TurboBC: A Memory Efficient and Scalable GPU Based Betweenness Centrality (BC) Algorithm in the Language of Linear Algebra

Oswaldo Artiles and Fahad Saeed

ICPP 2021 DUAC Workshop
August 9, 2021

FIU FLORIDA INTERNATIONAL UNIVERSITY

Content

TurboBC Algorithms

Experiments
Benchmarks

Experimental Results

TurboBC

The first implementation of a set of memory efficient Brandes' BC algorithms in the language of linear algebra, applicable to unweighted sparse graphs.

- ❑ Good performance
- ❑ High scalability

TurboBC

Optimization strategies

Exploiting the sparsity of the frontier and output vector of the top-down BFS algorithms

↓

Improving the performance of the BC algorithms

TurboBC

Optimization strategies

Minimizing the number of arrays on the GPU global memory

Reducing the memory footprint

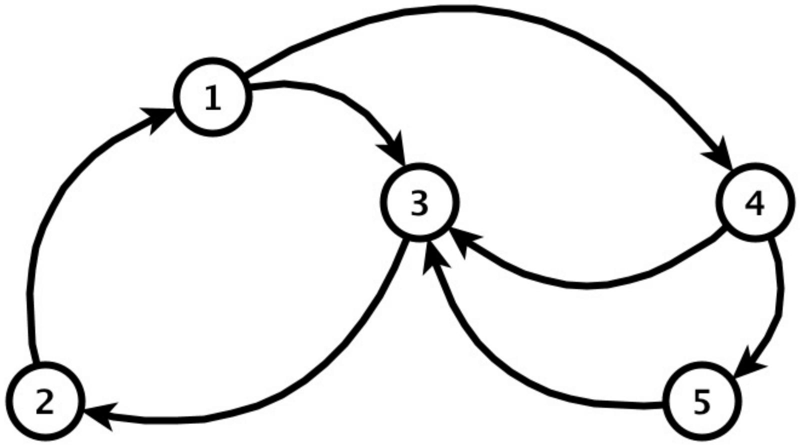
Increasing the memory efficiency and the scalability of the TurboBC algorithms

TurboBC applicable to:

Sparse unweighted graphs represented by binary sparse adjacency matrices in compressed sparse formats.

TurboBC

Sparse Compressed Formats



Sparse Adjacency Matrix

A

		1	1	
1				
	1			
		1		1
		1		

idx						
1	2	3	4	5	6	7

CSC format

row						
2	3	1	4	5	1	4

offset					
1	2	3	6	7	8

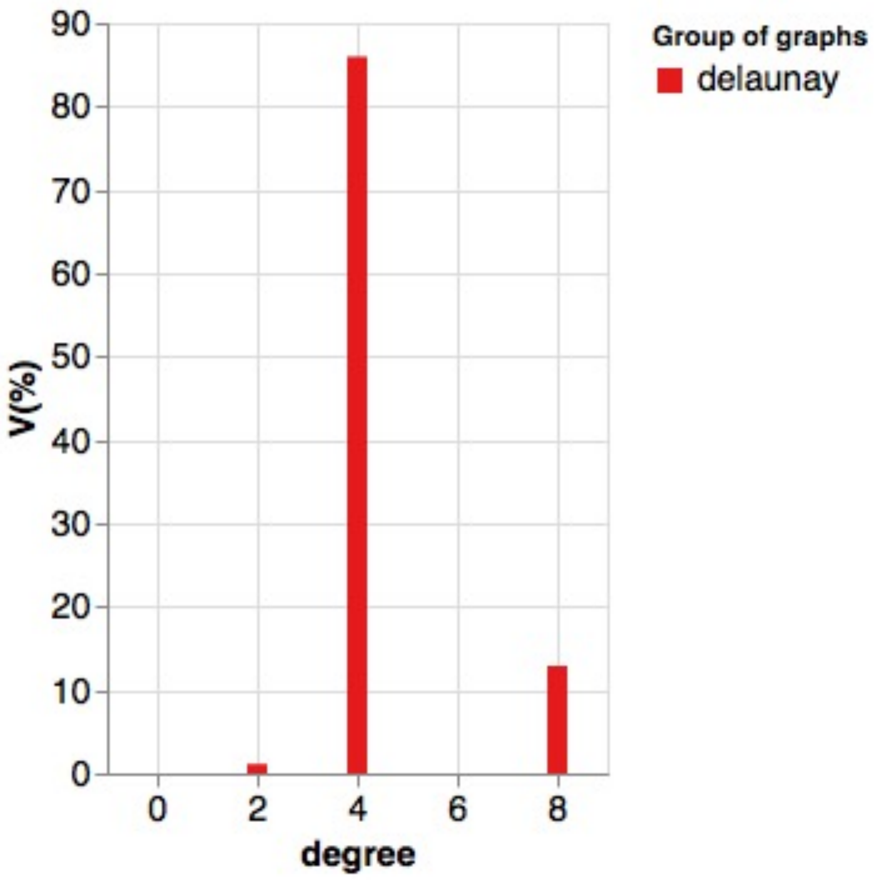
COOC format

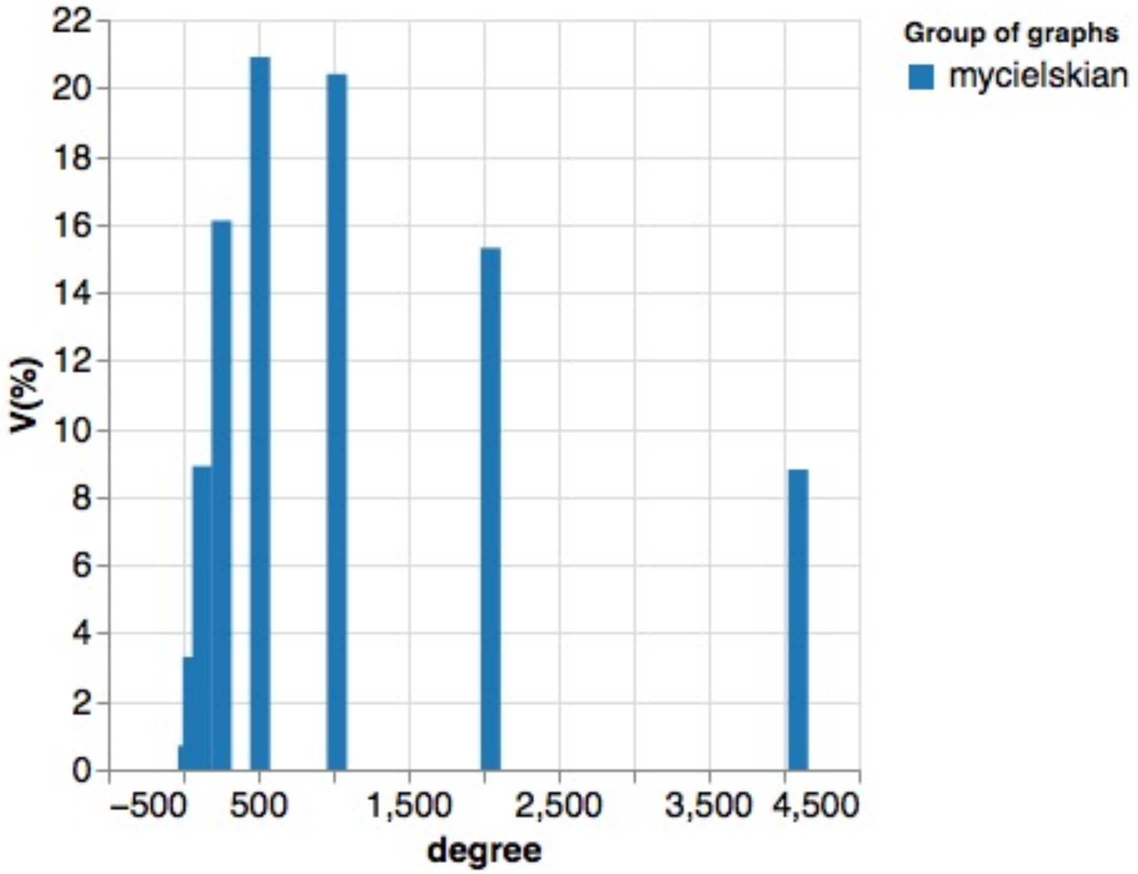
row						
2	3	1	4	5	1	4

col						
1	2	3	3	3	4	5

TurboBC

Regular and Irregular Graphs





TurboBC

Betweenness centrality of a vertex v of a graph G

$$BC(v) = \sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st} = \sum_{s \neq v \neq t} \delta_{st}(v)$$

One-sided dependences

$$\delta_s(v) = \sum_{t \in V} \delta_{st}(v)$$

Brandes' recurrence relation to compute the one-sided dependences

$$\delta_s(v) = \sum_{w:d(s,w)=d(s,v)+1} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_s(w))$$

TurboBC

Two-stages algorithm

Forward stage: Top-down BFS algorithm to compute the shortest paths/vertex

Backward stage: computation of one-sided dependences and update BC/vertex

TurboBC

Forward stage: Top-down BFS algorithm

```
for s ← 1, n do                         ▷ s: source vertex of BFS tree
    while f > 0 do                      ▷ BFS stage starts
        f_t ← A^T f
        if ∃σ(i) == 0 then
            f(i) ← f_t(i)
        end if
        if ∃f(i)! = 0 then
            S(i) ← d
            σ(i) ← σ(i) + f(i)
            c ← 1
        end if
    end while
end for
```

TurboBC

Backward stage: computation of one-sided dependences and update BC/vertex

for $s \leftarrow 1, n$ **do** ▹ s: source vertex of BFS tree
  while $d > 1$ **do** ▹ one-sided dependences vector stage
    if $S(i) == d$ and $\sigma(i) > 0$ **then**
      $\delta_u(i) \leftarrow (1.0 + \delta(i)) \div \sigma(i)$
    end if
    $\boldsymbol{\delta_{ut} \leftarrow A^T \delta_u}$
    if $S(i) == d - 1$ **then**
      $\delta(i) \leftarrow \delta(i) + \delta_{ut}(i) \times \sigma(i)$
    end if
    $d \leftarrow d - 1$
  end while
  for $v \leftarrow 1, n$ **do** ▹ updating betweenness centrality bc
    if $v \neq s$ **then**
      $bc(v) \leftarrow bc(v) + \delta(v)$
    end if
  end for
end for

TurboBC

Sparse Matrix-Vector Multiplication
Regular Graphs

One thread per edge
COOC-scalar
regular graphs

Sequential $f_t \leftarrow A^T f$ scalar operation
COOC format

for $k \rightarrow 1, nnz$ **do**
    if $f(row_A(k)) > 0$ **then**
        $f_t(col(k)) \leftarrow f_t(col(k)) + f(row(k))$

CSC format

for $v \rightarrow 1, n$ **do**
    if $\sigma(v) == 0$ **then**
        $\mathbf{sum} \leftarrow \mathbf{0}$
        $\mathbf{start} \leftarrow \mathrm{offset}(\mathbf{v})$
        $\mathbf{end} \leftarrow \mathrm{offset}(\mathbf{v}+\mathbf{1}) - \mathbf{1}$
        for $k \rightarrow start, end$ **do**
            $\mathbf{sum} \leftarrow \mathbf{sum} + \mathbf{f}(\mathbf{row}(\mathbf{k}))$
        end for
        if $sum > 0$ **then**
            $\mathbf{f_t}(\mathbf{v}) \leftarrow \mathbf{sum}$

One thread per vertex
CSC-scalar
Regular graphs

TurboBC

Sparse Matrix-Vector Multiplication

Warp Divergence

Irregular graphs: Nodes with higher degrees

↓

Load unbalance in warps

↓

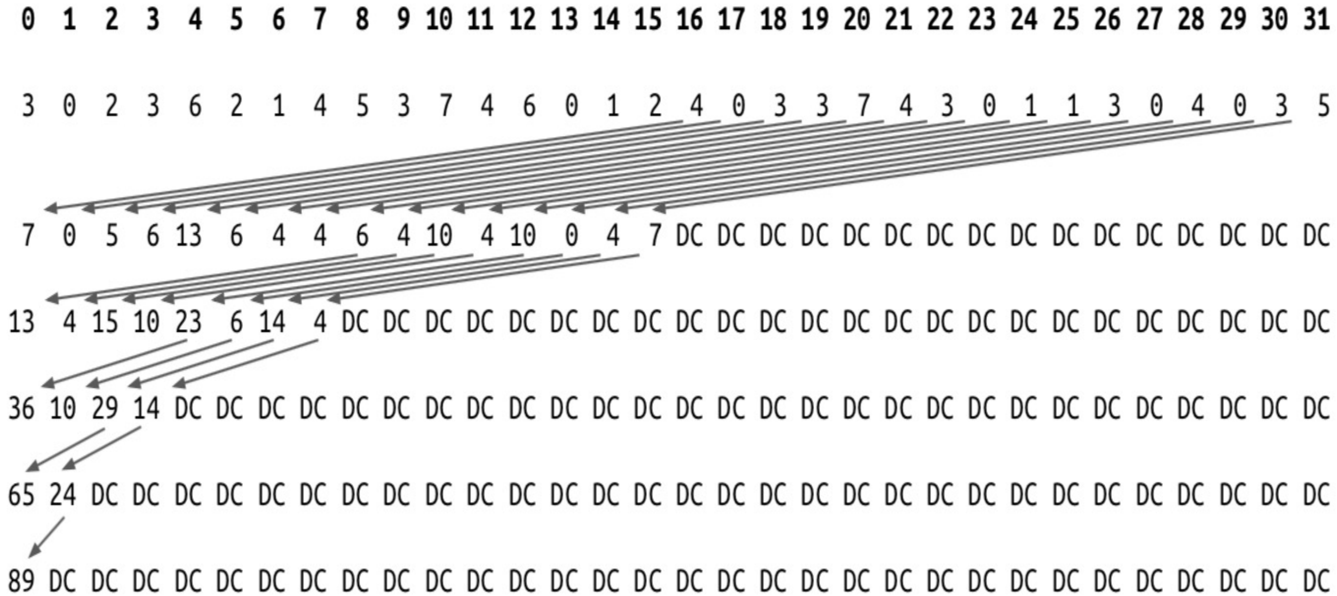
Warp divergence

TurboBC

Sparse Matrix-Vector Multiplication
Irregular Graphs

One Warp [32 Threads] per Vertex
Parallel reduction
CSC-vector

```
procedure veCSC-MVSp-Kernel(offset, row, f)
    if σ(col) == 0 then
        start ← offset(col)
        end ← offset(col + threadLane_id)
        while icp < end do
            sum ← sum + f(row_A(icp))
            icp ← icp + threadsPerWarp
        end while
        off ← threadsPerWarp/2
        while off > 0 do
            sum ← sum + shfl − down − sync(off)
            off ← off2
        end while
        if threadLane_id == 0 then
            f_t(warp_id) ← sum
        end if
```



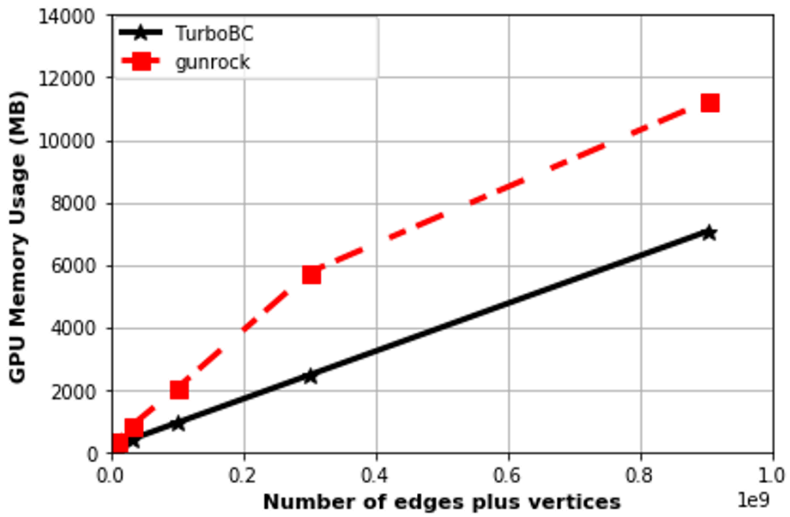
TurboBC

Experiments Benchmarks
Graphs and Graphs Analytic Libraries

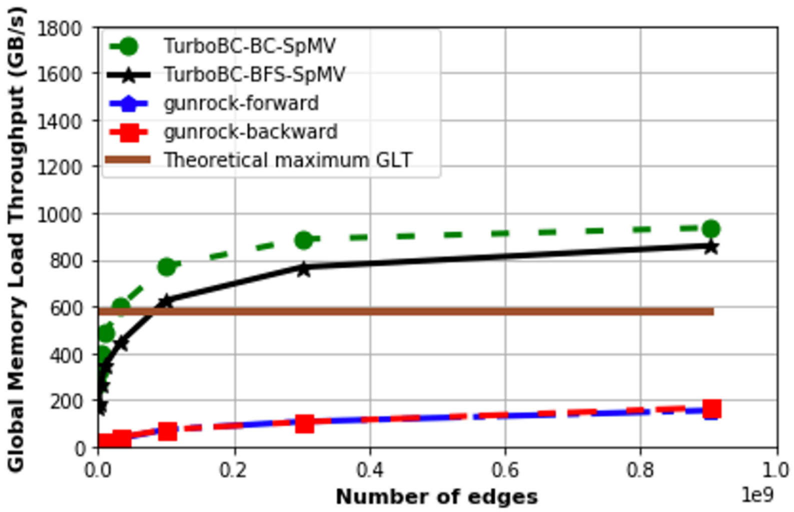
- Thirty-three graphs represented by sparse adjacency matrices from the SuiteSparse Matrix Collection, with up to 1.9 billion edges and 214 million vertices.
- gunrock: High-performance GPU-based graph analytic library
- ligra: High-performance CPU-based graph analytic library

TurboBC

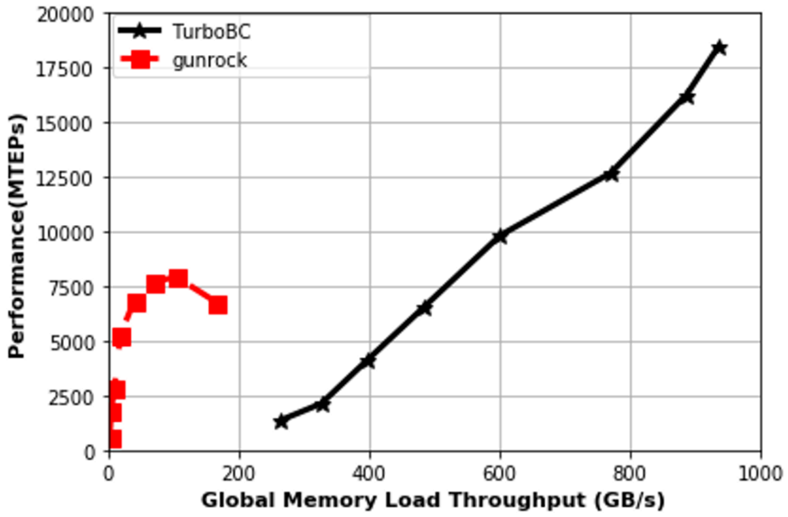
GPU Memory Efficiency: TurboBC vs gunrock



a) GPU memory usage



b) Global Memory Load Throughput (GB/s)

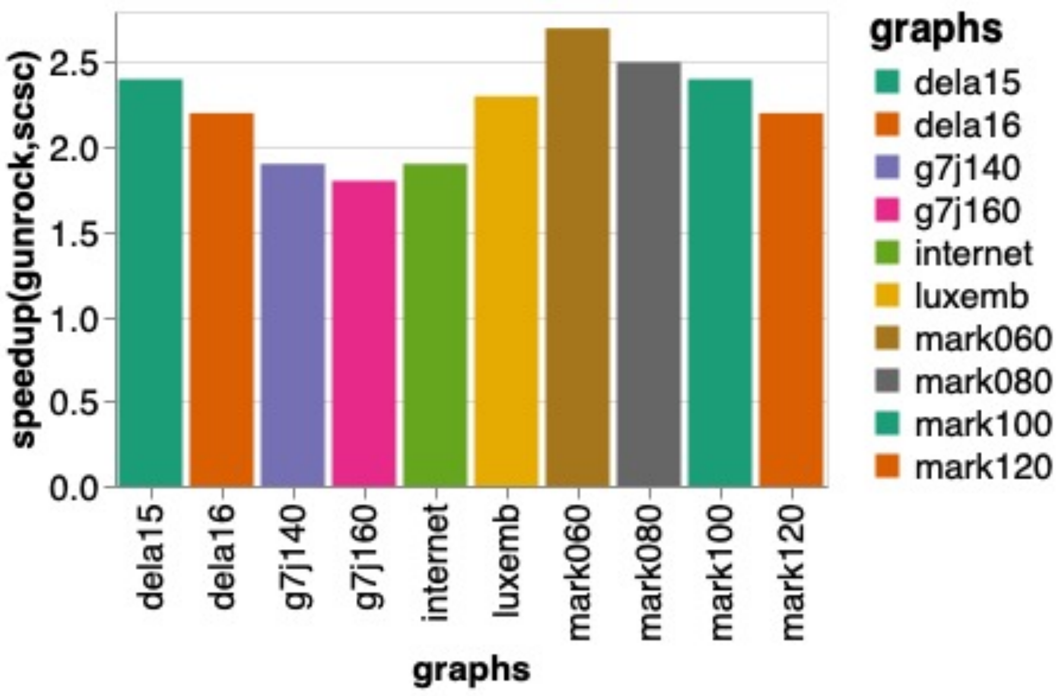


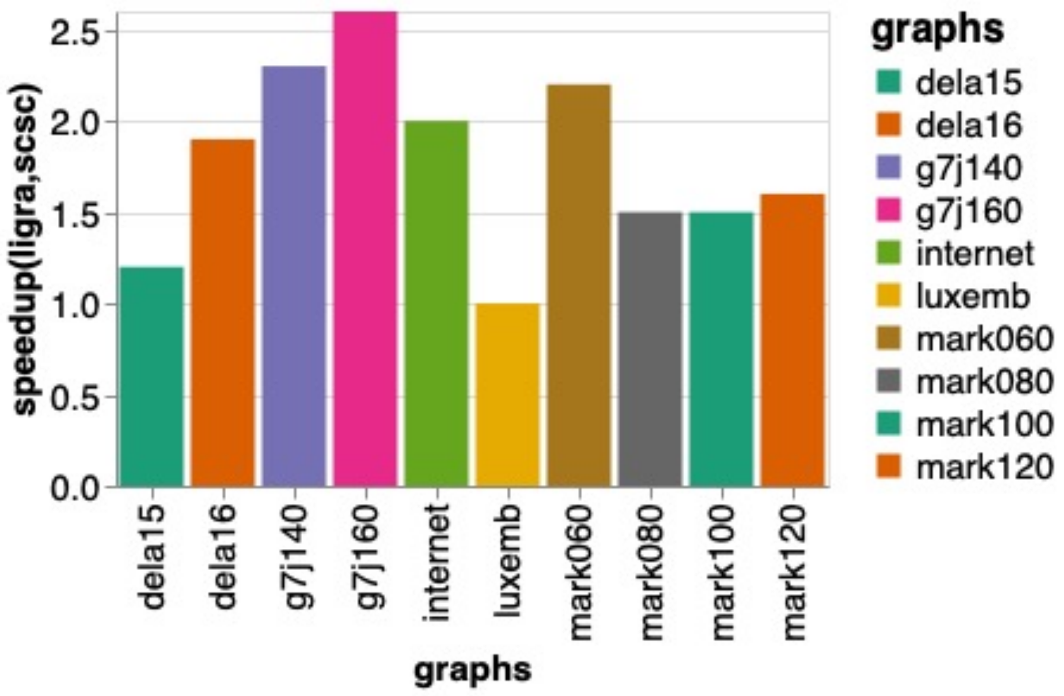
c) Performance(MTEPs)

TurboBC

Experimental Results for Regular Graphs

- Ten graphs → TurboBC CSC-scalar
- Up to 470 MTEPs (Million transverse edges/second)

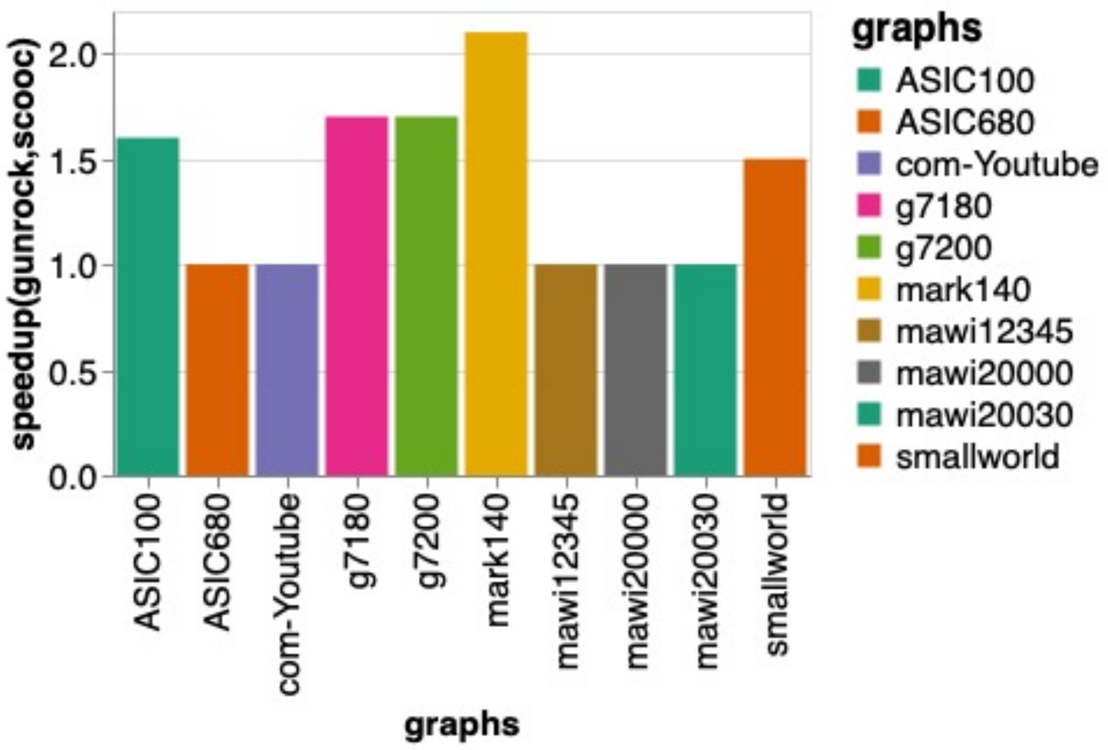


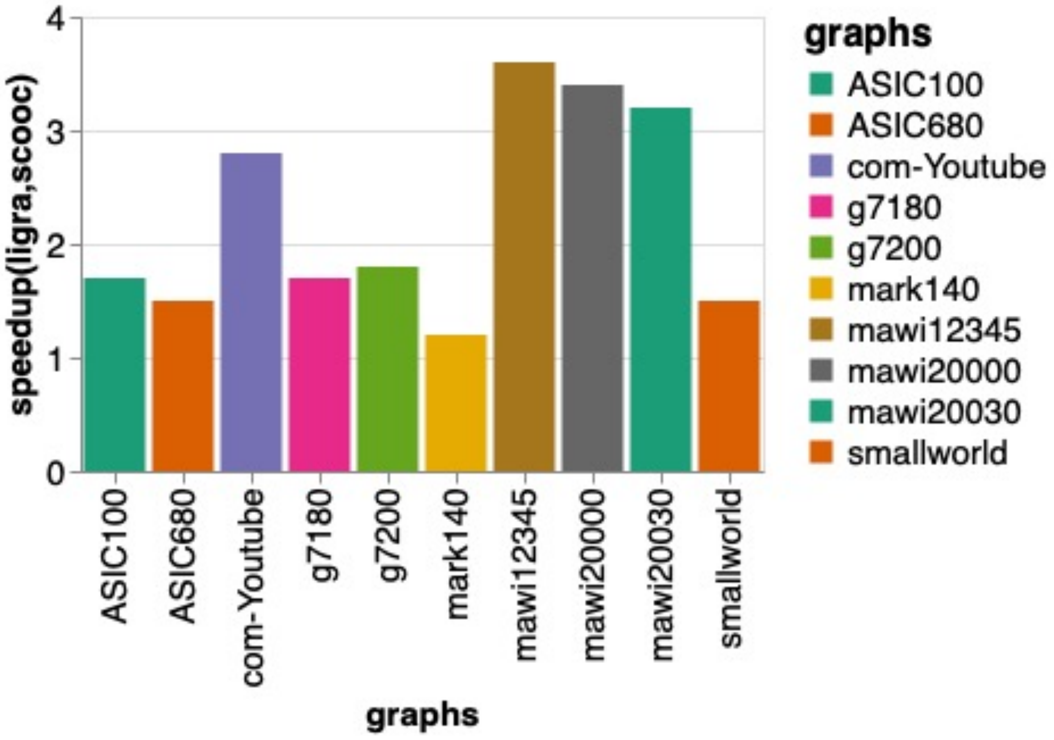


TurboBC

Experimental Results for Regular Graphs

- ❖ **Ten graphs** → **TurboBC COOC-scalar**
- ❖ **Up to 1000 MTEPs (Million transverse edges/second)**

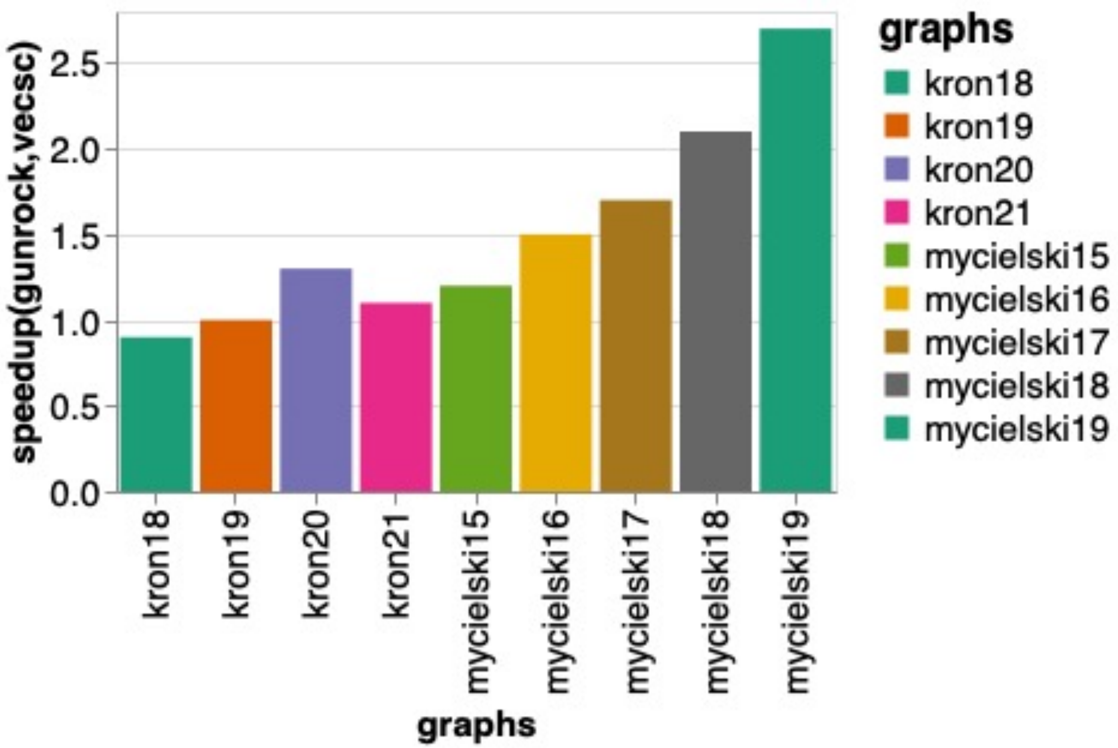


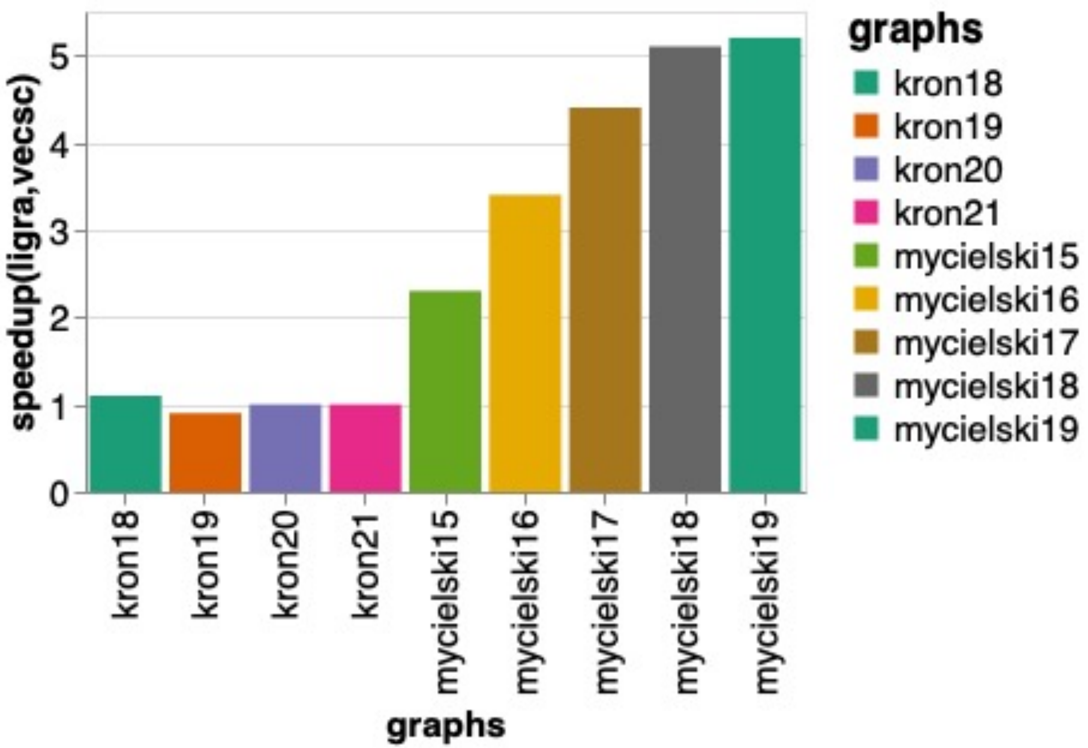


TurboBC

Experimental Results for Irregular Graphs

- ❖ **Nine graphs** ➡ **TurboBC CSC-vector**
- ❖ **Up to 18470 MTEPs (Million transverse edges/second)**





TurboBC

Experimental Results for Big Graphs

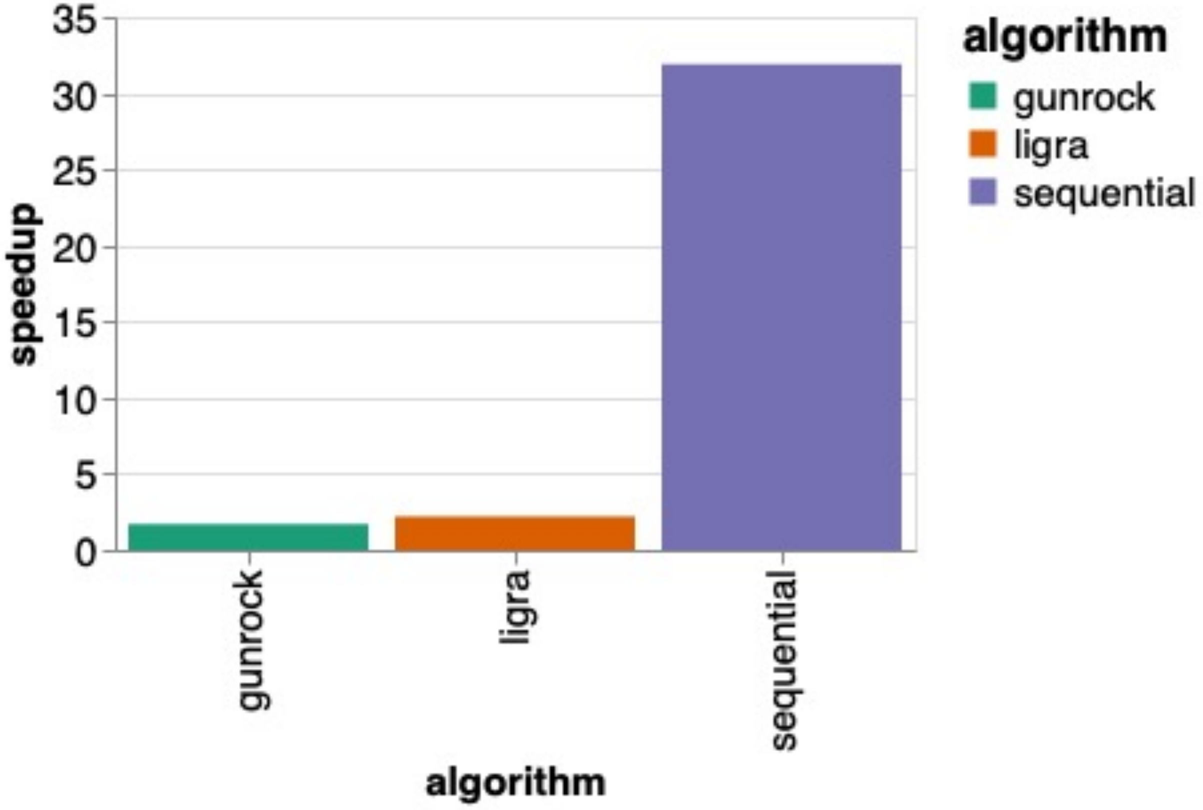
File	$n{\times}10^6$	$m{\times}10^6$	degree($max/\mu/\sigma$)	d	scf	runtime(s)	MTEPs	(sequential)x	(gunrock)x
kmer-V1r(U)	214	465	8/2/1	324	2	14.3	33	94.5	OOM
it-2004(D)	42	1151	9964/28/67	50	543	3.1	371	39.5	OOM
GAP-twitter(D)	62	1469	3×10^6/24/1990	15	126	7.3	201	50.4	OOM
sk-2005(D)	51	1950	12870/39/78	54	1262	6.8	287	30.5	OOM

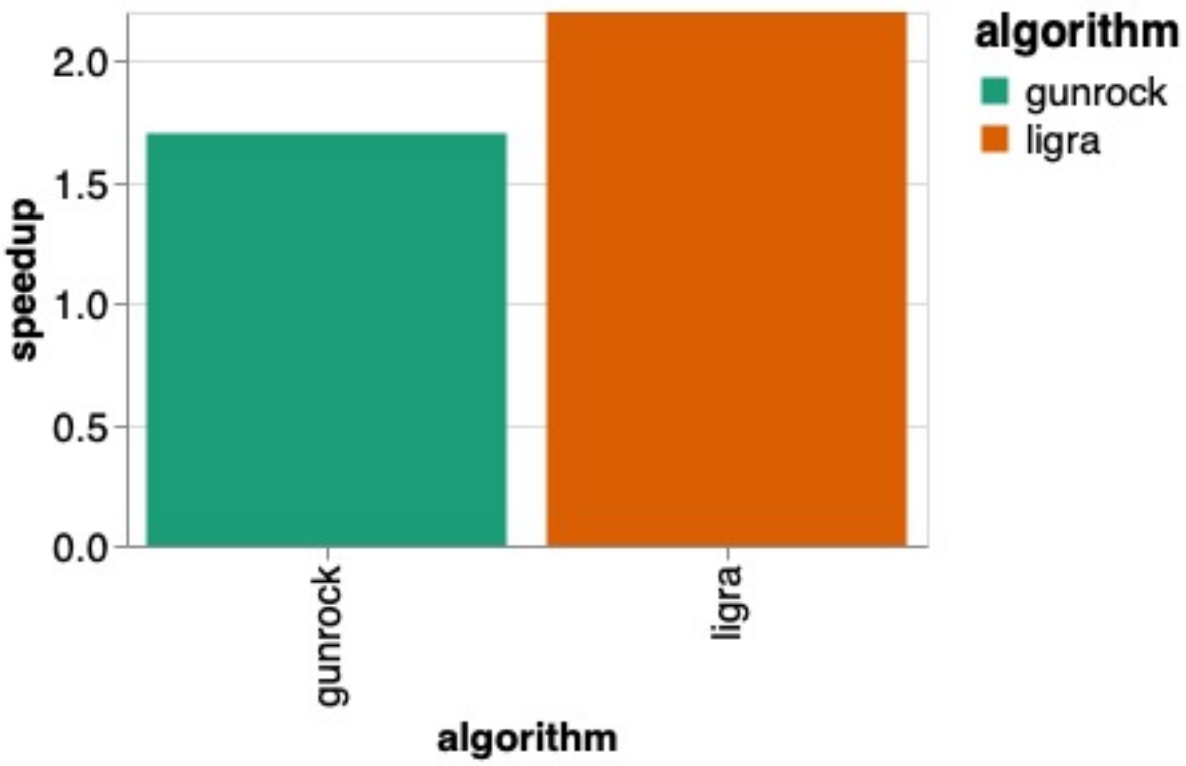
The BC algorithms available in the gunrock libraries ran out of memory for these big graphs

High scalable TurboBC algorithms

TurboBC

Experimental Results Summary





TurboBC main result

A memory efficient and highly scalable first implementation of GPU-based set of Brandes' BC algorithms in the language of linear algebra.

Thank you

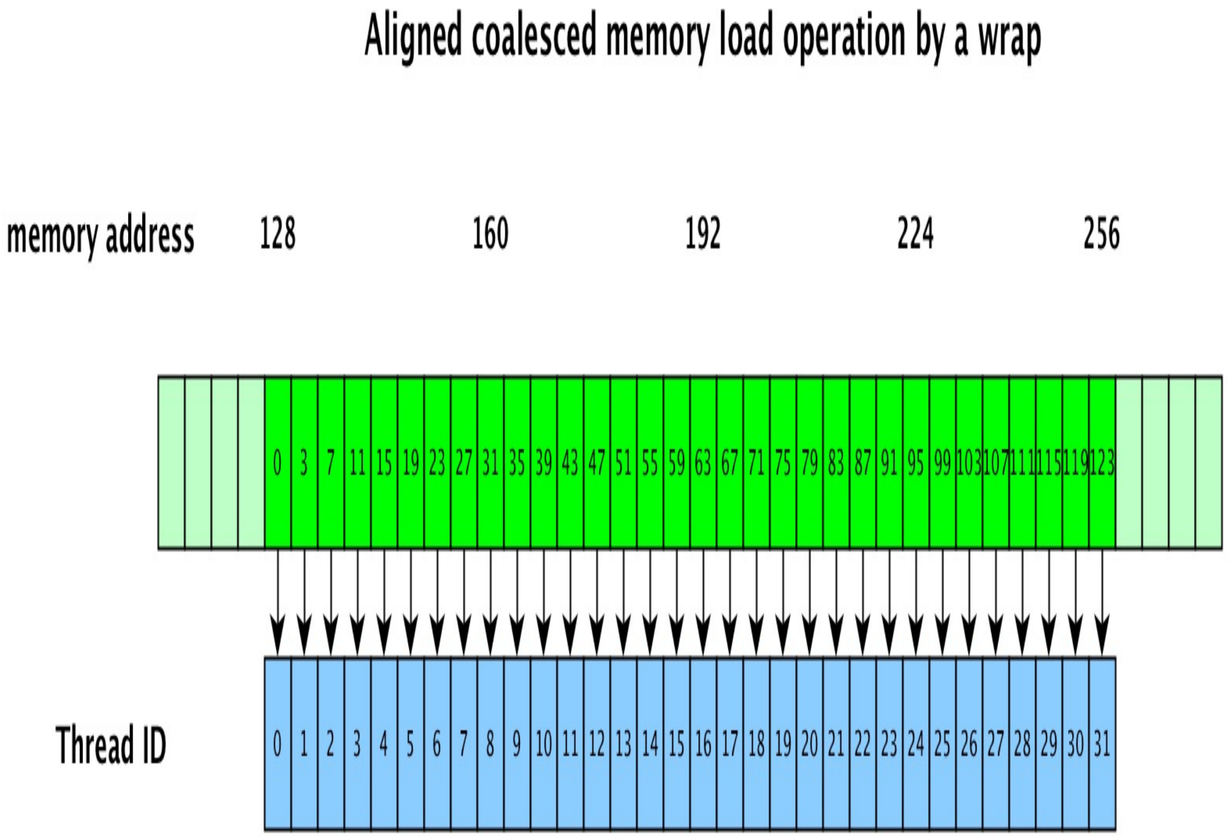
Questions?

TurboBFS
Experimental CPU-GPU Platform

- Linux server with Ubuntu operating system version 16.04.6, 22 Intel Xeon Gold 6152 processors, clock speed 2.1 GHz, and 125 GB of RAM.
- The GPU in this server was a NVIDIA Titan Xp, with 30 SM, 128 cores/SM, maximum clock rate of 1.58 GHz, 12196 MB of global memory, and CUDA version 10.1.243 with CUDA capability of 6.1.

TurboBFS
Sparse Matrix-Vector Multiplication
Warp Memory Load Operation

- **Aligned memory access** occurs when the first address of a device memory transaction is an even multiple of 32 bytes for L2 cache or 128 bytes for L1 cache.
- **Coalesced memory access** occurs when all 32 threads in a warp access a contiguous chunk of memory..
- **Aligned coalesced memory access** is ideal because it maximizes global load memory throughput. That is, the addresses requested by all threads in a warp fall within one cache line of 128 bytes. Only a single 128-byte transaction is required by the memory load operation.



TurboBFS

Sparse Matrix-Vector Multiplication Warp Memory Load Operation

- When the L1 cache is enabled, three 128-byte memory transactions may be required, resulting in wasted memory bandwidth because some of the bytes loaded are not used.
- Misaligned accesses can be verified by collecting information of the Global Memory Load Efficiency (GMLE) metrics