Ascetic: Enhancing Cross-Iterations Data Efficiency in Out-of-Memory Graph Processing on GPUs

Ruiqi Tang, Ziyi Zhao, Kailun Wang, Jin Zhang, Xiaoli Gong*
Nankai University

Wenwen Wang
University of Georgia

Pen-Chung Yew
University of Minnesota
Out-of-GPU-Memory Processing In Graph Processing

- Limited GPU memory
- GPU high throughput
- PCIe is bottleneck

How to transfer data efficiently?

Transfer data during Graph processing
Partition-Based Graph Processing

**Common methods**
- Partition dataset

**Drawback:**
- Sparse data accesses in partitions
- No reuse
- Data thrashing

**Example:**
- Keeping Pa in GPU across iterations
- Data transfer **reduced by 26%**

**Key take-away:**
- Re-using data across iterations can cut down data transfer
Typical Memory Access Patterns

Access Pattern
• Long reuse distance
• No hot spots
• Sparse access

Conclusion
• UVM-based LRU policy not suitable
• Same reuse patterns on entire dataset
Fine-Grained Data Transfer

CPU Side

GPU Side

Transfer CSR format

Transfer subgraph each iteration

Give an opportunity!

A small portion

(1) Sparse usage per iteration
(2) Keep some memory for cross-iteration reuse
Ascetic Framework

Key Features:
• Partition GPU memory into Static Region and On-demand Region
• Static region - storing data for cross-iteration reuse
• On-demand region - storing data for intra-iteration usage
Computation overlap

Baseline

- Organize Data Map
- GPU
- CPU
- Gather Data
- Transfer Data

Ascetic

- Organize Data Map
- GPU
- CPU
- Static Computing
- Static Update
- Gather Data
- Transfer Data
- On-demand Computing

- Avoid CPU/GPU idle
- Overlapping of Static Processing and CPU Gathering
- Overlapping of Static Update and On-demand Processing
Data Replacement Mechanism

**CPU Side**
- Fragment 1: Access count: 100
- Fragment 2: Access count: 80
- Fragment 3: Access count: 65
- Fragment 4: Access count: 70
- Fragment 5: Access count: 85
- Fragment 6: Access count: 57

**GPU Side**
- Fragment 1: Access count: 1000
- Fragment 2: Access count: 900
- Fragment 3: Access count: 950

- Static Region

- Static data will become stale
- Count the accessed times in each block
- Replace stale chunks with new updated blocks

Nothing to reuse

Nothing to reuse

Nothing to reuse
Performance and Data Transfer

Comparison with state-of-the-art
- On average 2x Speedups
- Average data transfer reduces by 61%
Why UVM Does Not Work

Comparison with UVM
• On average 6.2x Speedups
• Average data transfer reduces by 73%

UVM Drawback
• Frequent data transfers via paging
• LRU policy not suitable
• High overheads in demand paging
Conclusion

• We provide a comprehensive analysis on the access patterns of graph analytic applications

• We propose Ascetic, a novel graph processing framework to exploit data reuse across iterations.

• We have implemented a prototype of Ascetic with CUDA.

• Ascetic can achieve average 2.0x speedup over a state-of-the-art graph processing approach
Q&A