CUDA Graph Algorithms at Maximum Warp

Sungpack Hong, Tayo Oguntebi, Kunle Olukotun

Graph Algorithms

Introduction: Graph Algorithms

- Many interesting problems are constructed using graphs: social networks, supply chain analysis, genealogy, ...
- However, those problems are often immense in size; even simple algorithms take significant amounts of time on such large graphs
- Parallelism is required for faster execution.
- Challenge: Interesting graph instances are usually irregularly shaped.

Conventional Approaches

- Theory researchers have concentrated on abstract PRAM (Parallel Random Access Machine) algorithms; however, real commodity machine implementations are rare.
- Clusters are not favored due to their huge communication overhead. Partitioning irregular graphs is also difficult.
- PRAM machines have been implemented as supercomputers (e.g. cray-xmt). But they are expensive and hard to access.

Previous Work: Graph Algorithms on GPU

- Implementations of PRAM algorithms on GPUs [1, 2]
- Key observation: GPU architecture is a miniature replica of a PRAM supercomputer.
- Let each thread process one parallel task (e.g. one thread per node operation)
- Exploit parallelism and higher memory bandwidth.
- Problem: Performs badly for irregular graphs
- Example: Find BFS ordering of nodes from a source node.

Our approach: Methods for efficiently addressing irregularly shaped graphs

Deferring High Fan-out

- Detect large-sized works and put those works into a queue
- Deferred works are processed in subsequent kernel calls.
- Concurrent queues can be implemented reasonably cheaply, if they either only grow or decrease during a given phase.

Warp-Based Execution

- Each warp processes a chunk of nodes, serially.
- A warps utilize its threads for SIMD operations.
- Or for very short SIMD operations only

(pros) Less imbalance
(con) Multiple kernel calls

Dynamic Workload Distribution

- Implemented in conjunction with warp-based execution
- Instantiate only as many Thread-Blocks as the number of SMs
- Dynamically allocate workloads per warp from a work queue
(pros) Prevents SMs from stalling for one long-running warp
(con) Work queue overhead

Results

Performance Comparisons

- Speedup over single-threaded CPU version
- Our baseline GPU implementation is optimized over [1] by 20%

Properties of Input Graphs

- Graph is multi-staged kernel calls
- Each stage expands frontiers by one level

Future Work

- Further study of effectiveness of warp-based approach
- On other graph algorithms
- On other applications having similar workload imbalance issue

Automatic Generation for Warp-Based Kernels

- User description ➔ SIMD kernels and calls
  (e.g. SIMD update_nbr in our example)
- C Macros or Compiler front-end

References

[1] P Hanish and P Narayanan, Accelerating Large Graph Algorithms on the GPU using CUDA [HPSC 2007]

Contact Information

E-mail: (hongsup, tayo) @ stanford.edu

Rationale: Warp-Based Execution

- Actual thread execution is not exactly PRAM
  Threads in a warp ➔ unit of SIMD
  Warps in a thread-block ➔ unit of SMT
- Therefore...explicitly utilize those traits
  Assign a chunk of jobs to each warp.
  Each warp performs jobs serially.
  Each warp uses its 32 threads to perform SIMD operations.
  Each warp can own a private SMEM partition.
  Synchronization is inherent.

GPU Memory

Thread Block

[GPU Architecture]

[Conventional Programming Model]

[GPU Architecture]

[Conventional Programming Model]