we can extract some information:

- The performance gain wasn't as high as in OpenMP version, as already said before;
- As in OpenMP version, the performance gain is greater in the first tasks and, with the increase of the workload (number of clusters), the gain is greater;
- The is a downfall ("queda brusca" in Portuguese) in the performance gain from 20 to 22 taks (with 128 clusters it occurred from 18 to 20). We believe that, until 20 processes, SeARCH can use only one of its chips and with 22, as it already had to use the another chip, the overhead in the process communication increases;
- The performance peak, for different number of clusters, was always 18-20 tasks, except for 128 clusters, which was 36 tasks.

3) MPI and OMP: In this version, we decided to fix the number of clusters to 64 and see how it changes with several threads (2, 4, 10, 20 and 40) and tasks (from 2 until 40, from 2 in 2). We did the same tests for 32 clusters, but in our analysis, they were redundant.

We translated the execution times obtained in these results (see table VI) in the following diagram:



From this diagram, we can extract some information:

- The performance gain wasn't significant relatively to the use with only MPI, where most of the time was a lost. This is due to the join of the overheads, process communication treatment and use of threads ou even a fail from us in the code implementation;
- The performance peak was with 20 tasks and 2 processes with the execution time of 2,503s, which is greater that the peak of the version with only MPI, but is smaller than the version with OpenMP (1,757s) using 40 threads - this is only in relation of the tests with 64 clusters.

B. Tiago Silva's PC

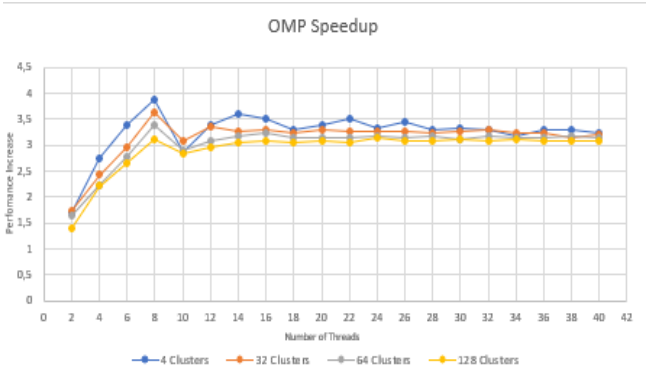
Similar to the tests done in SeArch, these ones were done in Tiago Silva's PC.

TABLE II
EXECUTION TIMES IN THE SEQUENTIAL VERSION - RESTRICTED TO 20 ITERATIONS

4 Clusters	32 Clusters	64 Clusters	128 Clusters
1,94s	8,832s	14,553s	26,465s

With this table, we can see that the sequential execution times was way faster than the obtained in the SeARCH. We can expect right from there lower values of performance gain, because the execution time used as base in the calculus is smaller. However, we can analyse the behavior from OpenMP and MPI versions in relation of the number of threads and tasks, respectively.

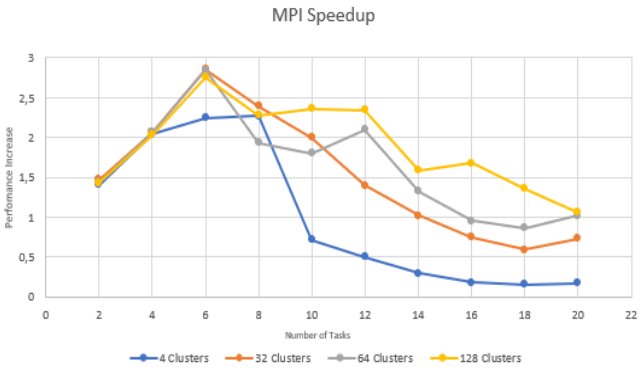
1) OpenMP: In this version, we did the same tests as what were used in SeARCH. We can create the following diagram using these results (see table VII):



From these diagram, we can extract some information:

- The performance peak was in 8 threads for all numbers of clusters, stabling in the following threads. The peak was before the one seen in SeARCH, due to the fact that the SeARCH has more cores than the Tiago's PC;
- The more load (number of clusters) we have, the greater is the performance gain, except for 128 clusters, as seen in the tests done in SeARCH.

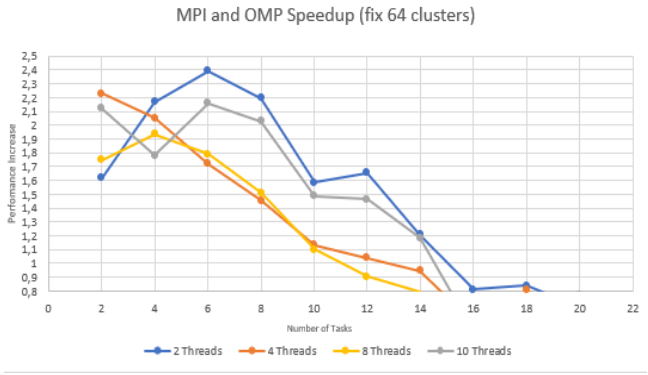
2) MPI: As this PC only has 4 cores (way less than in SeARCH), we downsize the number of tests until 20 tasks, having in mind that is an absurd and probably will decrease the performance in an significant way. The results (see table VIII) can be translated into the following performance diagram:



From this diagram it is possible to notice that:

- Our expectations that performance would drop were correct, since performance drops off from the peak of 6 tasks, resulting in losses from 10 tasks when using 4 clusters - also a loss at 18 tasks when the number of clusters is 32.

3) MPI and OMP: For the same reason as the restrictions above, we restricted the tests to only 2, 4, 8 and 10 threads on this PC. The results (see IX) were translated into this diagram:



From which we can draw the following observations:

- In general, it was worse than using only MPI.
- 2 threads and 6 tasks was the best performance achieved with an execution time of 6.084s, but even then, it was behind the best execution time using only MPI and OMP (5.115s, using 6 tasks and 4.305s, using 8 threads respectively).
- After 4 tasks, there is rarely any gain in performance, most of it is a loss.

C. Simão Cunha's PC

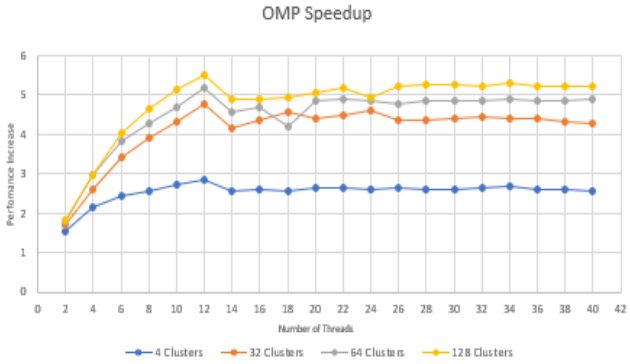
Similar to the tests done on the SeARCH machine and on Tiago Silva's PC, tests were also performed on Simão Cunha's PC.

TABLE III
EXECUTION TIMES IN SEQUENTIAL VERSION - RESTRICTED TO 20 ITERATIONS

4 Clusters	32 Clusters	64 Clusters	128 Clusters
1,412s	7,598s	15,066s	31,304s

This PC obtained the fastest sequential values for the tests with 4 and 32 clusters and was slightly behind Tiago's PC for the others.

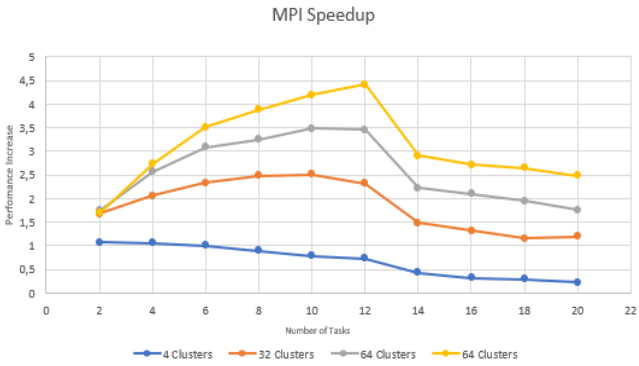
1) OMP: In the OMP version, the same tests performed previously on other machines were done. These were the results (see table X) obtained in the OMP version on Cunha's PC that were translated into the following graph.



From this diagram, we can conclude the following:

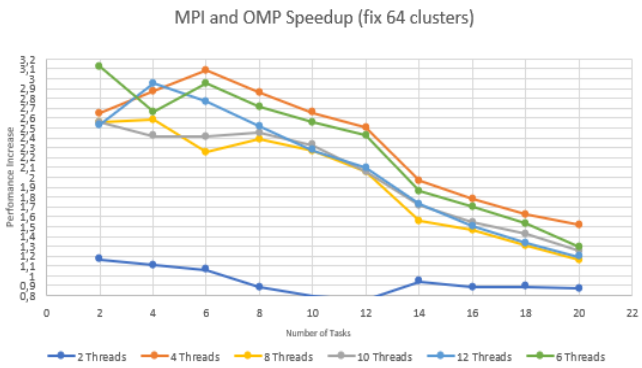
- Confirmation of the notations taken from the previous diagrams of the other machines. The only difference being that the peak of this was at 12 threads and then had a similar behavior to what was already seen in the previous diagrams post-peak.

2) MPI: We applied the same restrictions in these tests as seen on Tiago's PC, for the same reasons (low number of cores, in this case 6) - see table A.



Through this diagram, we reinforced the conclusions drawn from the previous ones and we saw that on this PC, the peak performance obtained in the MPI version was with 12 tasks.

3) MPI e OMP: For this version, we decided to test with 2 to 12 threads - see table XI.



From this diagram resulting from the results obtained above, we can conclude that:

- Similar to what happened on Tiago's PC, this version was generally below the version with only MPI, with the lowest execution time being 4.825s when using 2 tasks and 6 threads. This value was above the execution times in the solo MPI version (4.329s with 10 tasks) and solo OMP version (2.909s with 12 threads).
- The best configuration for this PC in this version is 6 threads and 2 tasks.

D. Comparisons between machines

1) OMP: Comparing the execution time of the various machines in the OMP version, we see that for an initial number of threads, Cunha's PC is faster, but it is surpassed in performance by SeARCH when the number of threads is increased. Tiago's PC was always slower than one of the two. See figures 1 and 2 in the annex.

2) MPI: Comparing the execution time of the various machines in the MPI version, we see that for a low number of tasks and load (4 clusters), Tiago's PC is faster, being overtaken by Cunha's PC when this load is greater (32 clusters). When the number of tasks increases, as expected, the execution time of SeARCH improves, overtaking both Tiago's PC and Cunha's PC, which worsen greatly. See figures 3 and 4 in the annex.

3) MPI e OMP: Here, SeARCH as expected was the one that was able to take better advantage of a mixed configuration, achieving the best execution time (2.503s with 20 tasks and 2 threads). Tiago's PC, as expected, was the worst (6.084s with 6 tasks and 4 threads).

VI. Conclusions

Through the development of this phase, we were able to conclude that:

- The implementation of MPI (distributed memory) is more complex than the implementation of OpenMP (shared memory).
- Shared memory for this environment and these machines has proven to be more efficient than distributed memory. However, perhaps using higher loads in the tests (larger number of points and clusters) this would be surpassed in efficiency.
- There is no perfect number of threads and tasks. For different machines and environments it is necessary to adjust this number to achieve maximum efficiency in performance.
- The parallelized code versions are much more efficient in terms of execution time than the sequential versions (as seen in the tests, it can be 15 times faster).

References

[1] OpenMPI website - https://www.open-mpi.org/

[2] OpenMP website - https://www.openmp.org/

[3] Slides from course of Parallel Computing, doned by AJProença and JLSobral

APPENDIX

TABLE IV
EXECUTION TIMES IN THE OPENMP VERSION - RESTRICTED TO 20 ITERATIONS

Threads	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	1,332s	7,037s	14,151s	26,901s
4	0,771s	3,786s	7,282s	14,823s
6	0,696s	2,725s	5,087s	9,715s
8	0,602s	2,156s	4,084s	7,693s
10	0,513s	1,913s	3,226s	8,525s
12	0,471s	1,6s	5,263s	5,123s
14	0,437s	1,45s	2,352s	4,583s
16	0,41s	1,4s	2,082s	4,195s
18	0,43s	1,32s	1,873s	3,656s
20	0,422s	1,47s	2,525s	4,445s
22	0,465s	1,4s	2,362s	4,255s
24	0,438s	1,35s	2,506s	4,121s
26	0,419s	1,3s	2,369s	3,938s
28	0,398s	1,286s	2,294s	3,919s
30	0,37s	1,221s	2,192s	3,802s
32	0,374s	1,149s	2,057s	3,736s
34	0,363s	1,1s	1,952s	3,61s
36	0,357s	1,044s	1,833s	3,446s
38	0,36s	1s	1,76s	4,349s
40	0,344s	0,969s	1,757s	3,982s

TABLE V
EXECUTION TIMES IN MPI VERSION - RESTRICTED TO 20 ITERATIONS

Tasks	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	1,818s	7,682s	15,318s	27,554s
4	1,405s	4,26s	7,715s	15,088s
6	1,183s	3,381s	5,776s	10,086s
8	1,307s	2,8s	4,779s	8,215s
10	1,109s	2,462s	4,028s	6,737s
12	1,089s	2,197s	3,543s	5,702s
14	1,049s	2,023s	3,145s	5,452s
16	1,063s	2,066s	2,909s	4,523s
18	1,056s	1,799s	2,666s	4,506s
20	1,045s	1,933s	2,695s	6,023
22	1,121s	2,156s	3,373s	5,703s
24	1,124s	2,067s	3,459s	5,468s
26	1,14s	2,052s	3,193s	5,478s
28	1,17s	2,03s	3,133s	5,03s
30	1,171s	1,969s	2,904s	4,794s
32	1,157s	1,995s	2,949s	4,695s
34	1,353s	1,965s	2,772s	4,441s
36	1,209s	1,9s	2,93s	4,295s
38	1,232s	2,055s	2,667s	4,352s
40	1,226s	1,896s	2,872s	6,13s

TABLE VI
EXECUTION TIMES IN THE MPI AND OMP VERSION - RESTRICTED TO 20 ITERATIONS

Tasks	2 Threads	4 Threads	10 Threads	20 Threads	40 Threads
2	15,055s	14,511s	14,619s	14,288s	14,593s
4	7,798s	7,841s	7,635s	7,956s	7,867s
6	5,754s	5,64s	5,593s	5,549s	5,703s
8	4,628s	4,813s	4,725s	4,877s	4,816s
10	3,888s	3,842s	3,852s	3,824s	3,99s
12	3,36s	3,657s	3,53s	3,476s	3,5s
14	3,138s	3,141s	3,304s	3,129s	3,137s
16	2,886s	2,845s	3,008s	3,015s	2,853s
18	2,649s	2,647s	2,645s	2,541s	2,643s
20	2,503s	3,169s	3,411s	2,854s	3,145s
22	3,324s	3,392s	3,354s	3,34s	3,348s
24	3,348s	3,33s	3,224s	3,383s	3,241s
26	3,074s	3,259s	3,194s	3,335s	3,133s
28	3,02s	2,962s	3,012s	2,943s	2,988s
30	2,971s	3,085s	2,874s	2,884s	2,873s
32	2,975s	2,986s	2,957s	2,845s	2,849s
34	2,766s	2,78s	2,763s	2,753s	2,786s
36	2,856s	2,855s	2,793s	2,696s	2,709s
38	2,662s	2,647s	2,642s	2,711s	2,658s
40	2,746s	4,245s	4,659s	4,447s	4,173s

TABLE VII
EXECUTION TIMES IN THE MPI VERSION - RESTRICTED TO 20 ITERATIONS

Threads	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	1,145s	5,127s	8,894s	18,946s
4	0,709s	3,65s	6,548s	12s
6	0,575s	2,977s	5,254s	10,022s
8	0,499s	2,426s	4,305s	8,528s
10	0,674s	2,871s	5,008s	9,316s
12	0,571s	2,639s	4,741s	8,976s
14	0,539s	2,696s	4,573s	8,72s
16	0,553s	2,685s	4,502s	8,575s
18	0,588s	2,726s	4,631s	8,686s
20	0,574s	2,681s	4,619s	8,637s
22	0,554s	2,704s	4,617s	8,695s
24	0,584s	2,706s	4,597s	8,47s
26	0,561s	2,706s	4,636s	8,586s
28	0,589s	2,724s	4,583s	8,572s
30	0,582s	2,709s	4,672s	8,512s
32	0,59s	2,685s	4,608s	8,561s
34	0,61s	2,721s	4,621s	8,542s
36	0,592s	2,745s	4,634s	8,566s
38	0,587s	2,812s	4,588s	8,562s
40	0,598s	2,768s	4,651s	8,615s

TABLE VIII
EXECUTION TIMES IN THE MPI VERSION - RESTRICTED TO 20 ITERATIONS

Tasks	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	1,376s	6,009s	10,084s	18,488s
4	0,95s	4,272s	7,041s	13,008s
6	0,863s	3,093s	5,115s	9,594s
8	0,855s	3,692s	7,522s	11,595s
10	2,723s	4,41s	8,069s	11,199s
12	3,861s	6,298s	6,939s	11,289s
14	6,506s	8,651s	11,007s	16,643s
16	10,962s	11,827s	15,232s	15,761s
18	12,317s	14,882s	16,772s	19,527s
20	11,09s	12,086s	14,259s	24,988s

TABLE IX
EXECUTION TIMES IN MPI AND OMP VERSION - RESTRICTED TO 20 ITERATIONS

Tasks	**2 Threads**	**4 Threads**	**8 Threads**	**10 Threads**
2	9,001s	6,532s	6,866s	14,631s
4	6,718s	7,099s	8,179s	13,005s
6	6,084s	8,454s	6,735s	10,419s
8	6,627s	10,001s	7,171s	10,4s
10	9,187s	12,851s	9,781s	11,757s
12	8,789s	14,017s	9,949s	16,827s
14	12,039s	15,381s	12,282s	18,345s
16	18,01s	25,525s	28,71s	22,599s
18	17,332s	18,074s	20,587s	26,672s
20	21,38s	22,462s	18,351s	23,758s

TABLE X
EXECUTION TIMES IN OMP VERSION - RESTRICTED TO 20 ITERATIONS

Threads	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	0,912s	4,479s	8,45s	17,236s
4	0,663s	2,909s	5,089s	10,478s
6	0,575s	2,229s	3,933s	7,781s
8	0,555s	1,946s	3,539s	6,764s
10	0,518s	1,767s	3,212s	6,105s
12	0,495s	1,591s	2,909s	5,683s
14	0,555s	1,834s	3,306s	6,423s
16	0,543s	1,741s	3,212s	6,391s
18	0,553s	1,658s	3,603s	6,342s
20	0,535s	1,723s	3,113s	6,2s
22	0,536s	1,691s	3,08s	6,032s
24	0,54s	1,651s	3,107s	6,347s
26	0,537s	1,751s	3,171s	5,978s
28	0,54s	1,743s	3,118s	5,971s
30	0,54s	1,725s	3,105s	5,959s
32	0,533s	1,708s	3,098s	5,984s
34	0,53s	1,731s	3,077s	5,918s
36	0,539s	1,729s	3,106s	5,986s
38	0,54s	1,757s	3,101s	6,006s
40	0,554s	1,771s	3,075s	5,999s

Tasks	4 Clusters	32 Clusters	64 Clusters	128 Clusters
2	1,308s	4,532s	8,631s	18,523s
4	1,324s	3,696s	5,87s	11,451s
6	1,407s	3,264s	4,877s	8,906s
8	1,574s	3,056s	4,624s	8,061s
10	1,792s	3,025s	4,329s	7,454s
12	1,916s	3,268s	4,35s	7,093s
14	3,303s	5,095s	6,783s	10,738s
16	4,341s	5,739s	7,178s	11,495s
18	4,725s	6,552s	7,71s	11,838s
20	6,118s	6,346s	8,561s	12,59s

TABLE XI
EXECUTION TIMES IN MPI AND OMP VERSION - RESTRICTED TO 20 ITERATIONS

Tasks	2	4	6	8	10	12
2	7,387s	5,679s	4,825s	5,886s	5,89s	5,955s
4	5,306s	5,235s	5,661s	5,818s	6,226s	5,107s
6	4,589s	4,889s	5,108s	6,692s	6,248s	5,449s
8	5,212s	5,262s	5,542s	6,315s	6,132s	5,993s
10	5,47s	5,673s	5,901s	6,627s	6,475s	6,621s
12	5,749s	6,019s	6,21s	7,341s	7,329s	7,198s
14	7,18s	7,68s	8,094s	9,679s	8,772s	8,74s
16	8,077s	8,442s	8,842s	10,275s	9,763s	10,051s
18	8,632s	9,27s	9,854s	11,494s	10,6s	11,333s
20	9,751s	9,92s	11,614s	12,963s	11,969s	12,611s

Fig. 1. Comparation of execution times between different machine in OpenMP version (with 4 clusters)

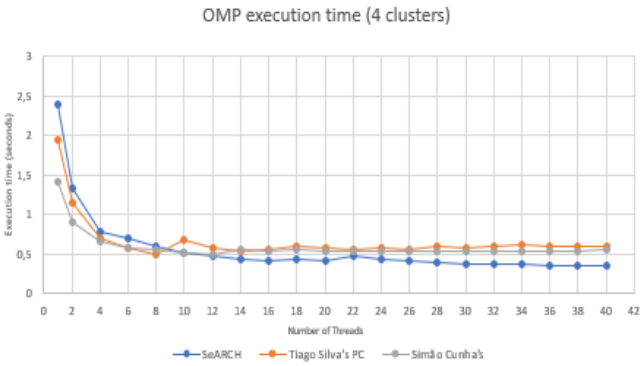


Fig. 2. Comparation of execution times between different machine in OpenMP version (with 32 clusters)

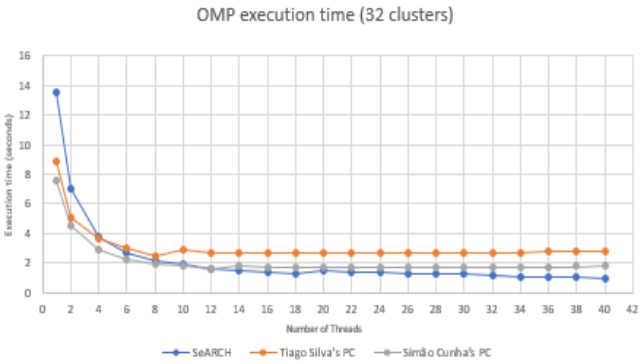


Fig. 3. Comparation of execution times between different machine in MPI version (with 4 clusters)

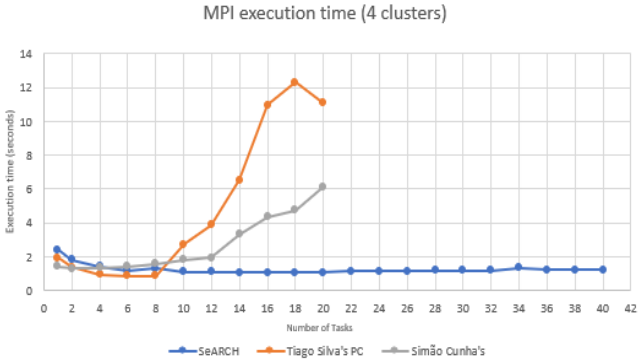


Fig. 4. Comparation of execution times between different machine in MPI version (with 32 clusters)