KRUSKAL'S ALGORITHM FOR MST

Final Presentation

By - Bhavan Anand (50418487)
CSE708 - Programming Massively Parallel Systems
Instructor - Dr. Russ Miller

University at Buffalo The State University of New York

MINIMUM SPANNING TREE

Algorithms Available

- Borůvka's Algorithm
- Prim's Algoirthm
- Kruskal's Algorithm

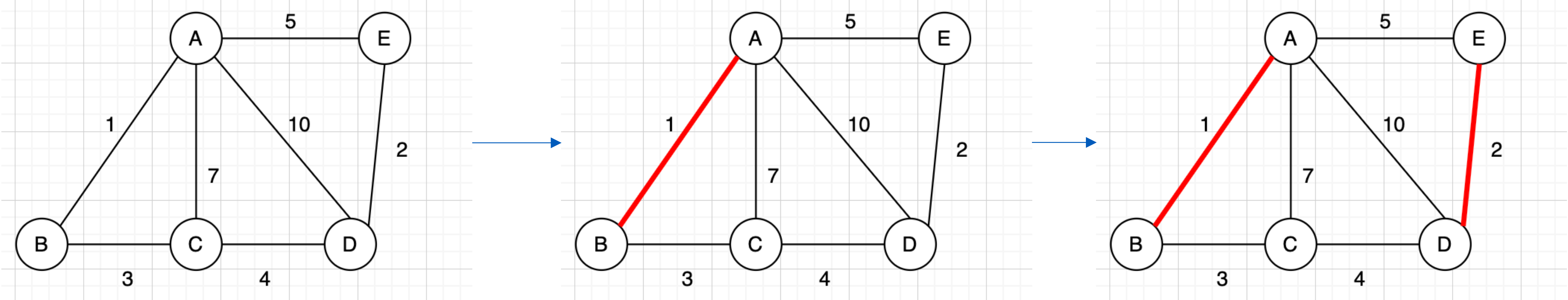
Kruskal’s Algorithm

Serial Algorithm Pseudo Code

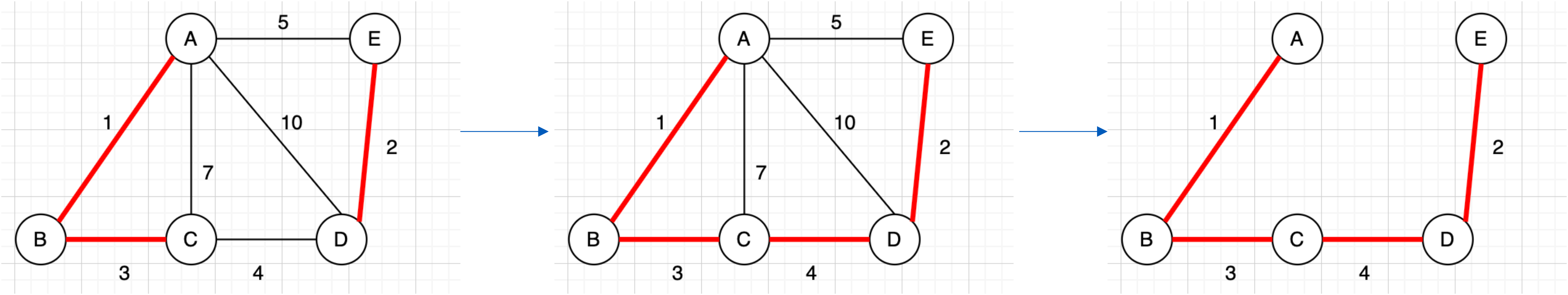
•Step 1: Sort all edges in increasing order of their edge weights.
•Step 2: Pick the smallest edge.
•Step 3: Check if the new edge creates a cycle or loop in a spanning tree.
•Step 4: If it doesn’t form the cycle, then include that edge in MST. Otherwise, discard it.
•Step 5: Repeat from step 2 until it includes |V| - 1 edges in MST.

```
for (i = 0; i < nEdges; i++)
{
    // Find parent sets of the nodes.
    dsNode *vParent = dsFind(&dsSet[edges[i].v]);
    dsNode *uParent = dsFind(&dsSet[edges[i].u]);
    // If they are from two different sets, merge them.
    if (vParent != uParent)
    {
        mst[nMstEdges++] = edges[i];
        mstLength += edges[i].w;
        dsUnion(vParent, uParent);
    }
}
```

Kruskal's Algorithm



Kruskal’s Algorithm



Minimum Spanning Tree-> AB: 1, BC: 3, CD: 4, DE: 2

Input sample

```
(0,1547) = 9
(0,3093) = 28
(0,4631) = 39
(0,5152) = 49
(0,2085) = 13
(0,2606) = 20
(0,3125) = 29
(0,7223) = 68
(0,3644) = 32
(0,4669) = 40
(0,8254) = 78
(0,576) = 2
(0,2118) = 14
(0,585) = 3
(0,9803) = 91
(0,4687) = 41
(0,5212) = 50
(0,2146) = 15
(0,3688) = 33
(0,3703) = 34
(0,120) = 0
(0,3195) = 30
(0,2688) = 21
(0,8322) = 79
(0,6283) = 61
(0,6801) = 64
(0,4242) = 37
(0,7315) = 69
(0,4246) = 38
(0,1182) = 5
(0,7839) = 75
(0,5286) = 51
(0,8361) = 80
(0,175) = 1
(0,7859) = 76
(0,8888) = 86
(0,4796) = 42
```

10000 Vertices Graph Input generated with different density of edges
And 30K Vertices

Output Sample

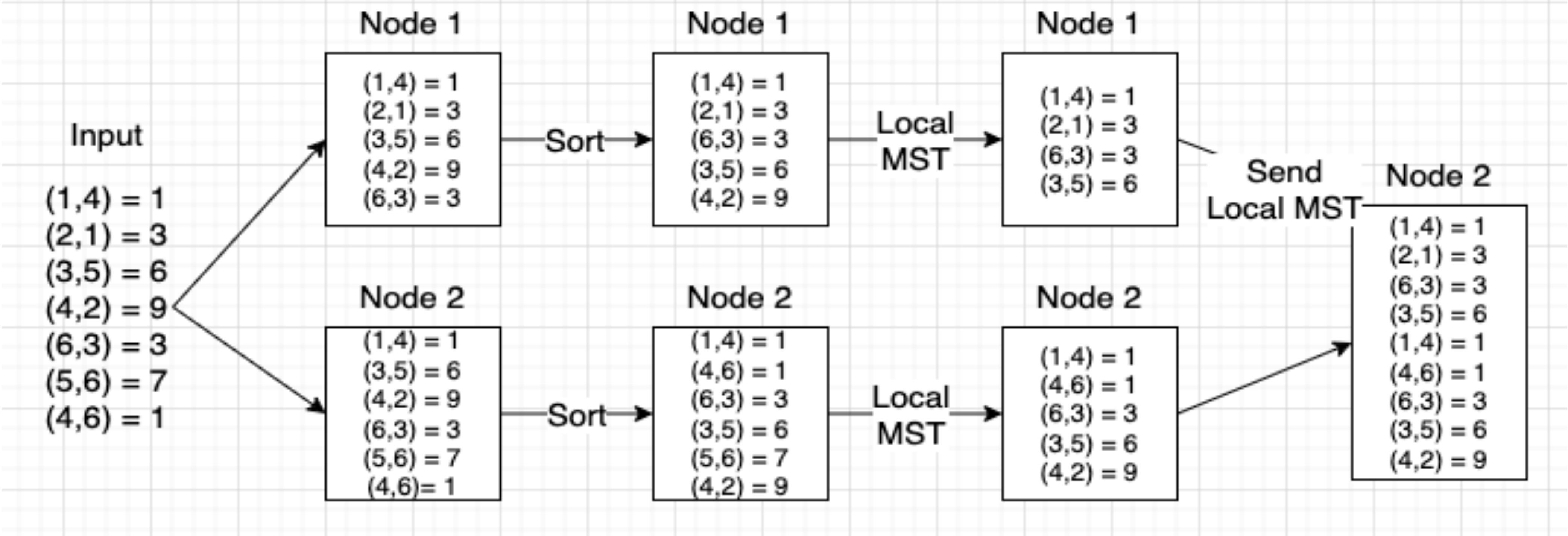
```
(0,7) = 0
(0,14) = 1
(0,29) = 2
(0,57) = 3
(0,73) = 4
(0,77) = 5
(0,81) = 6
(0,88) = 7
(0,110) = 8
(0,129) = 9
(0,135) = 10
(0,137) = 11
(0,159) = 12
(0,161) = 13
(0,174) = 14
(0,211) = 15
(0,217) = 16
(0,219) = 17
(0,222) = 18
(0,228) = 19
(0,237) = 20
(0,250) = 21
(0,255) = 22
(0,256) = 23
(0,259) = 24
(0,266) = 25
(0,274) = 26
(0,278) = 27
(0,331) = 28
```

V-1 edges make up a Minimum Spanning Tree. Hence 9999 records would be Expected for 10000 Vertices Input

Parallel Implementation

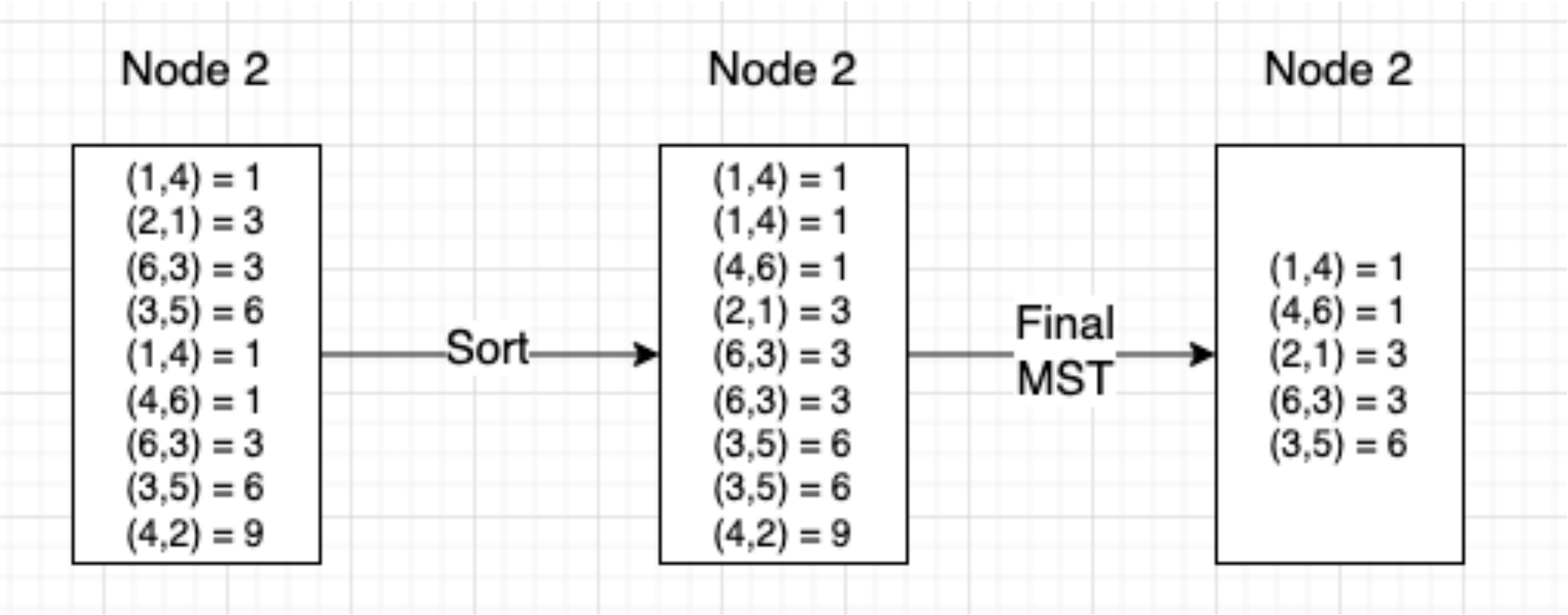
- Division of vertices equally across nodes
- Sort based on weights available in each node
- Find local minimum spanning tree for each node parallelly
- Merge two Local MST of nodes into one by forming a new MST
- Perform above operation until only one node is left

Example



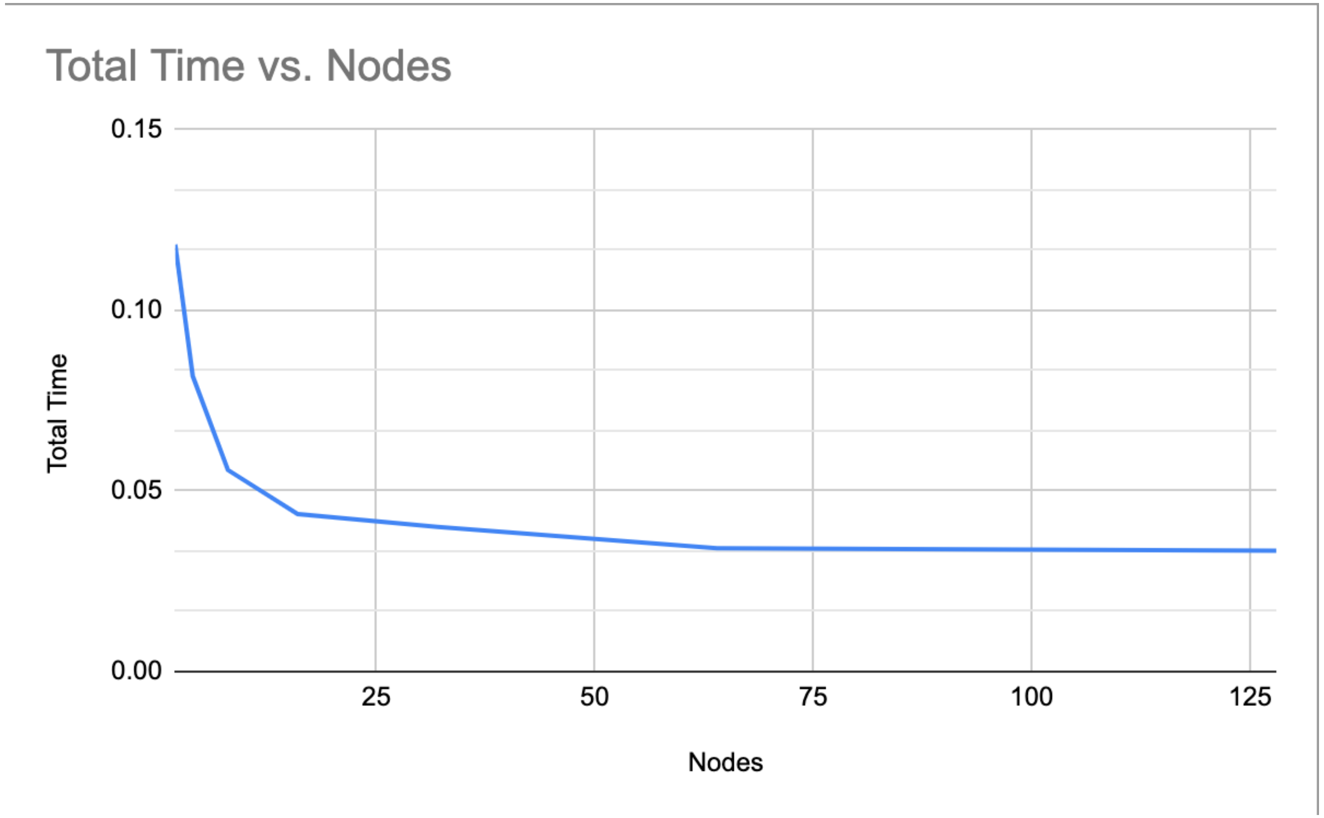
Input
(1,4) = 1
(2,1) = 3
(3,5) = 6
(4,2) = 9
(6,3) = 3
(5,6) = 7
(4,6) = 1
Node 1
(1,4) = 1
(2,1) = 3
(3,5) = 6
(4,2) = 9
(6,3) = 3
Sort
Node 1
(1,4) = 1
(2,1) = 3
(6,3) = 3
(3,5) = 6
(4,2) = 9
Local MST
Node 1
(1,4) = 1
(2,1) = 3
(6,3) = 3
(3,5) = 6
Send Local MST
Node 2
(1,4) = 1
(3,5) = 6
(4,2) = 9
(6,3) = 3
(5,6) = 7
(4,6)= 1
Sort
Node 2
(1,4) = 1
(4,6) = 1
(6,3) = 3
(3,5) = 6
(5,6) = 7
(4,2) = 9
Local MST
Node 2
(1,4) = 1
(4,6) = 1
(6,3) = 3
(3,5) = 6
(4,2) = 9
Node 2
(1,4) = 1
(2,1) = 3
(6,3) = 3
(3,5) = 6
(1,4) = 1
(4,6) = 1
(6,3) = 3
(3,5) = 6
(4,2) = 9

Example cont..



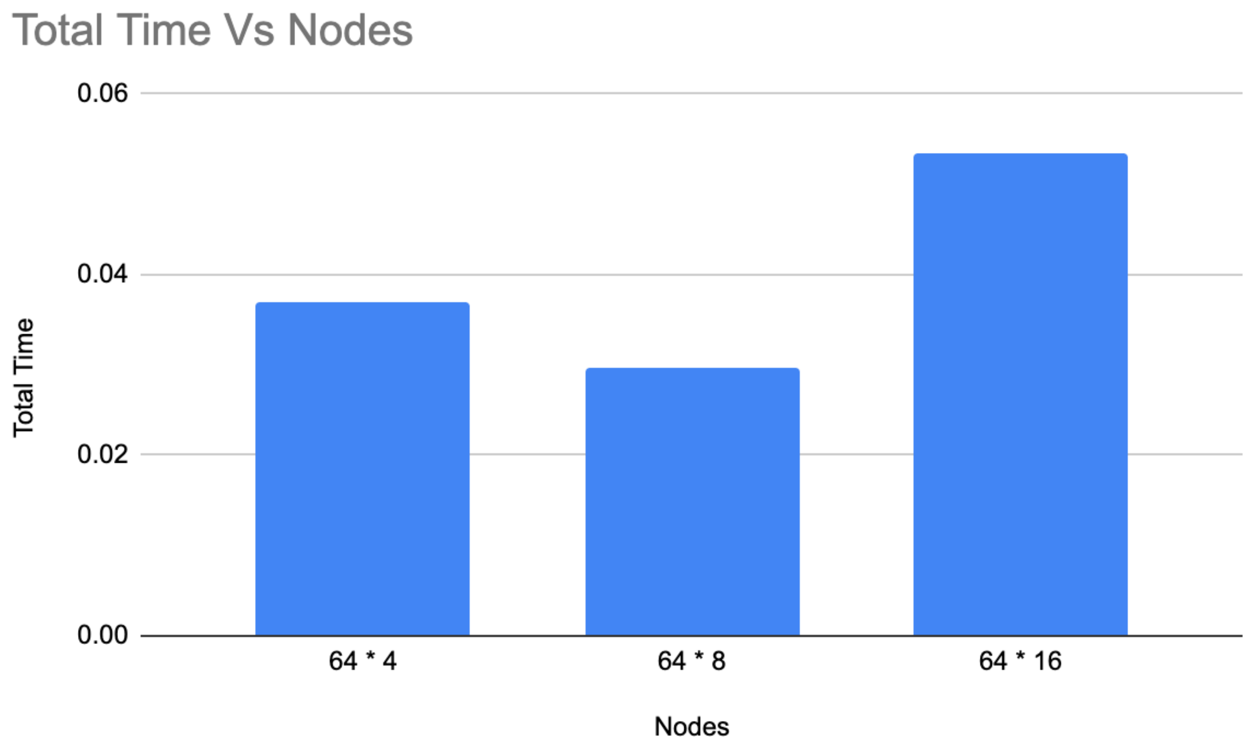
10000 Vertices ~ 500K Edges Input

Nodes	Total Time
2	0.1182135
4	0.0817805
8	0.0557865
16	0.043617
32	0.040047
64	0.0341535
128	0.0334475



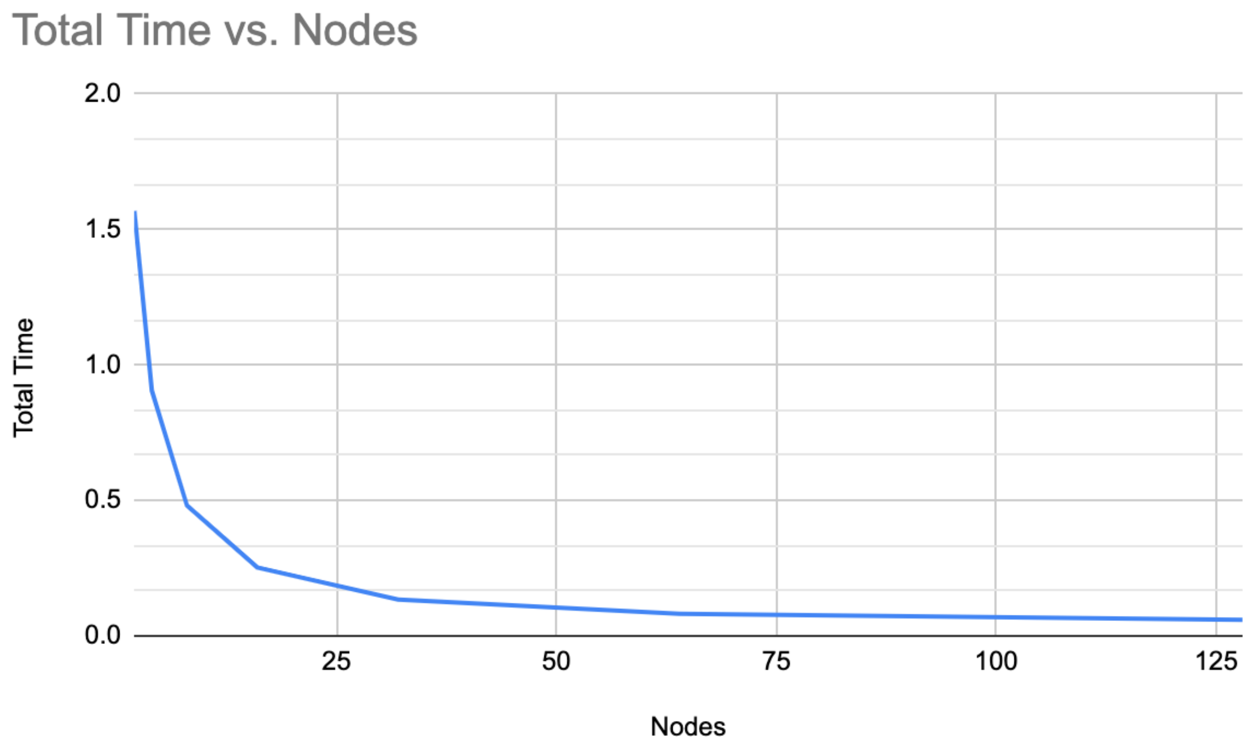
10000 Vertices ~ 500K Edges Input cont..

Nodes * Processors	Total Time
64 * 4	0.0369475
64 * 8	0.029645
64 * 16	0.0535355



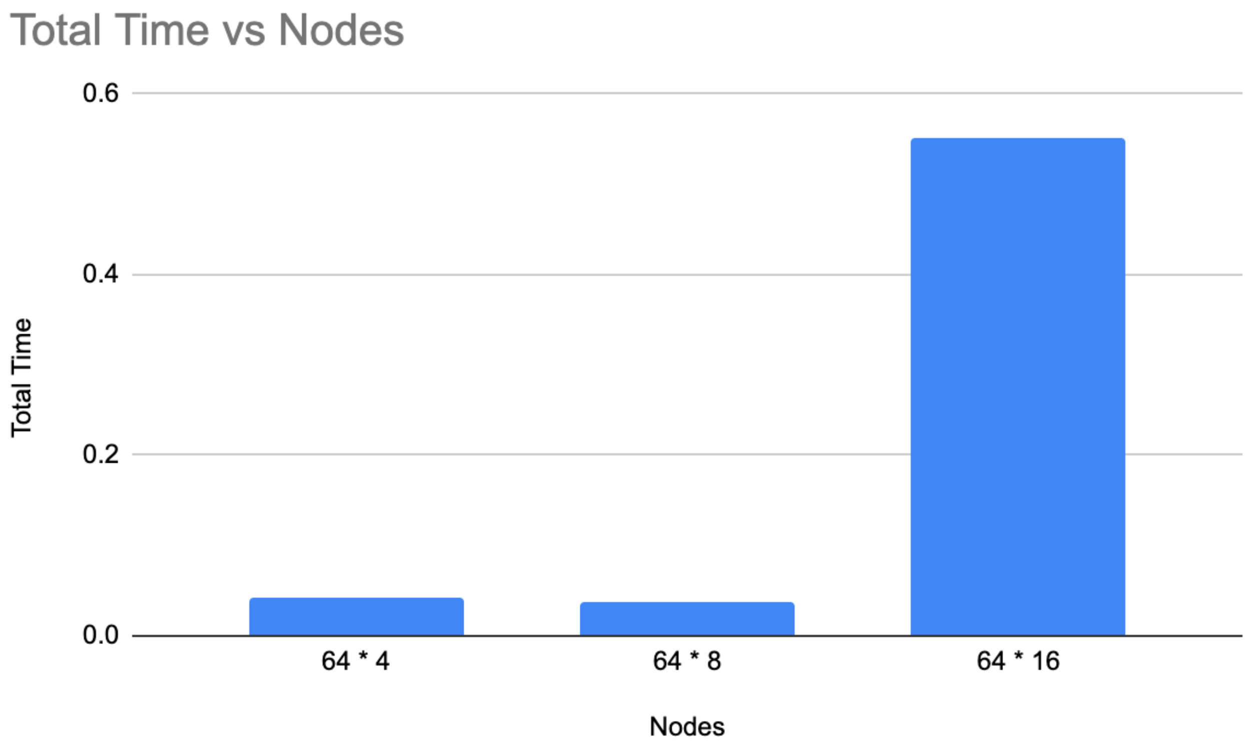
10000 Vertices ~ 10M Edges Input

Nodes	Total Time
2	1.5686585
4	0.905262
8	0.481359
16	0.2527845
32	0.1342505
64	0.081823
128	0.059391



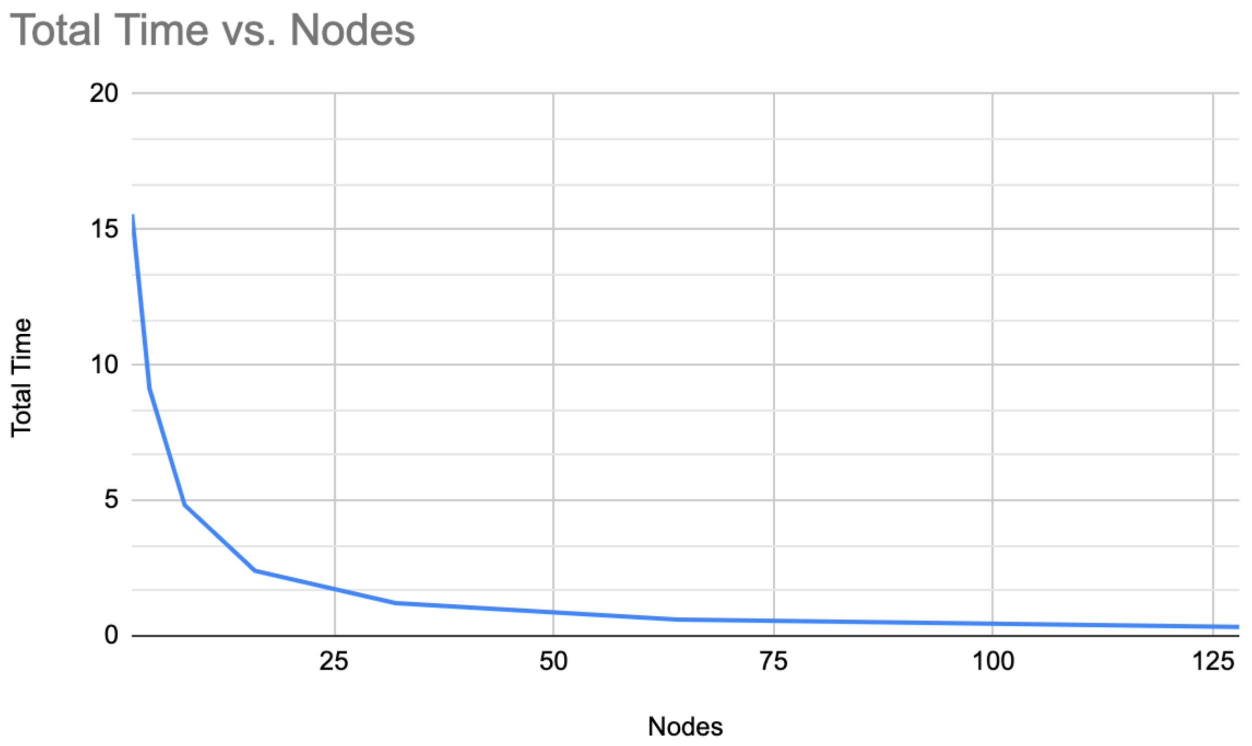
10000 Vertices ~ 10M Edges Input cont..

Nodes * Processors	Total Time
64 * 4	0.0422345
64 * 8	0.038256
64 * 16	0.551133



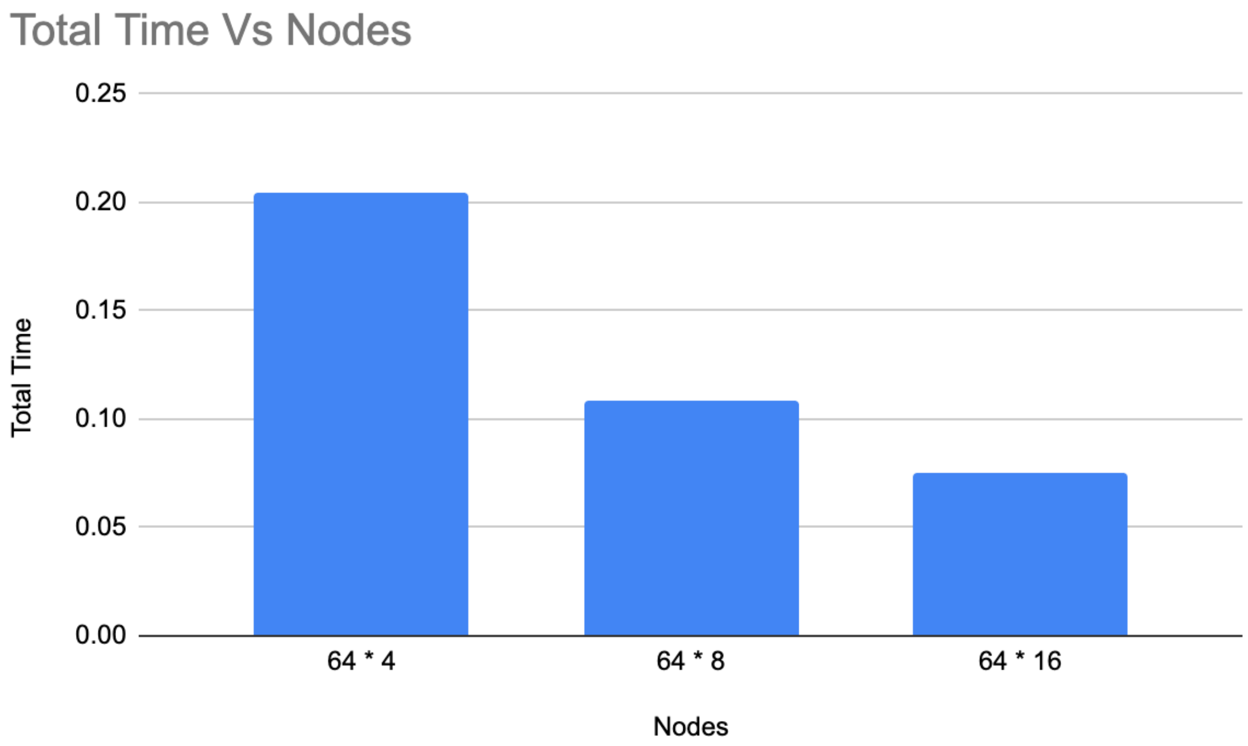
10000 Vertices ~ 37M Edges Input

Nodes	Total Time
2	15.5647325
4	9.1235645
8	4.829683
16	2.4028045
32	1.206955
64	0.6060085
128	0.3277855



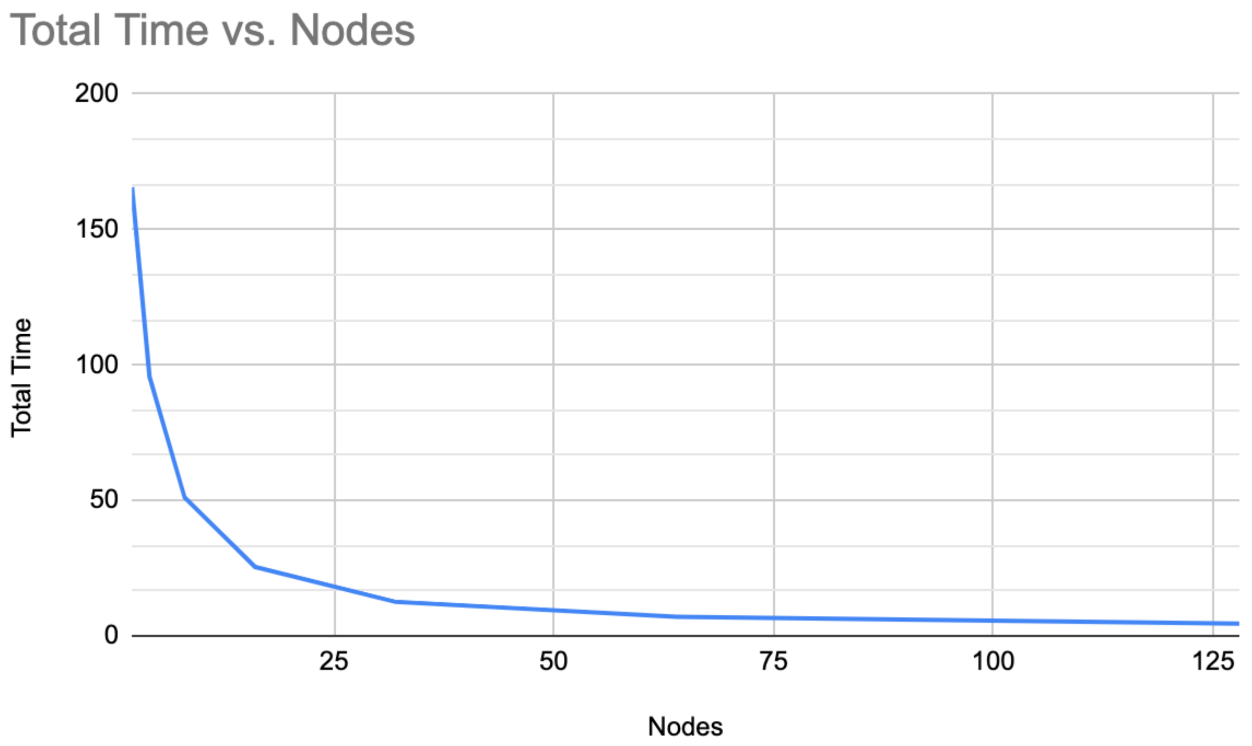
10000 Vertices ~ 37M Edges Input cont..

Nodes * Processors	Total Time
64 * 4	0.2048545
64 * 8	0.108445
64 * 16	0.0752625



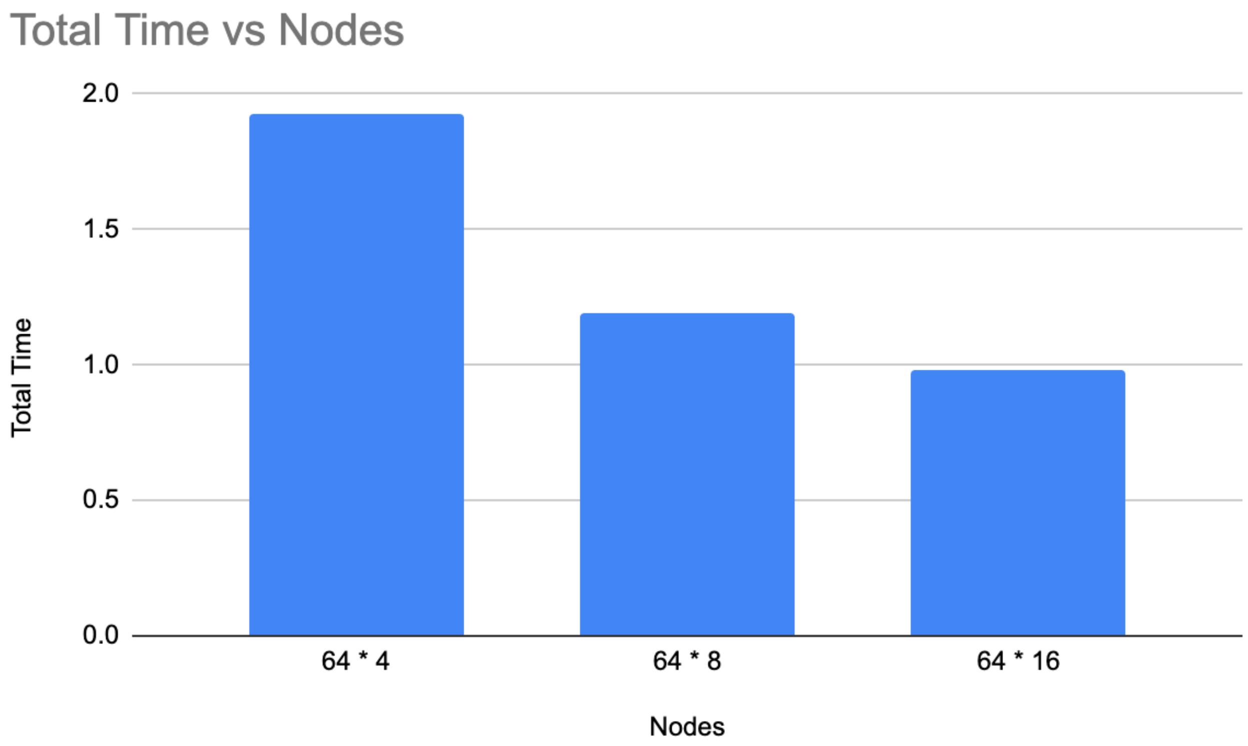
30000 Vertices ~ 337M Edges Input

Nodes	Total Time
2	165.577867
4	95.614936
8	51.074432
16	25.4376815
32	12.55099
64	7.036906
128	4.462514



30000 Vertices ~ 337M Edges Input cont..

Nodes * Processors	Total Time
64 * 4	1.9292105
64 * 8	1.191119
64 * 16	0.979804



Observations

- Majority Time spent on Sorting
- More Nodes -> More Communication Over head
- Scalability is good in Kruskals Algorithm.

Questions?

References

- https://en.wikipedia.org/wiki/Minimum_spanning_tree#Algorithms
- https://www.simplilearn.com/tutorials/data-structure-tutorial/kruskal-algorithm
- V. Osipov, P. Sanders, and J. Singler, "The filter-kruskal minimum spanning tree algorithm," in *Proceedings of theMeeting on Algorithm Engineering & Expermiments*, pp. 52–61, Society for Industrial and Applied Mathematics, 2009.
- https://www.cs.unm.edu/~treport/tr/03-12/MST-bader.pdf

Thank You