Instituto Superior Técnico

MEEC

Parallel and Distributed Computing

OpenMP
The Ball Algorithm

Grupo 32:
Guilherme Mascaranhas - nº 87011
João Santos - nº 87039
Miguel Martins - nº 87086

Contents

1 Our Approach

1.1 Approach used for parallelization

Due to the nature of the project, two different approaches were considered: having multiple threads processing a single node or having threads processing different nodes at the same time. Using multiple threads in the processing of a single node while smoothing the overheads' delay proved to be difficult to implement. For this reason, the option to have threads processing different nodes at the same time was chosen.

1.2 Decomposition was used

Given the nature of the algorithm, whenever each ball divides its points into the 'left' and 'right' groups, two different threads can run each group without any dependencies. Having this in mind, we developed our algorithm in such way that if there were threads available when computing a cluster, both of its sub-trees would be run in parallel. Due to the limited amount of available threads, the code will run in a sequential manner while waiting for those threads to become available.

1.3 Synchronization concerns

As described in 1.1 each group (left and right in the same cluster) can be processed in parallel. However, we use a global variable that indicates the number of available threads, and, since multiple threads could be reading and writing it at the same time, we used "pragma omp atomic" to avoid any synchronization problems.

1.4 Load balancing

By utilizing tasks to distribute the work load through the threads, and because of the way the project is constructed, we manage to create a balanced workload between all the threads that are working in the different nodes of the binary tree. Since the distribution of the nodes is done breath wise and not depth wise, we assure that the threads are getting groups of points with approximately the same complexity.

2 Performance Results

Our program was run for the different input files available, with different number of available threads. It's running times were saved in Table 1 and further plotted

in Figure 1. As expected, for all the bigger input files, the speedups would increase as the number of available threads also increases. As we could predict, this relation doesn't occur for the two smaller files due to the fact that the time spent with synchronization and load balancing isn't recovered due to the simplicity of the algorithm for smaller data sets.

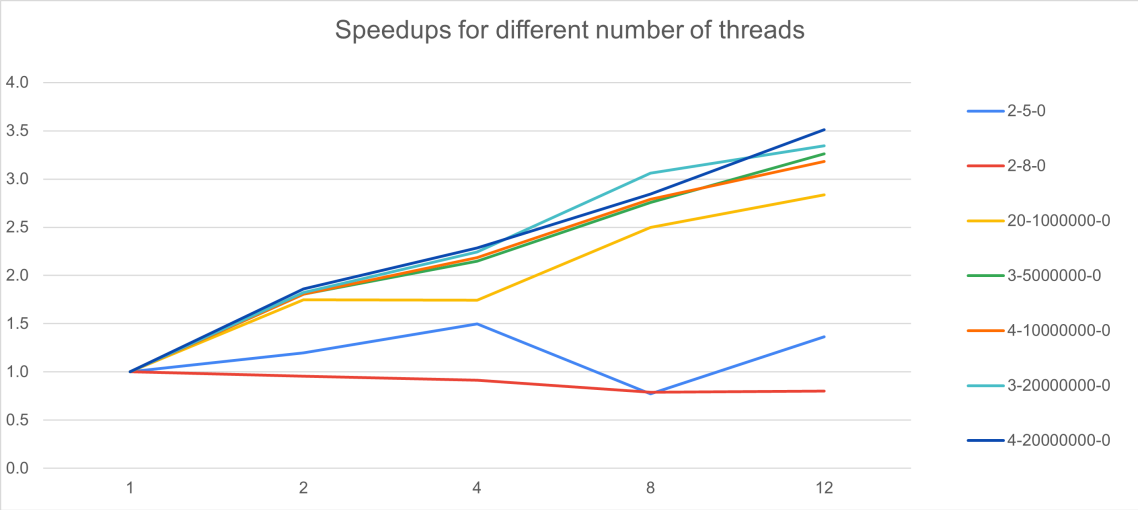

Figure 1: Increase in performance based on the number of threads

	nº de Threads								
	1	2		4		8		12	
	Time	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup
2-5-0	0.00	0.00	1.20	0.00	1.50	0.00	0.77	0.00	1.36
2-8-0	0.00	0.00	0.96	0.00	0.91	0.00	0.79	0.00	0.80
20-1000000-0	3.96	2.27	1.75	2.27	1.74	1.59	2.50	1.40	2.83
3-5000000-0	12.43	6.87	1.81	5.79	2.15	4.51	2.76	3.81	3.26
4-10000000-0	28.75	15.92	1.81	13.16	2.19	10.30	2.79	9.04	3.18
3-20000000-0	58.58	32.14	1.82	26.12	2.24	19.15	3.06	17.53	3.34
4-20000000-0	64.78	34.81	1.86	28.35	2.29	22.77	2.85	18.45	3.51

Table 1: Increase in performance based on the number of threads