Using MVAPICH2-GDR for multi-GPU data parallel graph analytics

T. James Lewis
Overview

1. Review of previous results
   • Slides from MUG 2014

2. Recent issues
   • Current CUDA driver/runtime reduced performance
   • Single-node system performs worse than cluster

3. Future plans
   • GPU accelerated graph database
   • Translation from Scala DSL
Slides from MUG 2014
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.

![Diagram of Gather, Apply, and Scatter operations in graph traversal](image)
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

![Diagram of graph exploration process](image-url)
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 a graph with nodes A, B, C, D and edges showing partitioning into GPUs]
The Algorithm and Communication

- Each GPU $G_{ij}$ takes in its input frontier bitmap $I_{n_i^t}$ 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 $C_i$. 

![Diagram of the algorithm and communication](image-url)
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)
- GTEPS = Giga(Billion) Traversed Edges Per Second = $10^9$ 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
Recent Issues

Reduced performance with version updates
Nvidia PSG cluster: 16 K40s across 4 nodes

- Cuda 5.5
- MVAPICH2-GDR 2.0b
- 18.74 GTeps

- Cuda 7.0
- MVAPICH2-GDR 2.1 rc2
- 9.13 GTeps

Only 48% of previous performance
Recent Issues

Disappointing Single Node Performance
16 MPI processes
Scale 25 graph

- Cirrascale
- 8x K80 (16 GPU)
- Full PCIe-3 16x
- Cuda 7.0
- MVAPICH2-GDR 2.1 rc2
- 2.20 GTeps

- Nvidia PSG Cluster
- 4 Nodes, 16 K40
- Cuda 7.0
- MVAPICH2-GDR 2.1 rc2
- 9.13 GTeps
Future Plans

• GPU accelerated graph database
  • BlazeGraph graph database (Java)
  • Accelerate SPARQL queries with GPU

• Translation from Scala DSL
  • DSL defined operators
  • Graph algorithms written in Scala
  • Scala translated to native GPU code
  • High performance without GPU experts
Questions