

Minimum Spanning Tree

-

DPHPC

Th. Cambier R. Dang-Nhu Th. Dardinier C. Trassoudaine

ETH Zürich

December 2018



- 1 Problem definition - reminder
 - The MST Problem
 - Use cases
- 2 Setup
 - Software
 - Hardware
- 3 Algorithms and parallel implementations
 - Base serial algorithms
 - Parallel improvements
- 4 Results overview

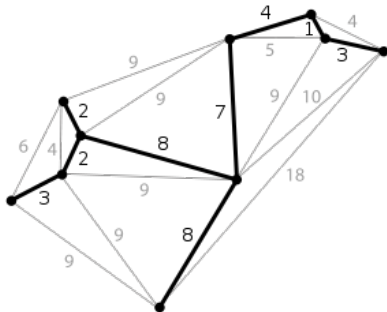


Problem definition - reminder



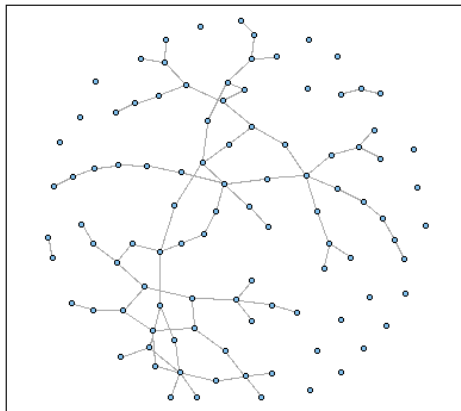
The MST problem

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted (un)directed graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.



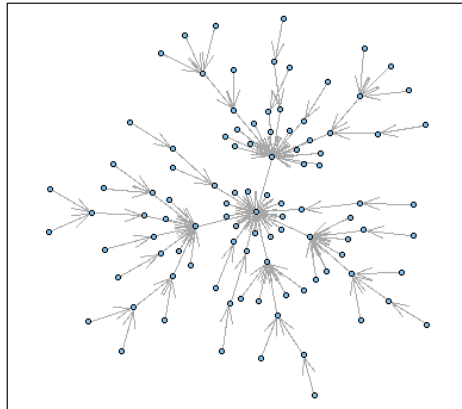
Input sets: $G(n, p)$

$G(100, 0.02)$



Input sets: $PA(n, m)$

$PA(100, 1)$



Input sets: 9th DIMACS challenge dataset

USA Roads

Name	Description	# nodes	# arcs
USA	Full USA	23,947,347	58,333,344
CTR	Central USA	14,081,816	34,292,496
W	Western USA	6,262,104	15,248,146
E	Eastern USA	3,598,623	8,778,114
LKS	Great Lakes	2,758,119	6,885,658
CAL	California and Nevada	1,890,815	4,657,742
NE	Northeast USA	1,524,453	3,897,636
NW	Northwest USA	1,207,945	2,840,208
FLA	Florida	1,070,376	2,712,798
COL	Colorado	435,666	1,057,066
BAY	San Francisco Bay Area	321,270	800,172
NY	New York City	264,346	733,846



Setup



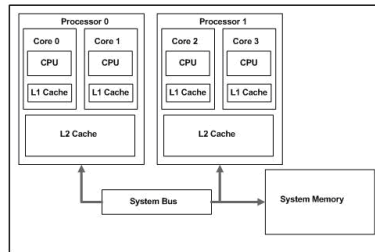
Software

- OMP
- Intel Threading Building Blocks (TBB)
- Intel Parallel Standard Template Library (PSTL)



EULER

- 1 node limitation (OMP)
- 2 sockets filled with 18 cores - up to 3.7Ghz
- Inter-sockets bus speed: 10.4 GT/s



Algorithms and parallel implementations



Sollin

- 1 For each connected component, find adjacent edge with minimum weight
- 2 Add edge to mst (each edge add at most twice)
- 3 Merge connected components



Kruskal

```
1:  $A = \emptyset$ 
2: for all  $v \in G.V$  do
3:   MAKE-SET( $v$ )
4: end for
5: Sort (asc.)  $(weight(u, v))_{(u,v) \in G.E}$ 
6: for all  $(u, v)$  in  $G.E$  ordered by weight do
7:   if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then
8:      $A = A \cup (u, v)$ 
9:     UNION( $u, v$ )
10:  end if
11: end for
12: return  $A$ 
```



Boost implementations

- Boost-Kruskal used as a reference



Parallel sorting on Kruskal



Filter Kruskal



Amdahl's law

S_p is the speedup with p cores, f is the part of the program that is sequential.

$$S_p = \frac{1}{\frac{1-f}{p} + f}$$

$$f = \frac{\frac{p}{S_p} - 1}{p - 1}$$



Amdahl's law: Kruskal

Graph: Erdos-Renyi (100,000 nodes, $p = 0.0005$)

Cores	Median speed-up	Standard deviation	f
1	1	0.0129805395	-
2	1.1881513396	0.0400305172	0.6832872491
4	1.3340592415	0.0130798641	0.6661225318
8	1.3134515984	0.01048841	0.7272602975
16	1.4399618642	0.0088220116	0.6740936918
32	1.3877640643	0.0442081933	0.7115701488



Amdahl's law: Filter Kruskal

Graph: Preferential attachment (10,000 nodes, 1,000 edges per vertex)

Cores	Median speed-up	Standard deviation	f
1	1	0.174182691	-
2	1.6268639022	0.2472258016	0.2293591353
4	2.669168898	0.5213238241	0.1661979381
8	5.3584384444	0.9757941767	0.0704246098
16	5.7447285937	1.108109986	0.1190108026
32	5.6322979489	1.497976882	0.1510166972



Kruskal vs Sollin

- Same complexity, but Sollin computing steps are more fine-grained
- Sollin is entirely paralelizable
- Filter Kruskal can be adapted to Filter Sollin

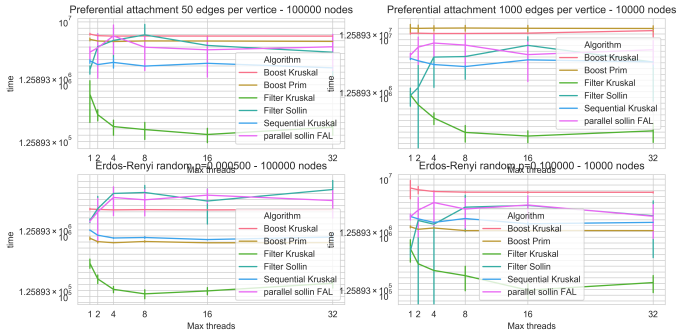


Results overview



Runtimes

Runtime per graph



Speedups

Speedup per graph compared to serial implementation

