# A6 Concurrency Assignment

- Division Labor:
  - Chi Chun Chen
  - Shaojie Wang
  - We discuss and implement the parallel algorithm together, and each do the experiments of different arguments in SSSP program

## File Structure

- All the python script is for benchmarking or testing SSSP.java

```
.
├── benchmark.py
├── Coordinator.java
├── ec2.py
├── ec4.py
├── Makefile
├── SSSP.java
└── test.py
```

## How to install

- install
  - `make`
- remove java .class files
  - `make clean`
- run
  - `java SSSP -a <0~3> -n <n> -t <t> -d <d> -g <g> -D <delta>`

## What we've done

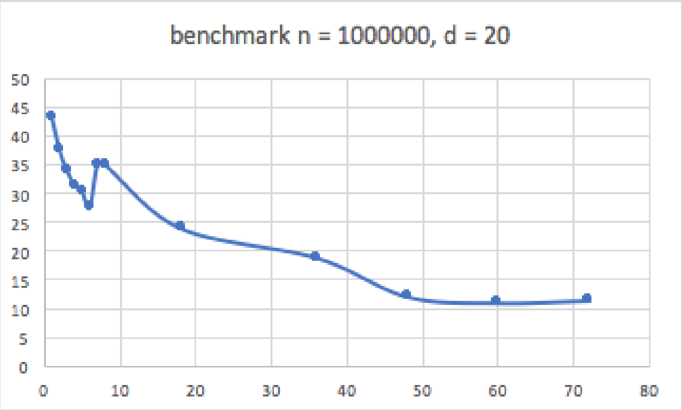
- We implement the parallelized delta stepping by creating thread number of message queue and thread number of buckets
- For correctness, we use a test script named `test.py` to check through different combinations of thread number, vertex number, seed number, and vertex degree number
  - By using this test, we found out that our original four barriers version can be tuned to three while still have correct answer
  - With fewer barriers, we sometimes get speed up for almost a second. (The speed up happens in the tests that have milions of vertexes and vertex degree of thirty to fifty)

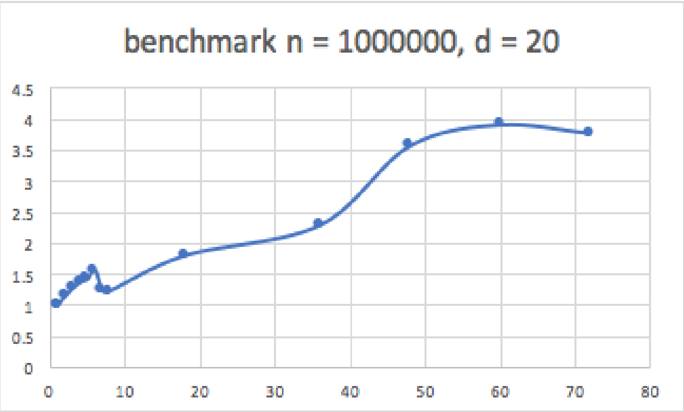
- [Extra Credit #2] We added `-D` for manually setting the delta value for our SSSP program
  - Same as other parameters, simply add `-D <delta>` in the argument list of SSSP program
  - For example: `java SSSP -a 0 -t 1 -n 1000000 -d 10 -D 10000000`
    - In this case the delta has been manually set to 10000000, and the buckets size has also been tuned to `maxCoord * 2 / delta`
    - The experiments for delta is in Experiment section
- [Extra Credit #4] We plotted and discussed this part below
  - Section: Experiment with number of vertices (Extra Credit #4)

## How we parallelize Delta Stepping Algorithm

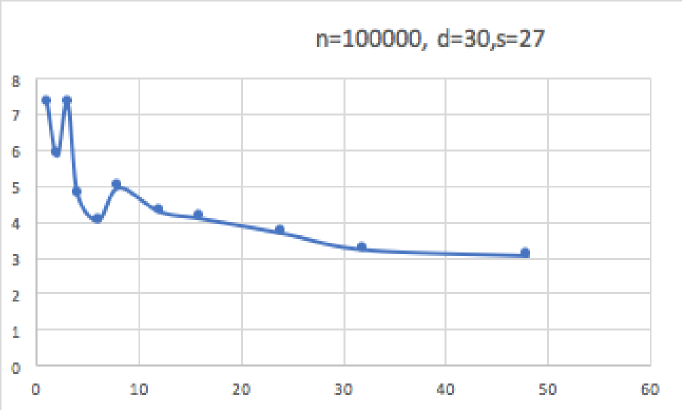
- We parallize the delta stepping by creating thread number of bucket list before start running threads
- The big `for(;;)` loop is moved into thread, so we merely start and join the thread in `DeltaSolve()`
- General idea of barrier positioning and message passing:
  - Instead of the 2n(n-1) message queues, we only created n queues
  - Each queue maintains a message receiver for a thread
  - We position four barriers in the `for(;;)` loop
  - The general procedure in the `for(;;)` loop is:
    - A **barrier** at the beginning of `do...while` loop
    - Poll all the messages in the message queue for current thread
    - `relax()` them
    - **Barrier**
    - Choose vertices and edges (stored in `Request` instances) to do light relax (not relax here)
    - Add all of them to the message queue for the goal thread (even for the thread itself), by the vertices' hash code.
    - **Barrier**
    - The end of while loop, checking condition when all message queues are empty.
      - There is no need to check whether all current buckets are empty, because the messages in buckets have already been added to queue and removed from bucket.
    - Outside after the `do...while` loop, we do heavy relax.
    - **Barrier**
    - Move to the next buckets, or break.

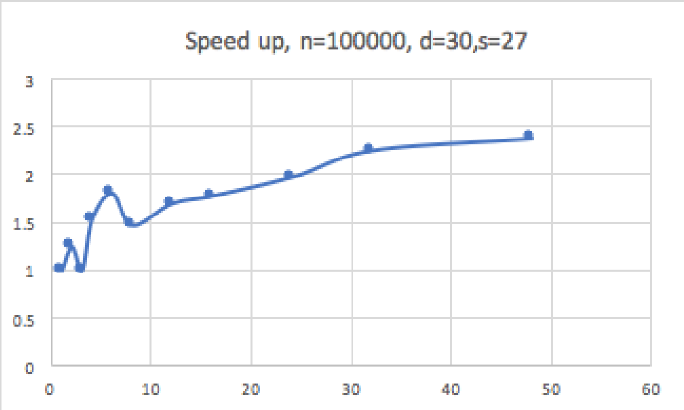
## Experiments

- We run the benchmark of our delta-stepping algorithm on `node2x18a.csug.rochester.edu`
- Graphs in this section:
  - y-axis is time (second)
  - x-axis is number of thread
- We found that in most of the experiments, there is a sudden rise in run time when thread number reaches 8, and then the time will drop as thread number increases.
- Overall, the parallelization is successful since the time cost decreases as thread number goes bigger
  - Especially, we found out that while the degree is greater than ten, the speed up ratio is better than degree less than ten
- Yet, we did not achieve the ideal speed up rate
  - It might be because of delta-stepping algorithm has lots of overhead when accessing message queues and barriers
- Also, we found out that our program is much slower than the newer version of Dijkstra
  - Main reason is the time complexity and overhead of Dijkstra is better than delta-stepping
- java SSSP -a 0 -n 1000000 -d 20 -t <n> -s 4
  - 

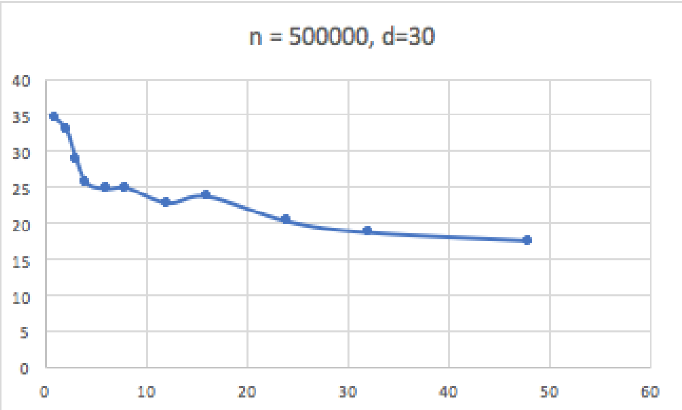
  - 

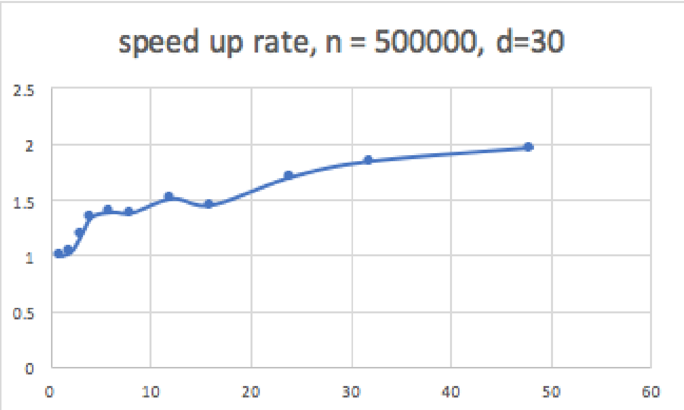
- java SSSP -a 0 -n 100000 -d 30 -t <n> -s 27

- 

- 

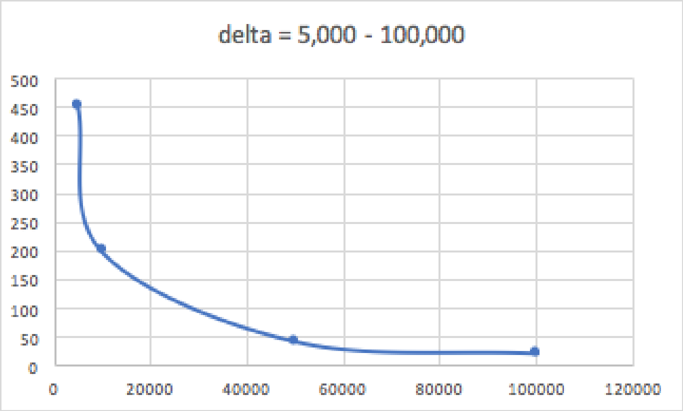

- java SSSP -a 0 -n 500000 -d 30 -t <n> -s 27

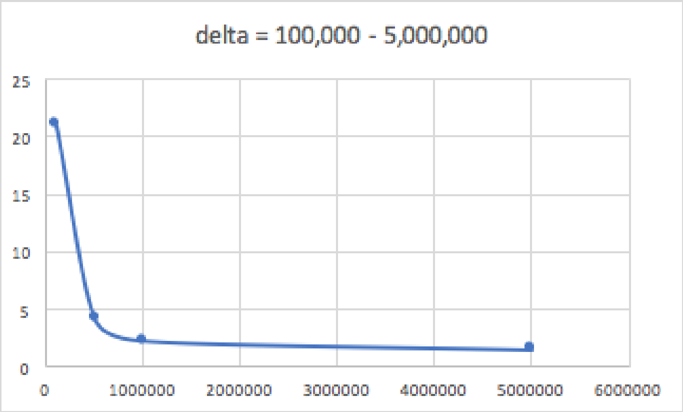
- 

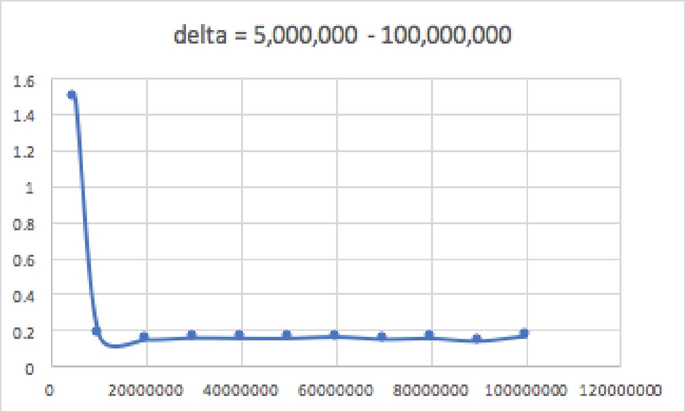
- 


## Experiment with Delta (Extra Credit #2)

- Number of buckets becomes ***one*** when delta is greater than `maxCoord * 2`
  - The speed of bucket with size one is just a little bit worse than the speed of delta being `maxCoord / degree`

- However, if the delta is much smaller than `maxCoord / degree`, the speed will become *intolerable slow*
- The following graphs:
  - y-axis is time (second)
  - x-axis is delta (different graph with different range of delta)
- This part of experiment do not need many threads, therefore, it's run on cycle machine (while no other people on the machine)
  - 

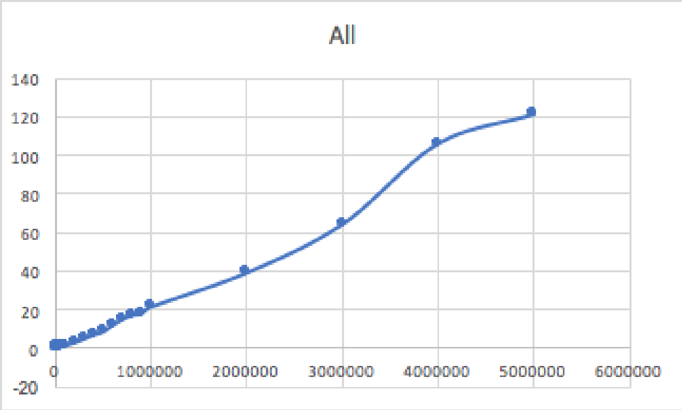
  - 

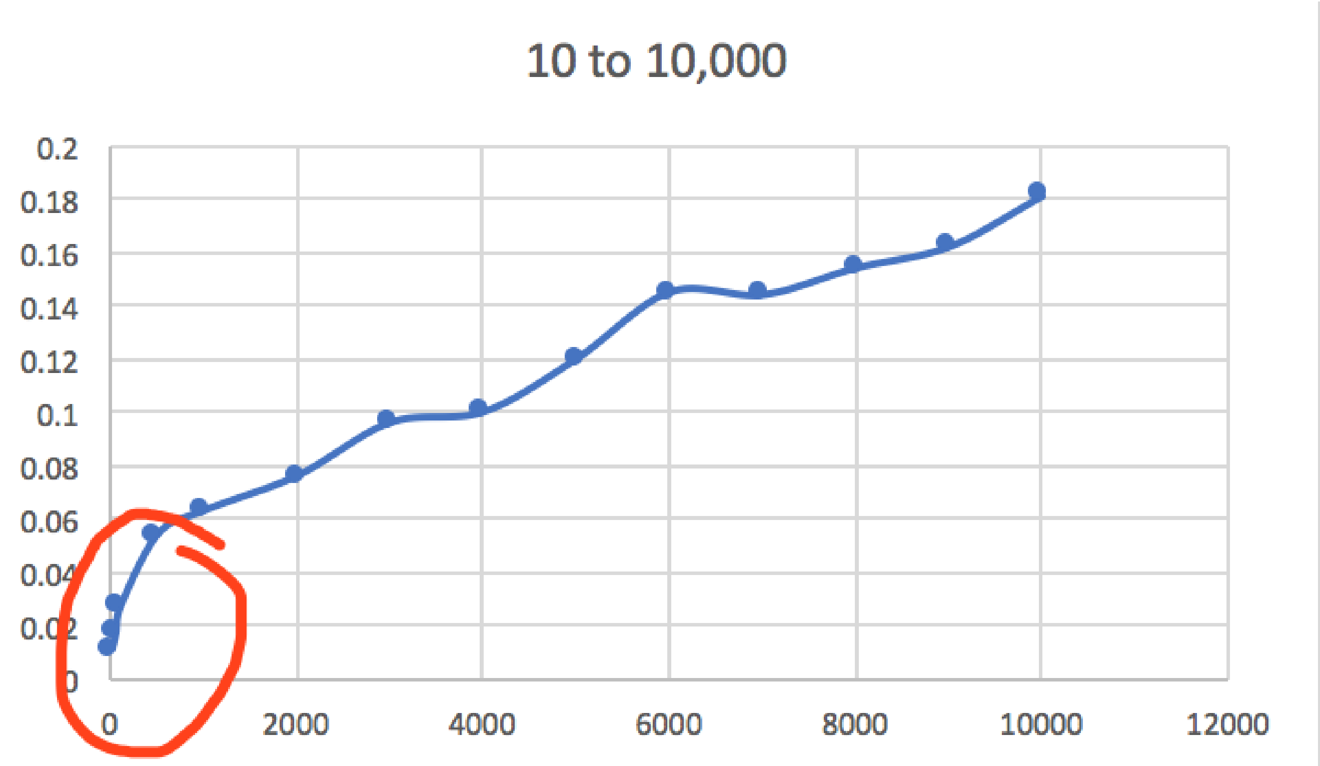
  - 


## Experiment with number of vertices (Extra Credit #4)

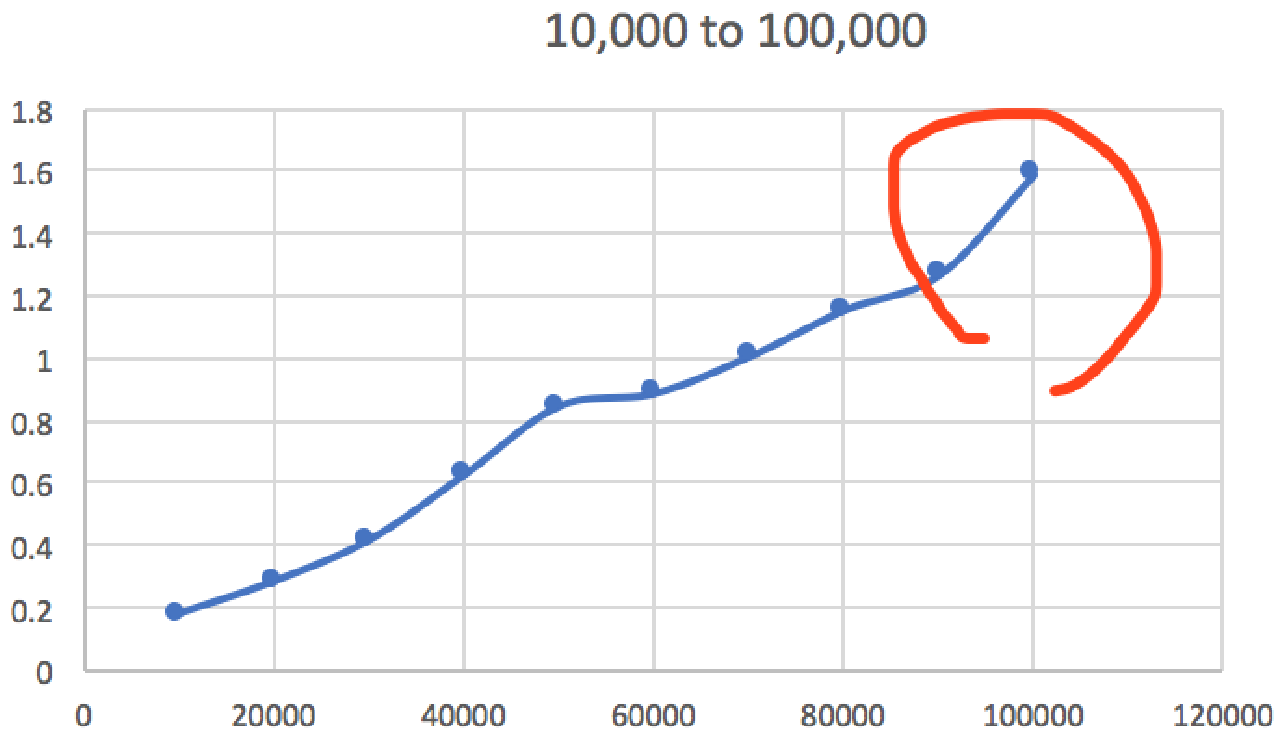
- This part experiment with different number of vertices with fixed thread, seed, degree, and other arguments
  - degree: 10, thread: 1, seed: 1
- Since this part of experiment do not need many threads, we run the experiment on cycle machine (while no other people on the machine)
- The following graphs:
  - y-axis is time (second)

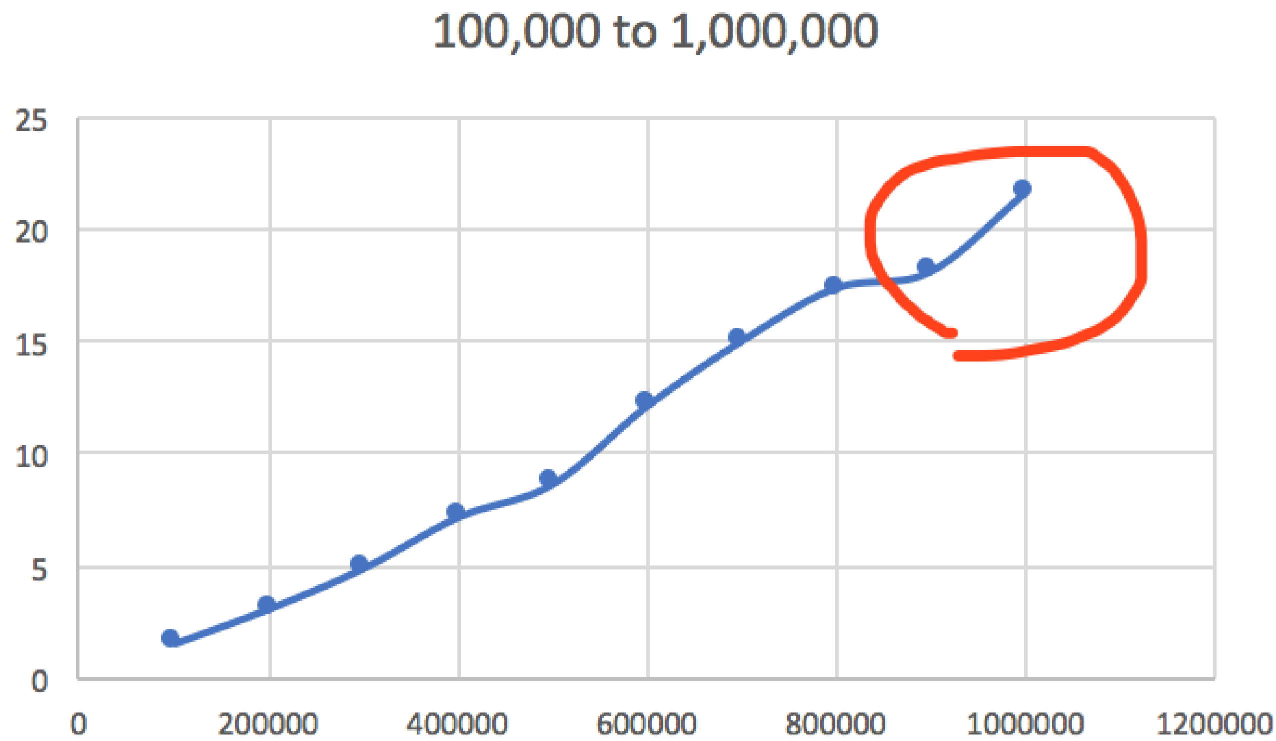
- x-axis is number of vertices
- Red ink circles out sudden rise of the curve



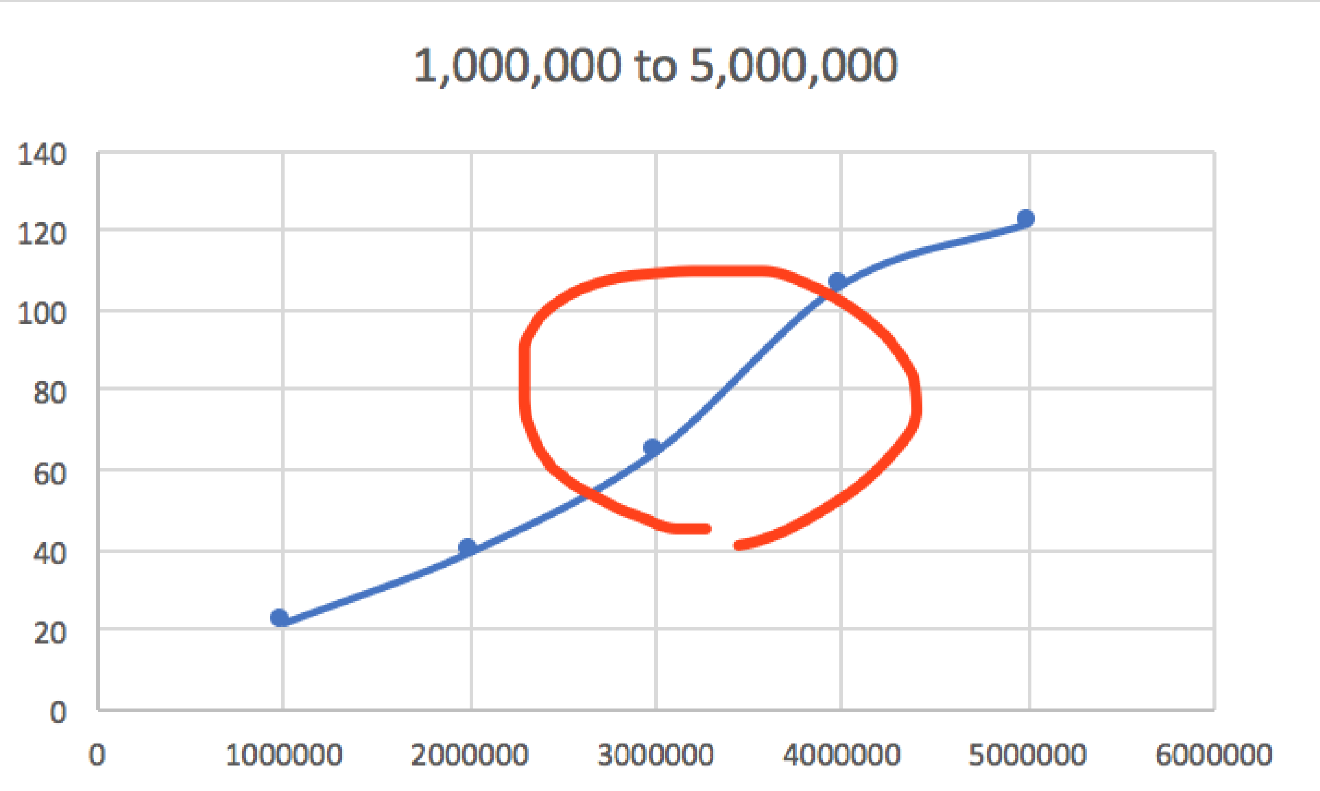

- 

  - This circled segment might indicates L1 cache miss ratio start to be way more than hit ratio
  - We guess that SSSP program start to use L2 cache while size of vertices more than `1000`

- 

    - This circled segment might indicates L2 cache miss ratio start to be way more than hit ratio
    - We guess that SSSP program start to use L3 cache while size of vertices more than `90000`
- 

    - This circled segment might indicates L3 cache miss ratio start to be way more than hit ratio
    - We guess that SSSP program start to use main memory while size of vertices more than `900,000`

- 


    - This circled segment might indicates main memory miss ratio start to be way more than hit ratio
    - We guess that SSSP program start to use disk while size of vertices more than `2,000,000`