Grazelle
Hardware-Optimized In-Memory Graph Processing

Samuel Grossman, Heiner Litz, and Christos Kozyrakis
Existing Work

Properties of Graph Problems

• Irregular graph data

• Difficult to partition

• Unpredictable access pattern

Scalability Optimizations

• Partitioning algorithms

• Dynamic scheduling, load balancing

• Sharing and synchronization optimizations
Existing Work

Properties of Graph Problems

• Irregular graph data
• Difficult to partition
• Unpredictable access pattern

Modern Hardware Features

✖ Vector processing units
✖ Sequential memory accesses
✖ Prefetchers
✖ NUMA
Grazelle

Properties of Graph Problems

• Irregular graph data

✓ Simple and easy to partition

✓ Predictable access pattern

Modern Hardware Features

✓ Vector processing units

✓ Sequential memory accesses

✓ Prefetchers

✓ NUMA
Grazelle

Grazelle is a **single-machine, in-memory** Gather-Apply-Scatter (GAS) graph processing engine that:

- Leverages modern hardware features
- Improves throughput by 4.4× to 36.2× over existing work

Grazelle is **not** a complete graph analytics framework.
Top-Level Execution Flow

1. Gather Phase
   *GAS Gather*

2. Combine Phase
   *GAS Apply, Scatter*

Finish
Key Design Principles

• Vector-optimized data structures with minimal indirection

• Thread-private memory writes

• Mostly sequential memory accesses

• Simple, static partitioning and scheduling

• Synchronization via thread barriers between phases
Gather: Topology Data Structures

Existing Work
• “Compressed Sparse Row”

Grazelle
• Vector-encoded edge list
Gather: Topology Data Structures

256 bits, 4 elements

Valid
Part of Destination Vertex ID
Source Vertex ID
Gather: Execution

Edges

✓ Private, read-only

Vector Load

Vertex

✓ Shared, read-only

Vector Gather

Accumulators

✓ Private, write-only

Scalar Store
Combine: Execution

Accumulator

- Private, read-only

Vertices

- Private, write-only

Vector Load

Vector Store
NUMA Partitioning

Edges always NUMA-local

Accumulators always NUMA-local
Vertices sometimes NUMA-remote
Evaluation

**Processor:** 4× Intel Xeon E7-4850 (14 cores, 2-way SMT, 35 MB LLC)

**RAM:** 1 TB total, 256 GB per socket

**Storage:** 12× 6 TB magnetic disks, RAID-10

**OS:** Ubuntu 14.04 LTS

**Compiler:** GCC 4.8
## Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviation</th>
<th>Vertices</th>
<th>Edges</th>
<th>Size</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>cit-Patents</td>
<td>C</td>
<td>3.7 M</td>
<td>16.5 M</td>
<td>250 MB</td>
<td>Citations web</td>
</tr>
<tr>
<td>dimacs-usa</td>
<td>D</td>
<td>23.9 M</td>
<td>58.3 M</td>
<td>900 MB</td>
<td>Road network</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>T</td>
<td>41.7 M</td>
<td>1.47 B</td>
<td>20 GB</td>
<td>Social</td>
</tr>
<tr>
<td>uk-2007</td>
<td>U</td>
<td>105.9 M</td>
<td>3.74 B</td>
<td>60 GB</td>
<td>Internet</td>
</tr>
<tr>
<td>(skewed synthetic)</td>
<td></td>
<td>≤ 134 M</td>
<td>≤ 17 B</td>
<td>≤ 250 GB</td>
<td></td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>X-Stream</th>
<th>Polymer</th>
<th>Grazelle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector processing units</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Sequential memory accesses</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Prefetching</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NUMA awareness</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Caching overheads</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Simultaneous multithreading</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Comparison: Throughput (Real Graphs)

1 Socket

- X-Stream
- Polymer
- Grazelle

4 Sockets

- X-Stream
- Polymer
- Grazelle
Comparison: Throughput (Synthetic Graphs)

1 Socket

![Graph 1 Socket]

4 Sockets

![Graph 4 Sockets]
Memory Bandwidth Utilization

- **cit-Patents**
  - Read (Gather): 20 GB/sec
  - Write (Gather): 5 GB/sec
  - Read (Combine): 15 GB/sec
  - Write (Combine): 5 GB/sec

- **dimacs-usa**
  - Read (Gather): 15 GB/sec
  - Write (Gather): 5 GB/sec
  - Read (Combine): 10 GB/sec
  - Write (Combine): 5 GB/sec

- **twitter-2010**
  - Read (Gather): 20 GB/sec
  - Write (Gather): 5 GB/sec
  - Read (Combine): 15 GB/sec
  - Write (Combine): 5 GB/sec

- **uk-2007**
  - Read (Gather): 20 GB/sec
  - Write (Gather): 5 GB/sec
  - Read (Combine): 15 GB/sec
  - Write (Combine): 5 GB/sec
Edge Vector Packing Efficiency

![Graph showing the average packing efficiency for different numbers of elements (4, 8, 16) against varying average degrees (1 to 4096).]
Load Balance Effectiveness

Time Division: Work vs. Barrier

- Work
- Barrier

30% off ideal

L2 Stall Cycles

Stall

% L2 Stall Cycles

Threads
Conclusion

• Grazelle maps graph problems to a *regular and predictable* software implementation without sacrificing scalability or balance

• Grazelle effectively leverages modern hardware and significantly outperforms the state-of-the-art

• Future work:
  • Expand to secondary storage devices like flash
  • Build higher-level optimizations on top of Grazelle