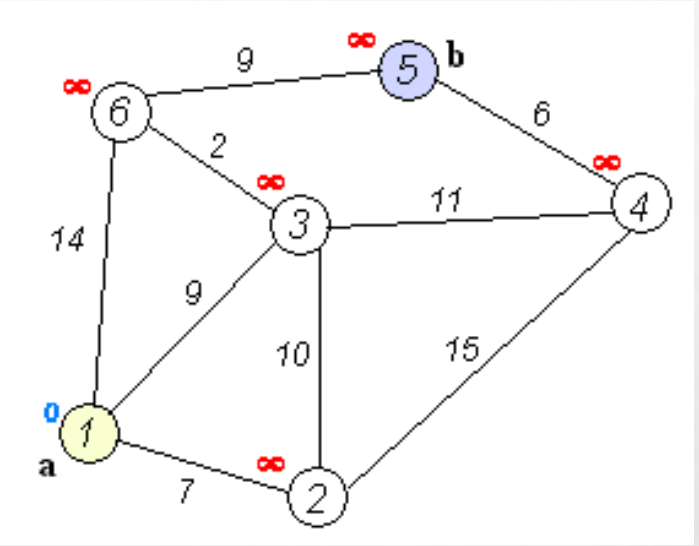
PARALLEL APPROACH TO DIJKSTRA'S ALGORITHM

LIBING WU

PROFESSOR: RUSS MILLER

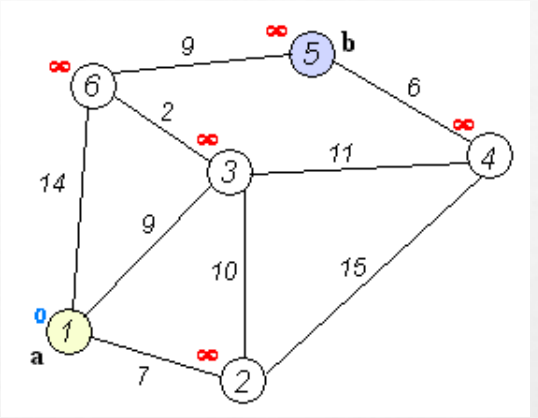
SPRING 2017, CSE 633, UNIVERSITY AT BUFFALO

DEFINITION



- Dijkstra's algorithm to find the shortest path between a and b. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited when done with neighbors.

SEQUENTIAL VERSION



- Dijkstra's algorithm to find the shortest path between a and b. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited when done with neighbors.
- Sequential running time of…
- Dijkstra's algorithm with list — $O(V^2)$
- Dijkstra's algorithm with modified binary heap — $O((E + V) \log V)$
- Dijkstra's algorithm with Fibonacci heap — $O(E + V \log V)$

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:              // Initialization
6          dist[v] ← INFINITY                   // Unknown distance from source to v
7          prev[v] ← UNDEFINED                  // Previous node in optimal path from source
8          add v to Q                           // All nodes initially in Q (unvisited nodes)
9
10     dist[source] ← 0                         // Distance from source to source
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]     // Node with the least distance will be selected first
14         remove u from Q
15
16         for each neighbor v of u:            // where v is still in Q.
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]:                // A shorter path to v has been found
19                 dist[v] ← alt
20                 prev[v] ← u
21
22     return dist[], prev[]
```

DRAWBACK

- Sequential running time of…
- Dijkstra's algorithm with list $O(V^2)$
- Dijkstra's algorithm with modified binary heap $O((E + V) \log V)$
- Dijkstra's algorithm with Fibonacci heap $O(E + V \log V)$
- If nodes are too many (scale is too large), it will become very inefficient.

A PARALLEL APPROACH

- Divide nodes into clusters
- Each cluster calculate local node closest to the starting node
- Identify global closest node using parallel prefix
- Broadcast to all cluster
- Perform list update for each cluster

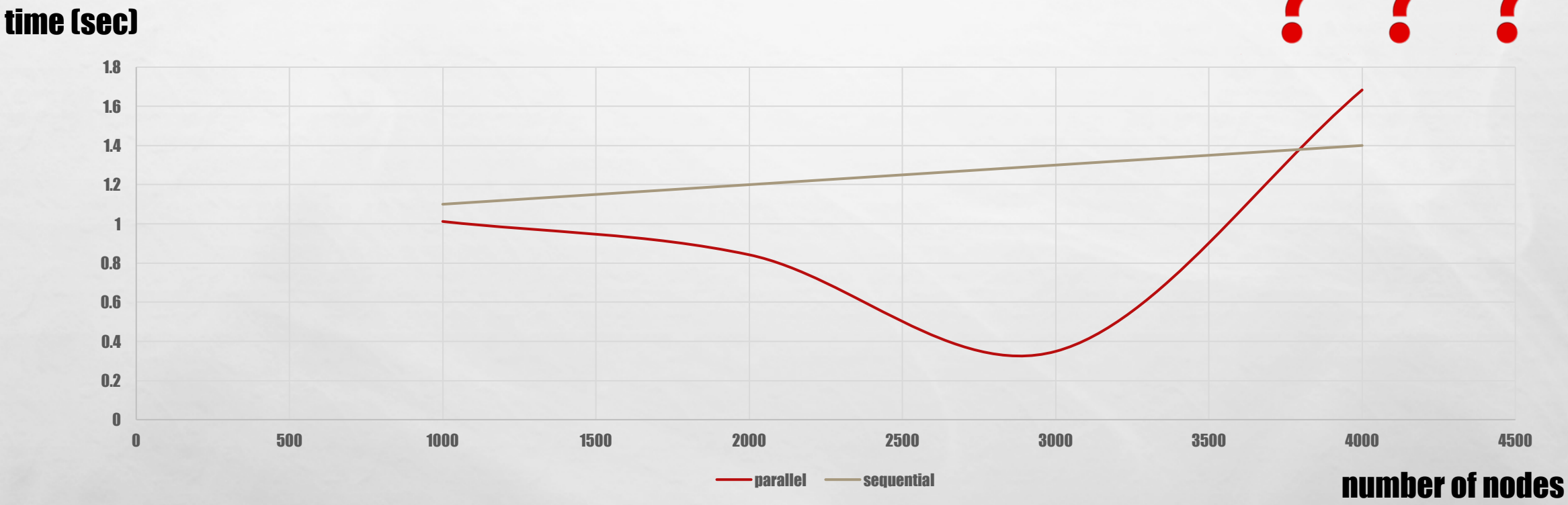
```
VOID DIJKSTRA(INT *LOCAL_MAT, INT *LOCAL_DIST, INT *LOCAL_PRED, INT N, INT LOCAL_N, INT MY_RANK, MPI_COMM COMM)
    // INITIAL LOCAL MINIMUM DISTANCE
    //SET NODE 0 WITH VISITED MARK
    // FIND A LOCAL NODE WHICH IS NOT VISITED WITH MINIMUM DISTANCE
    /*
    * SEND:
    * MIN_DIST_LOC[0]: THE MINIMUM DISTANCE TO A LOCAL NODE
    * MIN_DIST_LOC[1]: THE GLOBAL POSITION OF THE LOCAL NODE
    * RECEIVE:
    * MIN_DIST[0]: THE MINIMUM DISTANCE TO A NODE
    * MIN_DIST[1]: THE GLOBAL POSITION OF THE NODE
    */
    //COMBINES VALUES FROM ALL PROCESSES AND DISTRIBUTES THE RESULT BACK TO ALL PROCESSES
    MPI_ALLREDUCE(MIN_DIST_LOC, MIN_DIST, 1, MPI_2INT, MPI_MINLOC, COMM);
    //CALCULATE NEW MINIMUM DISTANCE
```

RUNNING TIME

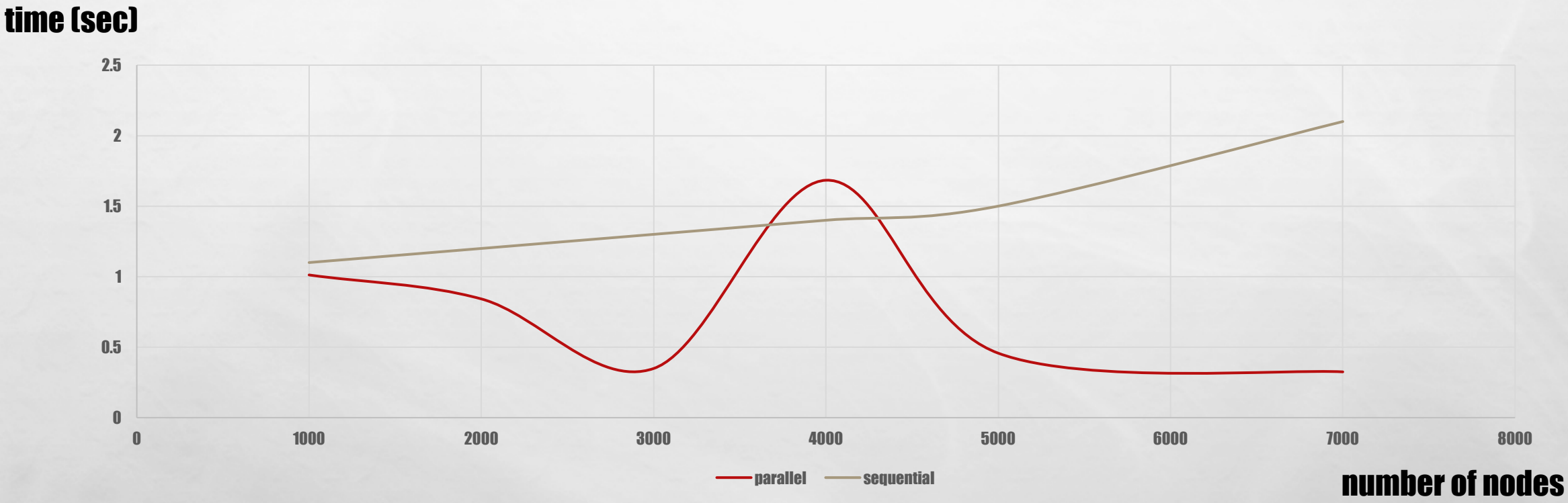
- P = number of processor
- During each iteration, update value of O(V) nodes with P processor -> O(V/P)
- During each iteration, find global closest node -> O(log P)
- O(V) iteration to finish routing table
- Running time: $O (V^2/P + V \log P)$
- More efficient to handle large scale nodes.

SEQUENTIAL VS. PARALLEL

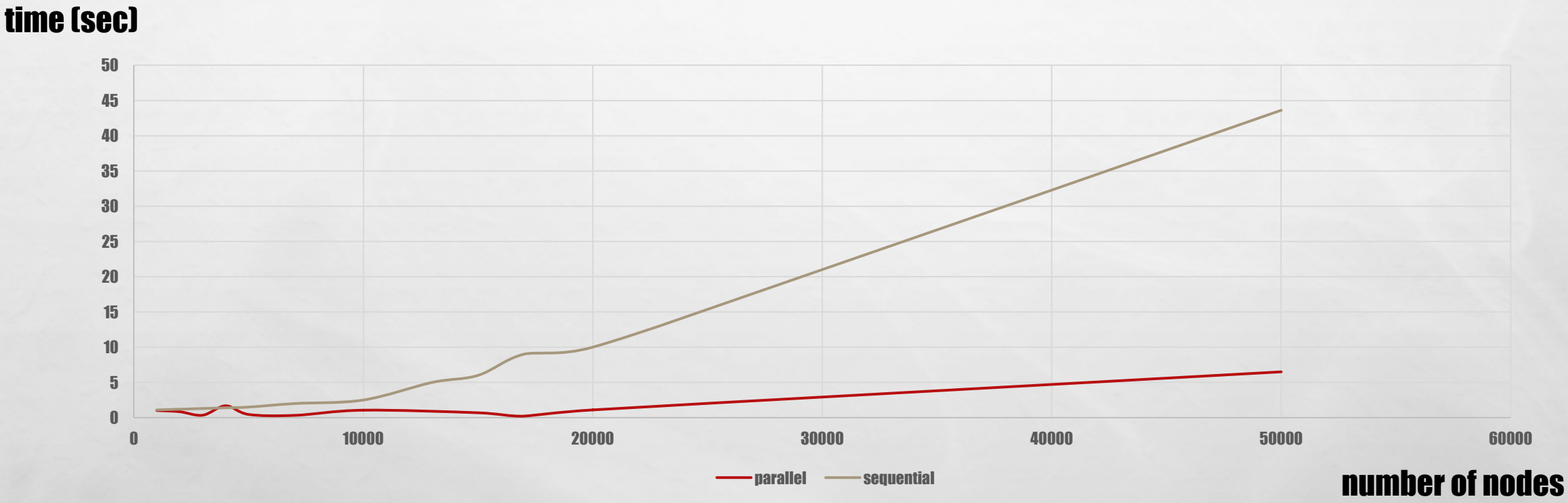
? ? ?



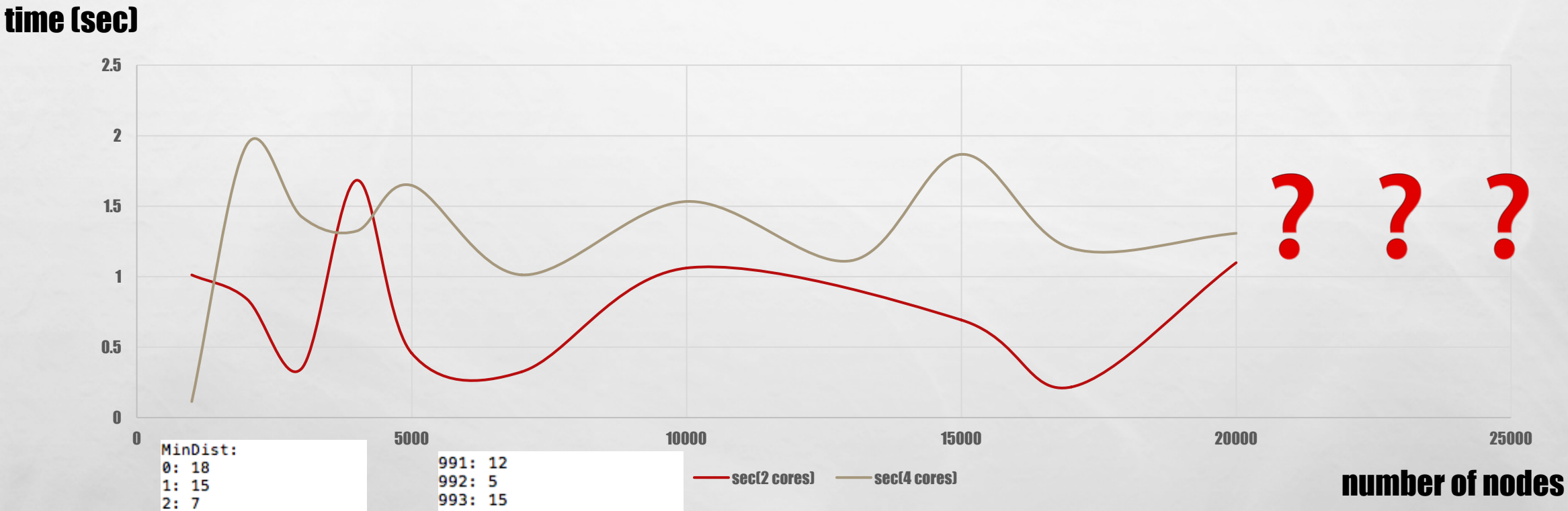
SEQUENTIAL VS. PARALLEL (LARGER SCALE...)



SEQUENTIAL VS. PARALLEL (LARGER SCALE...)



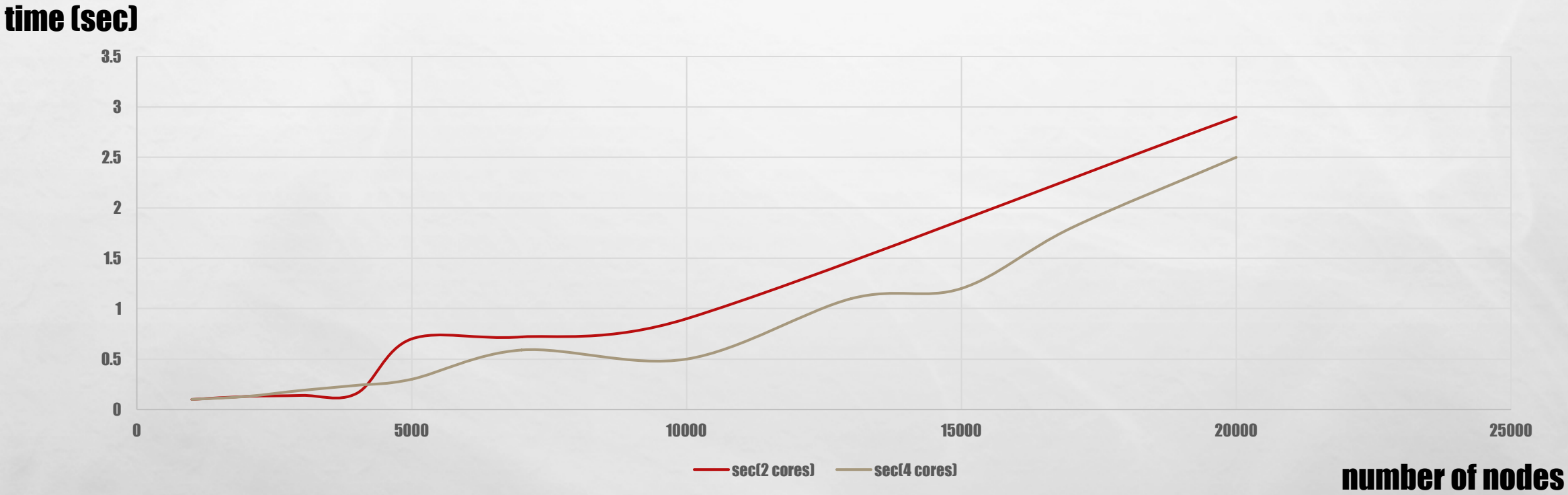
2 PROCESSORS VS. 4 PROCESSORS



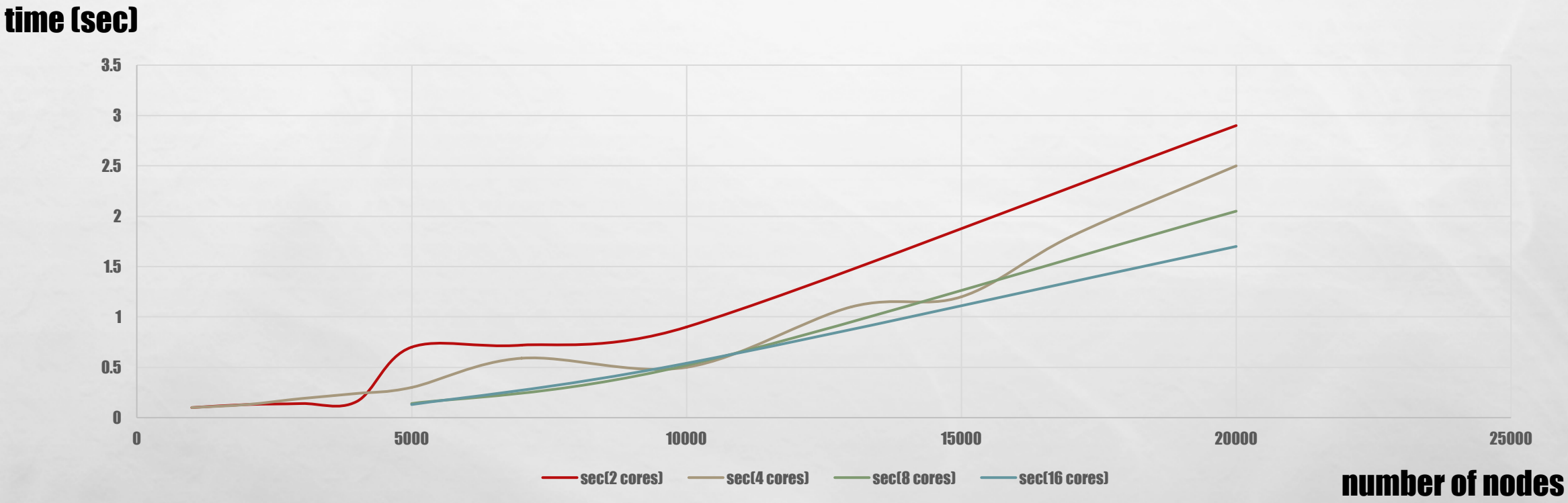

```
MinDist:
0: 18
1: 15
2: 7
3: 3
4: 3
5: 2
6: 12
7: 6
8: 16
9: 0
10: 2
```

```
991: 12
992: 5
993: 15
994: 0
995: 7
996: 11
997: 10
998: 3
999: 8
Time cost: 1012027 usec
```

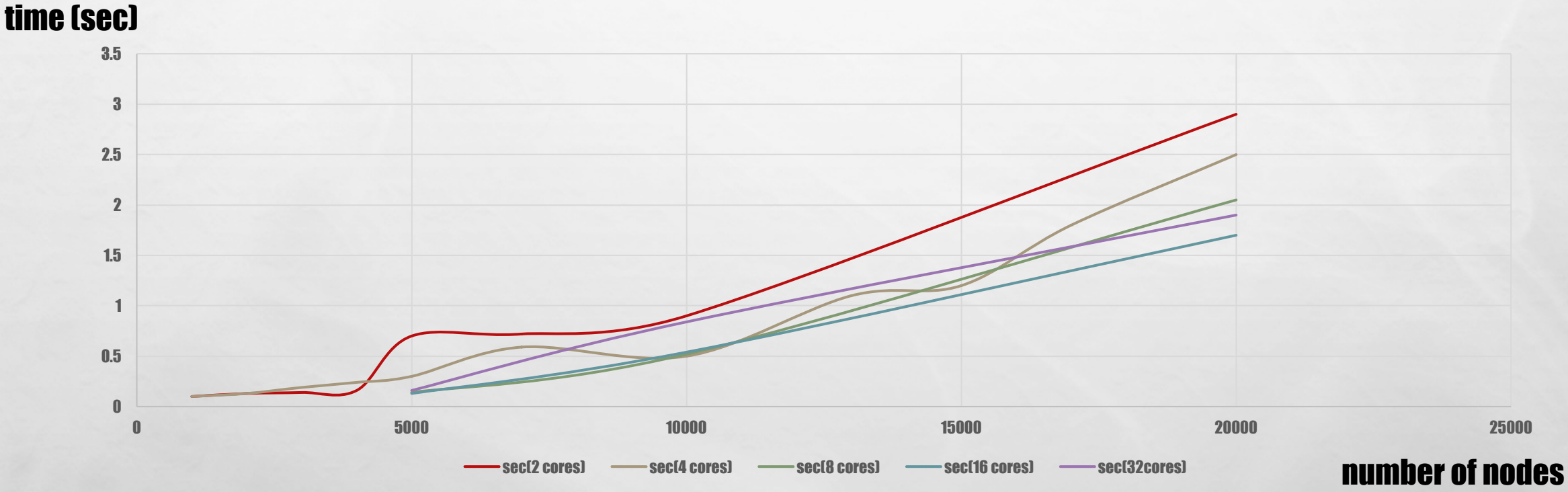
2 PROCESSORS VS. 4 PROCESSORS (FIXED BY INCREASING WEIGHTS ON EDGES)



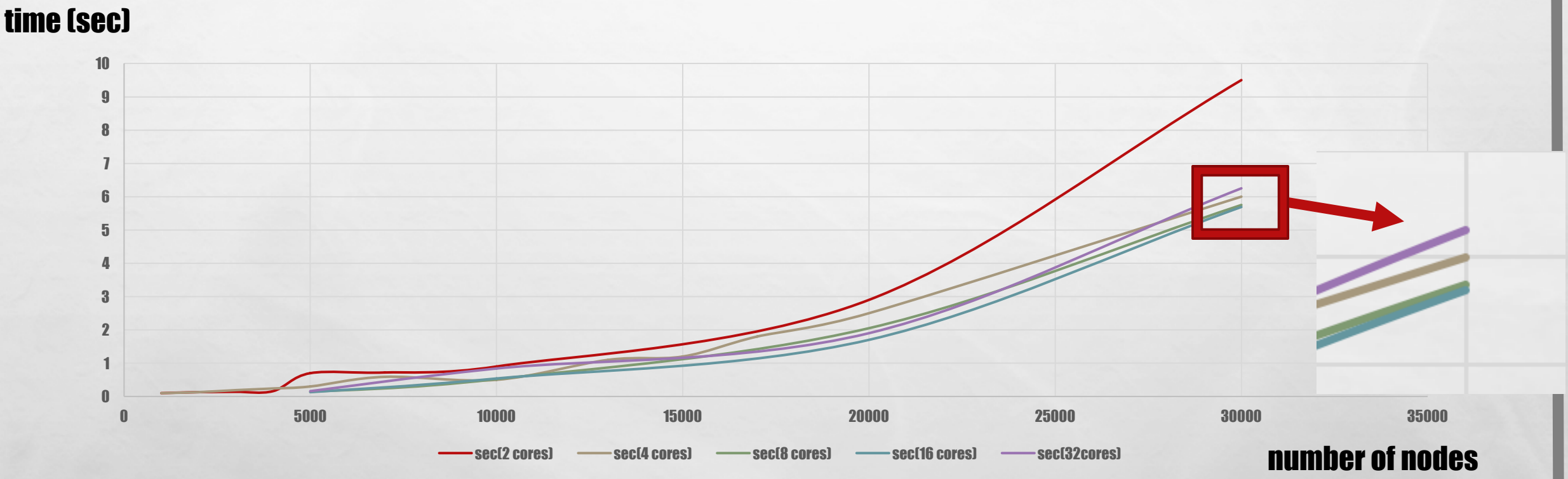
NUMBER OF PROCESSORS (2, 4, 8, 16)



SCENARIO OF 32 PROCESSORS... (NOT PERFORM BETTER THAN 16 PROCESSORS)

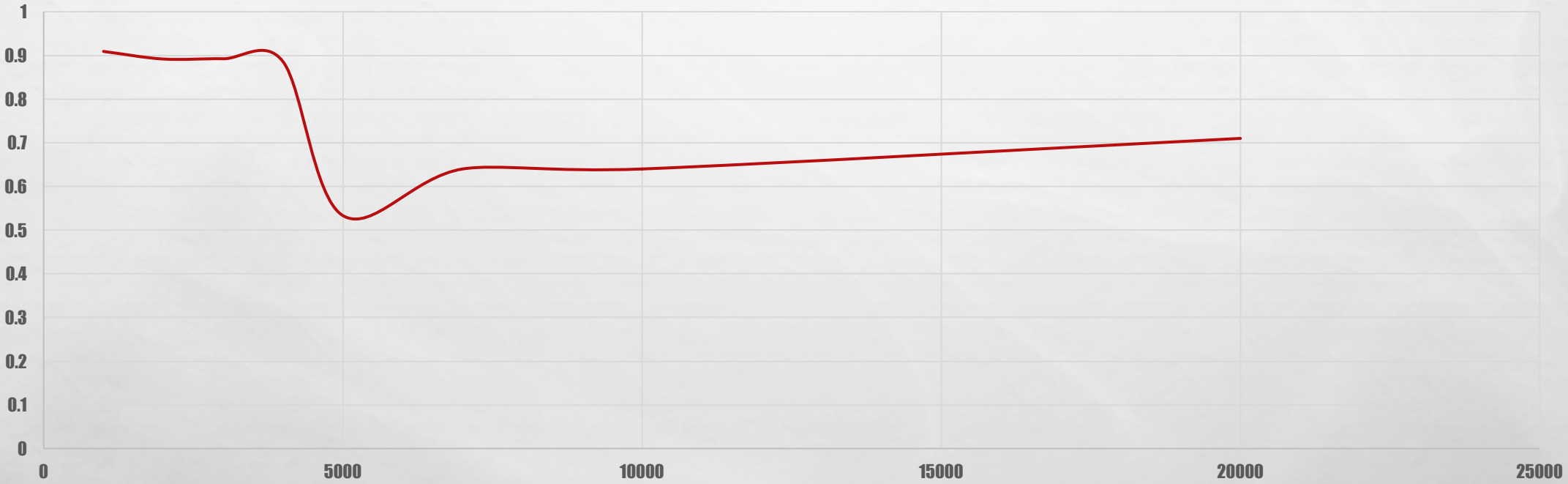


PERFORM BEST WITH 16 PROCESSORS



SPEED-UP

percentage of parallel faster than sequential



THINKING

- LIMITATION: IN DIJKSTRA'S ALGORITHM, ONLY THE INNER LOOP CAN BE IMPLEMENTED WITH PARALLEL ALGORITHM.
- FINDING: WHEN NUMBER OF PROCESSORS IS INCREASING, THE EFFICIENCY OF PARALLEL ALGORITHM DROPS, COST OF COMMUNICATION INCREASES, SO THAT IT SLOWS DOWN THE WHOLE PROGRAM'S SPEED.
- FURTHER: MORE NODES… FORMULA TO CALCULATE BEST AMOUNT OF PROCESSORS

THANKS