Advanced Computer Architecture Report

The project is based on the implementation of an algorithm that solves a maze given by the user. This report focuses on the analysis of performance obtained parallelising the code and using cloud computing to execute. To be solvable the maze should have specific properties:

- The maze must be extracted from a binary or greyscale image;
- The maze must have only one entrance and one exit.
- The maze must be a “perfect maze”, this means that only one path solves the labyrinth;

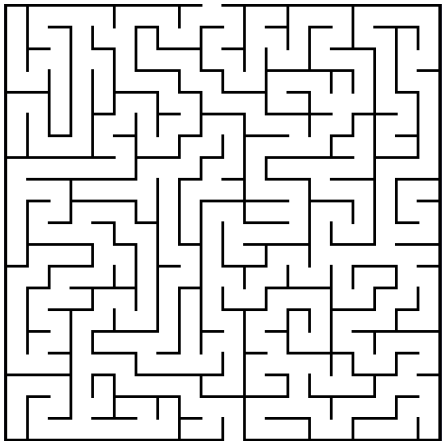
To reach the aim mentioned above the OpenCV library is used given that it provides optimised functions to deal with images and matrices.

Analysis of the serial algorithm (code here)

Step 1

The image is provided to the program by the user from command line at the moment of execution and it is transformed into a Mat object*.

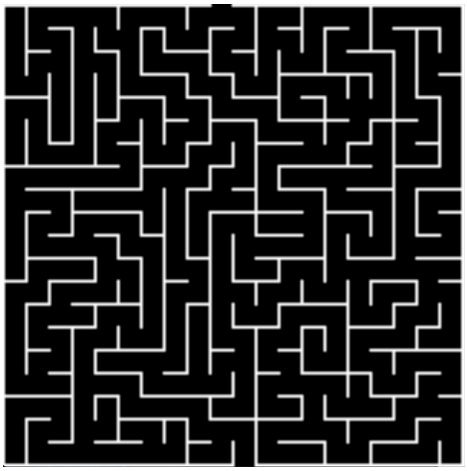
*Mat is a type of object handled by OpenCV library.



Example of input image

Step 2

The obtained matrix is thresholded in order to obtain a binary matrix, composed of 0s (black pixels) and 1s (white pixels). The resulting matrix is complemented so that later it will be possible to perform dilation on walls and not on all the possible paths.

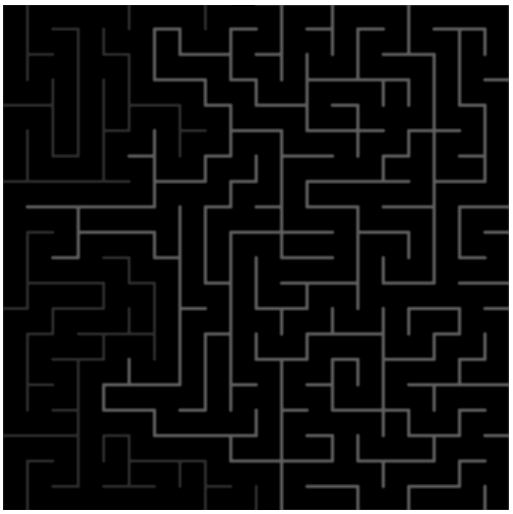


Thresholded & Complemented image

Step 3

In this phase it is necessary to find all connected components*. This operation allows to distinguish the walls of the labyrinth. Each value of the matrix is compared with adjacent pixels (4-connectivity). If the value of the two considered pixels is the same, then a certain label is associated to them, this label will be used until there is at least one adjacent pixel with the same value. As soon as adjacent elements are different from the reference one in all directions, the label is incremented and the component is regarded as completed. From that moment on, it is impossible to find other pixels belonging to that component. The process is repeated until the whole matrix is scanned and all possible elements have been detected and labelled. In case of a perfect maze, having two walls, the number of possible labels is three. “0” for the path, “1” for the first wall, “2” for the second wall.

The code to label the matrix can be found at https://stackoverflow.com/questions/14465297/connected-component-labeling-implementation

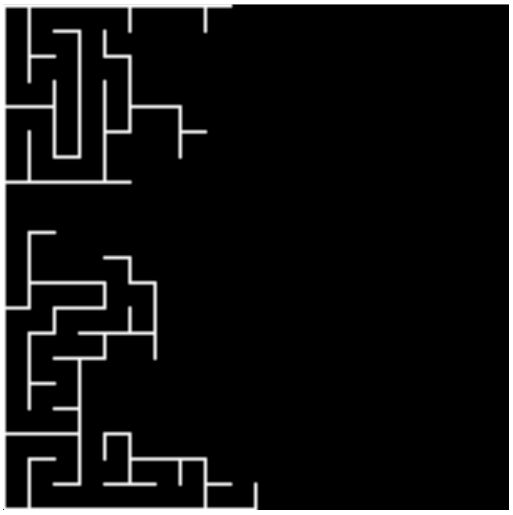


The black area identifies all possible paths (label=0).
The dark grey area corresponds to the first detected wall (label=1).
The light grey area signals the second wall (label=2).

Labeled image

Step 4

One of the walls is selected whilst the other one is completely removed. This operation is carried out deleting one of the two connected components and setting its corresponding values to zero. At this point the solution is given by following the remaining wall. Following steps will allow avoiding dead ends.



1 - wall detected

Step 5

The previous problem is solved thanks to mathematical morphology operators: dilation and erosion. Firstly, the dilation is performed on the matrix obtained in “Step 4”. Secondly, the erosion is performed on the dilated matrix. The size of the structuring element used for these morphological operations is chosen on the basis of the dimension of the path.

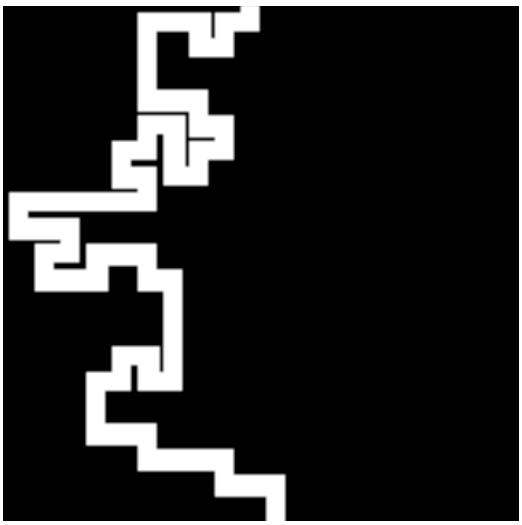
Dilated Image

Eroded Image

Morphological Gradient

Step 6

The final solution is obtained by the difference between the dilated matrix and the eroded one.



Maze Correct Track

A priori study of parallelism (code here)

In this section is discussed a possible way to parallelise the application.

Theoretical approach

The ideal case would be reading the given image, dividing it into sub-matrices and delivering them to slave processes. They would execute all needed steps aimed at obtaining the partial final solution letting the master collect the results to show the whole maze track. Nevertheless, this approach cannot be implemented due to some issues:

1. Mat objects are not accepted as parameters in MPI functions;
2. Slave processes do not know dimensions of the received matrix;
3. The labelling process must be carried out sequentially;
4. Slave processes need to know the structuring element size, evaluated by the master;
5. Dilation and Erosion operations may yield wrong results for border pixels of sub-matrices if not handled appropriately.
6. Reconstructing the matrix cannot be carried out simply putting together the pieces received from slaves.

1.

The image given in input by the user is transformed into a matrix (Mat object) that should be shared with slave processes. The problem is that it cannot be shared as it is because MPI functions do not accept objects of that kind. As a consequence, it is necessary to convert the matrix into a 1D-array.

2.

The master needs to send an array. Sending only the array would not allow slaves to reconstruct the sub-matrix because they would lack information about width and heigh. This implies that the master has to send to all slaves the number of rows and columns that they will be receiving. In this way, for each slave, it will be possible to rebuild the sub-matrix.

3.

After having performed the initial operations on the matrix (threshold and complement) it is necessary to individuate the two walls of the maze using a connected component algorithm. This cannot be paralleled because otherwise the link among all the pieces of a wall would be impossible to recover. For this reason all the slaves must send computed data back to the master. It will put the pieces back together and will perform the labelling operation. At this point the maze can be decomposed again and shared for the second time with slave processes but in a different way compared to the first time. See as follows.

4.

When redistributing the matrix among slaves it is fundamental to give them some additional information with respect to the previous case. As well as the number of rows and columns, the master sends the dimension of the structuring element. This datum is required by dilation and erosion that will be computed in parallel.

5.

Not only does the amount of data sent to slaves change but also the dimension of the sub-matrices. To avoid loosing dilation and erosion results of pixels near the top and lower borders it is essential to send, in addition to the number of rows previously given to slaves, also a part of the following sub-matrix. To ensure the correctness of the result the number of further rows is equal to the size of the structuring element.

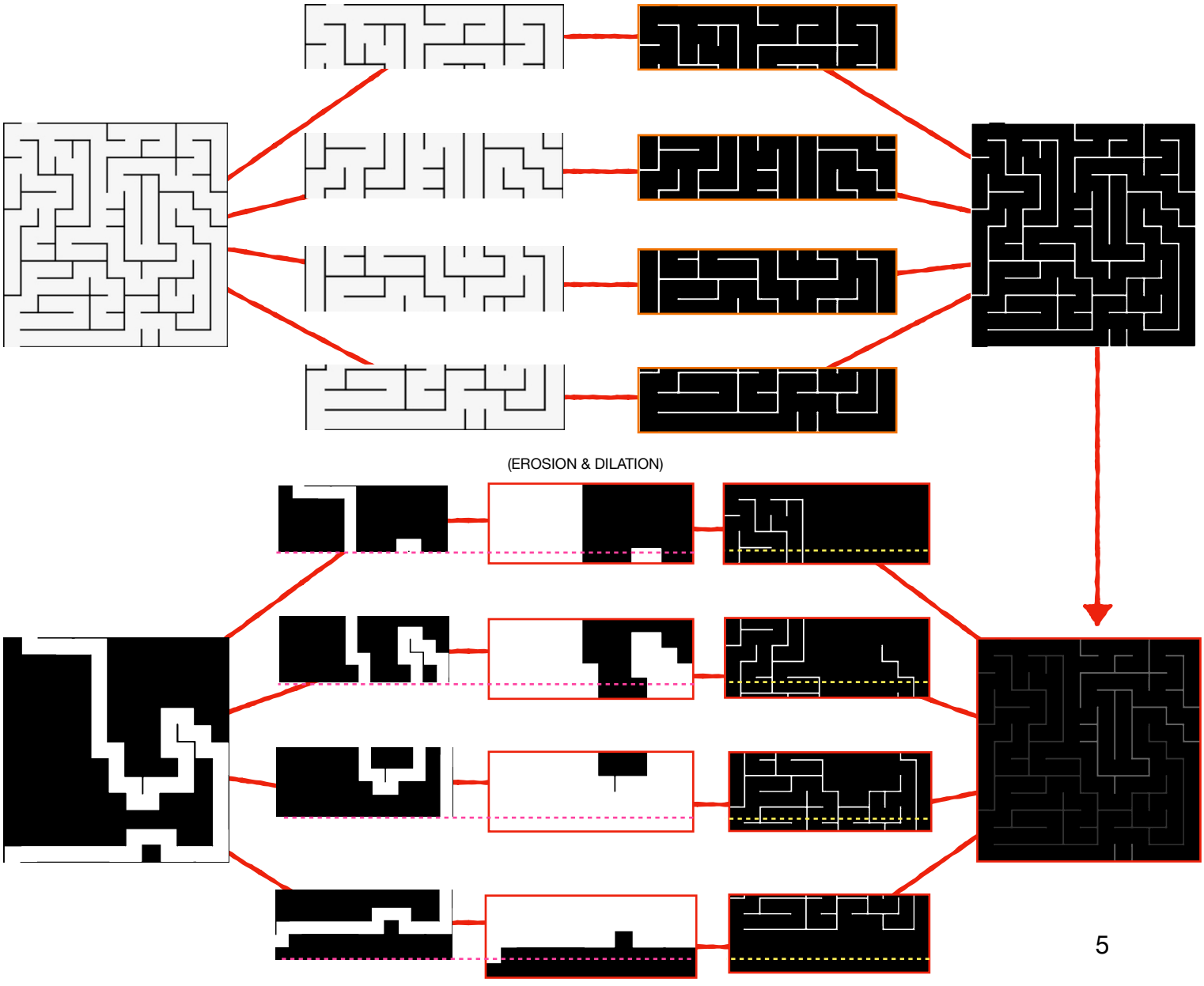
6.

Before rebuilding the whole matrix that contains the final solution, it is important to remove the additional rows used to perform dilation and erosion correctly. Each slave must send to the following slave elaborated extra rows. Then, the process which receives them can perform a logical OR between the corresponding rows and keep all the set pixel in both fragments (the one elaborated by itself and the one received by the other slave) so that the master can rebuild the matrix without any further operation.

The resulting approach

Having analysed the possible issues to deal with due to parallelisation, the resulting choice is to let the master accept the source image and transform it into a matrix to be distributed to slaves. They perform binarization and complement of the received sub-matrices and give their results back to the master. The parent process rebuilds the entire matrix and labels connected components. Having done that, the master decompose the result into new sub-matrices taking into account problems that could arise due to dilation and erosion. Slaves set to zero the part of the sub-matrix that does not contain the wall to be maintained and, subsequently, they perform morphological operations obtaining first partial results. At this point, all the slaves merge the rows shared with another slave. Consequently, the master receives all the sub-matrices with the final partial solutions and puts them together to show the solved maze.

SUMMARY SCHEMA - EXAMPLE USING 4 CORES



Another possible approach

Putting aside the previous implementation of the parallel application, another option to consider is the possibility to let the master execute all the initial operations until labelling. Then the matrix is divided into sub-matrices to let slaves perform the rest of the algorithm and give back the final partial results. In this case, there is a reduction of interprocess communication but sequential code increases and problems related to dilation and erosion still remain.

Communication and Synchronisation strategy in real implementation

In the course of execution loads of data must be transferred from a process to another. There can be identified communication functions, provided by OpenMPI, that allow the parallel program to execute properly:

- **MPI_Send**: this function is necessary for the master to communicate to slaves the dimensions of sub-matrices that they are going to receive. The MPI_Send is used two times: the first time to send the number of rows and columns thanks to an 1-D array with two elements inside; the second time as well as parameters just mentioned, structuring element' size is sent.

- **MPI_Recv:** this function is used in couple with MPI_Send to receive data aforementioned.

- **MPI_Scatter:** like the previous functions it is used two times in order to uniformly distribute the matrix among the slaves.

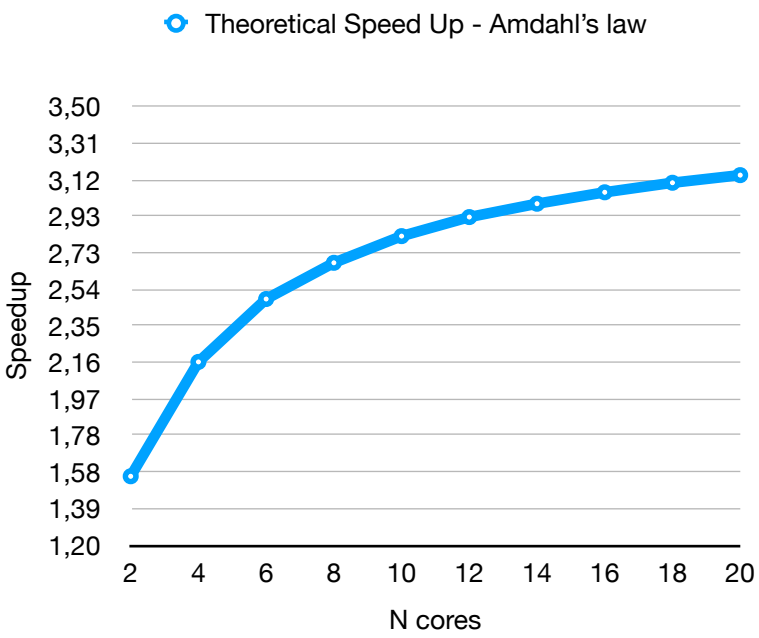
- **MPI_Gather:** this function is used in couple with MPI_Scatter to receive 1-D arrays containing sub-matrix' elements.

- **MPI_Sendrecv:** used by each process to send last rows of its sub-matrix to the following process and receive the initial rows from the previous slave to merge the results.

- **MPI_Barrier:** it is called in two occasions. Before slaves send to each other the fragment of shared rows on which they have to perform the logical OR and before giving back the result to the master, so before the gather.

A-priori theoretical assessment of the speed-up (through Amdahl's law)

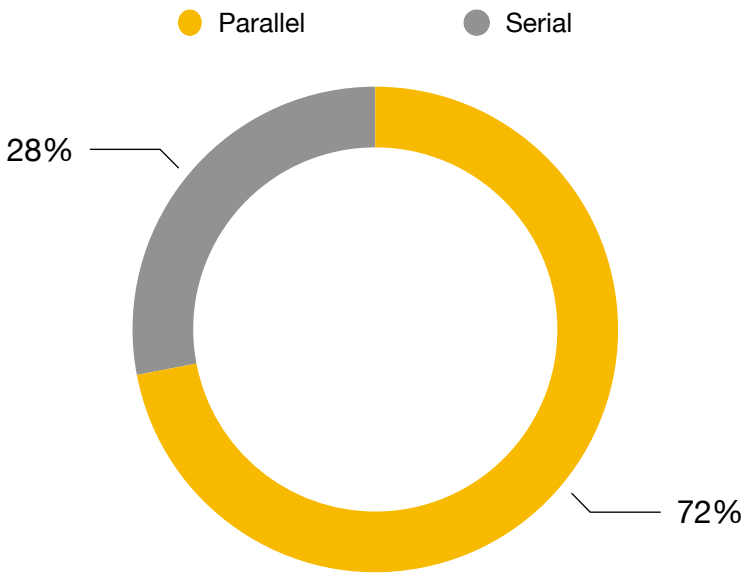
The aim of this section is evaluating the maximum possible speed-up given by the parallelisation of sequential code.
To reach the mentioned goal, an esteem of instructions needed to execute single operations has been carried out. From these data it is possible to evaluate the maximum speed-up using the Amdahl's law because sequential portion and parallel portion of code are identifiable.



Type of instruction	N. istr
IMG_READING	105
BINARIZATION	148
COMPLEMENT	16
FIND DIMENSION PATH	89
FIND CONNECTED COMPONETS	237
TRACK TO FOLLOW	66
DILATION	292
EROSION	290
DIFFERENCE	109
OTHERS	30
OTHERS	45
TOTAL	**1427**

Table 1

* Orange color indicates instructions that can be executed in parallel



Performance Analysis Using Google Cloud VMs

This section contains all analysis about the program execution. To monitor the time and clock cycles needed, some functions of the OpenCV library have been used. The technique is the following:

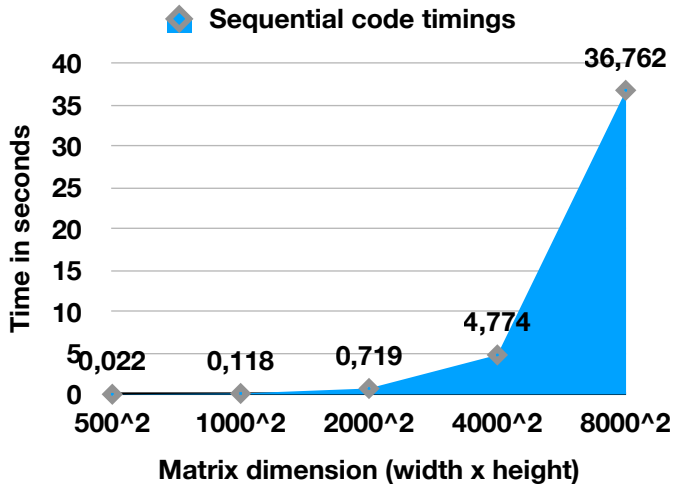
```
int64 t1, t2;
int64 clockcycles;
Double timing;
t1 = cv.getTickCount();
// code
t2 = cv.getTickCount();
clockcycles = t2 - t1;
timing = (t2 - t1) / cv.getTickFrequency();
```

Sequential tests

Initial tests have been run using only one core. The execution, therefore, is completely sequential.

N. ISTANCES	RAM (GB)	IMG SIZE	TIMING (s)	CLOCK CYCLES
1	8	500x500	0,022	22.228.708
1	8	1000x1000	0,118	118.310.544
1	8	2000x2000	0,719	7.199.139.680
1	8	4000x4000	4,774	4.773.892.697
1	8	8000x8000	36,762	36.762.375.954

Table 2



Results shown above are not significant by themselves but they will gain importance in relation to the followings. They will be used as benchmark.

N.B. In the tables of the next pages all speedups refer to the enhancement obtained with respect to the serial execution with 4000x4000 image (time of execution: 4,774 seconds).

Light cluster

The first asset is based on the usage of a relatively high number of instances with a reduced number of virtual cores, two per instance. This approach allows to have up to sixteen available virtual cores due to the fact that a maximum of eight addresses can be assigned to the same Google account.

Light cluster - Strong scalability tests

This section focuses on how performances change according to available resources to execute the program. In this case the problem has a fixed size: always the same picture, with fixed dimensions, is elaborated.

N. INSTANCES	N. vCORE TOTAL	RAM (GB)	IMG SIZE	TIMING (s)	CLOCK CYCLES	Enhancement %	MEMORY (KB)
1	2	8	4000x4000	4,751	4.751.055.382	0,48%	4725
2	4	8	4000X4000	2,917	2.916.696.388	38,90%	4831
3	6	8	4000x4000	2,277	2.277.383.536	52,30%	4884
4	8	8	4000x4000	2,023	2.023.164.176	57,62%	4956
5	10	8	4000x4000	1,812	1.812.112.175	62,04%	5001
6	12	8	4000x4000	1,701	1.701.450.702	64,36%	5067
7	14	8	4000x4000	1,620	1.619.783.003	66,07%	5141
8	16	8	4000x4000	1,541	1.540.510.232	67,73%	5194

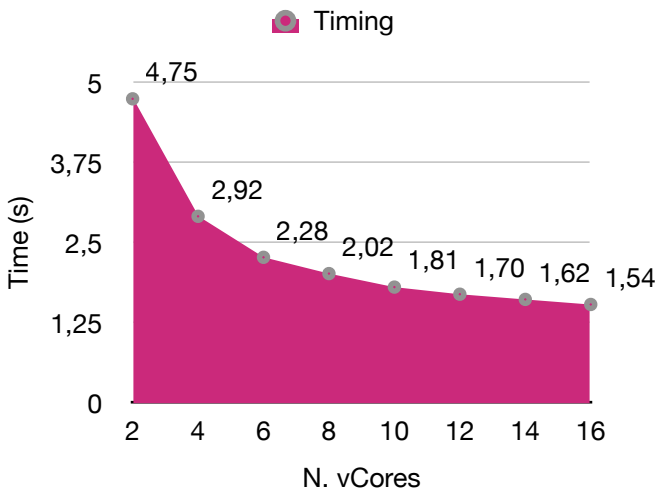
The speedup shown in the table is evaluated with reference to the sequential case (fourth row of Table 2).

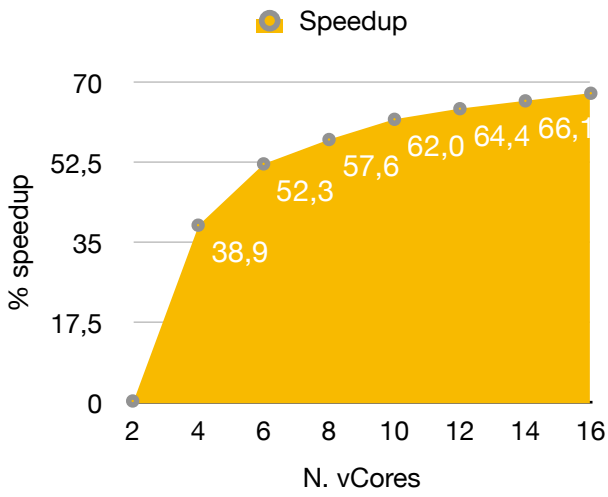
Table 3

The first important result to be taken into account is the reduction of time needed to complete the execution with the increase of virtual cores. Analysing the obtained values, what is immediately noticeable is that the improvement brought by the usage of **two vcores** is not as good as expected. The reason of this behaviour can be related to three possible aspects:

- the cost of creating a second process using MPI;
- the cost of interprocess communication and additional operations which were not present in sequential case;
- the dimensions of sub-matrices still too big to highlight the advantages of parallelisation.

The optimal improvement is obtained with four vcores. Comparing the results of this case with the ones got with two vcores, there is a speed-up of approximately 34%.
Looking at the whole picture, despite the continuous improvements in terms of time, it appears that the augment of the number of vcores does not correspond to progressive improvements. The situation shows that the enhancement is not proportional but it seems to converge.





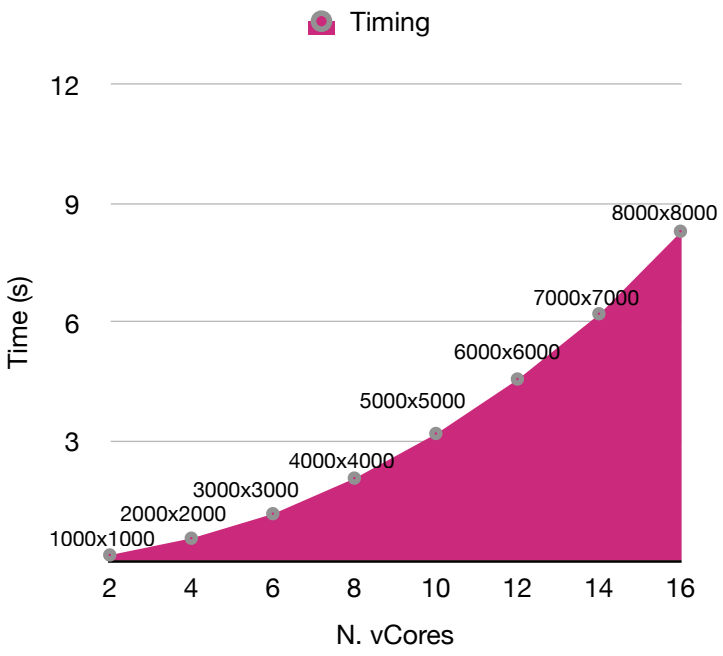
Light cluster - Weak scalability tests

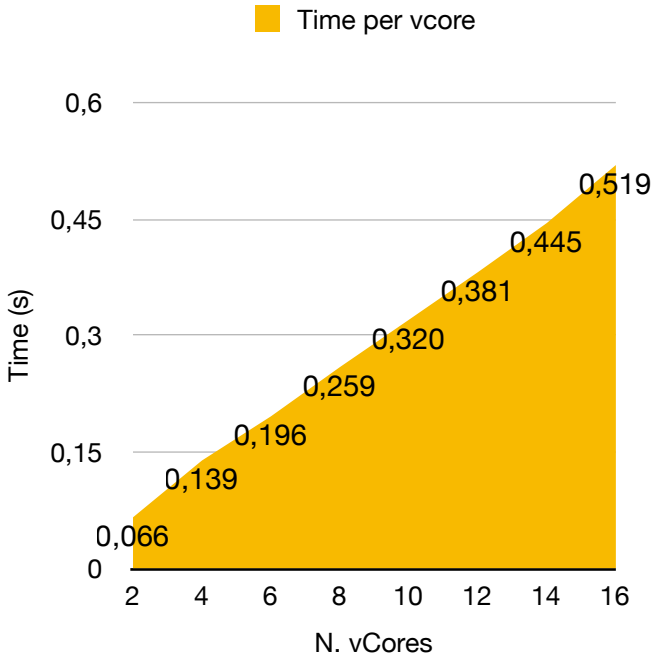
This section describes changes in terms of time of execution obtained when resources increase accordingly with the problem size. To execute these tests in a proper way it is important to maintain proportionality between number of vcores and image dimensions. In other words the approach is similar to assigning to each vcore a matrix of 500x500.

N. INSTANCES	N. vCORE EACH	N. vCORE TOTAL	RAM(GB)	IMG SIZE	TIMING (s)	CLOCK CYCLES
1	2	2	8	1000x1000	0,131	130.967.059
2	2	4	8	2000x2000	0,555	554.615.411
3	2	6	8	3000x3000	1,174	1.174.309.544
4	2	8	8	4000x4000	2,073	2.073.437.251
5	2	10	8	5000x5000	3,201	3.201.415.473
6	2	12	8	6000x6000	4,577	4.576.526.923
7	2	14	8	7000x7000	6,227	6.227.107.295
8	2	16	8	8000x8000	8,311	8.310.837.491

Table 4

The table shows results obtained from various tests, starting from an image with one million of elements (elaborated with two virtual cores) up to a matrix of sixty-four millions of elements (computed with 16 virtual cores).
As expected, execution time's increase is almost proportional to the increase of matrix dimensions and available resources.





The ideal case would be having always the same value as "time per vcore", independently from the number of virtual core used. Obviously this is impossible because of communication among processes. The increment in the graph on the right, indeed, highlights the cost of interprocess communication which rises along with the number of vcores.

Fat cluster

The second asset is based on the usage of a reduced number of instances with a high number of virtual cores, eight per instance. This approach allows to have up to twenty-four available virtual cores due to the fact that a maximum of twenty-four vcores can be assigned in the same region.

Fat cluster - Strong scalability tests

This section, as for the light cluster, focuses on how performances change according to available resources to execute the program having a problem with fixed size: always the same picture, containing 16 millions of elements, is elaborated.

N. INSTANCES	N. vCORE EACH	N. vCORE TOTAL	RAM (GB)	IMG SIZE	TIMING (s)	CLOCK CYCLES	SPEEDUP - TIME
1	8	8	8	4000x4000	1,947	1.946.510.391	59,23%
2	8	16	8	4000X4000	1,506	1.506.281.552	68,45%
3	8	24	8	4000x4000	1,429	1.429.309.836	70,06%

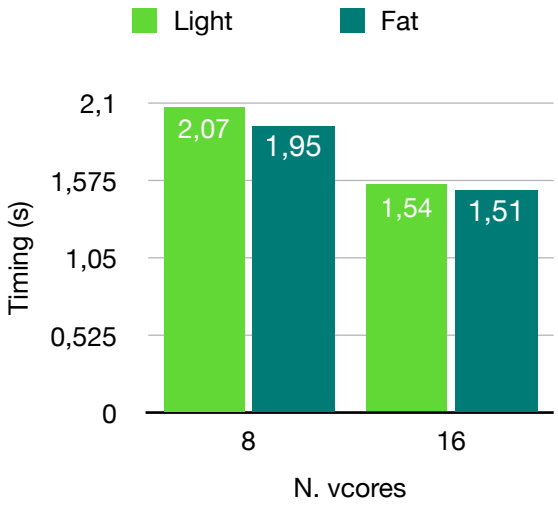
The speedup shown in the table is evaluated with reference to the sequential case (fourth row of Table 2)

Table 5

Sampled values are useful to make an analysis on which of the two possible assets (light cluster or fat cluster) is better in terms of performance. Making a comparison between results with eight vcores and sixteen vcores of the current configuration with those of the light cluster allows to notice that a fat cluster is more performant.

TYPE	N. vCORE	IMG SIZE	Timing (s)	Enhancement %
LIGHT	8	4000x4000	2,073	**6,08%**
FAT			1,947	
LIGHT	16		1,541	**2,25%**
FAT			1,506	

The table makes clear the benefit of using a fat cluster thanks to the smaller execution time sampled.



Fat cluster - Weak scalability test

Tests of this section are executed keeping a fixed sized problem per core. The reason behind images' dimensions is that the maximum dimension tested is 8000x8000, elaborated using 24 cores. Therefore, this configuration is comparable with a case in which about 333x333 matrices are assigned to each core.

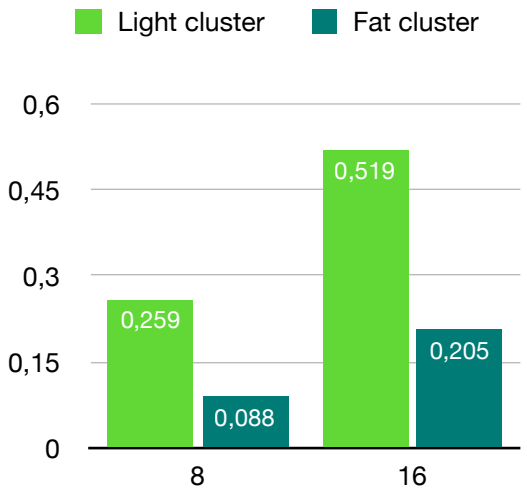
N. INSTANCES	N. vCORE EACH	N. vCORE TOTAL	RAM(GB)	IMG SIZE	TIMING (s)	CLOCK CYCLES
1	8	8	8	2667x2667	0,70567	705670450
2	8	16	8	5333x5333	3,28587	3285869556
3	8	24	8	8000x8000	6,91906	6919060802

Table 6

As expected, overall timings get higher test by test but the analysis of performance in this case must be carried on considering each core singularly.

N. OF vCORES	TIME PER vCORE (s)
8	0,08820875
16	0,205366875
24	0,288294166

As for the light cluster, also in this case the time does not remain constant but the increase of



Time per vcore of the two possible assets

cores brings in an overhead due to interprocess communications.
The graph above clarifies how beneficial the fat cluster is with respect to the light cluster comparing the estimated time per virtual core in case it is launched with 8 vcores or 16 vcores. The 24 vcores case has not be taken into account due to missing corresponding benchmark. These data show that communications result to be less demanding in terms of resources when virtual machines host a large number of virtual cores.

Conclusions

Summing up, given that it was not possible to parallelise 100% of the code due to the reasons described before, the speedup is limited. Nevertheless, the actual speed up complies with the theoretical one. Considering all the resulting data with relation to the possible assets to be used in cloud computing, the outcome is that a fat cluster is able to perform in a better way compared to the light cluster if the number of vcores is equal (for this case). Both assets, anyway, highlight a further uses of resources.

Individual Contribution

Marinelli Francesco: Planning Project, serial and parallel version developing focused on complement, labelling and dilation algorithms, light cluster implementation & performance analysis, report correctness, GitHub repository advisor.

Tagliani Fabio: Planning Project, serial and parallel version developing focused on threshold and erosion algorithms, fat cluster implementation & performance analysis, report structure, README on GitHub.

Resources:

GitHub Repository: https://github.com/theblackoreo/ACA-Project
README for execution: https://github.com/theblackoreo/ACA-Project/blob/main/README.md