Distributed All Pairs Shortest Path Algorithm

Written by Clayton Herbst (22245091)

Abstract

The all pairs shortest path problem is a computationally intensive task that seeks to find the shortest path between every pair of vertices in a given graph (Wolfram 2019). There exist many adaptations to the problem that vary in computational difficulty depending on certain graphical properties such as the existence of edge weights, negative cycles and directed or undirected vertex edges. In this report we consider the performance achieved when computing the all pairs shortest path on a weighted-directed graph that contains no negative-cycles. Distributed computing techniques have been implemented to further increase the efficiency of available computing power thus, achieve further speed-up in the computation of the all pairs shortest path graph.

Floyd-Warshall All-Pairs Shortest Path Algorithm

The Floyd-Warshall algorithm is used to find the shortest paths with positive or negative edge-weights between any two vertices. The algorithm achieved this by comparing all possible paths through the graph between every pair of vertex (Wikipedia 2019). The time complexity of the Floyd-Warshall algorithm is thus $O(V^3)$, where V is the number of vertices in the given graph. The pseudo code for Floyd-Warshall's algorithm can be seen in *figure 1*.

```
1   # Floyd-Warshall algorithm given the adjacency matrix dist[][]
2   procedure apsp( dist[][] )
3     for k from 1 to |V|
4       for i from 1 to |V|
5         for j from 1 to |V|
6           if dist[i][j] > dist[i][k] + dist[k][j]
7             dist[i][j] ← dist[i][k] + dist[k][j]
8           end if
9   end procedure
```

Figure 1: Floyd-Warshall algorithm pseudocode

Distributed All Pairs Shortest Path Approach

Floyd-Warshall's algorithm can be distributed by partitioning all edge weights amongst available nodes and communicating computed results to the *root* node. Such an approach would partition the inner most loop of Floyd-Warshall's algorithm however will require the synchronisation of all node values after every iteration of k. This is due to the dependencies that exist amongst graph vertices in the Floyd-Warshall algorithm when minimising the existing path length with the new path length. For instance, to compute the shortest path between vertex i and j, the edge weights between vertices i and k as well as k and j must be known. Equation *(i)* has been extracted from the innermost loop of the Floyd-Warshall algorithm. This is the computational task that is distributed. The dependency is evident in the equation due to the retrieval of the path edge weights other than the path, i to j.

$$(i) : \ dist[i][j] \ = \ min(\ dist[i][j], \ dist[i][k] \ + \ dist[k][j] \ )$$

In order to synchronise computed values between nodes, values will need to be passed to all nodes in the communication group over the network. Even with high bandwidth availability and close node-proximities the network transmission is drastically slower than local system memory. This characteristic will greatly impact on the performance of the distributed algorithm and will result in bottlenecked performance. Thus in order to optimise the performance of the distributed algorithm the number of messages passed between nodes in a communication group need to be minimised.

This can be achieved by making greater use of the data access patterns of the Floyd-Warshall algorithm. Instead of partitioning every vertex, nodes within the distributed system are allocated blocks in a row-wise and/or column-wise fashion resulting in the respective *owner* of a block maintaining the correct state of the shortest paths after every *k-th* iteration. *Figure 2* illustrates how the blocks are distributed amongst the available processes.

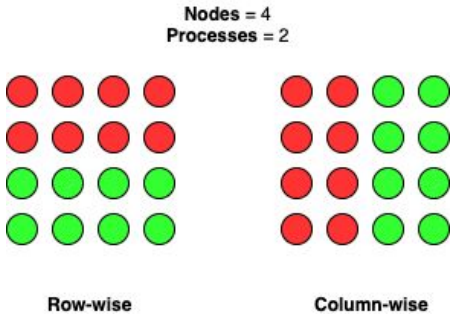

Figure 2: Row-wise vs column-wise block distribution

The choice between row-wise and column-wise block distribution does not have a significant impact on the algorithms performance. Referring to *equation (i),* the row-wise distribution simply results in the *dist[i][k]* being known or *'owned'* by the executing process and the *dist[k][j]* component needing to be *received* from other nodes in the communication group. Column-wise distribution simply alters the execution order since the nodes are responsible for different blocks (i.e *dist[k][j]* and receive *dist[i][k])*. I have chosen to implement row-wise block distribution to facilitate the natural data access patterns of matrices as well accommodating for simpler printing of matrices at any point in time.

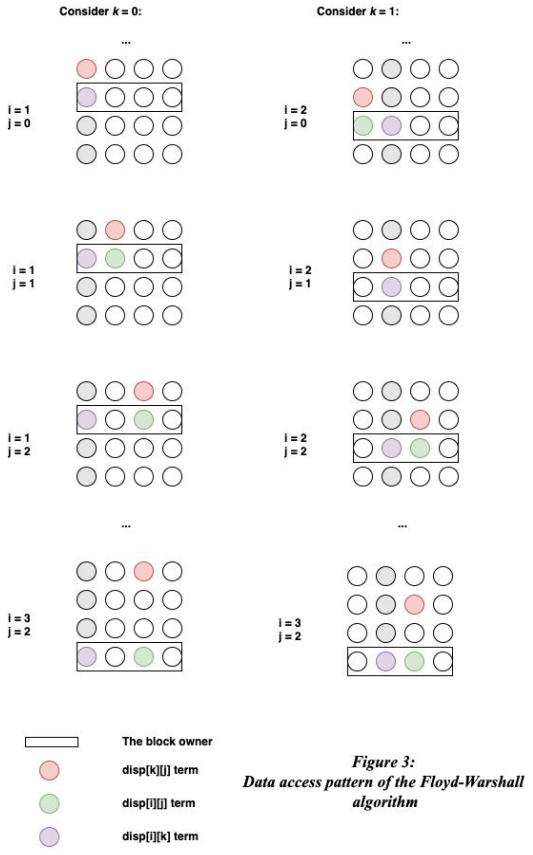

Figure 3: Data access pattern of the Floyd-Warshall algorithm

Despite decreasing the number of messages needing to be transmitted between distributed nodes an added benefit to the above approach is the loosening of a nodes requirement to start with the entire adjacency matrix of the graph. Rather, distributed nodes only need to store the edge weights of the block they are responsible for. Thus the space complexity of the distributed algorithm at each node is reduced to $O(\frac{V^2}{P})$, where V is the number of vertices and P is the number of available processes. This can be achieved due to the data access patterns of the Floyd-Warshall algorithm, which *figure 3* attempts to illustrate.

For a particular node in the communication group, if the local memory contains the up to date shortest paths for state *k,* then the node only needs to have one additional item of information to satisfy *equation (i).* This is illustrated in *figure 3* by visualising the violet and green circles remaining in the transparent rectangular box for changes in *i* and *j*. The red circle however remains at all times outside this transparent box in a row-wise fashion (the *k-th* row). The rectangular box represents the ownership block of a particular node and the circles represent terms in *equation (i) line 6* . *Figure 3* thus demonstrates that in order to satisfy *equation (i)* the *k-th* row must be broadcasted amongst all nodes in the

communication group. The effect of this property on the relationships between nodes is a changing *'master'* or *'driving'* node for each *k-th* iteration. For each *i-th* iteration only the process that is the *'owner'* of that block *(*the rectangle in *figure 3*) executes that loop of *j* values and thus updates its local copy of the shortest path after state *k.* The node that broadcast the *k-th* row (the red circles in *figure 3)* is only ever the owner of that row and thus guarantees the accuracy of the shortest paths broadcasted. Following the execution of the final *k-th* loop each distributed node will contain local copies of the calculated shortest paths, hence a final gathering process needs to be executed in order to compile the resulting all pairs shortest path graph.

The software diagram for the distributed Floyd-Warshall algorithm is provided in *figure 4.* The message parsing interface (mpi) that I have elected to use is OpenMPI; an open source message parsing interface maintained by academics (OpenMPI 2019). I have made use of the OpenMPI library to read the adjacency matrix from file in a distributed fashion, broadcasted the *k-th row* within the distributed Floyd-Warshall algorithm and finally gathered all shortest path metadata from distributed nodes and combine this result in the head node using the MPI_Gatherv function call.

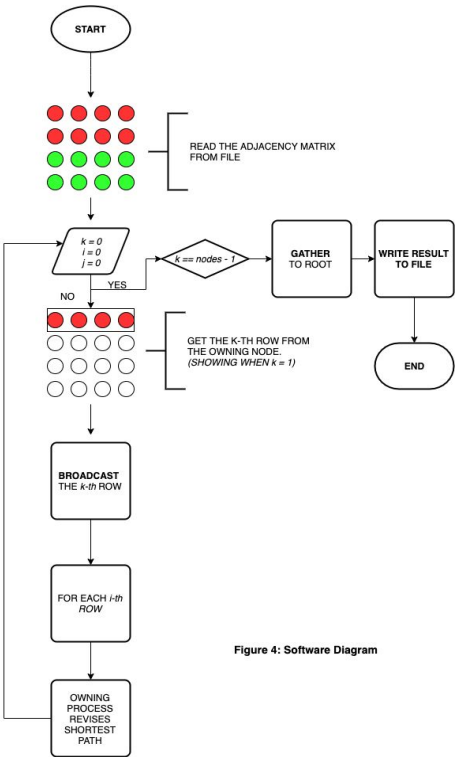

Figure 4: Software Diagram

The time complexity of the distributed algorithm can be calculated by considering the relative work performed by distributed nodes at the outer, middle and inner most *for-loops.* The outer and the innermost loops are executed V times by each node; a complexity of $O(V)$ respectively, where V is the number of vertices in the given graph. The middle loop however is distributed amongst all available processes resulting in a time complexity of $O(\frac{V}{P})$, where P is the number of processes available in the communication group. As a result the total time complexity of the distributed algorithm is $O(\alpha \cdot V^2 \cdot \frac{V}{P})$ where the scalar coefficient a is used to represent the execution time impact of message transmission between distributed nodes.

As seen in *figure 5,* significant speedup was achieved by the distributed algorithm in comparison to the synchronous Floyd-Warshall algorithm. The results depicted were achieved using four distributed nodes and varied graph sizes. For comparison the single process execution of the Floyd-Warshall algorithm is provided alongside the distributed approach's results.

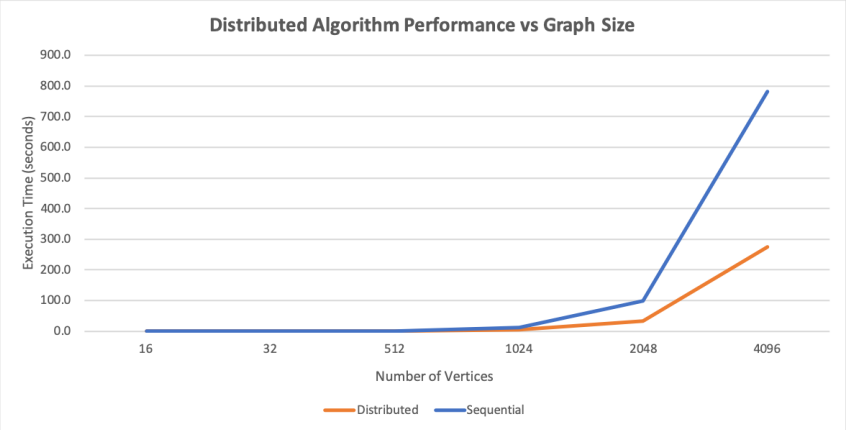

Figure 5: Distributed algorithm performance across varied graph sizes

The relationship between the execution time and the number of graph vertices remains polynomial confirming time complexity calculations made earlier which expected the execution time to increase in a polynomial fashion. The distributed graph grows at a significantly slower rate than the sequential algorithm. This trend confirms previous calculations where a factor of $\frac{a}{p}$ was the difference between the two algorithms, where a is the execution time impact of the message transmission and P is the number of processes available to the distributed communication group.

Variances in the results captured for varied number of distributed nodes are exhibited in *figure 6* where particular gathered data points seem to deviate from the expected trend-line. The line-of-best fit is observed to follow the expected inversely proportional relationship between the execution time and the number of processes available to the algorithm ($t \propto \frac{1}{P}$). Deviations from this trend-line can be attributed to run-time variances. These variances seen in the data set captured may be the result of networking delays/interferences between distributed nodes and/or kernel process suspension at particular nodes due to the queuing of programs. The impact of these factors may be reduced in further cases by ensuring nodes remain relatively idle and focused on completing the task at hand. This will also ensure the bandwidth availability between nodes remains high.

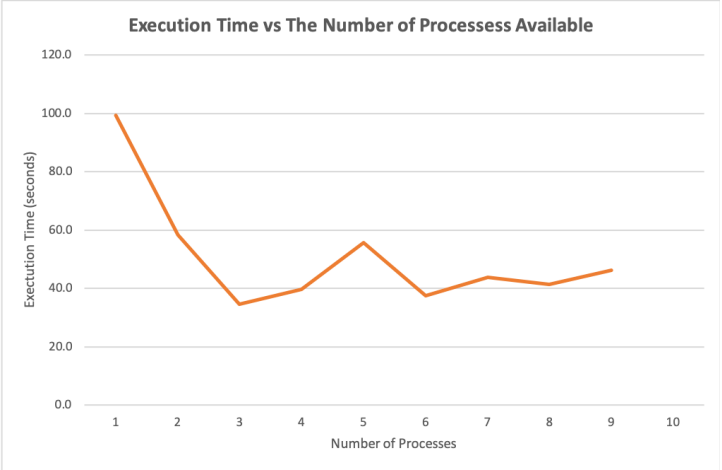

Figure 6: Inversely proportional relationship between execution time and number of processes available

Appendix A

Execution time of the distributed algorithm relative to graph size:

Processes	Vertices	Distributed Execution Time (seconds)	Single Process Execution Time (seconds)
4	16	0.00006	0.00006
4	32	0.00018	0.00041
4	512	0.46435	1.49141
4	1024	4.20341	11.72472
4	2048	33.87517	98.03816
4	4096	275.40570	781.65726

Appendix B

Execution of the distributed algorithm relative to the number of processes available:

Processes	Vertices	Distributed Execution Time (seconds)
1	2048	99.23270
2	2048	58.33426
4	2048	34.65399
6	2048	39.78043
8	2048	55.79771
10	2048	37.52628
12	2048	43.88729811
14	2048	41.44043708
16	2048	46.34219384

References

Wikipedia 2019, Distributed Computing, 23 September 2019.Wikipedia Encyclopaedia. Available from: https://en.wikipedia.org/wiki/Distributed_computing. [Accessed 23 Oct 2019]

Wikipedia 2019, Graph Theory, 24 October 2019. Wikipedia Encyclopaedia. Available from: https://en.wikipedia.org/wiki/Graph_theory. [Accessed 23 Oct 2019].

Weisstein, Eric W, 17 September 2019. All-Pairs Shortest Path Problem. MathWorld - Wolfram Resource. Available from: http://mathworld.wolfram.com /All-PairsShortestPath.html. [Accessed 22 Oct 2019].

Dan Nagle, "MPI – The Complete Reference, Vol. 1, The MPI Core, 2nd ed., Scientific and Engineering Computation Series, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra," Scientific Programming, vol. 13, no. 1, pp. 57-63, 2005. https://doi.org/10.1155/2005/653765.

Department of Computer Science and Engineering (DEI), 6 November 2012. All-Pairs Shortesst Paths - Floyd's Algorithm. Instituto Superior T´ecnico. [Accessed 22 Oct 2019].