A Sparse Dynamic Graph and Matrix Data Layout

Oded Green

Georgia Tech
College of Computing
Computational Science and Engineering
Going to talk about 2 things

• Hornet
  – A scalable and dynamic data structure for graph algorithms and linear algebra based problems
  – Formerly known as cuSTINGER

• HornetsNest
  – A framework for static and dynamic analytics
Hornet – Upfront Summary

- Can support over **250 million** updates per second
- Low overhead in comparison with CSR
  - Initializing is also relatively in-expensive – usually less than **3X** slower
  - Equal performance
- Currently implemented for CUDA
  - We are porting Hornet to the CPU
- Really easy to use
Graph Primitives – Upfront summary

• Great performance for static and dynamic graph algorithms

• Scalable

• Simple to use
Big Data problems need Graph Analysis

Communication networks:
- World-wide connectivity
- High velocity changes
- Different types of extracted data:
  - Physical communication network.
  - Person-to-person communication network.

Financial networks:
- Transactions between players.
- Different transactions types (property graph)

Health-Care networks:
- Various players.
- Pattern matching and epidemic monitoring.
- Problem sizes have doubled in last 5 years.

Graphs are a unifying motif for data analytics.

Oded Green, GTC-DC-17
STINGER

- **STINGER**: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation
- Enable algorithm designers to implement dynamic & streaming graph algorithms with ease.
- Portable semantics for various platforms
  - Linked list of edge blocks not ideal for the GPU
- Good performance for all types of graph problems and algorithms - static and dynamic.
- Assumes globally addressable memory access
STINGER and cuSTINGER Properties

✓ A Simple programming model
✓ Millions of updates per second to graph
  ✓ Updates are not bottlenecks for analytics.
✓ Advanced memory manager
  ✓ Transfers data between host and device automatically
  ✓ Reduces initialization time
  ✓ Allows for simple update processes

STINGER Papers: [Bader et al.; 2007; Tech Report], [Ediger et al.; HPEC; 2012], [McColl et al.; PPAA; 2014]
cuSTINGER paper: [Green&Bader; HPEC, 2016]:

**cuSTINGER: Supporting dynamic graph algorithms for GPUs**
cuSTINGER is now HORNET
Definitions

• Dynamic graphs (matrices)
  - Graph can change over time.
  - Changes can be to topology, edges, or vertices.
    • For example new edges between two vertices.
  - Changes to edge or vertex weights

• Streaming graphs:
  - Graphs changing at high rates.
  - 100s of thousands of updates per second.
Dynamic graph example

• Only a subset of the entire graph...

• Dynamic:
  – At time $t$:
    • $v$ and $w$ become friends.
    • $\text{insert} \_\text{edge}(v,w)$
  – At time $\hat{t}$:
    • $u$ and $v$ no longer friends
    • $\text{delete} \_\text{edge}(u,v)$

• Additional operations include vertex insertions & deletions
“Separation of powers”

• Dynamic graph data structure and dynamic graph algorithms are in two different repositories
  – Easy to integrate with external library
  – Can also be used with matrices

• First part of today’s talk will be on the dynamic data structure
Part 1 – Data Structure

cuSTINGER Version 2.0

- Improved initialization time
  - 100x of time faster than Version 1.0
- New memory manager
  - Reduces fragmentation
  - Enables memory reclamation
  - Offers good memory bounds
- Scalable data structure
  - Can easily grow 1000X its initial size without needing to be re-initialized
- Faster updates
So what else do we need?

- We need a dynamic graph data structure
- These data structures don’t cut it

<table>
<thead>
<tr>
<th>Names</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense Adjacency Matrix</td>
<td>• Flexible</td>
<td>• Limited utilization for sparse data</td>
</tr>
<tr>
<td>Linked lists</td>
<td>• Flexible</td>
<td>• Poor locality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allocation time is costly</td>
</tr>
<tr>
<td>COO (Edge list) - unsorted</td>
<td>• Has some flexibility</td>
<td>• Poor locality</td>
</tr>
<tr>
<td></td>
<td>• Updates are simple</td>
<td>• Stores both the source and destination</td>
</tr>
<tr>
<td>CSR</td>
<td>• Uses exact amount of memory</td>
<td>• Inflexible</td>
</tr>
</tbody>
</table>
Compressed Sparse Row (CSR)

Pros:
• Uses precise storage requirements
• Great locality
  - Good for GPUs
• Handful of arrays
  - Simple to use and manage

Cons:
• Inflexible.
• Network growth unsupported
• Topology changes unsupported
• Property graphs not supported
**Hornet – A High Level View**

### User-Interface

<table>
<thead>
<tr>
<th>Vertex Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Supports updates**
  - Supports edge insertion/deletion and deletion.
  - Supports vertex insertion/deletion.

Over-allocated space

```
<table>
<thead>
<tr>
<th>Dest./Col. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2</td>
</tr>
<tr>
<td>1 2 0 5 2 6 2 5 1 4 0 3 4 4 1 4</td>
</tr>
</tbody>
</table>
```
Hornet – Property Graph Support

**USER-INTERFACE**

<table>
<thead>
<tr>
<th>Vertex Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dest./Col.

<table>
<thead>
<tr>
<th>Weight 2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Time 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>User 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>User 2</th>
</tr>
</thead>
</table>

....
Hornet in Detail

<table>
<thead>
<tr>
<th>USER-INTERFACE</th>
<th>Vertex Id</th>
<th>Used (#Neighbors/nnz)</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Over-allocated space for vertex insertions

Over-allocated space for power-of-two rule

MEMORY MANAGER

Vec-Tree Bit status

Dest./Col. Weight

MEMORY MANAGER

bsize = 1

bsize = 2

bsize = 2

bsize = 4

bsize = 2

bsize = 2

bsize = 4

bsize = 4

bsize = 4

bsize = 4

bsize = 4

\[ BA_{0,1} \]

\[ BA_{1,1} \]

\[ BA_{1,2} \]

\[ BA_{2,1} \]

\[ B \]

\[ A \]

\[ 0, 1 \]

\[ 1, 1 \]

\[ 1, 1 \]

\[ 0, 0 \]

\[ 1, 1 \]

\[ 1, 1 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]

\[ 0, 0 \]
Hornet Performance Analysis

• Memory Utilization

• Initialization Overhead

• Update rate
  – Number of sustainable updates per second
### Experimental Setup

<table>
<thead>
<tr>
<th>GPU</th>
<th>Architecture</th>
<th>SMs</th>
<th>SPs</th>
<th>Memory (GB)</th>
<th>Memory Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>K40</td>
<td>Kepler</td>
<td>15</td>
<td>2880</td>
<td>12</td>
<td>GDDR5</td>
</tr>
<tr>
<td>P100</td>
<td>Pascal</td>
<td>56</td>
<td>3584</td>
<td>16</td>
<td>HBM2</td>
</tr>
</tbody>
</table>

- Unless noted otherwise, all performance analysis is for the **P100**
Inputs Graphs

- DIMACS 10 Graph Implementation Challenge
- SNAP – Stanford Network Analysis Project
- Florida Matrix Collection

The following is only a subset of these graphs:

| Name      | Type          | |V|  | |E|  | Source   |
|-----------|---------------|-----|-----|----------------|
| coAuthorsDBLP | Collaboration | 299k | 1.95M | DIMACS          |
| as – skitter | Trace route   | 1.69M | 11.1M | SNAP            |
| kron_21    | Random        | 2M   | 201M | DIMACS          |
| cit – patents | Citation    | 3.77M | 16.5M | SNAP            |
| cage15     | Matrix        | 5.15M | 94M  | DIMACS          |
| uk – 2002  | Webcrawl      | 18.52M | 523M | DIMACS          |
Memory Utilization - Overall

- 70% average utilization of CSR
- Better utilization in comparison to: COO, cuSTINGER, AIMS
Insertion Rates

- Supports over 250M updates per second
- Hornet
  - $4X - 10X$ faster than cuSTINGER
  - Does not have *performance dip* like cuSTINGER
- Scalable growth in update rate

![](chart1.png)  ![](chart2.png)

- cuSTINGER
- Hornet
Part 2: HornetsNest

- Algorithm framework for Hornet data structure
  - We support CSR as well

- All algorithms are implemented using a small set of operations
  - We show that these operators are efficient for static graph algorithms and can be used for dynamic graph algorithms

- Uses features from C++11 and C++14
Algorithmic Graph Primitives

- All algorithms are implemented through this API

- Simple primitives
  - `ForAllVerticesInG(G, f(v ∈ V), struct)`
  - `ForAllEdgesInG(G, f(src ∈ V, dest ∈ V), struct)`
  - `InsertEdges(G, E_{new})`
  - `RemoveEdges(G, E_{rem})`
  - `InsertVertices(G, V_{new})`
  - `RemoveVertices(G, V_{rem})`
Algorithmic Graph Primitives

- **ForAllVerticesInArray** \( (G, A, f \ (v \in V), \text{struct}) \)

- **ForAllEdgesInArray** \( (G, A_V, f \ (src \in V, \text{dest} \in V), \text{struct}) \)
  - Array of vertices that will traverse all neighbors
  - Breadth first search and betweenness centrality

- **ForAllEdgesInArray** \( (G, A_E, f \ (src \in V, \text{dest} \in V), \text{struct}) \)
  - Array of explicit edge pairs
  - Great for processing edges batches
Performance Analysis

• Sparse Vector Matrix Multiplication

• Breadth First Search

• Triangle Counting
Sparse Matrix Vector Multiplication

- In comparison to DCSR [King et al; 2016; ISC]
  - DCSR requires customized SpMV
- Hornet uses identical algorithm code as CSR.
Actual BFS Code

```cpp
void BfsTopDown2::run() {
    while (queue.size() > 0) {
        forAllEdges(hornet, queue,
            BFSOperatorAtomic { current_level, d_distances, queue },
            load_balacing);
        queue.swap();
        current_level++;
    }
    OPERATOR(Vertex& vertex, Edge& edge) {
        auto dst = edge.dst_id();
        if (atomicCAS(d_distances + dst, INF, current_level) == INF)
            queue.insert(dst);
    }
}
```

- Har
- This code actually runs on the GPU
Breadth First Search

- Using a similar algorithm in Gunrock
  - Gunrock has additional optimizations that can make it faster than cuSTINGER
  - “Apples to Apples” comparison

Oded Green, GTC-DC-17
Triangle Counting: CSR Vs. Hornet

| Name          | |V| | |E| | Time-CSR (sec.) | Time-cuSTINGER (sec.) | Execution Difference |
|---------------|---------------|---------------|---------------|-------------------|----------------------|----------------------|
| coAuthorsDBLP | 299k          | 1.95M         | 0.218         | 0.242             | +10%                 |
| as – skitter  | 1.69M         | 11.1M         | 57.14         | 59.37             | +3.8%                |
| kron_21       | 2M            | 201M          | 2992          | 2996              | +0.14%               |
| cit – patents | 3.77M         | 16.5M         | 0.814         | 0.830             | +2%                  |
| cage15        | 5.15M         | 94M           | 6.544         | 7.204             | +10%                 |
| uk – 2002     | 18.52M        | 523M          | 424.9         | 431.4             | +1.6%                |

- Triangle counting algorithm taken from [Green et al; IA³;2014]
- Simply replace CSR accesses with Hornet
- Executed on a K40
# Library Overview

## Completed algorithms and on-going

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Static</th>
<th>Dynamic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth first search</td>
<td>✓</td>
<td>on-going</td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td>✓</td>
<td>✓</td>
<td>Static - [Green et al; IA32014] Dynamic - [Makkar; HiPC 2017]</td>
</tr>
<tr>
<td>Connect components</td>
<td>✓</td>
<td>on-going</td>
<td>[McColl; HiPC 2013]</td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td>✓</td>
<td>on-going</td>
<td>[Green; SocialCom 2012]</td>
</tr>
<tr>
<td>Page Rank</td>
<td>✓</td>
<td>on-going</td>
<td>New algorithm (non linear algebra formulation)</td>
</tr>
<tr>
<td>Katz Centrality</td>
<td>✓</td>
<td>✓</td>
<td>New algorithm (non linear algebra formulation)</td>
</tr>
<tr>
<td>KTruss</td>
<td>✓</td>
<td></td>
<td>[Green; HPEC 2017] – HPEC Graph Challenge Innovation Award</td>
</tr>
</tbody>
</table>

Of course many more algorithms to come...
Take away

• Dynamic data structure for sparse data sets

• Supports high update rates

• Simple and high-level programming model
  – Utilizes graph primitives

• Scalable in both data size and in performance
Hornet Team (Past & Current)
Thank you

• Email: ogreen@gatech.edu

• Hornet:
  – https://github.com/hornet-gt/hornet

• HornetsNest:
  – https://github.com/hornet-gt/hornetsnest
Backup slides