Computing Betweenness Centrality for Small World Networks on a GPU

Pushkar R. Pande
Georgia Institute of Technology

David A. Bader
Georgia Institute of Technology
Introduction

• Betweenness Centrality of a vertex \( v \) is defined as,

\[
BC(v) = \sum_{S \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}
\]

\( \sigma_{st} \) : Number of shortest paths between vertices \( s \) and \( t \)

\( \sigma_{st}(v) \) : Number of shortest paths between vertices \( s \) and \( t \) through \( v \)

• For a graph with 1 million vertices and 10 million edges, betweenness computation on GPU is accelerated by \(~20\times\) compared to single thread CPU performance.

• Our algorithm for computing betweenness is based on the sequential algorithm [Brandes 2001],

  For each source vertex, perform:

  1. Breadth-First traversal for enumerating shortest paths.
  2. Accumulate dependencies by back propagation.

• Bader and Madduri (ICPP 2006) gave the first parallel implementation. This targeted the highly multithreaded supercomputer, the Cray XMT.
Our GPU Implementation using CUDA

- CUDA offers a hierarchy of threads that allows for multi-level parallelism.
- Vertices \(\{v_1, v_2, ..., v_r\}\) of \(i_{th}\) frontier are distributed among \(NB\) CUDA thread blocks.
- Adjacencies \(\{w_1, w_2, ..., w_s\}\) of the corresponding frontier chunk are then distributed to \(\frac{B}{W}\) warps per thread block.
Performance and Optimizations

- Performance is improved by,
  - Using fast shared memory for buffering writes to global memory.
    - Coalesces global memory access
    - Reduces number of atomic operations on global data structures.
  - Frontier pre-processing for balanced load.
  - Accumulating dependencies by assigning a warp per vertex.
  - Improved work assignment per warp using virtualized warp size.

- GPU performance for betweenness compared to CPU with number of threads $nt = \{1, 16\}$ for a subset of 500 source vertices,

<table>
<thead>
<tr>
<th>Graph</th>
<th>#Vertices</th>
<th>#Edges</th>
<th>CPU Time (s)</th>
<th>GPU Time (s)</th>
<th>Speed up on Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$nt = 1$</td>
<td>$nt = 16$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tesla</td>
<td>Fermi</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$nt = 1$</td>
<td>$nt = 16$</td>
<td></td>
</tr>
<tr>
<td>syn1.gr</td>
<td>262,144</td>
<td>2,097,152</td>
<td>24.19</td>
<td>5.47</td>
<td>9.16</td>
</tr>
<tr>
<td>syn2.gr</td>
<td>524,288</td>
<td>4,194,304</td>
<td>80.72</td>
<td>15.78</td>
<td>12.75</td>
</tr>
<tr>
<td>syn3.gr</td>
<td>1,048,576</td>
<td>8,388,608</td>
<td>184.11</td>
<td>32.68</td>
<td>16.28</td>
</tr>
<tr>
<td>syn4.gr</td>
<td>1,000,000</td>
<td>10,000,000</td>
<td>93.63</td>
<td>17.35</td>
<td>15.32</td>
</tr>
<tr>
<td>syn5.gr</td>
<td>1,000,000</td>
<td>10,000,000</td>
<td>232.79</td>
<td>33.87</td>
<td>19.68</td>
</tr>
</tbody>
</table>

GPU: NVIDIA C1060 (Tesla), 1.3 GHz, 30 × 8 cores
NVIDIA M2070 (Fermi), 1.15 GHz, 14 × 32 cores
CPU: 2 quad-core Intel Xeon E5530, 2.4 GHz, hyper-threading enabled