# Concurrent PageRank Report

**Author:** Jan Wawszczak 418479

## Approach to parallelization

I have started with optimizing **SingleThreadedPageRankComputer** by decreasing the number of **unordered_map** accesses by keeping as much data as possible in a single map element. I have also reduced the number of object copies by replacing them with references, which alone resulted in a considerable speedup.

The main speed advantage of a multithreaded program comes from optimizing the ‘iterations loop’ in which page ranks are updated. In each iteration, the page rank of a page in the network can be updated separately from other pages, so the required work can be efficiently split between many threads.

An obvious and simple solution is to evenly split the collection of pages between all threads, but it results in threads that finish sooner waiting for other to complete.

The more optimal solution is the one where all threads work as long as there is work remaining. It can be achieved by using a pointer shared by threads, which points to the next page to be updated. Each thread accesses the pointer with an exclusive lock and advances it to the next element. However, my testing has shown that this solution is not efficient for sparse graphs (like the one in the test with 50k nodes) because threads spend a lot of time waiting on the mutex compared to the time required to process that graph.

So my final solution is the combination of both mentioned - each thread starts with its own chunk of work. Only after completing its own job, the thread checks whether other threads have completed their work and if not, it helps them. My testing has confirmed that this reduces the amount of time threads spend waiting on mutexes.

I have tried using this approach in other parts of the program, like creating edges or generating page ids, but the benefit was very little compared to the speedup of the ‘iterations loop’. This is because creating edges is pretty quick and generating a single page id pretty slow, both compared to the ‘iterations loop'. So in these cases, I opted for the simpler code and used a single shared pointer to the next element.

## Machines specification

The performance of my program has been tested on the following machines (CPU data obtained using `lscpu`)

| Property | Huawei Matebook 13 | students.mimuw.edu.pl |
|---|---|---|
| **OS** | Ubuntu 20.04.2 LTS | Debian GNU/Linux 10 |
| **CPU model name** | Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz | Intel Xeon Processor (Skylake, IBRS) |
| **CPUs** | 8 | 64 |
| **Threads per core** | 2 | 1 |
| **Cores** | 4 | 64 |
| **RAM** | 7.5GB | 328GB |

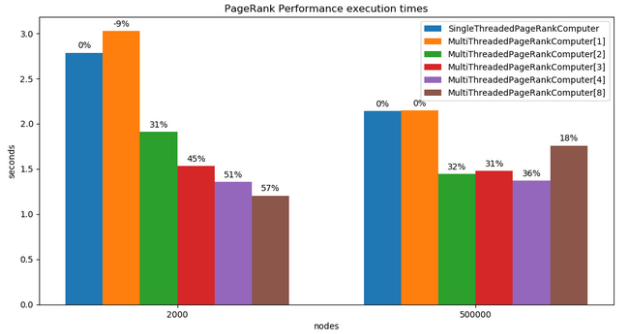
# Page 2

## Execution times

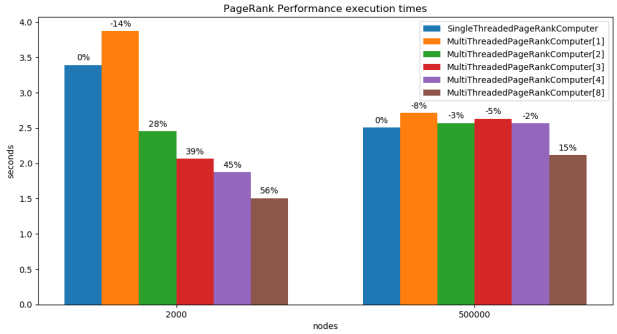
The following graphs show the execution times of tests on both machines. The percentages above bars show differences in execution times compared to **SingleThreadedPageRankComputer**.

**PageRankPerformance** is presented for **2000** and **50000** nodes.

**Huawei Matebook 13**



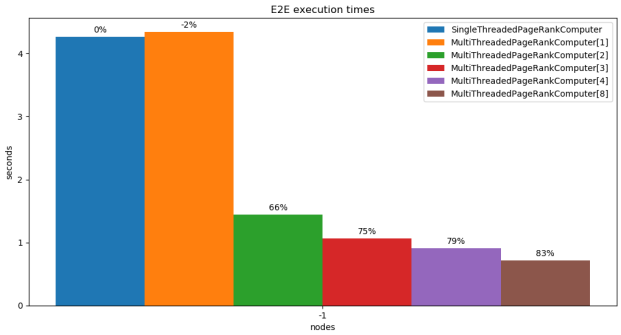

**students.mimuw.edu.pl**



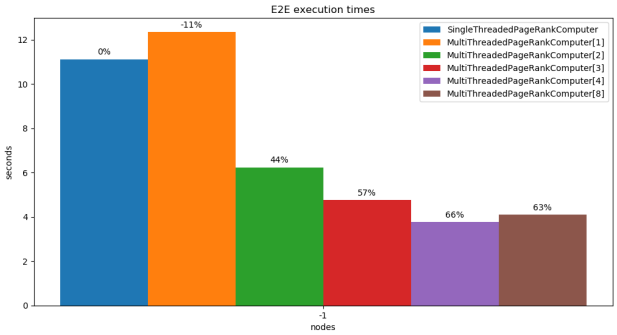

As I noted before, in the case of the **sparse graph** with **50K** nodes, waiting on mutexes and other losses associated with multithreading start to outweigh the gains.

---

**Huawei Matebook 13**




**students.mimuw.edu.pl**




**E2e** consists of only one test, so the number of nodes is not needed to distinguish it, hence the value of -1 on the graph.

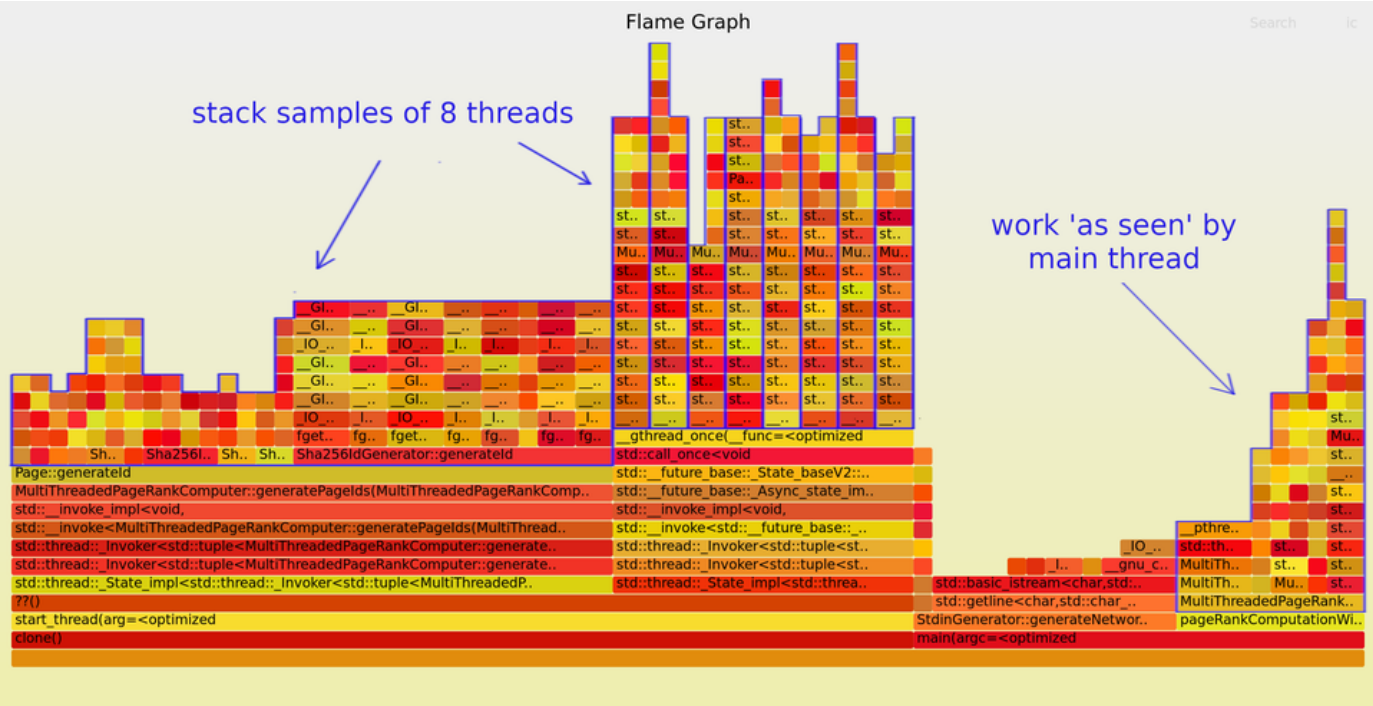
I believe that the dramatic improvement in speed of the **E2e** test on my laptop is caused by **Hyper-Threading Technology**, which allows for more efficient utilization of a single physical core by executing some instructions simultaneously.

**students.mimuw.edu.pl** machine does not poses this capability and execution times are within the reasonable range.

# Page 3

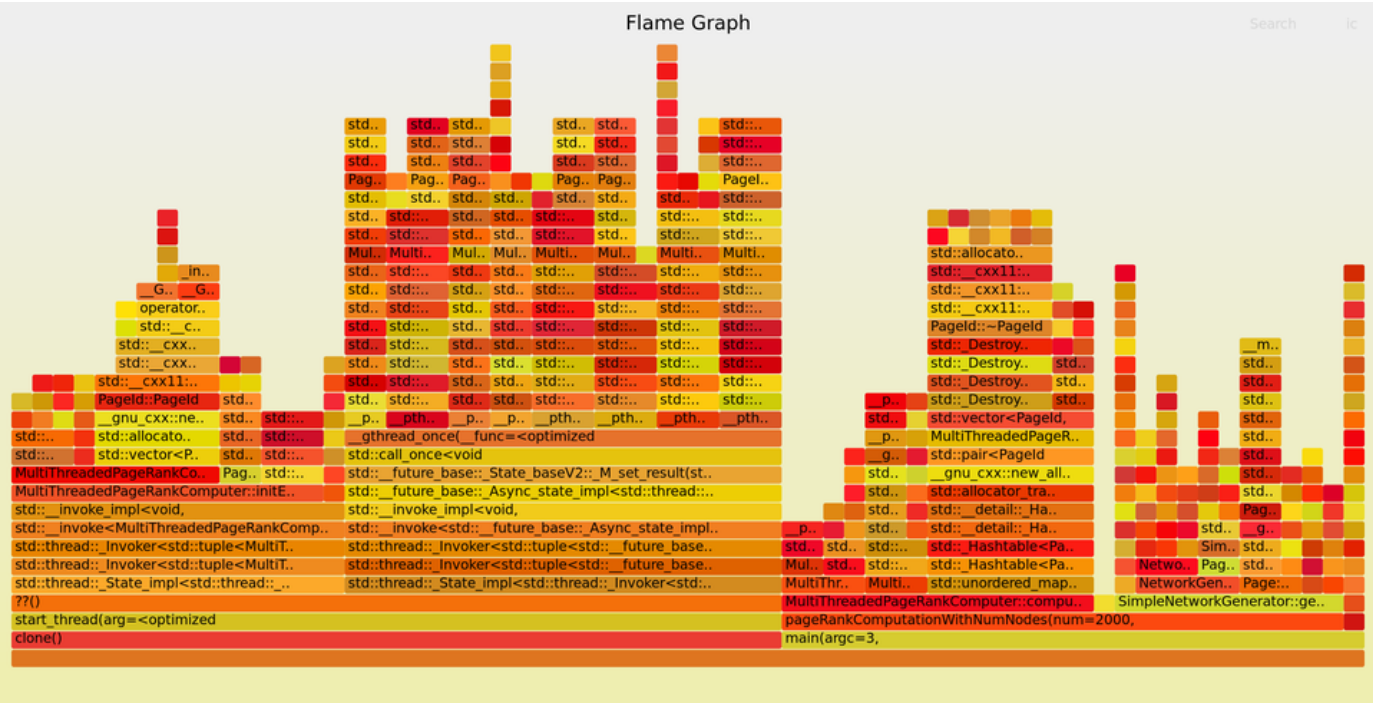
## Flame Graphs 🔥

**E2e [8 threads]** Huawei Matebook 13

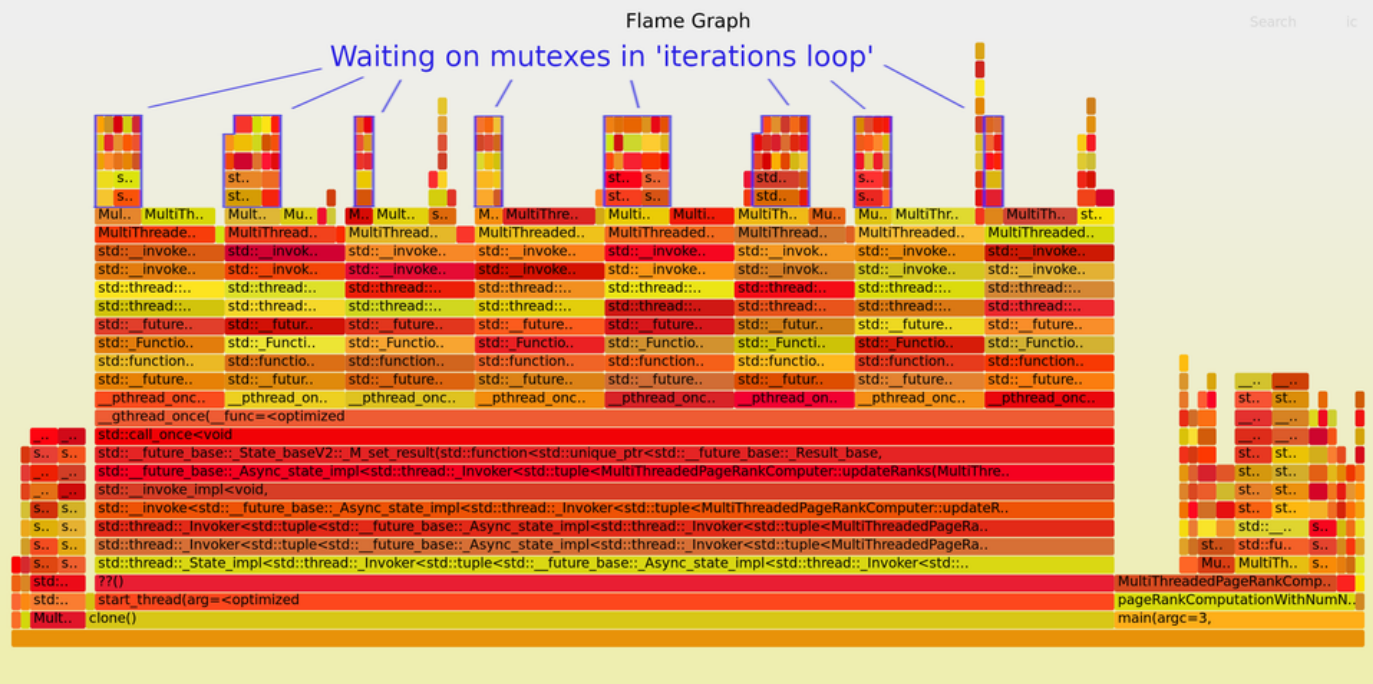



Stack samples structure on other graphs is analogous so it is not highlighted

**PageRankPerformance [2K nodes][8 threads]** Huawei Matebook 13

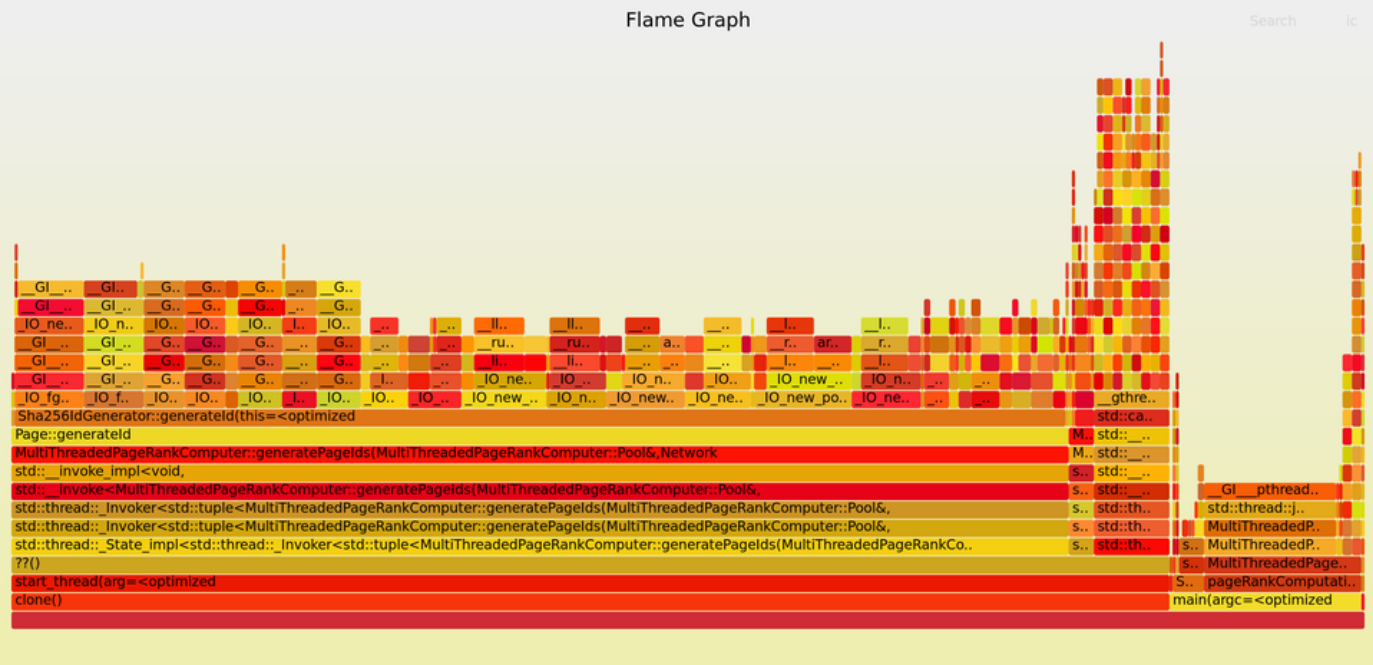


**PageRankPerformance [50K nodes][8 threads]** Huawei Matebook 13

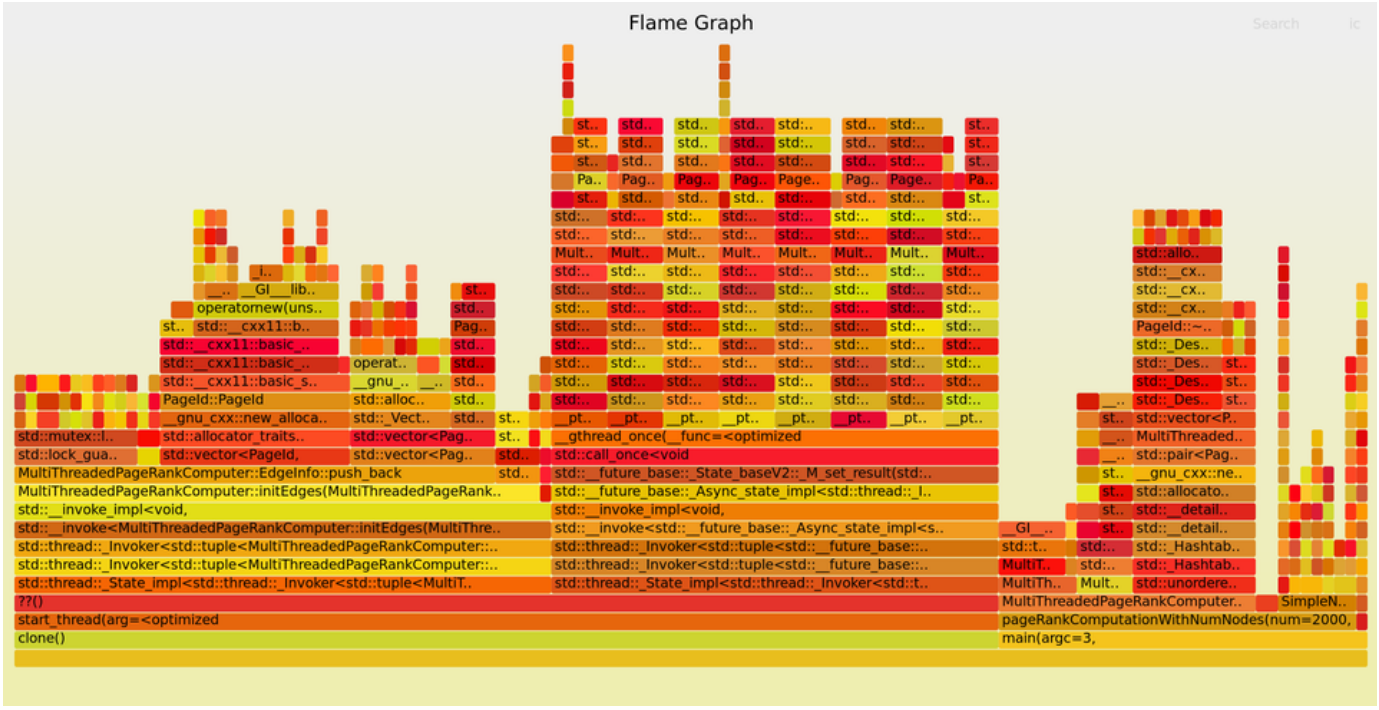



Same tests on **students.mimuw.edu.pl** for reference

**E2e [8 threads]** students.mimuw.edu.pl



**PageRankPerformance [2K nodes][8 threads]** students.mimuw.edu.pl




**PageRankPerformance [50K nodes][8 threads]** students.mimuw.edu.pl