Parallel Triangle Counting and K-Truss Identification Using Graph-Centric Methods

Chad Voegele, Yi-Shan Lu, Sreepathi Pai, Keshav Pingali
The University of Texas at Austin

09/13/2017
Graph-Centric vs. Matrix-Centric Abstractions

- **Active element**
  - Node/edge where computation is needed

- **Operator**
  - Computation at active element
  - Neighborhood: Set of nodes/edges read/written by the update

- **Parallelism**
  - Disjoint updates
  - Read-only operators, e.g. triangle counting

- **Bulk operations**
  - Matrix-matrix/vector multiplication
  - Element-wise manipulation
  - Reduction

- **Parallelism**
  - Inside individual operations
Galois: Graph-Centric Programming Framework

Shared-Memory Galois [1]
(C++ Library)
• Parallel data structures
  • Graphs, bags, etc.
• Parallel loops over active elements
  • for_each, do_all, etc.
• Support for
  • Load balancing
  • Scheduling
  • Dynamic work

IrGL [2]
(Compiler)
• Translates Galois programs to CUDA
• Applies GPU-specific optimizations
  • Iteration outlining
  • Cooperative conversion
  • Nested parallelism

Advantages of Graph-Centric Approach
Eliminating Barriers in a Round

Graph-centric methods:
Operator for edges

Operator for $e_1$
Operator for $e_2$
Operator for $e_3$
... Operator for $e_n$

Barrier between rounds

Matrix-centric methods:
Matrix operation for each step

K-Truss begins

Enumerate triangles

Count number of triangles for edges

Do all edges have sufficient support?

Yes

Remove edges w/ insufficient support

K-Truss done

No

Matrix operation for triangle enumeration

Barrier in a round

Matrix operation for counting # triangles for edges

Barrier in a round

Reduction to check for edges w/ insufficient support

Barrier in a round

Matrix operation for removing selected edges

Barrier between rounds
Exploiting Domain Knowledge in Operators

Graph as Compressed Sparse Row (CSR)

- **EdgeRemoved**
  - 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- **EdgeDst**
  - 1 2 3 0 2 3 0 1 3 4 0 1 2 4 5 2 3 5 3 4

- **EdgeRange**
  - 0 3 6 10 15 18 20

Early termination when edge support reaches k – 2.

Sorted edge lists to speed up edge list intersection from $O(\text{deg}(u) \times \text{deg}(v))$ to $O(\text{deg}(u) + \text{deg}(v))$.

Sorted edge lists to locate edges using binary search when removing edges.

Edge removals may be visible in current round, reducing the number of rounds.
Avoiding Runtime Memory Management

Graph-centric methods: Load graphs and update node/edge data in the graphs

EdgeData:
```
e e e e e e e e e e e e e e e e e e --
```

EdgeDst:
```
1 2 3 0 2 3 0 1 3 4 0 1 2 4 5 2 3 5 3 4 --
```

EdgeRange:
```
0 3 6 10 15 18 20
```

NodeData:
```
n n n n n n --
```

Matrix-centric methods: Construct matrices at runtime

```
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

* =

Fixed after graphs are loaded.

Needs runtime memory management.
Advantages of Graph-Centric Approach

• Eliminates barriers in a round
• Exploits domain knowledge in operators
• Avoids runtime memory management
Experimental Setup

Platform

- CPU
  - Broadwell-EP Xeon E5-2650 v4 @ 2.2 GHz
  - 30 MB LLC, 192 GB RAM
  - g++ 4.9
  - 1, 12 or 24 threads
- GPU
  - Pascal-based NVIDIA GTX 1080
  - 8 GB RAM
  - NVCC 8.0

Baseline from IEEE HPEC static graph challenge [3]

- Triangle counting: serial miniTri in C++
- K-truss computation: reference implementation in Julia 0.60

Parameter

- Compute $k_{\text{max}}$-truss for each graph.
- $k_{\text{max}}$: the maximum k for a graph to return non-empty truss.

---

Runtime
K-Truss Runtime

End-to-end runtime after the graph is loaded and before the results are printed.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Speedup over Julia**
K-Truss Runtime

End-to-end runtime after the graph is loaded and before the results are printed.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>1.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>428.87</td>
</tr>
</tbody>
</table>

**Speedup over Julia**
K-Truss Runtime

End-to-end runtime after the graph is loaded and before the results are printed.

Speedup over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>1.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>428.87</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>623.62</td>
</tr>
</tbody>
</table>

Maximum speedup of cpu-24 over cpu-01: 14.30X (~117M edges)
K-Truss Runtime

End-to-end runtime after the graph is loaded and before the results are printed.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>1.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>428.87</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>623.62</td>
</tr>
<tr>
<td>Gpu</td>
<td>2,213.14</td>
</tr>
</tbody>
</table>

Maximum speedup of cpu-24 over cpu-01: 14.30X (~117M edges)
End-to-end runtime after the graph is loaded and before the results are printed.

**Triangles Runtime**

Maximum speedup of cpu-24 over cpu-01: 17.22X (~15.7M edges)

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniTri</td>
<td>1.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>163.23</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>380.57</td>
</tr>
<tr>
<td>Gpu</td>
<td>1,760.47</td>
</tr>
</tbody>
</table>
Memory Usage
K-Truss Memory Usage

Measurement
Julia: @time

% over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Lower is better

192GB Total CPU memory

Memory Usage (Bytes)

# Edges

Julia
K-Truss Memory Usage

Measurement
Julia: @time
CPU: Galois’ internal allocator

% over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>0.54</td>
</tr>
</tbody>
</table>
K-Truss Memory Usage

Measurement
Julia: `@time`
CPU: Galois’ internal allocator

% over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>0.54</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>11.05</td>
</tr>
</tbody>
</table>

Lower is better
K-Truss Memory Usage

Measurement
Julia: @time
CPU: Galois’ internal allocator
GPU: cudaMemGetInfo

% over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>0.54</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>11.05</td>
</tr>
<tr>
<td>Gpu</td>
<td>1.09</td>
</tr>
</tbody>
</table>
Triangles Memory Usage

Measurement

**MiniTri: malloc_stats in glibc**

CPU: Galois’ internal allocator

GPU: cudaMemGetInfo

% over MiniTri

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniTri</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>94.31</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>791.64</td>
</tr>
<tr>
<td>Gpu</td>
<td>50.14</td>
</tr>
</tbody>
</table>

Measurement

CPU: Galois’ internal allocator

GPU: cudaMemGetInfo
Energy Usage
K-Truss Energy Usage

Measurement
Julia: Intel RAPL counters
CPU: Intel RAPL counters
GPU: nvprof

% over Julia

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>2.27</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>2.03</td>
</tr>
<tr>
<td>Gpu</td>
<td>0.48</td>
</tr>
</tbody>
</table>

Lower is better
Triangles Energy Usage

Measurement

MiniTri: Intel RAPL counters
CPU: Intel RAPL counters
GPU: nvprof

% over MiniTri

<table>
<thead>
<tr>
<th>Variant</th>
<th>Geo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiniTri</td>
<td>100.00</td>
</tr>
<tr>
<td>Cpu-01</td>
<td>12.95</td>
</tr>
<tr>
<td>Cpu-24</td>
<td>12.07</td>
</tr>
<tr>
<td>Gpu</td>
<td>2.55</td>
</tr>
</tbody>
</table>

Lower is better
Conclusions

• Graph-centric methods deliver two to three orders of magnitude improvements over matrix-centric IEEE HPEC static graph challenge reference implementations.

• Advantages of graph-centric methods over matrix-centric methods
  • Eliminates barriers in a round.
  • Exploits domain knowledge in operators.
    • Early operator termination
    • On-the-spot edge removals
    • Sorting of edge lists for faster edge list intersections and edge removals
  • Avoids runtime memory management.
Thank you!

Questions? Comments?