INSTITUTO SUPERIOR TÉCNICO

Parallel and Heterogeneous Computing Systems

Assignment 3

85229 – Rui Cavalheiro
93179 – Santiago Quintas
94030 – Luís Canas

Group 11
Shift LS03

November 4, 2022

1. Implementing Page Rank

To implement our paralleled Page Rank algorithm we first did a serial version of it. We start by initializing the nodes so that each node has the same probability of every other node, after this we go through every node in the graph producing the new score by computing the set of operations described below:

1. score_new[node] = sum over all nodes reachable from incoming edges;
2. score_new[node] = (damping * score_new[node]) + (1.0-damping) / numNodes;
3. score_new[node] += sum over all nodes v in graph with no outgoing edges.

We only stop changing the score of the different nodes when the global difference, (1), is lower than a certain threshold.

$$global\_Diff = \sum_{i=0}^{numNodes} [abs(score\_new[i] - score\_old[i])]; \quad (1)$$

To achieve parallelism we use openMP, we apply it when we initialize the equal probability to each node, also to the for loop going through every node in the graph and lastly when we calculate the global difference and when we copy the updated score vector to the solution array. We made use of the following commands:

```
#pragma omp parallel for schedule(dynamic,32)
#pragma omp parallel for reduction(+:sum) schedule(dynamic,32)
#pragma omp parallel for reduction(+:global_diff) schedule(dynamic,32)
#pragma omp parallel for schedule(dynamic,32)
```

Worth noting that the schedule(dynamic,32) was used because it achieved the best results during testing. The obtained execution times for 4 graphs, each with a different amount of nodes, (68 million, 117 million, 200 million and 500 million) can be seen in Table 1

Graph	Ref Time (s)	Exe time (s)
soc-livejournal1_68m.graph	3.6	4.00
com-orkut_117m.graph	4.8	4.95
rmat_200m.graph	24.3	22.67
random_500m.graph	51.9	53.28

Tab. 1 – PAGERANK time Table

Threads	Exec time (s)	Speedup
1	6.0438	1.0000
2	5.0920	1.1869
4	4.6134	1.3100
8	3.9990	1.5113

Tab. 2 – PAGERANK speedup Table for soc-livejournal1_68m.graph

With the results of Table 1 we can conclude that the results obtained are close to the reference time. The slight differences may be explained by two factors: first our parallelization is not optimal and could still be improved that can be observed in Table 2; secondly the load on the CUDA machine when we were testing could have been significantly higher than when reference timings were obtained, increasing the execution time in comparison. Furthermore, there are some load balancing issues to the nature of the problem (graphs) limit or speedup.

2. Parallel Breadth-First Search

2.1. ”Top Down” BFS

The top down version of this searching algorithm starts at the top of the graph and works its way down, it does this by creating a frontier set which contains the nodes to be visited next, every time a node is visited it puts the nodes destine to its outgoing edges in the frontier.

We added parallelism using openMP instructions, we do it when we initialize the nodes in the graph as NOT-VISITED, and inside the step function in the for loop that goes through every node in the frontier, using a parallel for schedule(dynamic,32) which gave the best results. This means that every node in the frontier is adding its ”children” to the new-frontier simultaneously, race condition can occur when different nodes try to increment the new-frontier count simultaneously. To avoid this problem we use an atomic capture to increment the count and store it in private thread variable (instead of using a critical which increases overhead).

Graph	Ref Time (s)	Exec time (s)
grid1000x1000.graph	0.02	0.03
soc-livejournal1_68m.graph	0.20	0.39
com-orkut_117m.graph	0.23	0.21
rmat_200m.graph	2.29	2.94
random_500m.graph	5.13	5.88

Tab. 3 – Top Down time Table

Threads	Exec time (s)	Speedup
1	0.37	1.00
2	0.28	1.35
4	0.22	1.70
8	0.21	1.79

Tab. 4 – Top Down speedup Table for com-orkut_117m.graph

If we observe Table 3 we can infer that the results obtained are close to the reference time. The differences may be explained by two factors: first our parallelization is not optimal and could still be improved that can be observed in Table 4; secondly the load on the CUDA machine when we were testing could have been significantly higher than when reference timings were obtained, increasing the execution time in comparison. We also don't achieve perfect speedup by reasons mentioned previously, as well as because of the nature of the problem (graph causes workload imbalance).

2.2. "Bottom-up" BFS

In this version of the searching algorithm instead of going through the nodes in the frontier we go through every node in the graph and for every node we check if it should be added to the frontier, the way it will work will be if an unvisited node has an incoming edge from a node already present in the frontier then we add it to the new-frontier set, so for every node we go through every of it's incoming edges to see if any of them belong to a node already in the frontier.

We applied parallelism in a similar way to the Top Down method, so every node in the graph checks simultaneously if they should be added to the frontier or not by using a parallel for with schedule(dynamic,32), while also having in mind the race condition when increasing the count of the frontier for which we use an atomic capture (instead of using a critical which increases overhead).

Graph	Ref Time (s)	Exe time (s)
grid1000x1000.graph	1.52	3.16
soc-livejournal1_68m.graph	0.14	0.70
com-orkut_117m.graph	0.15	0.31
rmat_200m.graph	1.98	3.03
random_500m.graph	19.67	19.33

Tab. 5 – Bottom Up time Table

Threads	Exec time (s)	Speedup
1	0.75	1.00
2	0.53	1.42
4	0.38	1.98
8	0.31	2.41

Tab. 6 – Bottom-up speedup Table for com-orkut_117m.graph

By analyzing Table 5 we can infer that the results obtained are correct but higher than the reference time. The differences may be explained by three factors: first our parallelization could be not optimal and still be improved which can be observed in Table 6 as well as by the discrepancy of times between execution and reference; secondly the load on the CUDA machine when we were testing could have been significantly higher than when reference timings were obtained, increasing the execution time in comparison; thirdly our serial version is not sufficiently optimized to produce the best timings. Note that we don't achieve perfect speedup by reasons mentioned before, as well as due to the nature of the problem (graph causes workload imbalance).

2.3. Hybrid BFS

Both versions have their good and bad scenarios, in this hybrid version we try to bring together the best of both algorithms, the way it works is in certain parts of the search we apply top down and in other parts we apply bottom up, this decision is made by having in mind what part of the search we are currently in. If we are in the very beginning or end, the top down version is the way to go because the frontier is still very small and the algorithm computes faster as we can see in Table 10. It is indeed slower when the frontier starts increasing in size. By contrast if we see the Bottom up version we can observe that for larger frontiers the algorithm is faster and for smaller frontiers the computation is slower. Our objective was to find this threshold to decide between the two algorithms using the equation:

$$FrontierSize < \frac{NumberOfNodes}{\alpha} \tag{2}$$

We selected the alpha by trial and error and the best times were around the value $\alpha = 24$. Our results are presented next:

Graph	Ref Time (s)	Exe time (s)
grid1000x1000.graph	0.52	0.03
soc-livejournal1_68m.graph	0.09	0.35
com-orkut_117m.graph	0.05	0.09
rmat_200m.graph	1.22	1.88
random_500m.graph	3.42	4.09

Tab. 7 – Hybrid time Table

Threads	Exec time (s)	Speedup
1	0.06	1.00
2	0.11	0.52
4	0.10	0.59
8	0.09	0.65

Tab. 8 – Hybrid speedup Table for com-orkut_117m.graph

Threads	Exec time (s)	Speedup
1	5.49	1.00
2	4.39	1.25
4	4.28	1.28
8	4.06	1.35

Tab. 9 – Hybrid speedup Table for random_500m.graph

Frontier	Top Down	Bottom Up
1	0.0000	0.1039
46	0.0000	0.3149
2155	0.0025	0.2898
101386	0.0610	0.2075
1174905	0.3241	0.1483
2166715	0.2781	0.0612
811128	0.0578	0.0230
122063	0.0071	0.0157
17877	0.0010	0.0142
3176	0.0001	0.0141
683	0.0000	0.0140
165	0.0000	0.0140
28	0.0000	0.0140
15	0.0000	0.0140
4	0.0000	0.0140

Tab. 10 – Top down VS Bottom Up step time comparison for soc-livejournal1_68m.graph

By analyzing Table 7 we can infer that the results obtained are correct but higher than the reference time. Note that the results are conditioned by the implementations of both BFS and The differences may be explained by three factors: first our parallelization could be not optimal and still be improved which can be observed in Table 8 and 9 as well as by the discrepancy of times between execution and reference; secondly the load on the CUDA machine when we were testing could have been significantly higher than when reference timings were obtained, increasing the execution time in comparison; thirdly our serial version is not sufficiently optimized to produce the best timings.

There are other noteworthy things, for example In table 8 we notice speed downs while in table 9 we notice speed ups. This is due to our Hybrid algorithm using the best characteristics of each of the previous methods. those methods in themselves take better advantage of the paralelization when they function with higher loads (where they preform the worst the paralelization is best). this means that by only using them when they preform the best we minimize the impact of the using of threads and we end up getting worse results. this is evident when we compare the 68 million node graph and the 500 million node graph, by increasing the load we take advantage of paralelization. Finally we can infer that we don't achieve perfect speedup by reasons mentioned before as well as due to the nature of the problem (graph causes workload imbalance).

3 Conclusion

There is a common issue that plagues the whole project, PageRank and BFS, which is the load balancing issue due to the nature of the graphs. The graphs are not consistent and there are some nodes with a lot of edges and some nodes without and this results in some threads taking higher loads than others resulting in sub-par speed ups.

However our results are all correct and the times are close to the reference provided.