Making Pull-Based Graph Processing Performant

Samuel Grossman\textsuperscript{1}, Heiner Litz\textsuperscript{2}, and Christos Kozyrakis\textsuperscript{1}

\textsuperscript{1}Stanford University \quad \textsuperscript{2}University of California, Santa Cruz

PPoPP 2018 · February 27, 2018
Graph Processing

• Problem modelled as **objects** (vertices) and **connections between them** (edges)

• Examples:
  • Internet (pages and hyperlinks)
  • Social network (people and friendships)
  • Roads and intersections
  • Products and ratings
Graph Processing
Graph Processing

Repeat until convergence
Graph Processing

Push

Group by source vertex

Pull

Group by destination vertex

Hybrid: dynamically select push or pull for each iteration
Graph Processing

```plaintext
foreach vertex v in graph.vertices
  foreach edge e in v.(in|out)edges
    // process the edge
    ...
```
Parallelizing Graph Processing

• Outer loop parallelization
  • Between cores: assign *entire vertices* to threads

• Inner loop parallelization
  • Between cores: subdivide the edges within each vertex
  • Within one core: vectorize the loop
Parallelizing Graph Processing

Running Ligra on twitter-2010 graph
Parallelizing Graph Processing

Running Ligra on *twitter-2010* graph
Pull’s Performance Challenge

<table>
<thead>
<tr>
<th>Serial Inner Loop</th>
<th>Parallel Inner Loop</th>
</tr>
</thead>
</table>

**Contribution #1: “Scheduler Awareness”**

A technique that can be applied to the inner loop of a pull engine to parallelize it without introducing conflicts.

- One thread per vertex
- Updates are thread-private
- Multiple threads per vertex
- Updates will conflict
Pull’s Performance Opportunity

• Further gains possible using SIMD vectorization
  • Improve parallelism of the computation
  • Improve memory bandwidth utilization

Contribution #2: “Vector-Sparse”

A low-level modification to a data structure commonly used to represent graphs, intended to enhance vectorization.
Grazelle

• A hybrid graph processing framework that embodies both of our contributions

• Outperforms the state-of-the-art by over 10× in some cases

• Available for download at https://github.com/stanford-mast/Grazelle-PPoPP18
Scheduler Awareness

Contribution #1
Serial Inner Loop

Vertex Data

Vertex (Outer Loop)

Edge (Inner Loop)
Serial Inner Loop

Vertex Data

Vertex (Outer Loop)

Edge (Inner Loop)
Scheduler Un-Awareness

Vertex Data

Vertex (Outer Loop)

Edge (Inner Loop)
Scheduler Un-Awareness

Vertex Data

Vertex (Outer Loop)

Edge (Inner Loop)

Chunk A

Chunk B

Chunk C
Scheduler Awareness

Edge (Inner Loop)

Vertex Data

Vertex (Outer Loop)

Chunk A

Chunk B

Chunk C
Scheduler Awareness

1. Writes **at the end of a chunk** are redirected to a private per-chunk merge buffer.
1. Writes at the end of a chunk are redirected to a private per-chunk merge buffer.
Scheduler Awareness

1. **Writes at the end of a chunk** are redirected to a private per-chunk merge buffer.

2. **Writes in the middle of a chunk** can be committed to shared state without synchronization.
1. **Writes at the end of a chunk** are redirected to a private per-chunk merge buffer.

2. **Writes in the middle of a chunk** can be committed to shared state without synchronization.
Analyzing Scheduler Awareness

- Performance impact depends primarily on the scheduling granularity

- **Scheduler Un-Awareness**: trade-off between load balance and probability of write conflicts

- **Scheduler Awareness**: finer granularity leads to increased merge operation overhead
PageRank: Performance vs. Scheduling Granularity

**dimacs-usa**
(low, even degree distribution)

- Scheduler Un-Aware

**uk-2007**
(extremely skewed)

- Scheduler-Aware

![Graphs showing performance comparison](image)

- 10× Different

- 1.2×

- 50×

- 3.3×
PageRank: Performance vs. Number of Cores

Key Insights

- Huge improvement for extremely skewed graphs
- Still beneficial for evenly-distributed low-degree graphs

dimacs-usa (low, even degree distribution) vs. uk-2007 (extremely skewed)
Vector-Sparse

Contribution #2
Compressed-Sparse

Vertex Index

Edges
Vectorizing Compressed-Sparse
Vectorizing Compressed-Sparse

Vertex 0
Vectorizing Compressed-Sparse

23  10  50  4  0  53  62  1  78  50  23  4

Vertex 0  Vertex 1
Vectorizing Compressed-Sparse
Vector-Sparse

Vertex 0

Vertex 1

Vertex 2

Padding
Vector-Sparse

Padding + “valid” bits
Vector-Sparse

Padding + “valid” bits + top-level vertex spread-encoding
Analyzing Vector-Sparse

Packing Efficiency

Average Efficiency

0% 25% 50% 75% 100%

di-macs-usa livejournal twitter-2010 friendster uk-2007

Performance Impact

Speedup

0.0 0.5 1.0 1.5 2.0 2.5 3.0

PageRank CC BFS
di-macs-usa livejournal twitter-2010 friendster uk-2007

Generally ≥ 75%

1.5× to 2.5×
Performance Comparison

Putting it all together
Evaluation Scope

• Grazelle is compared with Ligra, Polymer, GraphMat, and X-Stream

• Three applications: PageRank, Connected Components, Breadth-First Search

• Running on a machine equipped with four Intel Xeon E7-4850 v3 processors
  • 14 physical cores / 28 logical cores per socket
PageRank: Peak Processing Throughput

- Grazelle-Pull
- Grazelle-Push
- Ligra-Pull
- Ligra-Push
- Polymer
- Graph Mat
- X-Stream

Execution Time (ms)

- di-macs-usa: 3.6x
- livejournal: 2.3x
- twitter-2010: 2.3x
- friendster: 1.4x
- uk-2007: 15.2x

Logarithmic
Connected Components: Dynamic Control Flow

Execution Time (ms)

Grazelle  Ligra  Ligra-Dense  Polymer  Graph Mat  X-Stream

di-macs-usa  livejournal  twitter-2010  friendster  uk-2007

4.9x  1.5x  1.6x  21.1x

Logarithmic
Breadth-First Search: Compatibility of Optimizations

Execution Time (ms)

- Grazelle
- Ligra
- Ligra-Dense
- Polymer
- Graph Mat
- X-Stream

Logarithmic
Conclusion

• Two contributions to improve inner loop parallelization for pull-based graph processing
  • Scheduler Awareness: eliminate write conflicts
  • Vector-Sparse: enable SIMD vectorization

• Grazelle significantly out-performs state-of-the-art, in some cases by over 10×

• Grazelle is available for download at https://github.com/stanford-mast/Grazelle-PPoPP18
Thank You

Questions?