Outline

• Introduction
• Network Analysis
• Static Parallel Algorithms
• GraphCT
• Dynamic Parallel Algorithms
• STINGER
  - Data Structure Creation
  - Batch Insertions & Deletions
  - Graph Traversal
  - Q&A
STINGER

• Enhanced representation developed for dynamic graphs developed in consultation with David A. Bader, Johnathan Berry, Adam Amos-Binks, Daniel Chavarría-Miranda, Charles Hastings, Kamesh Madduri, and Steven C. Poulos.

• Design goals:
  - Be useful for the entire “large graph” community
  - Portable semantics and high-level optimizations across multiple platforms & frameworks (XMT C, MTGL, etc.)
  - Permit good performance: No single structure is optimal for all.
  - Assume globally addressable memory access
  - Support multiple, parallel readers and a single writer

• Operations:
  - Insert/update & delete both vertices & edges
  - Aging-off: Remove old edges (by timestamp)
  - Serialization to support checkpointing, etc.
STINGER

- Semi-dense edge list blocks with free space
- Compactly stores timestamps, types, weights
- Maps from application IDs to storage IDs
- Deletion by negating IDs, separate
**STINGER**

- Georgia Tech implementation runs in parallel on Cray XMT and multicore desktop
- Shows little or no performance overhead for many kernels
- Recent publication using STINGER:
  - Demonstrates good performance for small graphs on Intel Nehalem and large streaming datasets on the Cray XMT.
STINGER Package

- See ROADMAP in doc/ for more information
- Copy an appropriate make.inc file first

  - STINGER/
    - compat/ (Cray XMT compatibility layer)
    - doc/ (Library Documentation)
    - examples/ (Several static & streaming analytics)
    - genstreams/ (Streaming RMAT graph generator)
    - include/ (Header files)
    - src/ (STINGER source files)
    - Makefile
    - main.c (My application)
STINGER Header & Source Files

• Separate header & source files by function
• Users only need be familiar with **stinger.h**

```plaintext
STINGER/
  - include/                      - src/
    - stinger-atomics.h           - stinger.c
    - stinger-defs.h              - stinger-utils.c
    - stinger.h                   - timer.c
    - stinger-internal.h          - xmalloc.c
    - stinger-traversal.h
    - stinger-utils.h
    - timer.h
    - xmalloc.h
```
#include "stinger-atomics.h"
#include "stinger.h"

struct stinger * S;
S = stinger_new ();
stinger_set_initial_edges (S, nv, 0, off, ind, weight, NULL, NULL, 0);
STINGER Parameters

In stinger-defs.h:

- `#define STINGER_MAX_LVERTICES (1L<<25)`
  - Maximum number of vertices (here SCALE 25)
- `#define STINGER_EDGEBLOCKSIZE 100`
  - Edges per block
  - Choose based on average degree and cache line size
- `#define STINGER_NUMETYPES 1`
  - Number of edge types to support
Edge Insertion

• `stinger_insert_edge (S, type, from, to, weight, timestamp)`
  - Finds an open space, or gets a new block, for a directed edge `<from, to>` with weight, type, and timestamp

• `stinger_insert_edge_pair (S, type, from, to, weight, timestamp)`
  - Same as above
  - Also inserts the reverse edge with identical properties
  - Not atomic
Edge Removal

- **stinger_remove_edge** (S, type, from, to)
  - Removes a single edge

- **stinger_remove_edge_pair** (S, type, from, to)
  - Removes the specified edge and its reverse
  - Not atomic
Edge Weight Increment

- **stinger_incr_edge (S, type, from, to, weight, timestamp)**
  - Atomically adds “weight” to existing edge weight and updates both timestamps

- **stinger_incr_edge_pair (S, type, from, to, weight, timestamp)**
  - Same as above + the reverse edge
  - Not atomic between edges
Batch Remove & Insert

• `stinger_remove_and_insert_edges (S, type, from, nremove, remove[], ninsert, insert[], weight[], timestamp)`
  - Removes a batch of edges `remove[]` with a given type originating at a single vertex “from”
  - Inserts a batch of edges `insert[]` with `weight[]` originating at a single vertex “from”
  - Passing NULL for `weight[]` sets all weights to 1
Visiting a Neighbor List

• Safe way: Copy out

• stinger_gather.Typed_SUCCESSORS (S, type, vertex, &num, buf[], buflen)
  - For edges of a given type originating from a single vertex
  - Copy neighbors into buf[] with maximum length “buflen”
  - Returns the number of neighbors copied “num”
while (!is_empty(stack, &top)) {
  int64_t k, myStart, myEnd;
  u = pop(stack, &top);
  myStart = off[u];
  myEnd = off[u+1];
  for (k = myStart; k < myEnd; k++) {
    v = ind[k];
    if (int_fetch_add(marks + v, 1) == 0) {
      d[v] = my_root;
      push(v, stack, &top);
    } else {
      if (!(d[v]==d[my_root])) {
        int64_t t = int_fetch_add(&cross_count, 1);
        crossU[t] = u;
        crossV[t] = v;
      }
    }
  }
}
Adapting for STINGER: Modifications

```c
while (!is_empty(stack, &top)) {
    int64_t k, myStart, myEnd;
    size_t md;
    u = pop(stack, &top);
    deg_u = stinger_outdegree(S, u);
    myStart = stinger_int64_fetch_add(&head, deg_u);
    myEnd = myStart + deg_u;
    stinger_gather_TYPED_successors(S, 0, u, &md, &neighbors[myStart], deg_u);
    for (k = myStart; k < myEnd; k++) {
        v = neighbors[k];
        if (stinger_int64_fetch_add(marks + v, 1) == 0) {
            d[v] = my_root;
            push(v, stack, &top);
        } else {
            if (!(d[v] == d[my_root])) {
                int64_t t = stinger_int64_fetch_add(&cross_count, 1);
                crossU[t] = u;
                crossV[t] = v;
            }
        }
    }
}
```

Assuming a pre-allocated buffer, `neighbors`. 
Adapting for STINGER: Compatibility Layer

```c
1 while (!is_empty(stack, &top)) {
2    int64_t k, myStart, myEnd;
3    size_t md;
4    u = pop(stack, &top);
5    deg_u = stinger_outdegree(S, u);
6    myStart = stinger_int64_fetch_add(&head, deg_u);
7    myEnd = myStart + deg_u;
8    stinger_gather_typed_successors(S, 0, u, &md, &neighbors[myStart], deg_u);
9    for (k = myStart; k < myEnd; k++) {
10       v = neighbors[k];
11       if (stinger_int64_fetch_add(marks + v, 1) == 0) {
12          d[v] = my_root;
13          push(v, stack, &top);
14       } else {
15          if (!(d[v] == d[my_root])) {
16             int64_t t = stinger_int64_fetch_add(&cross_count, 1);
17             crossU[t] = u;
18             crossV[t] = v;
19          } }
20      }
21  }
```
Adapting for STINGER: Copy Out Transformation

```c
while (!is_empty(stack, &top)) {
    int64_t k, myStart, myEnd;
    size_t md;
    u = pop(stack, &top);
    deg_u = stinger_outdegree(S, u);
    myStart = stinger_int64_fetch_add(&head, deg_u);
    myEnd = myStart + deg_u;
    stinger_gather_typeof_successors(S, &marked, u, &md, &neighbors[myStart], deg_u);
    for (k = myStart; k < myEnd; k++) {
        v = neighbors[k];
        if (stinger_int64_fetch_add(marks + v, 1) == 0) {
            d[v] = my_root;
            push(v, stack, &top);
        } else {
            if (!(d[v] == d[my_root])) {
                int64_t t = stinger_int64_fetch_add(&cross_count, 1);
                crossU[t] = u;
                crossV[t] = v;
            }
        }
    }
}
```

- Copying neighbors isolates from dynamic changes.
- Keeps compiler-optimizable loop structure.
Traversal Macros

• Directly in the data structure with **Traversal Macros**

• Process a set of edges one edge at a time

• Reduce code bloat

• Provide data structure abstraction

• Can be nested (but remember scoping)

• Can contain other STINGER API calls
Basic Traversal Macros

- Direct access to data structure
- Fast but no safety
- Possibly inconsistent view if structure is changing underneath

- STINGER_FORALL_EDGES_OF_VTX_BEGIN(stinger *, vtxID)
- STINGER_FORALL_EDGES_OF_TYPE_OF_VTX_BEGIN(stinger *, type, vtxID)
- STINGER_FORALL_EDGES_BEGIN(stinger *, type)
Extended Options

• Read only versions of all macros
  - Local copy of a single edge at a time, higher safety and consistency
  - Insert \texttt{READ\_ONLY}\_
  - \texttt{STINGER\_READ\_ONLY\_FORALL\_EDGES\_OF\_VTX\_BEGIN(stinger *, vtxID)}

• Parallel versions of all macros
  - Insert \texttt{PARALLEL\_}
  - \texttt{STINGER\_PARALLEL\_FORALL\_EDGES\_OF\_TYPE\_OF\_VTX\_BEGIN(stinger *, type)}

• Read only parallel versions of all macros
  - Insert \texttt{READ\_ONLY\_PARALLEL\_}
  - \texttt{STINGER\_READ\_ONLY\_PARALLEL\_FORALL\_EDGES\_BEGIN(stinger *, type)}
Edge-based Data Macros

- Macros to access the current edge metadata:
  - STINGER_EDGE_SOURCE
  - STINGER_EDGE_DEST
  - STINGER_EDGE_TYPE
  - STINGER_EDGE_WEIGHT
  - STINGER_EDGE_TIME_FIRST
  - STINGER_EDGE_TIME_RECENT
Let’s sum weights of V’s out edges!

```c
uint64_t sum_weights(struct stinger * S, uint64_t v) {
    uint64_t result = 0;
    STINGER_FORALL_EDGES_OF_VTX_BEGIN(S, v) {
        result += STINGER_EDGE_WEIGHT;
    }
    return result;
}
```

• Note the matching `BEGIN` and `END` macros
• Note the `semicolon`
How about some parallelism?

```c
uint64_t sum_weights(struct stinger * S, uint64_t v) {
    uint64_t result = 0;
    STINGER_PARALLEL_FORALL_EDGES_OF_VTX_BEGIN(S, v) {
        atomic_add(&result, STINGER_EDGE_WEIGHT);
    }
    return result;
}
```

- Simply add `PARALLEL_` to the outer macros
- `STINGER_EDGE_WEIGHT` remains the same
- Must protect result with atomic
Nesting Macros

1 STINGER_FORALL_EDGES_OF_VTX_BEGIN(S, V) {
2    uint64_t v_neighbor = STINGER_EDGE_DEST;
3    STINGER_FORALL_EDGES_OF_VTX_BEGIN(S, v_neighbor) {
4       do_something(v_neighbor, STINGER_EDGE_DEST);
5    } STINGER_FORALL_EDGES_OF_VTX_END();
6 } STINGER_FORALL_EDGES_OF_VTX_END();

• For all neighbors of the neighbors of V
• Do something with the edge <neighbor_of_v, neighbor_of_neighbor_of_v>
STINGER API

1 STINGER_FORALL_EDGES_OF_VTX_BEGIN(S, V) {
2   stinger_insert_edge (S, 0,
       STINGER_EDGE_DEST,
       STINGER_EDGE_SOURCE,
       STINGER_EDGE_WEIGHT,
       STINGER_EDGE_TIME_RECENT);
3 } STINGER_FORALL_EDGES_OF_VTX_END();

• For all neighbors of the neighbors of V, insert a corresponding reverse edge
A Simple Breadth-first Search

1. while(qhead != qnext) {
2.     STINGER_FORALL_EDGES_OF_VTX_BEGIN(S, queue[qnext]) {
3.         if(!marked[STINGER_EDGE_DEST]) {
4.             queue[qhead] = STINGER_EDGE_DEST;
5.             level[qhead] = level[qnext] + 1;
6.             marked[STINGER_EDGE_DEST] = 1;
7.             qhead++;
8.         }
9.     } STINGER_FORALL_EDGES_OF_VTX_END();
10.    qnext++;
11. }
Max timestamp for marked vertices

```c
1 uint64_t max_timestamp(struct stinger * S, uint64_t marked[NV])
2 {
3     uint64_t result = INT64_MIN;
4
5     STINGER_FORALL_EDGES_BEGIN(S, 0) {
6         if(marked[STINGER_EDGE_SOURCE] ||
7             marked[STINGER_EDGE_DEST]) {
8             if(STINGER_EDGE_TIME_FIRST > result)
9                 result = STINGER_EDGE_TIME_FIRST;
10            if(STINGER_EDGE_TIME_RECENT > result)
11                result = STINGER_EDGE_TIME_RECENT;
12         }
13     } STINGER_FORALL_EDGES_END();
14
15     return result;
16 }
```

- Highly constructed example case, but we can do it!
Shiloach-Vishkin Connected Components

```c
1 while (1) {
2     int changed = 0;
3     STINGER_FORALL_EDGES_BEGIN (S, 0) {
4         if (component_map[STINGER_EDGE_DEST] <
5             component_map[STINGER_EDGE_SOURCE]) {
6             component_map[STINGER_EDGE_SOURCE] = component_map[STINGER_EDGE_DEST];
7             changed++;
8         }
9     }
10    STINGER_FORALL_EDGES_END ();
11 
12    if (!changed)
13        break;
14 
15    for (uint64_t i = 0; i < nv; i++) {
16        while (component_map[i] != component_map[component_map[i]])
17            component_map[i] = component_map[component_map[i]];
18    }
19 }
```

- Input is `component_map[NV] = {0, 1, .. NV − 1}`
- Output is `component_map[NV] = {component id’s}`
Massive Streaming Data Analytics

- Accumulate as much of the recent graph data as possible in main memory.

Diagram:

1. Pre-process, Sort, Reconcile
2. “Age off” old vertices
3. Alter graph
4. Update metrics
5. STINGER graph
6. Change detection

Insertions / Deletions

Affected vertices
Insertions-only connected components algorithm

**Edge insertion (in batches):**

- Relabel batch of insertions with component numbers.
- Collapse the graph, removing self-edges. Any edges that remain cross components.
- Compute components of component ↔ component graph. Relabel smaller into larger.
- Problem size reduces from number of changes to number of components.
- Can proceed concurrently with STINGER modification.
int64_t incr = 0;

MTA(“mta assert nodep”)
for (int64_t k = 0; k < nInsert; k++) {
    const int64_t i = ACTI2(insoff[k]);
    const int64_t ncompi = ncomp[i];
    willchange[k] = ncompi;

    for (int64_t k2 = insoff[k]; k2 < insoff[k+1]; k2++) {
        const int64_t ncompk = ncomp[ACTJ2(k2)];
        if (ncompk != ncompi) {
            int where = stinger_int64_fetch_add(&incr, 1);
            m1[where] = ncompi;
            m2[where] = ncompk;
        }
    }
}

num_comp = connected_components_edge(nv, incr, m1, m2, ncomp);

for(int64_t k = 0; k < nInsert; k++) {
    const int64_t i = willchange[k];
    if (ncomp[i] != i) {
        int64_t change = ncompsize[i];
        stinger_int64_fetch_add(&ncompsize[ncomp[i]], change);
        ncompsize[i] = 0;
    }
}
int64_t incr = 0;

MTA(“mta assert nodep”)
for (int64_t k = 0; k < nInsert; k++) {
    const int64_t i = ACTI2(insoff[k]);
    const int64_t ncompi = ncomp[i];
    willchange[k] = ncompi;

    for (int64_t k2 = insoff[k]; k2 < insoff[k+1]; k2++) {
        const int64_t ncompk = ncomp[ACTJ2(k2)];
        if (ncompk != ncompi) {
            int where = stinger_int64_fetch_add(&incr, 1);
            m1[where] = ncompi;
            m2[where] = ncompk;
        }
    }
}

num_comp = connected_components_edge(nv, incr, m1, m2, ncomp);

for(int64_t k = 0; k < nInsert; k++) {
    const int64_t i = willchange[k];
    if (ncomp[i] != i) {
        int64_t change = ncompsize[i];
        stinger_int64_fetch_add(&ncompsize[ncomp[i]], change);
        ncompsize[i] = 0;
    }
}


Insertions-only Components
Update

```c
int64_t incr = 0;

MTA("mta assert nodep")
for (int64_t k = 0; k < nInsert; k++) {
    const int64_t i = ACTI2(insoff[k]);
    const int64_t ncompi = ncomp[i];
    willchange[k] = ncompi;

    for (int64_t k2 = insoff[k]; k2 < insoff[k+1]; k2++) {
        const int64_t ncompk = ncomp[ACTJ2(k2)];
        if (ncompk != ncompi) {
            int where = stinger_int64_fetch_add(&incr, 1);
            m1[where] = ncompi;
            m2[where] = ncompk;
        }
    }
}

num_comp = connected_components_edge(nv, incr, m1, m2, ncomp);
```

for(int64_t k = 0; k < nInsert; k++) {
    const int64_t i = willchange[k];
    if (ncomp[i] != i) {
        int64_t change = ncompsize[i];
        stinger_int64_fetch_add(&ncompsize[ncomp[i]], change);
        ncompsize[i] = 0;
    }
}
```

**Use the list of edges that span components to calculate new component IDs for every vertex**

**Variables:**
- NV : number of vertices
- num_comp : number of components
- ncomp[NV] : vertex-component map
- ncompsize[NV] : size of each component
- nInsert : number of vertices in batch
- insoff[] : vertex offset into batch
- ACTI2() : macro for edge insertion src
- ACTJ2() : macro for edge insertion dest

**Macros:**
- MTA("mta assert nodep")
int64_t incr = 0;

MTA("mta assert nodep")
for (int64_t k = 0; k < nInsert; k++) {
    const int64_t i = ACTI2(insoff[k]);
    const int64_t ncompi = ncomp[i];
    willchange[k] = ncompi;

    for (int64_t k2 = insoff[k]; k2 < insoff[k+1]; k2++) {
        const int64_t ncompk = ncomp[ACTJ2(k2)];
        if (ncompk != ncompi) {
            int where = stinger_int64_fetch_add(&incr, 1);
            m1[where] = ncompi;
            m2[where] = ncompk;
        }
    }
}

num_comp = connected_components_edge(nv, incr, m1, m2, ncomp);

for(int64_t k = 0; k < nInsert; k++) {
    const int64_t i = willchange[k];
    if (ncomp[i] != i) {
        int64_t change = ncompsize[i];
        stinger_int64_fetch_add(&ncompsize[ncomp[i]], change);
        ncompsize[i] = 0;
    }
}
ACKNOWLEDGMENTS
Collaborators and Acknowledgments

• Jason Riedy & Henning Meyerhenke, Research Scientists (Georgia Tech)

• Graduate Students (Georgia Tech):
  – David Ediger
  – Rob McColl
  – Pushkar Pande

• Bader PhD Graduates:
  – Seunghwa Kang (Pacific Northwest National Lab)
  – Kamesh Madduri (Lawrence Berkeley National Lab)
  – Guojing Cong (IBM TJ Watson Research Center)

• John Feo and Daniel Chavarría-Miranda (Pacific Northwest Nat’l Lab)

• Jon Berry, Bruce Hendrickson (Sandia National Labs)

• Jeremy Kepner (MIT Lincoln Laboratory)
Bader, Related Recent Publications (2005-2008)


Bader, Related Recent Publications (2009-2010)


- Karl Jiang, David Ediger, and David A. Bader. “Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation.” The 38th International Conference on Parallel Processing (ICPP), Vienna, Austria, September 2009.


- Seunghwa Kang, David A. Bader. “Large Scale Complex Network Analysis using the Hybrid Combination of a MapReduce cluster and a Highly Multithreaded System.” Fourth Workshop in Multithreaded Computing.
Acknowledgment of Support
Q&A