Effective Evaluation of Betweenness Centrality on Multi-GPU systems

Massimo Bernaschi\textsuperscript{1},
Giancarlo Carbone\textsuperscript{2} Flavio Vella\textsuperscript{2}.

\textsuperscript{1}IAC-National Research Council of Italy
\textsuperscript{2}Sapienza University of Rome
Betweenness Centrality

A metric to measure the influence or relevance of a node in a network

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- $\sigma_{st}$ is the number of shortest paths from $s$ to $t$
- $\sigma_{st}(v)$ is the number of shortest paths from $s$ to $t$ passing through a vertex $v$

\[\sigma_{st}(v) = 1\]
\[\sigma_{st} = 2\]

BC$(v) = 0.5$
Betweenness Centrality

Measure of the influence or relevance of a node in a network

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

• \( \sigma_{st} \) is the number of shortest paths from \( s \) to \( t \)
• \( \sigma_{st}(v) \) is the number of shortest paths from \( s \) to \( t \) passing through a vertex \( v \)

Time Complexity (Brandes’ Algorithm)

• \( O(nm) \) for unweighted graphs
• \( O(nm + n^2 \log n) \) for weighted graphs
Brandes’ algorithm (2001)

1. Compute BFS starting from $s$
   1. $\sigma[v]$ for each $v$ visited during the traversal of $G$
   2. Predecessors array
2. Sum all dependencies of $s$ on predecessors array
3. Update BC score for each $v \neq s$
Brandes’ algorithm (2001)

1. Compute BFS starting from $s$
   1. $\sigma[v]$ for each $v$ visited during the traversal of $G$
   2. Predecessors array
2. Sum all dependencies of $s$ on predecessors array
3. Update BC score for each $v \neq s$

Unfeasible for large-scale graphs!!!
GPU-based Brandes implementations

Exploiting GPU parallelism to improve the performance

Well-know problems due to Irregular Access Patterns and unbalanced load distribution on traversal-based algorithms

Vertex vs Edge Parallelism
• Vertex-Parallelism: each thread is assigned to its own vertex
• Edge-Parallelism: each thread is in charge of a single edge
• Hybrid techniques (i.e., McLaughlin, A. and Bader, D. "Scalable and high performance betweenness centrality on the GPU [SC 2014]")
Multi-GPU-based Brandes implementations

“Scalable and high performance betweenness centrality on the GPU” [McLaughlin2014]

• Strategy
  • The graph is replicated among all computational nodes
  • Each root vertex can be processed independently
  • Use MPI_Reduce to update the bc score

• Advantages
  • Good scalability on graphs with one connected component

Main drawback

Data replication **limits the maximum size** of the graph!
Algebraic Approach


• Strategy
  • Synchronous SpMM Multi-source Traversal based on Batch Algorithm [Robinson2008]
  • Graph partitioning based on a 2-D decomposition [Yoo2005]

• Drawback
  • No Heuristics
  • Different BFS-trees may have different depths

Load unbalancing intra- and inter-node on Real world graphs
MGBC Parallel Distributed Strategy

Multilevel parallelization of Brandes’ algorithm + Heuristics

- Node-level parallelism
  - CUDA threads work on the same graph within one computing node

- Cluster-level parallelism
  - The graph is distributed among multiple computing nodes (each node owns a subset)

- Subcluster-parallelism
  - Computing nodes are grouped in subsets each working independently on its own replica of the same graph

4-7 April 2016
GTC16, Santa Clara, CA, USA
Node-level parallelism

- Distance BFS
  - Exploiting atomic-operations on Nvidia Kepler architecture
  - Data-thread mapping based on prefix sum and binary search.

First Optimization!
Save extra-computation paid to have the regular access pattern
Avoiding prefix scan in dependency accumulation
Cluster-level parallelism

- 2-Dimensional partitioning
  - The graph is distributed across a 2-D mesh
  - Only $\sqrt{p}$ processors involved in the communication at time during traversal steps
- No Predecessors (contrary to Brandes) → No predecessor exchanging in distributed-system

Second Optimization!

Pipelining CPU-GPU and MPI Communications
Subcluster-level parallelism

• Multiple independent searches
  • a batch of vertices is assigned to each SC
  • a vertex at time inside a SC
• Configurable graph replication ($fr$) and graph distribution ($fd$) factors
  • the $fr$-replicas are assigned one for each SC
  • the graph is mapped onto each SC according to $fd$.
• MPI Communicators Hierarchy

Advantage!
Each subcluster updates BC score only at the end of its own searches.
No synchronization among subclusters

$p=16$, $fd=4$
$fr = \frac{p}{d} = 4$
Experimental Setup

Piz Daint@CSCS
#6 TOP500.org
(http://www.top500.org/system/177824)

• Cray XC30 system with 5272 computing nodes
• Each node:
  • CPU Intel Xeon E5-2670 with 32GB of DDR3
  • GPU Nvidia Tesla K20x with 6 GB of DDR5
• SW Environment:
  • GCC 4.8.2
  • CUDA 6.5
  • Cray MPICH 6.2.2
### Comparison Single-GPU

<table>
<thead>
<tr>
<th>SNAP Graph</th>
<th>SCALE</th>
<th>EF</th>
<th>MC</th>
<th>S1</th>
<th>S2</th>
<th>G</th>
<th>MGBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoadNet-CA</td>
<td>20.91</td>
<td>1.41</td>
<td><strong>0.067</strong></td>
<td>0.371</td>
<td>0.184</td>
<td>0.298</td>
<td><strong>0.085</strong></td>
</tr>
<tr>
<td>RoadNet-PA</td>
<td>20.05</td>
<td>1.40</td>
<td><strong>0.035</strong></td>
<td>0.210</td>
<td>0.114</td>
<td>0.212</td>
<td><strong>0.071</strong></td>
</tr>
<tr>
<td>com-Amazon</td>
<td>18.35</td>
<td>2.76</td>
<td>0.008</td>
<td>0.009</td>
<td>0.006</td>
<td>-</td>
<td><strong>0.005</strong></td>
</tr>
<tr>
<td>com-LJ</td>
<td>21.93</td>
<td>8.67</td>
<td>0.210</td>
<td>0.143</td>
<td><strong>0.084</strong></td>
<td>-</td>
<td>0.100</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>21.55</td>
<td>38.14</td>
<td>0.552</td>
<td>0.358</td>
<td><strong>0.256</strong></td>
<td>-</td>
<td>0.314</td>
</tr>
</tbody>
</table>

**S** = scale  
**EF** = Edge Factor  
|V| = 2^{SCALE} and  
|M| = EF x 2^{SCALE}  
Avg. Time (sec)

Mc = McLaughlin et al. "Scalable and high performance betweenness centrality on the GPU" [SC 2014].  
S1 and S2 = Saryuce et al. "Betweenness centrality on GPUs and heterogeneous architectures" [GPGPU 2013].  
Strong Scaling

<table>
<thead>
<tr>
<th>G</th>
<th>SCALE</th>
<th>EF</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-MAT</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>Twitter</td>
<td>~25</td>
<td>~35</td>
</tr>
<tr>
<td>com-Friendster</td>
<td>~26</td>
<td>~27</td>
</tr>
</tbody>
</table>
Subcluster

16 Processors
1 Cluster in a 4x4 Mesh

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

GPUs (p) | fd | fr | Time (hours)
---------|----|----|--------------
2        | 2x1| 1  | ≈ 211
128      | 2x1| 64 | ≈ 3.5
256      | 2x1| 128| ≈ 1.7
256      | 2x2| 64 | ≈ 2.3

16 Processors
4 sub-clusters in a 2x2 Mesh

SNAP com-Orkut graph: Vertices ≈ 3E+06 – Edges = 2E+08
Optimizations Impact

a) R-MAT S23 EF32

b) Twitter

c) Prefix-sum optimization

4-7 April 2016
1-Degree Reduction

- Vertices with only one neighbour
- Removing 1-degree nodes from the graph (preprocessing)
- Reformulating the evaluation of the dependency
- First distributed implementation
1-Degree Results

Benefits of 1-degree reduction
1. Avoid execution of BC calculation for 1-degree vertices
2. Reduce number of vertices to traverse

<table>
<thead>
<tr>
<th>Graph</th>
<th>1-degree</th>
<th>Preprocessing (sec)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-Youtube</td>
<td>53%</td>
<td>0.62</td>
<td>2.8x</td>
</tr>
<tr>
<td>roadNet-CA</td>
<td>16%</td>
<td>0.55</td>
<td>1.2x</td>
</tr>
<tr>
<td>com-DBLP</td>
<td>14%</td>
<td>0.19</td>
<td>1.2x</td>
</tr>
<tr>
<td>com-Amazon</td>
<td>8%</td>
<td>0.16</td>
<td>1.1x</td>
</tr>
<tr>
<td>R-MAT 20 E-16</td>
<td>13%</td>
<td>1.2</td>
<td>1.3x</td>
</tr>
</tbody>
</table>

*Source: Stanford Large Network Dataset Collection

Impact of 1-degree: computation (top), communication (middle) and sigma-overlap (bottom) on R-MAT 20 and EF 4, 16 and 32.
2-Degree Heuristics

Key Idea
Deriving BFS-tree of a 2-degree vertex from BFS-trees of its own neighbours

Let \(a\) be a 2-degree vertex and \(b, c\) be its own neighbours. \(d\) is the distance from the source vertex.

\[ d_a(v) = \min\{d_b(v), d_c(v)\} + 1 \]
DMF Algorithm

we can derive SSSP from $a$

$$\sigma_a(v) = \begin{cases} 
\sigma_b(v) & \text{if } d_b(v) < d_c(v) \\
\sigma_c(v) & \text{if } d_b(v) > d_c(v) \\
\sigma_b(v) + \sigma_c(v) & \text{if } d_b(v) == d_c(v) 
\end{cases}$$

Dynamic Merging of Frontiers Algorithm
1. Compute the SSSP from $b$ and $c$ storing the number of shortest path and distance vectors of both
2. Compute level-by-level the Dependency Accumulation of $b$ and $c$ concurrently. The contributions of $a$ for each visited vertex $v$ is computed on-the-fly
DMF Algorithm Example

Vertex \( b \) does not compute the dependency of \( a \)

Dependency of \( c \) \{d,e\} at \( d=3 \)
- \( d_b(d) = 2 \)
- \( d_c(e) = 3 \)

Nothing to do for \( a \)!!!!

<table>
<thead>
<tr>
<th>Vertices</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
DMF Algorithm Example

Dependency of \( b \{3,7,9 \} \) at \( d=2 \)
\[
\begin{align*}
da(c) &= 2 \quad db(c) = 2 \\
da(g) &= 2 \quad db(g) = 2 \\
da(i) &= 2 \quad db(i) = 2
\end{align*}
\]

Vertex \( b \) computes the dependency of \( a \) on \( i \) (partially)

Dependency of \( c \{b,f,h\} \) at \( d=2 \)
\[
\begin{align*}
db(b) &= 1 \quad dc(b) = 2 \\
db(f) &= 1 \quad dc(f) = 2 \\
db(i) &= 2 \quad dc(i) = 2
\end{align*}
\]

**b** and **c** contributes to Dependency of **a** !!!!

<table>
<thead>
<tr>
<th>Vertices</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
BC Analysis of a real-world graph
Amazon product co-purchasing network

Dataset statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>334863</td>
</tr>
<tr>
<td>Edges</td>
<td>925872</td>
</tr>
<tr>
<td>Nodes in largest WCC</td>
<td>334863 (1.000)</td>
</tr>
<tr>
<td>Edges in largest WCC</td>
<td>925872 (1.000)</td>
</tr>
<tr>
<td>Nodes in largest SCC</td>
<td>334863 (1.000)</td>
</tr>
<tr>
<td>Edges in largest SCC</td>
<td>925872 (1.000)</td>
</tr>
<tr>
<td>Average clustering coefficient</td>
<td>0.3967</td>
</tr>
<tr>
<td>Number of triangles</td>
<td>667129</td>
</tr>
<tr>
<td>Fraction of closed triangles</td>
<td>0.07925</td>
</tr>
<tr>
<td>Diameter (longest shortest path)</td>
<td>44</td>
</tr>
<tr>
<td>90-percentile effective diameter</td>
<td>15</td>
</tr>
</tbody>
</table>
Conclusion and future works

• Data-thread mapping approach is effective on graphs with different characteristics
• First 2-D fully distributed BC
  • Good scaling up to 128 GPUs
• Sub-clustering easily scale even with many GPUs
  • Linear scaling up to 256 GPUs
• Distributed 1-degree reduction heuristics
• New heuristics to solve 2-degree vertex on-the-fly
• MGBC computes real-world graph 234M < 2 hours
• Future works
  • Generalization of DFM
  • Heuristics on Algebraic Approach
THANK YOU!

massimo.bernaschi@cnr.it