Theory and Implementation of Dynamic Data Structures for the GPU

John Owens  
UC Davis

Martín Farach-Colton  
Rutgers
NVIDIA OptiX & the BVH

The problem

- Many data structures are built on the CPU and used on the GPU
- Very few data structures can be *built* on the GPU
  - Sorted array
  - (Cuckoo) hash table
  - Several application-specific data structures (e.g., BVH tree)
- No data structures can be *updated* on the GPU
Scale of updates

- Update 1–few items
  - Fall back to serial case, slow, probably don’t care
- Update very large number of items
  - Rebuild whole data structure from scratch
- Middle ground: our goal
  - Questions: How and when?
Approach

- Pick data structures useful in serial case, try to find parallelizations?
- Pick what look like parallel-friendly data structures with parallel-friendly updates?
Log-structured merge tree

- Supports dictionary and range queries
- $\log n$ sorted levels, each level $2x$ the size of the last
- Insert into a filled level results in a merge, possibly cascaded. Operations are coarse (threads cooperate).
LSM results/questions

• Update rate of 225M elements/s
  • 13.5x faster than merging with a sorted array
• Lookups: 7.5x/1.75x slower than hash table/sorted array
• Deletes using tombstones
• *Semantics for parallel insert/delete operations?*
• *Minimum batch size?*
• *Atom size for searching?*
• *Fractional cascading?*

Quotient Filter

- Probabilistic membership queries & lookups: false positives are possible

- Comparable to a Bloom filter but also supports deletes and merges

QF results/questions

- Lookup perf. for point queries: 3.8–4.9x vs. BloomGPU
- Bulk build perf.: 2.4–2.7x vs. BloomGPU
- Insertion is significantly faster for BloomGPU
- Similar memory footprint
- 3 novel implementations of bulk build + 1 of insert
- *Bulk build == non-associative scan*
- *Limited to byte granularity*

Cross-cutting issues

- Useful models for GPU memory hierarchy
- Independent threads vs. cooperative threads?
  - More broadly, what’s the right work granularity?
- Memory allocation (& impact on hardware)
- Cleanup operations, and programming model implications
- Integration into higher-level programming environments
- Use cases! Chicken & egg problem