Hierarchical Parallel A* Algorithm

Abhishek Subramaniam
CSE 633: Parallel Algorithms
Instructor: Dr. Russ Miller
A* algorithm

- A* algorithm is a popular pathfinding algorithm used in game theory and navigation.
- It makes use of a heuristic cost function to find the solution quickly.
- The heuristic cost function uses the sum of two parameters, ‘current cost’ and ‘predicted cost’ to calculate the optimal path.
- All relevant nodes are kept in 2 lists, ‘visited’ and ‘next’.
- The implementation resembles a dynamic programming approach, with the heuristic cost function determining the order in which the nodes are visited.
- Examples of admissible heuristic functions to minimize distance include Manhattan distance and Euclidean distance.
Hierarchical Parallel A* Algorithm

- The graph is broken down into equal size chunks.
- Each processor is assigned a chunk.
Hierarchical Parallel A* Algorithm

• The graph is broken down into equal size chunks.
• Each processor is assigned a chunk.
• The processor then finds the entry/exit nodes for the chunk assigned to it.
• It calculates the actual cost of traversal for each combination of entry/exit nodes.
Hierarchical Parallel A* Algorithm

- The graph is broken down into equal size chunks.
- Each processor is assigned a chunk.
- Each processor then finds the entry/exit nodes for the chunk assigned to it.
- They calculate the actual cost of traversal for each combination of entry/exit nodes.
- Estimate the average cost of horizontal or vertical traversal through the chunk and sends the value to the master.
- The master broadcasts the average cost of traversal through each chunk.
- Run A* algorithm on the node containing the starting node and all other nodes.
- When one solution is found, broadcast the cost to all nodes.
- Run until cutoff.
## Results

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Graph size</td>
<td>2 procs</td>
<td>4 procs</td>
<td>8 procs</td>
<td>16 procs</td>
<td>32 procs</td>
<td>64 procs</td>
</tr>
<tr>
<td>2</td>
<td>10000</td>
<td>64.88</td>
<td>50.26</td>
<td>50.58</td>
<td>47.58</td>
<td>47.04</td>
<td>53.56</td>
</tr>
<tr>
<td>3</td>
<td>40000</td>
<td>272.49</td>
<td>251.81</td>
<td>235.67</td>
<td>221.56</td>
<td>225.43</td>
<td>232.16</td>
</tr>
</tbody>
</table>
Time vs Num Procs

- 1000x1000
- 2000x2000

- 2 procs
- 4 procs
- 8 procs
- 16 procs
- 32 procs
- 64 procs
Factors influencing the results

- Size of the graph.
- Choice of algorithm to find the pairwise shortest path within a chunk.
- Nature of the graph.
- Unexpected behavior of code.
### Path Error

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.411911491</td>
<td>-1.623252616</td>
<td>-1.512489497</td>
<td>-1.932625468</td>
<td>-1.888065592</td>
<td>-2.030657195</td>
</tr>
</tbody>
</table>

- Percentage error
- RMS error = 1373.225
- Which is a 1.748% deviation from the optimal path
Future scope

- Run path smoothing algorithms on the obtained path.
- Find better estimates block length for Manhattan distance.
- Remove ‘optimizations’ and check time.
- Run the algorithm on actual maps.
References


• Implementation of Parallel Path Finding in a Shared Memory Architecture, David Cohen and Matthew Dallas, https://pdfs.semanticscholar.org/9201/badbffa25a272852e05401cedf68f8043a23.pdf


• Implementation of A*, RedBlob Games, https://www.redblobgames.com/pathfinding/a-star/implementation.html

• Game maps, Nathan Sturtevant, http://www.movingai.com/benchmarks/