Concurrent Minimum Spanning Tree Algorithm

- Paper: by R. Setia, A.Nedunchezhian, S. Balachandaran, in HiPC 2009

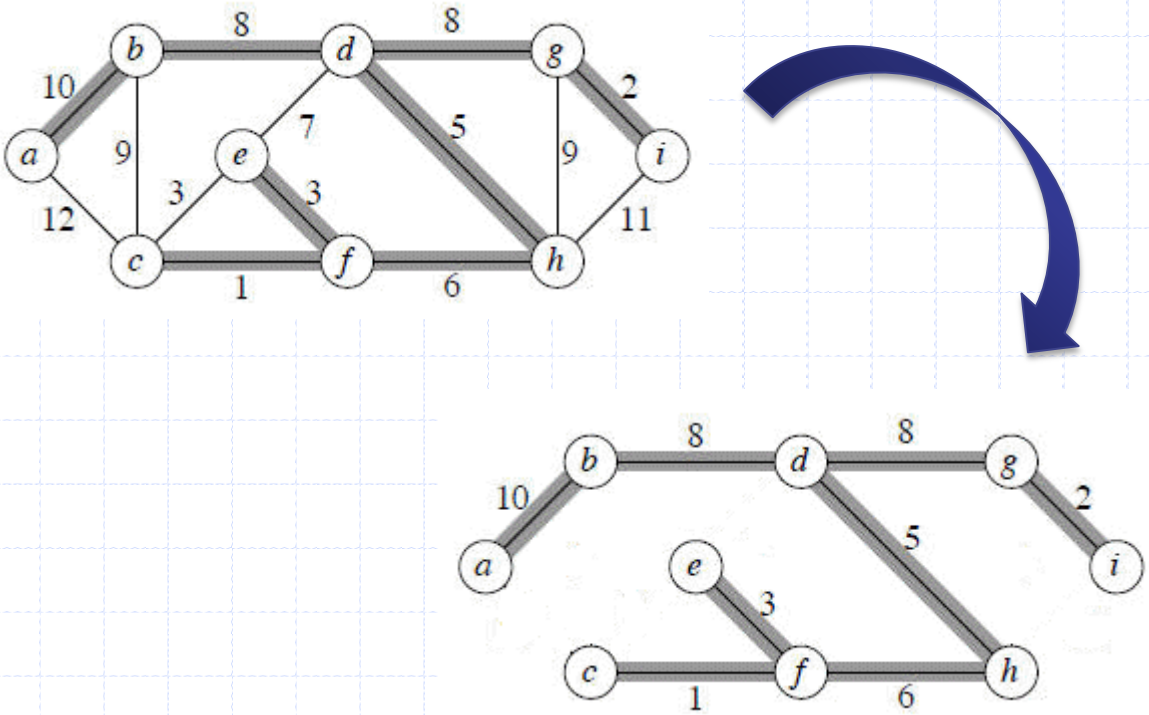
Xiwen Chen

DisCoVeri Group, York University, Toronto

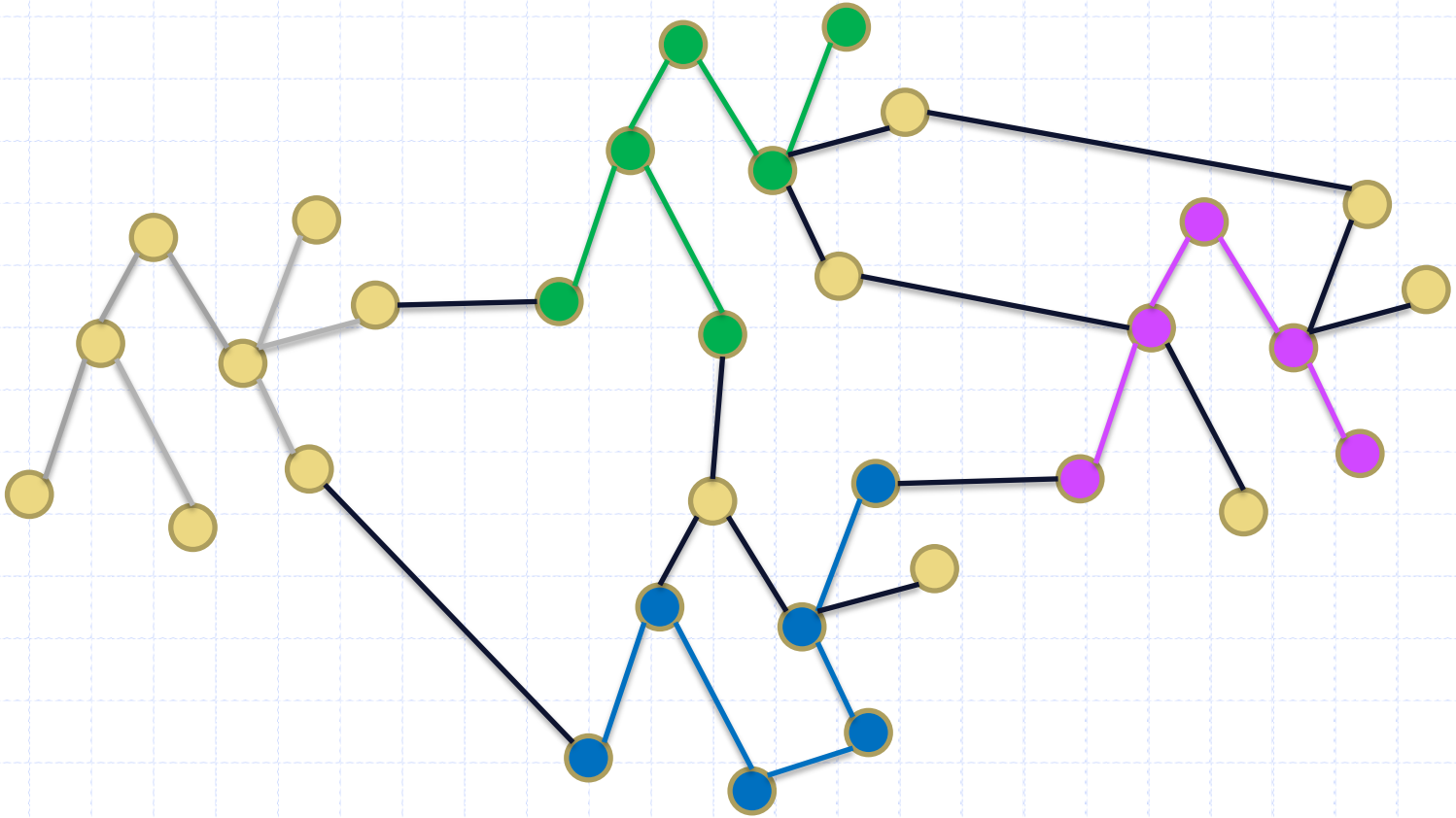
Outline

- A quick review.
- Two implementations.
- Balance schemes.
- Performances.

Minimum Spanning Tree



Parallel Prim's Algorithm



Two implementations

Naïve compareAndSet version:

```
Loop:

do{  get the node color
}while(compareAndSet()==false);

if(an uncolored node)
{  grows its MST; }
else if(colored by other thread)
{  mergeTree(); }
else if(already colored by itself)
{  continue;  }
else{
    somebody is working;
    //try again and hopes he finished!
}

end-loop
```

Semaphore version:

```
node.semaphore.acuqire();
if(an uncolored node)
{  grows its MST;  }
else if(colored by other thread)
{  mergeTree();  }
else if(colored by itself)
{  continue;  }

node.semaphore.release();
```

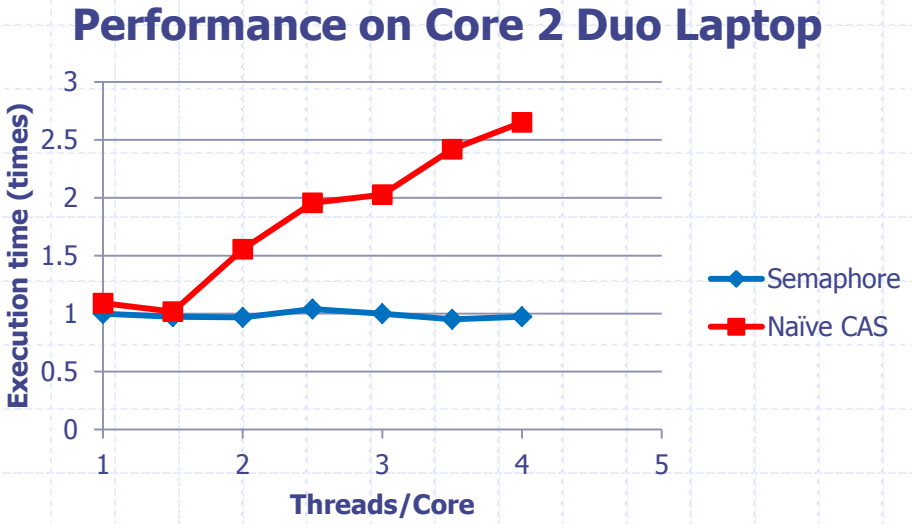
Key ideas:

- Each thread grows/colors their own single trees in parallel.
- When collision occurs between two threads i and j ($i<j$), thread j merges into i. j chooses another root to restart while i keeping growing.

Naïve CAS versus Semaphore

- Busy waiting v.s. sleep for a while.
- Semaphore in Java is implemented by CAS.

- The result :



Correctness Tests

- 80 threads by 20 cores with 1000-nodes 499500-edges graph.
  - 99902 out of 100000 tests passed.
- 80 threads by 20 cores with 2000-nodes 1999000-edges graph
  - 99896 out of 100000 tests passed.

  - while(!PriorityQueue.isEmpty()){
    - edge=PriorityQueue.findMin(); → by little chance, it returns null!
    - do something with edge…
  - }

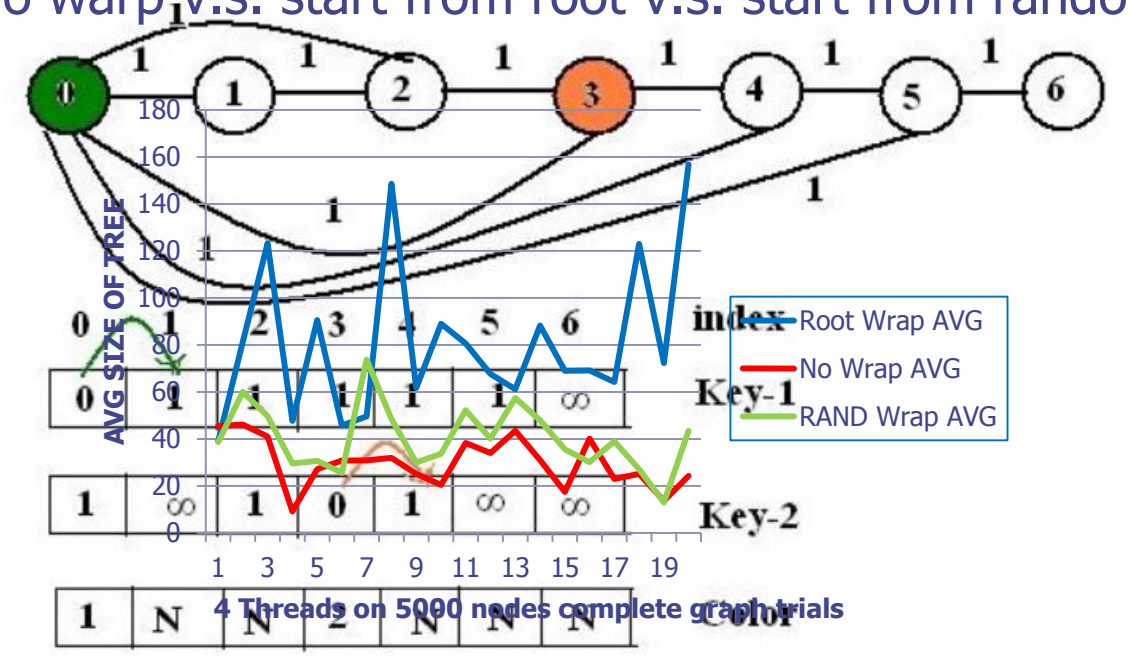
100000 out of 100000 passed!!!

Load Balancing schemes

- Base Problem Size
  - A threshold value for the number of uncolored nodes. If # of nodes fall below that threshold, we terminate the thread instead of let it pick a new random root to grow a new tree.
  - Instead of doing that, I kept tracking the times that a thread failed to randomly picked a root.

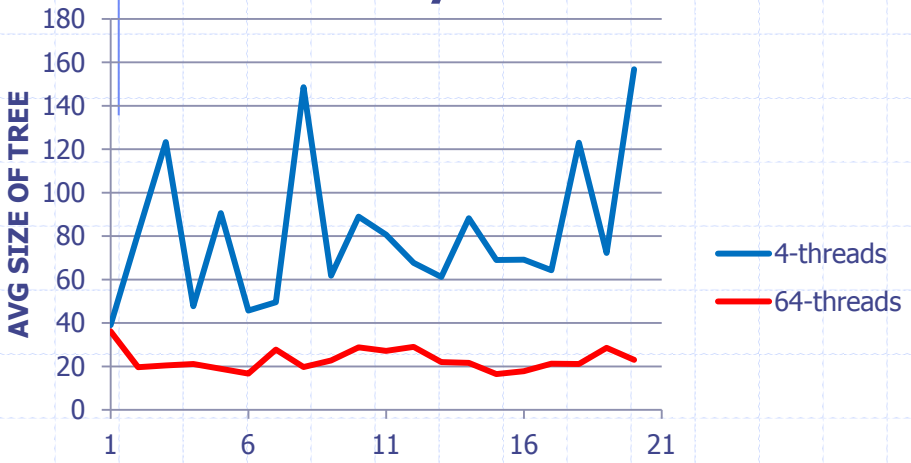
Heuristics

- Warp-around find-min:
  - No warp v.s. start from root v.s. start from random node

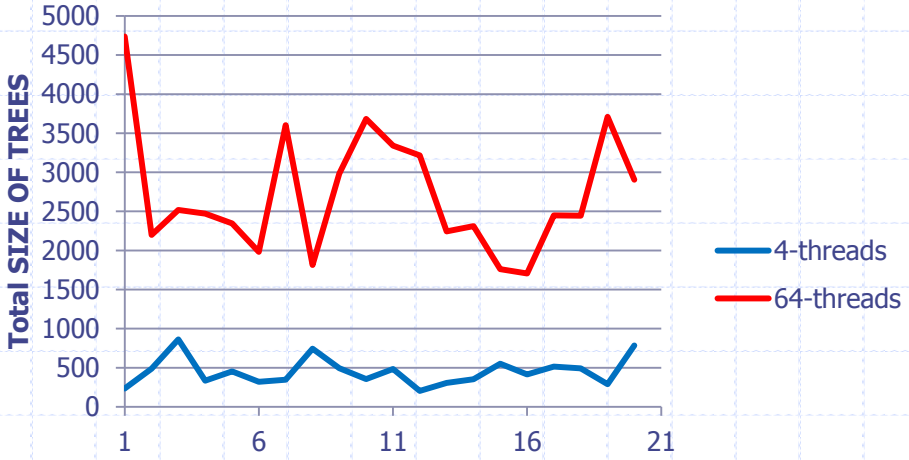


Threads Number

- From intuition, nobody wants to get too many collisions. So, we don't need many threads?



Average size of MST grown by children threads on 5000 nodes complete graph. Tested 20 times.



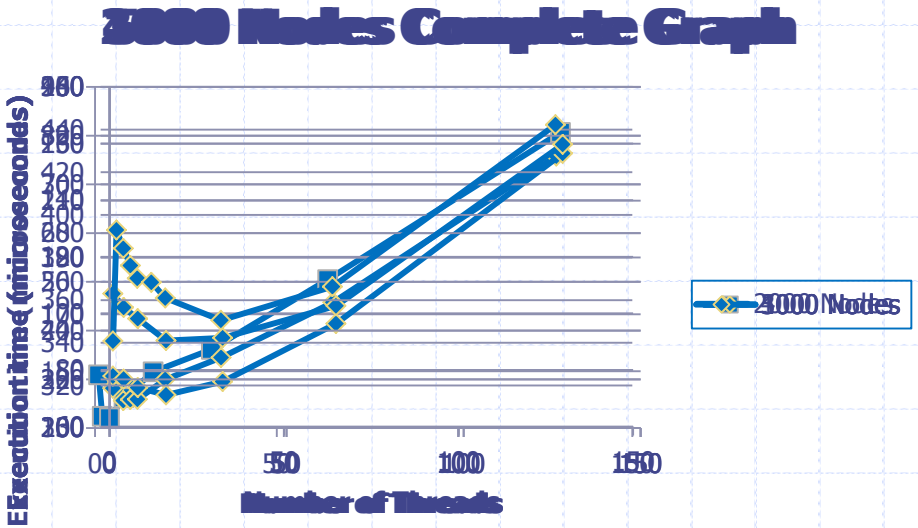
Total size of MST grown by children threads on 5000 nodes complete graph. Tested 20 times

Time Complexity

- Ideally, the total amount of time we spend is roughly $\Omega(N \cdot V \log V + \frac{E}{N})$, where N is number of threads we used.
- The concurrent algorithm will perform better, only when the edges access cost much.

Int[][] versus TreeMap

- To get access to int[][] is cheap. So the right part of $\Omega\left(N \cdot V \log V + \frac{E}{N}\right)$ is hard to become a bigger term.

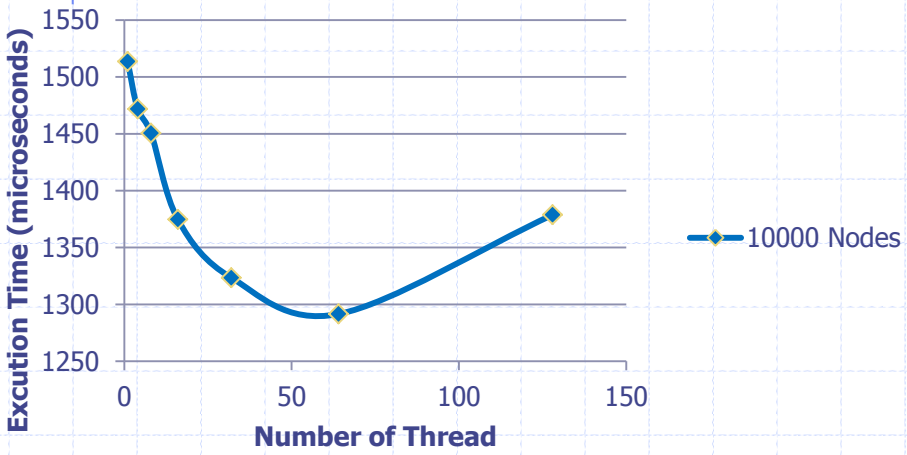


Int[][] versus TreeMap

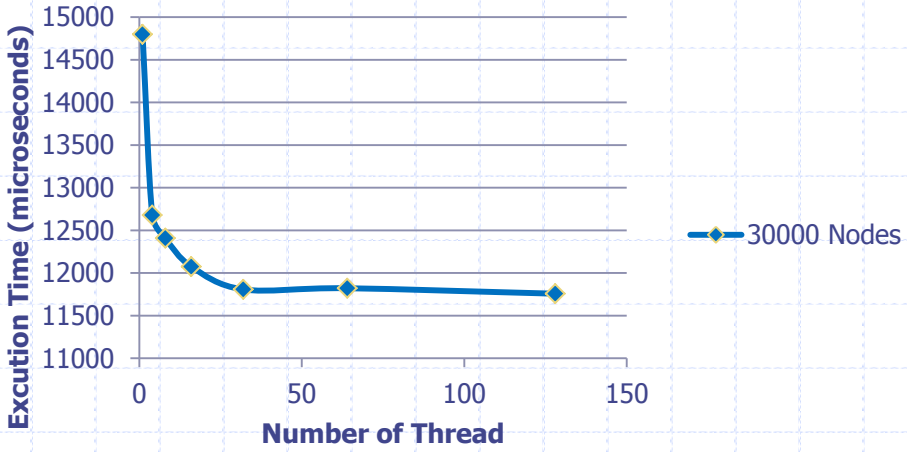
- It's so fast to access the int[][], where $\Omega\left(N \cdot V \log V + \frac{E}{N}\right)$ is hard to become a bigger term.

- Let's try more…

10000 Nodes Complete Graph

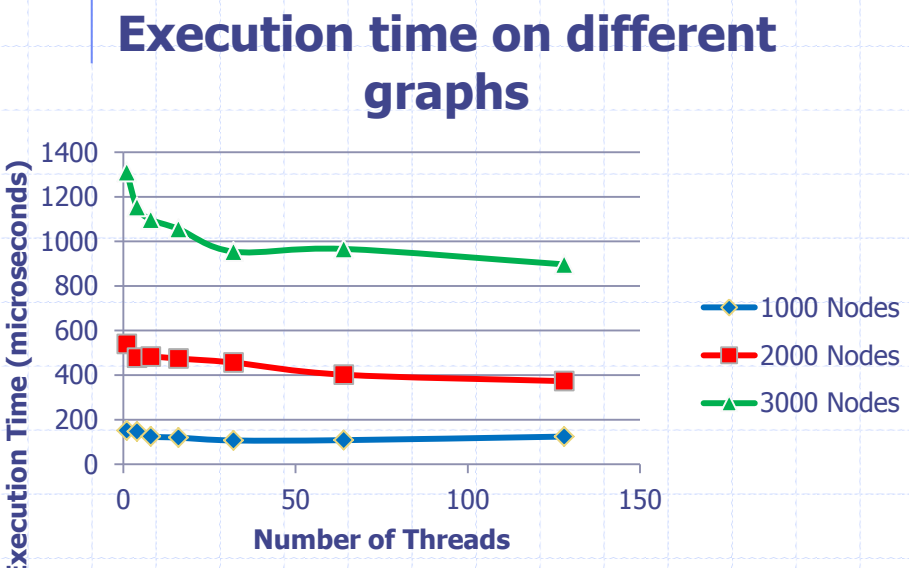


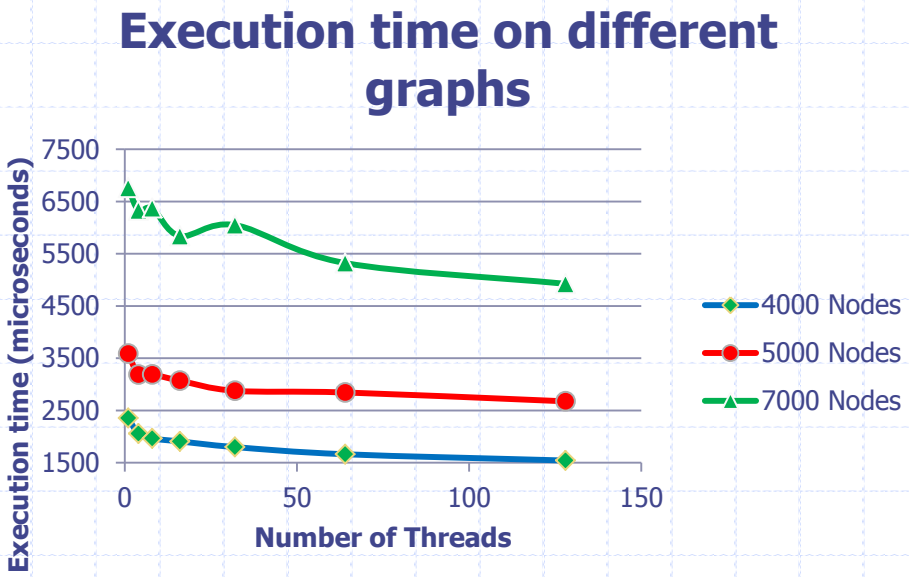
30000 Nodes Complete Graph



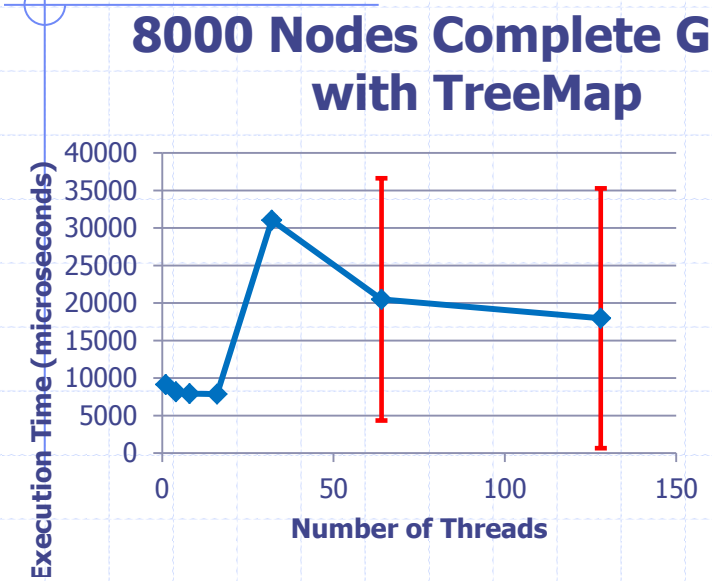
Int[][] versus TreeMap

- It cost $O(\log V)$ to access the elements in TreeMap. So the right term in $\Omega\left(N \cdot V \log V + \frac{E}{N}\right)$ becomes a larger part!





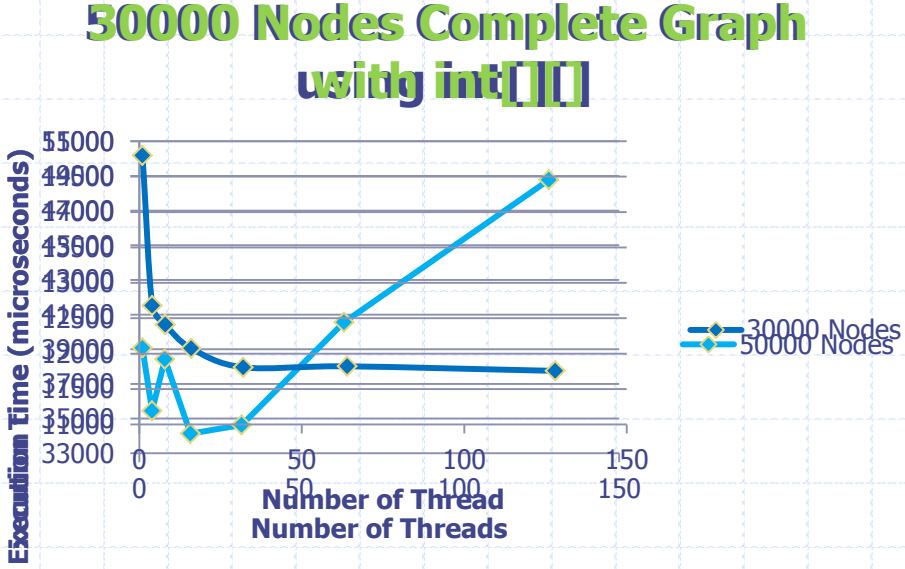
Unstable Performance



The MTL machine becomes unstable if one uses more than 12GB memory space.

8,000 nodes complete graph with TreeMap cost 8GB discrete memory space.

50,000 nodes complete graph with int[][] need 10GB continuous memory space.



Authors' results