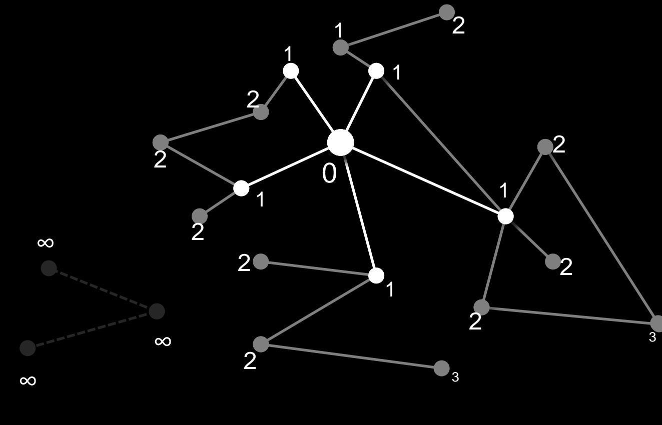
Basic Parallel Graph Algorithms

Duane Merrill (NVIDIA)
Michael Garland (NVIDIA)

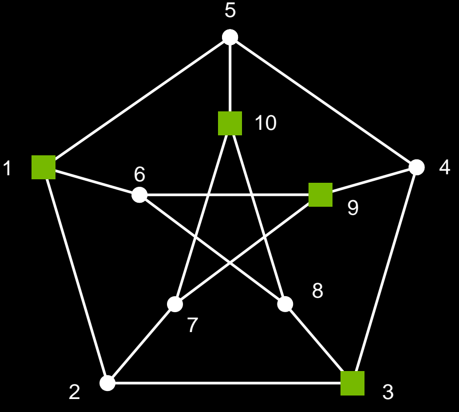
GPU TECHNOLOGY CONFERENCE

Agenda

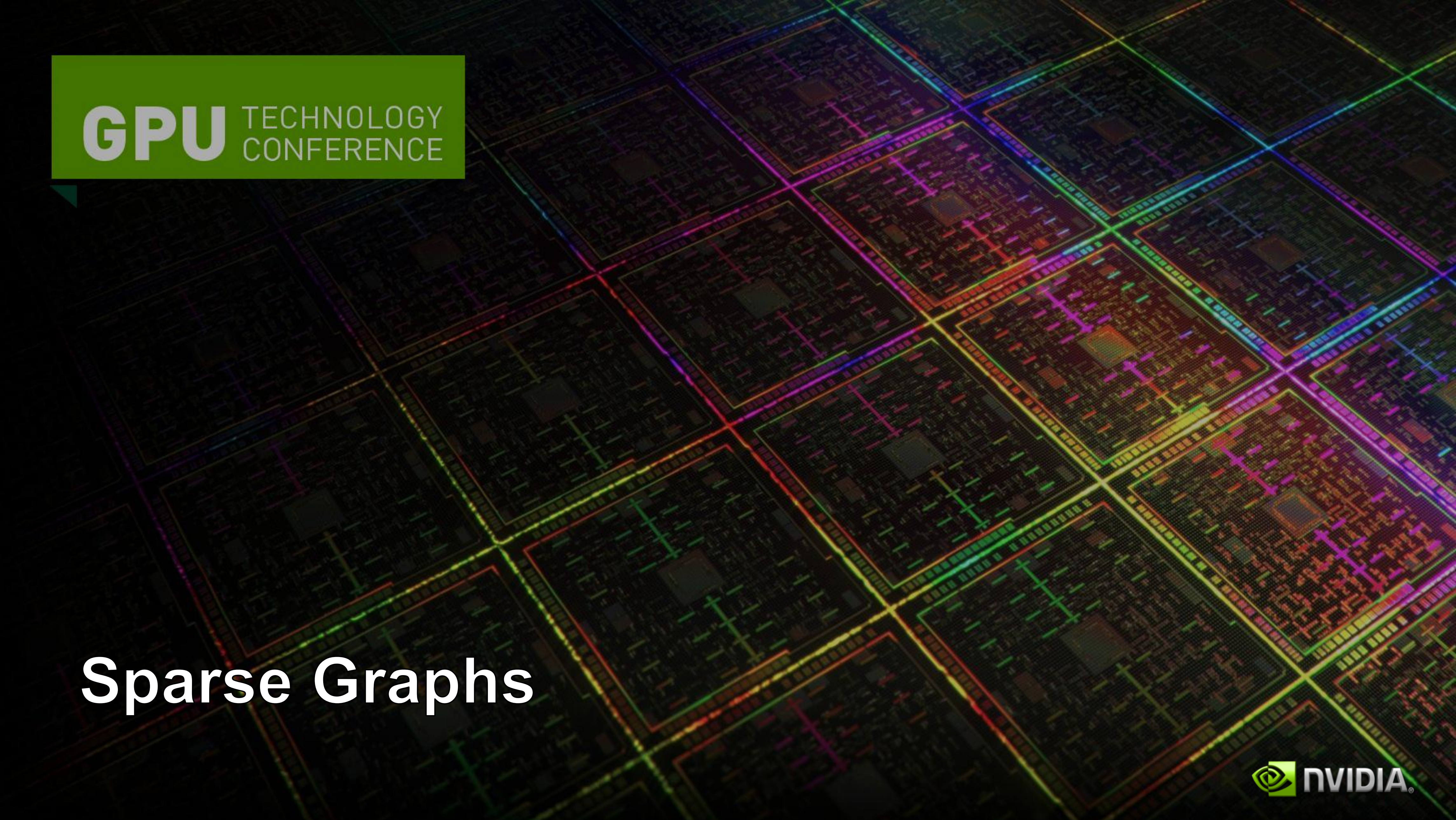
- Sparse graphs
- Breadth-first search (BFS)
- Maximal independent set (MIS)



BFS



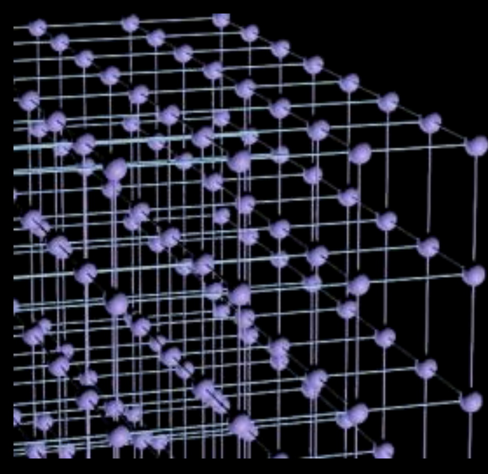
MIS


GPU TECHNOLOGY CONFERENCE
Sparse Graphs
NVIDIA

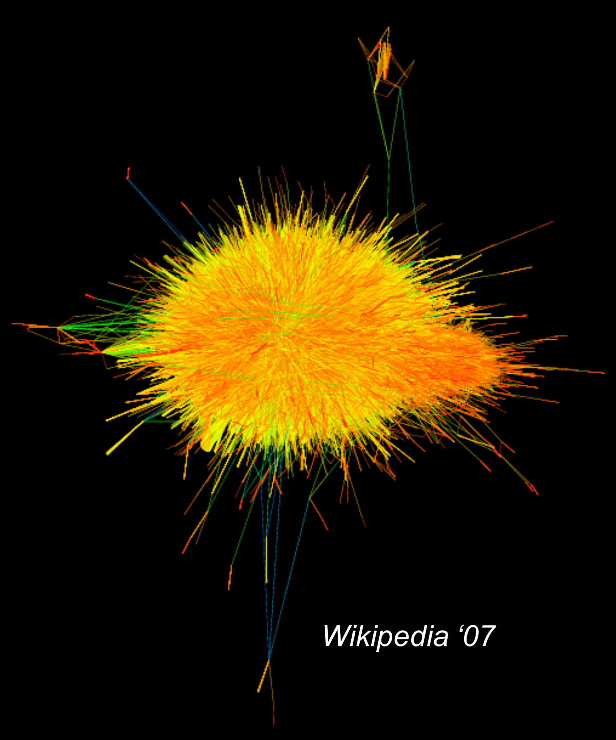
GPU TECHNOLOGY CONFERENCE

Sparse graphs

- Graph $G(V, E)$ where $n = |V|$ and $m = |E|$
- Convey structure via relationships



3D lattice

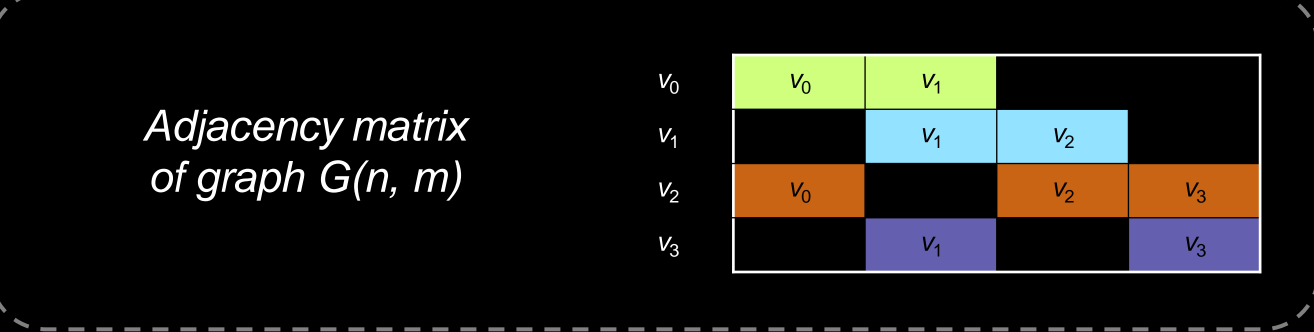


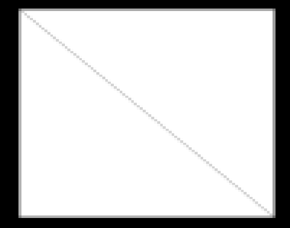
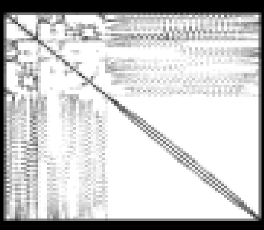
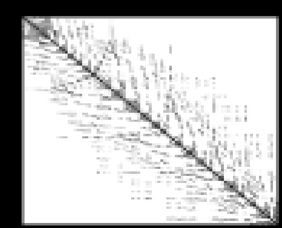
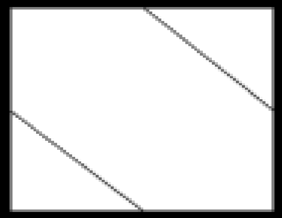
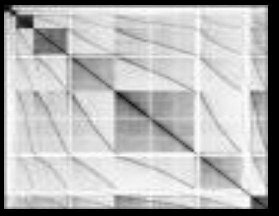
Wikipedia '07

Gleich@wikipedia-20070206. 3512462 nodes, 42374383 edges.

GPU TECHNOLOGY CONFERENCE

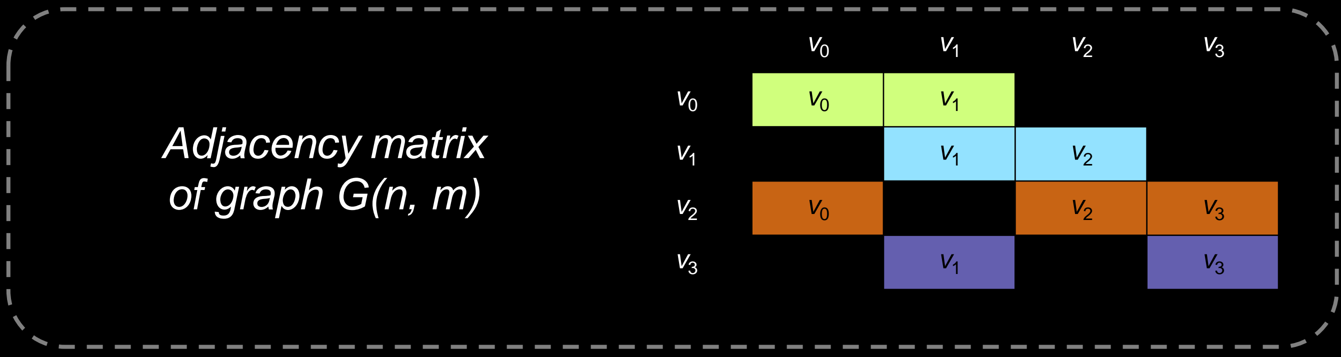
Graph sparsity

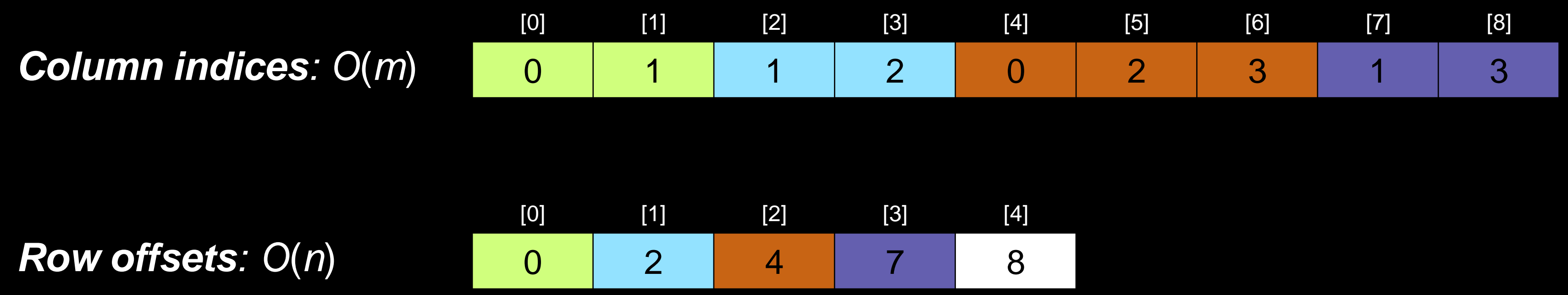


"Spy" plots

Compressed sparse row (CSR) representation

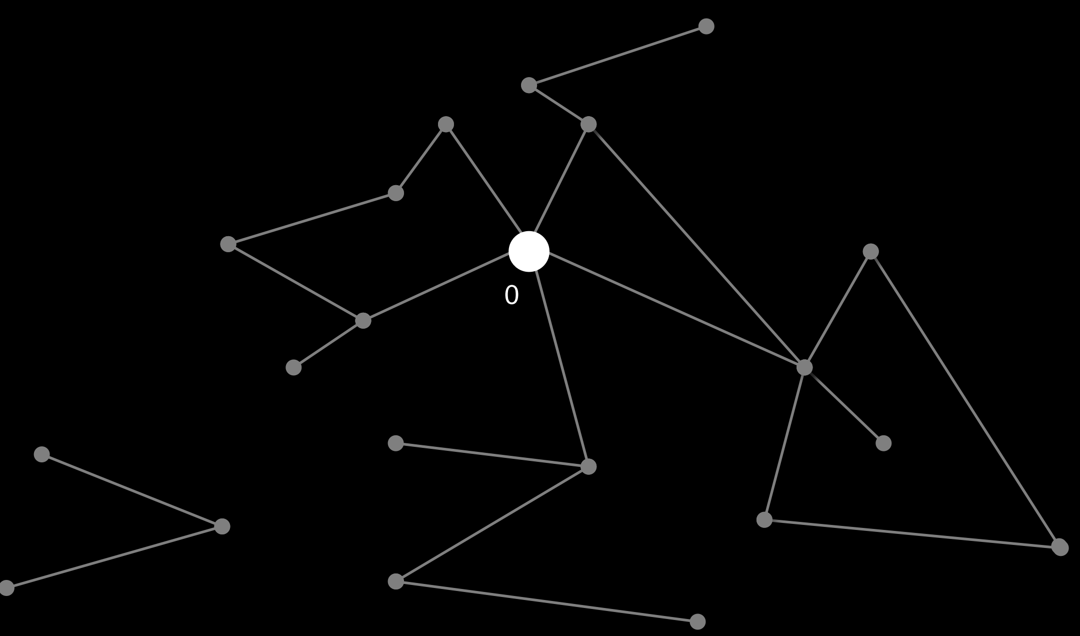




GPU TECHNOLOGY CONFERENCE
Breadth-first search
NVIDIA

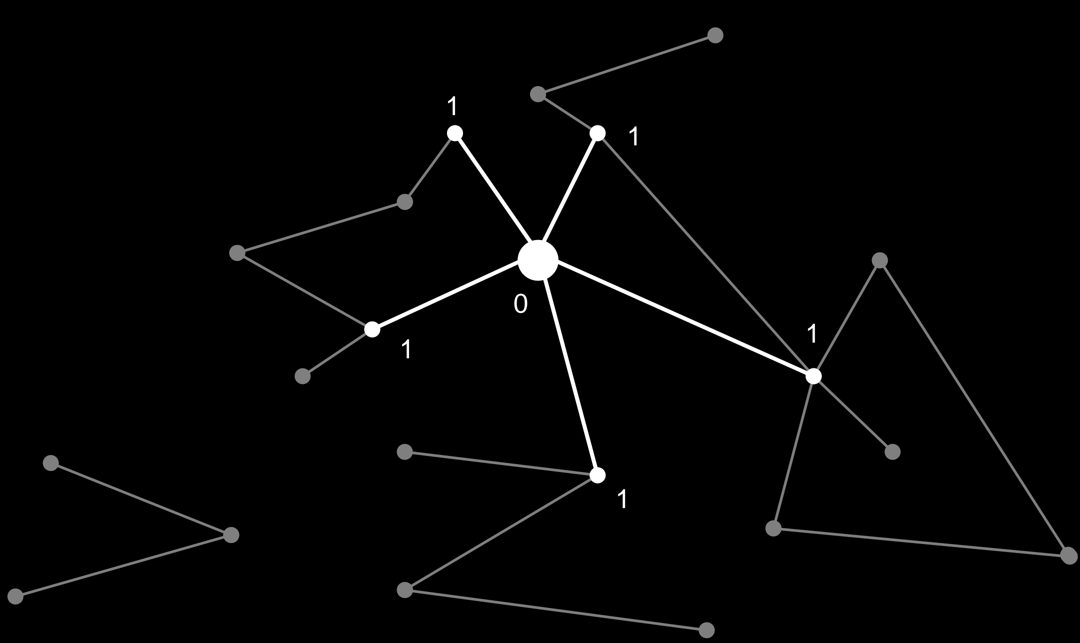
Breadth-first search (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source



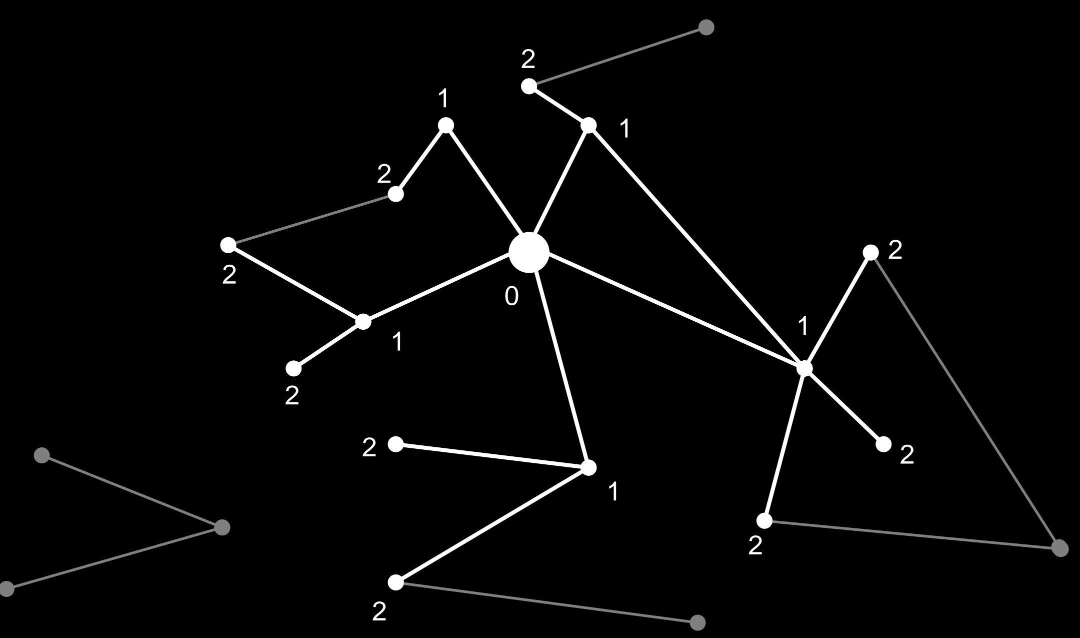
Breadth-first search (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source



Breadth-first search (BFS)

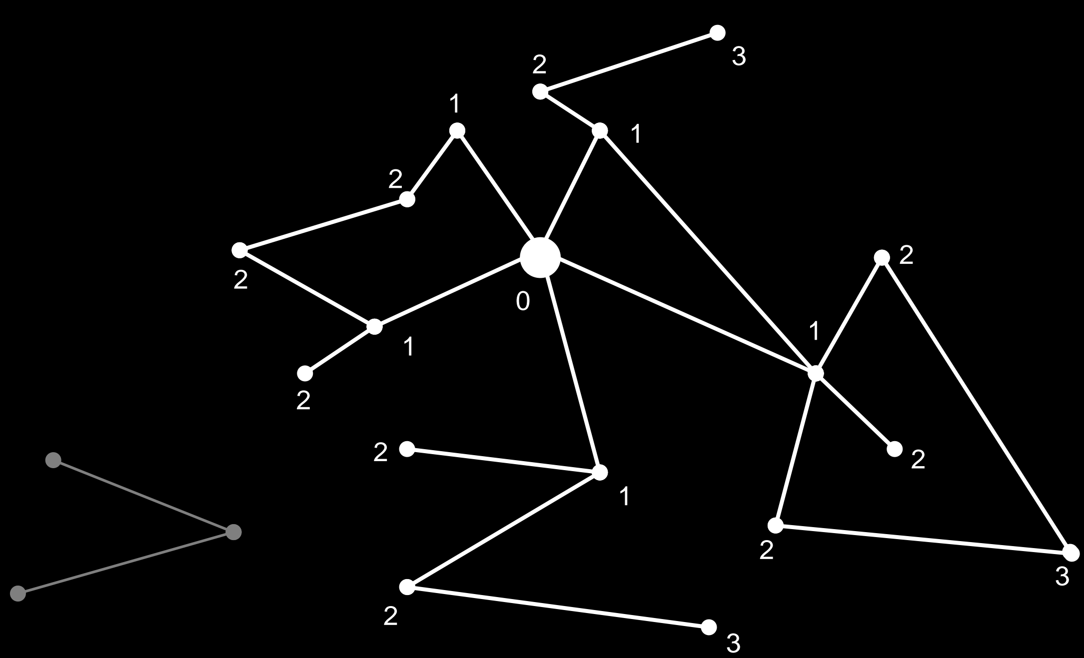
1. Pick a source node
2. Rank every vertex by the length of shortest path from source



GPU TECHNOLOGY CONFERENCE

Breadth-first search (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source



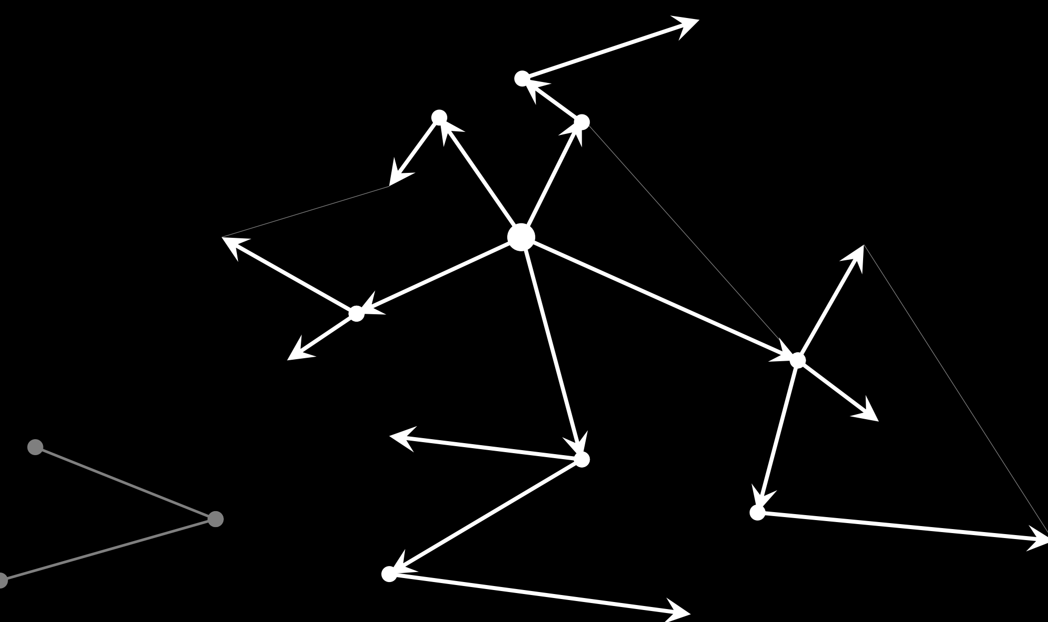
GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

Breadth-first search (BFS)

1. Pick a source node
2. Rank every vertex by the length of shortest path from source

(or by predecessor vertex)

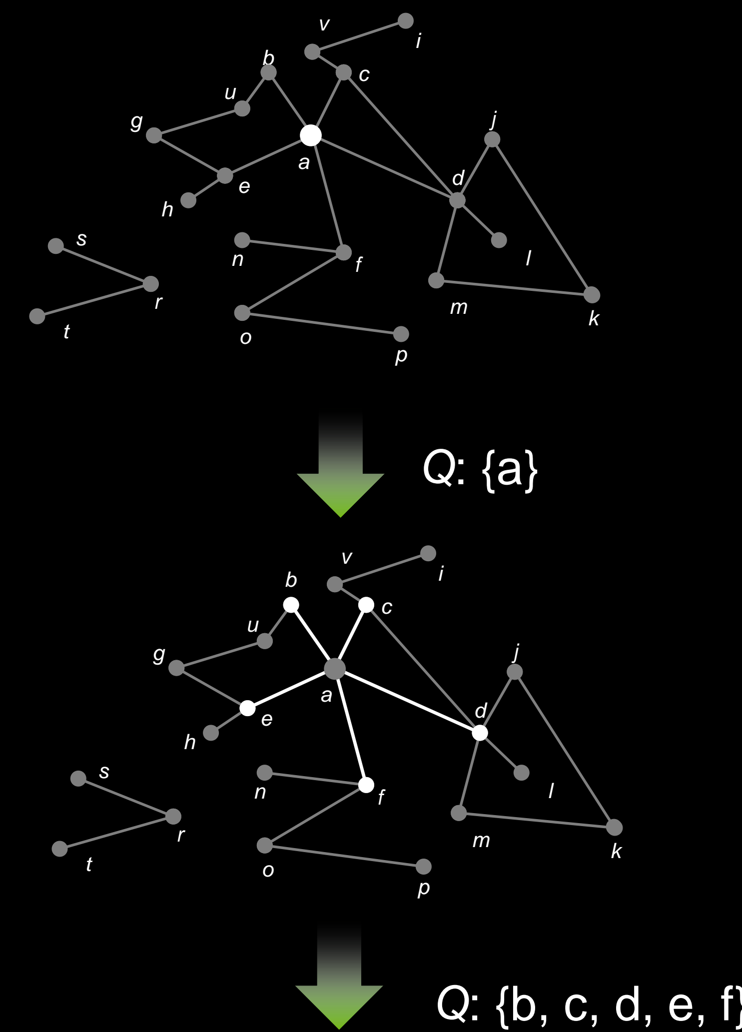


BFS applications

- Algorithmic primitive
  - Reachability (e.g., garbage collection)
  - Belief propagation (statistical inference)
  - Finding community structure
  - Path-finding
- Core benchmark kernel
  - Graph 500
  - Rodinia
  - Parboil
- Simple performance-analog for many applications
  - Pointer-chasing
  - Work queues

The sequential algorithm

- Cycles newly-discovered-but-unvisited vertices through a queue
  1. Dequeue vertex *v*
  2. For all neighbors n_i of *v*:
     - $dist[n_i] = dist[v] + 1$
     - Enqueue n_i



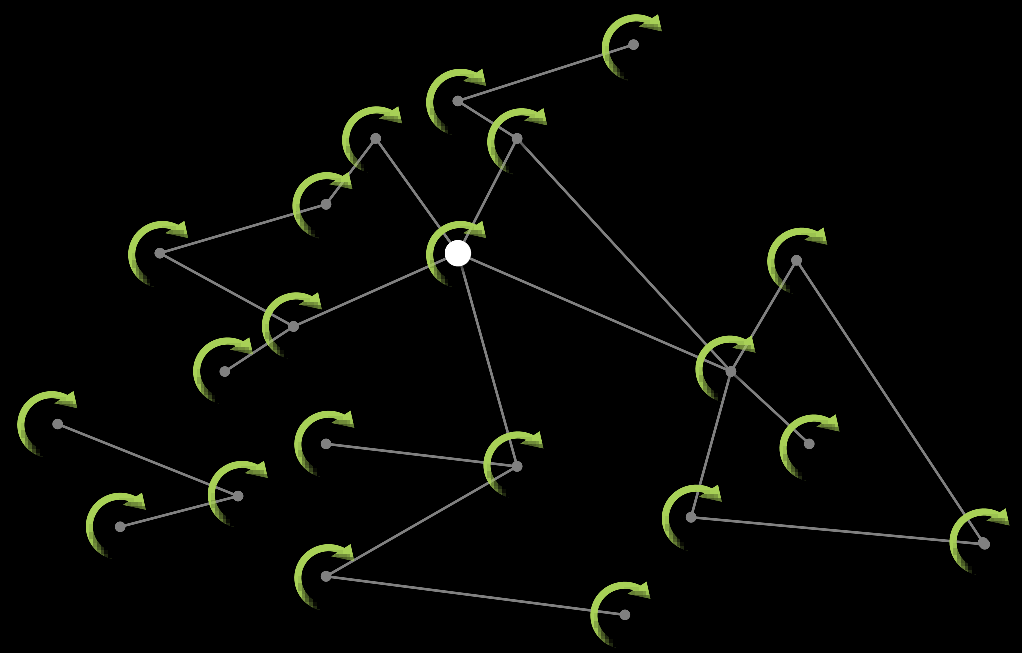
GPU TECHNOLOGY CONFERENCE

Common parallelization strategies

- *Level-synchronous*: parallel work done in “epochs” by level
  a) Quadratic work
  b) Linear work
- *Randomized*: parallel work done within independent sub-problems
  a) Ullman-Yannakakis

Quadratic-work approach

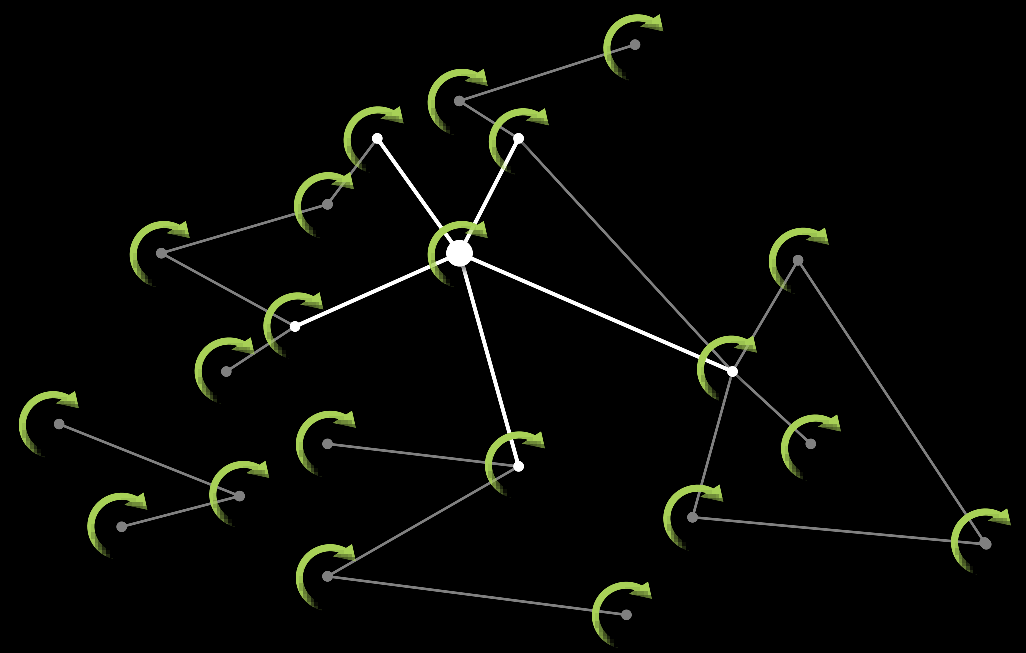
1. Inspect every vertex at every time step
   - E.g., one thread per vertex
2. If updated, update its neighbors
3. Repeat until no further updates



GPU TECHNOLOGY CONFERENCE

Quadratic-work approach

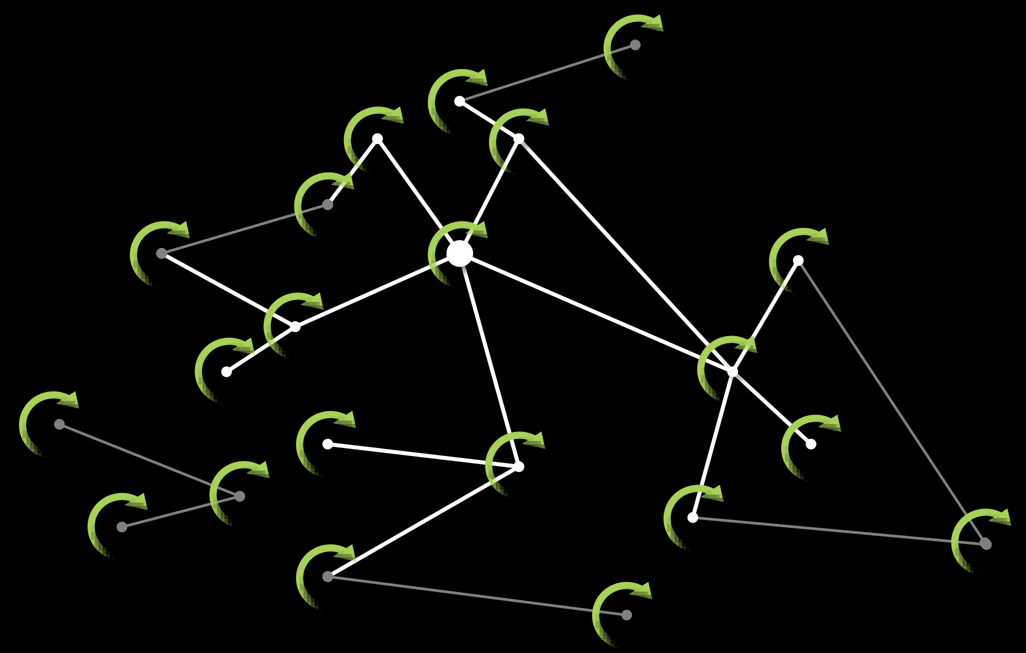
1. Inspect every vertex at every time step
   - E.g., one thread per vertex
2. If updated, update its neighbors
3. Repeat until no further updates



GPU TECHNOLOGY CONFERENCE

Quadratic-work approach

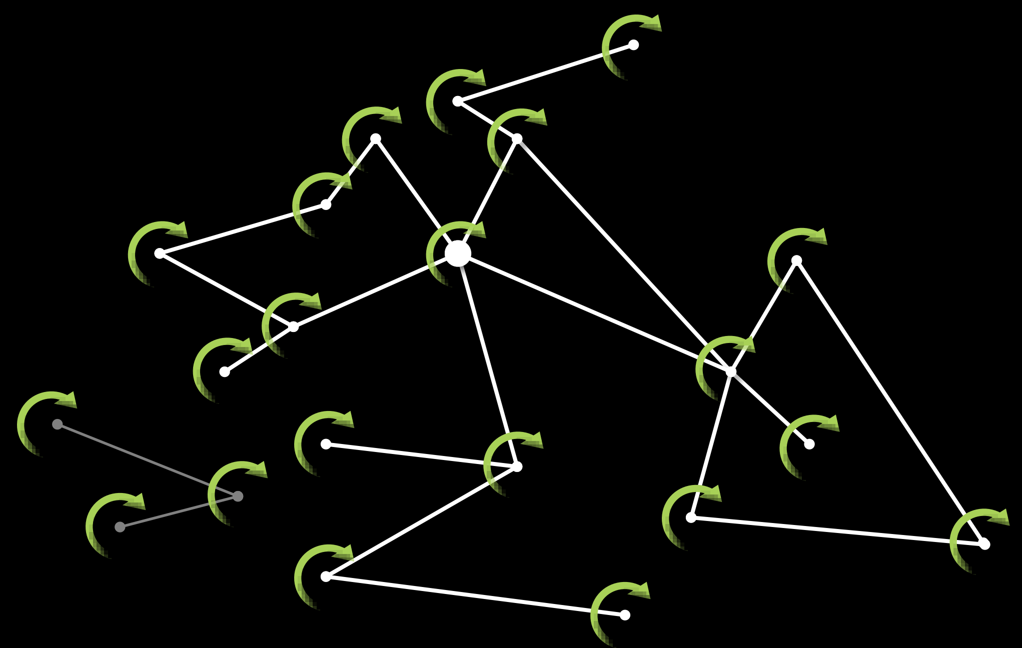
1. Inspect every vertex at every time step
   - E.g., one thread per vertex
2. If updated, update its neighbors
3. Repeat until no further updates



GPU TECHNOLOGY CONFERENCE

Quadratic-work approach

1. Inspect every vertex at every time step
   - E.g., one thread per vertex
2. If updated, update its neighbors
3. Repeat until no further updates



GPU TECHNOLOGY CONFERENCE

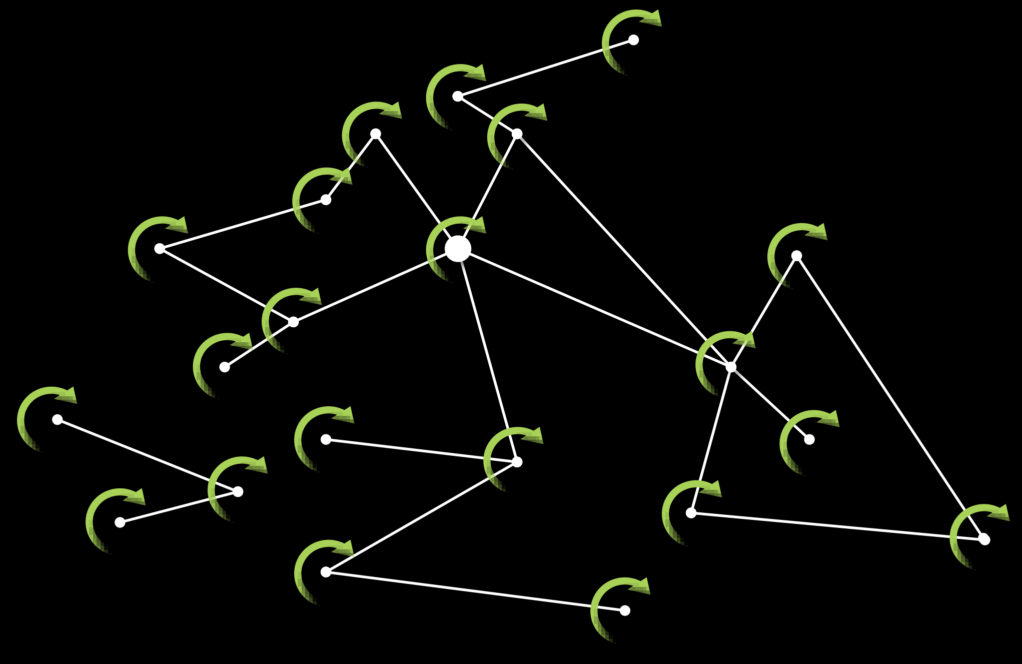
Quadratic-work approach

- **Input:** Vertex set *V*, row-offsets array *R*, column-indices array *C*, source vertex *s*
- **Output:** Array *dist*[0..*n*-1] with *dist*[*v*] holding the distance from *s* to *v*

```
 1.  parallel for (i in V) :
 2.    dist[i] := ∞
 3.  dist[s] := 0
 4.  iteration := 0
 5.  do :
 6.    done := true
 7.    parallel for (i in V) :
 8.      if (dist[i] == iteration)
 9.        done := false
10.        for (offset in R[i] .. R[i+1]-1) :
11.          j := C[offset]
12.          dist[j] = iteration + 1
13.    iteration++
14.  while (!done)
```

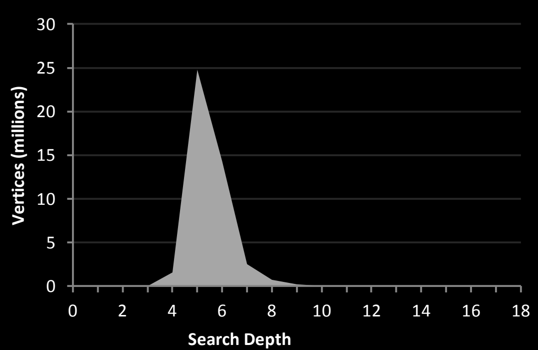
kernel (line 2)

kernel (lines 8–12)

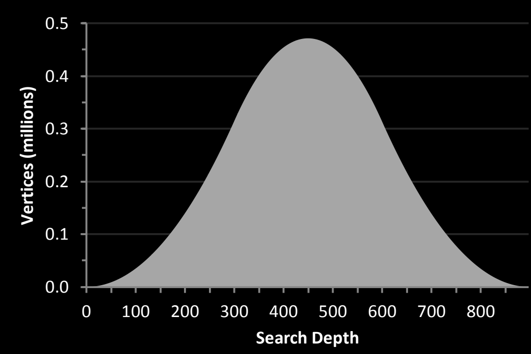


- ***Good***: trivial data-parallel GPU stencils
  - Harish et al., Deng et al. Hong et al
- ***Bad***: worst-case quadratic $O(n^2+m)$ work

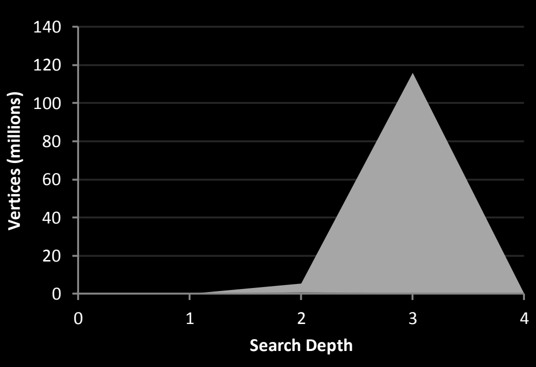
Active edges per iteration



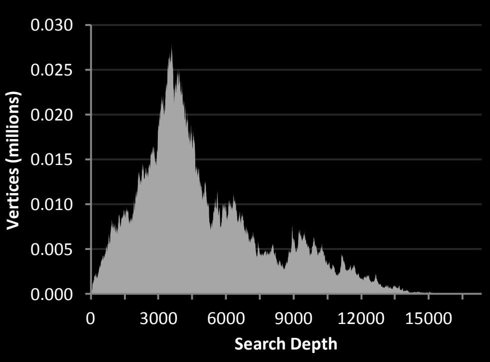
Wikipedia



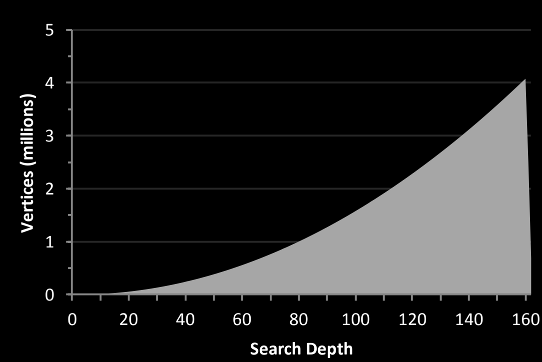
3D cubic lattice



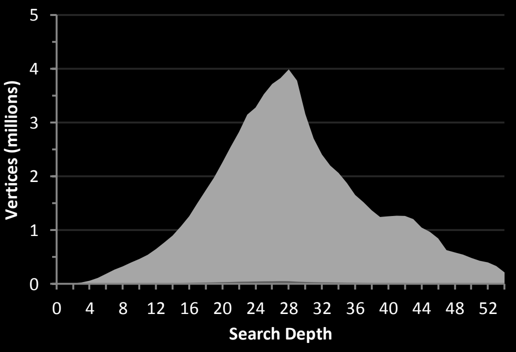
R-MAT small-world



Europe road atlas



PDE-constrained optimization

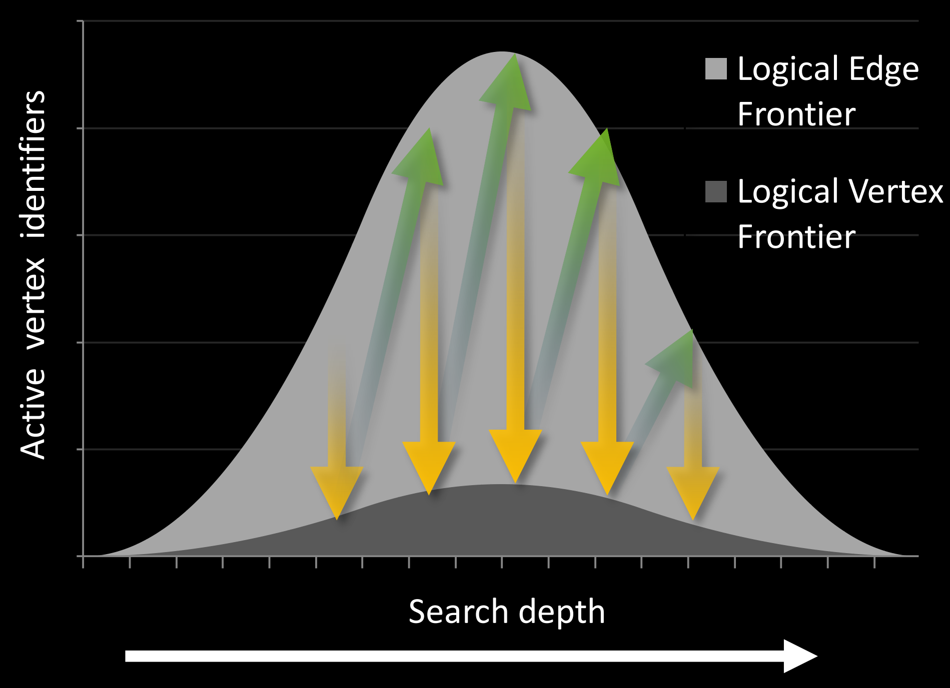


Auto transmission mesh

GPU TECHNOLOGY CONFERENCE

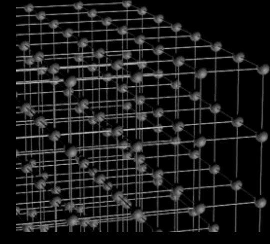
Linear-work approach

1. Expand neighbors in parallel
   - Enqueue adjacency lists
   - Vertex frontier → Edge-frontier
2. Filter out previously-visited vertices in parallel
   - Edge-frontier → Vertex frontier
3. Repeat

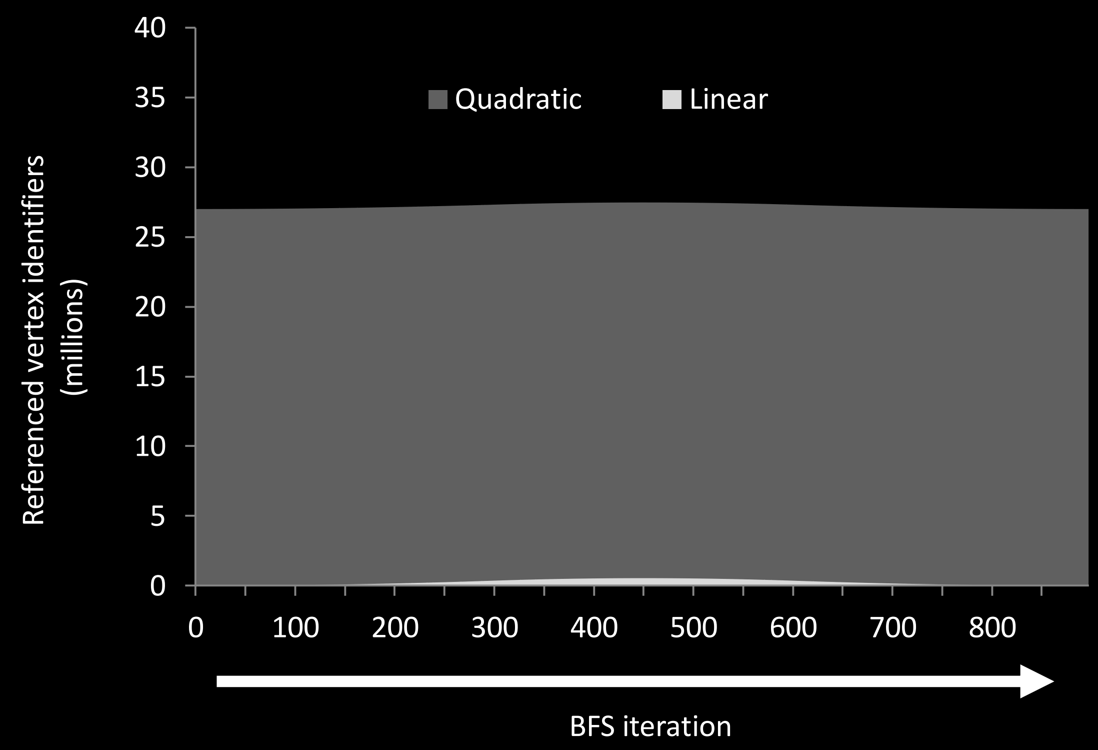


- ***Good***: Linear $O(n+m)$ work
  - Merrill et al. (PPoPP'12), Luo et al., etc.
- ***Challenging***: Need shared queues for tracking frontiers

Quadratic vs. linear



3D Poisson Lattice
(300^3 = 27M vertices)



130x more work!!

GPU TECHNOLOGY CONFERENCE

Linear-work challenges for parallelization

- Generic challenges
  1. Load imbalance between cores
     - Coarse workstealing queues
  2. Bandwidth inefficiency
     - Use “status bitmask” when filtering already-visited nodes

- GPU-specific challenges
  1. Cooperative allocation (global queues)
  2. Load imbalance within SIMD lanes
  3. Poor locality within SIMD lanes
  4. Simultaneous discovery (i.e., the benign race condition)

GPU TECHNOLOGY CONFERENCE

Linear-work challenges for parallelization

- Generic challenges
  1. Load imbalance between cores
     - Coarse workstealing queues
  2. Bandwidth inefficiency
     - Use "status bitmask" when filtering already-visited nodes

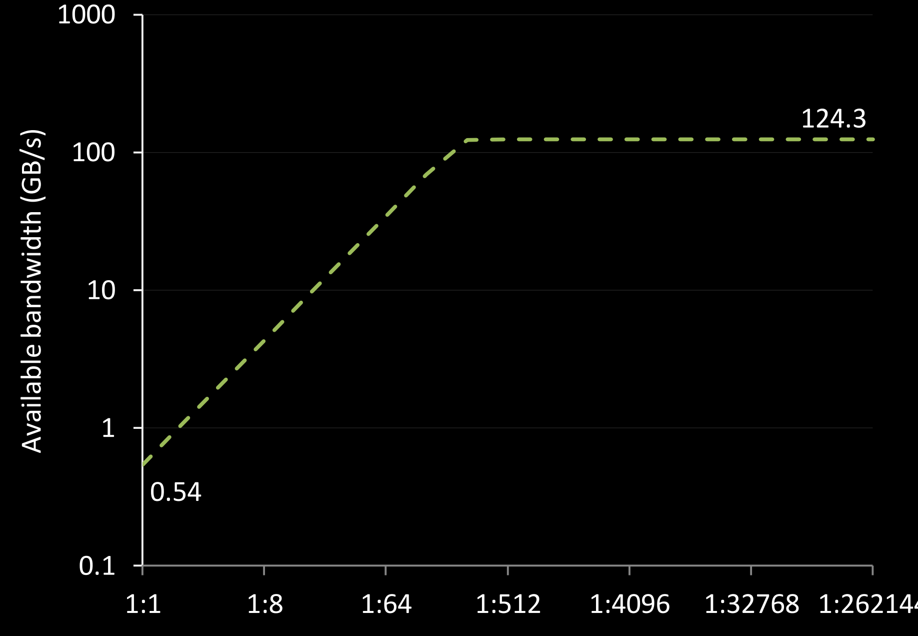
- GPU-specific challenges
  1. Cooperative allocation (global queues)
  2. Load imbalance within SIMD lanes
  3. Poor locality within SIMD lanes
  4. Simultaneous discovery (i.e., the benign race condition)

GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

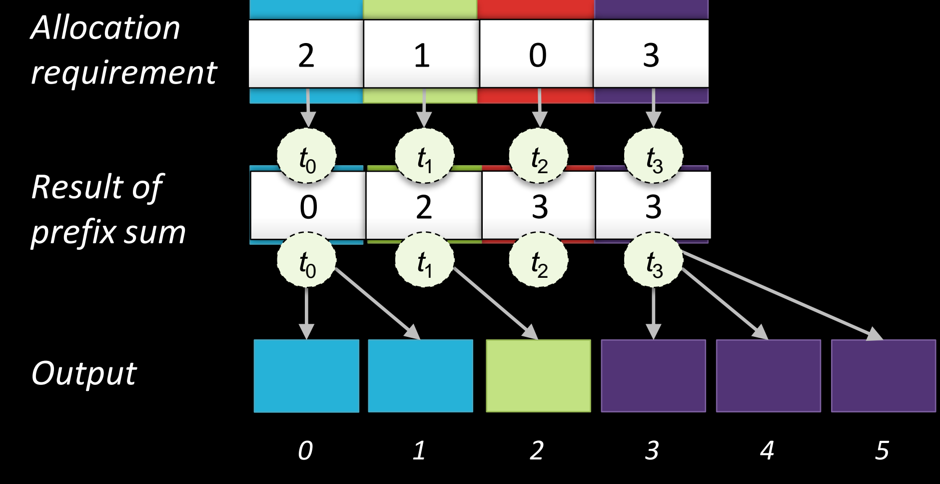
1) Cooperative data movement (queuing)

- Problem:
  - Need shared work "queues"
  - GPU rate-limits (C2050):
    - 16 billion vertex-identifiers / sec (124 GB/s)
    - Only 67M global-atomics / sec (238x slowdown)
    - Only 600M smem-atomics / sec (27x slowdown)



- Solution:
  - Compute enqueue offsets using prefix-sum

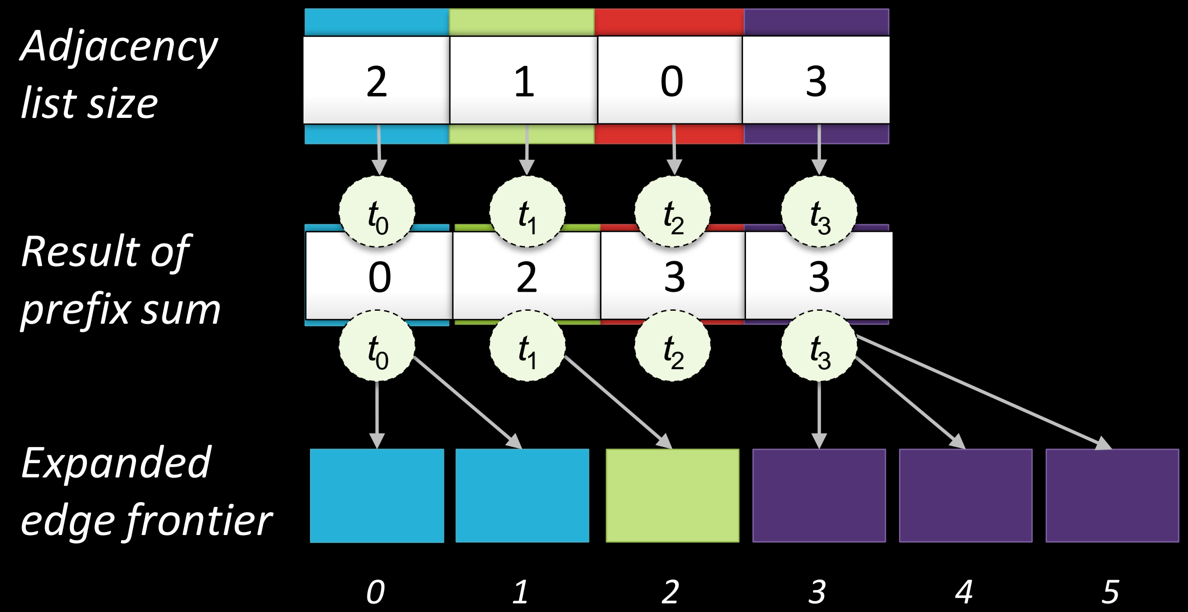
Prefix sum for allocation

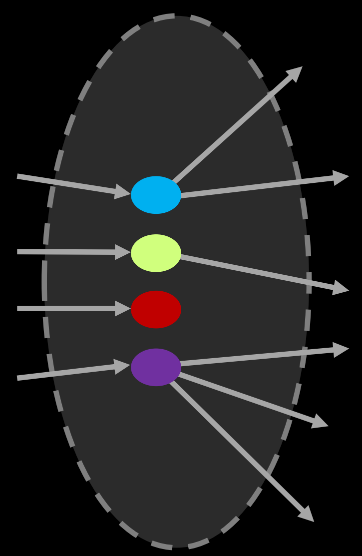


- Each output is computed to be the sum of the previous inputs
  - $O(n)$ work
  - Use results as a scatter offsets
- Fits the GPU machine model well
  - Proceed at copy-bandwidth
  - Only ~8 thread-instructions per input

GPU TECHNOLOGY CONFERENCE

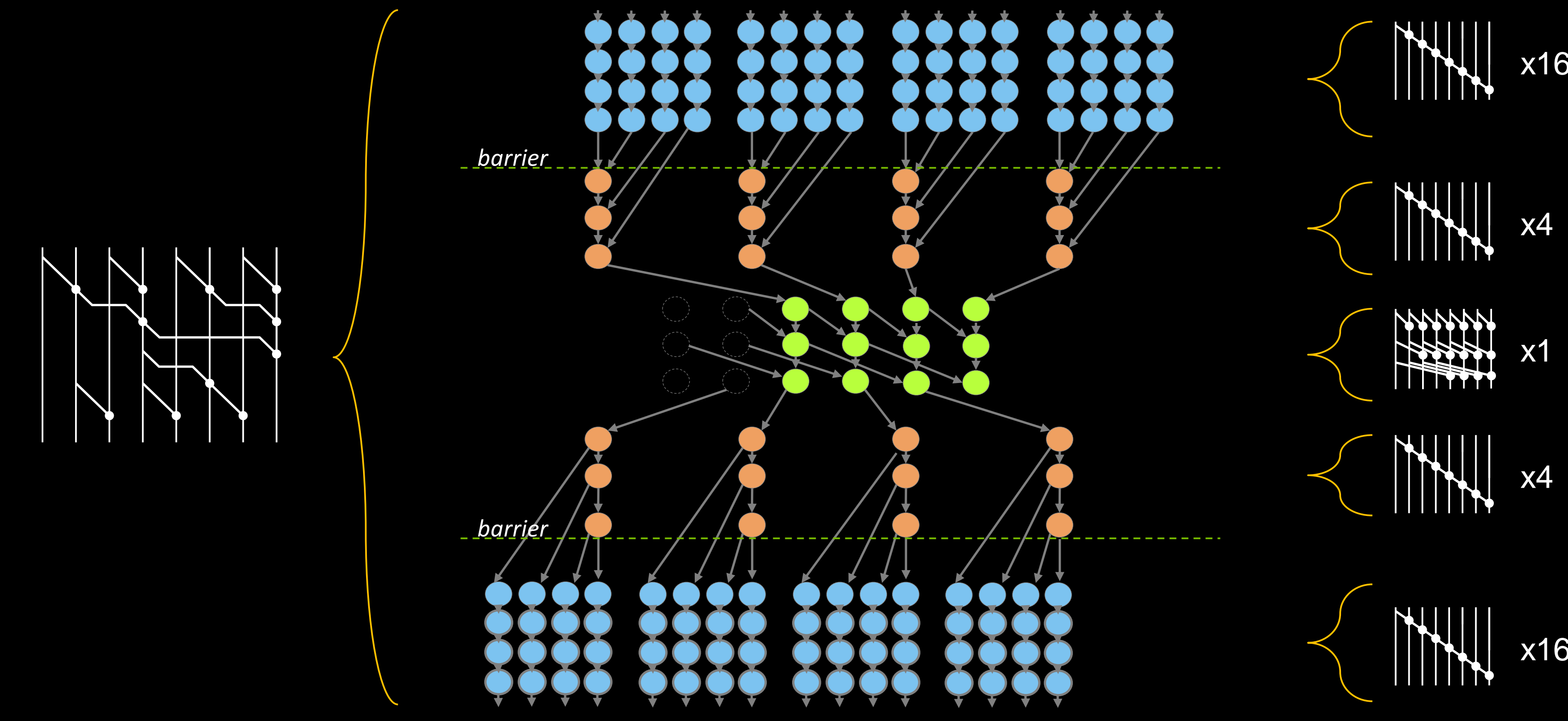
Prefix sum for edge expansion





Data-flow of efficient CTA scan

(granularity coarsening)

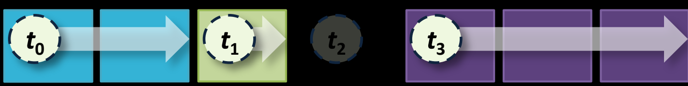


(For good GPU scan details, see Baxter's tutorials @ http://www.moderngpu.com/)

2) Poor load balancing within SIMD lanes

- Problem:
  - Large variance in adjacency list sizes
    - Exacerbated by wide SIMD widths
- Solution:
  - Cooperative neighbor expansion
    - Enlist nearby threads to help process each adjacency list in parallel

a) **Bad:** *Serial expansion & processing*



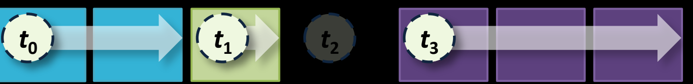
b) **Slightly better:** *Coarse warp-centric parallel expansion*

c) **Best:** *Fine-grained parallel expansion (packed by prefix sum)*

2) Poor load balancing within SIMD lanes

- Problem:
  - Large variance in adjacency list sizes
    - Exacerbated by wide SIMD widths
- Solution:
  - Cooperative neighbor expansion
    - Enlist nearby threads to help process each adjacency list in parallel

a) **Bad:** *Serial expansion & processing*



b) **Slightly better:** *Coarse warp-centric parallel expansion*

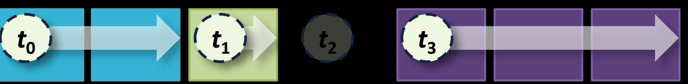
c) **Best:** *Fine-grained parallel expansion (packed by prefix sum)*

GPU TECHNOLOGY CONFERENCE

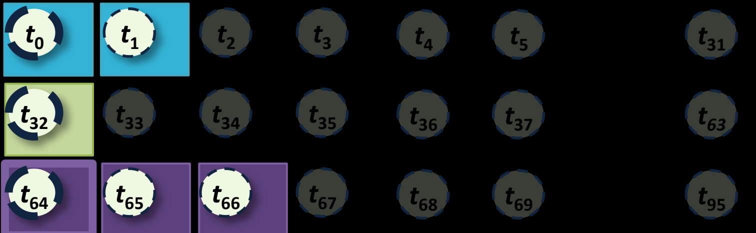
3) Poor locality within SIMD lanes

- Problem:
  - The referenced adjacency lists are arbitrarily located
    - Exacerbated by wide SIMD widths
    - Can't afford to have SIMD threads streaming through unrelated data
- Solution:
  - Cooperative neighbor expansion

a) **Bad:** *Serial expansion & processing*

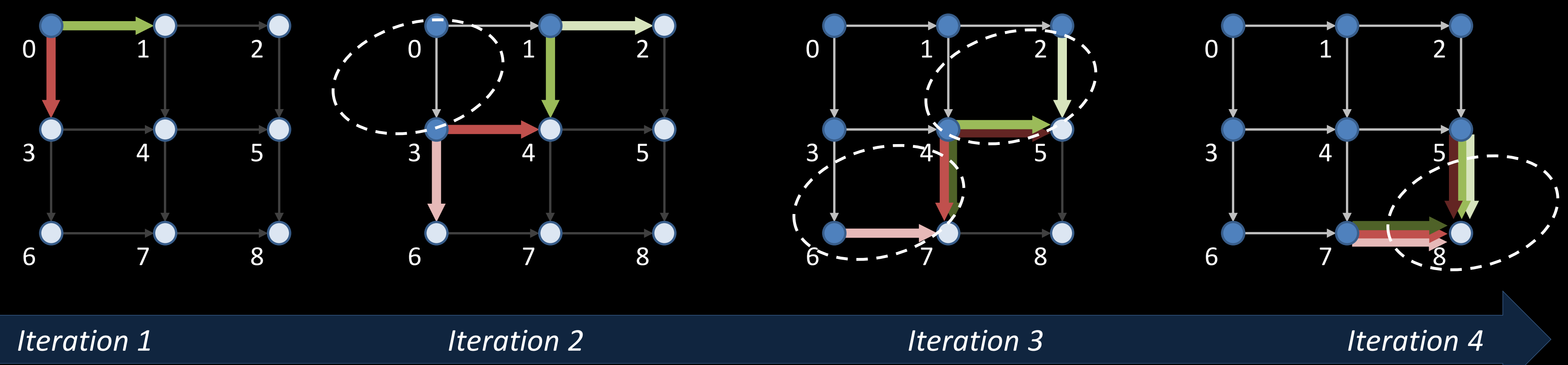


b) **Slightly better:** *Coarse warp-centric parallel expansion*



c) **Best:** *Fine-grained parallel expansion (packed by prefix sum)*

4) SIMD simultaneous discovery

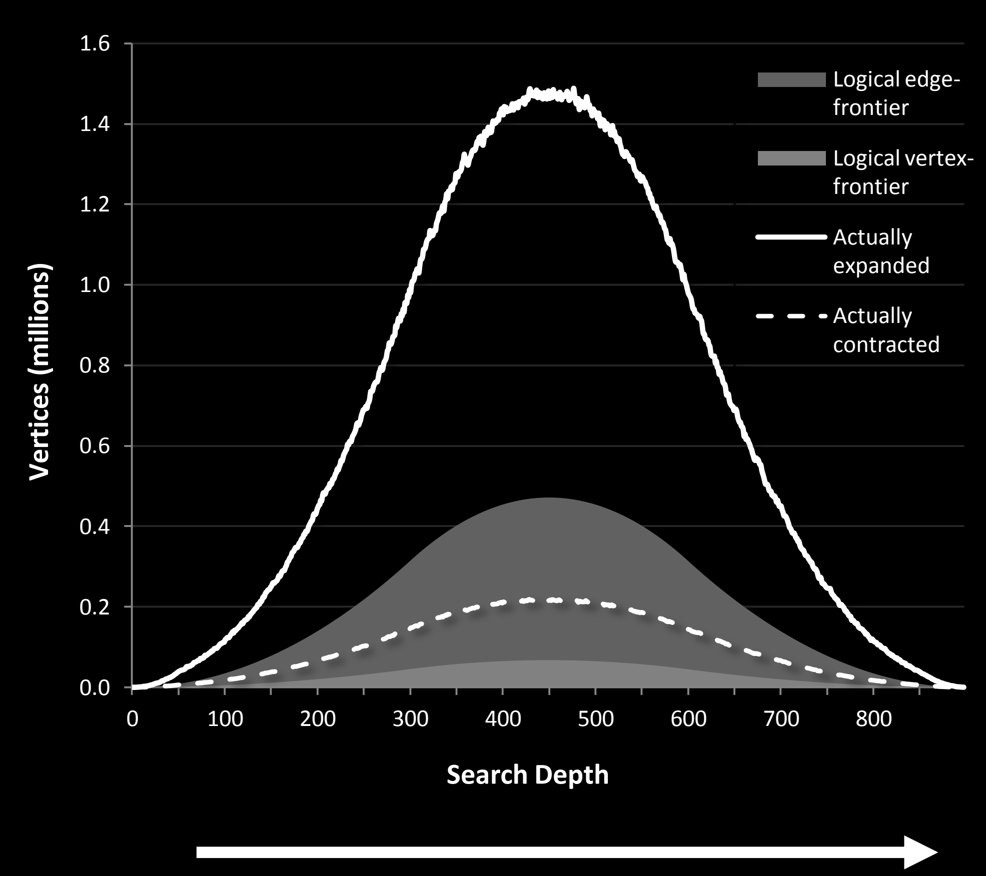


- "Duplicates" in the edge-frontier → redundant work
  - Exacerbated by wide SIMD
  - Compounded every iteration

GPU TECHNOLOGY CONFERENCE

4) SIMD simultaneous discovery

- Normally not a problem for
  - CPU implementations
  - Serial adjacency list inspection
- A ***big*** problem for cooperative SIMD expansion
  - Spatially-descriptive datasets
  - Power-law datasets
- Solution:
  - Localized duplicate removal using hashes in local scratch



GPU TECHNOLOGY CONFERENCE

Single-socket comparison

Graph	Spy Plot	Average Search Depth	Nehalem		Tesla C2050	
			Billion TE/s	Parallel speedup	Billion TE/s	Parallel speedup†††
europe.osm		19314			0.3	11 x
grid5pt.5000		7500			0.6	7.3 x
hugebubbles		6151			0.4	15 x
grid7pt.300		679	0.12 (4-core†)	2.4 x	1.1	28 x
nlpkkt160		142	0.47 (4-core†)	3.0 x	2.5	10 x
audikw1		62			3.0	4.6 x
cage15		37	0.23 (4-core†)	2.6 x	2.2	18 x
kkt_power		37	0.11 (4-core†)	3.0 x	1.1	23 x
coPapersCiteseer		26			3.0	5.9 x
wikipedia-2007		20	0.19 (4-core†)	3.2 x	1.6	25 x
kron_g500-logn20		6			3.1	13 x
random.2Mv.128Me		6	0.50 (8-core††)	7.0 x	3.0	29 x
rmat.2Mv.128Me		6	0.70 (8-core††)	6.0 x	3.3	22 x

† 2.5 GHz Core i7 4-core (Leiserson *et al.*) †† 2.7 GHz EX Xeon X5570 8-core (Agarwal *et al*) ††† vs 3.4GHz Core i7 2600K (Sandybridge)

Single-socket comparison

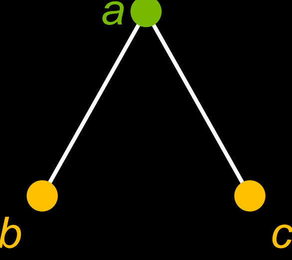
Graph	Spy Plot	Average Search Depth	Nehalem		Tesla C2050	
			Billion TE/s	Parallel speedup	Billion TE/s	Parallel speedup†††
europe.osm		19314			0.3	11 x
grid5pt.5000		7500			0.6	7.3 x
hugebubbles		6151			0.4	15 x
grid7pt.300		679	0.12 (4-core†)	2.4 x	1.1	28 x
nlpkkt160		142	0.47 (4-core†)	3.0 x	2.5	10 x
audikw1		62			3.0	4.6 x
cage15		37	0.23 (4-core†)	2.6 x	2.2	18 x
kkt_power		37	0.11 (4-core†)	3.0 x	1.1	23 x
coPapersCiteseer		26			3.0	5.9 x
wikipedia-2007		20	0.19 (4-core†)	3.2 x	1.6	25 x
kron_g500-logn20		6			3.1	13 x
random.2Mv.128Me		6	0.50 (8-core††)	7.0 x	3.0	29 x
rmat.2Mv.128Me		6	0.70 (8-core††)	6.0 x	3.3	22 x

† 2.5 GHz Core i7 4-core (Leiserson *et al.*) †† 2.7 GHz EX Xeon X5570 8-core (Agarwal *et al*) ††† vs 3.4GHz Core i7 2600K (Sandybridge)

Maximal independent set

Maximal independent set

- A vertex subset $U \subseteq V$ of an undirected graph $G = (V, E)$
  - U is *independent* if no two vertices in U are adjacent
  - U is *maximal* if no node can be added without violating independence

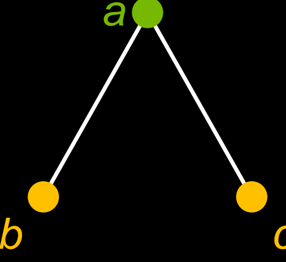


Both $\{a\}$ and $\{b, c\}$ are maximal independent sets

GPU TECHNOLOGY CONFERENCE

Maximal independent set

- A vertex subset $U \subseteq V$ of an undirected graph $G = (V, E)$
  - U is *independent* if no two vertices in U are adjacent
  - U is *maximal* if no node can be added without violating independence



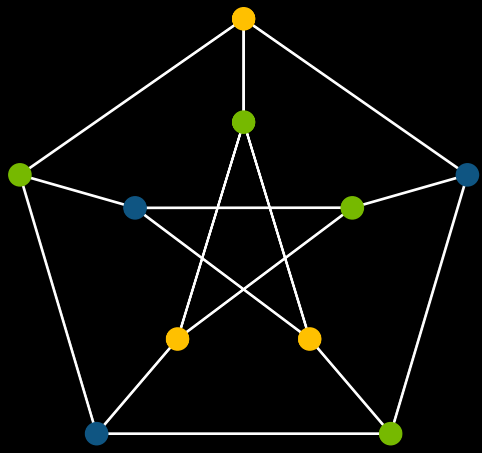
Both $\{a\}$ and $\{b, c\}$ are maximal independent sets

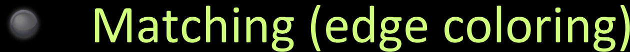
(Finding the *maximum* independent set is generally NP-Complete.)

GPU TECHNOLOGY CONFERENCE

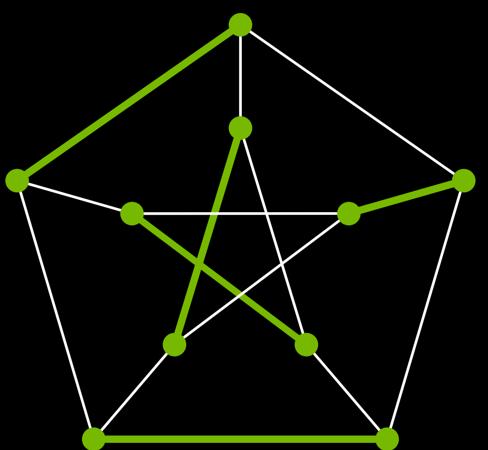
MIS applications

- Graph coloring (vertex coloring)
  - Co-scheduling sets of independent resources
  - Useful for GPUs: avoid atomic updates by touching items in different kernels



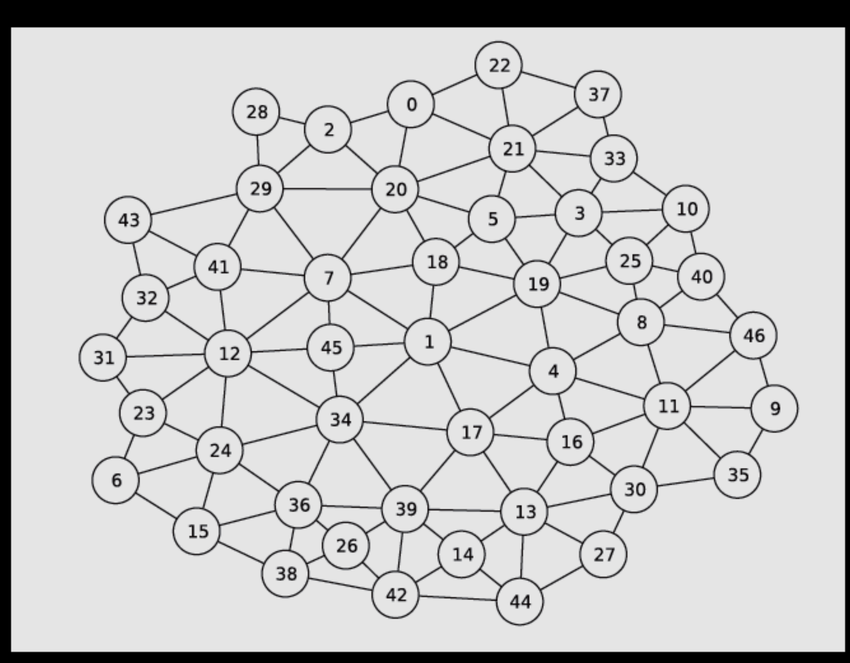


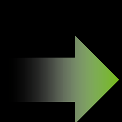
- Matching (edge coloring)
  - Pair-wise assignments
    - Scheduling of jobs onto hosts
    - Matching people with desks, dance partners, kidneys, etc.
  - Run MIS on *G'* where:
    - There is a node in *G'* for every edge in *G*
    - Nodes are connected in *G'* if edges in *G* are adjacent

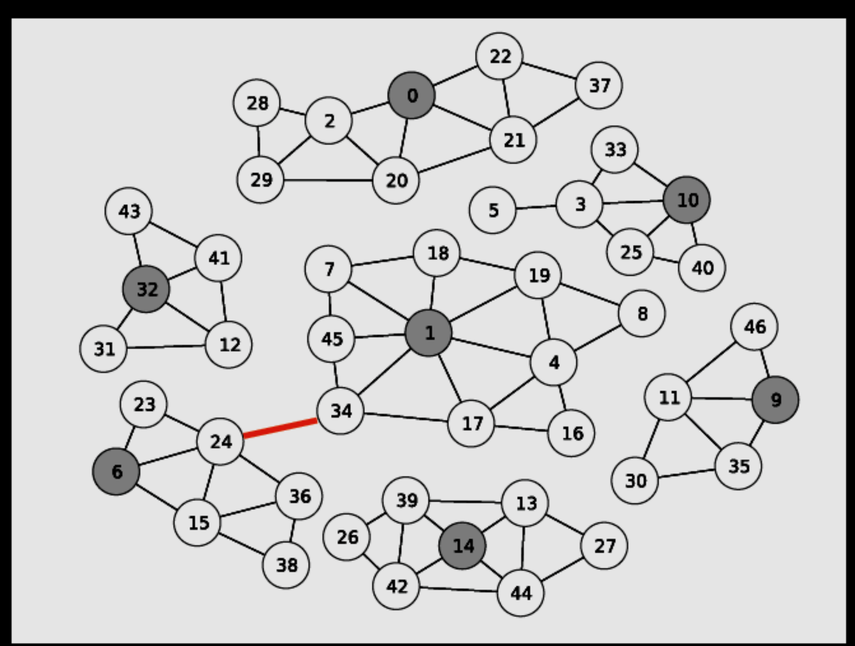


GPU TECHNOLOGY CONFERENCE

Example: MIS(2) for mesh refinement in algebraic multi-grid



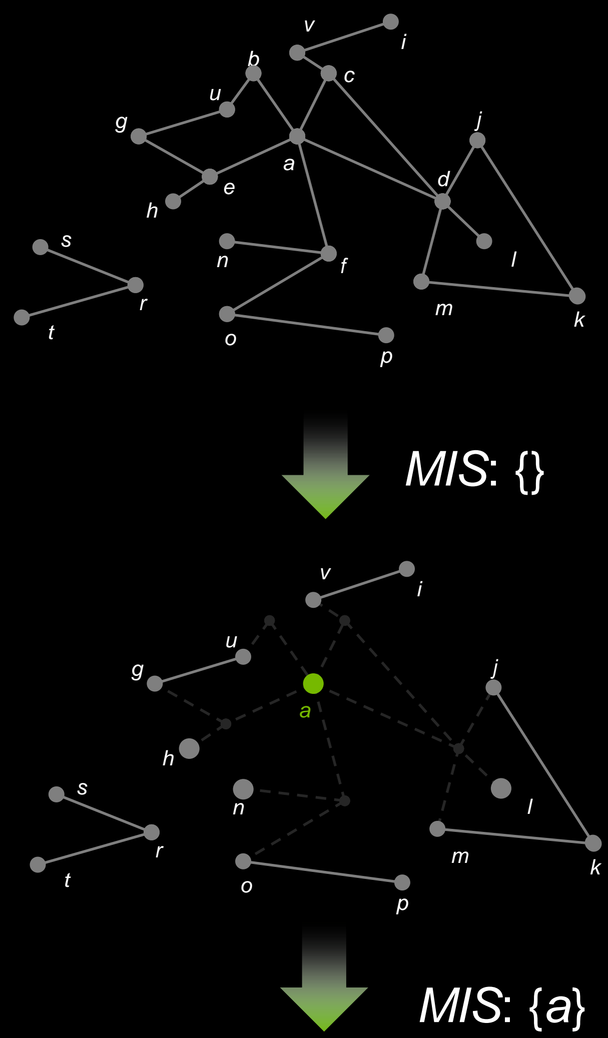




Bell et al., Cohen et al., etc.

The sequential algorithm

- Repeatedly add a vertex to the MIS, removing it and its neighbors from the graph:
  1. Select first identifier *v* from *V*
  2. Add *v* to *MIS*
  3. Remove *v* and *neighbors*(*v*) from *V*
  4. Repeat until *V* = {}



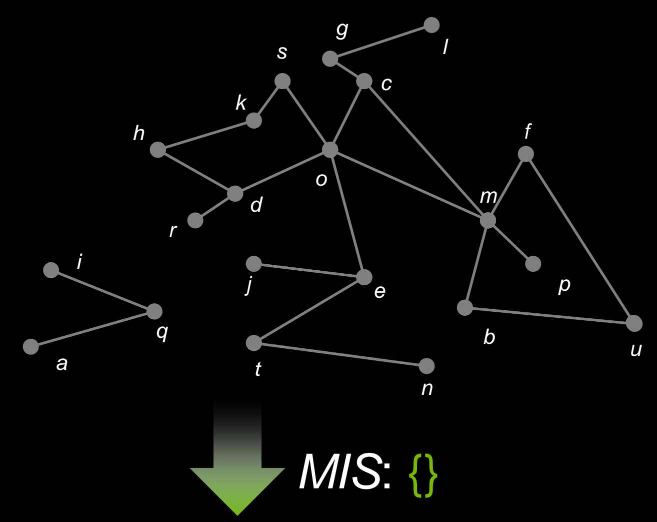
GPU TECHNOLOGY CONFERENCE

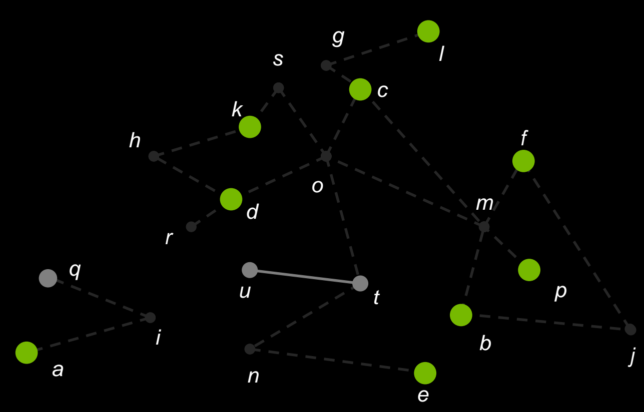
The parallel algorithm (Luby85)

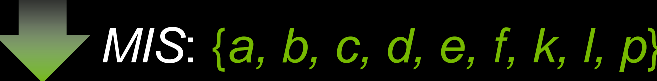
- Repeatedly add sets to the MIS, removing them and their neighbors from the graph:

    1. Randomly permute identifier ordering of V
    2. Identify all m_i such that identifier $m_i <$ all $neighbors(m_i)$
    3. Add all m_i to MIS
    4. Remove m_i and $neighbors(m_i)$ from V
    5. Repeat until $V = \{\}$

- Generally converges after $O(\log n)$ steps







…

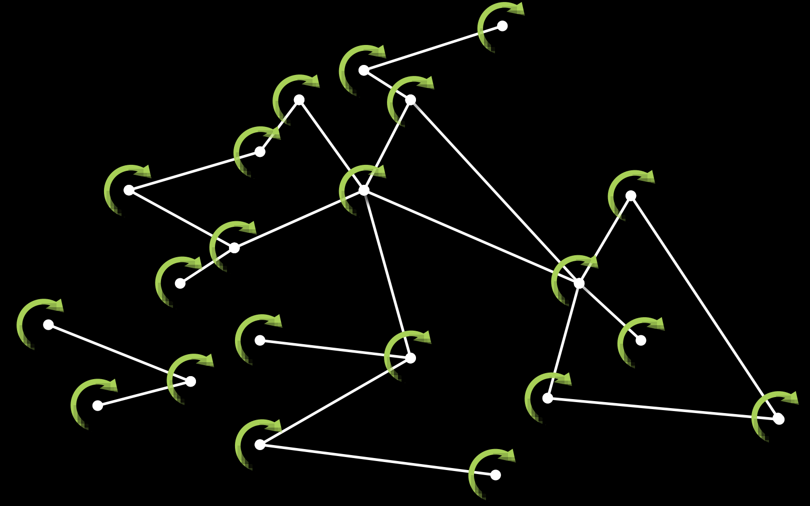
Data-parallel GPU approach

- **Input:** Vertex set V, row-offsets array R, column-indices array C, source vertex s
- **Output:** Array $MIS[0..n\text{-}1]$ with $MIS[v]$ holding membership flag (1|-1) in the maximal independent set constructed.

kernel

kernel

```
1.   parallel for (i in V) :
2.     MIS[i] := 0
3.   do :
4.     done := true
5.     parallel for (i in V) :
6.       if (MIS[i] == 0)
7.         dependent_i := false
8.         done := false
9.         local_min_i := i
10.        for (offset in R[i] .. R[i+1]-1) :
11.          j := C[offset]
12.          if (MIS[j] == 1)
13.            dependent_i = true
14.          else if ((MIS[j] == 0) && (permute(j) < permute(i))
15.            local_min_i := j
16.        if (dependent_i)
17.          MIS[i] := -1
18.        else if (local_min_i == i)
19.          MIS[i] := 1
20.  while (!done)
```



- ***Good***: trivial data-parallel GPU stencils
  - E.g., CUSP library, Hoberock, Cohen, Wang, etc.
- ***Bad***: average case $O(m \log n)$ work because edges are never removed

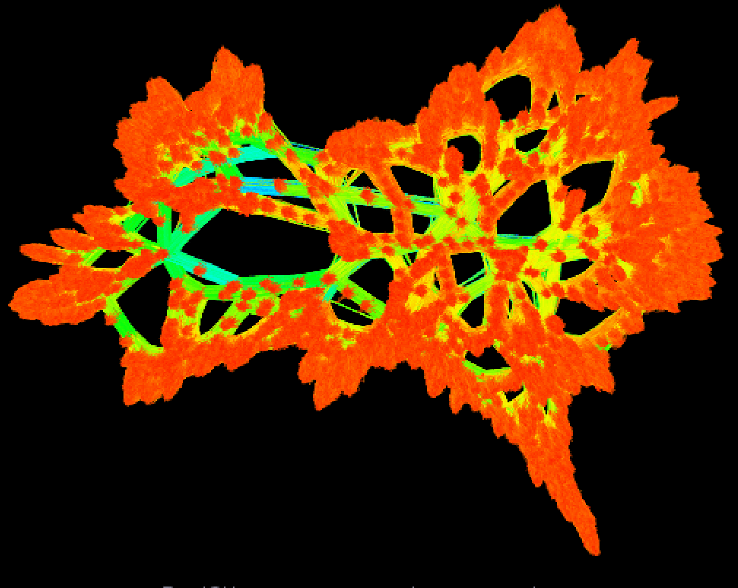
GPU TECHNOLOGY CONFERENCE

Future work: actually remove vertices from *G*

- Don’t visit all *n* vertices at every time step!
- Should significantly improve utilization in subsequent rounds
  - Higher density of active vertices
- CSR isn’t a particularly friendly format for vertex removal
  - ***Easy:*** Use prefix sum to compact the *row_offsets* array
    - Double the size to pair segment-begin offsets with explicit segment-end offsets
  - ***Hard:*** No great way to find and remove vertex identifiers from adjacency lists

Summary

- Purely data-parallel approaches can be uncompetitive
  - E.g., one thread per vertex/edge @ each time step
- Dynamic workload management:
  - Prefix sum (vs. atomic-add)
  - Utilization requires fine-grained expansion and contraction
- GPUs can be very amenable to dynamic, cooperative problems



Zaoui@kkt_power. 2063494 nodes, 6482320 edges.

Questions?

dumerrill@nvidia.com

GPU TECHNOLOGY CONFERENCE

GTC 2013:The Premier Event in Accelerated Computing

Registration is Open!

- Four days – March 18-21, San Jose, CA
- Three keynotes
- 300+ sessions
- One day of pre-conference developer tutorials
- 100+ research posters
- Lots of networking events and opportunities

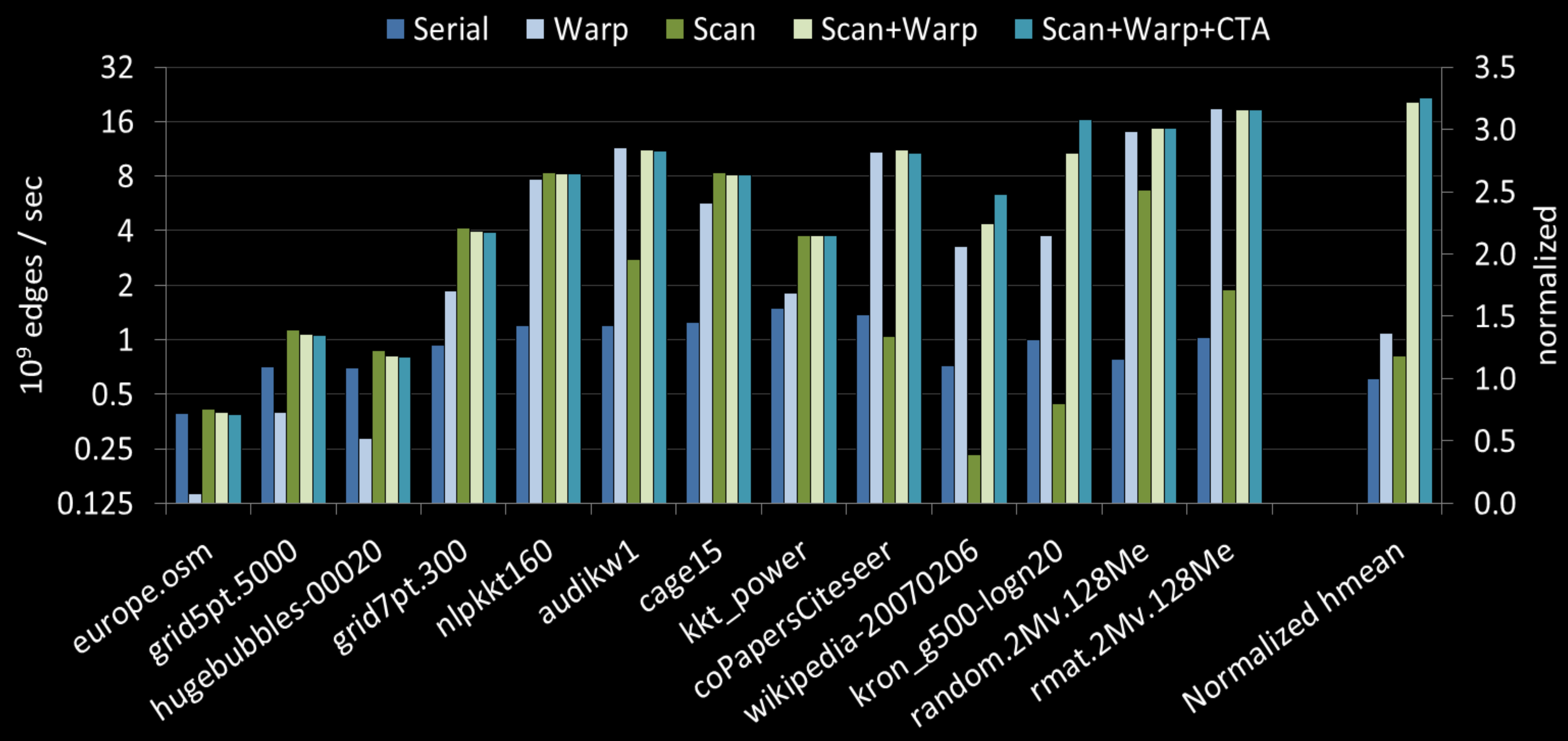
Visit www.gputechconf.com for more info.

GPU TECHNOLOGY CONFERENCE

Experimental corpus

Graph	Spy Plot	Description	Average Search Depth	Vertices (millions)	Edges (millions)
europe.osm		Road network	19314	50.9	108.1
grid5pt.5000		2D Poisson stencil	7500	25.0	125.0
hugebubbles-00020		2D mesh	6151	21.2	63.6
grid7pt.300		3D Poisson stencil	679	27.0	188.5
nlpkkt160		Constrained optimization problem	142	8.3	221.2
audikw1		Finite element matrix	62	0.9	76.7
cage15		Transition prob. matrix	37	5.2	94.0
kkt_power		Optimization (KKT)	37	2.1	13.0
coPapersCiteseer		Citation network	26	0.4	32.1
wikipedia-20070206		Wikipedia page links	20	3.6	45.0
kron_g500-logn20		Graph500 random graph	6	1.0	100.7
random.2Mv.128Me		Uniform random graph	6	2.0	128.0
rmat.2Mv.128Me		RMAT random graph	6	2.0	128.0

Comparison of expansion techniques

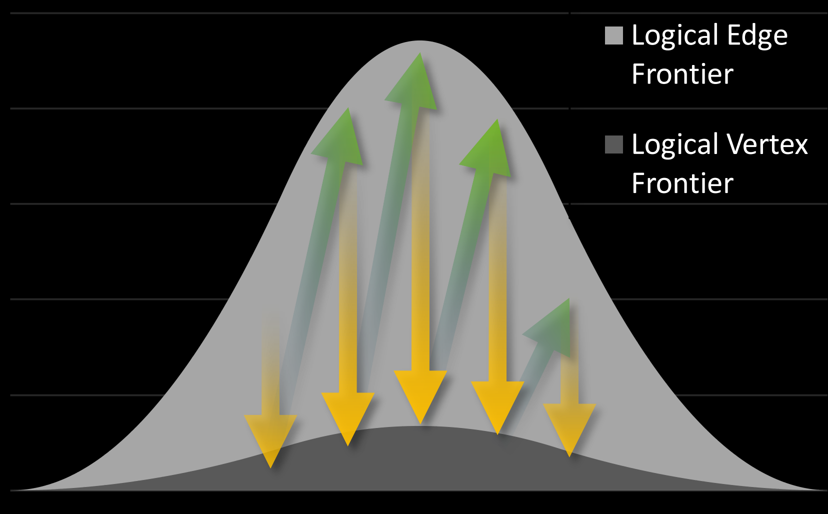


GPU TECHNOLOGY CONFERENCE

GPU TECHNOLOGY CONFERENCE

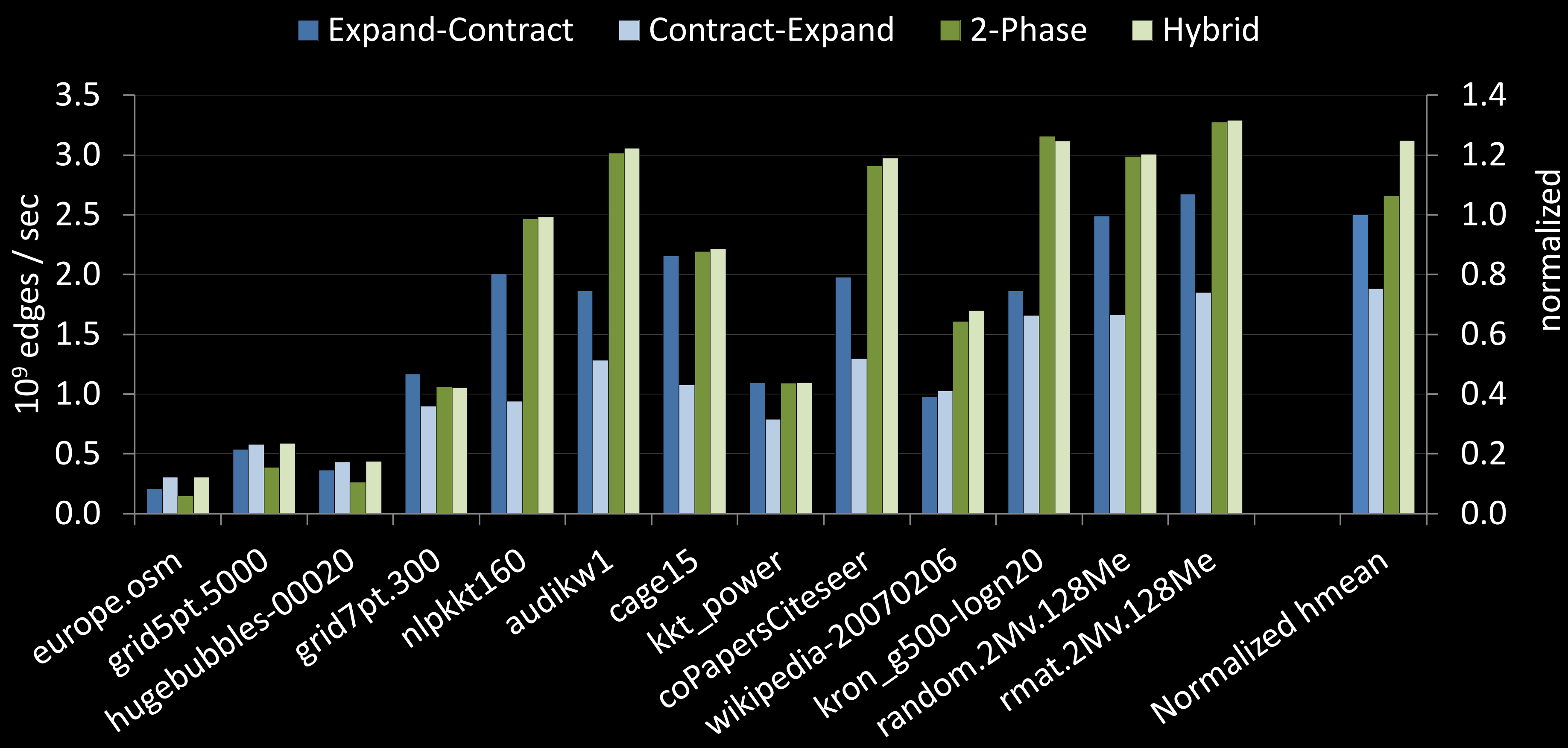
Coupling of expansion & contraction phases

- Alternatives:
  - Expand-contract
    - Wholly realize vertex-frontier in DRAM
    - Suitable for all types of BFS iterations
  - Contract-expand
    - Wholly realize edge-frontier in DRAM
    - Even better for small, fleeting BFS iterations
  - Two-phase
    - Wholly realize both frontiers in DRAM
    - Even better for large, saturating BFS iterations (surprising!!)



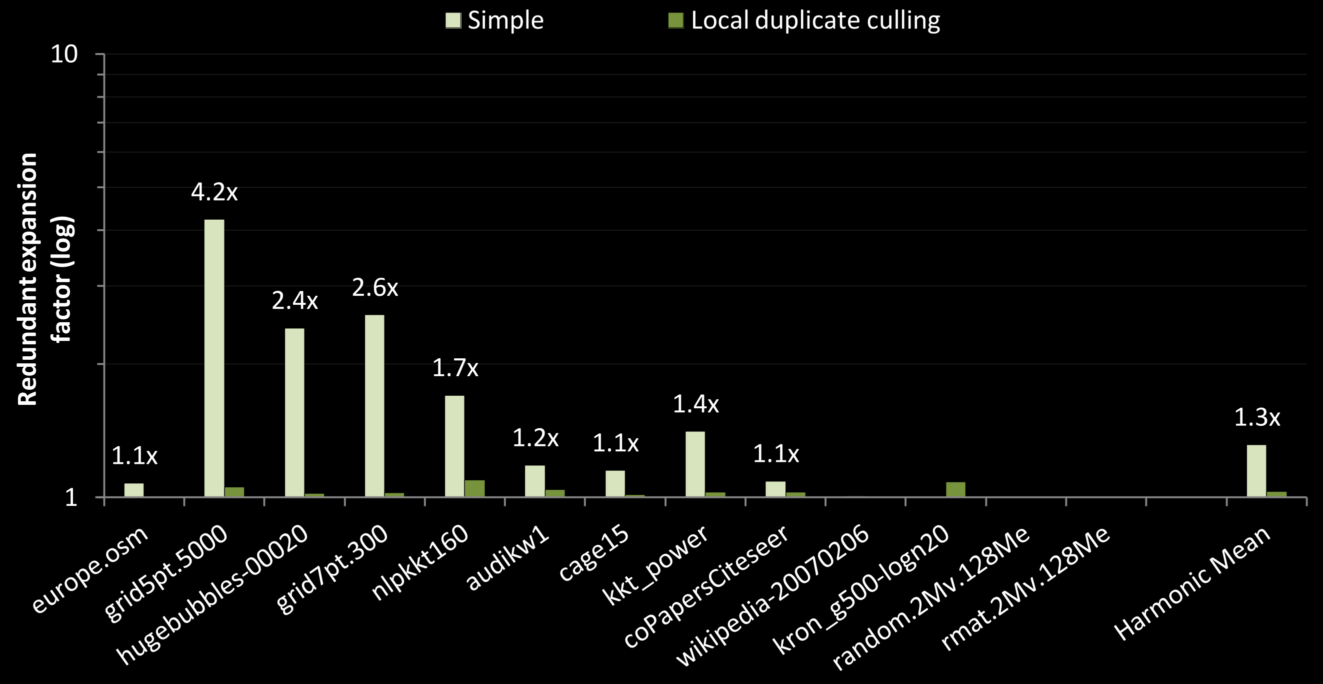
GPU TECHNOLOGY CONFERENCE

Comparison of coupling approaches



Local duplicate culling

- Contraction uses local collision hashes in smem scratch space:
  - Hash per warp (instantaneous coverage)
  - Hash per CTA (recent history coverage)
- Redundant work < 10% in all datasets
- No atomics needed



GPU TECHNOLOGY CONFERENCE

Multi-GPU traversal

Expand neighbors → sort by GPU (with filter) → read from peer GPUs (with filter) → repeat