Parallel Breadth First Search on GPU Clusters using MPI and GPUDirect

Speaker: Harish Kumar Dasari, Scientific Computing and Imaging Institute, University of Utah

Advisor: Dr. Martin Berzins, SCI, University of Utah

In collaboration with Dr. Zhisong Fu, Bryan Thompson, Systap, LLC.

http://sourceforge.net/projects/mpgraph/
Introduction

- **Breadth First Search**: It is a graph search algorithm that begins at the root vertex and explores all the connected vertices, traversing all vertices of a particular level before traversing the vertices of the next level.
- At the end of the BFS we can find out the level of a vertex if it is connected to the root element and also its predecessor.
- Useful in social media, logistics and supply chains, e-commerce, counter-terrorism, fraud detection etc.
Introduction

● Why BFS?
  ○ Least work/byte of the graph algorithms
  ○ Building blocks for many other graph problems

● Why GPUs?
  ○ High Performance: NVIDIA K40 peak performance: 1.43 Tflops
  ○ High Energy Efficiency
  ○ Central for next generation of architectures
Related Work

- Scalable GPU Graph Traversal - Single node multi-GPU, Merrill, Garland et al.
  - Around 12x speedup over idealized multi-core CPU
  - 3 GTEPS on single node

- MapGraph, Fu, Thompson et al.
  - Generalized for many graph algorithms using Gather Apply Scatter (GAS) abstraction
  - Provides an easy framework for the developer to develop solutions to other graph problems like SSSP (Single Source Shortest Path), PageRank etc.
Related Work

- Breaking the Speed and Scalability barriers for graph exploration on distributed-memory machines by Checconi, Petrini et al from IBM
  - BFS on Bluegene supercomputers, uses CPUs
  - On Graph500 data sets, on the order of $2^{40}$ edges
  - 254 billion edges/sec with 64k cores
  - Uses 2D partitioning and waves for communication
Partitioning of the Graph

- RMAT graph generated using the Graph500 generator
  - Scale Free
  - Follows power law, at least asymptotically
  - Undirected edges are converted to directed edges

- 2-D Partitioning of directed edges with a square layout

- Each subgraph resides in GPU memory

- Bitmaps used to represent the frontiers
  - Bit is set to 1 to represent active vertex

![Diagram of partitioned graph and bitmap]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bitmap

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The Algorithm and Communication

- Each GPU $G_{ij}$ takes in its input frontier bitmap $In_i^t$ and perform BFS on its subgraph to produce $Out_{ij}^t$
- Parallel Scan for bitmaps along the row $R_i$ to produce prefix sum $Prefix_{ij}$ in Bitwise-OR
The Algorithm and Communication

- The Prefix is used to determine the vertices the GPU is assigned for predecessor updates.
- $Out_i^t$ is broadcast across row $R_i$, and also as $In_i^{t+1}$ across column $C_i$.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Frontier IN for A and B
Frontier IN for C and D
Scan for A and B
Scan for C and D

Frontier OUT for A and B
Frontier OUT for C and D

In$_1^t$, In$_2^t$, In$_3^t$, In$_4^t$ | G$_{11}$ | | | |
| R$_1$ | G$_{22}$ | | | |
| R$_2$ | | G$_{33}$ | | |
| R$_3$ | | | G$_{44}$ | |
| R$_4$ | | | | Out$_1^t$
| | | | Out$_2^t$
| | | | Out$_3^t$
| | | | Out$_4^t$
Experimental Setup

- 32 nodes and 64 NVIDIA K20c GPUs with 5GB DDR5 memory
- Two Mellanox InfiniBand SX6025 cards per node
- CUDA 5.5 used for these results
- Used GPUDirect support in MVAPICH2-GDR to avoid explicit copy of messages to host memory
Results - Strong Scaling

- The scale of the problem remains the same as we increase the computational resources (GPUs)
- \( \text{GTEPS} = \text{Giga(Billion) Traversed Edges Per Second} = 10^9 \text{ edges per second} \)

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Scale</th>
<th>Time</th>
<th>GTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>25</td>
<td>0.075</td>
<td>2.5</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>0.066</td>
<td>6.3</td>
</tr>
<tr>
<td>36</td>
<td>25</td>
<td>0.059</td>
<td>15.0</td>
</tr>
<tr>
<td>64</td>
<td>25</td>
<td>0.047</td>
<td>29.1</td>
</tr>
</tbody>
</table>

Number of Vertices in graph = \( 2^{\text{SCALE}} \)
Number of Directed Edges in graph = \( 32 \times 2^{\text{SCALE}} \)
Results - Weak Scaling

- Problem size grows proportional to the growth in computational resources (GPUs)
- Each GPU has same amount of work?

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Scale</th>
<th>Time</th>
<th>GTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>0.0254</td>
<td>14.3</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>0.0429</td>
<td>16.4</td>
</tr>
<tr>
<td>16</td>
<td>25</td>
<td>0.0715</td>
<td>18.1</td>
</tr>
<tr>
<td>64</td>
<td>27</td>
<td>0.1478</td>
<td>22.7</td>
</tr>
</tbody>
</table>

Number of Vertices in graph = $2^{\text{Scale}}$
Number of Directed Edges in graph = $32 \times 2^{\text{Scale}}$
Communication vs Computation

- Even if the work per GPU remains the same, the communication costs grow
- Impacts weak Scalability
Breakdown of Timings

- Near constant communication times across iterations
- Load Imbalance in the first iterations
Challenges faced

- Working with cutting edge Hardware and Software stack
- Lack of expertise on the new software
- Crashes due to the old GPU Drivers
- Thanks for the OSU MVAPICH team for the excellent support
Future Scope

- GAS abstraction for the distributed graph processing platform
- Message compression in iterations with small frontiers
- Graph compression to allow us to load bigger graphs to GPU memory
- Runtime engine to overlap computation and Communication, similar to the one in Uintah (www.uintah.utah.edu)
Conclusion

- Implemented a high Performance distributed parallel Breadth First Search (BFS) on GPU cluster
- Implemented a parallel scan for Bitwise-OR reduction of bitmaps
- Implemented compression methods for messages and found that it increases wait times
- This work has been funded by DARPA STTR Phase-1
Queries?

Thank You