GPU Sparse Graph Traversal

Duane Merrill (NVIDIA)
Michael Garland (NVIDIA)
Andrew Grimshaw (Univ. of Virginia)
Breadth-first search (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source
   (or by predecessor vertex)
BFS applications

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

- Core benchmark kernel
  - Graph 500
  - Rodinia
  - Parboil

- Simple performance-analog for many applications
  - Pointer-chasing
  - Work queues

3D lattice

Wikipedia '07

“Conventional” wisdom

- GPUs would be poorly suited for BFS
  - Not pleasingly data-parallel (if you want to solve it efficiently)
    - Insufficient atomic throughput for cooperation
“Conventional” wisdom

- GPUs would be poorly suited for BFS
  - Not pleasingly data-parallel (if you want to solve it efficiently)
    - Insufficient atomic throughput for cooperation
  - Issues with scaling to 10,000s of processing elements
    - Workload imbalance, SIMD underutilization
    - Not enough parallelism in all graphs
The punchline

- We show GPUs provide exceptional performance…
  - Billions of traversed-edges per second per socket
  - For diverse synthetic and real-world datasets
  - Linear-work algorithm

- …using efficient parallel prefix sum
  - For cooperative data movement

---

![Graph showing traversal performance](image)

- Billions of traversed edges per second per socket
- Average traversal depth

---

Agarwal et al. (SC’10) & Leiserson et al. (SPAA’10)
Level-synchronous parallelization strategies
(a) Quadratic-work approach

“Typical” GPU BFS (e.g., Harish et al., Deng et al. Hong et al.)

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

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

1. Expand neighbors
   (vertex frontier $\rightarrow$ edge-frontier)

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

3. Repeat

- Linear $O(n+m)$ work
- Need cooperative buffers for tracking frontiers
Quadratic vs. linear
Quadratic is too much work!!

3D Poisson Lattice
($300^3 = 27$M vertices)

2.5x–2,300x speedup vs. quadratic GPU methods

Goal: GPU competence 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

Generic challenges

1. Load imbalance between cores
   - Coarse workstealing queues
2. Bandwidth inefficiency
   - Use “status bitmask” when filtering already-visited nodes

GPU-specific challenges

1. Cooperative allocation (global queues)
2. Load imbalance within SIMD lanes
3. Poor locality within SIMD lanes
4. Simultaneous discovery (i.e., the benign race condition)
Linear-work challenges for parallelization

Generic challenges

1. Load imbalance between cores
   - Coarse workstealing queues
2. Bandwidth inefficiency
   - Use “status bitmask” when filtering already-visited nodes

GPU-specific challenges

1. Cooperative allocation (global queues)
2. Load imbalance within SIMD lanes
3. Poor locality within SIMD lanes
4. Simultaneous discovery (i.e., the benign race condition)
(1) Cooperative data movement (queuing)

**Problem:**
- Need shared work “queues”
- GPU rate-limits (C2050):
  - 16 billion vertex-identifiers / sec (124 GB/s)
  - Only 67M global atomics / sec (238x slowdown)
  - Only 600M smem atomics / sec (27x slowdown)

**Solution:**
- Compute enqueue offsets using prefix sum

![Graph showing transfer rate vs. vertex stride between atomic operations.](image-url)
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
Prefix sum for edge expansion

**Adjacency list size**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**Expanded edge frontier**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Efficient local scan
(granularity coarsening)
(2) Poor load balancing within SIMD lanes

Problem:
- Large variance in adjacency list sizes
  - Exacerbated by wide SIMD widths

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

a) Bad: Serial expansion & processing

b) Slightly better: Coarse warp-centric parallel expansion

c) Best: Fine-grained parallel expansion (packed by prefix sum)
(2) Poor load balancing within SIMD lanes

- **Problem:**
  - Large variance in adjacency list sizes
    - Exacerbated by wide SIMD widths

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

**Examples:**

- **a) Bad:** Serial expansion & processing
  - ![Bad example diagram]

- **b) Slightly better:** Coarse warp-centric parallel expansion
  - ![Slightly better example diagram]

- **c) Best:** Fine-grained parallel expansion (packed by prefix sum)
  - ![Best example diagram]
(3) Poor locality within SIMD lanes

Problem:
- The referenced adjacency lists are arbitrarily located
  - Exacerbated by wide SIMD widths
- 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 the edge-frontier $\rightarrow$ redundant work
  - Exacerbated by wide SIMD
  - Compounded every iteration
Simultaneous discovery

- Normally not a problem for
  - CPU implementations
  - Serial adjacency list inspection
- A **big** problem for cooperative SIMD expansion
  - Spatially-descriptive datasets
  - Power-law datasets
- Solution:
  - Localized duplicate removal using hashes in local scratch

![Graph](image)
Absolute and relative performance
## Single-socket comparison

<table>
<thead>
<tr>
<th>Graph</th>
<th>Spy Plot</th>
<th>Average Search Depth</th>
<th>Nehalem</th>
<th>Tesla C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Billion TE/s</td>
<td>Parallel speedup</td>
</tr>
<tr>
<td>europe.osm</td>
<td><img src="image" alt="Europe OSM" /></td>
<td>19314</td>
<td>0.12 (4-core†)</td>
<td>2.4 x</td>
</tr>
<tr>
<td>grid5pt.5000</td>
<td><img src="image" alt="Grid 5pt.5000" /></td>
<td>7500</td>
<td>0.47 (4-core†)</td>
<td>3.0 x</td>
</tr>
<tr>
<td>hugebubbles</td>
<td><img src="image" alt="Hugebubbles" /></td>
<td>6151</td>
<td>0.23 (4-core†)</td>
<td>2.6 x</td>
</tr>
<tr>
<td>grid7pt.300</td>
<td><img src="image" alt="Grid 7pt.300" /></td>
<td>679</td>
<td>0.11 (4-core†)</td>
<td>3.0 x</td>
</tr>
<tr>
<td>nlpkkt160</td>
<td><img src="image" alt="Nlpkt160" /></td>
<td>142</td>
<td>0.19 (4-core†)</td>
<td>3.2 x</td>
</tr>
<tr>
<td>audikw1</td>
<td><img src="image" alt="Audikw1" /></td>
<td>62</td>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td>cage15</td>
<td><img src="image" alt="Cage15" /></td>
<td>37</td>
<td></td>
<td>2.2</td>
</tr>
<tr>
<td>kkt_power</td>
<td><img src="image" alt="Kkt_power" /></td>
<td>37</td>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td>coPapersCiteseer</td>
<td><img src="image" alt="Copapersciteseer" /></td>
<td>26</td>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td>wikipedia-2007</td>
<td><img src="image" alt="Wikipedia-2007" /></td>
<td>20</td>
<td>0.50 (8-core††)</td>
<td>7.0 x</td>
</tr>
<tr>
<td>kron_g500-logn20</td>
<td><img src="image" alt="Kron_g500-logn20" /></td>
<td>6</td>
<td></td>
<td>3.1</td>
</tr>
<tr>
<td>random.2Mv.128Me</td>
<td><img src="image" alt="Random.2Mv.128Me" /></td>
<td>6</td>
<td>0.70 (8-core††)</td>
<td>6.0 x</td>
</tr>
<tr>
<td>rmat.2Mv.128Me</td>
<td><img src="image" alt="Rmat.2Mv.128Me" /></td>
<td>6</td>
<td></td>
<td>3.3</td>
</tr>
</tbody>
</table>

†2.5 GHz Core i7 4-core (Leiserson et al.) ††2.7 GHz EX Xeon X5570 8-core (Agarwal et al.) †††vs 3.4GHz Core i7 2600K (Sandybridge)
# Single-socket comparison

<table>
<thead>
<tr>
<th>Graph</th>
<th>Spy Plot</th>
<th>Average Search Depth</th>
<th>Nehalem</th>
<th>Parallel speedup</th>
<th>Tesla C2050</th>
<th>Parallel speedup†††</th>
</tr>
</thead>
<tbody>
<tr>
<td>europe.osm</td>
<td><img src="europe.osm" alt="Image" /></td>
<td>19314</td>
<td>0.3</td>
<td>11 x</td>
<td>0.3</td>
<td>11 x</td>
</tr>
<tr>
<td>grid5pt.5000</td>
<td><img src="grid5pt.5000" alt="Image" /></td>
<td>7500</td>
<td>0.6</td>
<td>7.3 x</td>
<td>0.6</td>
<td>7.3 x</td>
</tr>
<tr>
<td>hugebubbles</td>
<td><img src="hugebubbles" alt="Image" /></td>
<td>6151</td>
<td>0.4</td>
<td>15 x</td>
<td>0.4</td>
<td>15 x</td>
</tr>
<tr>
<td>grid7pt.300</td>
<td><img src="grid7pt.300" alt="Image" /></td>
<td>679</td>
<td>0.12 (4-core)</td>
<td>2.4 x</td>
<td>1.1</td>
<td>28 x</td>
</tr>
<tr>
<td>nlpkkt160</td>
<td><img src="nlpkkt160" alt="Image" /></td>
<td>142</td>
<td>0.47 (4-core)</td>
<td>3.0 x</td>
<td>2.5</td>
<td>10 x</td>
</tr>
<tr>
<td>audikw1</td>
<td><img src="audikw1" alt="Image" /></td>
<td>62</td>
<td>0.23 (4-core)</td>
<td>2.6 x</td>
<td>3.0</td>
<td>4.6 x</td>
</tr>
<tr>
<td>cage15</td>
<td><img src="cage15" alt="Image" /></td>
<td>37</td>
<td>0.11 (4-core)</td>
<td>3.0 x</td>
<td>1.1</td>
<td>23 x</td>
</tr>
<tr>
<td>kkt_power</td>
<td><img src="kkt_power" alt="Image" /></td>
<td>37</td>
<td>0.19 (4-core)</td>
<td>3.2 x</td>
<td>3.0</td>
<td>5.9 x</td>
</tr>
<tr>
<td>coPapersCiteseer</td>
<td><img src="coPapersCiteseer" alt="Image" /></td>
<td>26</td>
<td>0.5</td>
<td>13 x</td>
<td>3.1</td>
<td>13 x</td>
</tr>
<tr>
<td>wikipedia-2007</td>
<td><img src="wikipedia-2007" alt="Image" /></td>
<td>20</td>
<td>0.19 (4-core)</td>
<td>3.2 x</td>
<td>1.6</td>
<td>25 x</td>
</tr>
<tr>
<td>kron_g500-logn20</td>
<td><img src="kron_g500-logn20" alt="Image" /></td>
<td>6</td>
<td>0.50 (8-core††)</td>
<td>7.0 x</td>
<td>3.0</td>
<td>29 x</td>
</tr>
<tr>
<td>random.2Mv.128Me</td>
<td><img src="random.2Mv.128Me" alt="Image" /></td>
<td>6</td>
<td>0.70 (8-core††)</td>
<td>6.0 x</td>
<td>3.3</td>
<td>22 x</td>
</tr>
<tr>
<td>rmat.2Mv.128Me</td>
<td><img src="rmat.2Mv.128Me" alt="Image" /></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

†2.5 GHz Core i7 4-core (Leiserson et al.) ††2.7 GHz EX Xeon X5570 8-core (Agarwal et al) ††† vs 3.4GHz Core i7 2600K (Sandybridge)
Summary

- Quadratic-work approaches are uncompetitive

- Dynamic workload management:
  - Prefix sum (vs. atomic-add)
  - Utilization requires fine-grained expansion and contraction

- GPUs can be very amenable to dynamic, cooperative problems
Questions?
# 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>
Comparison of expansion techniques

![Bar chart showing comparison of expansion techniques. The x-axis represents different datasets, and the y-axis represents the number of edges per second (in 10^8). The chart compares Serial, Warp, Scan, Scan+Warp, and Scan+Warp+CTA methods. Each dataset is normalized to the highest value.]
Coupling of expansion & contraction phases

Alternatives:

- **Expand-contract**
  - Wholly realize vertex-frontier in DRAM
  - Suitable for all types of BFS iterations

- **Contract-expand**
  - Wholly realize edge-frontier in DRAM
  - Even better for small, fleeting BFS iterations

- **Two-phase**
  - Wholly realize both frontiers in DRAM
  - Even better for large, saturating BFS iterations (surprising!!)
Comparison of coupling approaches

- Expand-Contract
- Contract-Expand
- 2-Phase
- Hybrid

![Comparison of coupling approaches](image_url)
Local duplicate culling

- Contraction uses local collision hashes in smem scratch space:
  - Hash per warp (instantaneous coverage)
  - Hash per CTA (recent history coverage)
- Redundant work < 10% in all datasets
- No atomics needed
Multi-GPU traversal

Expand neighbors → sort by GPU (with filter) → read from peer GPUs (with filter) → repeat

![Graph showing traversal performance for different graphs and GPU configurations.](image-url)