K-Truss Decomposition

ECE508 - MANYCORE PARALLEL ALGORITHMS

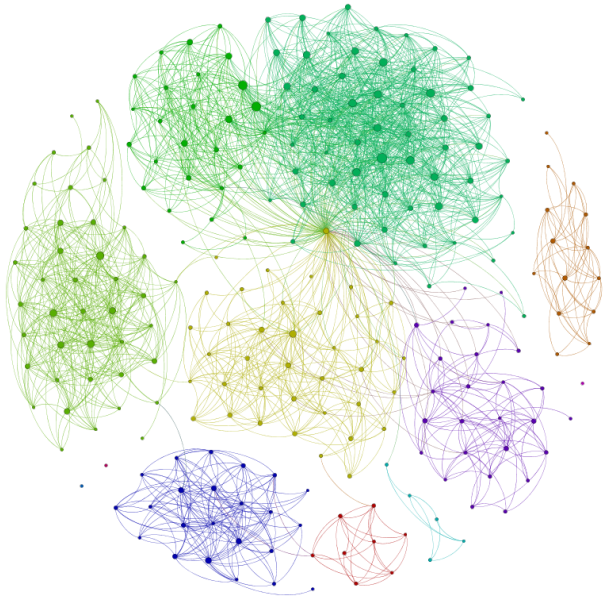
PROF. WEN-MEI HWU

HENGZHE DING, YIJIAN DUAN, DONG LIU

Design Overview

Problem

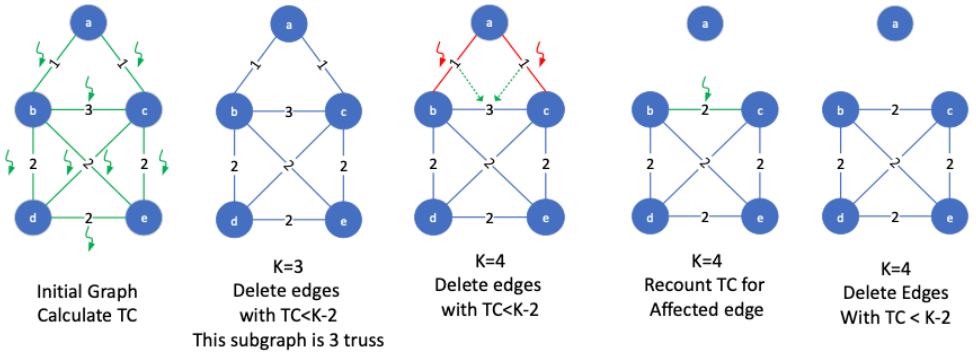
- K-truss - a type of cohesive subgraphs of a network
- High collaboration sub-network identification
- Visualization of large-scale networks and analysis of network connectivity.
- Large online social networks, e.g., Facebook and Twitter,



http://allthingsgraphed.com/2014/08/28/facebook-friends-network/

Problem

- K-truss decomposition
- Measures cohesiveness of community
- Analyze community network connection

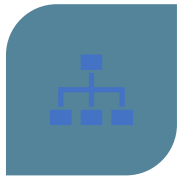


Challenges

LARGE NETWORKS

BILLIONS OF EDGES AND MILLIONS OF VERTICES.

RUNNING TIME



DATA STRUCTURE

PARALLELISM

Logic

```
While (graph is not empty for current k)
    mark all edges as affected for test tc < k-2
    While (still exist affected edges)
        test tc of affected
        if tc < k-2
            fake delete, // mark as -1
            mark affected edges // need to clear
            previous affected
    check currentEdges
```

Implementation

Data Structure

One thread per edge

Graph

Shrink each round of k

- **edgeSrc**
  - # numEdges
  - Int Array
- **edgeDes**
  - # numEdges
  - Int Array
- **rowPtr**
  - # numEdges + 1
  - Int Array

Edge flag

Inner while loop

- **affected**
  - # numEdges
  - Int Array
- **to_delete**
  - # numEdges
  - Int Array
- **e_aff**
  - # numEdges
  - Int Array

Pseudocode

- Outer while loop each for one k

```
Input:  G = (V, E)
Output: k-truss for 3 ≤ k ≤ k_max
1:  k ← 3
2:  Mark all e ∈ E as "affected"
3:  for each e = (u, v) ∈ E do                    ▷ Parallel for
4:      Binary search e' = (v, u) ∈ E
5:      I_r[e] ← e'
6:  end for
```

- Inner while loop

```
34: E ← StmCmp(E, "not deleted")                  ▷ Long update
35: Output E as k-truss edge list
36: if E ≠ ∅ then k ← k + 1 and goto 2
```

```
7:  while true do
8:      (E_aff, I'_f, I'_r) ← StmCmp(E, "affected and u < v")
9:      if E_aff = ∅ then goto 34;
10:     Mark all e ∈ E as "not affected"
11:     for each e = (u, v) ∈ E_aff do                ▷ Parallel for
12:         Δ(e) ← |adj(u) ∩ adj(v)|                  ▷ Algorithm 2
13:         if Δ(e) < k − 2 then
14:             Stage 1:
15:             Lookup e' = (u, v) ← I'_f[e]
16:             Lookup e'' = (v, u) ← I'_r[e]
17:             Mark e' and e'' in E as "delete"
18:             Stage 2:
19:             W ← adj(u) ∩ adj(v)                   ▷ Algorithm 2
20:             if W ≠ ∅ then
21:                 e1 ← (u, w); e2 ← (v, w); s.t. w ∈ W
22:                 Lookup e'1 = (w, u) ← I_r[e1]
23:                 Lookup e'2 = (w, v) ← I_r[e2]
24:                 Mark e1, e2, e'1, e'2 as "affected"
25:             end if
26:         end if
27:     end for
28:     for each e = (u, v) ∈ E do                    ▷ Parallel for
29:         if e labeled "delete" then
30:             u ← −1; v ← −1                        ▷ Short update
31:         end if
32:     end for
33: end while
```

Date, K., Feng, K., Nagi, R., Xiong, J., Kim, N. S., & Hwu, W. M. (2017, September). Collaborative (cpu+ gpu) algorithms for triangle counting and truss decomposition on the minsky architecture: Static graph challenge: Subgraph isomorphism. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-7). IEEE.

Implementation Demo

Parallel Optimization

Input: $G = (V, E)$
Output: k-truss for $3 \leq k \leq k_{max}$

1: $k \leftarrow 3$
2: Mark all $e \in E$ as "affected"
3: **for each** $e = (u, v) \in E$ **do** ▷ Parallel for
4: Binary search $e' = (v, u) \in E$
5: $\mathbf{I}_r[e] \leftarrow e'$
6: **end for**
7: **while** *true* **do**
8: $(E_{aff}, \mathbf{I}'_f, \mathbf{I}'_r) \leftarrow StmCmp(E,$ "affected and $u < v$")
9: **if** $E_{aff} = \emptyset$ **then goto** 34;
10: Mark all $e \in E$ as "not affected"
11: **for each** $e = (u, v) \in E_{aff}$ **do** ▷ Parallel for
12: $\Delta(e) \leftarrow |adj(u) \cap adj(v)|$ ▷ Algorithm 2
13: **if** $\Delta(e) < k - 2$ **then**
14: **Stage 1:**
15: Lookup $e' = (u, v) \leftarrow \mathbf{I}'_f[e]$
16: Lookup $e'' = (v, u) \leftarrow \mathbf{I}'_r[e]$
17: Mark e' and e'' in E as "delete"
18: **Stage 2:**
19: $W \leftarrow adj(u) \cap adj(v)$ ▷ Algorithm 2
20: **if** $W \neq \emptyset$ **then**
21: $e_1 \leftarrow (u, w)$; $e_2 \leftarrow (v, w)$; s.t. $w \in W$
22: Lookup $e'_1 = (w, u) \leftarrow \mathbf{I}_r[e_1]$
23: Lookup $e'_2 = (w, v) \leftarrow \mathbf{I}_r[e_2]$
24: Mark e_1, e_2, e'_1, e'_2 as "affected"
25: **end if**
26: **end if**
27: **end for**
28: **for each** $e = (u, v) \in E$ **do** ▷ Parallel for
29: **if** e labeled "delete" **then**
30: $u \leftarrow -1$; $v \leftarrow -1$ ▷ Short update
31: **end if**
32: **end for**
33: **end while**
34: $E \leftarrow StmCmp(E,$ "not deleted") ▷ Long update
35: Output E as k-truss edge list
36: **if** $E \neq \emptyset$ **then** $k \leftarrow k + 1$ and **goto** 2

first mark all edge available to be affected and un-deleted

```
mark<<<dimGrid, dimBlock>>>(edgeSrcDevice, affectedDevice, e_affDevice, to_deleteDevice, numEdges);
```

select e_aff out of affected

```
selectAff<<<dimGrid, dimBlock>>>(e_aff.data(), affected.data(), currEdges, lengthDevice);
cudaDeviceSynchronize();
```

mark all e as "not affected"

```
markAll<<<dimGrid, dimBlock>>>(affectedDevice, numEdges, -1);
```

check all affected edges

```
checkAffectedEdges<<<dimGrid, dimBlock>>>(
        affected.data(), affected.size(), edgeSrcDevice, edgeDstDevice,
        to_delete.data(), e_aff.data(),
        rowPtrDevice, k, triangleCounts.data());
```

Short update

```
shortUpdate<<<dimGrid, dimBlock>>>(edgeSrcDevice, edgeDstDevice, to_delete.data(), currEdges);
```

No more long update, count current number of edges to determine k-truss availability

```
countEdges<<<dimGrid, dimBlock>>>(edgeSrcDevice, numEdges, edgeCountDevice);
```

Results

```
int numEdges = 16;
int edgeSrc[16] = {0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4};
int edgeDst[16] = {1, 2, 0, 2, 3, 4, 0, 1, 3, 4, 1, 2, 4, 1, 2, 3};
int rowPtr[6] = {0, 2, 6, 10, 13, 16};
int numRow = 6;
```

```
For the first graph:
Order of truss is 3. Remaining # of edges is 16
Order of truss is 4. Remaining # of edges is 12
k-truss order for the first graph is: 4
```

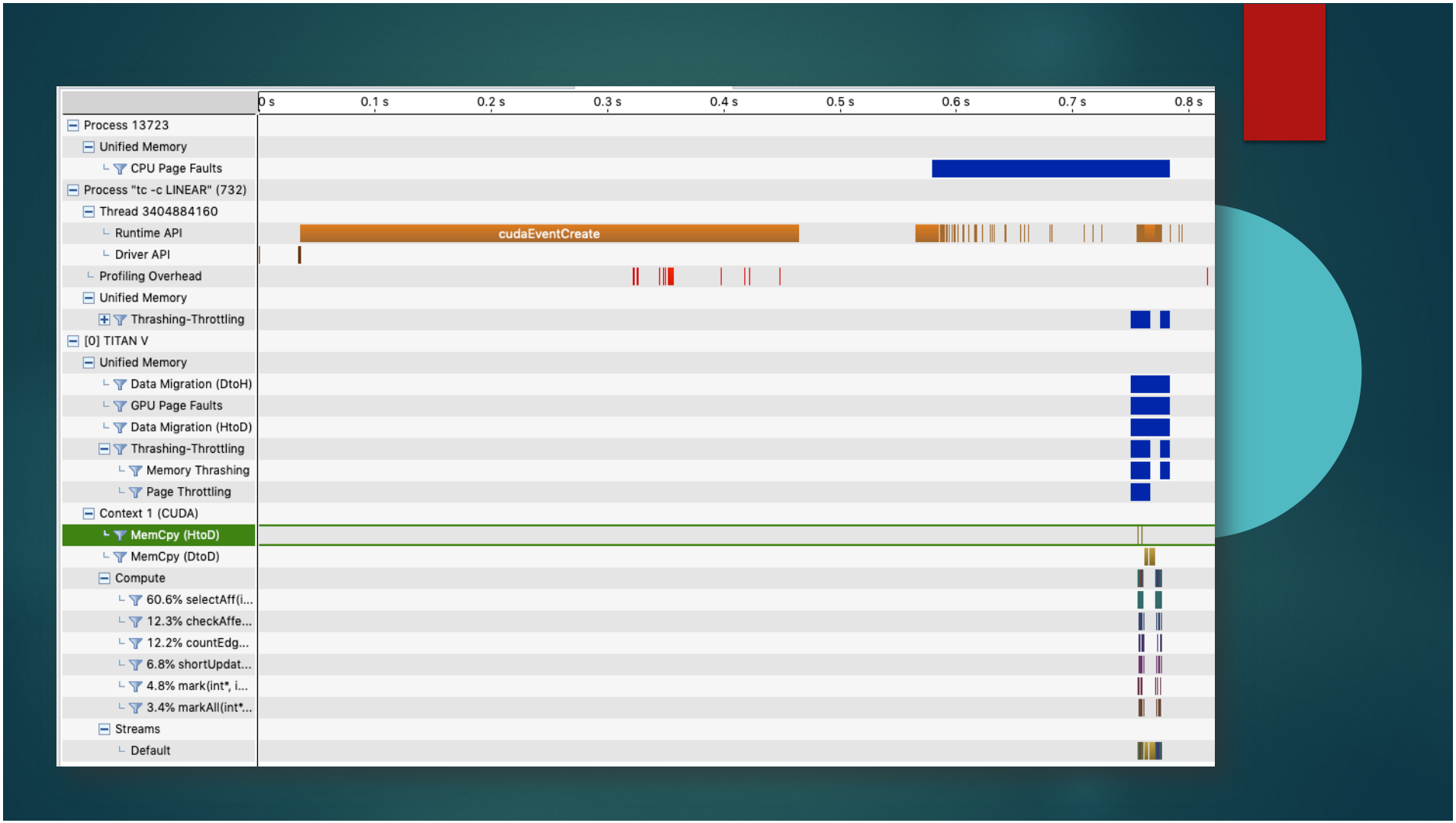
```
int numEdges2 = 30;
int edgeSrc2[30] = {0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 8, 8, 8};
int edgeDst2[30] = {1, 2, 7, 0, 3, 4, 0, 5, 6, 7, 1, 4, 8, 1, 3, 5, 8, 2, 4, 6, 2, 5, 7, 8, 0, 2, 6, 3, 4, 6};
int rowPtr2[10] = {0, 3, 6, 10, 13, 17, 20, 24, 27, 30};
int numRow2 = 10;
```

```
For the second graph:
Order of truss is 3. Remaining # of edges is 24
k-truss order for the second graph is 3
```

```
int numEdges3 = view.nnz();
int * edgeSrc3 = const_cast<int*>(view.row_ind());
int * edgeDst3 = const_cast<int*>(view.col_ind());
int * rowPtr3 = const_cast<int*>(view.row_ptr());
int numRow3 = view.num_rows();
```

```
For the real input graph - California Road Network:
number of edges: 5533214
number of nodes: 1965207
Order of truss is 3. Remaining # of edges is 719620
Order of truss is 4. Remaining # of edges is 504
k-truss order for the input graph is 4
TIMER:: do k truss decomposition took 18.468191ms
```

Performance analysis

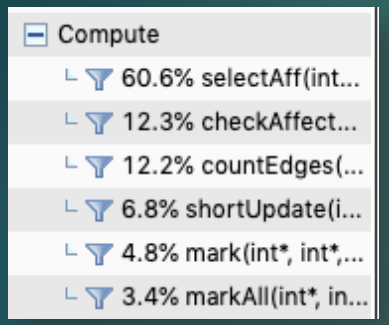

0 s
0.1 s
0.2 s
0.3 s
0.4 s
0.5 s
0.6 s
0.7 s
0.8 s
Process 13723
Unified Memory
CPU Page Faults
Process "tc -c LINEAR" (732)
Thread 3404884160
Runtime API
cudaEventCreate
Driver API
Profiling Overhead
Unified Memory
Thrashing-Throttling
[0] TITAN V
Unified Memory
Data Migration (DtoH)
GPU Page Faults
Data Migration (HtoD)
Thrashing-Throttling
Memory Thrashing
Page Throttling
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoD)
Compute
60.6% selectAff(i...
12.3% checkAffe...
12.2% countEdg...
6.8% shortUpdat...
4.8% mark(int*, i...
3.4% markAll(int*...
Streams
Default

MemCpy

MemCpy (HtoD)	
▼Duration	
Session	815.56148 ms (...
Memcpys	7.392 µs
Invocations	6
Total Bytes	432 B
Avg. Throughput	58.442 MB/s

MemCpy (DtoD)	
▼Duration	
Session	815.56148 ms (...
Memcpys	8.4775 ms (8,4...
Invocations	3
Total Bytes	52.127 MB
Avg. Throughput	6.149 GB/s

GPU Details



Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
markAll(int*, int, int)	14	15.556 µs	16	0	0
shortUpdate(int*, int*, int...	14	30.811 µs	16	0	0
mark(int*, int*, int*, int*, int)	8	37.931 µs	16	0	0
checkAffectedEdges(int*...	14	56.038 µs	25	0	0
countEdges(int*, int, int*)	8	97.055 µs	16	0	0
selectAff(int*, int*, int, int*)	22	175.646 µs	16	0	0

Properties

selectAff(int*, int*, int, int*)	
▼Duration	
Session	815.56148 ms (...
Kernel	3.86422 ms (3,...
Invocations	22
Importance	60.6%

checkAffectedEdges(int*, int, int*, int*, int*, int*, int*, int)		
▼Duration		
Session	815.56148 ms (...	
Kernel	784.538 µs	
Invocations	14	
Importance	12.3%	

countEdges(int*, int, int*)	
▼Duration	
Session	815.56148 ms (...
Kernel	776.443 µs
Invocations	8
Importance	12.2%

shortUpdate(int*, int*, int*, int)	
▼Duration	
Session	815.56148 ms (...
Kernel	431.356 µs
Invocations	14
Importance	6.8%

mark(int*, int*, int*, int*, int)	
▼Duration	
Session	815.56148 ms (...
Kernel	303.454 µs
Invocations	8
Importance	4.8%

markAll(int*, int, int)	
▼Duration	
Session	815.56148 ms (...
Kernel	217.789 µs
Invocations	14
Importance	3.4%

Q & A