Single Source Shortest Path in Parallel

Team Nirnroot
Michael Adams and Howard Ross
Problem Intro

• Single Source Shortest Path Graph Traversal.
• Given graph with non-negative edge weights and a source node, find shortest path to all other nodes in graph.
• Increasingly important in today’s world
  o Social media, agency relations, WWW, traffic routing, etc.
• Our example: emergency dispatch for ambulance/police or quick pizza delivery
Paper 1

• Delta stepping parallel algorithm
• Vertices are placed in “bins” based on their distance
  o Define a delta value
• Relax “light edges” as we reach them. Create request set
  o Light edge = edge weight less than delta
• Relax “heavy edge” after all light edges are relaxed
  o Heavy edge = edge weight greater than delta
Paper 1 continued

- Vertices are “relocated”
  - May be placed back in same bin or later bin
- Done when all bins are empty
- Cray MTA-2, word level synchronization, no memory hierarchy
- ~ 75% efficiency, 40 cores
- More complex data structures to ensure quick insertion/removal.
Paper 2

• Implementing Parallel Shortest Path for Parallel Transportation Applications

• Researches the Traffic Equilibrium Problem
  o An example is what impact would building an interchange have on the 405 Freeway in Los Angeles at rush hour.

• Focus is on how to decompose a traffic grid and choice of algorithm to produce the shortest run time

• Iterative problem where each iteration is a slice of real time.

• Variables are changing road conditions, traffic density, time of day, updated position of vehicles.
Network Decomposition

- Grids studied were road networks in New York City, Chicago, and Waltham, MA
  - Between 4795 to 26347 nodes.
- Geometrically focused, such as NxN squares, and NxM or MxN rectangles.
  - Between 33 x 33 to 257X257 nodes.
- Networks were divided using Linear Decomposition, Equally Sized Squares, Horizontal Strips, Vertical Strips
- Each Processor is responsible for a piece of the decomposition.
Network Decomposition
Results

• Small Diameter resulted in fewer changes to the edge nodes which resulted in fewer iterations.
• Large number of edge nodes increased the amount of information each processor received per update.
• Small number of interfaces reduced the number of messages across the processors.
Algorithms

• Bellman-Ford-Moore using a label setting algorithm.
  o Nodes are stored in a list implemented as a balanced binary tree.
  o Nodes are removed which is a $O(\log(n))$ operation.

• Bellman-Ford-Moore label using a label correcting algorithm.
  o Nodes are stored in one queue.
  o Node removal or updating is $O(1)$

• Pallattino Algorithm using a label correcting algorithm.
  o Nodes are stored in two queues. First queue is for nodes that have been seen before. Otherwise the nodes are stored in the second queue.
  o Node removal or updating is $O(1)$
Algorithms Results

• Label Correcting Algorithms have a lower run time for Networks under 129x129 Nodes.
  o This is because the time to updated a node in the label correcting algorithm is $O(1)$ while it’s $O(\log(n))$ for label setting.

• For networks at or above 129 x 129 Nodes label setting algorithms performed better.
  o This is because the number of updates for label correcting is much higher than for label setting.
  o Label setting did 1110917 updates, label correcting did 2643164 updates.

• On a 257x257 graph
  o Label setting did 4423215 updates, label correcting did 18731866 updates.
Paper 3

- Multithreaded Asynchronous Graph Traversal for In-Memory and Semi-External Memory
- Focuses on an algorithm and the use of flash memory.
- Uses a hybrid of the Bellman-Ford label correcting algorithm and Dijkstra’s algorithm.
- Uses flash memory in place of hard disk drive reads and writes.
The Algorithm

• Similar to the Bellman Ford approach in that it uses label correction during the traversal.
• Similar to Dijkstra’s algorithm in that it traverses paths in a greedy manner.
• Graph is split among the processors.
• Achieved speedups of 15.3 versus the DIMACS-SSSP sequential algorithm.
The Hardware

- For the in memory baseline, they used a 16 core AMD Opteron with 256 GB of RAM.
- For flash memory, they used the same hardware but only 16GB RAM.
- Flash used for external memory was:
  - Fusion IO that can do ~ 200000 reads per second.
  - Intel that can do ~60000 reads per second.
  - Corsair that can do 30000 reads per second.
- Achieved a 5 times speedup on the FusionIO.
Generating a Graph

• Wanted to avoid I/O. Generate the graph randomly.
• Number of nodes, outgoing edges per node, edge weight range, random seed.
• For each node
  o Get random number of edges within bounds
  o Create a random subset of all nodes except self
  o For each edge
    • Get a random node from subset
    • Get a random weight within bounds.
    • Add to current nodes edge list
Generating a Graph

- If number of outgoing edges was same for all nodes, could be parallel.
- Each Node contains (int) id, (int) minimumDistance, and (Edge[]) edges.
- Each Edge contains (int) adjacentID, (int) weight
- Originally was a weight matrix – huge waste of memory
- Entire graph is stored in a Node[], where all nodes know their outgoing edges, id, and current distance value (initialized to Integer.MAX_VALUE)
Sequential Moore’s
Sequential Moore’s

- **INPUT:** Graph $g$, Source Node $n$
- Create parent array of size $\text{numNodes}$
- Set $n.\text{minimumDistance} = 0; \ text{parent}[n] = n.\text{id}$
- Add $n$ to queue of nodes to be examined
- **While** (node queue is not empty)
  - $\text{curNode} = \text{queue}.\text{removeFirst}$
  - For Edge $e$ in $\text{curNode}.\text{edges}$
    - $\text{newDist} = \text{curNode}.\text{minimumDistance} + e.\text{weight}$
    - If $\text{newDist} < \text{neighbor}.\text{minimumDistance}$
      - $\text{neighbor}.\text{minimumDistance} = \text{newDist}$
      - $\text{parent}[\text{neighbor}.\text{id}] = \text{curNode}.\text{id}$
      - $\text{queue}.\text{add}(\text{neighbor})$
Parallel Moore’s
Parallel Moore’s

- Design pattern is master worker
- GraphMessage class is used to create REQUESTTASK, PROCESSNODE, UPDATEDNODE, and STOP messages.
- An array of ConcurrentLinkedQueues provides communications between the Master and Workers.
- Master does all the reads and writes to the node [] and the parent []
- The only contention is in the workers accessing the master message queue.
Master Thread

- INPUT: Graph g, Source Node n
- Create parent array of size numNodes
- Set n.minimumDistance = 0; parent[n] = n.id;
- Add n to the queue.
- While there are nodes in the queue and workers are processing
  - When a REQUESTTask message comes in pull a node from the queue.
    - Create a PROCESSNODE message and sent it to the worker.
  - When an UPDATENODE message comes in:
    - parent[neighbor.id] = currentNode::Id
    - node[adjacentNode.id] = adjacentNode
    - Add the adjacentNode to the queue.
- Issue a STOP message once all the nodes have been processed and workers have sent request task messages.
Worker Thread

• Send in a REQUESTTASK message
• Wait for messages.
• While a STOP message has not been received
  o If a PROCESSNODE message comes in
    • Look at each adjoining node and compute the distance from the source if it is less than the distance in that adjoining node send a UPDATENODE Message with the adjoining node’s information.
    • When done processing the Process Node message send a REQUESTTASK message.
  o If a STOP message comes in exit the main loop close the thread.
Performance
Results

• Tested each variable independently
• Speedups differed greatly depending number of vertices/edges
• Verified smaller graphs by hand, larger graphs were verified against each other and against an implementation of Dijkstra’s SSSP
• Verified output by running diff command
• Worst and best timings for each set of tests will be shown
Changing number of nodes

- We ran from 1000 to 100000 nodes. Worst and best timings will be shown

```
Command line arguments: 1000 248679345 3 6 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>0.108</td>
<td>0.054</td>
<td>17.476</td>
</tr>
<tr>
<td>2</td>
<td>208</td>
<td>194</td>
<td>211</td>
<td>194</td>
<td>0.142</td>
<td>0.047</td>
<td>10.071</td>
</tr>
<tr>
<td>3</td>
<td>148</td>
<td>173</td>
<td>168</td>
<td>148</td>
<td>0.121</td>
<td>0.030</td>
<td>10.651</td>
</tr>
<tr>
<td>4</td>
<td>173</td>
<td>177</td>
<td>179</td>
<td>173</td>
<td>0.119</td>
<td>0.015</td>
<td>9.490</td>
</tr>
<tr>
<td>8</td>
<td>212</td>
<td>312</td>
<td>177</td>
<td>177</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Changing number of nodes

Command line arguments: 100000 248679345 3 6 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>111344</td>
<td>110389</td>
<td>109336</td>
<td>109336</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3993</td>
<td>3524</td>
<td>3606</td>
<td>3524</td>
<td>31.026</td>
<td>15.513</td>
<td>-0.936</td>
</tr>
<tr>
<td>3</td>
<td>2587</td>
<td>2599</td>
<td>2624</td>
<td>2587</td>
<td>42.264</td>
<td>14.088</td>
<td>-0.465</td>
</tr>
<tr>
<td>4</td>
<td>2686</td>
<td>2322</td>
<td>3401</td>
<td>2322</td>
<td>47.087</td>
<td>11.772</td>
<td>-0.305</td>
</tr>
<tr>
<td>8</td>
<td>2363</td>
<td>3434</td>
<td>2204</td>
<td>2204</td>
<td>49.608</td>
<td>6.201</td>
<td>-0.120</td>
</tr>
</tbody>
</table>
Changing number of edges

- We ran from 2-4 outgoing edges to 5-7 outgoing edges per node

```
Command line arguments: 50000 248679345 2 4 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>10085</td>
<td>9803</td>
<td>10136</td>
<td>9803</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1283</td>
<td>1260</td>
<td>1275</td>
<td>1260</td>
<td>7.780</td>
<td>3.890</td>
<td>-0.743</td>
</tr>
<tr>
<td>3</td>
<td>999</td>
<td>1003</td>
<td>1010</td>
<td>999</td>
<td>9.813</td>
<td>3.271</td>
<td>-0.347</td>
</tr>
<tr>
<td>4</td>
<td>897</td>
<td>928</td>
<td>927</td>
<td>897</td>
<td>10.929</td>
<td>2.732</td>
<td>-0.211</td>
</tr>
<tr>
<td>8</td>
<td>1154</td>
<td>939</td>
<td>981</td>
<td>939</td>
<td>10.440</td>
<td>1.305</td>
<td>-0.033</td>
</tr>
</tbody>
</table>
```
Changing number of edges

Command line arguments: 50000 248679345 5 7 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>40899</td>
<td>41029</td>
<td>41141</td>
<td>40899</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2117</td>
<td>2181</td>
<td>2099</td>
<td>2099</td>
<td>19.485</td>
<td>9.742</td>
<td>-0.897</td>
</tr>
<tr>
<td>3</td>
<td>1500</td>
<td>1506</td>
<td>1621</td>
<td>1500</td>
<td>27.266</td>
<td>9.089</td>
<td>-0.445</td>
</tr>
<tr>
<td>4</td>
<td>1428</td>
<td>1435</td>
<td>1462</td>
<td>1428</td>
<td>28.641</td>
<td>7.160</td>
<td>-0.287</td>
</tr>
<tr>
<td>8</td>
<td>1426</td>
<td>1390</td>
<td>1359</td>
<td>1359</td>
<td>30.095</td>
<td>3.762</td>
<td>-0.105</td>
</tr>
</tbody>
</table>
Changing weight value range

- We weight range from 1-50 to 1-50000

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>24604</td>
<td>25578</td>
<td>24733</td>
<td>24604</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1759</td>
<td>1799</td>
<td>1729</td>
<td>1729</td>
<td>14.230</td>
<td>7.115</td>
<td>-0.859</td>
</tr>
<tr>
<td>3</td>
<td>1381</td>
<td>1278</td>
<td>1280</td>
<td>1278</td>
<td>19.252</td>
<td>6.417</td>
<td>-0.422</td>
</tr>
<tr>
<td>4</td>
<td>1164</td>
<td>1170</td>
<td>1153</td>
<td>1153</td>
<td>21.339</td>
<td>5.335</td>
<td>-0.271</td>
</tr>
<tr>
<td>8</td>
<td>1107</td>
<td>1127</td>
<td>1159</td>
<td>1107</td>
<td>22.226</td>
<td>2.778</td>
<td>-0.091</td>
</tr>
</tbody>
</table>
Changing weight value range

Command line arguments:  50000  248679345  3  6  1  50000

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>27009</td>
<td>26268</td>
<td>26567</td>
<td>26268</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1836</td>
<td>1815</td>
<td>1830</td>
<td>1815</td>
<td>14.473</td>
<td>7.236</td>
<td>-0.862</td>
</tr>
<tr>
<td>3</td>
<td>1349</td>
<td>1344</td>
<td>1327</td>
<td>1327</td>
<td>19.795</td>
<td>6.598</td>
<td>-0.424</td>
</tr>
<tr>
<td>4</td>
<td>1255</td>
<td>1259</td>
<td>1232</td>
<td>1232</td>
<td>21.321</td>
<td>5.330</td>
<td>-0.271</td>
</tr>
<tr>
<td>8</td>
<td>1254</td>
<td>1252</td>
<td>1178</td>
<td>1178</td>
<td>22.299</td>
<td>2.787</td>
<td>-0.092</td>
</tr>
</tbody>
</table>
Side effect of changing weight range

• Although weight range barely affects run time, it does affect the resulting parent array.

• Imagine a graph of 500000 nodes but a weight range of only 1-50.
  - Many of the edges will have identical weights.

• Let’s say nodes x and y both connect to z with weight w, and x.minDist = y.minDist.

• Sequential – may see x first and therefore x becomes z’s parent.

• Parallel – may see y first and therefore y becomes z’s parent; path is still as efficient.
Changing seed

- We ran 4 different seed values (changing the seed changes the structure of the graph)

```
Command line arguments: 50000 608974289 3 6 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>25261</td>
<td>24942</td>
<td>25528</td>
<td>24942</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1772</td>
<td>1737</td>
<td>1755</td>
<td>1737</td>
<td>14.359</td>
<td>7.180</td>
<td>-0.861</td>
</tr>
<tr>
<td>3</td>
<td>1290</td>
<td>1265</td>
<td>1272</td>
<td>1265</td>
<td>19.717</td>
<td>6.572</td>
<td>-0.424</td>
</tr>
<tr>
<td>4</td>
<td>1134</td>
<td>1173</td>
<td>1179</td>
<td>1134</td>
<td>21.995</td>
<td>5.499</td>
<td>-0.273</td>
</tr>
<tr>
<td>8</td>
<td>1231</td>
<td>1188</td>
<td>1273</td>
<td>1188</td>
<td>20.995</td>
<td>2.624</td>
<td>-0.088</td>
</tr>
</tbody>
</table>
```
## Changing seed

Command line arguments: 50000 129865487 3 6 1 200

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>26301</td>
<td>27095</td>
<td>25980</td>
<td>25980</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1788</td>
<td>1769</td>
<td>1750</td>
<td>1750</td>
<td>14.846</td>
<td>7.423</td>
<td>-0.865</td>
</tr>
<tr>
<td>3</td>
<td>1254</td>
<td>1284</td>
<td>1296</td>
<td>1254</td>
<td>20.718</td>
<td>6.906</td>
<td>-0.428</td>
</tr>
<tr>
<td>4</td>
<td>1183</td>
<td>1160</td>
<td>1186</td>
<td>1160</td>
<td>22.397</td>
<td>5.599</td>
<td>-0.274</td>
</tr>
<tr>
<td>8</td>
<td>1182</td>
<td>1158</td>
<td>1133</td>
<td>1133</td>
<td>22.930</td>
<td>2.866</td>
<td>-0.093</td>
</tr>
</tbody>
</table>
## Special Cases

- Sequential cannot finish 500000+ node graphs
- We ran parallel up to 5 million nodes
- 3-8 edges, 1-100000 weight, 248679345 seed
- Timings for different runs below
- Capable of more...

<table>
<thead>
<tr>
<th>N = 500000</th>
<th>N = 1000000</th>
<th>N = 2500000</th>
<th>N = 5000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-core: 32363 msec</td>
<td>2-core: 66104 msec</td>
<td>2-core: 200370 msec</td>
<td>2-core: 325853 msec</td>
</tr>
<tr>
<td>3-core: 22553 msec</td>
<td>3-core: 53398 msec</td>
<td>3-core: 152225 msec</td>
<td>3-core: 324056 msec</td>
</tr>
<tr>
<td>4-core: 22225 msec</td>
<td>4-core: 46962 msec</td>
<td>4-core: 135312 msec</td>
<td>4-core: 287511 msec</td>
</tr>
<tr>
<td>8-core: 18146 msec</td>
<td>8-core: 46287 msec</td>
<td>8-core: 126762 msec</td>
<td>8-core: 282442 msec</td>
</tr>
</tbody>
</table>
### Ultimate speedup

**This was the best result we were able to achieve.**

<table>
<thead>
<tr>
<th>NT</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T</th>
<th>Spdup</th>
<th>Eff</th>
<th>EDSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>143798</td>
<td>144738</td>
<td>143711</td>
<td>143711</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3855</td>
<td>4012</td>
<td>3822</td>
<td>3822</td>
<td>37.601</td>
<td>18.800</td>
<td>-0.947</td>
</tr>
<tr>
<td>3</td>
<td>2752</td>
<td>2773</td>
<td>3500</td>
<td>2752</td>
<td>52.221</td>
<td>17.407</td>
<td>-0.471</td>
</tr>
<tr>
<td>4</td>
<td>2441</td>
<td>2381</td>
<td>2423</td>
<td>2381</td>
<td>60.357</td>
<td>15.089</td>
<td>-0.311</td>
</tr>
<tr>
<td>8</td>
<td>2715</td>
<td>2319</td>
<td>2311</td>
<td>2311</td>
<td>62.186</td>
<td>7.773</td>
<td>-0.124</td>
</tr>
</tbody>
</table>
Final Thoughts

• Upgrade to handle doubles – more realistic
• Read in RMAT generated graphs or real road networks
• Cluster version (meh)
Lessons Learned

- The importance of getting the code feature complete early in the quarter. Left plenty of time for validation, performance improvement and analysis.
- It is possible to have speedups greater than K due to parallelization of the algorithm.
- Regression testing is your friend. Periodic regression testing revealed bugs that were introduced by that "one little change".
References


• http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html
Questions?