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

T. James Lewis

systap™

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.

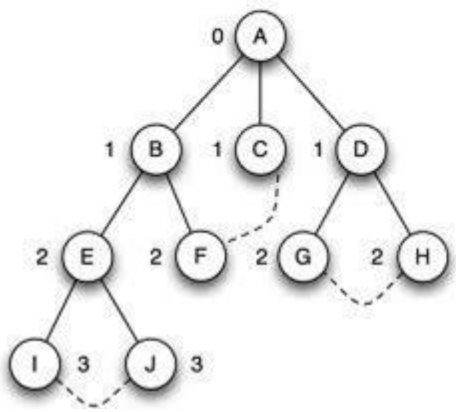
http://sourceforge.net/projects/mpgraph/

www.sci.utah.edu

THE UNIVERSITY OF UTAH

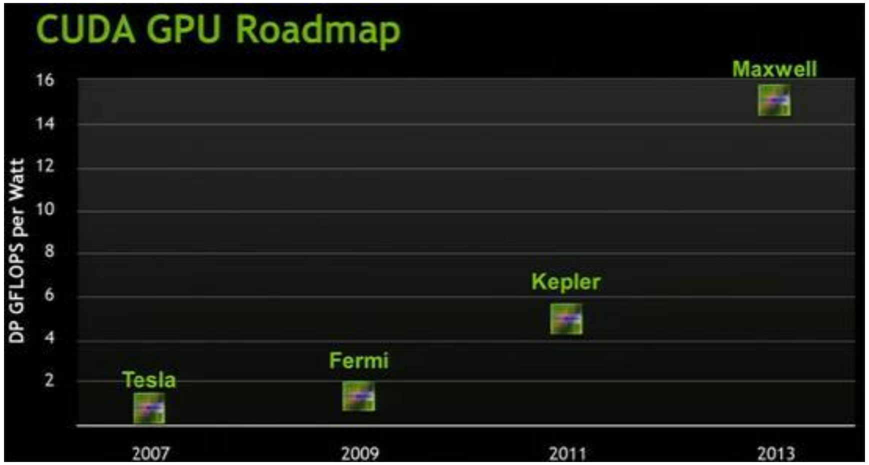
Introduction

- 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.



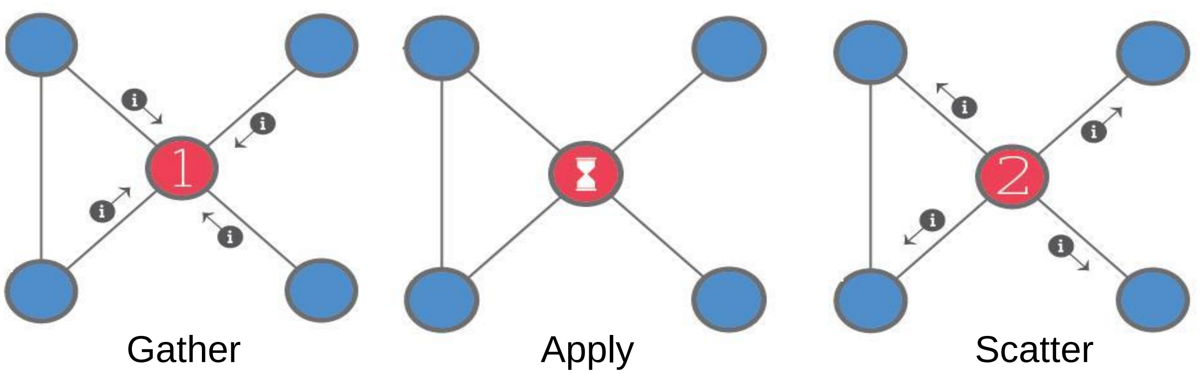
Introduction

- 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



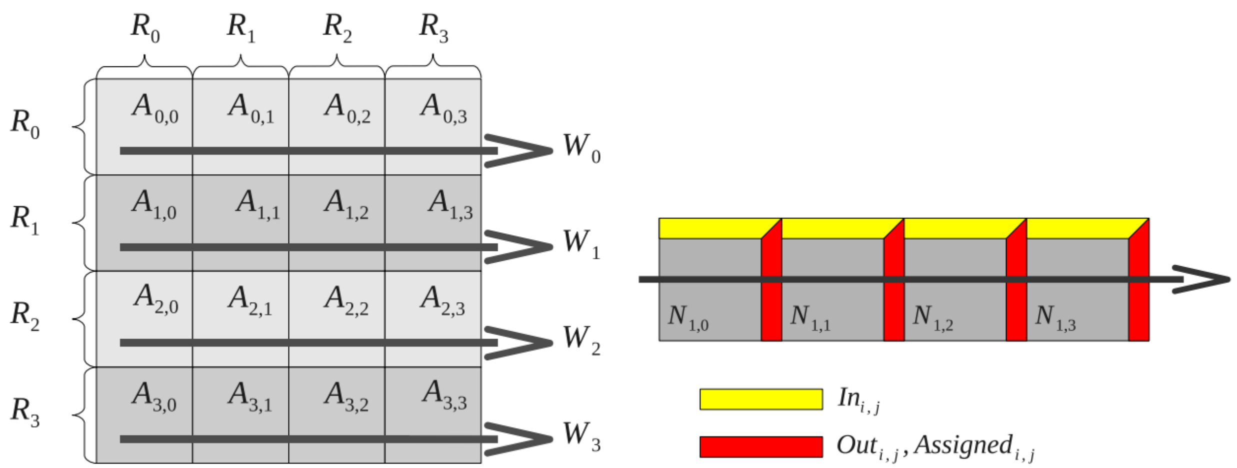
Related Work

- 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.



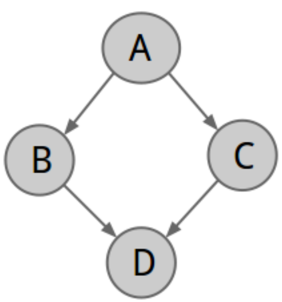
Related Work

- 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



	A	B	C	D
A	0	0	0	0
B	1	0	0	0
C	1	0	0	0
D	0	1	1	0

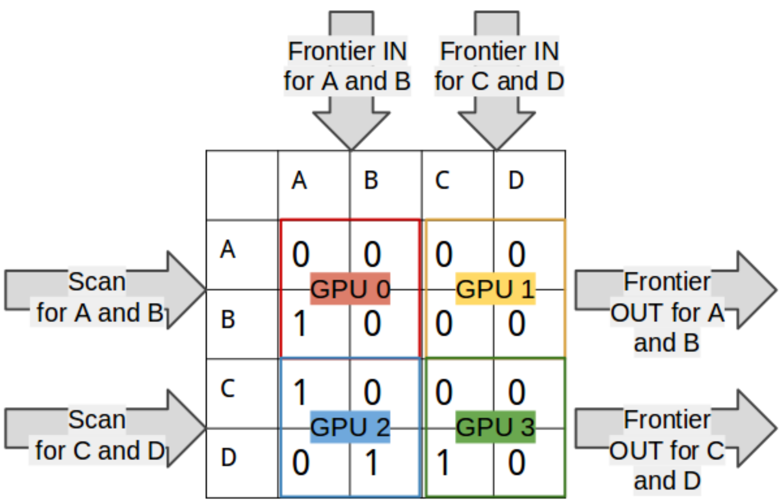
GPU 0 (rows A–B, columns A–B); GPU 1 (rows A–B, columns C–D); GPU 2 (rows C–D, columns A–B); GPU 3 (rows C–D, columns C–D)

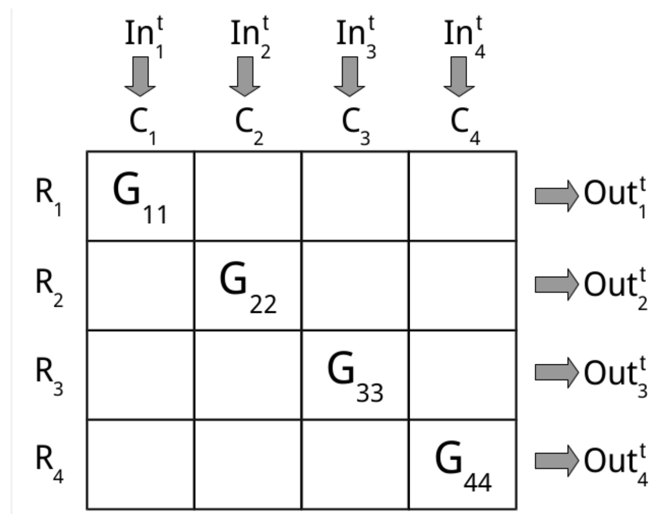
Bitmap

A	B
0	1

The Algorithm and Communication

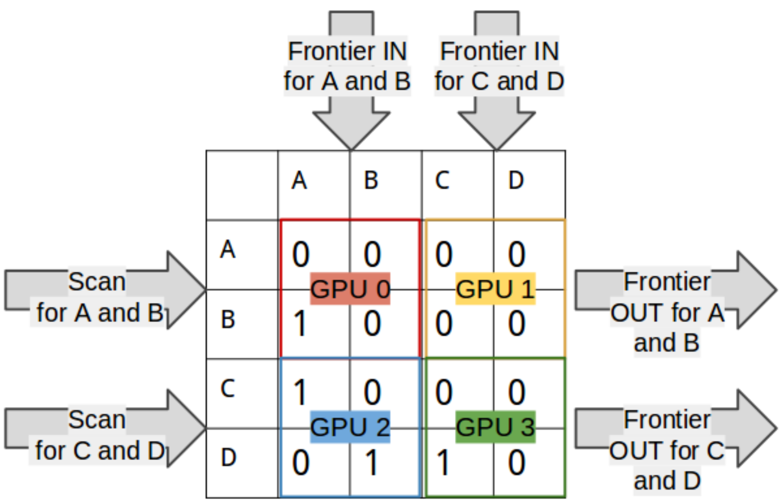
- Each GPU G_{ij} takes in its input frontier bitmap In_i^t and perform BFS on its subgraph to produce Out_{ij}^t
- Parallel Scan for bitmaps along the row R_i to produce prefix sum $Prefix_{ij}$ in Bitwise-OR

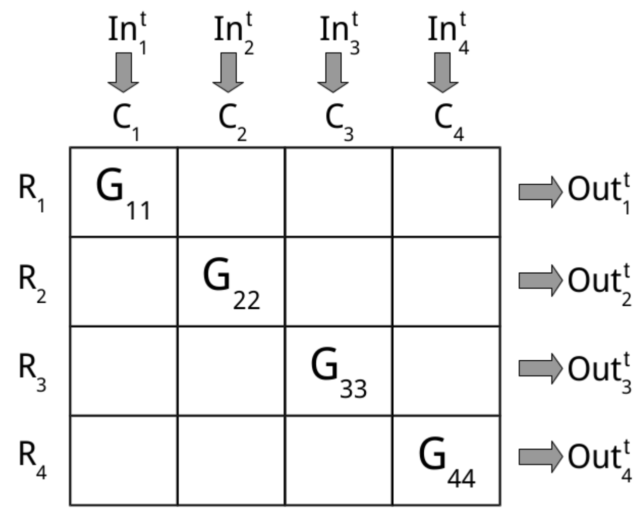




The Algorithm and Communication

- 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

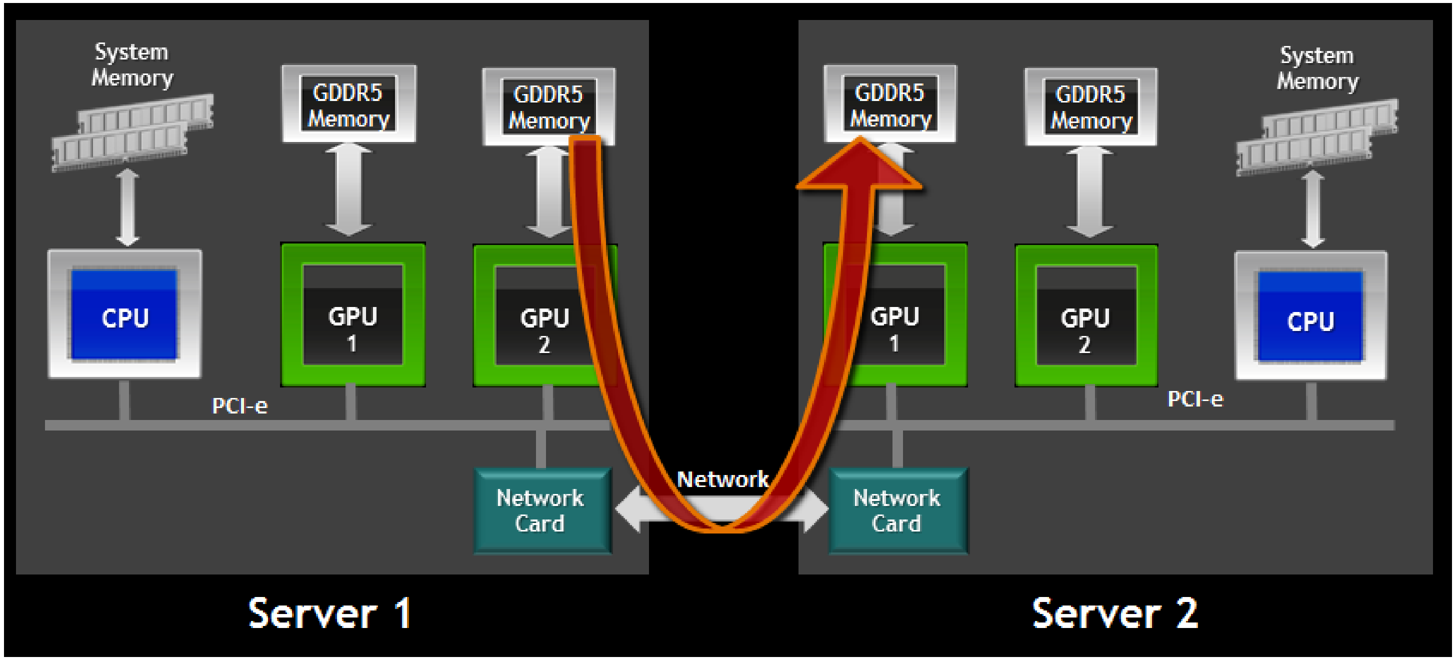




www.sci.utah.edu
THE UNIVERSITY OF UTAH

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

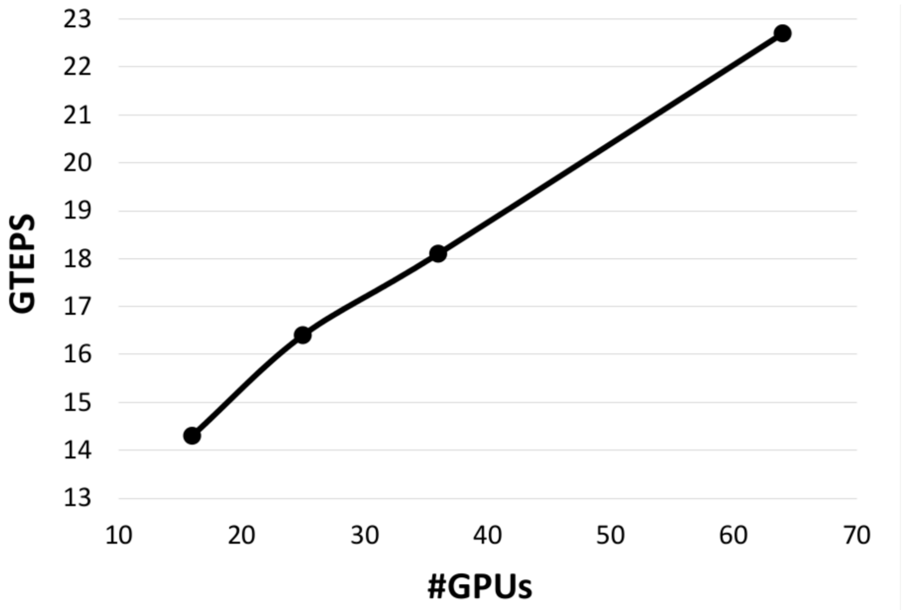


Results - Strong Scaling

- 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

GPUs	Scale	Time	GTEPS
16	25	0.075	2.5
25	25	0.066	6.3
36	25	0.059	15.0
64	25	0.047	29.1

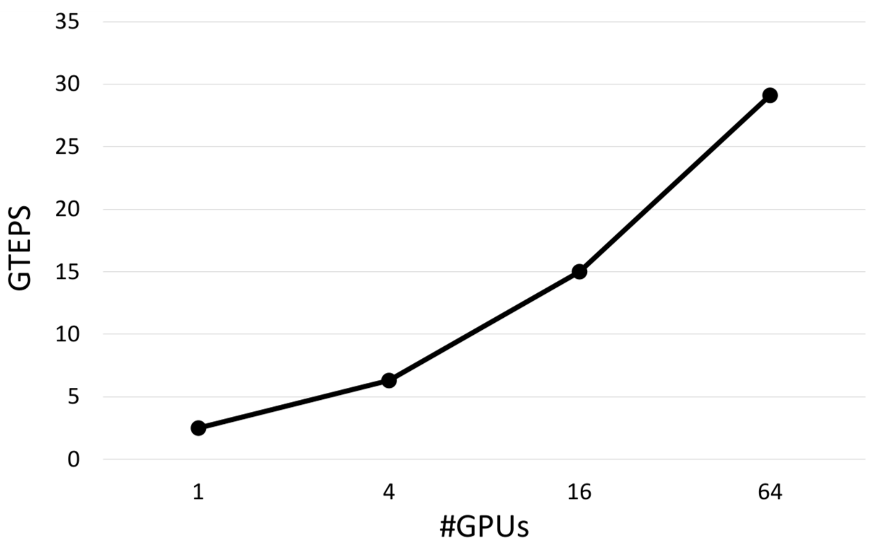
Number of Vertices in graph = 2^{SCALE}
Number of Directed Edges in graph = $32*2^{SCALE}$



Results - Weak Scaling

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

GPUs	Scale	Time	GTEPS
1	21	0.0254	14.3
4	23	0.0429	16.4
16	25	0.0715	18.1
64	27	0.1478	22.7

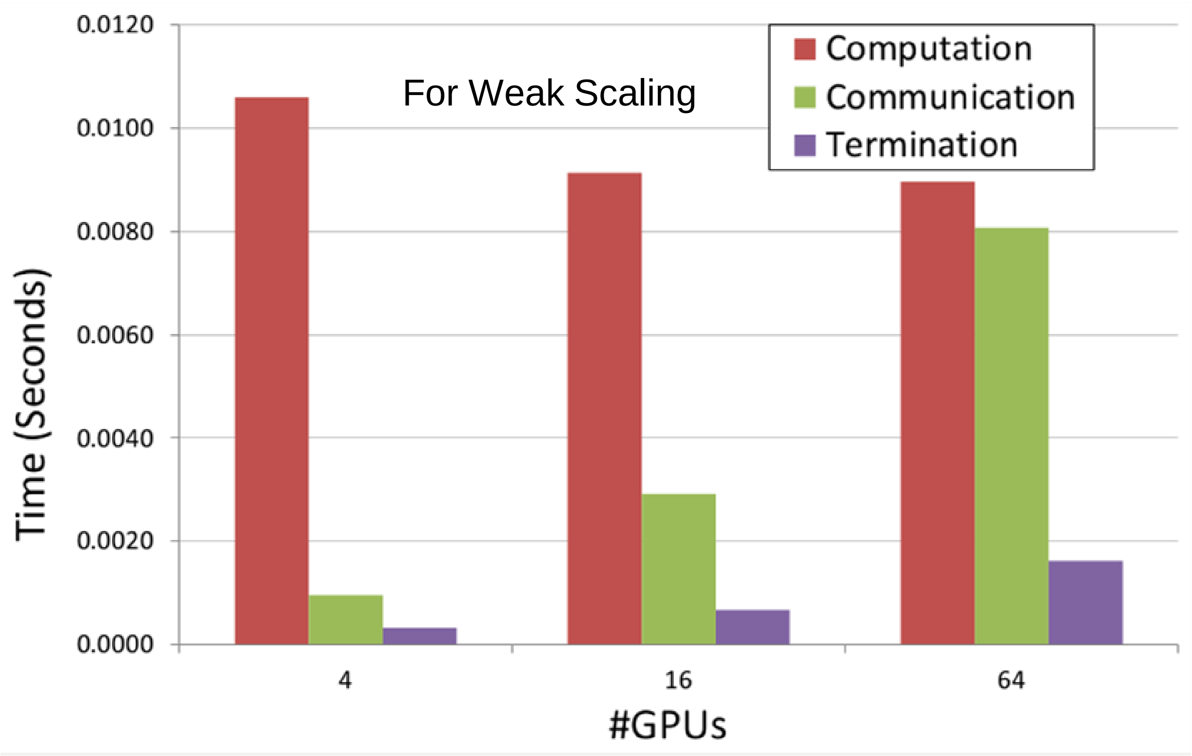


Number of Vertices in graph = 2^{SCALE}

Number of Directed Edges in graph = $32*2^{SCALE}$

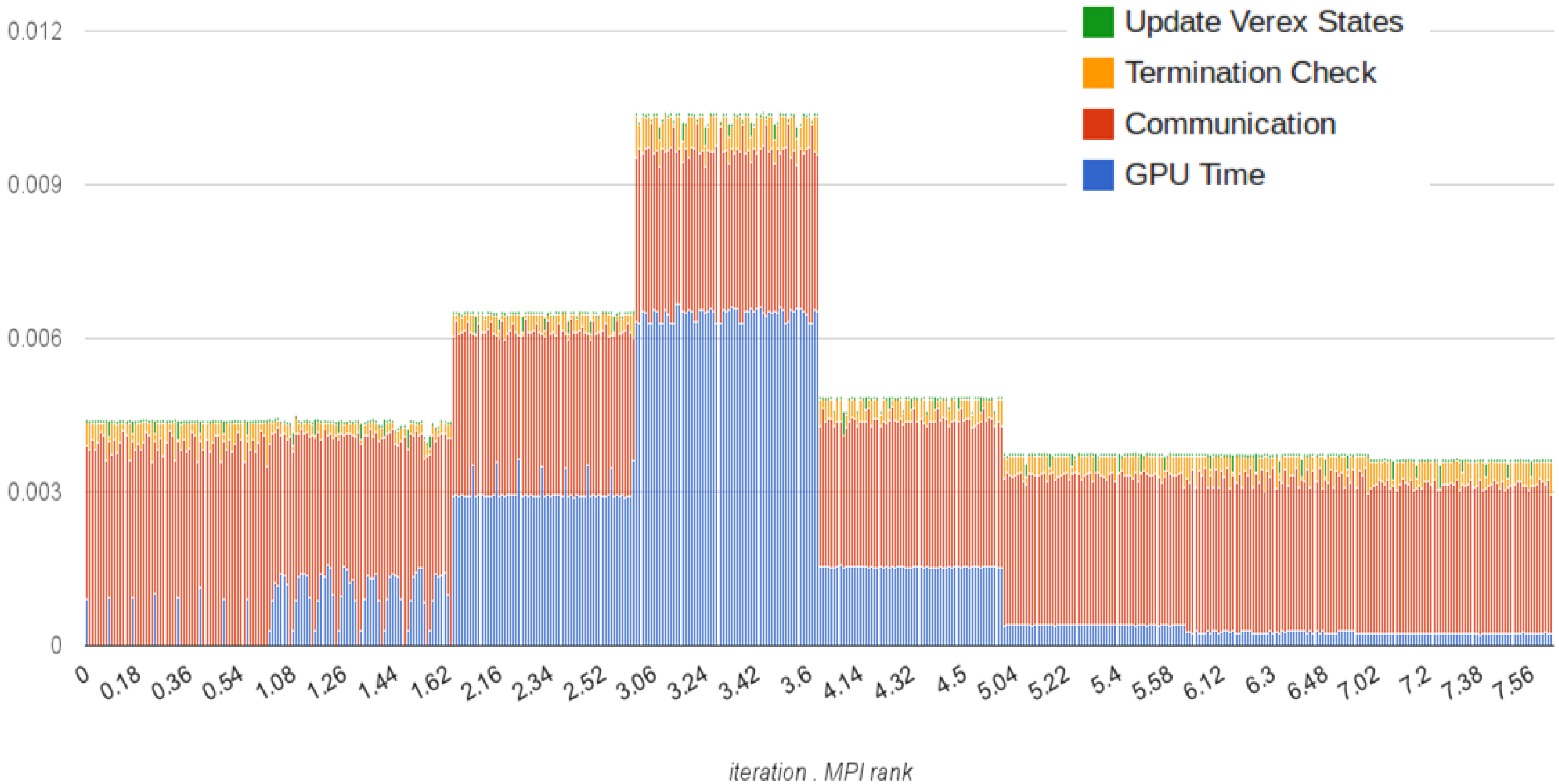
Communication vs Computation

- 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