Pagerank approximation on GPU

High Performance Processors and Systems (UIC 569)

Prof.s Donatella Sciuto, Marco D. Santambrogio

Author
Francesco Sgherzi - *915377*

Tutors
Alberto Parravicini
Rolando Brondolin

Contents

1 Introduction

Since the launch of the very first GPU as we know it today in 1999 [3] this market has seen continuous improvements due to the need of ever growing processing power in order to render increasingly complex scenes.

NVIDIA would allow the general public to harness the power of the parallel computing only later, in 2007, with the release of the *Compute Unified Device Architecture* (*CUDA*) framework, thus giving the developer opportunity to exploit *Data Level Parallelism* in other frames of application (e.g scientific calculus) and thus giving birth of a new era of *General Purpuse GPU*

computing.

The *Pagerank* algorithm [4] was initially proposed in 1998 by *Larry Page* and *Sergey Brin* as a way to rank web pages based on the *number* and the *importance* of pages that have an outward link to the to-be-ranked page.

Nowadays, the sheer growth of the number of web pages requires the use of different techniques in order properly index them in a reasonable amount of time. That's why we propose two implementation of the aforementioned algorithm using the *CUDA* framework and applying different *approximation paths* in order to achieve lower execution times while still yielding correct results.

1.1 Goal of the project

The goal of the project is to explore different approaches in *approximating* the computation of the *Pagerank* algorithm while testing the their limits with respect to the following graph categories: *Erdős–Rényi*, *Small World*, *Scale Free*.

The goal has been achieved by building two main variations of the algorithm (a floating point one and a fixed point one) and applying approximation both on the convergence criteria and on the computation (2.3.1). After that, the results are evaluated with respect to the indexing and value error, the overall execution time and iterations to convergence.

2 Implementation

2.1 Algorithm Overview

A high level implementation of the algorithm, as presented in the paper [4], is the following:

$R_0 \leftarrow S$
loop:
    $\mathrm{R}_{i+1} \leftarrow \mathrm{AR}_0$
    $d \leftarrow \|R_i\|_1$ - $\|R_{i+1}\|_1$
    $R_{i+1} \leftarrow \mathrm{R}_{i+1} + dE$
    $\delta \leftarrow \|R_{i+1} - R_i\|$
while $\delta > \epsilon$

Where:

- S represents *Almost every vector over Web Pages*

- E represents *The distribution of web pages that a random surfer periodically jumps to.*
- A represents the adjacency matrix for the web graph.

The chosen implementation was the *Power iteration* method, which uses the value α in place of the matrix E in order to express the probability of a surfer to jump randomly to another page and handles dangling nodes (nodes with no outward arcs) by treating them separately and then summing their contribution, uniformly, to all nodes.

$$
\begin{aligned}
&R_0 \leftarrow \langle \tfrac{1}{N}, \tfrac{1}{N}, ... \rangle \\
&loop: \\
&\quad n\_iter \leftarrow n\_iter + 1 \\
&\quad R_{i+i} \leftarrow spmv(H, R_i) \\
&\quad dangling\_contribution \leftarrow R_i D \\
&\quad scaling\_factor \leftarrow \tfrac{1-\alpha}{N} + (\tfrac{\alpha}{N} * dangling\_contribution) \\
&\quad R_{i+i} \leftarrow \text{axpb}(\text{R}_{i+1}, \alpha, scaling\_factor) \\
&\quad \delta \leftarrow \text{compute\_error}(\text{R}_{i+1}, R_i) \\
&while\ \delta > \epsilon \text{ and } n\_iter \leq max\_iter
\end{aligned}
$$

Where:

- H is the graph in its CSC (*Compressed Sparse Column*) representation.
- α is the dampening factor, otherwise known as the probability of a web surfer to randomly jump to another page.
- N is the number of pages
- D is the dangling bitmap, if a node is *dangling* the value in its position in the vector is 1, 0 otherwise.
- *spmv* is the sparse vector-matrix multiplication
- *axpb* is the scale and shift operation
- *compute_ error* represents a placeholder for the different convergencence criteria used in our analysis(2.3.1)

2.2 Data Structure

Since the main points of application of the *Pagerank* algorithm are *web-like* graph which are inherently sparse the use of an adjacency matrix would have resulted in a huge amount of memory used to store unnecessary information, that is, given any pair of nodes, the information of them not being linked is not relevant for the algorithm.

Additionally, since the data need to be accessed in *row-wise* order, we couldn't exploit spacial locality which would have resulted in a non negligible memory overhead.
Due to this, we opted for the *CSC* format to store the adjacency matrix. The *CSC* format stores the elements of the matrix in 3 separate vectors:

- vector of the non zero elements of the adjacency matrix, read in *column major order*
- vector containing a 0 followed by the number of non zero elements for each column summed with the number of non zero elements of the previous column
- vector of the column indices of each non zero element

For instance, the following matrix:

$$\begin{pmatrix} 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 1 & 0 & 0 \\ 0.33 & 0.33 & 0 & 0 & 0.33 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Gets converted to the following representation:

$$\begin{aligned} values &= \langle 0.33, 1, 0.33, 1, 0.5, 0.5, 0.33 \rangle \\ non\_zero &= \langle 0, 2, 3, 4, 5, 7 \rangle \\ col\_idx &= \langle 2, 4, 2, 1, 0, 0, 2 \rangle \end{aligned}$$

The use of this representation is not only much more memory efficient for sparse graphs, since it is linear in the number of edges (instead of being quadratic in the number of nodes) but allows also to better exploit spacial locality due to the fact that values in the vectors are naturally stored in contiguous memory locations.

2.3 Approximation

2.3.1 Floating Point

The first approach we explored was to build a simple *low-level* implementation of the algorithm, using `float` as the base data type, and then approximate both on the *convergence criterion* and the computation.

The baseline convergence criterion was to check, for each individual node, if the absolute difference between its current and previous *Pagerank* value was

below the tolerance threshold.

The next optimization involved changing the convergence criterion, instead of checking each node individually we computed the *l2-norm* of the vector and then examined if the value was under the tolerance threshold.

In order to achieve an *approximated* computation we built a *bitmap* vector where, for each node position, there were either a `true` or `false` indicating if that node had already reached convergence. If, for a given node, convergence was already reached we simply skip the *Pagerank* computation for that node.

2.3.2 Fixed Point

The second approach involved building a *fixed point* version of the *Pagerank* algorithm, using `unsigned long long` as the base data type, and then approximate the *convergence criterion* and the computation using the techniques described in section 2.3.1.

We opted for an unsigned *Q1.63* representation since:

- The output of our *Pagerank* computation is a vector of elements bounded in $[0, 1]$.
- Albeit extremely unlikely, it is still possible that a node has a *Pagerank* value of one, therefore ruling out the possibility of using a *Q0.64* representation.

In order to perform this approximation, we couldn't use the usual *product* operation as it would have resulted in an incorrect result since the multiplication between 2 *n bits* numbers needs *2n* bits for the result.
The following approaches were explored:

```
unsigned ull fixed_mult (unsigned ull x, unsigned ull y) {
    return to_fixed(to_double(x) * to_double(y))
}
```

```
unsigned ull fixed_mult (unsigned ull x, unsigned ull y) {
    return ((x >> (SCALE / 2)) * (y >> (SCALE / 2)))
}
```

Due to the fact that the second implementation tended to loose too much precision in the scaling operation and was only marginally faster than the first one, we decided to opt for the first version.

This allowed us to gain another bit in the fractional part since the second algorithm required the scaling factor to be even in order to achieve the best resolution.

3 Experimental Results

3.1 Benchmarking Methodology

Initially, the different implementations were compared against the implementation provided by the python module `networkx`. From the three provided by the aforementioned framework, `pagerank_scipy` has been chosen since it is a fairly optimized CPU version and thererfore yielded the best execution times amongst the other two.

Successively, since the execution times on CPU were still too high, in order to make a proper comparison we decided to check the optimized implementations against a non optimized GPU one.
The *Pagerank* implementation of `networkx` was still used as a reference to assess the correctness of the results.
The fixed and floating point implementations were tested in terms of:

- execution time: measured considering the time needed to reach convergence
- correctness: measured using the percentage of values in the wrong position in the final *Pagerank* iteration
- iterations to convergence and convergence error: measured in terms of
- displacement of *Pagerank* values: measured in terms of absolute distance of each node in the final *Pagerank* vector to that same node in the baseline implementation.

A note on correctness: the *acceptable values* hugely depend on the frame of application of the algorithm. If applied, for instance, on a search engine performing a query, even a high error, let's say, *50 %*, could still be considered acceptable if the errors were located in the lower part of the ranking. On the other side, a high error could still be generated by a single page being wrongly indexed thus shifting all the subsequent pages off by one place, in this case the ranking should be considered mostly correct, since the relative position of the pages is still preserved.

Being a only a partial measure, we decided to pair it with the *displacement of pagerank values* in order to have a deeper comprehension of the correctness of the implementation.
The *Pagerank* algorithms were run with the following parameters:

- $\alpha = 0.85$
- $\epsilon = 0.0$
- $max\_iter = 200$

3.2 Benchmarks

The graph were created using the following functions of the `networkx` framework:

- `fast_gnp_random_graph` for the Erdős-Rényi graph [2]
- `scale_free_graph` for the Scale Free graph [1]
- `watts_strogatz_graph` for the Small World graph [5]

Graph Type	Number of nodes	Average out degree
Erdős-Rényi	10000	~ 3
Scale Free	10000	~ 2.5
Small World	10000	~ 4

Note: In the following charts all variants of the two algorithm are shown for execution times, whereas only the best performer in the two category (floating and fixed) is shown for the other metrics.

3.3 Erdős–Rényi graph

[2] The Erdős–Rényi model is a graph model where all nodes have an equal probability to create an arc to another node, independently of other arcs, that is, the probability of an edge creation is uniform towards all nodes.

3.3.1 Execution Times

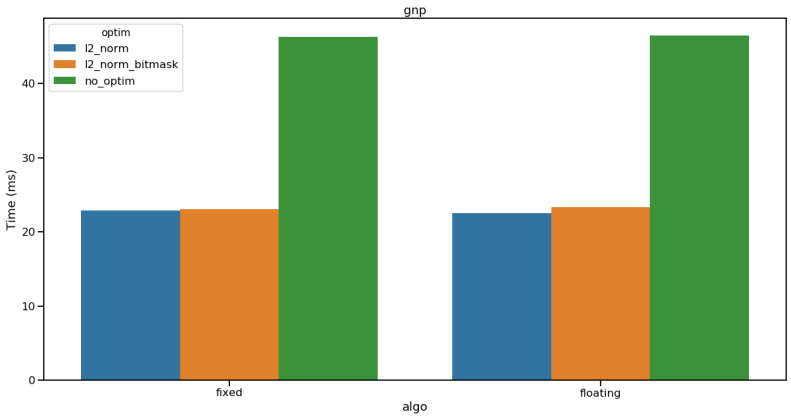

Figure 1: Execution time of every implementation

A noticeable time improvement is shown with the adoption of a more relaxed convergence criterion, the *l2-norm*, whereas the addition of the *bitmask* slightly decreases performances due to the overhead introduced by the branching required to skip the computation of the *Pagerank* value for a node if it has already reached convergence.
Overall, the best performers in the floating point and fixed point category (in both cases the *l2-norm* variant) achieve a 2.06x and a 2.01x speedup, respectively, compared to the non optimized version and an average speedup of 4.79x over the baseline.

3.3.2 Correctness

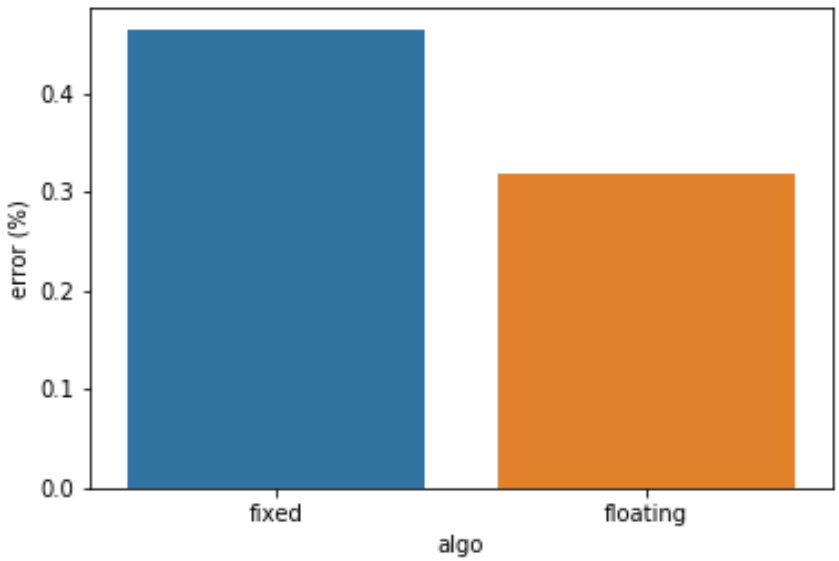

Figure 2: Percentage of wrong *Pagerank* ranks of the best performer of every implementation (*l2-norm*)

A small error in the ranking can be observed in both variants of the algorithm, this is mostly due to the fact that the *l2-norm* allows for a much faster convergence (with respect to the number of iterations) than individually checking every node. Nonetheless, the ranking error is still negligible (the worst performer, fixed point, assigned the wrong position to the node in than 500 times over 100000).

3.3.3 Iterations to convergence and convergence error

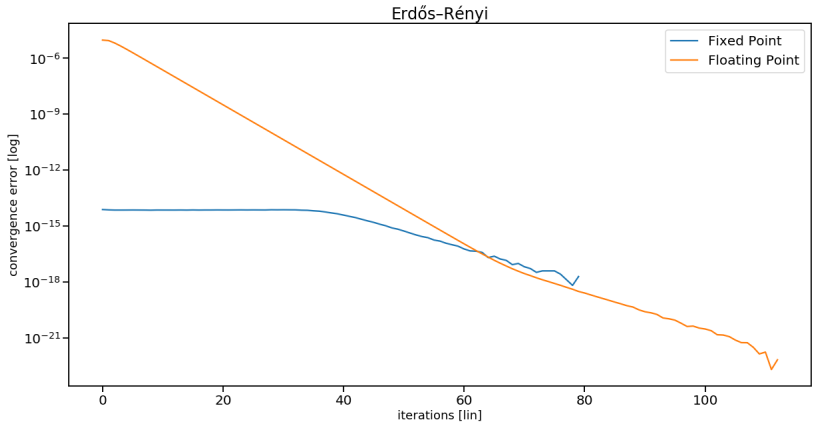

Figure 3: Rate of convergence of the best performer of every implementation (*l2-norm*)

The fixed point representation allowed the algorithm to converge much faster than the floating point version, this is due to the fact that the first isn't able to distinguish the variation with respect to the previous iteration after a earlier than the floating point version. Additionally since both representations achieved almost the same error in the ranking, this chart shows that the last iterations are only needed to lower the convergence error but don't produce meaningful changes in the ranking.

3.3.4 Displacement of Pagerank values

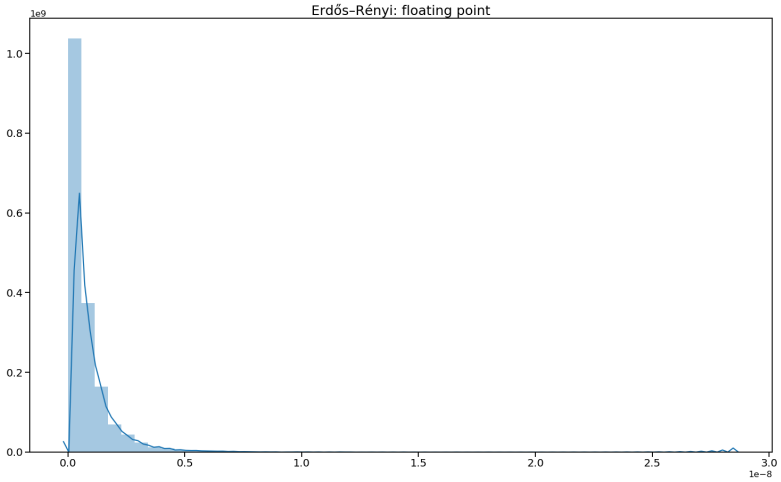

Figure 4: Distribution of displacement error of the best floating point performer (*l2-norm*)

- **Mean**: $2.23 * 10^{-6}$
- **Variance**: $2.08 * 10^{-12}$

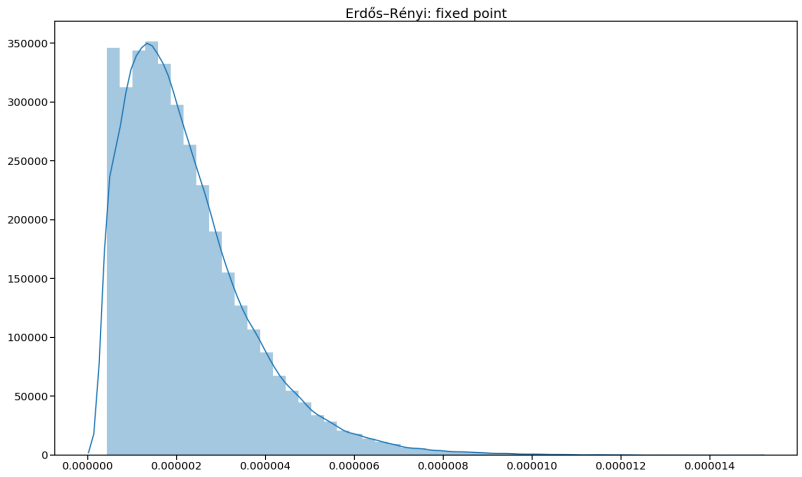

Figure 5: Distribution of displacement error of the best fixed point performer (*l2-norm*)

- **Mean**: $7.36 * 10^{-10}$
- **Variance**: $1.08 * 10^{-18}$

While the distribution of the displacement in terms of *Pagerank* values is extremely skewed, which is to be expected, fixed point shows a much higher variance.

3.4 Small World

[5] The Small World network is a type of graph where nodes tend to be linked to other in clusters, therefore yielding a small node-to-node average distance. Many real world graphs, like social networks, exhibit a small-world like behaviour.

3.4.1 Execution Times

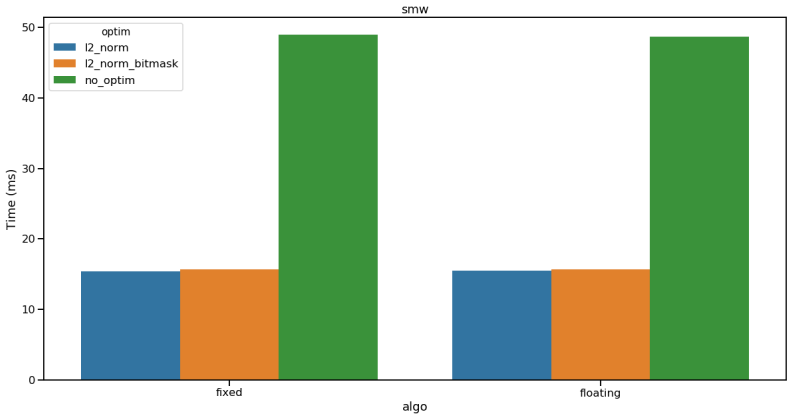

Figure 6: Execution time of every implementation

A wider decrease in execution times can be observed in both subcharts for from the non optimized version to the *l2-norm* one. This time however, the use of the more relaxed convergence criterion yielded a greater speedup (3.16x on average) compared to the Erdős–Rényi graph (2.03x on average). This is mostly due to the fact that the pages tend to be grouped in small clusters, thus a convergence criterion that checks individual nodes for convergence lacks the ability to comprehend that pages in the same cluster have similar *Pagerank* values therefore leading to a much slower convergence.
Overall, the best performer of each category achieved an average 5.45x speedup over the baseline implementation

3.4.2 Correctness

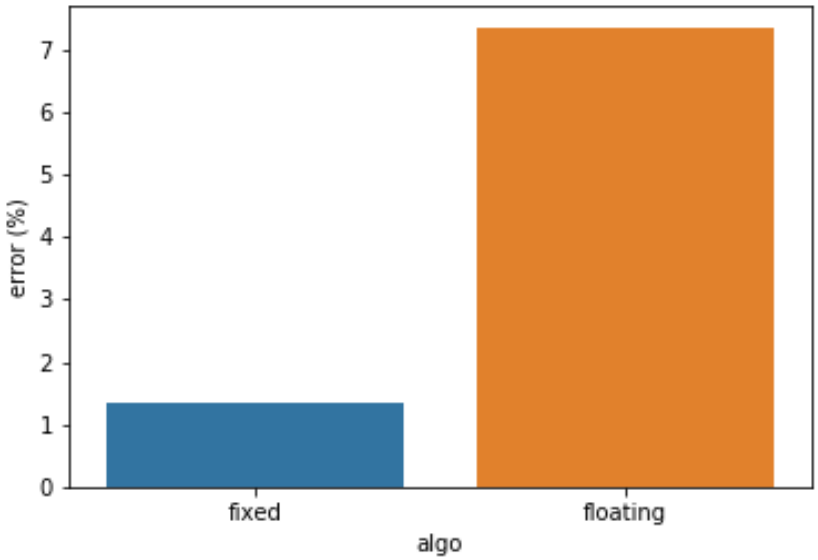

Figure 7: Percentage of wrong *Pagerank* ranks of the best performer of every implementation (*l2-norm*)

The fixed point version achieves much lower error than the floating point one. This is due to the fact that, even though the graph is still *globally sparse*, it can be considered *locally dense* due to the cluster like behaviour of the *Small World* model. Here, the extra *32 bit* of the fixed point representation allow for a much higher resolution thus leading to a lower error overall.

3.4.3 Iterations to convergence and convergence error

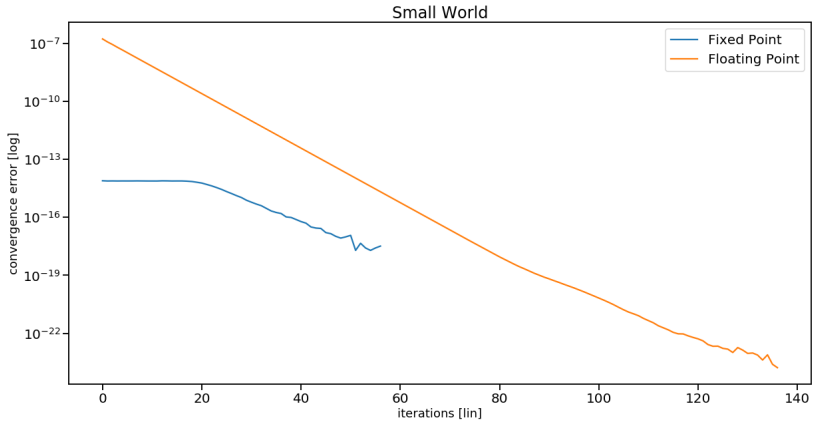

Figure 8: Rate of convergence of the best performer of every implementation (*l2-norm*)

The fixed point representations converges in less than half iterations compared to the floating point one. Once again, since the *Pagerank* values in the cluster are mostly similar, floating point looses time in unnecessary iterations trying to bring to convergence nodes with small differences in values.

3.4.4 Displacement of Pagerank values

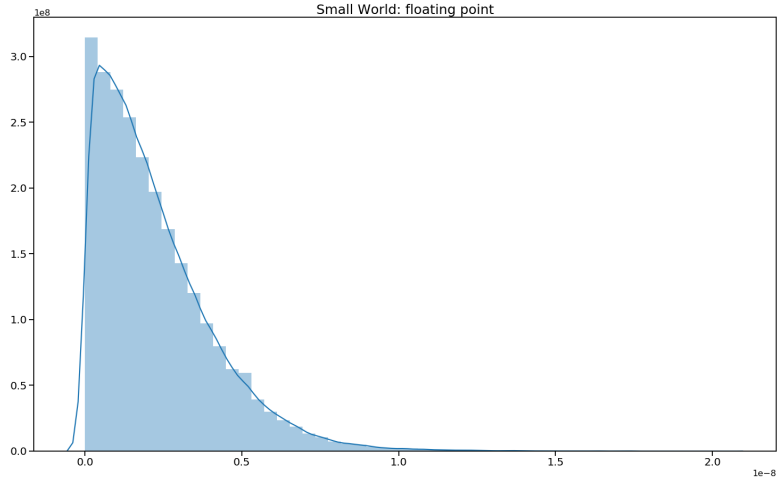

Figure 9: Distribution of displacement error of the best floating point performer (*l2-norm*)

- **Mean**: $2.25 * 10^{-9}$
- **Variance**: $3.47 * 10^{-18}$

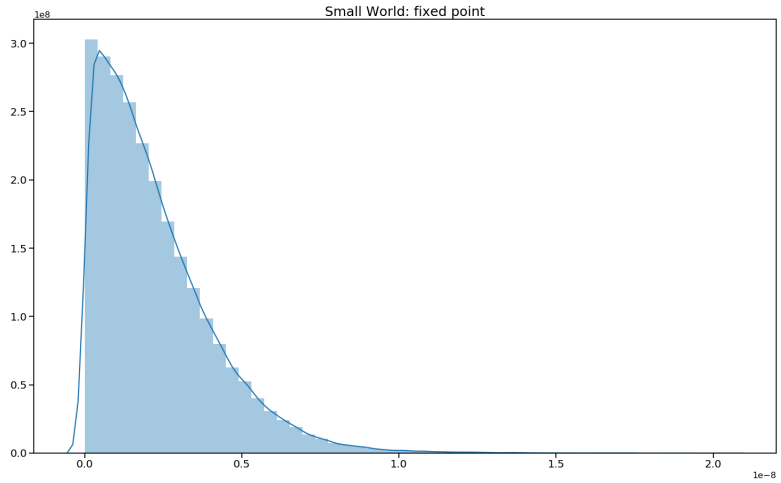

Figure 10: Distribution of displacement error of the best fixed point performer (*l2-norm*)

- **Mean**: $2.22 * 10^{-9}$
- **Variance**: $3.48 * 10^{-18}$

Small World graphs both approaches display a similar shape in the distribution of the displacement in the *Pagerank* values, showing that there is no particular benefit in using a floating point representation in this situation, since both variation of the algorithm achieve almost identical levels of consistency.

3.5 Scale Free

[1]The Scale Free network is a graph type whose degree distribution is unaffected by the scale of the graph, that is, for a given outdegre k the cardinality of the set of nodes having k outgoing arcs is proportional to $k^{-\gamma}$, for some γ in [2, 3]

3.5.1 Execution Times

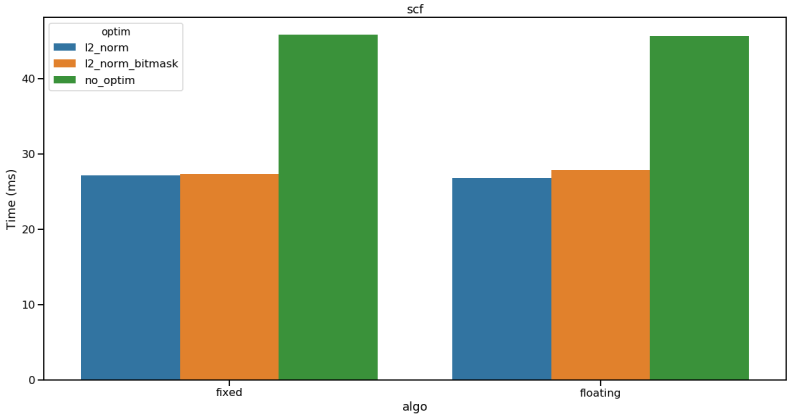

Figure 11: Execution time of every implementation

The *l2-norm* variant of the algorithm is still the best performer of the two categories but this time with a lower improvement over the non optimized version. This is due to the fact that the characteristics of this graph model aren't as pronounced as in the *Small World* graph. Nonetheless, an average 1.7x improvement is achieved with this approach.
Overall the best performer of each implementation achieved an average 4.3x speedup over the baseline implementation.

3.5.2 Correctness

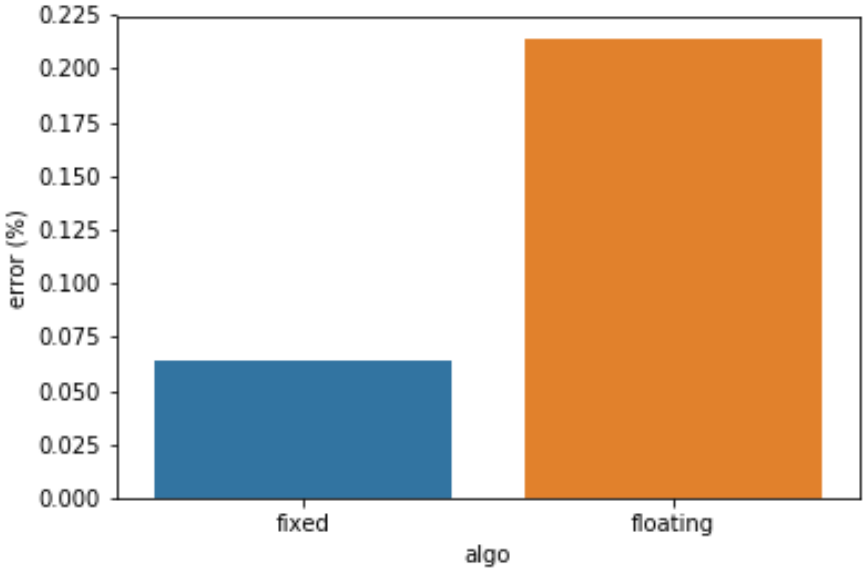

Figure 12: Percentage of wrong *Pagerank* ranks of the best performer of every implementation (*l2-norm*)

The higher resolution of the fixed point representation allowed for a lower ranking error overall, even though it is still negligible for both representations.

3.5.3 Iterations to convergence and convergence error

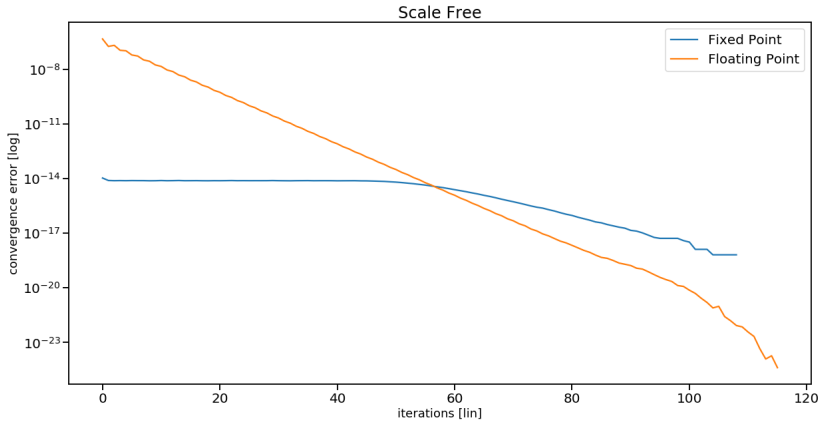

Figure 13: Rate of convergence of the best performer of every implementation (*l2-norm*)

The *inverse exponential* nature of the graph gives the rise to few dense areas (nodes that have an high number of inward arcs) and multiple sparse areas. This shifts the focus of the computation to the sparser areas and, being many of them slows down the computation for both variants of the algorithm.

3.5.4 Displacement of Pagerank values

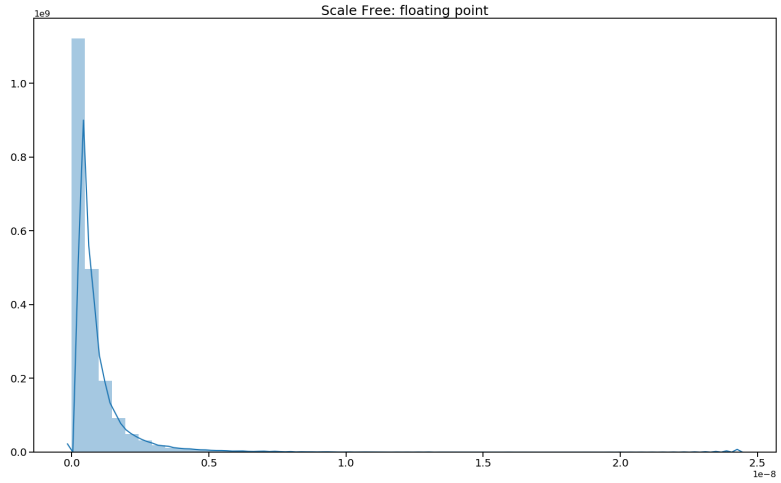

Figure 14: Distribution of displacement error of the best floating point performer (*l2-norm*)

- **Mean**: $7.25 * 10^{10}$
- **Variance**: $1.06 * 10^{-18}$

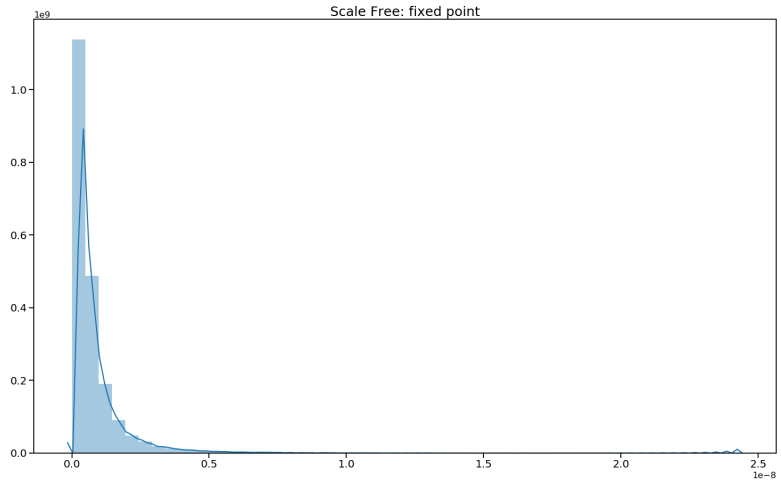

Figure 15: Distribution of displacement error of the best fixed point performer (*l2-norm*)

- **Mean**: $7.25 * 10^{-10}$
- **Variance**: $1.06 * 10^{-18}$

Since both version had a negligible error rate, a similar displacement error distribution is observed, indicating that they achieve similar levels of consistency.

4 Conclusion and Future Works

Even though *GPUs* are extremely optimized for *floating point, single precision* mathematical operations we showed that, by building a fixed point implementation of the algorithm, similar execution times are achievable while keeping error levels appropriately low and, in some cases, obtaining lower error than the floating point representation.
Using that in conjunction with an *approximated* convergence criterion was shown to achieve the best execution times in every test case while not impacting on precision.

Execution Times

		Erdős–Rényi	Small World	Scale Free
Floating	No optimizations	46.44ms	48.70ms	45.62ms
	l2-norm	22.54ms	15.48ms	26.78ms
	l2-norm + bitmask	23.31ms	15.68ms	27.88ms
Fixed	No optimizations	46.25ms	48.94ms	45.77ms
	l2-norm	22.90ms	15.41ms	27.12ms
	l2-norm + bitmask	23.06ms	15.71ms	27.30ms

Speedup compared to the baseline

		Erdős–Rényi	Small World	Scale Free
Floating	No optimizations	2.34x	1.70x	2.54x
	l2-norm	4.83x	5.49x	4.33x
	l2-norm + bitmask	4.67x	5.42x	4.16x
Fixed	No optimizations	2.35x	1.73x	2.53x
	l2-norm	4.75x	5.41x	4.27x
	l2-norm + bitmask	4.72x	5.41x	4.24x

Future research can be developed on two major aspects:
On the *Pagerank* side, other approximation tecniques can be tested, such as varying the scaling factor or purposely removing least significant bits in order to enforce faster convergence.
On *general purpose* side, other algorithms that use a *distance-like* convergence criterion to terminate can be implemented using the aforementioned techniques, in order to assess the limitations of a fixed point representation and the choice of different, faster converging, distances.

References

[1] Béla Bollobás et al. "Directed Scale-free Graphs". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '03. Baltimore, Maryland: Society for Industrial and Applied Mathematics, 2003, pp. 132–139. ISBN: 0-89871-538-5. URL: `http://dl.acm.org/citation.cfm?id=644108.644133`.

[2] P. Erdös and A. Rényi. "On Random Graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.

[3] NVIDIA. "NVIDIA Launches the World's First Graphics Processing Unit: GeForce 256". In: (1999). URL: `https://web.archive.org/web/20160412035751/http://www.nvidia.com/object/IO_20020111_5424.html`.

[4] L. Page et al. “The PageRank citation ranking: Bringing order to the Web”. In: *Proceedings of the 7th International World Wide Web Conference*. Brisbane, Australia, 1998, pp. 161–172. URL: citeseer.nj.nec.com/page98pagerank.html.

[5] Duncan J. Watts and Steven H. Strogatz. “Collective dynamics of ’small-world’ networks”. In: *Nature* 393.6684 (June 1998), pp. 440–442. ISSN: 0028-0836. DOI: 10.1038/30918. URL: http://dx.doi.org/10.1038/30918.