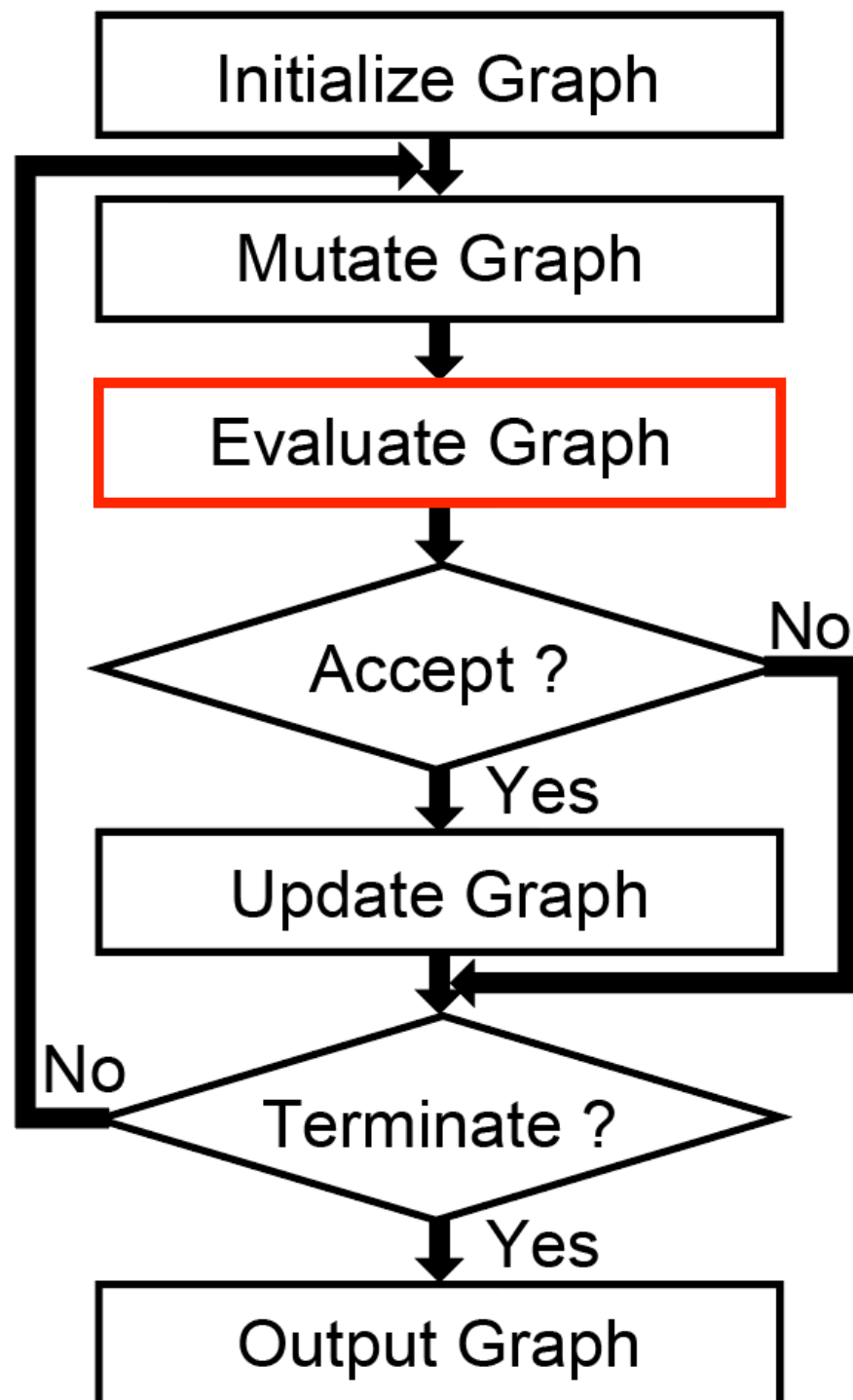


Introduction of fast APSP algorithm and optimization algorithms for grid graphs

Masahiro Nakao,[†] Maaki Sakai,[‡] Yoshiko Hanada[‡]

([†]RIKEN R-CCS, [‡]Kansai University)

Obtain ASPL and diameter



- Metaheuristic such as SA and GA are often used
- To evaluate a graph, its diameter and ASPL are needed which are calculated by APSP algorithm **many times**
- **It is very important to calculate APSP at high-speed**

e.g. For a problem $(n, d) = (1M, 32)$, the time required for one APSP is about 37 hours by the method based on Breadth-First Search (BFS) on Intel Gold 6126

Our APSP algorithms

- Our previous research provides a parallel APSP algorithm based on BFS (BFS-APSP) [1-3]
- This presentation introduces a new parallel APSP algorithm based on adjacency matrix (ADJ-APSP) [4-5]. The original ADJ-APSP was developed by Ryuhei Mori [6]

You can download our program from
<https://github.com/mnakao/APSP/>



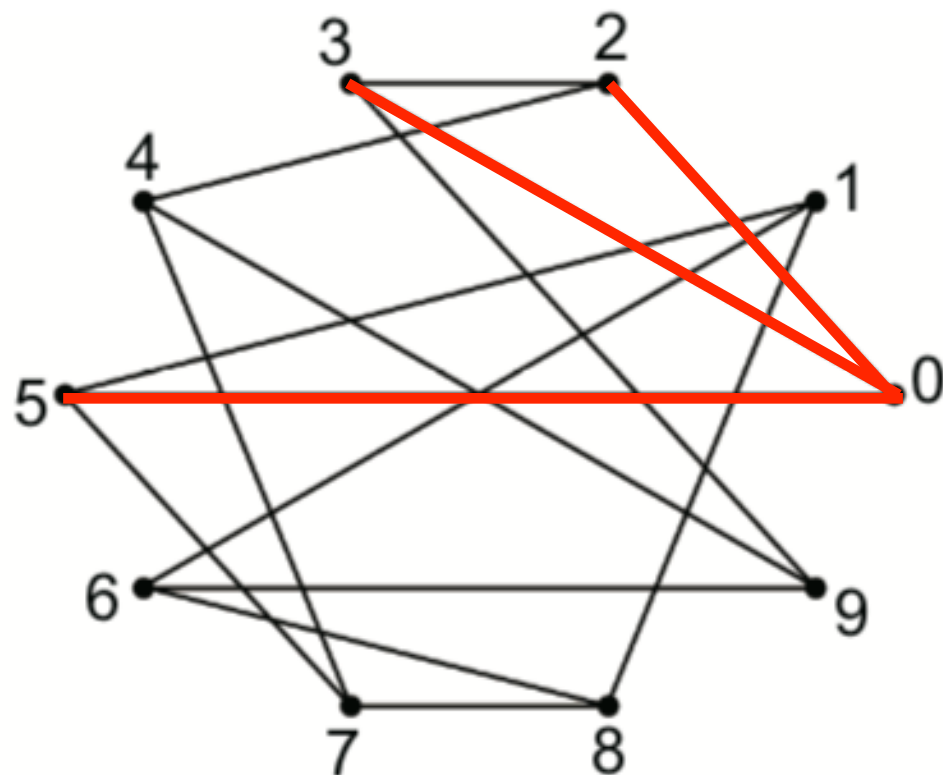
- [1] 中尾昌広ほか. MPI/OpenMP並列によるグラフ対称性とSimulated Annealingを用いたOrder/Degree問題の一解法, HPC研究会. 2018.
- [2] Masahiro Nakao et. al. A Method for Order/Degree Problem Based on Graph Symmetry and Simulated Annealing with MPI/OpenMP Parallelization, HPC Asia 2019.
- [3] 中尾昌広ほか. 大規模Order/Degree問題に対する最適化アルゴリズムの並列化と解探索性能の評価. 計算工学講演会論文集, 2019.
- [4] 中尾昌広ほか. Order/Degree問題のための重みなしグラフにおける全点对間最短経路アルゴリズムの並列化. HPC研究会. 2019.
- [5] Masahiro Nakao et al. Parallelization of All-Pairs-Shortest-Path Algorithms in Unweighted Graph, HPC Asia 2020.
- [6] https://github.com/ryuhei-mori/graph_ASPL

ADJ-APSP(1 / 3)

- Let A be an adjacency matrix of a graph
- If the value of an element $a_{\{i, j\}}$ in A^k is 1, it means that the vertex i can reach the vertex j within k hops

```
for(int i=0;i<n;i++)
    A[0][i] |= A[2][i] | A[3][i] | A[5][i];
```

$(n, d) = (10, 3)$



A	adjlst	A^1
0 0 0 0 0 0 0 0 0 1	<u>2 3 5</u>	0 0 0 0 1 0 1 1 0 1
0 0 0 0 0 0 0 0 1 0	5 6 8	0 1 0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0	0 3 4	0 0 0 0 0 1 1 1 0 1
0 0 0 0 0 0 1 0 0 0	0 2 9	1 0 0 0 0 0 0 1 1 0 1
0 0 0 0 0 1 0 0 0 0	2 7 9	1 0 1 0 0 1 0 1 0 0
0 0 0 0 1 0 0 0 0 0	0 1 7	0 0 1 0 1 0 0 0 0 1 1
0 0 0 1 0 0 0 0 0 0	1 8 9	1 1 0 1 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0	4 5 8	0 1 1 0 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0	1 6 7	0 1 1 1 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 0	3 4 6	1 0 0 1 0 1 1 0 0 0

ADJ-APSP(2/3)

A	adjlst		A^1		A^2		A^3
0 0 0 0 0 0 0 0 0 1	2 3 5		0 0 0 0 1 0 1 1 0 1		1 0 1 0 1 1 1 1 1		1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 0	5 6 8		0 1 0 1 1 0 0 0 1 0		1 1 1 1 1 0 0 0 1 1		1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 1 0 0	0 3 4		0 0 0 0 0 1 1 1 0 1		1 0 1 0 1 1 1 1 0 1		1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 1 0 0 0	0 2 9		1 0 0 0 0 0 0 1 1 0 1		1 0 0 1 1 1 1 1 0 1		1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 1 0 0 0 0	2 7 9	→	1 0 1 0 0 1 0 1 0 0	→	1 1 1 1 1 1 1 1 0 1	→	1 1 1 1 1 1 1 1 1 1
0 0 0 0 1 0 0 0 0 0	0 1 7		0 0 1 0 1 0 0 0 1 1		0 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 1 1 1
0 0 0 1 0 0 0 0 0 0	1 8 9		1 1 0 1 0 0 0 0 1 0		1 1 1 1 1 1 1 0 1 0		1 1 1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0 0 0	4 5 8		0 1 1 0 1 1 0 0 0 0		1 1 1 1 1 1 0 1 1 1		1 1 1 1 1 1 1 1 1 1
0 1 0 0 0 0 0 0 0 0	1 6 7		0 1 1 1 0 0 0 0 1 0		1 1 1 1 1 1 0 0 1 0		1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0	3 4 6		1 0 0 1 0 1 1 0 0 0		1 1 1 1 0 1 1 1 1 1		1 1 1 1 1 1 1 1 1 1

- As k is increased in increments of 1, the value of k is the **diameter** when all elements are 1
- Every time k is increased from 1 to the diameter, **the total distance** is obtained by summing all the elements whose value is 0
 - ASPL** is calculated by dividing **the total distance** by the number of elements

ADJ-APSP(3/3)

```
1 function SERIAL_ADJ_APSP(vertices, nodes)
2   diameter  $\leftarrow$  1
3   distance  $\leftarrow$  nodes*(nodes-1)
4   elements  $\leftarrow$   $\lceil$ nodes/E $\rceil$ 
5   A, B  $\leftarrow$  INITIALIZE(nodes, elements)
6   for k=1 ... nodes-1
7     for i=1 ... nodes
8       for n  $\in$  neighbors(i, vertices)
9         for j=1 ... elements
10          B[i][j]  $\leftarrow$  B[i][j] | A[n][j]
11
12   num  $\leftarrow$  0
13   for i=1 ... nodes
14     for j=1 ... elements
15       num  $\leftarrow$  num+POPCNT(B[i][j])
16
17   if(num = nodes*nodes) break
18
19   SWAP(A, B)
20   diameter++
21   distance  $\leftarrow$  distance+(nodes*nodes-num)
22   average_distance  $\leftarrow$  distance/((nodes-1)*nodes)
23   return diameter, average_distance
```

\leftarrow logical sum operation
(the most time-consuming part)

We have developed Serial, Multi-threads
GPU, Multi-GPUs versions

Experiment environment

Cygnus system in Univ. of Tsukuba

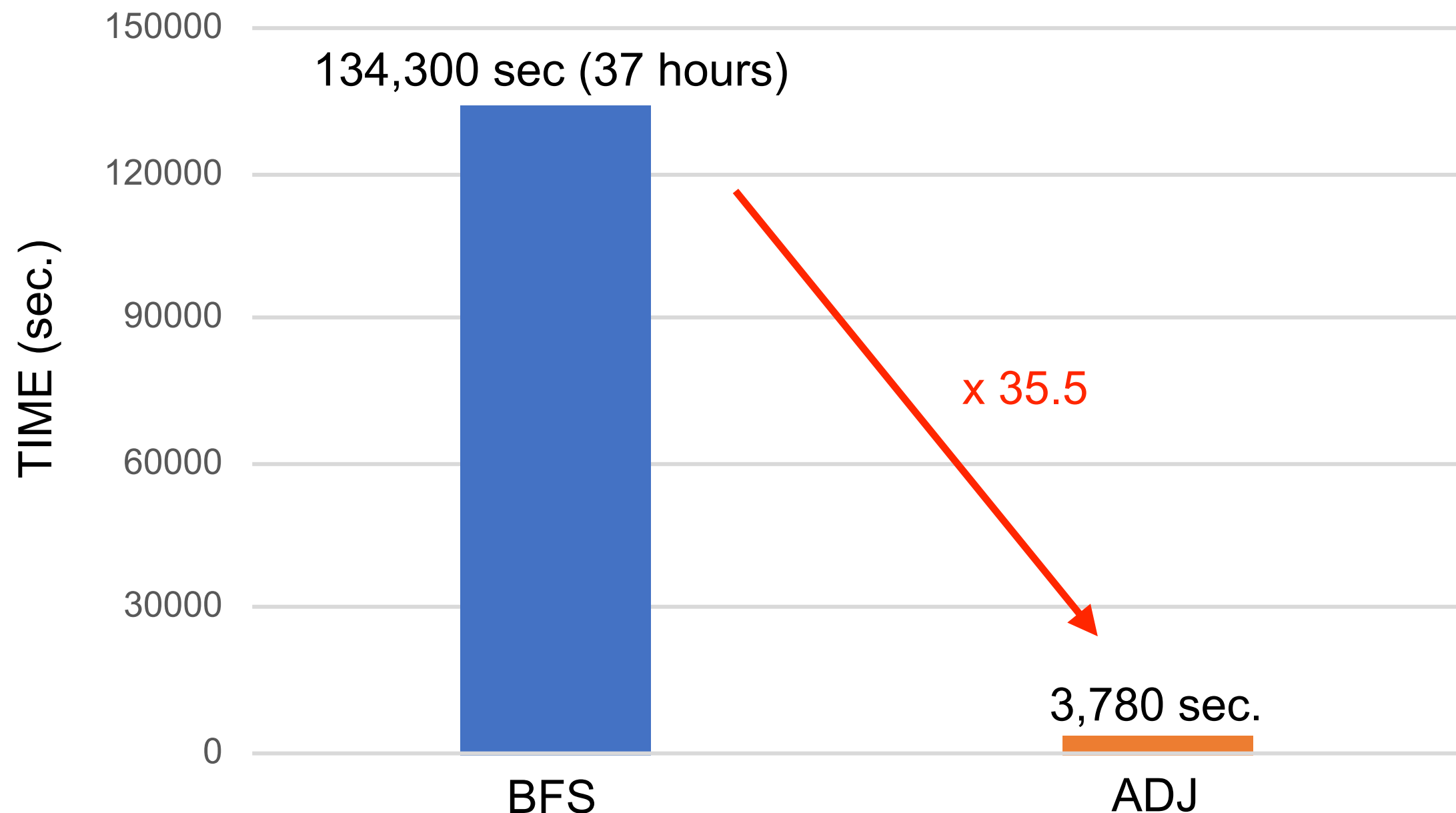
CPU	Intel Xeon Gold 6126 (12Cores, 2.6GHz) \times 2
Memory	DDR4 (128GB/s \times 2, 192GB)
GPU	NVIDIA Tesla V100 (900GB/s, 32GB) \times 4
Network	InfiniBand HDR100 (12.5GB/s) \times 4
Software	intel/19.0.3, mvapich/2.3.1, cuda/10.1



- The Cygnus system is provided by Interdisciplinary Computational Science Program in the Center for Computational Sciences, University of Tsukuba
- Computing resources such as Cygnus and Oakforest-PACS can be used **for free**
- This entry is held around mid-December

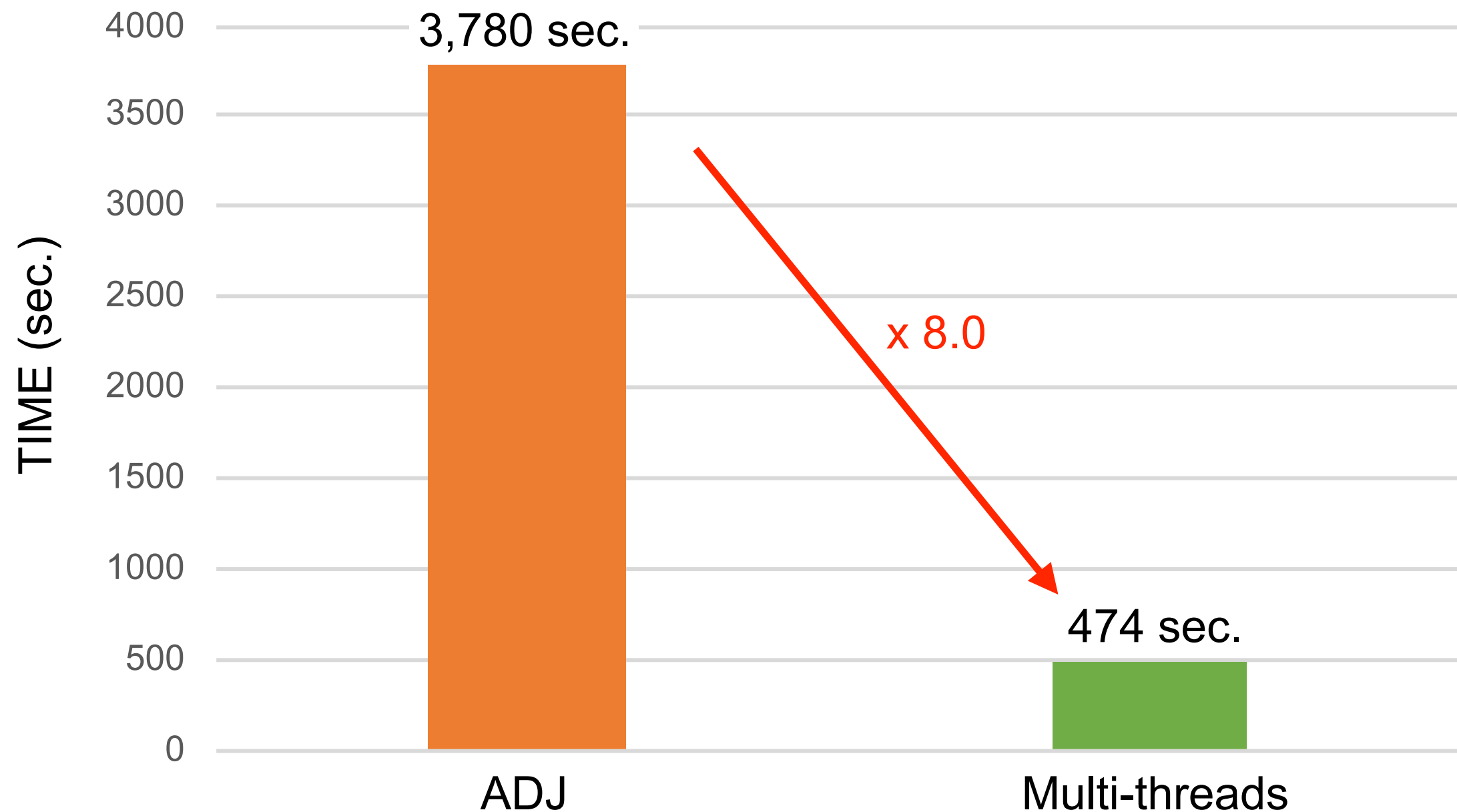
Result (Change from BFS to ADJ)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz



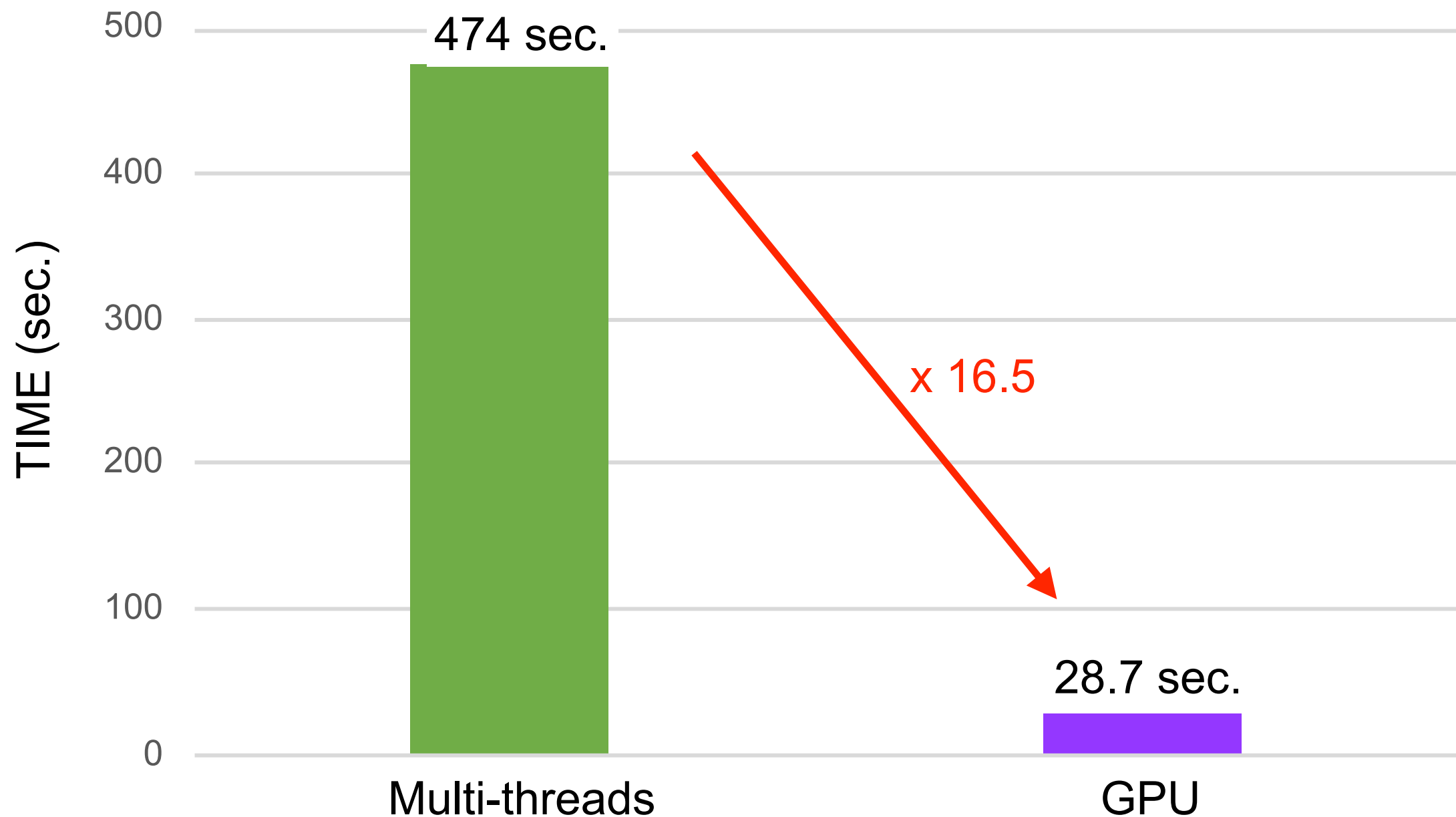
Result (Multi-threads)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores)



Result (GPU)

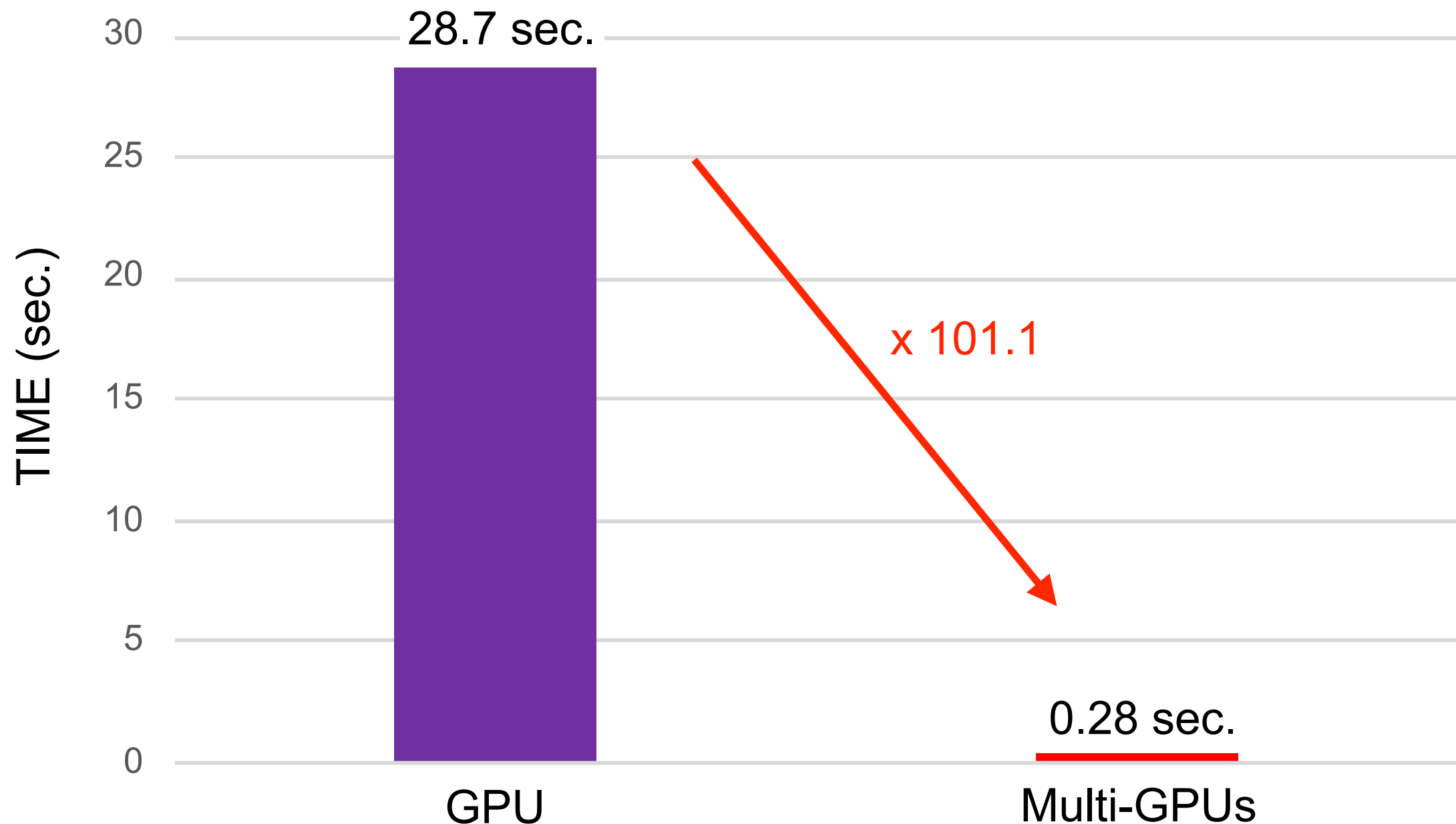
Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100



Result (Multi-GPUs)

Speed to calculate APSP for graph with (1M, 32)

Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100 x 128

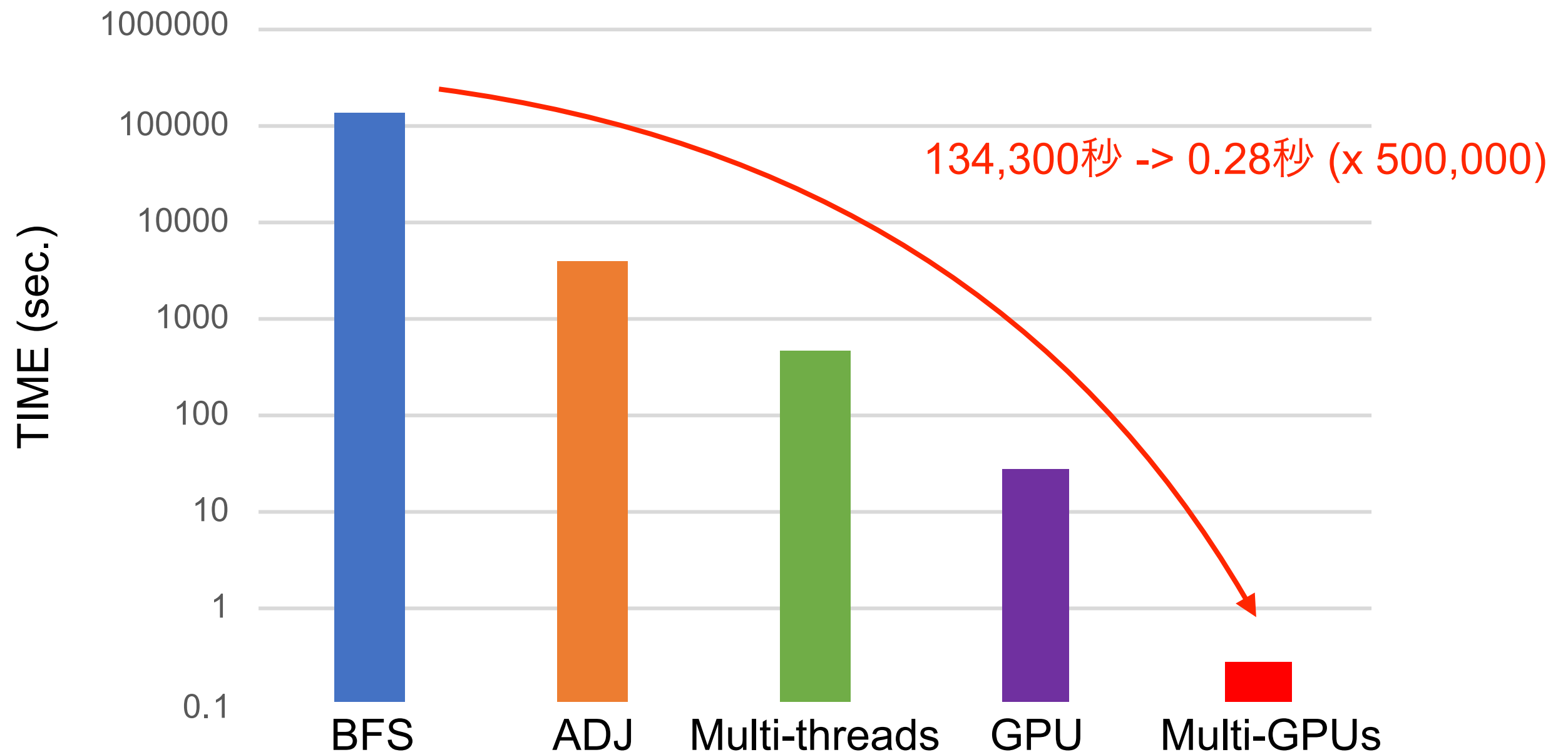


Summary

Of course, when using graph symmetry, the calculation time can be reduce more greatly.

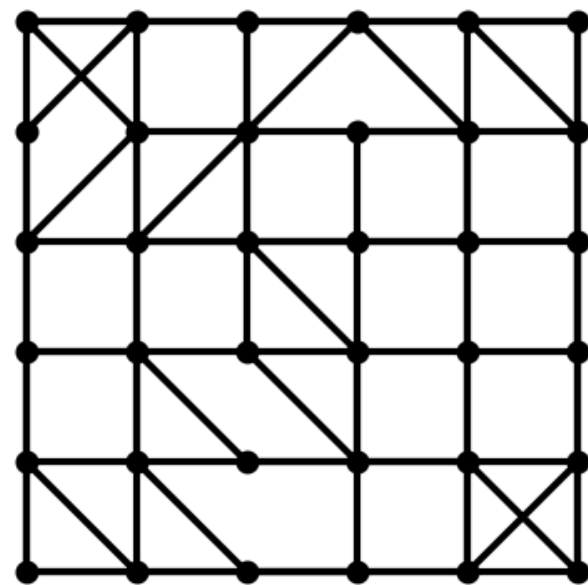
Speed to calculate APSP for graph with (1M, 32)

Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100 x 128

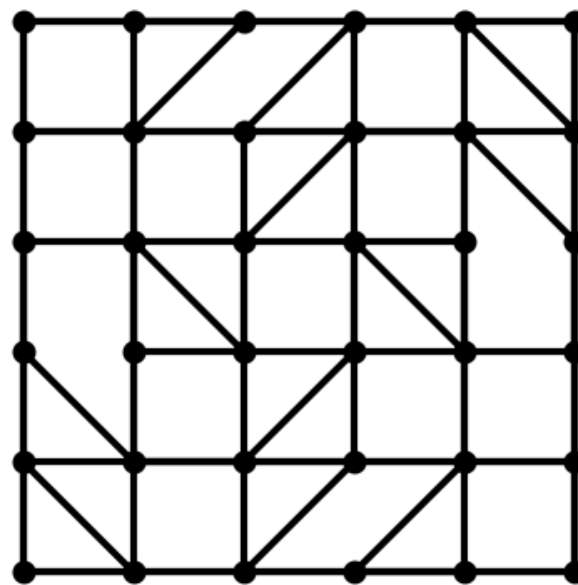


Graph symmetry

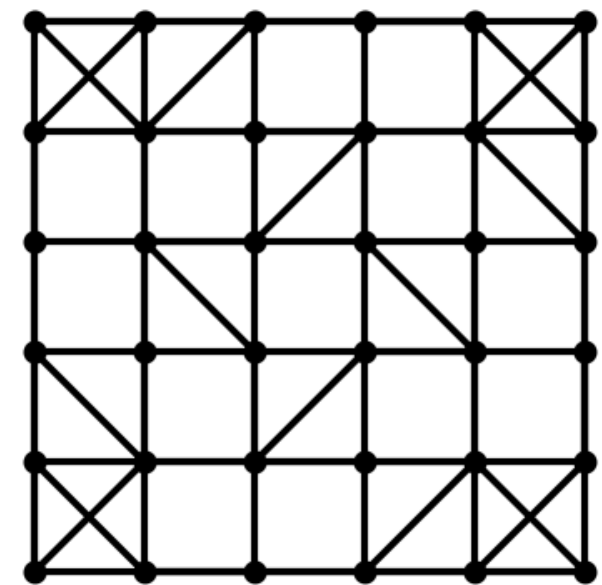
- The algorithm is inherited from mine for general graph last year [1-3]
- Examples of the graph symmetry with $(W, H, D, R) = (6, 6, 4, 2)$



g=1 (normal graph)



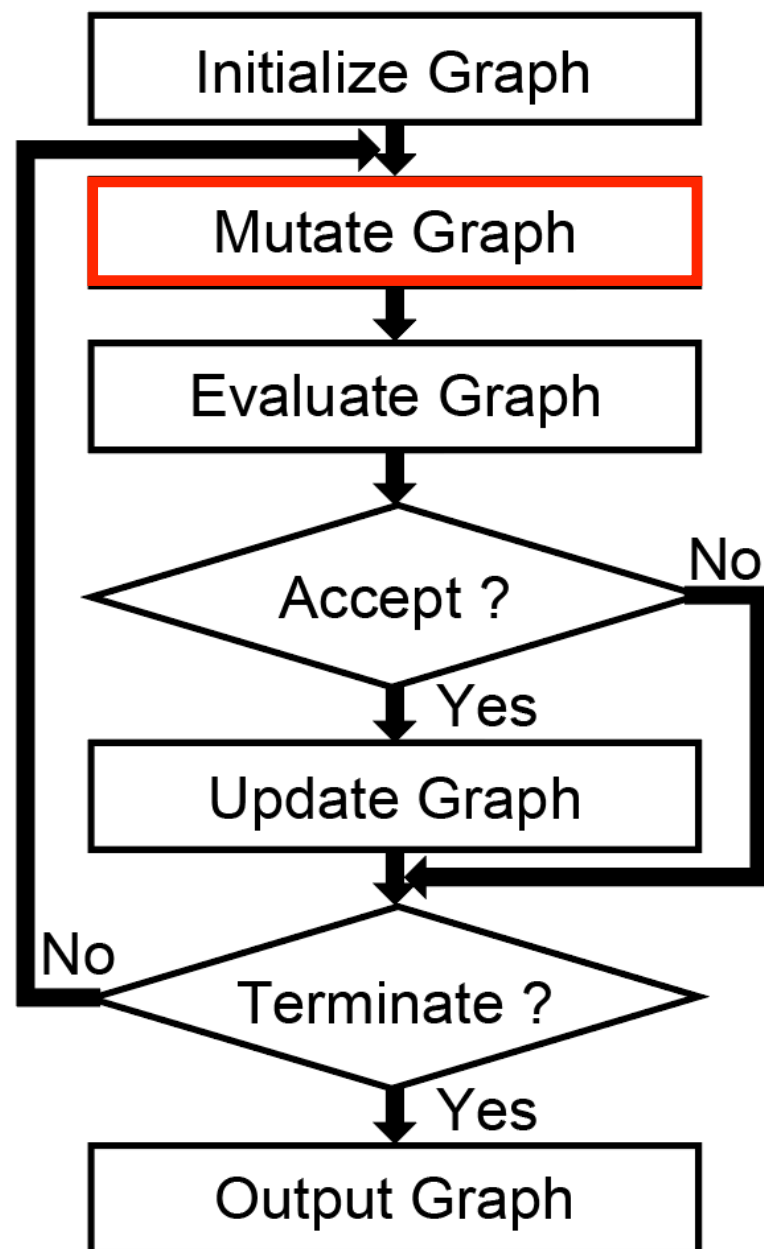
g=2



g=4

- The variable **g** is the number of groups (g must be 1 or 2 or 4)
- If a graph is rotated by $360/g$ degrees, the connection relationship between the vertex and edge becomes the original one

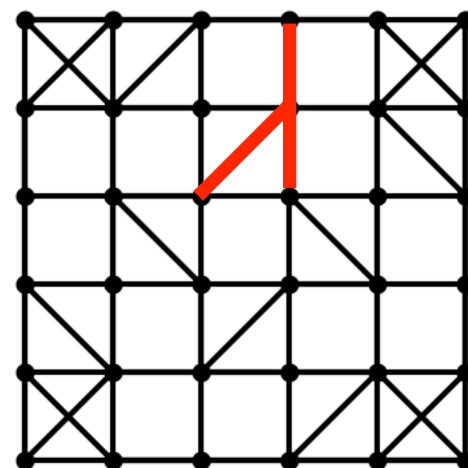
Edge exchange based on 2-opt



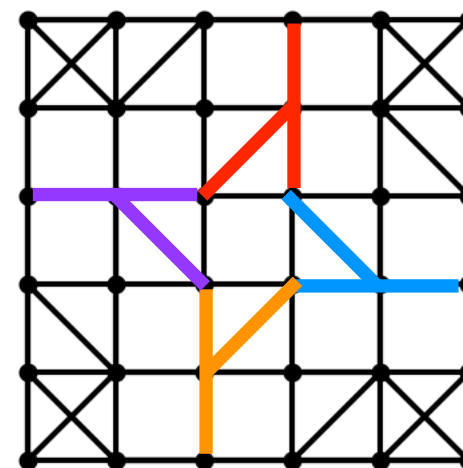
Perform 2-opt method while maintaining symmetry

In case of $g=4$

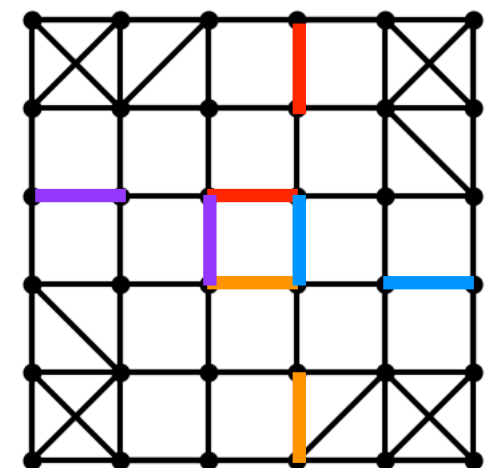
(1)



(2)

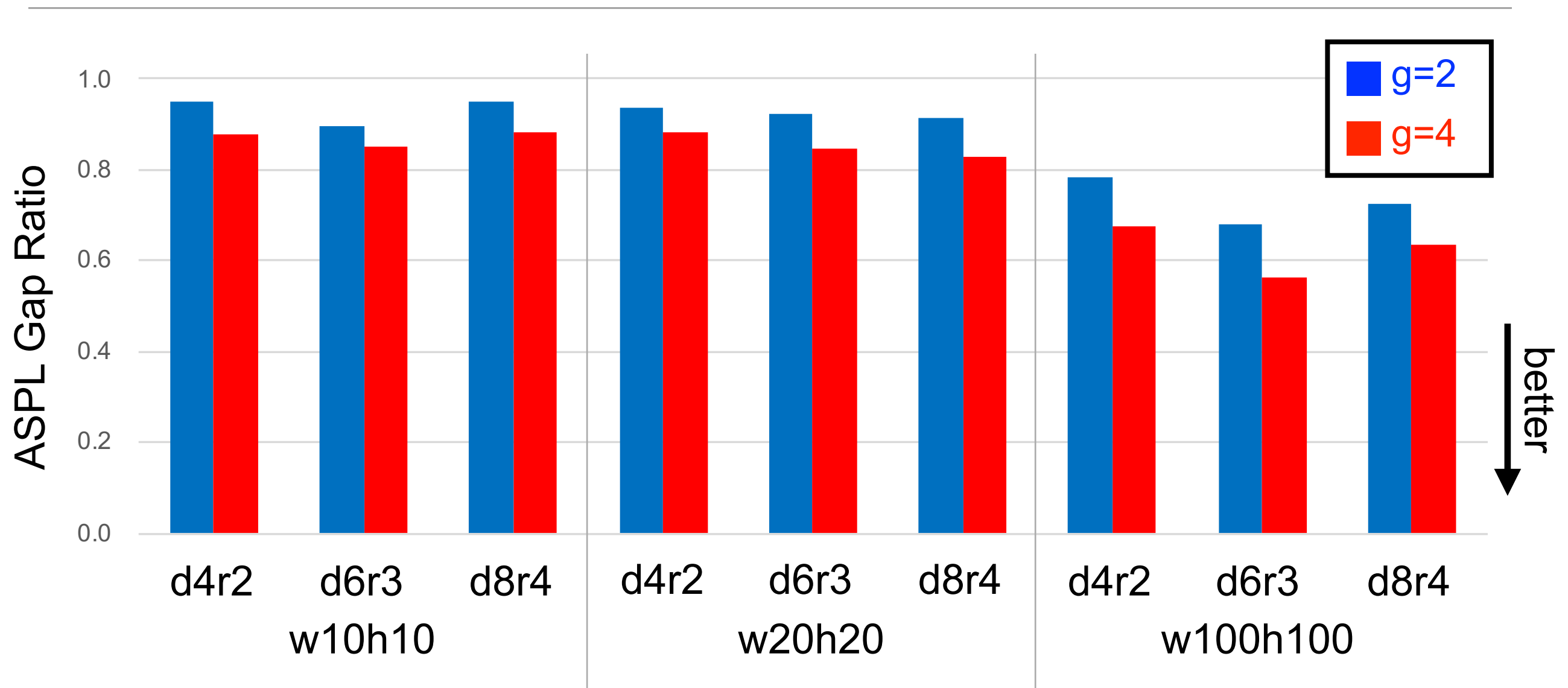


(3)



- (1) Randomly select two edges from all the edges
- (2) Select edges symmetrically related to (1)
- (3) Apply the 2-opt method to above edges each other

Results



- The vertical axis is the ASPL Gap Ratio when the result of $g=1$ is 1.0
- The larger the value of g is, the smaller the ASPL Gap is
 - Larger problems tend to have larger performance differences
- **The results show that graph symmetry is also useful in grid graphs**