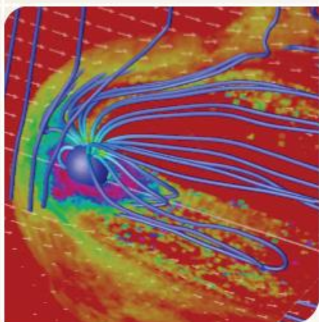
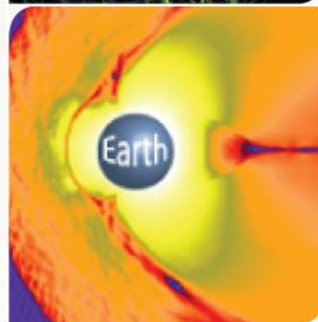
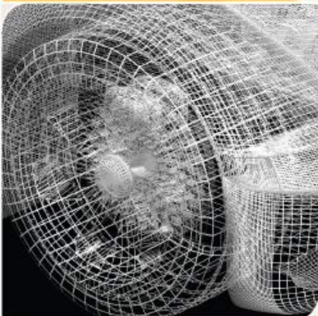



Coordinating More Than 3 Million CUDA Threads for Social Network Analysis

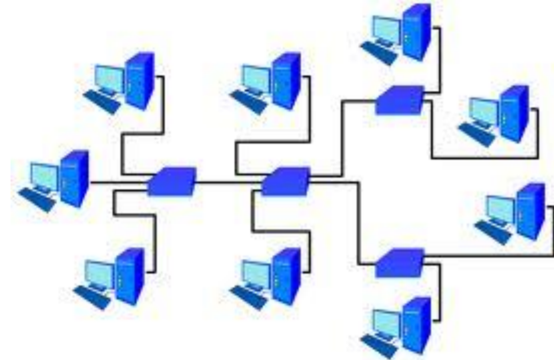
Adam McLaughlin



Georgia Tech
College of Computing
Computational Science and Engineering

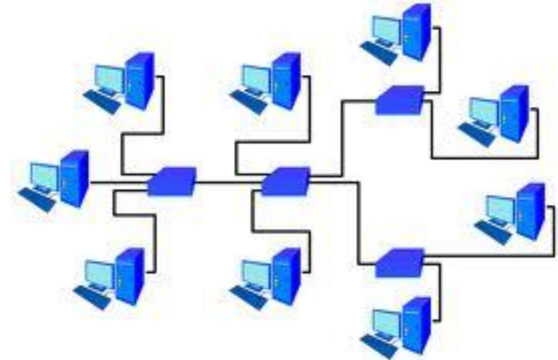
Applications of interest…

- Computational biology
- Social network analysis
- Urban planning
- Epidemiology
- Hardware verification



Applications of interest…

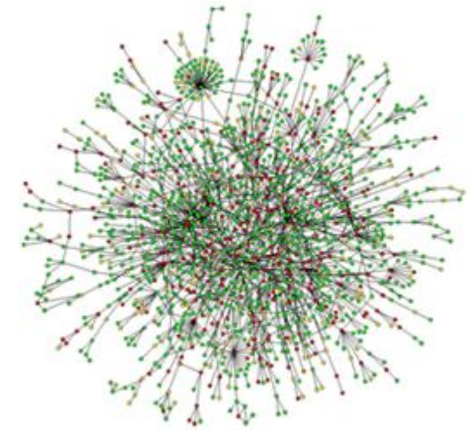
- Computational biology
- Social network analysis
- Urban planning
- Epidemiology
- Hardware verification



- Common denominator:
  Graph Analysis

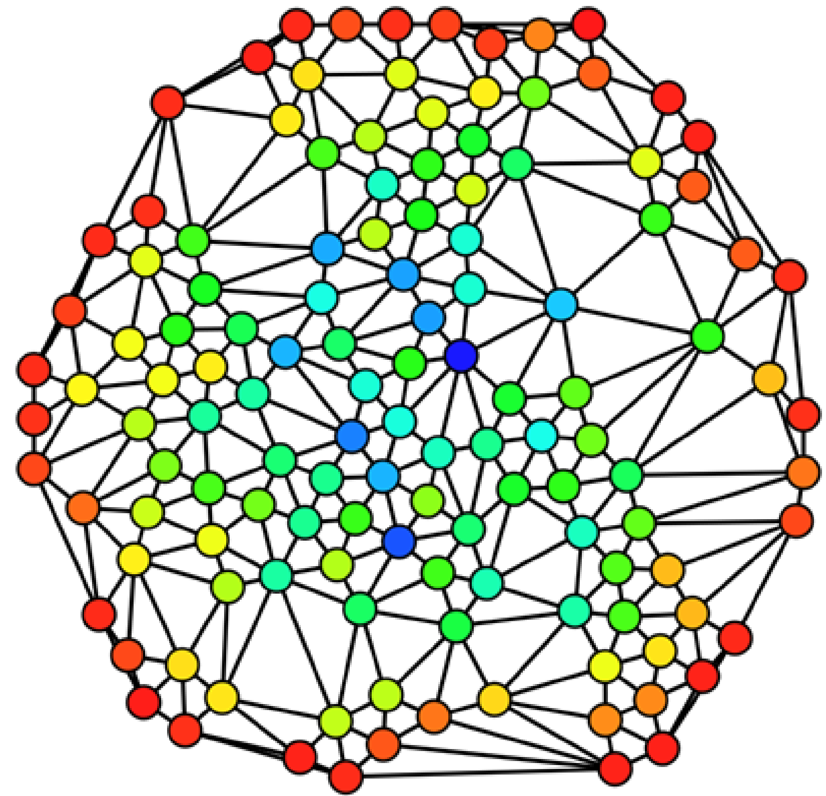
Challenges in Network Analysis

- Size
  - Networks cannot be manually inspected
- Varying structural properties
  - Small-world, scale-free, meshes, road networks
    - Not a one-size fits all problem
- Unpredictable
  - Data-dependent memory access patterns



Betweenness Centrality

- Determine the importance of a vertex in a network
  - Requires the solution of the APSP problem
- Applications are manifold
- Computationally demanding
  - $O(mn)$ time complexity

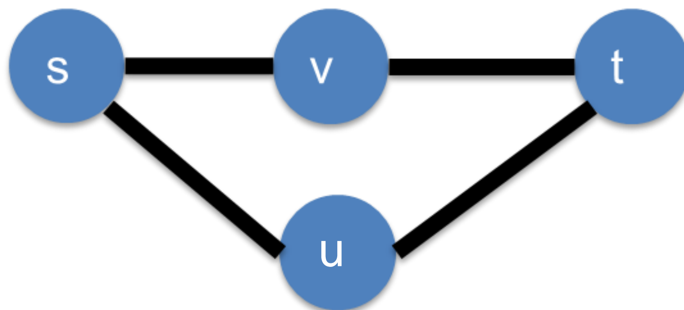


Defining Betweenness Centrality

- Formally, the BC score of a vertex is defined as:

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

  - σ_{st} is the number of shortest paths from s to t
  - $\sigma_{st}(v)$ is the number of those paths passing through v



$\sigma_{st} = 2$
$\sigma_{st}(v) = 1$

Brandes's Algorithm

1. **Shortest path calculation** (*downward*)
2. **Dependency accumulation** (*upward*)
   - Dependency:

$$\delta_{sv} = \sum_{w \in succ(v)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{sw})$$

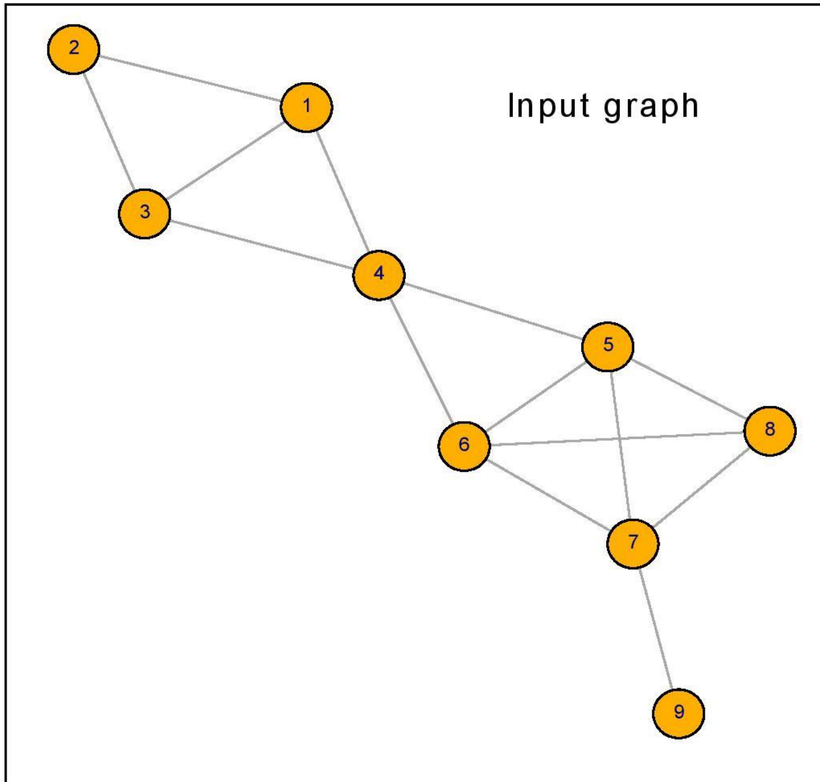
   - Redefine BC scores as:

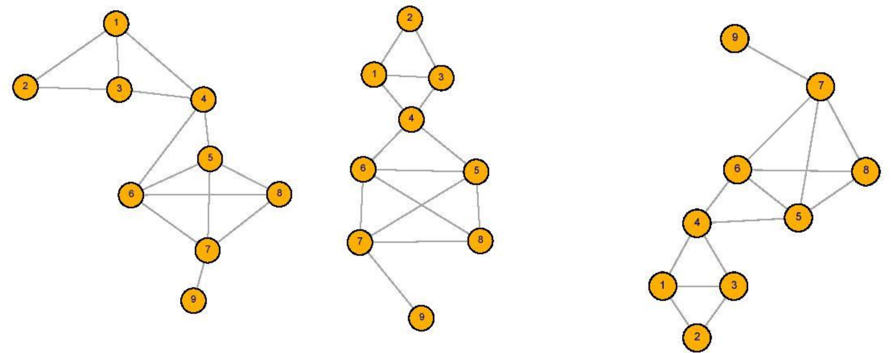
$$BC(v) = \sum_{s \neq \mathrm{v}} \delta_{sv}$$

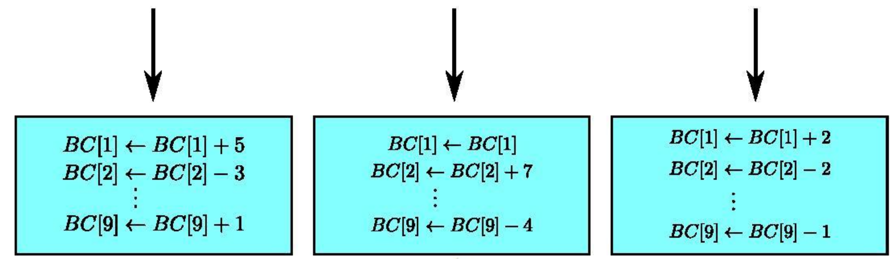
Prior GPU Implementations

- Vertex and Edge Parallelism [Jia *et al*. (2011)]
  - Same coarse-grained strategy
  - Edge-parallel approach better utilizes the GPU
- GPU-FAN [Shi and Zhang (2011)]
  - Reported 11-19% speedup over Jia *et al.*
    - Results were limited in scope
  - Devote entire GPU to fine-grained parallelism
- Both use large $\{O(m), O(n^2)\}$ predecessor arrays
  - Our approach: **eliminate this array**
- Both use $O(n^2 + m)$ graph traversals
  - Our approach: **trade-off memory bandwidth and excess work**

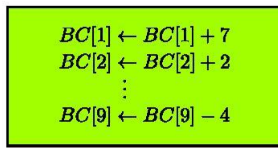
Coarse-grained Parallelization Strategy





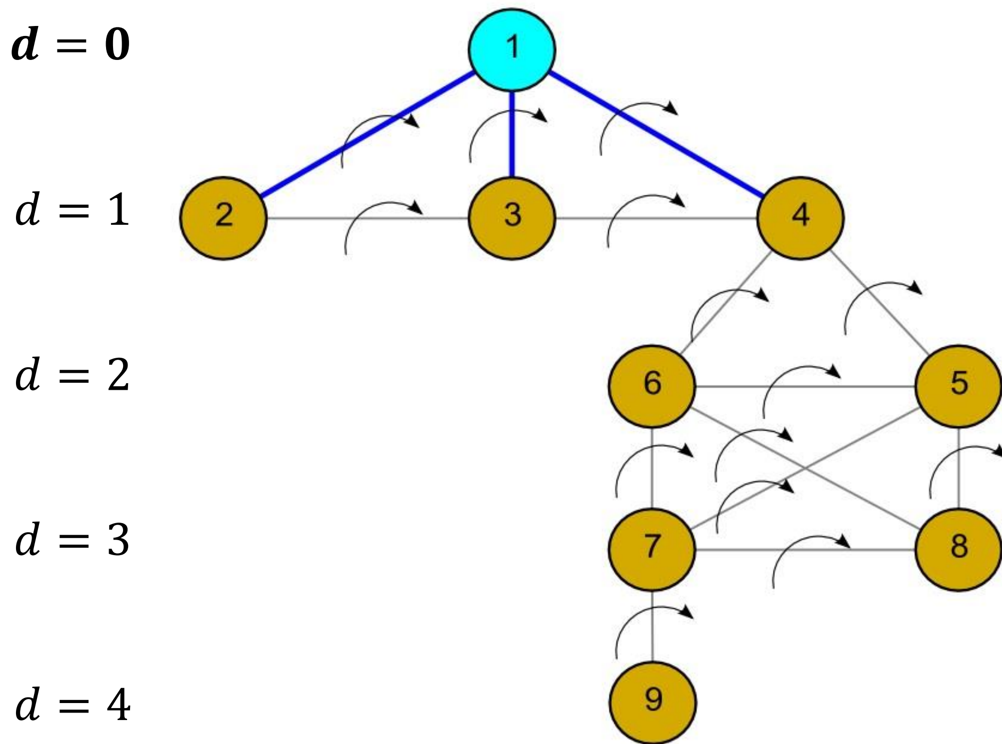


Merge together atomically



Fine-grained Parallelization Strategy

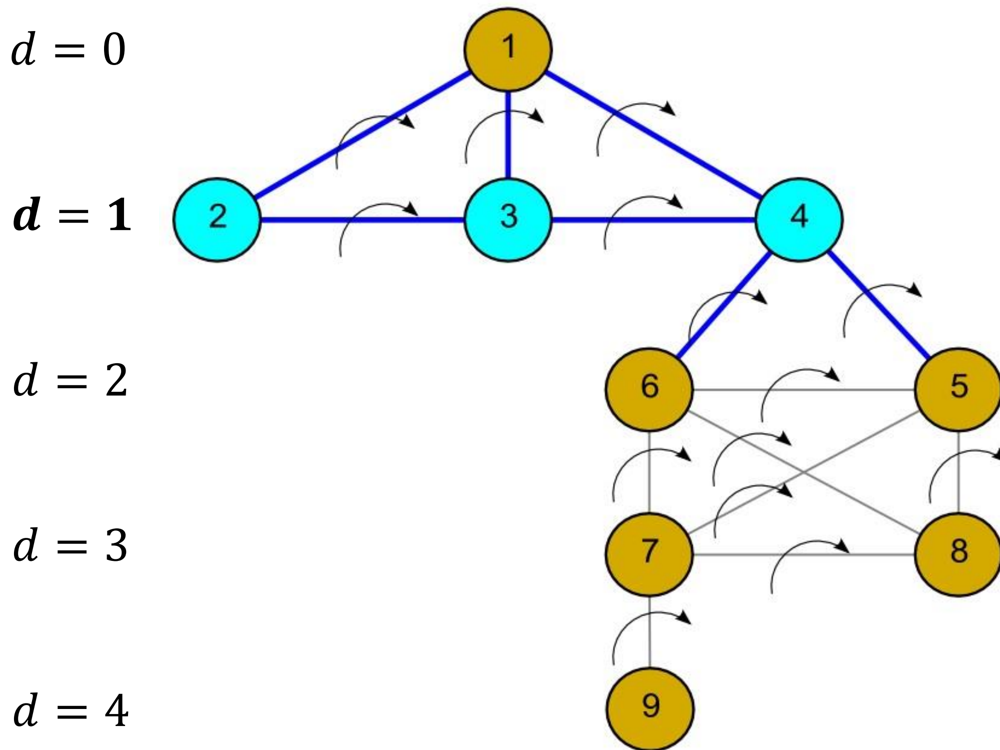
- Edge-parallel downward traversal



- Threads are assigned to each edge
  - Only a subset is active
- Balanced amount of work per thread

Fine-grained Parallelization Strategy

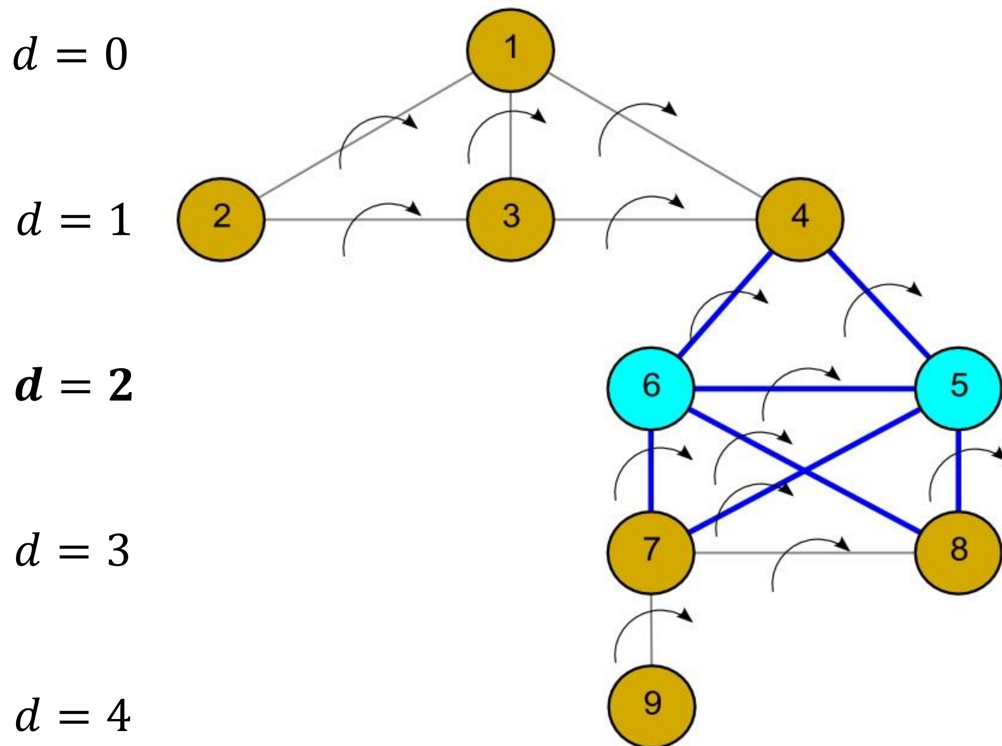
- Edge-parallel downward traversal



- Threads are assigned to each edge
  - Only a subset is active
- Balanced amount of work per thread

Fine-grained Parallelization Strategy

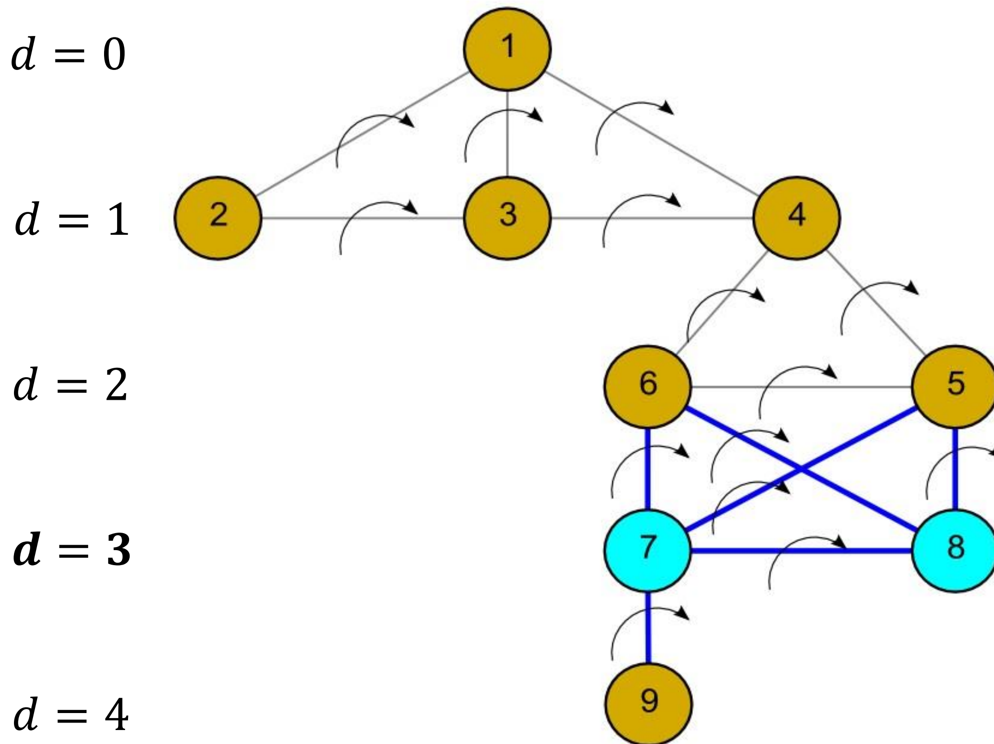
- Edge-parallel downward traversal



- Threads are assigned to each edge
  - Only a subset is active
- Balanced amount of work per thread

Fine-grained Parallelization Strategy

- Edge-parallel downward traversal

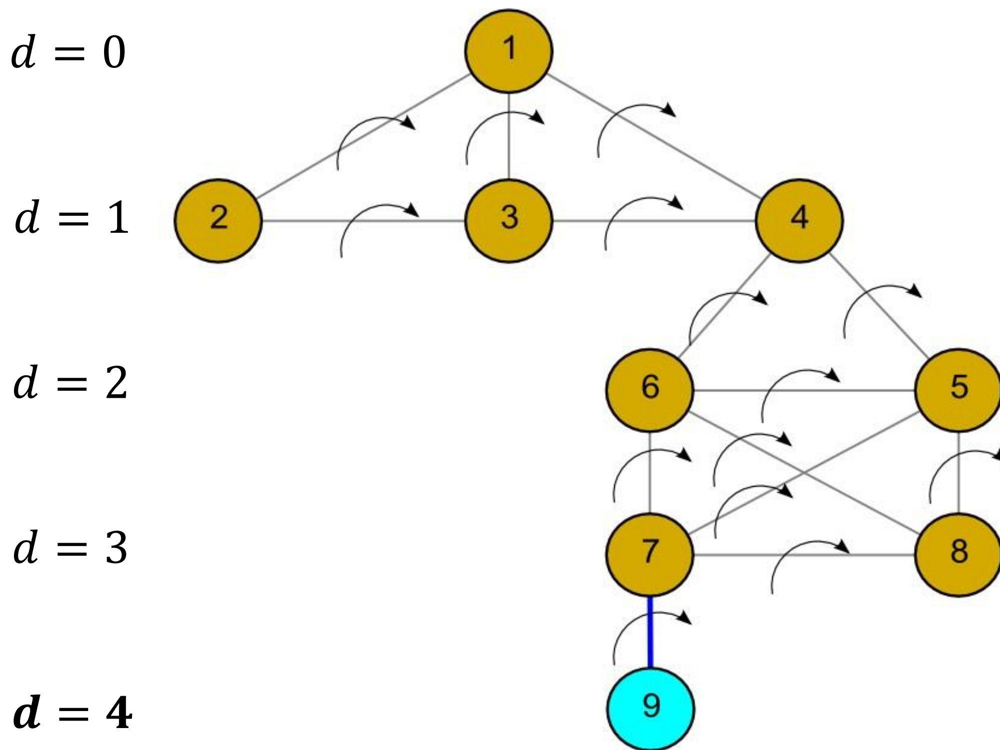


- Threads are assigned to each edge
  - Only a subset is active
- Balanced amount of work per thread

Georgia Tech College of Computing

Fine-grained Parallelization Strategy

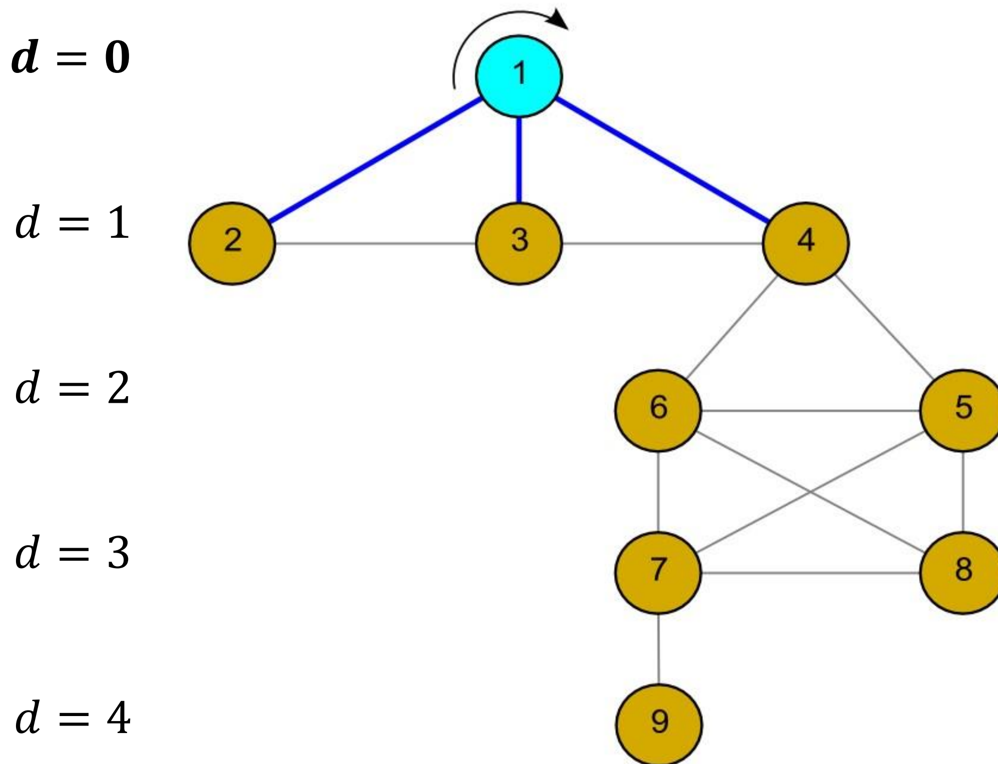
- Edge-parallel downward traversal



- Threads are assigned to each edge
  - Only a subset is active
- Balanced amount of work per thread

Fine-grained Parallelization Strategy

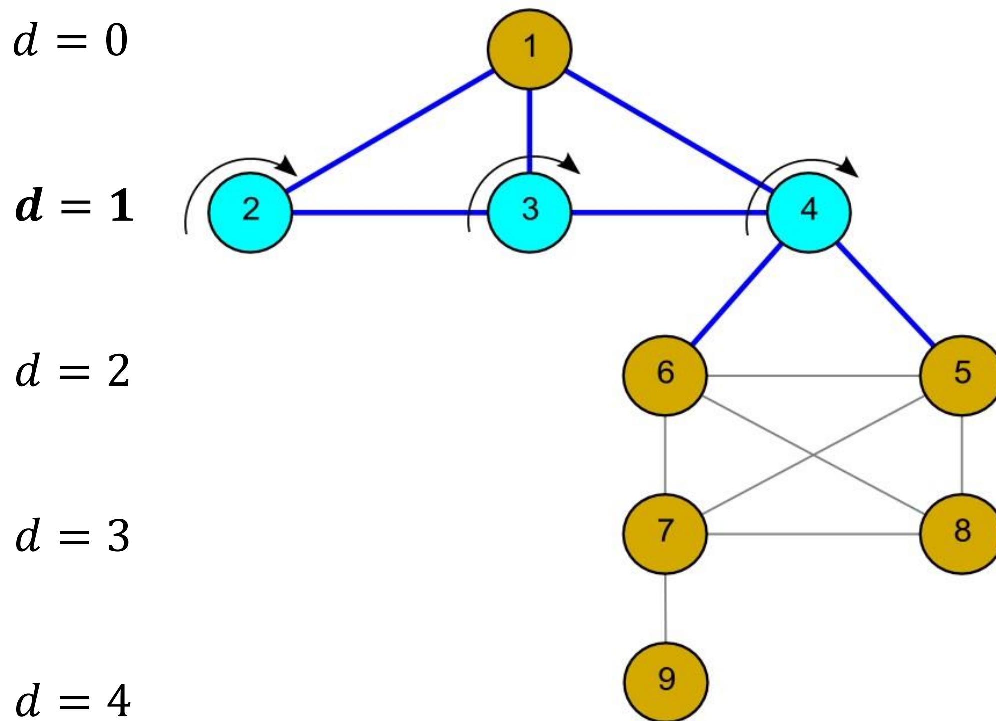
- Work-efficient downward traversal



- Threads are assigned vertices ***in the frontier***
  - Use an explicit queue
- Variable number of edges to traverse per thread

Fine-grained Parallelization Strategy

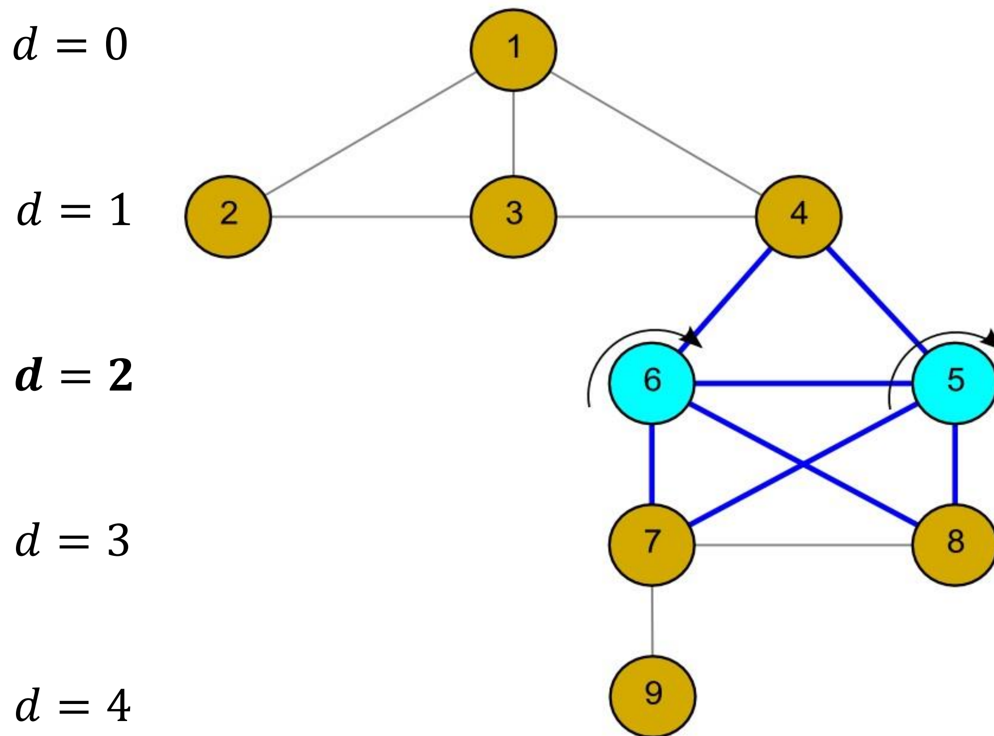
- Work-efficient downward traversal



- Threads are assigned vertices ***in the frontier***
  - Use an explicit queue
- Variable number of edges to traverse per thread

Fine-grained Parallelization Strategy

- Work-efficient downward traversal

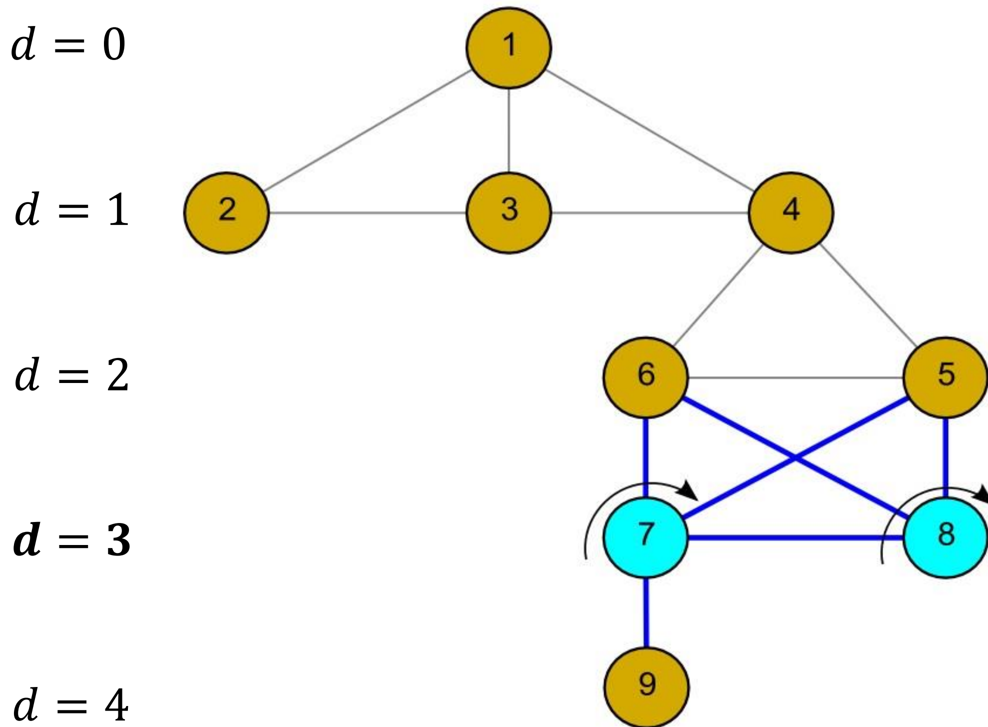


- Threads are assigned vertices ***in the frontier***
  - Use an explicit queue
- Variable number of edges to traverse per thread

Georgia Tech College of Computing

Fine-grained Parallelization Strategy

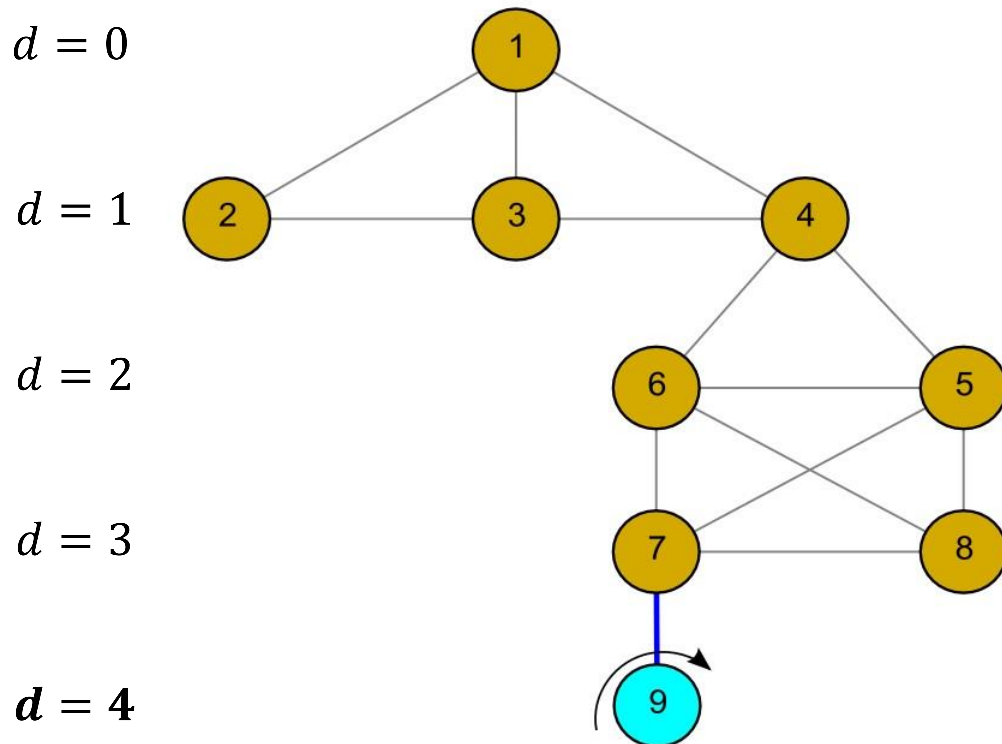
- Work-efficient downward traversal



- Threads are assigned vertices ***in the frontier***
  - Use an explicit queue
- Variable number of edges to traverse per thread

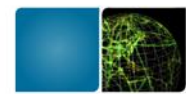
Fine-grained Parallelization Strategy

- Work-efficient downward traversal



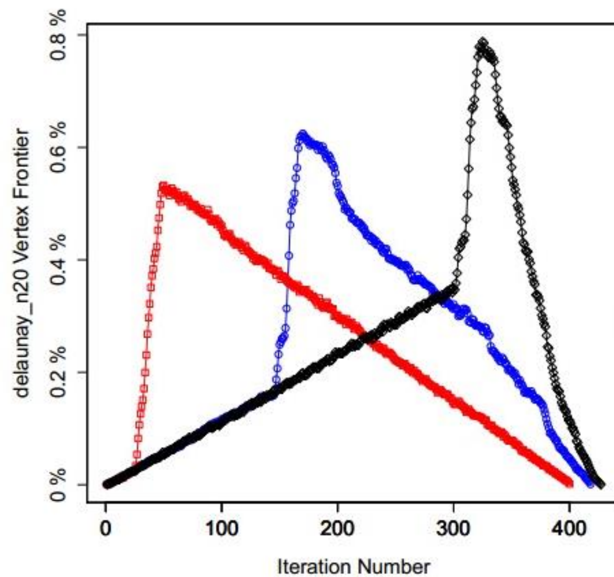
- Threads are assigned vertices ***in the frontier***
  - Use an explicit queue
- Variable number of edges to traverse per thread

Georgia Tech College of Computing

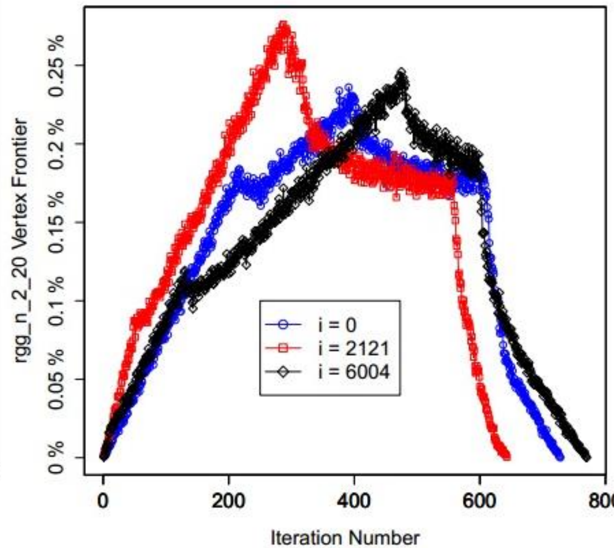


Motivation for Hybrid Methods

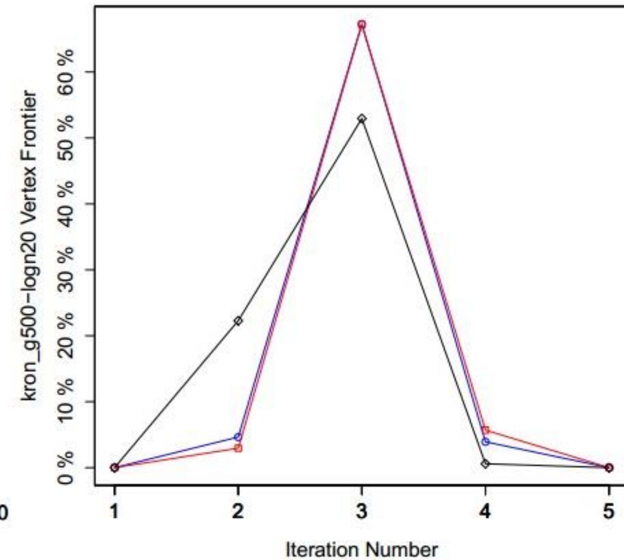
- No one method of parallelization works best



(a) delaunay_n20



(b) rgg_n_2_20



(c) kron_g500-logn20

- High diameter: Only do useful work
- Low diameter: Leverage memory bandwidth

Sampling Approach

- Idea: Processing one source vertex takes $O(m+n)$ time
  - Can process a small sample of vertices fast!
- Estimate the diameter of the graph's connected components
  - Store the maximum BFS distance found from each of the first k vertices
  - $diameter \approx median(distances)$
- Completes useful work rather than preprocessing the graph!

Experimental Setup

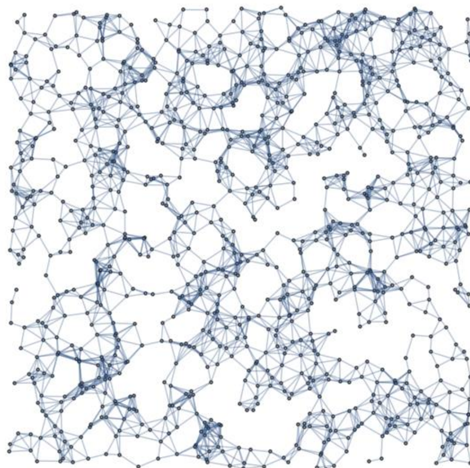
- Single-node
  - CPU (4 Cores)
    - Intel Core i7-2600K
    - 3.4 GHz, 8MB Cache
  - GPU
    - NVIDIA GeForce GTX Titan
    - 14 SMs, 837 MHz, 6 GB GDDR5
    - Compute Capability 3.5
- Multi-node (KIDS)
  - CPUs (2 x 4 Cores)
    - Intel Xeon X5560
    - 2.8 GHz, 8 MB Cache
  - GPUs (3)
    - NVIDIA Tesla M2090
    - 16 SMs, 1.3 GHz, 6 GB GDDR5
    - Compute Capability 2.0
  - Infiniband QDR Network
- All times are reported in seconds

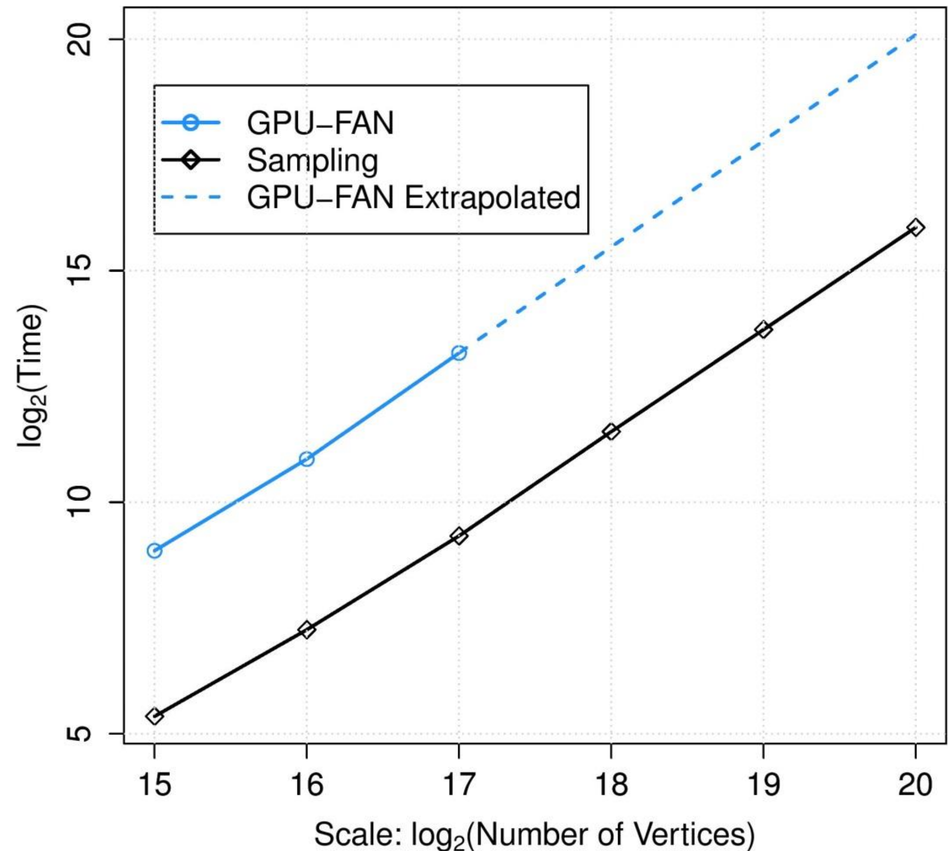
Benchmark Data Sets

Name	Vertices	Edges	Diam.	Significance
af_shell9	504,855	8,542,010	497	Sheet Metal Forming
caidaRouterLevel	192,244	609,066	25	Internet Router Level
cnr-2000	325,527	2,738,969	33	Web crawl
com-amazon	334,863	925,872	46	Product co-purchasing
delaunay_n20	1,048,576	3,145,686	444	Random Triangulation
kron_g500-logn20	524,288	21,780,787	6	Kronecker Graph
loc-gowalla	196,591	1,900,654	15	Geosocial
luxembourg.osm	114,599	119,666	1,336	Road Network
rgg_n_2_20	1,048,576	6,891,620	864	Random Geometric
smallworld	100,000	499,998	9	Logarithmic Diameter

Scaling Results (rgg)

- Random geometric graphs
- Sampling beats GPU-FAN by 12x for all scales

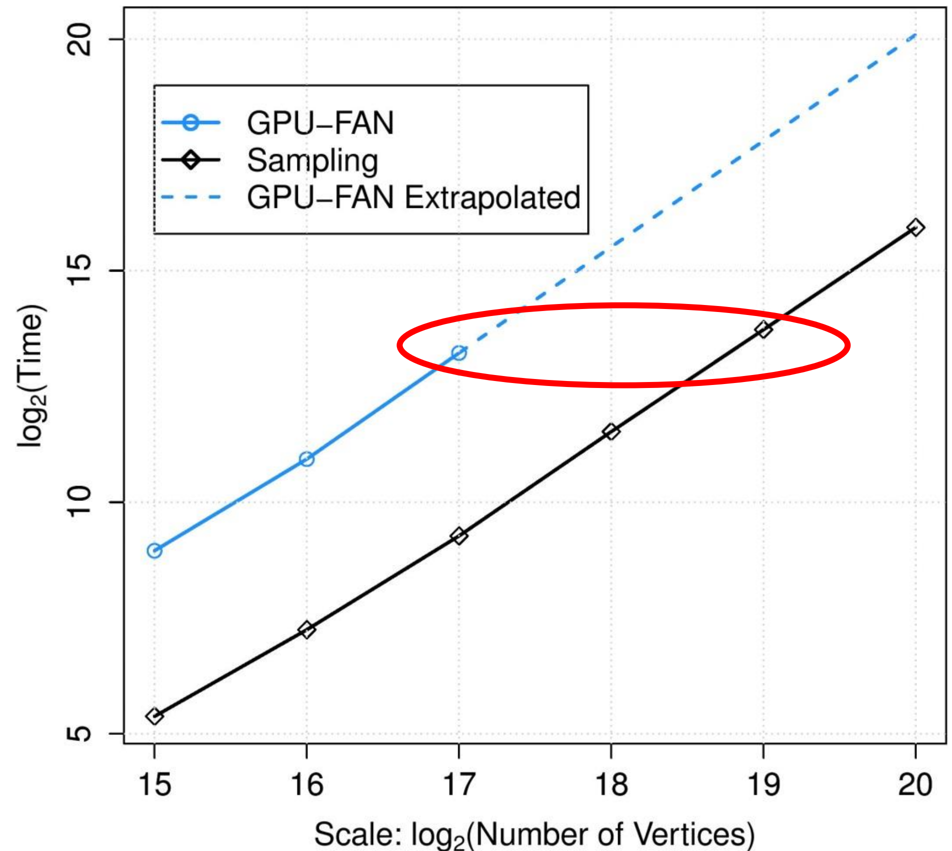




Georgia Tech College of Computing

Scaling Results (rgg)

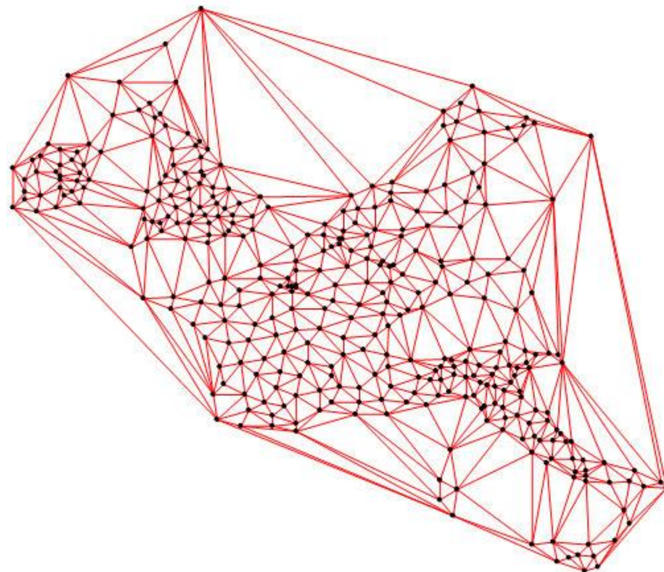
- Random geometric graphs
- Sampling beats GPU-FAN by 12x for all scales
- *Similar amount of time to process a graph 4x as large!*

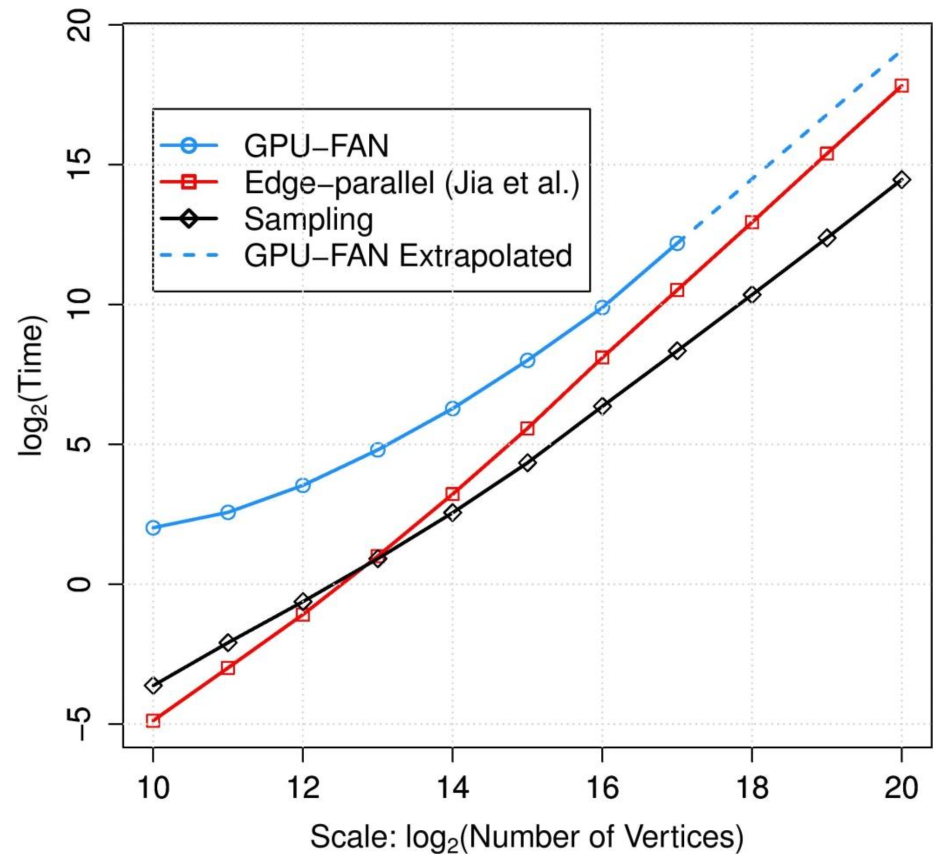


Georgia Tech College of Computing

Scaling Results (Delaunay)

- Sparse meshes
- Speedup grows with graph scale

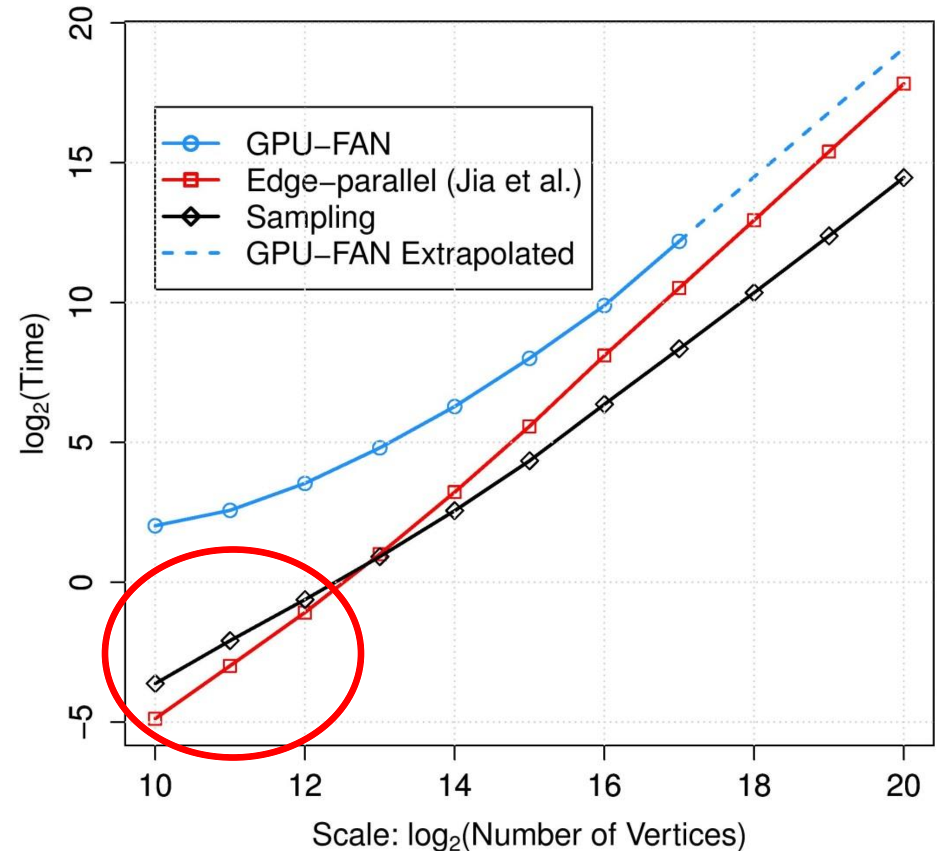




Georgia Tech College of Computing

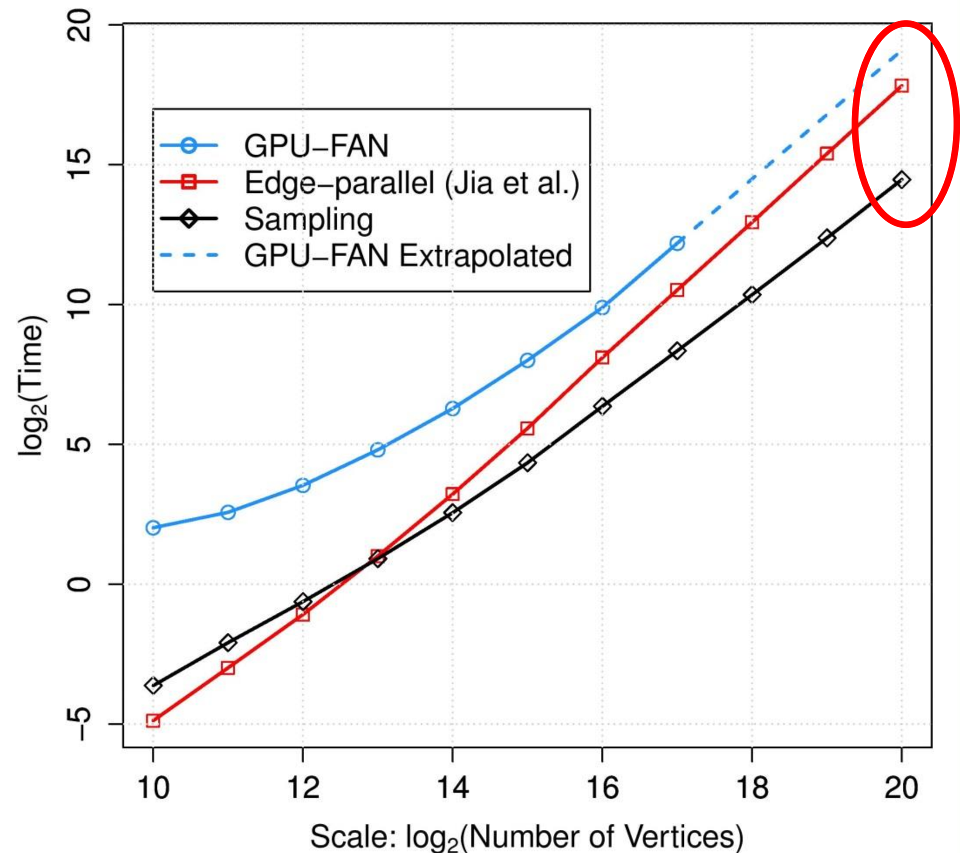
Scaling Results (Delaunay)

- Sparse meshes
- Speedup grows with graph scale
- When edge-parallel is best it's best by a matter of *ms*



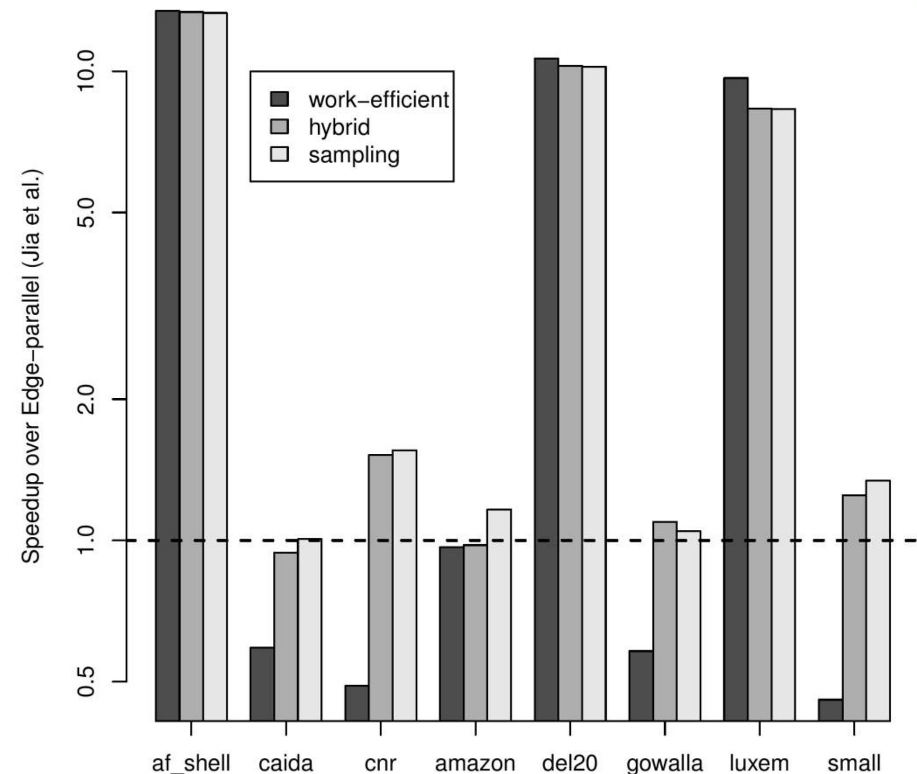
Scaling Results (Delaunay)

- Sparse meshes
- Speedup grows with graph scale
- When edge-parallel is best it's best by a matter of *ms*
- When sampling is best it's by a matter of *days*



Benchmark Results

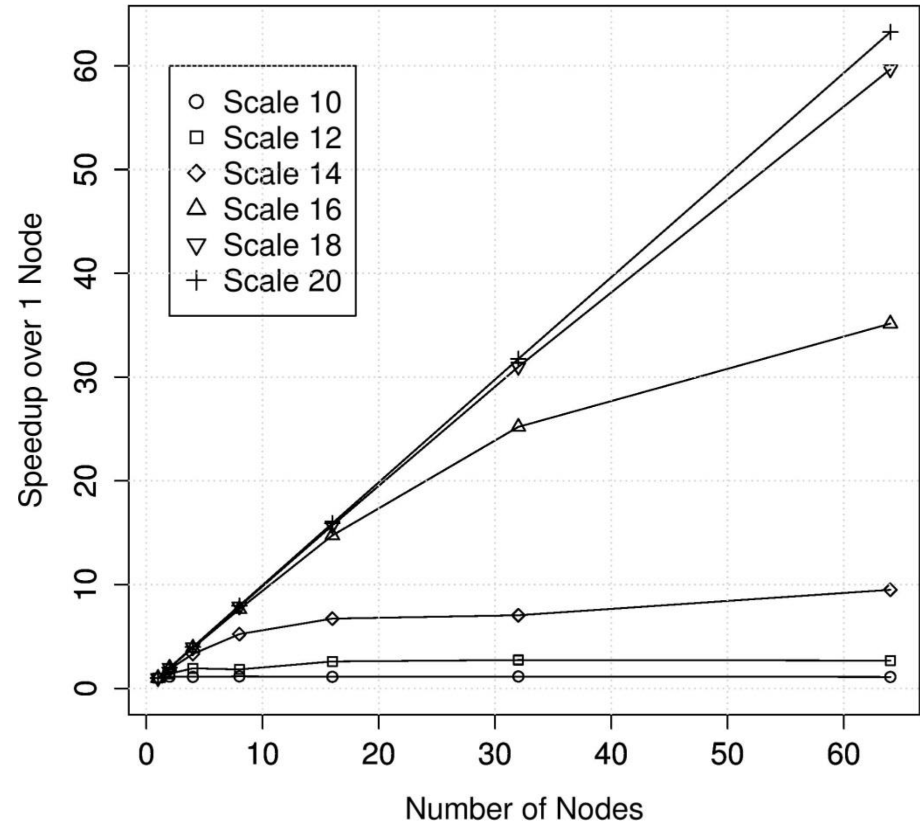
- Road networks and meshes see ~10x improvement
  - af_shell: 2.5 days → 5 hours
- Modest improvements otherwise
- 2.71x Average speedup



Georgia Tech College of Computing

Multi-GPU Results

- Linear speedups when graphs are sufficiently large
- 10+ GTEPS for 192 GPUs
- Scaling isn't unique to graph structure
  - Abundant coarse-grained parallelism



A Back of the Envelope Calculation…

- 192 Tesla M2090 GPUs
- 16 Streaming Multiprocessors per GPU
- Maximum of 1024 Threads per Block

- $192 * 16 * 1024 = 3{,}145{,}728$
- Over 3 million CUDA Threads!

Conclusions

- Work-efficient approach obtains **up to 13x speedup** for high-diameter graphs
- Tradeoff between work-efficiency and DRAM utilization maximizes performance
  - **Average speedup is 2.71x** for all graphs
- Our algorithms easily scale to many GPUs
  - Linear scaling on **up to 192 GPUs**
- Our results are **consistent across network structures**

Questions?

- Contact: Adam McLaughlin, Adam27X@gatech.edu
- Advisor: David A. Bader, bader@cc.gatech.edu

- Source code: https://github.com/Adam27X/hybrid_BC https://github.com/Adam27X/graph-utils

NVIDIA®

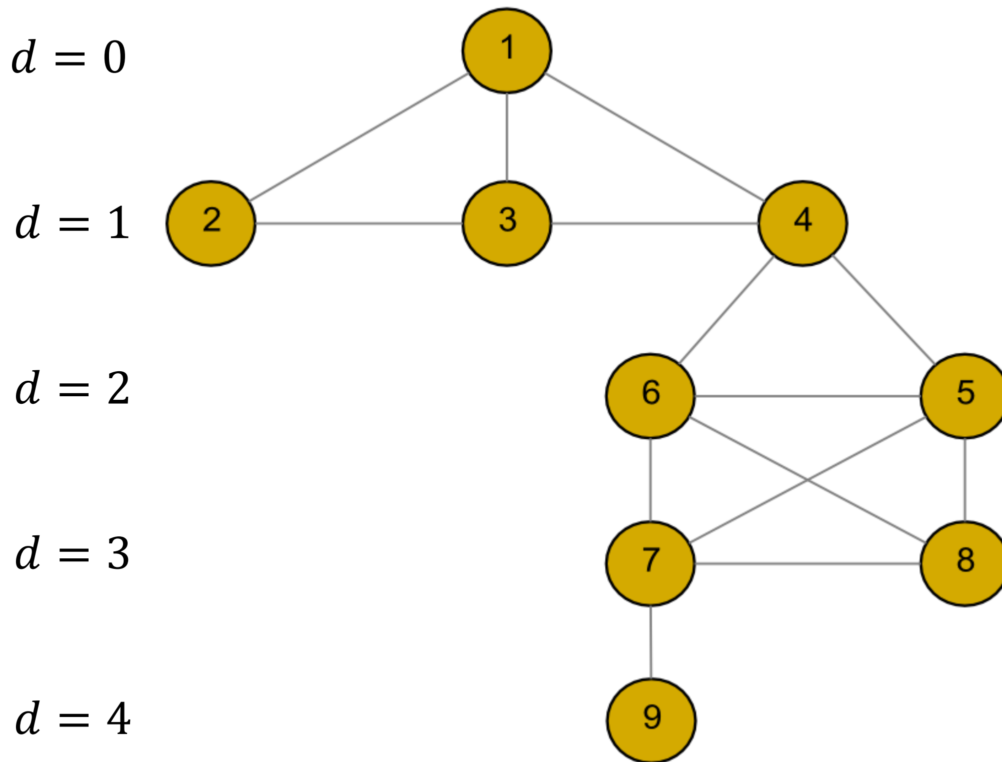
Backup

Contributions

- A **work-efficient algorithm** for computing Betweenness Centrality on the GPU
  - Works especially well for high-diameter graphs
- On-line **hybrid approaches** that coordinate threads based on graph structure
- An **average speedup of 2.71x** over the best existing methods
- A **distributed implementation** that **scales linearly to up to 192 GPUs**
- Results that are **performance portable across** the gamut of **network structures**

Brandes's Algorithm

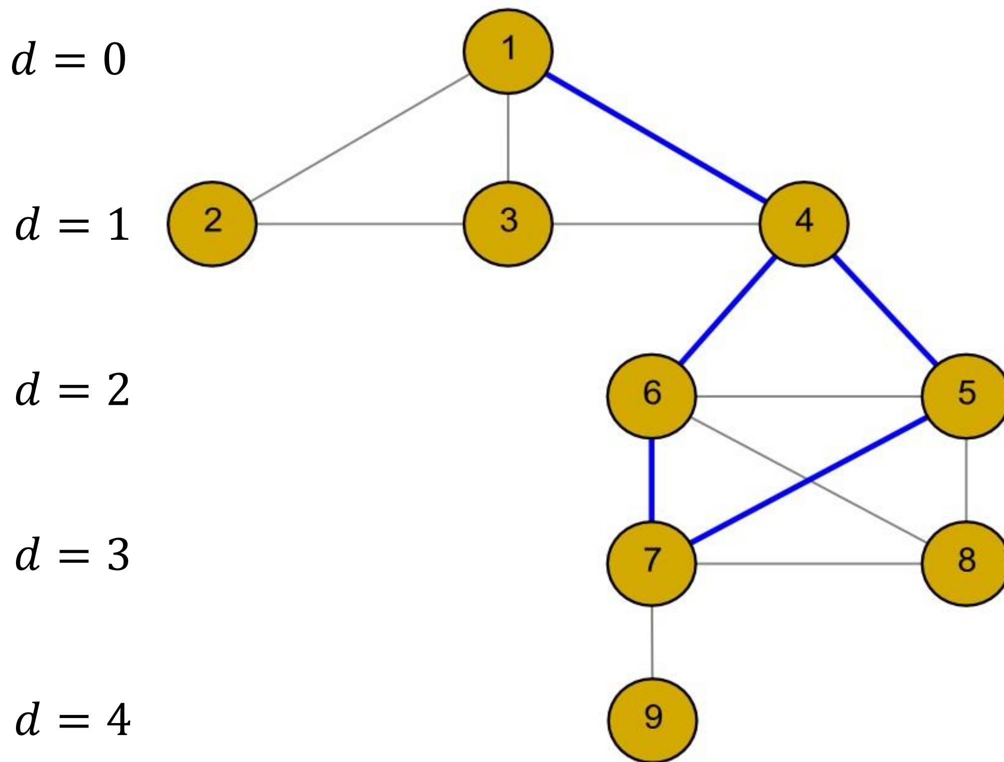
- Let vertex 1 be the source, s



- First, downward traversal from s
- Obtain the number of shortest paths from s to each vertex ($\sigma_{ss} = 1$)

Brandes's Algorithm

- Downward traversal from s

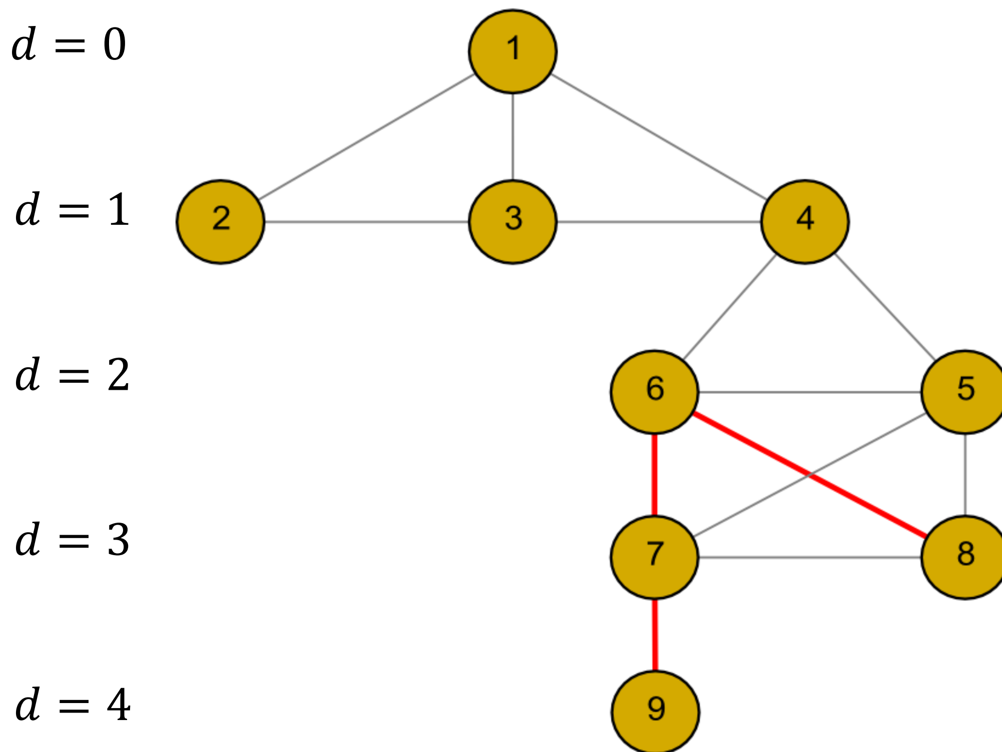


$$\sigma_{sw} = \sum_{v \in pred(w)} \sigma_{sv}$$

- $\sigma_{17} = \sigma_{15} + \sigma_{16}$
- $\sigma_{1\cdot} = [1,1,1,1,1,1,2,2,2]$

Brandes's Algorithm

- Upward dependency accumulation toward s

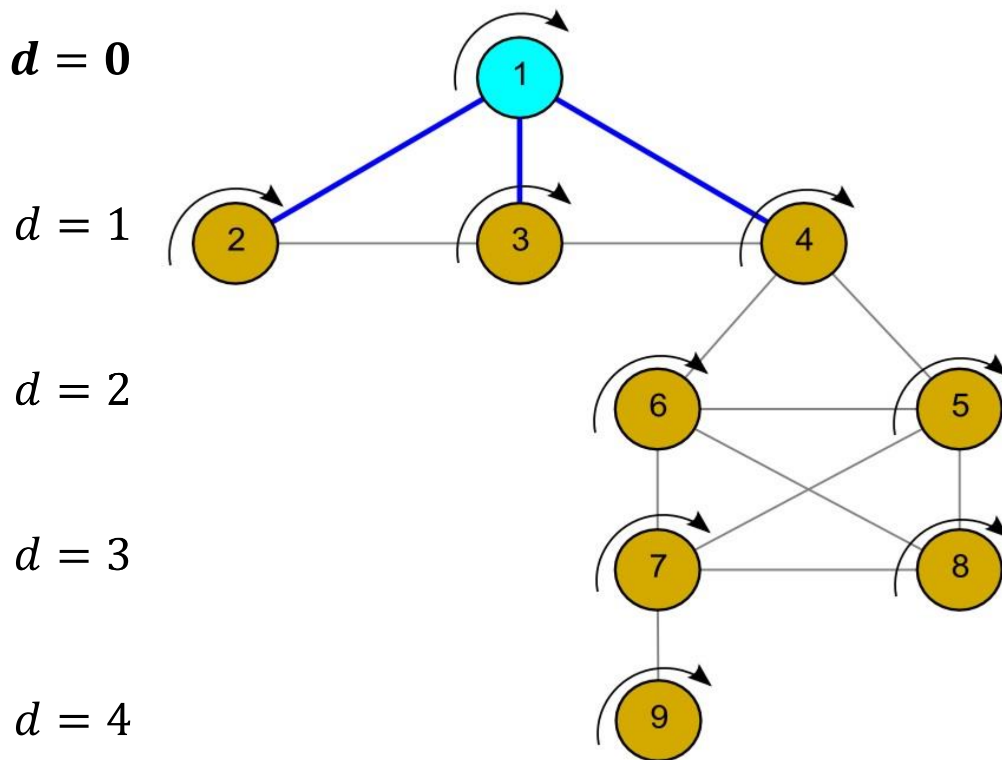


$$\delta_{sv} = \sum_{w \in succ(v)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{sw})$$

- $\delta_{16} = \frac{\sigma_{16}}{\sigma_{17}}(1 + \delta_{17}) + \frac{\sigma_{16}}{\sigma_{18}}(1 + \delta_{18})$
- $\delta_{1\cdot} = [8,0,0,5,\frac{3}{2},\frac{3}{2},1,0,0]$

Fine-grained Parallelization Strategy

- Vertex-parallel downward traversal

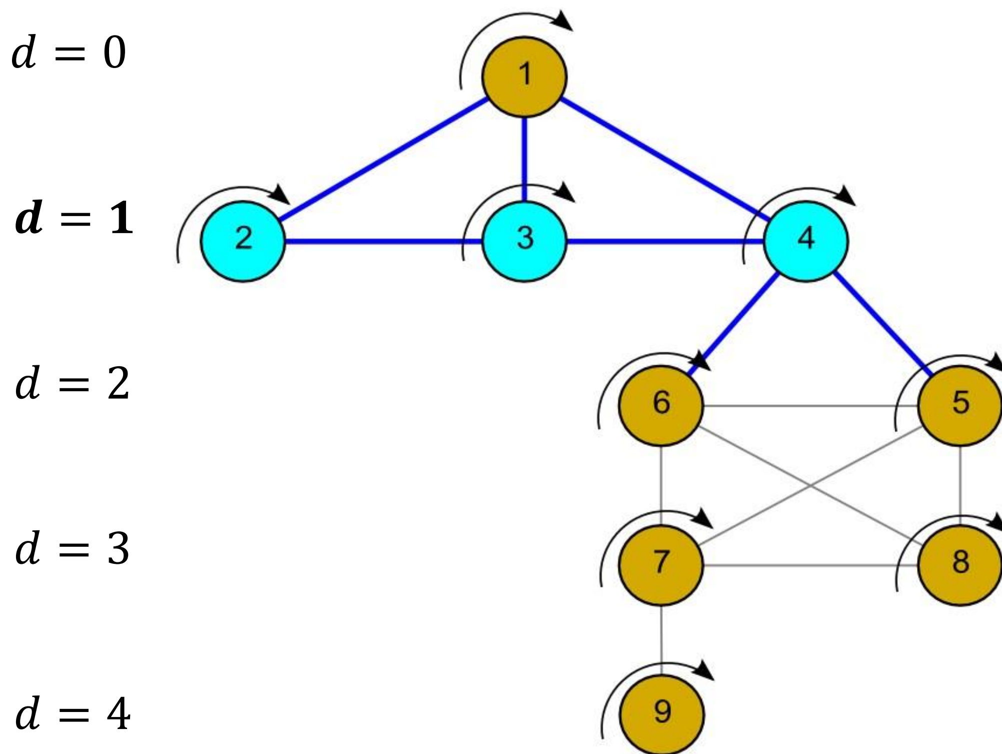


- Threads are assigned to each vertex
  - Only a subset is active
- Variable number of edges to traverse per thread

Georgia Tech College of Computing

Fine-grained Parallelization Strategy

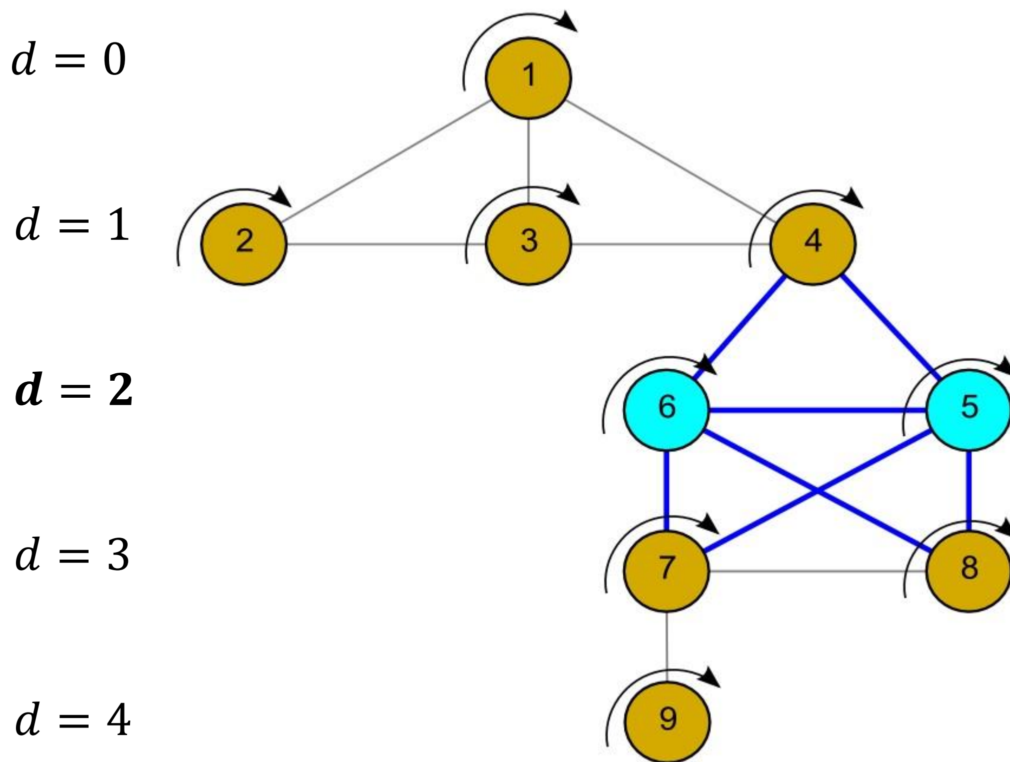
- Vertex-parallel downward traversal



- Threads are assigned to each vertex
  - Only a subset is active
- Variable number of edges to traverse per thread

Fine-grained Parallelization Strategy

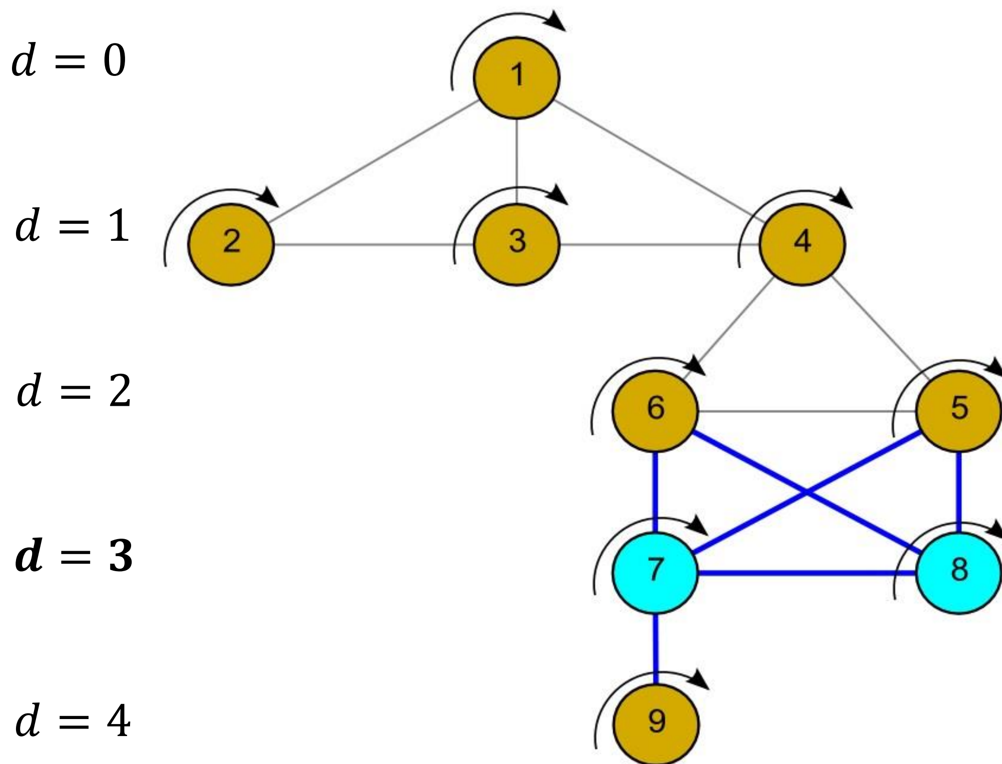
- Vertex-parallel downward traversal



- Threads are assigned to each vertex
  - Only a subset is active
- Variable number of edges to traverse per thread

Fine-grained Parallelization Strategy

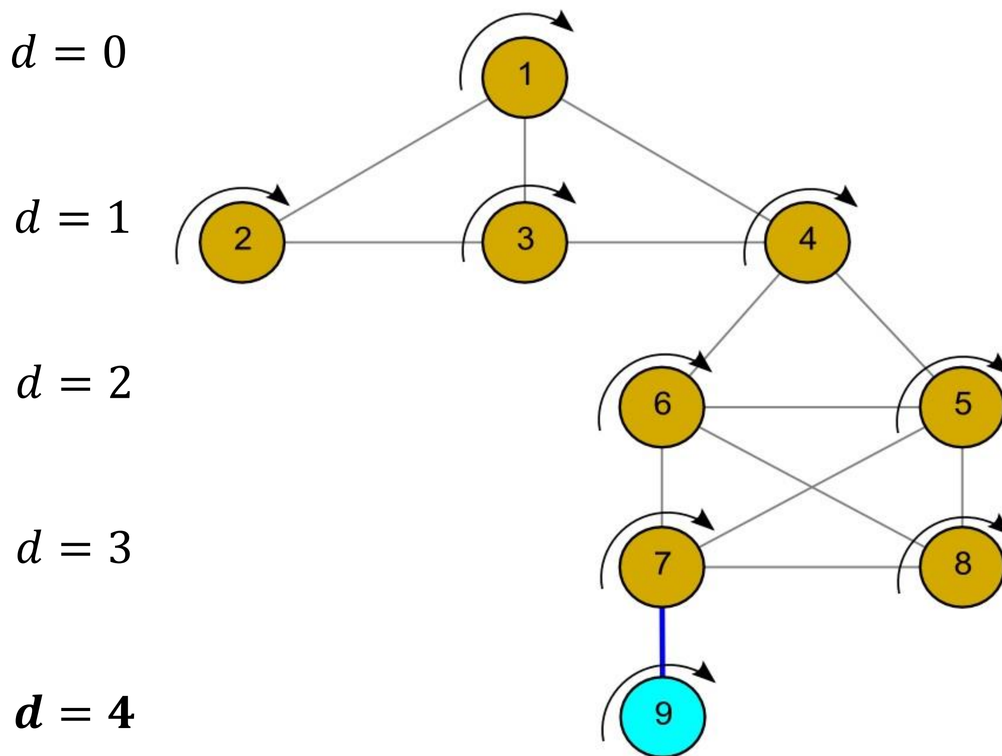
- Vertex-parallel downward traversal



- Threads are assigned to each vertex
  - Only a subset is active
- Variable number of edges to traverse per thread

Fine-grained Parallelization Strategy

- Vertex-parallel downward traversal



- Threads are assigned to each vertex
  - Only a subset is active
- Variable number of edges to traverse per thread

Georgia Tech College of Computing