STINGER: Spatio-Temporal Interaction Networks and Graphs Extensible Representation

Tim Shaffer
Big problems are often represented as graphs

- **Bioinformatics:** identifying target proteins
- **Astrophysics:** outlier detection, clustering
- **Social networks:** tracking information spread, relationships

Graphs are enormous and *vary over time*
Example: Social Media

Facebook
- ~1 billion users
- Average of 130 friends, some accounts much higher
- 30 billion user interactions per month

Twitter
- 500 million active users
- 340 million tweets per day
Challenges

Large scale
- Memory efficiency
- Parallelism

Frequent updates
- Fast operations
- Partial recomputation
**STINGER** is a high-performance graph data structure

- Fast insertions, deletions, and updates
- Types and weights for edges and vertices
- Filtering by timestamp, type, etc.
- Parallel graph operations
Also a library/API for using the data structure

- Written in C/C++
- Bindings for Python and Java
- Assumes shared memory system
- Targets x86 and Cray XMT machines
Graph consists of
- Vertex array
- EType array

Edges are stored in linked lists of edge blocks

EType array is an additional index with pointers to all edges of a given type
Each vertex has a
- type
- weight
- linked list of its outgoing edges

Each edge has a
- type
- weight
- creation and modification timestamp
Multiple parallel readers and a single writer

Algorithms can operate serially or in parallel over all nodes, edges, neighbors, etc.

**Does not** provide ACID semantics
C/C++ library implementation, C interface (no templates)

Types, weights are 64 bit integer values

Parallelism through OpenMP and XMT pragmas

No cluster support
Implemented Algorithms

- Streaming clustering coefficients
- Streaming connected components
- Streaming community detection
- Parallel agglomerative clustering
- Streaming Betweenness Centrality
- K-core Extraction
- Classic breadth-first search
Example: Shortest Path

```c
STINGER_FORALL_OUT_EDGES_OF_VTX_BEGIN(S, current.vertex) {
    //for all the neighbors of the current vertex
    int64_t new_cost = cost_so_far[current.vertex] + STINGER_EDGE_WEIGHT;
    if (new_cost < cost_so_far[STINGER_EDGE_DEST]
        || cost_so_far[STINGER_EDGE_DEST] == std::numeric_limits<int64_t>::max()) {
        cost_so_far[STINGER_EDGE_DEST] = new_cost;
        weighted_vertex_t next;
        next.vertex = STINGER_EDGE_DEST;
        next.cost = new_cost;
        frontier.push(next);
    }
}
```
Performance

- Optimized for extremely frequent updates
- Fast locking implementation (esp. on Cray XMT)
- Batched updates (big improvement)
Performance

Memory Usage – Small (|V|: 32K, |E|: 256K)

Graph Package
Performance

Page Rank – Medium (|V|: 1M, |E|: 8M)

Time (s)

<table>
<thead>
<tr>
<th>Graph Package</th>
<th>6.76</th>
<th>45.8</th>
<th>181</th>
<th>553</th>
<th>2880</th>
<th>3390</th>
<th>4130</th>
<th>4420</th>
<th>63900</th>
</tr>
</thead>
<tbody>
<tr>
<td>stinger</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>giraph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mfg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>netwux</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>orientdb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pegasus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bagel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>titan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Benchmarked at 3 million edge updates per second for graphs with 1 billion edges

Streaming and parallel operations greatly reduce computation cost

Partial recomputation greatly reduces cost of updates
Example: connected components of a graph of 500 million edges
- Measured 1.26 million updates/second
- 137x faster than recomputing

Allows desktop computer to process graphs with millions of vertices and edges

Big machines can reach billions of edges