GraphReduce: Processing Large-Scale Graphs on Accelerated-Based Systems

Shuaiwen Leon Song
Pacific Northwest National Lab

Graphs are Ubiquitous

- High Volume: Billions of edges and vertices carrying rich metadata
- High Velocity: 100s of billions photos, posts, tweets etc per month
- Fast graph analytics on large graphs

Motivation

- Why use GPUs? – GPU-based frameworks are orders of magnitude faster
- Previous GPU-based graph processing doesn’t handle datasets that doesn’t fit in GPU memory
- Several challenges in large-scale graph processing
  - How to partition the graph?
  - How and when to move the partitions between host and GPU?
  - How to best extract multi-level parallelism in GPUs?

Hybrid Programming Model

- Gather phase: each vertex aggregates values associated with its incoming edges and source vertices
- Apply phase: each vertex updates its state using the gather result
- Scatter phase: each vertex updates the state of every outgoing edge.

GraphReduce Architecture

- User defined functions: gatherMap(), gatherReduce(), apply() and scatter()
- User defined graph data types: VertexDataType and EdgeDataType

Optimizations

- Asynchronous execution and Spray (deep-copy) operation
- Dynamic frontier management
- Dynamic phase fusion and elimination

Conclusions

- GraphReduce develops a high performance graph processing framework for input datasets that may or may not fit in GPU memory
- Adopts a hybrid model of a combination of both edge- and vertex-centric implementation of GAS programming model
- Leverages CUDA streams and hardware supports like hyper-Qs to stream data in and out of GPU for high performance
- Optimizations like dynamic phase fusion/elimination and frontier management further reduces data transfer time
- Outperforms CPU-based out-of-core graph processing frameworks across a variety of real data sets achieving up to 79x speedup