TÉCNICO LISBOA

Parallel and Distributed Computing

Second Delivery of the Project

Authors:
Manuel Reis 67031
Miguel Borges 67041
Paulo Monteiro 67052

November 30, 2012

1 Approach used for parallelization

To parallelize our serial version using MPI, we started by choosing what structures each process would have, to minimize the number of communications. We chose to have each process having their share of documents, passing the partial averages that it had calculated, and an array indicating the cabinet of each belonging document.

This is a much better alternative to have the processes having their share of cabinets and pass documents among themselves, since we deal with a larger number of documents, which would increase the number of communications.

While doing this, we also tried to divide the workload evenly, because each process only has a portion of the information with it, and it can't, for instance, calculate the averages of each cabinet. We solved this by doing the maximum calculations possible given the possessed information, and then give the responsibility of doing the rest of the calculations to a specific process.

With this approach, we have a good balance between work distribution and number of communications established between processes.

2 Decomposition used

Of the three types of decomposition studied, we only used data decomposition. For this particular problem, the use of recursive/functional decomposition didn't seem appropriate.

Since we distribute the documents across the processes a priori, the decomposition is implicit, i.e, each process takes care of computing its own data (except the calculation of the global averages).

For the MPI version, we used data decomposition while calculating the distances between the documents and the cabinets, while changing documents from the cabinets and while calculating the partial averages of the cabinets.

For the OpenMP + MPI version, we use data decomposition in the previous mentioned tasks, but also while reading the input.

3 Synchronization concerns

The only synchronization concern regarding the MPI version is the communication, since we have overhead sending information to a process, or wait for its receive. Also, each process deals with its own memory, and therefore there are no synchronization concerns while computing data. Having said that, we tried to reduce to the most the number of communications.

It is important to mention that while profiling the code, we found out that sending chunks of documents to each process was the most heavy computational section on our code, this is, there was a great amount of time being wasted at synchronization. Since the celerity of the synchronization depends on the network bandwidth, we coped with this problem by sending the initial information as a char (1 byte), instead of sending it as a double (8 bytes).

Regarding the OpenMP + MPI version, we had the same concerns of the previous delivery, together with the concerns mentioned before.

4 Load Balancing

Since the structures we use are matrices and arrays, there weren't those many concerns about load balancing, because the structures themselves are symmetric. This way, each process has more or less the same amount of documents, so the workload is distributed evenly, this is, they do approximately the same amount of computations.

In the OpenMP + MPI version the idea is the same: the chunk of documents that each process has is distributed evenly for each thread to process.

5 Memory Requirements

In terms of memory requirements, each process has to store all the information about the cabinets averages and only *Number of Documents / Number of Processes* for the documents that belong to that process. This way, the memory usage changes with O(*Number of Cabinets + Number of Docs/Number Of Processes*) making the program more scalable and efficient.

6 Performance results

To see the performance of both the MPI and MPI+OpenMP versions, we ran the tests provided by the teachers: first with the sequential version, and then with the parallel versions. We excluded the first two tests, since they are relatively small too see any important results.

For the MPI version, we ran the tests with a varying number of hosts, and varying number of processes in those hosts. For the MPI+OpenMP version, we ran the tests for a varying number of hosts, and the same number of cores used in each host. Afterwards, we compared the times of both parallel versions against the sequential version and calculated the speed up.

In theory, if we run the MPI version with 1 process for N hosts, we won't have a speed up of N against the sequential version. There are many factors that determine the speed up (MPI initialization and finalization, communication between processes, accesses to memory, etc.), which can be shown in practical terms.

Since the communication overhead greatly degrades the speed up, we also took the times that the program took to execute, excluding the initial communication.

					Time (seconds)		Speed Up	
Cabs	Documents	Subs	Procs	Hosts	Total	Algorithm	Total	Algorithm
3	1000	50	1	1	0,1312	0,1130	-	-
4	100000	200	1	1	74,6082	71,3861	-	-
3	1000000	100	1	1	117,3687	103,4276	-	-
3	1000	50	4	4	0,1297	0,1167	1,0113	0,9678
4	100000	200	4	4	41,0905	35,2401	1,8157	2,0257
3	1000000	100	4	4	55,7786	26,0833	2,1042	3,9653
3	1000	50	8	4	0,1063	0,0867	1,2335	1,3030
4	100000	200	8	4	24,1598	18,6478	3,0881	3,8281
3	1000000	100	8	4	43,5649	15,5713	2,6941	6,6422
3	1000	50	16	4	0,1158	0,0990	1,1333	1,1416
4	100000	200	16	4	17,5843	12,2286	4,2429	5,8376
3	1000000	100	16	4	35,4040	8,2953	3,3151	12,4682
3	1000	50	16	16	0,2027	0,1730	0,6473	0,6530
4	100000	200	16	16	18,5976	11,9669	4,0117	5,9653
3	1000000	100	16	16	41,0093	8,2201	2,8620	12,5822

Table 1: Execution of the MPI version with teacher tests

After careful analysis of the tables, we can see that as the number of documents increases, the speed up is also bigger. However, the initial communication cost is so high, that if we increase the number of hosts the speed up decreases dramatically. If we increase the number of processes for host, we can see that speed up increases, since there isn't communication between different machines.

If we only look to the algorithm execution speed up, it is clear that it is nearer to its ideal values than if we take into accont the global times, even though the speed up is not that close to them since there are communications between machines during the execution of the algorithm.

It's very important to mention that the machines where the tests were executed, were also used by other students, which may have compromised the times taken. Any presented values may not match the actual performance of our program.

In order to check the performance of our MPI version for larger computation times, we ran some tests with a fixed number of *one million* documents and a varying number of cabinets and hosts, one process for host. Below, we show a graphic with the speed up of our program for different parameters.

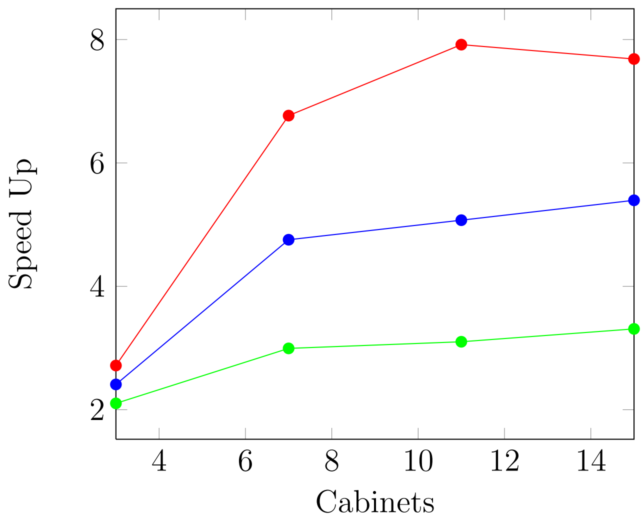

Figure 1: Speed up of the MPI version with: 4 Hosts; 8 Hosts; 16 Hosts;

If we look at the graphic, we can state that if we increase the number of cabinets, this is reflected in an increase of the speedup. This is due to the fact that we increase the computational time of the program, while maintaining the initial communication cost.

There is a strange value for the test with 16 hosts and 14 cabinets, which may be explained if the machines where the tests were run were overloaded at that time. The execution times of the MPI version and its algorithm part for this graphic can be seen in the *Table 2* and *Table 3* of the *Appendix*.

You can also see the execution times of the MPI+OpenMP version in the *Appendix*, both for the teacher tests and our tests. In the presented results for this version (see *Table 5* and *Table 6*), we can see that the results are the expected, this is, with an increasing speedup for an increase of the ammount of processing units.

Even though the speed up is increasing, there is still communication between the hosts for the MPI part of the program, and for this rreason the speed up isn't nearer the ideal. Also, despite the facts that we ran were sufficient to see good results, we could have seen better results if we increased the parameters of the tests.

Appendix

			Time (seconds)			
Cabinets	Documents	Subjects	Serial	4 Hosts	8 Hosts	16 Hosts
3	1000000	100	117,8446	56,0338	48,9012	43,3735
7	1000000	100	419,6656	140,1569	88,2543	62,0086
11	1000000	100	749,7803	241,7335	147,8210	94,6985
15	1000000	100	1021,8835	308,7253	189,4325	132,9741

Table 2: Approximate execution time of the MPI version with our tests

			Algorithm Time (seconds)			
Cabinets	Documents	Subjects	Serial	4 Hosts	8 Hosts	16 Hosts
3	1000000	100	101,3446	28,9900	15,5512	7,6750
7	1000000	100	403,1656	109,1569	56,2543	28,1086
11	1000000	100	733,2803	211,7335	112,9376	60,6685
15	1000000	100	1005,3835	277,1153	154,2927	98,6254

Table 3: Approximate execution time of the algorithm in the MPI version with our tests

			Speed Up			Algorithm Speed Up		
Cabinets	Documents	Subjects	4 Hosts	8 Hosts	16 Hosts	4 Hosts	8 Hosts	16 Hosts
3	1000000	100	2,1030	2,4098	2,7169	3,4958	6,5168	13,2043
7	1000000	100	2,9942	4,7551	6,7678	3,6934	7,1668	14,3431
11	1000000	100	3,1016	5,0722	7,9175	3,4632	6,4927	12,0866
15	1000000	100	3,3100	5,3944	7,6847	3,6280	6,5160	10,1939

Table 4: Speed up of the MPI version with our tests

					Time (seconds)		Speed Up	
Cabs	Documents	Subs	Hosts	Cores	Total	Algorithm	Total	Algorithm
3	1000	50	4	16	0,0902	0,0744	1,4528	1,5176
4	100000	200	4	16	16,2573	10,8032	4,5892	6,60752
3	1000000	100	4	16	35,8678	8,2729	3,2722	12,5019
3	1000	50	8	32	0,1434	0,1247	0,9143	0,9054
4	100000	200	8	32	12,5186	6,3621	5,9597	11,2205
3	1000000	100	8	32	35,8111	4,5592	3,2774	22,6851
3	1000	50	16	64	0,1734	0,1543	0,7561	0,7317
4	100000	200	16	64	11,9361	5,3663	6,2505	13,3026
3	1000000	100	16	64	35,7195	2,7921	3,2858	37,0420

Table 5: Execution of the version OpenMP + MPI with teacher tests

					Time (seconds)		Speed Up	
Cabs	Documents	Subs	Hosts	Cores	Total	Algorithm	Total	Algorithm
7	1000000	100	4	4	61,0891	31,6870	6,8697	13,2440
7	1000000	100	8	4	57,9386	23,8969	7,2432	17,5614
7	1000000	100	16	4	55,4041	17,9386	7,5746	23,3945

Table 6: Execution of the OpenMP + MPI version with our tests