Speeding Up GPU Graph Processing Using Structural Graph Properties

Merijn Verstraaten

Advisor: Ana Lucia Varbanescu
Promotor: Cees de Laat

University of Amsterdam
Graphs Analytics

Graphs:

- Vertices
- Edges

Who cares?
General Purpose GPU Computing

Speeding Up GPU Graph Processing

Diagram of a GPU architecture with details on processors, memory hierarchy, and interconnections.
My Work So Far…

Systematic Benchmarking

Analytical Modeling

Graph Generator

Real World Datasets

Machine Learning

• Vary individual parameters
• Evolutionary graph generator
• Scaling to graph generation to large sizes

• Many different BFS implementations
• Benchmark on SNAP & KONECT
• Different vertex/edge orderings
• Per level timings

• Determine important parameters
• Predict fastest implementation
• Implementation switching BFS

• Multiple PageRank implementations
• Sequential workload model
• Parallel execution model

• Vary individual parameters
• Evolutionary graph generator
• Scaling to graph generation to large sizes

• Many different BFS implementations
• Benchmark on SNAP & KONECT
• Different vertex/edge orderings
• Per level timings

• Determine important parameters
• Predict fastest implementation
• Implementation switching BFS
Breadth-First Search: Implementations

**Edge-centric**

**Vertex Push**

**Vertex Pull**

Useless Frontier Thread
Useful Frontier Thread
Frontier Node
Updated Node
Accessed Node
Relative Performance of Implementations

There is no “best”!

M. Verstraaten
Speeding Up GPU Graph Processing
Relative Performance Within a Single Traversal

Sticking to one implementation costs us!

M. Verstraaten

Speeding Up GPU Graph Processing
Predicting the Best Implementation

Weapon of Choice: Decision Trees

Features:
- black-box approach
- predictive power and high accuracy
- require small number of samples

Training Parameters:
- Degree distribution
- Frontier size
- Percentage discovered
- Vertex count
- Edge count
Trained Models

Feasibility:
Accuracy: ~98%
Avg. Prediction Time: 144 ns (σ = 165 ns)
Min. BFS Step: 20 ms
(Re)loading graph representation: Stupidly slow

Classic time-space trade-off.
## Overall Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Optimal</th>
<th>1–2×</th>
<th>&gt;5×</th>
<th>&gt;20×</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>56%</td>
<td>41%</td>
<td>1%</td>
<td>0.5%</td>
<td>1.40×</td>
<td>236×</td>
</tr>
<tr>
<td>Oracle</td>
<td>23%</td>
<td>55%</td>
<td>2%</td>
<td>0%</td>
<td>1.65×</td>
<td>8.5×</td>
</tr>
<tr>
<td>Edge List</td>
<td>10%</td>
<td>61%</td>
<td>7%</td>
<td>0.4%</td>
<td>2.22×</td>
<td>38×</td>
</tr>
<tr>
<td>Rev. Edge List</td>
<td>5%</td>
<td>59%</td>
<td>15%</td>
<td>0.6%</td>
<td>2.92×</td>
<td>50×</td>
</tr>
<tr>
<td>Vertex Pull</td>
<td>0%</td>
<td>15%</td>
<td>27%</td>
<td>24%</td>
<td>38.62×</td>
<td>2,671×</td>
</tr>
<tr>
<td>Vertex Push</td>
<td>9%</td>
<td>15%</td>
<td>53%</td>
<td>29%</td>
<td>39.66×</td>
<td>1,048×</td>
</tr>
<tr>
<td>Vertex Push Warp</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>30%</td>
<td>18.69×</td>
<td>97×</td>
</tr>
</tbody>
</table>

Averaged over 248 KONECT graphs.
Comparison with State-of-the-Art: Best & Worst

Even better if we include Gunrock in model?
Related Work

**Single Node:**
Boost Graph Library (BGL), GraphMat, Ligra

**Distributed Systems:**
Giraph, GraphLab, GraphX, PGX.D, Pregel

**GPU Frameworks:**
CuSha, Gunrock, MapGraph, Medusa, nvGraph

**Hybrid Systems:**
Galois, Totem
Takeaway

No single best implementation for irregular GPU algorithms

Large potential performance gains for graph algorithms

Not all machine learning leaves you clueless

Variable importance can guide analytical modelling

Questions?
S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, Verstraaten, Merijn, and H. Chafi.  
**PGX.D: a fast distributed graph processing engine.**  

S. Hong, T. Manhardt, J. van der Lugt, Verstraaten, Merijn, and H. Chafi.  
**Distributed graph processing system that support remote data read with proactive bulk data transfer, Apr. 3 2015.**  

**Towards benchmarking IaaS and PaaS clouds for graph analytics.**  

**Statistical performance analysis of an ant-colony optimisation application in S-Net.**  
W. L. Ngai, A. L. Varbanescu, and **Verstraaten, Merijn**.
*Towards benchmarking IaaS and PaaS clouds for graph analytics.*

R. Poss, **Verstraaten, Merijn**, F. Penczek, C. Grelck, R. Kirner, and A. Shafarenko.
*S+Net: extending functional coordination with extra-functional semantics.*

R. Poss, **Verstraaten, Merijn**, and A. Shafarenko.


*On mapping distributed S-Net to the 48-core Intel SCC processor.*
In *3rd MARC Symposium, Fraunhofer IOSB, Ettlingen, Germany*, 2011.
Verstraaten, Merijn, S. Kok, R. Poss, C. Grelck, et al.
Task migration for S-Net/LPEL.
In 2nd HiPEAC Workshop on Feedback-Directed Compiler Optimization for Multicore Architectures (FD-COMA’13), Berlin, Germany. HiPEAC, 2013.

Verstraaten, Merijn and S.-B. Scholz.
On predicting the impact of resource redistributions in streaming applications.

Verstraaten, Merijn, A. L. Varbanescu, and C. de Laat.
Quantifying the performance impact of graph structure on neighbour iteration strategies for pagerank.

Verstraaten, Merijn, A. L. Varbanescu, and C. de Laat.
Synthetic graph generation for systematic exploration of graph structural properties.
In European Conference on Parallel Processing, pages 557–570. Springer, Cham, 2016.
J. van der Lugt, Verstraaten, Merijn, S. Hong, and H. Chafi.
Method of achieving intra-machine workload balance for distributed graph-processing systems, May 21 2015.

Efficient memory copy operations on the 48-core Intel SCC processor.

Can portability improve performance?: An empirical study of parallel graph analytics.