RESULTS OF THE EXPERIMENTS

VINCENT RÉBISCOUL

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In this report, we are going to try several configurations and with Simgrid, we will evaluate our programs and compare the sequential algorithms with the parallel algorithms. The topology I used for this is a clique. The platform file and the hostfile were generated using a modification of the script that was given to us during TP 2 and was originally meant to generate a ring topology. I precise that I will not use ring topology because I think it is not necessary (there is no interest to run a graph algorithm on a ring). Moreover, I find it too bad that if one link is broken, the network can not work.

1 METHODOLOGY

We are going to separate the analysis of the Kruskal's algorithm and Prim's algorithm. To do the analysis, we will test several cases. First I note $G = (V, E)$ our graph and $n = |V|$ and $m = |E|$. We will test three cases for m:

- $m = 2 \times n$ (linear)
- $m = \frac{n^{\frac{3}{2}}}{2}$

And I will test those configurations for $n = 500$ (my laptop can not handle bigger graphs, the generation of a graph never ends). I will test this with 4 cores and 10 cores. I will take 500 for the maximum weight. Indeed, this value should not have a lot of consequences on the speed of the algorithm (see appendix A 4). I will also test a latency of $1\mu s$ and $100\mu s$. I will call the linear graph (with $m = 2 \times n$) the linear graph and the graph with $m = \frac{n^{\frac{3}{2}}}{2}$ the $\frac{3}{2}$ graph.

2 RESULTS OF THE TESTS

2.1 Sequential algorithms

Table 1: Speed of the sequential algorithms (in seconds)

	Linear	3/2
Prim	5×10^{-3}	5.3×10^{-3}
Kruskal	2×10^{-3}	3.4×10^{-3}

2.2 Parallel Prim's algorithm

Table 2: Speed of the parallel Prim's algorithm (in seconds)

	1µs/4 cores	1µs/8 cores	100µs/4 cores	100µs/8 cores
Linear	3×10^{-3}	4×10^{-3}	3×10^{-1}	4×10^{-1}
3/2	3×10^{-3}	4×10^{-3}	3×10^{-1}	4×10^{-1}

2.3 Parallel Kruskal's algorithm

Table 3: Speed of the parallel Kruskal's algorithm (in seconds)

	1µs/4 cores	1µs/8 cores	100µs/4 cores	100µs/8 cores
Linear	9×10^{-6}	1.3×10^{-5}	8.6×10^{-4}	1.3×10^{-3}
3/2	9.3×10^{-6}	1.4×10^{-5}	8.5×10^{-4}	1.2×10^{-3}

First, we can see that there is little different between the linear graph and the 3/2 graph. That could be expected because in every cases, because the graph is represented by a matrix, the complexity will always depends on n^2 and not on m. This is why we do not have a lot of differences between the linear graph and the 3/2 graph, the number of edges has little effect on the number of steps that the algorithm will do.

Moreover, we see that when running the parallel algorithms on 8 cores, it is a bit slower that on 4 cores. I think this comes from the fact that my processor has only 4 cores and when running the algorithms on 8 cores, I do not gain any computation time but I loose time in communications between the "virtual" processors. Now we will compare the sequential algorithms. I will just study the linear graph because we saw that the results were similar with the 3/2 graph. I will only study the case with 4 cores because this is when my computer is the more efficient.

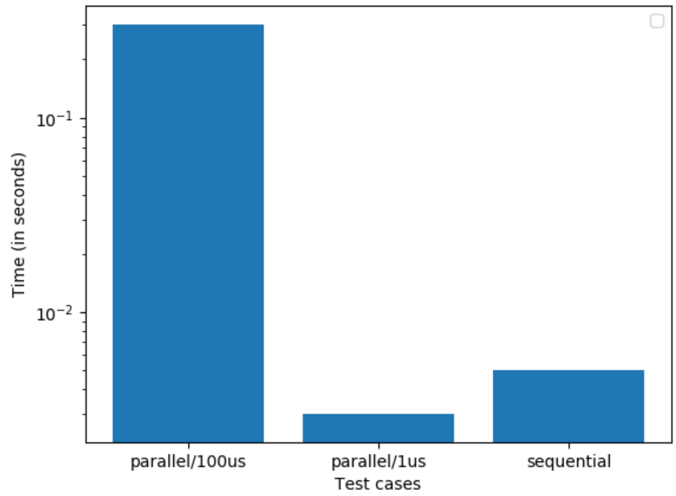

Figure 1: Comparison of sequential Prim's algorithm and parallel Prim's algorithm

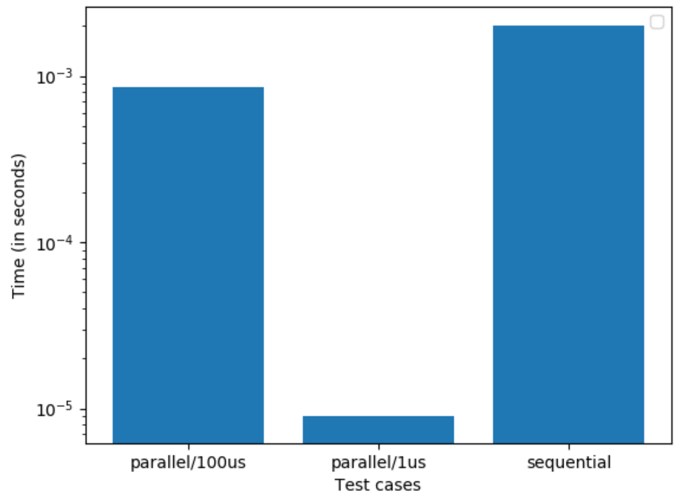

Figure 2: Comparison of sequential Kruskal's algorithm and parallel Kruskal's algorithm

3 COMPARISON BETWEEN SEQUENTIAL AND PARALLEL ALGORITHMS

3.1 Sequential Prim VS Parallel Prim

As we can see on figure 1 the sequential algorithm is a little worst than the parallel one with a latency of $1\mu s$ but a lot better than the parallel one with a latency of 100μ (please note that the scale is logarithmic). One can conclude that the latency has a lot of influence on the efficiency of the parallel Prim's algorithm. This is probably due to the fact that when all the processors have their candidates, they have to send it the processor 0 that has to handle a lot of communications at one time, and during this time the other processor are not working.

3.2 Sequential Kruskal VS Parallel Kruskal

As we can see on figure 2, for a latency of 1µs, the parallel algorithm is a lot better than the sequential one. For a latency of 100µs, the parallel algorithm is a little better than the sequential one. First conclusion, the efficiency of the parallel algorithm is a lot better. I think we can explain this because in the parallel Kruskal, a processor wait little for another one. Indeed a processor do some computations send or receive information and then continue working but it does not need to wait for an answer of the processor it communicated with. Also, the algorithm is a lot less "centralized". There is no processor that have to regularly get information from everyone, a processor only work with one close neighbour. I think this is why the parallel algorithm in Kuskal is a lot better.

4 CONCLUSION

To conclude, we have seen that the communications have a strong impact on the efficiency of the distributed algorithms. Kruskal seems to be the the faster probably because of the face that we do not have to send something to one processor regularly. But to be really efficient, the communications need to have a low latency.

	$W = 500$	$W = 2$
Prim parallel	3×10^{-3}	3×10^{-3}
Kruskal parallel	9×10^{-6}	9.1×10^{-6}
Prim sequential	5×10^{-3}	5×10^{-3}
Kruskal sequential	2×10^{-3}	2×10^{-3}

Table 4: Comparison of the computation time (in second) of the different algorithms on the same graph but one with the maximal weight being 500 and the other one being 2

APPENDIX A

I want to check that the maximal weight as little incidence on the computation time. I take a graph with $n = 500$ and $m = 1000$, with the maximal weight $W = 500$ and an other identical graph but with $W = 2$ and I compared the computation time. I used Simgrid with 4 processors for the parallel algorithms, and bandwith of 100Gpbs and a latency of 1µs.

One can see one table 4 that this has no influence on the final speed. This is normal because the equality between two edges does not make more computations.