GPU Sparse Graph Traversal

Duane Merrill
Breadth-first search of graphs (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source
   - Or label every vertex by its predecessor in this traversal order
The punchline

“Conventional wisdom”
- GPUs are poorly suited for dynamic problems like BFS
  - Dynamic work distribution seems hard to implement
  - Requires cooperation

We show GPUs provide exceptional performance
- Billions of traversed-edges / second / socket
  - vs. hundreds-of-millions for multi-core (SC'10, SPAA'10)
- For diverse synthetic and real-world datasets
  - Small-diameter (e.g., RMAT and social networks)
  - Large-diameter (e.g., road-network & spatial lattices)
BFS applications

- A common algorithmic building block
  - Tracking reachable heap during garbage collection
  - Belief propagation in statistical inference
  - Finding community structure in networks
  - Path-finding

- Core benchmark kernel
  - Graph 500
  - Rodinia
  - Parboil

- Simple performance-analog for many applications
  - Pointer-chasing
  - Work queues
Level-synchronous BFS parallelization strategies
Quadratic-work approach

1. Inspect every vertex at every timestep
2. If updated, update its neighbors
3. Repeat

Details
- Quadratic $O(n^2 + m)$ work
- Trivial data-parallel stencils
  - One thread per vertex
Linear-work approach

1. Expand neighbors of unexplored nodes
   - Expand: vertex frontier $\rightarrow$ edge-frontier

2. Filter neighbors (previously-visited & duplicates)
   - Contract: edge-frontier $\rightarrow$ vertex frontier

3. Repeat

Details
- Linear $O(n+m)$ work
- Need cooperative buffers for tracking frontiers
The problem with quadratic approach

- Too much work!!!

3D Poisson Lattice
(300³ = 27M vertices)

Referenced nodes & edges (millions)

BFS iteration

130x work
Goal: Demonstrate $O(m+n)$ traversal on diverse datasets
(Not a one-trick pony…)

Wikipedia
(social)

3D Poisson grid
(cubic lattice)

R-MAT
(random, power-law, small-world)

Europe road atlas
(Euclidian space)

PDE-constrained optimization
(non-linear KKT)

Auto transmission manifold
(tetrahedral mesh)
Challenges
Linear-work challenges for parallelization

General challenges

1. Load imbalance between processing elements
   • Workstealing queues
2. Bandwidth inefficiency
   • Use “status bitmask” when filtering already-visited nodes

GPU-specific challenges

1. Cooperative placement within global queues
2. Poor load balancing within SIMD lanes
3. Poor locality within SIMD lanes
4. Simultaneous discovery (benign race condition)
(1) Cooperative placement within global queues

**Problem:**
- Need shared work “queues”
  - Threads must *cooperatively* determine enqueue offsets
- GPU rate-limits (C2050):
  - Can copy 16B vertex-identifiers / s
  - Only 67M global-atomics / s (238x slowdown)
  - Only 600M smem-atomics / s (27x slowdown)

**Solution:**
- Compute local offsets using CTA-wide prefix-sum
- Compute CTA offset using a single coarse-grained atomic-add

*Dynamic expansion*
Prefix sum for allocation

Each output is computed to be the sum of the previous inputs
- $O(n)$ work
- Use results as a scatter offsets

Fits the GPU machine model well
- Proceed at copy-bandwidth
- Only ~8 thread-instructions per input

(2) Poor load balancing within SIMD lanes

Problem:
- Large variance in adjacency list sizes
  - E.g., power-law degree distributions
- Exacerbated by wide SIMD widths

Solution:
- Cooperative neighbor expansion
- Enlist nearby threads to help process each adjacency list in parallel

(a) \textbf{Bad}: Serial expansion & processing

(b) \textbf{Slightly better}: Coarse warp-centric parallel expansion

(c) \textbf{Best}: Fine-grained parallel expansion (packed by prefix sum)
(3) Poor locality within SIMD lanes

Problem:
- The referenced adjacency lists are arbitrarily located
- Exacerbated by wide SIMD widths and DRAM transaction sizes
- Can’t afford to have SIMD threads streaming through unrelated data

Solution:
- Cooperative neighbor expansion

(a) Bad: Serial expansion & processing

(b) Slightly better: Coarse warp-centric parallel expansion

(c) Best: Fine-grained parallel expansion (packed by prefix sum)
(4) SIMD simultaneous discovery

- “Duplicates” in edge-frontier yield redundant work
  - Exacerbated by wide SIMD
  - Compounded every iteration

\[ \text{Iteration 1} \quad \text{Iteration 2} \quad \text{Iteration 3} \quad \text{Iteration 4} \]
Simultaneous discovery

- Normally not a problem for
  - CPU implementations
    - Low hardware parallelism
  - Serial adjacency list inspection

- A big problem for cooperative SIMD expansion
  - Spatially-descriptive datasets
  - Power-law datasets

**Solution:**
- Collision hashes in local scratch
  1. Hash per warp (instantaneous coverage)
  2. Hash per CTA (recent history coverage)
Absolute and relative performance
<table>
<thead>
<tr>
<th>Graph</th>
<th>Spy Plot</th>
<th>Average Search Depth</th>
<th>Sequential Sandybridge†</th>
<th>Parallel Nehalem</th>
<th>Parallel speedup</th>
<th>Tesla C2050 Billion TE/s</th>
<th>Parallel speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>europe.osm</td>
<td><img src="image1" alt="Spy Plot" /></td>
<td>19314</td>
<td>0.03</td>
<td></td>
<td></td>
<td>0.3</td>
<td>11 x</td>
</tr>
<tr>
<td>grid5pt.5000</td>
<td><img src="image2" alt="Spy Plot" /></td>
<td>7500</td>
<td>0.08</td>
<td></td>
<td></td>
<td>0.6</td>
<td>7.3 x</td>
</tr>
<tr>
<td>hugebubbles</td>
<td><img src="image3" alt="Spy Plot" /></td>
<td>6151</td>
<td>0.03</td>
<td></td>
<td></td>
<td>0.4</td>
<td>15 x</td>
</tr>
<tr>
<td>grid7pt.300</td>
<td><img src="image4" alt="Spy Plot" /></td>
<td>679</td>
<td>0.04</td>
<td>0.12 (4-core††)</td>
<td>3.0 x</td>
<td>1.1</td>
<td>28 x</td>
</tr>
<tr>
<td>nlppkt160</td>
<td><img src="image5" alt="Spy Plot" /></td>
<td>142</td>
<td>0.26</td>
<td>0.47 (4-core††)</td>
<td>1.8 x</td>
<td>2.5</td>
<td>10 x</td>
</tr>
<tr>
<td>audikwi</td>
<td><img src="image6" alt="Spy Plot" /></td>
<td>62</td>
<td>0.65</td>
<td>0.23 (4-core††)</td>
<td>1.8 x</td>
<td>2.2</td>
<td>18 x</td>
</tr>
<tr>
<td>cage15</td>
<td><img src="image7" alt="Spy Plot" /></td>
<td>37</td>
<td>0.13</td>
<td>0.11 (4-core††)</td>
<td>2.2 x</td>
<td>1.1</td>
<td>23 x</td>
</tr>
<tr>
<td>kkt_power</td>
<td><img src="image8" alt="Spy Plot" /></td>
<td>37</td>
<td>0.05</td>
<td></td>
<td></td>
<td>3.0</td>
<td>5.9 x</td>
</tr>
<tr>
<td>coPapersCiteeseer</td>
<td><img src="image9" alt="Spy Plot" /></td>
<td>26</td>
<td>0.50</td>
<td></td>
<td></td>
<td>1.6</td>
<td>25 x</td>
</tr>
<tr>
<td>wikipedia-2007</td>
<td><img src="image10" alt="Spy Plot" /></td>
<td>20</td>
<td>0.07</td>
<td>0.19 (4-core††)</td>
<td>2.7 x</td>
<td>3.1</td>
<td>13 x</td>
</tr>
<tr>
<td>kron_g500-logn20</td>
<td><img src="image11" alt="Spy Plot" /></td>
<td>6</td>
<td>0.24</td>
<td></td>
<td></td>
<td>3.0</td>
<td>29 x</td>
</tr>
<tr>
<td>random.2Mv.128Me</td>
<td><img src="image12" alt="Spy Plot" /></td>
<td>6</td>
<td>0.10</td>
<td>0.50 (8-core†††)</td>
<td>5.0 x</td>
<td>3.0</td>
<td>29 x</td>
</tr>
<tr>
<td>rmat.2Mv.128Me</td>
<td><img src="image13" alt="Spy Plot" /></td>
<td>6</td>
<td>0.15</td>
<td>0.70 (8-core†††)</td>
<td>4.6 x</td>
<td>3.3</td>
<td>22 x</td>
</tr>
</tbody>
</table>

**Harmonic mean** (across all / common datasets) 12x / 20x

---

Duane Merrill, University of Virginia

†3.4GHz Core i7 2600K

††2.5 GHz Core i7 4-core, Leiserson et al.

†††2.7 GHz EX Xeon X5570 8-core, Agarwal et al.
Summary

BFS conclusions

- Quadratic-work approaches are uncompetitive
  - Need linear-work algorithm
- Dynamic workload management:
  - Prefix sum instead of atomic read-modify-write ops
  - Fine-grained expansion and contraction for high utilization

Overall conclusions

- GPUs can be very amenable to dynamic, cooperative problems
  - BFS is actually well-suited to GPU strengths
    - Exposes lots of concurrency
    - Is memory-bound

Questions?

Arbitrary locality

3D (7-point) Poisson Lattice
Adjacency lists reference nearby vertices

Wikipedia '07
Adjacency lists do not reference nearby vertices

Sparsity pattern (adjacency matrix)
## Experimental corpus

<table>
<thead>
<tr>
<th>Graph</th>
<th>Spy Plot</th>
<th>Description</th>
<th>Average Search Depth</th>
<th>Vertices (millions)</th>
<th>Edges (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>europe.osm</td>
<td></td>
<td>Road network</td>
<td>19314</td>
<td>50.9</td>
<td>108.1</td>
</tr>
<tr>
<td>grid5pt.5000</td>
<td></td>
<td>2D Poisson stencil</td>
<td>7500</td>
<td>25.0</td>
<td>125.0</td>
</tr>
<tr>
<td>hugebubbles-00020</td>
<td></td>
<td>2D mesh</td>
<td>6151</td>
<td>21.2</td>
<td>63.6</td>
</tr>
<tr>
<td>grid7pt.300</td>
<td></td>
<td>3D Poisson stencil</td>
<td>679</td>
<td>27.0</td>
<td>188.5</td>
</tr>
<tr>
<td>nlpkkt160</td>
<td></td>
<td>Constrained optimization problem</td>
<td>142</td>
<td>8.3</td>
<td>221.2</td>
</tr>
<tr>
<td>audikw1</td>
<td></td>
<td>Finite element matrix</td>
<td>62</td>
<td>0.9</td>
<td>76.7</td>
</tr>
<tr>
<td>cage15</td>
<td></td>
<td>Transition prob. matrix</td>
<td>37</td>
<td>5.2</td>
<td>94.0</td>
</tr>
<tr>
<td>kkt_power</td>
<td></td>
<td>Optimization (KKT)</td>
<td>37</td>
<td>2.1</td>
<td>13.0</td>
</tr>
<tr>
<td>coPapersCiteseer</td>
<td></td>
<td>Citation network</td>
<td>26</td>
<td>0.4</td>
<td>32.1</td>
</tr>
<tr>
<td>wikipedia-20070206</td>
<td></td>
<td>Wikipedia page links</td>
<td>20</td>
<td>3.6</td>
<td>45.0</td>
</tr>
<tr>
<td>kron_g500-logn20</td>
<td></td>
<td>Graph500 random graph</td>
<td>6</td>
<td>1.0</td>
<td>100.7</td>
</tr>
<tr>
<td>random.2Mv.128Me</td>
<td></td>
<td>Uniform random graph</td>
<td>6</td>
<td>2.0</td>
<td>128.0</td>
</tr>
<tr>
<td>rmat.2Mv.128Me</td>
<td></td>
<td>RMAT random graph</td>
<td>6</td>
<td>2.0</td>
<td>128.0</td>
</tr>
</tbody>
</table>
Compressed sparse row (CSR) representation

"Logical" adjacency matrix

Column indices: $O(m)$

Row offsets: $O(n)$
Coupling of expansion & contraction phases

Alternatives:

a) **Expand-contract (one kernel)**

- Maintain vertex-frontier (unexplored nodes) in DRAM between BFS iterations
  - $O(2n)$ bytes
- Produce and consume edge-frontier online, in-core

b) **Contract-expand (one kernel)**

- Maintain edge-frontier (untraversed edges) in DRAM between BFS iterations
  - $O(2m)$ bytes
- Produce and consume vertex frontier online, in-core

c) **Two-phase (two kernels)**

- Wholly produce both edge and vertex-frontiers in DRAM
  - $O(m+n)$ bytes
Coupling of expansion & contraction phases

Alternatives:

a) **Expand-contract (vertex-frontier in DRAM)**
   - Suitable for all types of BFS iterations

b) **Contract-expand (edge-frontier in DRAM)**
   - Even better for small, fleeting BFS iterations

c) **Two-phase (both frontiers in DRAM)**
   - Even better for large, saturating BFS iterations (surprising!!)
   - The two “fused” strategies (a & b above) suffer the interaction between:
     - TLB misses during expansion (neighbor gathering)
     - Uncoalesced reads during contraction (status lookup)
Comparison of expansion techniques

![Comparison of expansion techniques graph]

- Serial
- Warp
- Scan
- Scan+Warp
- Scan+Warp+CTA

Normalized hmean
Multi-GPU traversal

Expand neighbors $\rightarrow$ sort by GPU (with filter) $\rightarrow$ read from peer GPUs (with filter) $\rightarrow$ repeat

![Graph showing performance comparison for different datasets and configurations.](image-url)
Comparison of coupling approaches

![Graph showing comparison of coupling approaches between Expand-Contract, Contract-Expand, 2-Phase, and Hybrid methods.](image)

- **x-axis**: 10^9 edges / sec
- **y-axis**: Normalized

The graph compares the performance of different coupling approaches across various datasets, indicating how each method handles data flow and maintenance in a distributed system context.
Local duplicate culling

- Contraction uses local collision hashes in smem scratch space:
  1. Hash per warp (instantaneous coverage)
  2. Hash per CTA (recent history coverage)

- Redundant work reduced < 10% in all datasets
- No atomics needed
CTA local prefix sum
Filter “duplicates” within edge-frontier

- Implement local collision hashes in smem scratch space during contraction
  1. Hash per warp (instantaneous coverage)
  2. Hash per CTA (recent history coverage)
- Redundant work reduced < 10% in all datasets
- No atomics needed