An Implementation of Parallelizing Dijkstra's Algorithm

CSE633 Course Project

Zilong Ye
ID: 3715-8138
zilongye@buffalo.edu

π

Outline

- Problem statement
- Dijkstra’s algorithm
- Parallel Dijkstra’s algorithm
- Simulation results and analysis
- Reference

Problem statement

- Given a graph, Let $G = (V, E)$ be a directed graph, $|V| = n$, $|E| = m$, let s be a distinguished vertex of the graph, and w be the non-negative value to the weight of each edge, which represents the distance between the two vertexes.
- Single source shortest path: The single source shortest path (SSSP) problem is that of computing, for a given source vertex s and a destination vertex t, the weight of a path that obtains the minimum weight among all the possible paths.

Dijkstra's algorithm

- Dijkstra's algorithm is a graph search algorithm that solves single-source shortest path for a graph with nonnegative weights.
- Widely used in network routing protocol, e.g., Open Shortest Path First (OSPF) protocol.

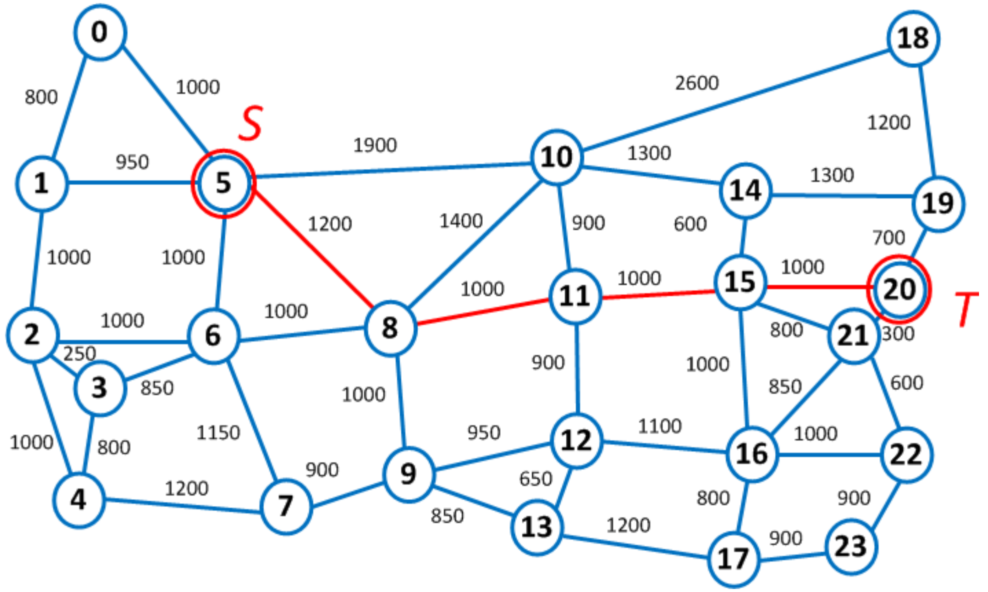


Fig. 1   24-node U.S. mesh network

Dijkstra’s algorithm

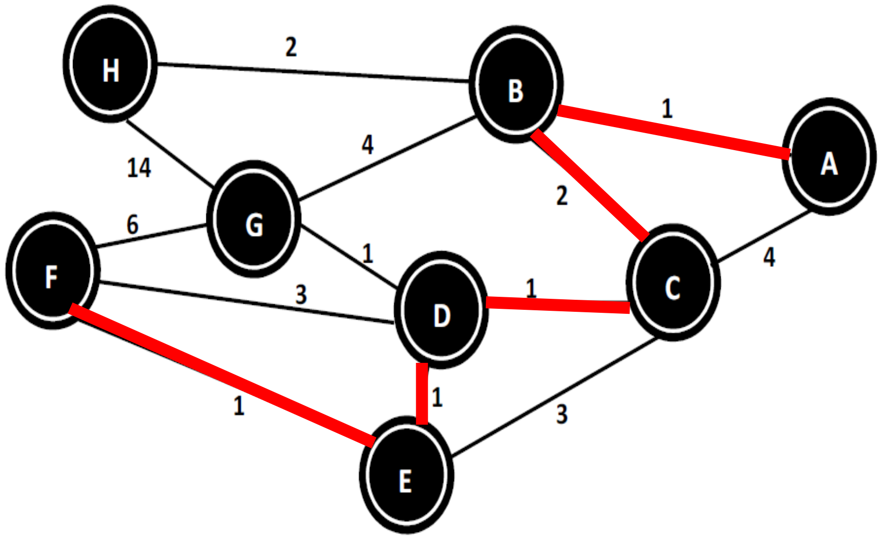


Fig. 2 8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			4, C	6, C	∞	5, B	
ABCHD				5, D	7, D	5, B	
ABCHDE					6, E	5, B	
ABCHDEG					6, E		
ABCHDEGF							

Table 1. The routing table for node A

Dijkstra's algorithm-1st round

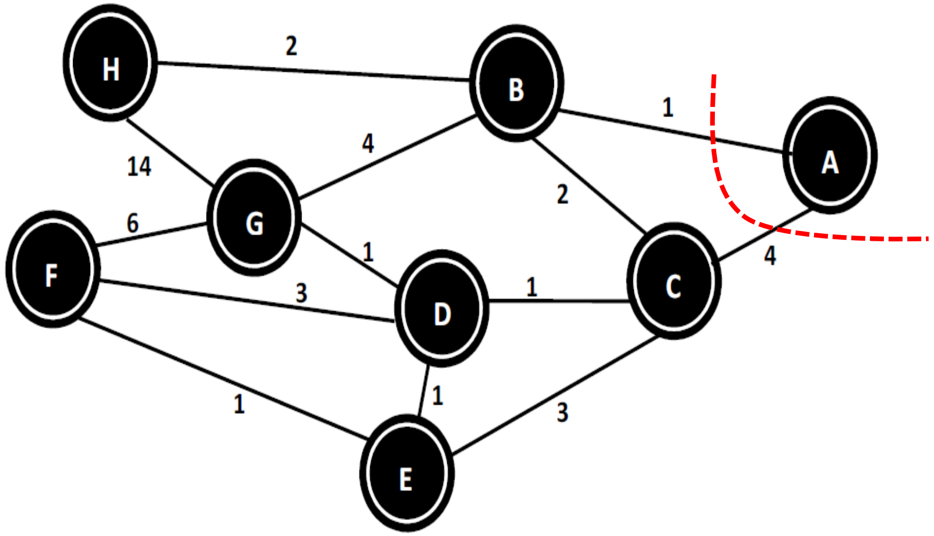


Fig. 2 8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB							

Table 1. The routing table for node A

Dijkstra's algorithm-2nd round

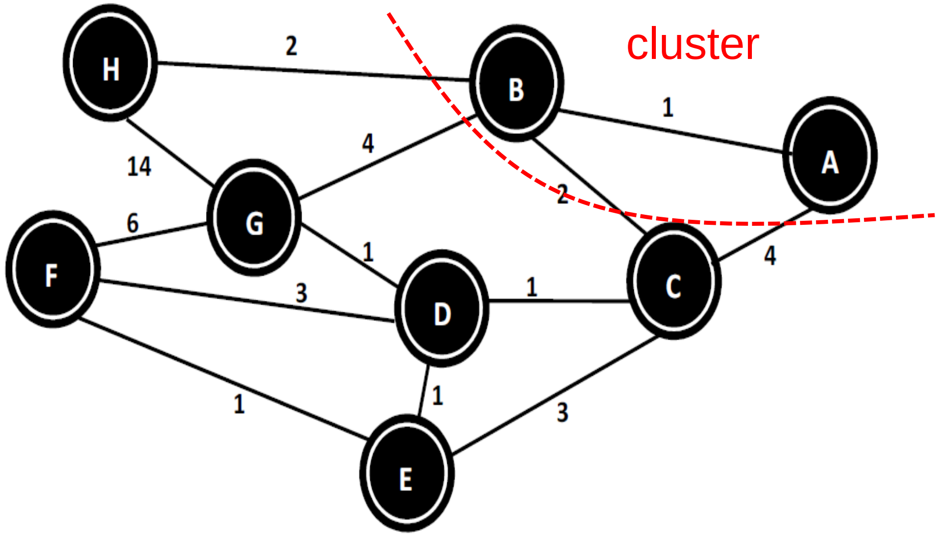


Fig. 2   8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC							

Table 1.   The routing table for node A

Dijkstra's algorithm-3rd round

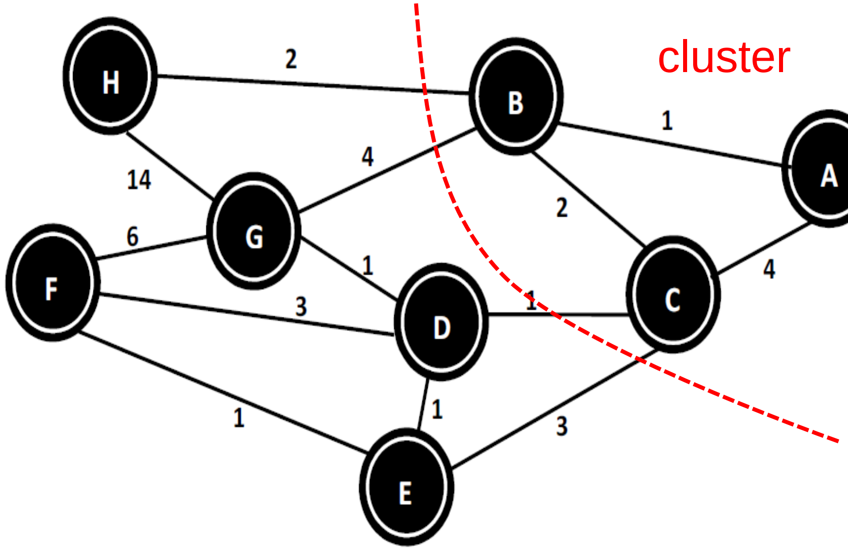


Fig. 2   8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH							

Table 1.   The routing table for node A

Dijkstra’s algorithm-4th round

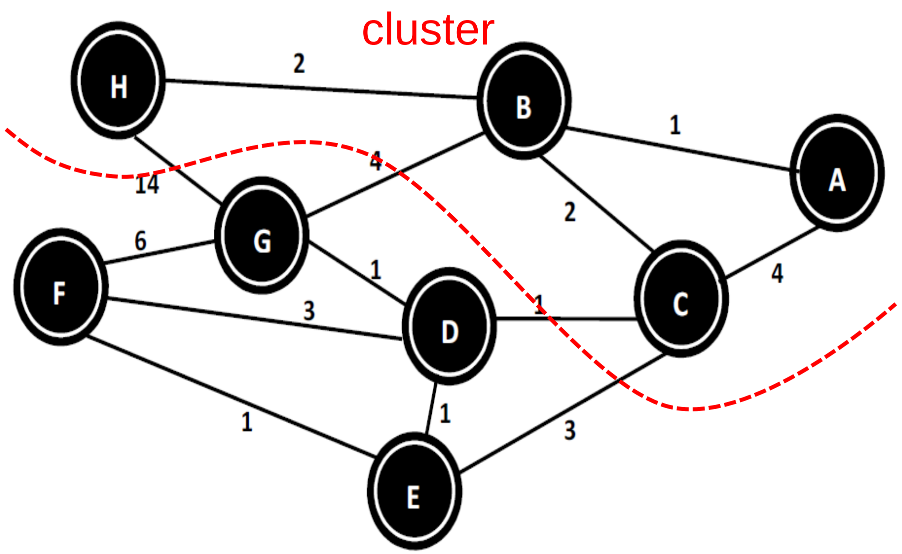


Fig. 2   8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			$\boxed{4, C}$	6, C	∞	5, B	
ABCHD							

Table 1.   The routing table for node A

Dijkstra's algorithm-5th round

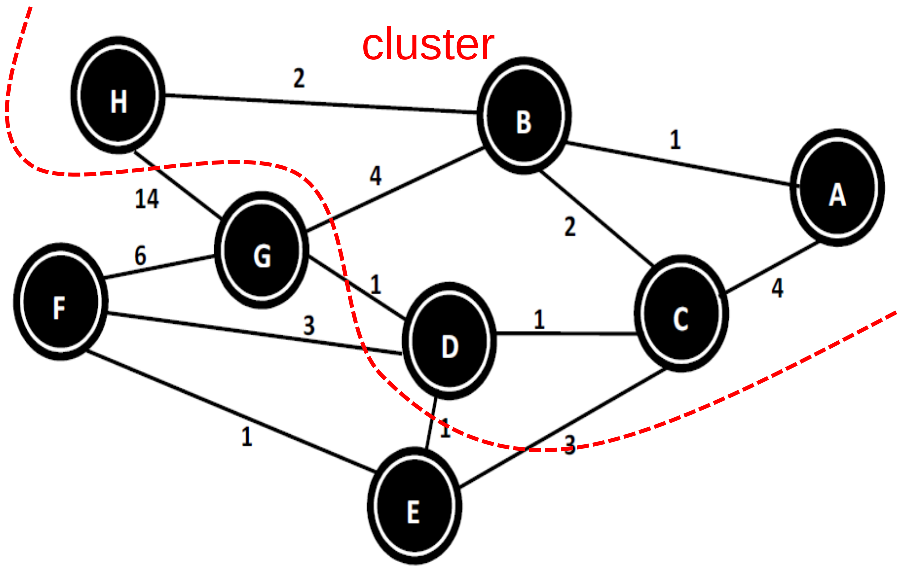


Fig. 2 8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			4, C	6, C	∞	5, B	
ABCHD				$\boxed{5, D}$	7, D	5, B	
ABCHDE							

Table 1. The routing table for node A

Dijkstra’s algorithm-6th round

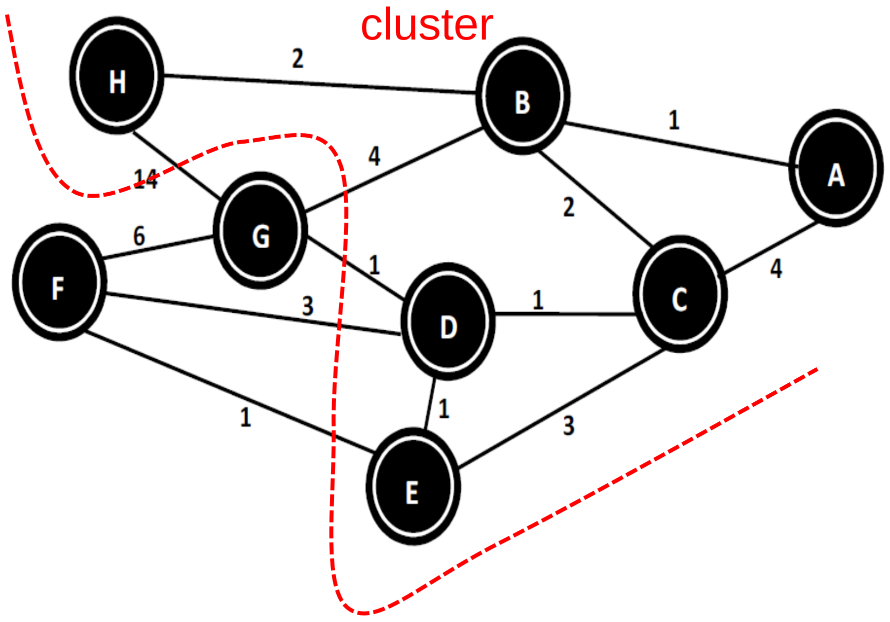


Fig. 2 8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			4, C	6, C	∞	5, B	
ABCHD				5, D	7, D	5, B	
ABCHDE					6, E	$\boxed{5, B}$	
ABCHDEG					6, E		

Table 1. The routing table for node A

Dijkstra's algorithm-6th round

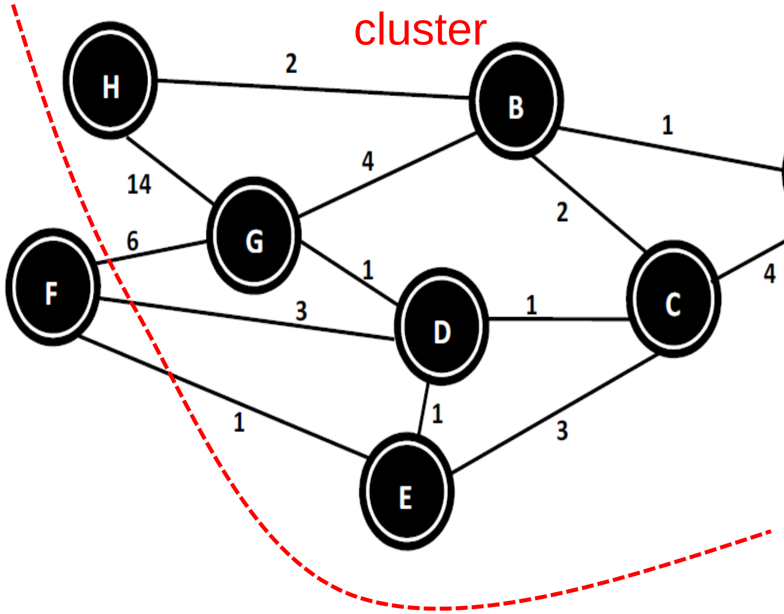


Fig. 2 8-node simple network

	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			4, C	6, C	∞	5, B	
ABCHD				5, D	7, D	5, B	
ABCHDE					6, E	5, B	
ABCHDEG					6, E		
ABCHDEGF							

Table 1. The routing table for node A

Sequential Dijkstra's algorithm

```
Create a cluster cl[V]
Given a source vertex s
While (there exist a vertex that is not in the cluster cl[V])
{
    FOR (all the vertices outside the cluster)
        Calculate the distance from non-member vertex
        to s through the cluster
    END
    ** O(V) **
    Select the vertex with the shortest path and add it to
    the cluster
    ** O(V) **
}
```

Dijkstra's algorithm

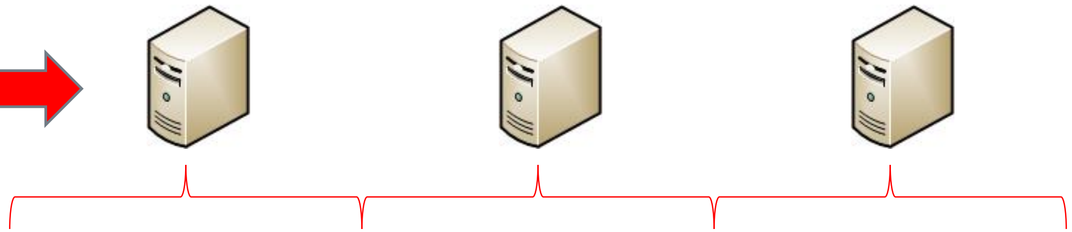
- Running time $O(V^2)$
  - In order to obtain the routing table, we need *O(V)* rounds iterations (until all the vertices are included in the cluster). In each round, we will update the value for O(*V)* vertices and select the closest vertex, so the running time in each round is *O(V)*. So, the total running time is $O(V^2)$.
- Disadvantages:
  - If the scale of the network is too large, then it will cost a long time to obtain the result.
  - For some time-sensitive app or real-time services, we need to reduce the running time.

Parallel Dijkstra’s algorithm

- Approach:
  - Each core identifies its closest vertex to the source vertex;
  - Perform a parallel prefix to select the globally closest vertex;
  - Broadcast the result to all the cores;
  - Each core updates its cluster list.

Parallel Dijkstra’s algorithm

- Step 1: find the closest node in my subgroup.
- Step 2: use parallel prefix to find the global closest.



	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)	(d, n)
cluster	B	C	D	E	F	G	H
A	1, A	4, A	∞	∞	∞	∞	∞
AB		3, B	∞	∞	∞	5, B	3, B
ABC			4, C	6, C	∞	5, B	3, B
ABCH			4, C	6, C	∞	5, B	
ABCHD				5, D	7, D	5, B	
ABCHDE					6, E	5, B	
ABCHDEG					6, E		
ABCHDEGF							

Parallel Dijkstra's algorithm

```
Create a cluster cl[V]
Given a source vertex s
Each core handles a subgroup of V/P vertices
While (there exist a vertex that is not in the cluster cl[V])
{
    FOR (vertices in my subgroup but outside the cluster)
        Calculate the distance from non-member vertex to s
        through the cluster;
        Select the vertex with the shortest path as the local
        closest vertex;
    END
    ** Each processor work in parallel O(V/P) **
    Use the parallel prefix to find the global closest vertex
    among all the local closest vertices from each core.
     ** Parallel prefix log(P) **
}
```

Parallel Dijkstra's algorithm

- Running time $O\left(\frac{V^2}{P} + V \cdot \log(P)\right)$
  - *P* is the number of cores used. In order to obtain the routing table, we need *O(V)* rounds iteration (until all the vertices are included in the cluster). In each round, we will update the value for *O(V)* vertices using *P* cores running independently, and use the parallel prefix to select the global closest vertex, so the running time in each round is *O(V/P)+O(log(P)).* So, the total running time is $O\left(\frac{V^2}{P} + V \cdot \log(P)\right)$.

Simulation results and analysis

- Experiment 1:
  - Run on one 32-core node, with different size of mesh network model (50*50, 100*100, 150*150).
  - Analyze the performance in terms of different size of network
- Experiment 2:
  - The mesh network size is fixed-150*150. The task is run on one 32-core node, three 12-core nodes, sixteen 2-core nodes, respectively.
  - Analyze the performance in terms of different distribute way.
- Implement using OpenMP and all the statistics are the average values for 10 rounds of running.

Experiment 1

- The running time
  - It is obvious that, for the large size network (150*150), the running time is decreasing as the number of cores increases until it reaches the smallest value, then the running time will increase because of the communication latency.
  - For middle size network (100*100), the phenomenon of a reducing running time is not that obvious.
  - For a small size network (50*50), the running time is even increasing as the number of cores increases, because the communication latency outperforms the benefit from using more cores.

	1	2	4	8	16	32
50*50	0.06587	0.04175	0.03268	0.04238	0.07257	0.23035
100*100	1.04358	0.55511	0.30676	0.23684	0.26861	0.44056
150*150	5.23908	2.69014	1.43890	0.83117	0.77554	1.12642

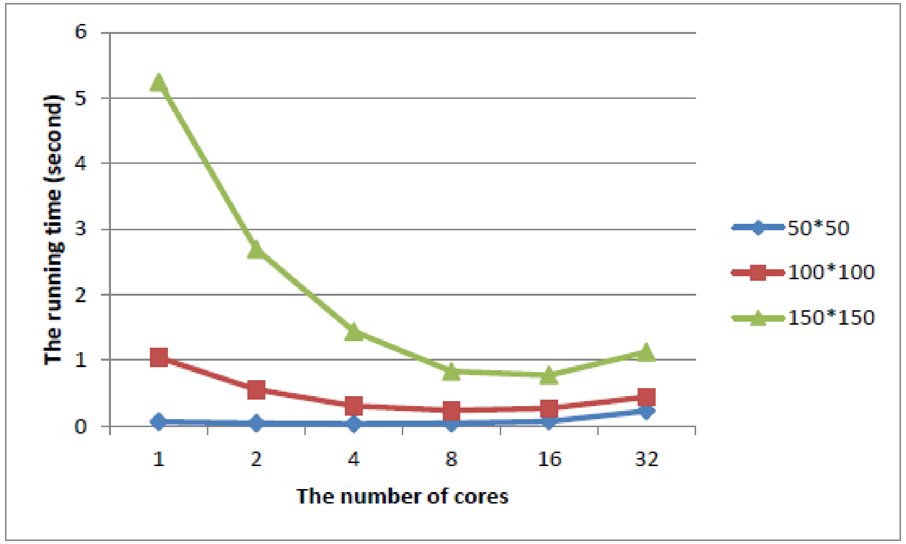


Fig. 3 The running time v.s. the number of cores

Experiment 1

- The speed up
  - The speed up is increasing as the number of cores increases until it reaches the maximum value, then the speed up is decreasing.
  - The speed up is increasing because of using more cores.
  - The speed up is decreasing because the communication latency outperforms the benefit from using more cores.
  - As the network size increases, the number of cores used to get the maximum speed up increases. (As shown in the figure, 50*50-4 cores, 100*100-8 cores, 150*150-16 cores)

	1	2	4	8	16	32
50*50	1	1.57755	2.01554	1.55425	0.90770	0.28596
100*100	1	1.87995	3.40185	4.40609	3.88505	2.36871
150*150	1	1.94751	3.64102	6.30324	6.7554	4.65106

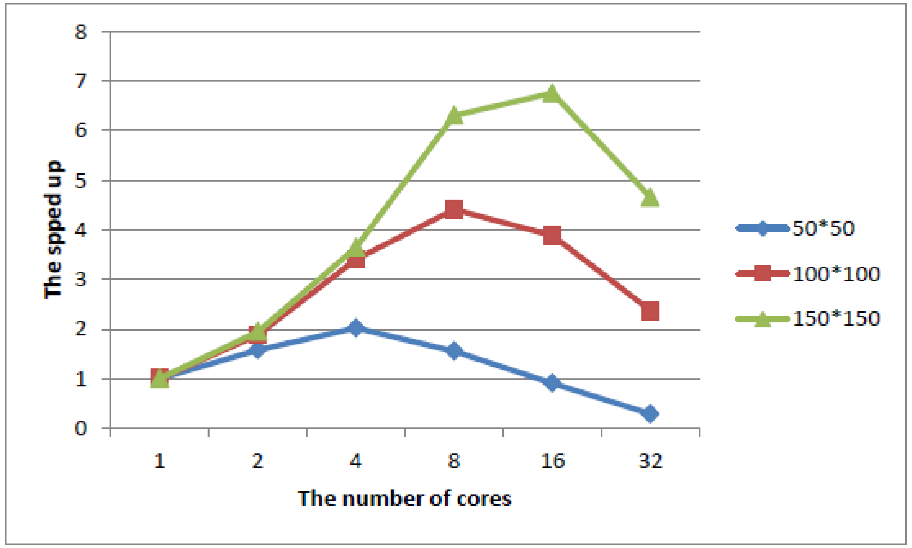


Fig. 4 The speed up v.s. the number of cores

Experiment 1

- The cost
  - The cost is increasing because the speed up (or the benefit of a reduced running time) cannot outperforms the cost of using more cores.

	1	2	4	8	16	32
50*50	0.06587	0.08351	0.13073	0.33906	1.16115	7.37132
100*100	1.04358	1.11022	1.22707	1.89479	4.29782	14.0981
150*150	5.23908	5.38029	5.75562	6.64937	12.4086	36.0456

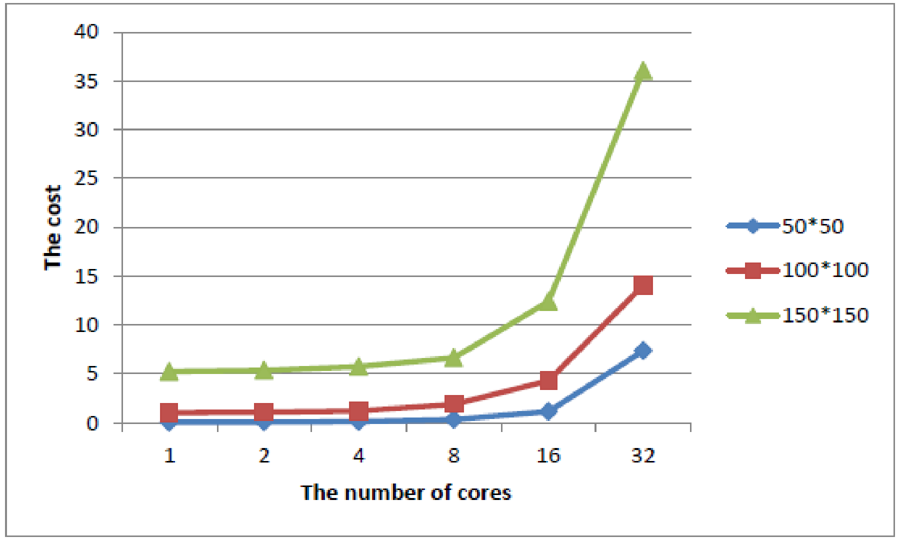


Fig. 5 The cost v.s. the number of cores

Experiment 2

- The running time
  - The running time is decreasing as the number of cores increases when all the cores are in the same node.
  - When cores from different nodes are used, the running time is increasing dramatically as shown for 16*2-core and 3*12-core

	1	2	4	8	16	32
16*2-core	4.37263	2.36723	3.97442	5.38834	7.91071	12.9382
3*12-core	4.65692	2.40176	1.24577	0.69465	2.58422	5.41149
1*32-core	5.23908	2.69014	1.43890	0.83117	0.77554	1.12642

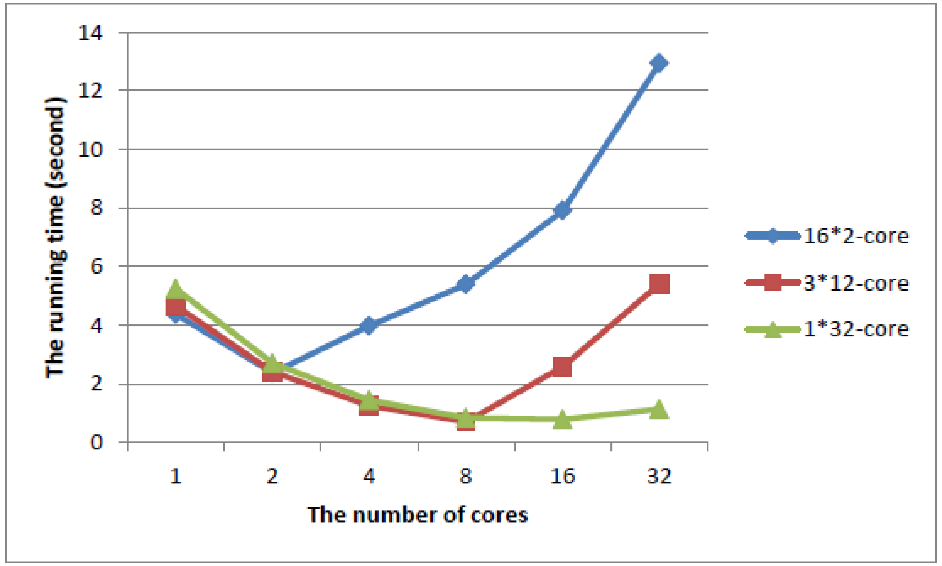


Fig. 6 The running time v.s. the number of cores

Experiment 2

- The speed up
  - The speed up is increasing as the number of cores increases if the cores are from the same node.
  - When cores from different nodes are used, the speed up is decreasing significantly as shown for 16*2-core and 3*12-core.

	1	2	4	8	16	32
16*2-core	1	1.84715	1.10019	0.81149	0.55274	0.33796
3*12-core	1	1.93895	3.73818	6.70394	1.80205	0.86056
1*32-core	1	1.94751	3.64102	6.30324	6.7554	4.65106

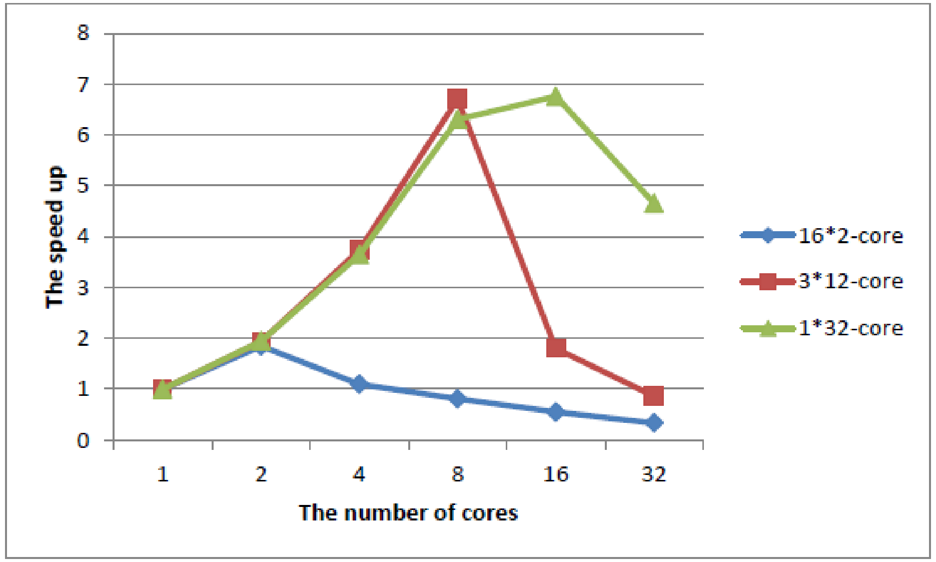


Fig. 7 The speed up v.s. the number of cores

Experiment 2

- The cost
  - The cost is increasing as the number of cores increases.
  - The cost of a 16*2-core is much higher than the cost of 3*12-core and 1*32-core.

	1	2	4	8	16	32
16*2-core	4.37263	4.73446	15.8976	43.1067	126.571	414.024
3*12-core	4.65692	4.80353	4.98308	5.55720	41.3475	173.167
1*32-core	5.23908	5.38029	5.75562	6.64937	12.4086	36.0456

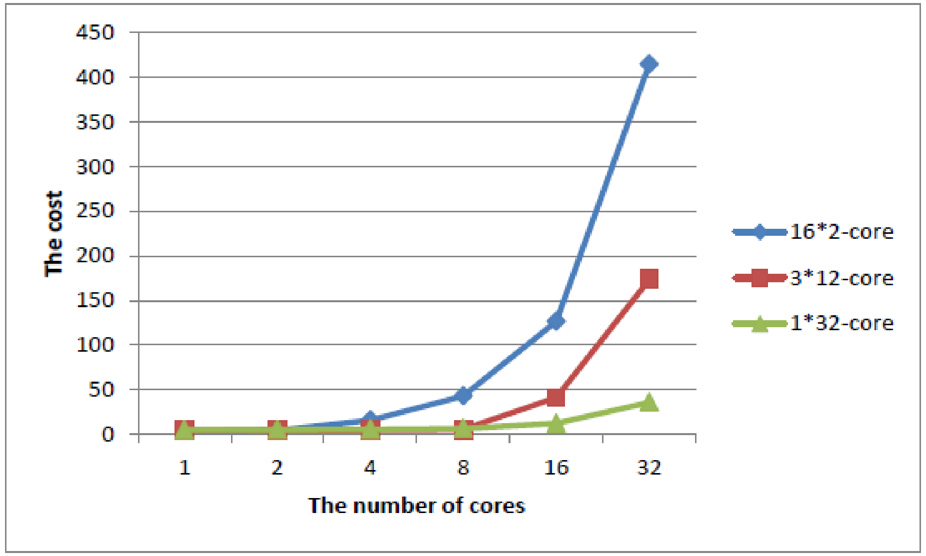


Fig. 3 The cost v.s. the number of cores

Reference

- Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs,". Numerische Mathematik 1: 269–271. doi:10.1007/BF01386390.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw-Hill. pp. 595–601. ISBN 0-262-03293-7.
- A. Crauser, K. Mehlhorn, U. Meyer, P. Sanders, “A parallelization of Dijikstra’s shortest path algorithm”, in Proc. of MFCS’98, pp. 722-731, 1998.
- Y. Tang, Y. Zhang, H. Chen, “A Parallel Shortest Path Algorithm Based on Graph-Partitioning and Iterative Correcting”, in Proc. of IEEE HPCC’08, pp. 155-161, 2008.
- G. Stefano, A. Petricola, C. Zaroliagis, “On the implementation of parallel shortest path algorithms on a supercomputer”, in Proc. of ISPA’06, pp. 406-417, 2006.

Questions?

Thank you!