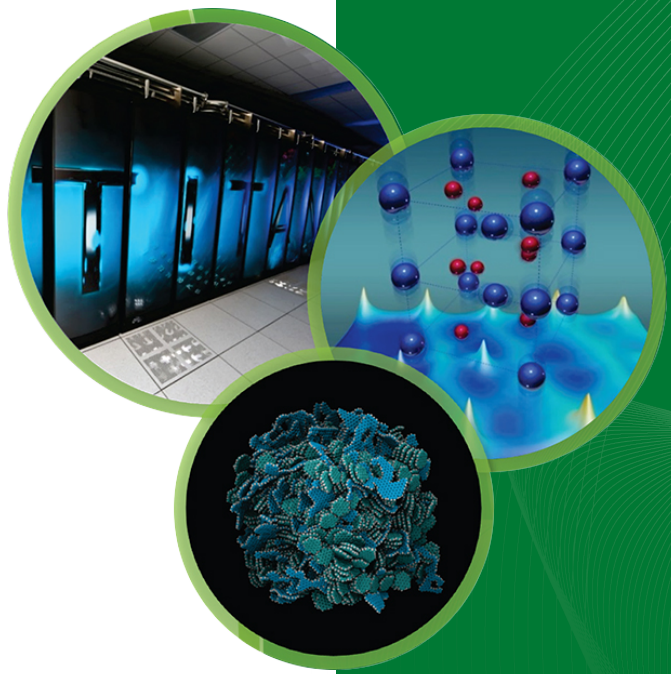
GPU-based Parallel Algorithm for Generating Massive Scale-free Networks Using the Preferential Attachment Model



Maksudul Alam **Kalyan S. Perumalla**

Discrete Computing Systems Group

Computer Science and Mathematics Division

Oak Ridge National Laboratory, TN, USA

IEEE BigGraphs, Boston | Dec 11, 2017

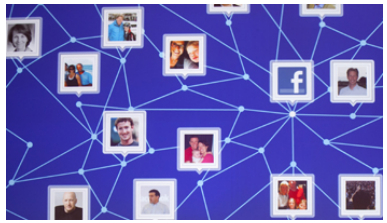
OAK RIDGE National Laboratory

Emergence of Large Graphs

Technological advancement led to

- Explosive growth of complex systems
- Availability of huge volume of data

316***M*** *active Twitter users*
500***M*** *tweets per day*



1.49***B*** *active Facebook users*
4.75***B*** *content per day*

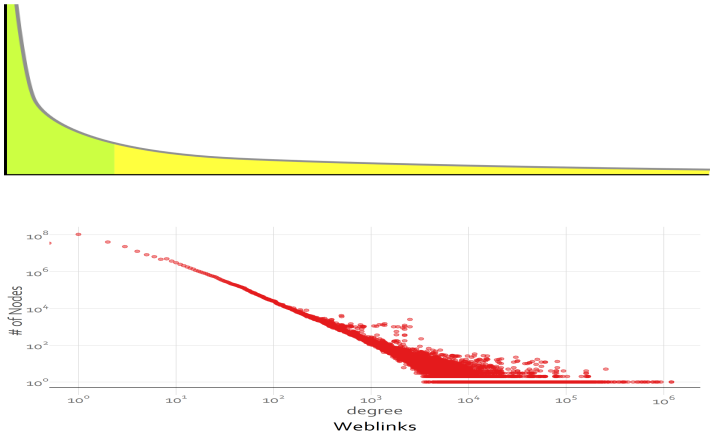
3.3***B*** *active Internet users*
958***B*** *websites,* 3.5***B*** *searches/day*

- Interactions among entities can be modeled by graphs
- Large complex systems and big data lead to ***large graphs***
- Some ***patterns*** and ***behavior*** emerge only in large graphs

OAK RIDGE National Laboratory

Scale-Free Graph Models and GPU-based Generation

Various random graph models exist to capture the scale-free property



- Barabási–Albert
- Copy-Model
- Recursive Matrix (RMAT)
- Stochastic Kronecker
- Etc.

GPU-Based Graph Generation

- Few GPU-based generators exist for networks
- No GPU-based generators exist for **scale-free** networks

Challenges in GPU-based Execution

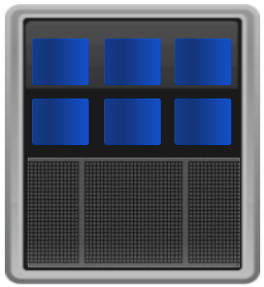
- Using shared memory efficiently
- Load balancing across many threads
- Reducing thread and block synchronization overhead

[1]GPU=Graphical Processing Unit

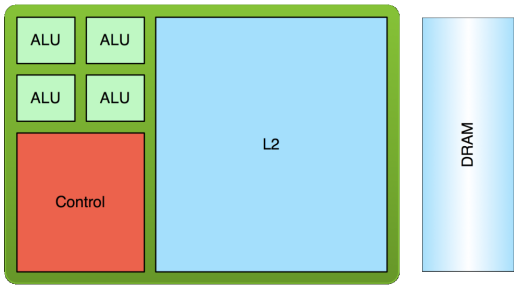
OAK RIDGE National Laboratory

Central Processing Unit (CPU) vs. Graphical Processing Unit (GPU)

- GPUs are highly parallel, multi-threaded, many-core processors
  - Offer high throughput data processing *Single Instruction Multiple Data (**SIMD**)*
  - Extensively used in big data analytics and time-critical scientific computing



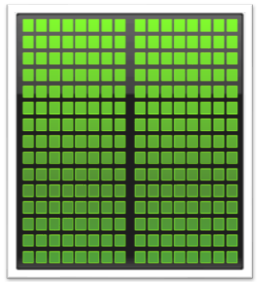
Cores



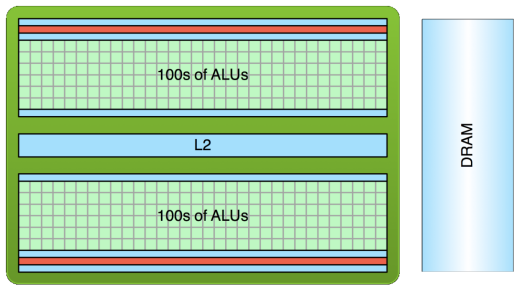
Architecture

CPU

- Optimized for low-latency memory access and coarse-grained threads
- General-purpose, suitable for many applications



Cores



Architecture

GPU

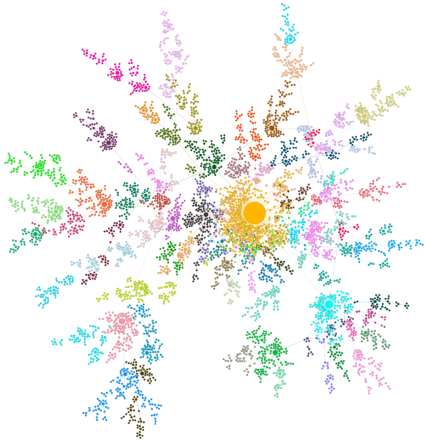
- Optimized for fine-grained data-parallel SIMD execution
- Programmed via special interface

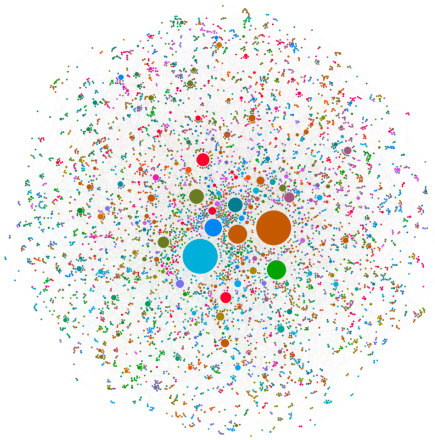
  E.g., *Common Unified Memory Architecture (**CUDA**)*

OAK RIDGE National Laboratory

Contributions in this Paper

- Developed an efficient GPU–based network generator, called **cuPPA** *[**CU**DA-based **P**arallel **P**referential **A**ttachment]*
- Generated graphs with 2B edges in less than 3 seconds using a single GPU



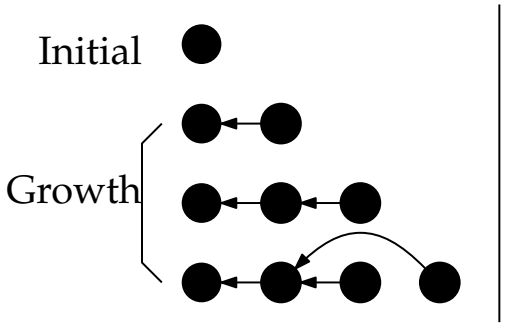


OAK RIDGE National Laboratory

Background: Preferential Attachment using the *Copy Model*

- One node is added at a time using the following steps:
  - A new node t is being added ($0 \leq t < n$)
  - $F(t) =$ a random node to which the new node t connects ($F(t) < t$)
    - $F(t)$ is the "outgoing end" of the edge from t i.e., $(t, F(t))$

1. Step 1: Randomly select $k \in [1, t-1]$
2. Step 2: Determine $F(t)$ as follow:
   - **Direct Edge**: $F(t) = k$ with probability p
   - **Copy Edge**: $F(t) = F(k)$ with probability $1 - p$
3. Step 3: Repeat the above d times to create d edges with t

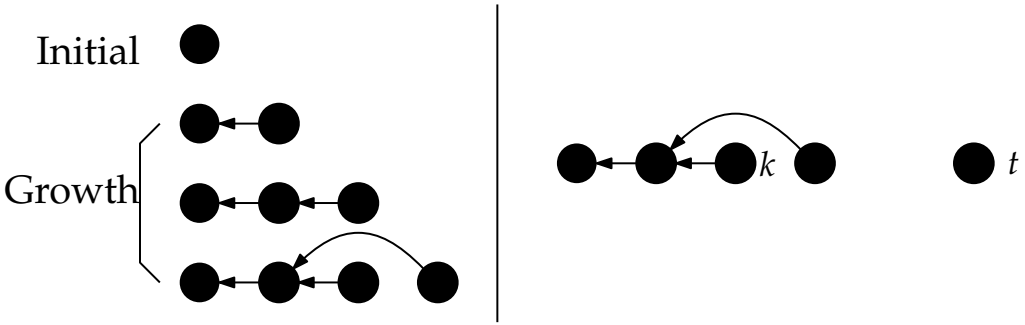


OAK RIDGE National Laboratory

Background: Preferential Attachment using the *Copy Model*

- One node is added at a time using the following steps:
  - A new node t is being added ($0 \leq t < n$)
  - $F(t) =$ a random node to which the new node t connects ($F(t) < t$)
    - $F(t)$ is the "outgoing end" of the edge from t i.e., $(t, F(t))$

1. Step 1: Randomly select $k \in [1, t-1]$
2. Step 2: Determine $F(t)$ as follow:
   - **Direct Edge**: $F(t) = k$ with probability p
   - **Copy Edge**: $F(t) = F(k)$ with probability $1 - p$
3. Step 3: Repeat the above d times to create d edges with t

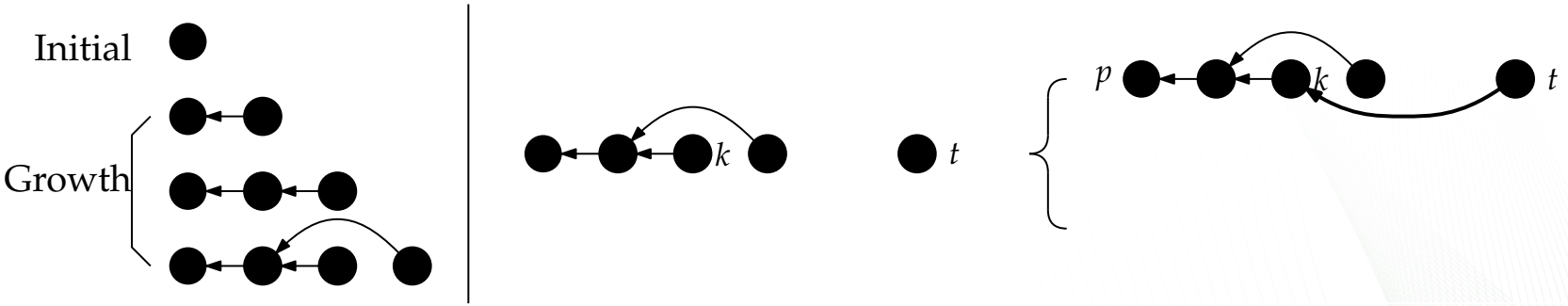


OAK RIDGE National Laboratory

Background: Preferential Attachment using the *Copy Model*

- One node is added at a time using the following steps:
  - A new node t is being added ($0 \leq t < n$)
  - $F(t) =$ a random node to which the new node t connects ($F(t) < t$)
    - $F(t)$ is the "outgoing end" of the edge from t i.e., $(t, F(t))$

1. Step 1: Randomly select $k \in [1, t-1]$
2. Step 2: Determine $F(t)$ as follow:
   - **Direct Edge**: $F(t) = k$ with probability p
   - **Copy Edge**: $F(t) = F(k)$ with probability $1 - p$
3. Step 3: Repeat the above d times to create d edges with t

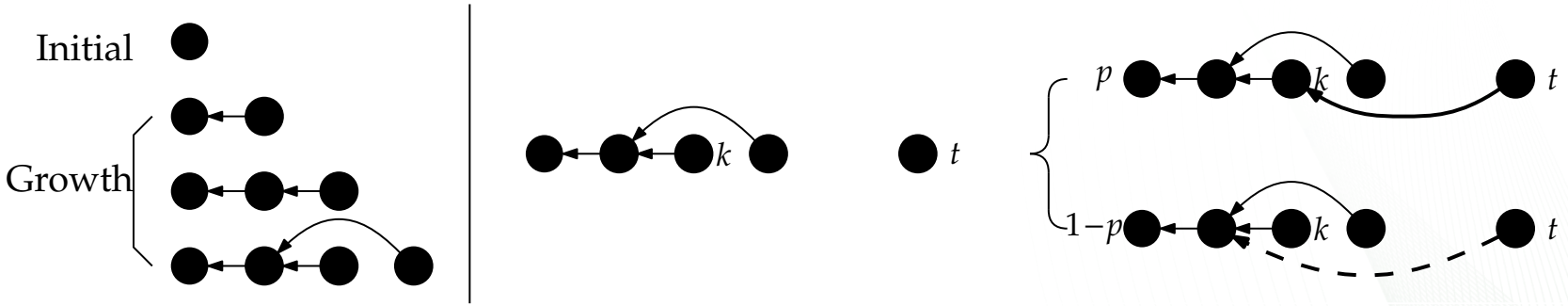


OAK RIDGE National Laboratory

Background: Preferential Attachment using the *Copy Model*

- One node is added at a time using the following steps:
  - A new node t is being added ($0 \leq t < n$)
  - $F(t) =$ a random node to which the new node t connects ($F(t) < t$)
    - $F(t)$ is the "outgoing end" of the edge from t i.e., $(t, F(t))$

1. Step 1: Randomly select $k \in [1, t-1]$
2. Step 2: Determine $F(t)$ as follow:
   - **Direct Edge**: $F(t) = k$ with probability p
   - **Copy Edge**: $F(t) = F(k)$ with probability $1 - p$
3. Step 3: Repeat the above d times to create d edges with t

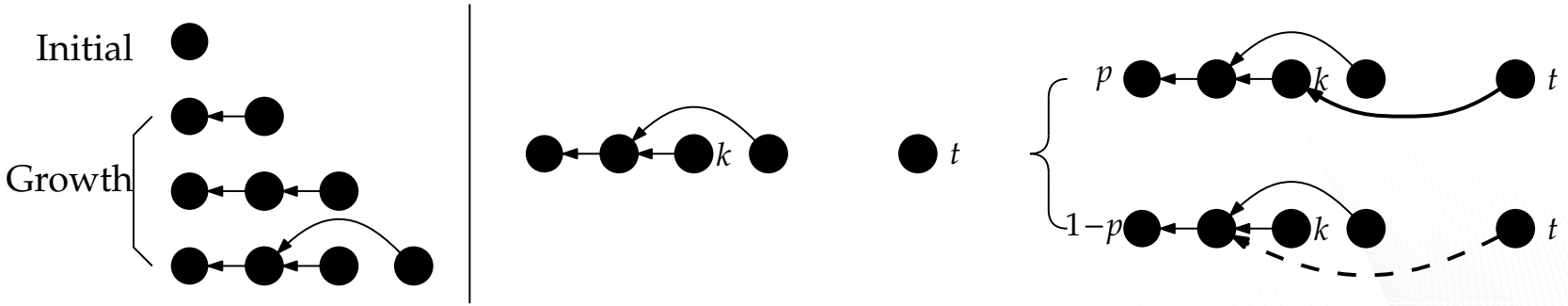


OAK RIDGE National Laboratory

Background: Preferential Attachment using the *Copy Model*

- One node is added at a time using the following steps:
  - A new node t is being added ($0 \leq t < n$)
  - $F(t) =$ a random node to which the new node t connects ($F(t) < t$)
    - $F(t)$ is the "outgoing end" of the edge from t i.e., $(t, F(t))$

1. Step 1: Randomly select $k \in [1, t-1]$
2. Step 2: Determine $F(t)$ as follow:
   - **Direct Edge**: $F(t) = k$ with probability p
   - **Copy Edge**: $F(t) = F(k)$ with probability $1 - p$
3. Step 3: Repeat the above d times to create d edges with t

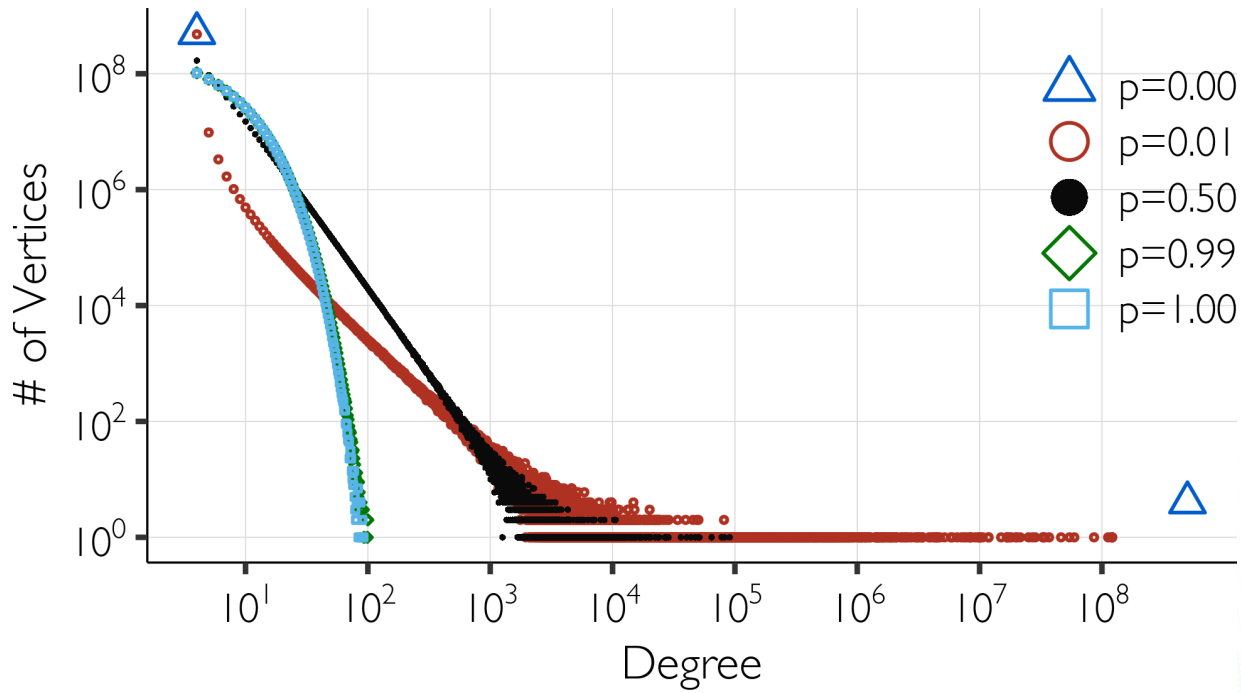


OAK RIDGE National Laboratory

Background: Properties of the Copy Model

- More general than the Barabási–Albert Model
- When $p = \frac{1}{2}$ Copy Model = Barabási–Albert Model

Alam, Khan and Marathe, "Distributed-memory parallel algorithms for generating massive scale-free networks…," SC'13



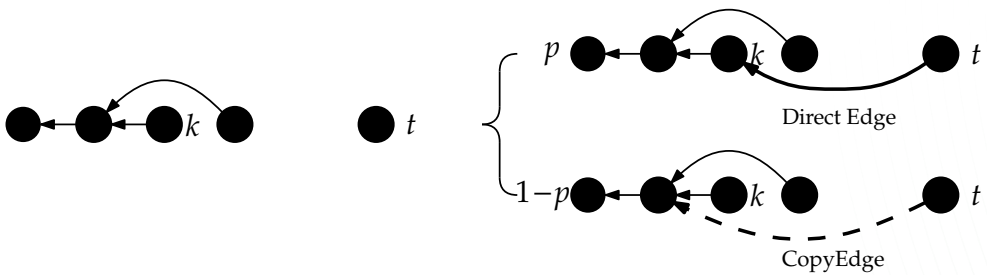
Degree distribution of the generated networks for $n = 250M$, $d = 4$, *and* $p = [0.0, 0.01, 0.50, 0.99, 1.00]$ *in* $\log - \log$ *scale.*

OAK RIDGE National Laboratory

cuPPA: Parallel Algorithm

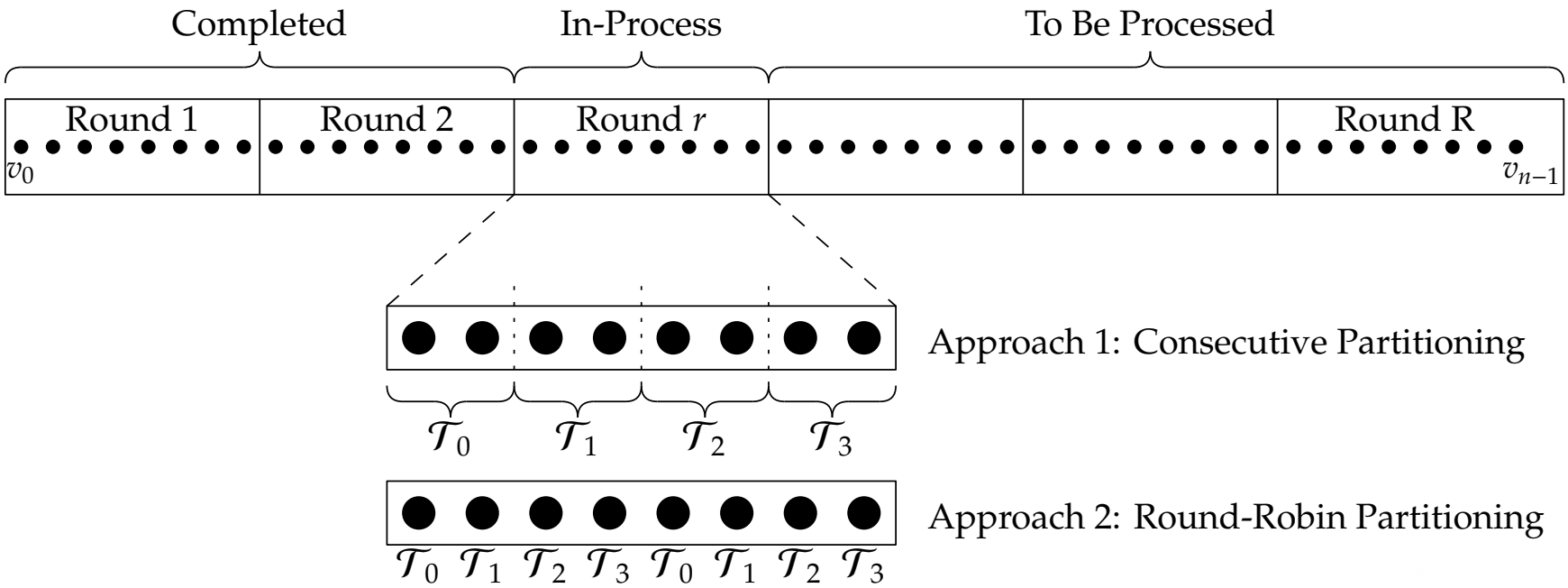
- Generate the edges in a series of R rounds
- In each round r, process $n_r = \frac{n}{R}$ vertices using T available GPU threads

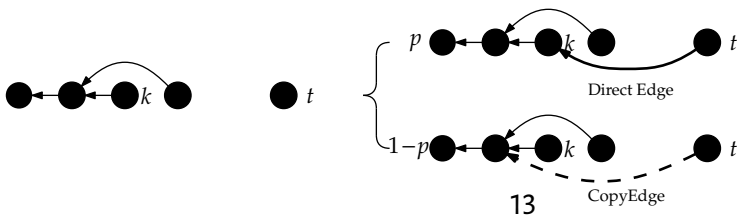
1. Execute Copy Model on each vertex in current round in parallel:
   - Direct edges are generated immediately
   - Copy edges may have some dependencies
     1. ***No dependency***, if the chosen node k is processed in previous rounds
     2. ***Only case of dependency:*** k in current round and $F(k)$ unresolved
   - Put the edges with dependencies in a queue called ***Waiting Queue*** to be processed later
2. Process items on the Waiting Queue after executing Copy Model



OAK RIDGE National Laboratory

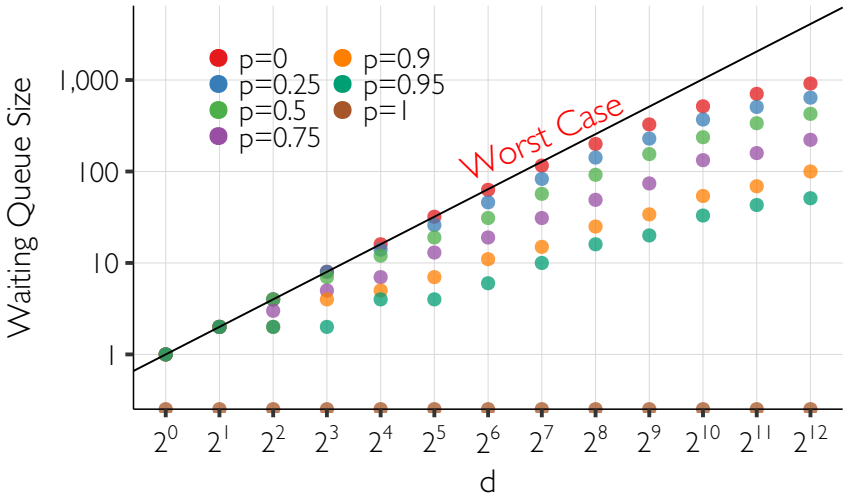
cuPPA: Parallel Algorithm



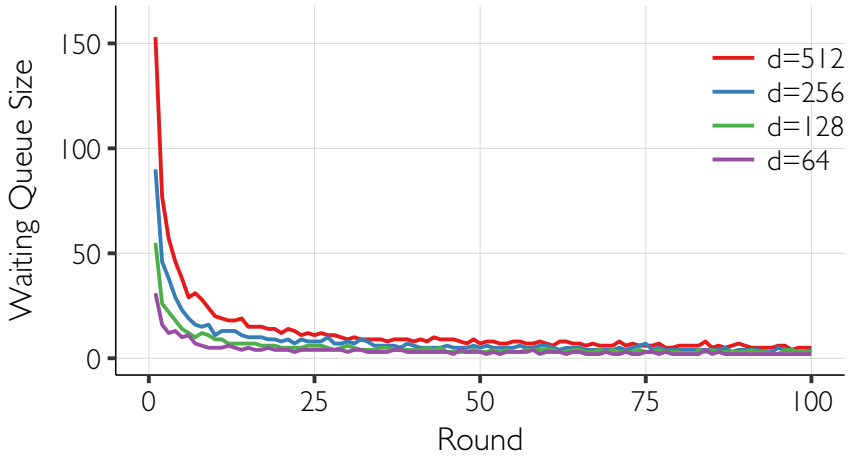


OAK RIDGE National Laboratory

cuPPA: Dynamic Load Balancing



Maximum queue size per vertex increases with d

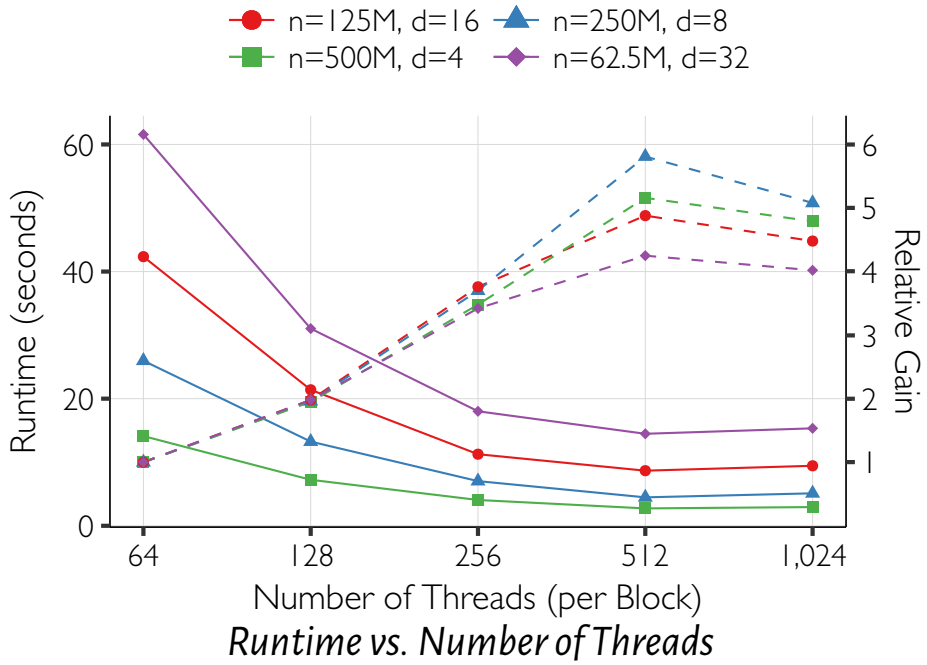


Queue size decreases significantly with progressive rounds

- In the worst case, a Waiting Queue with capacity d per vertex is required
  - Let waiting queue capacity be C items per thread
  - Due to limited GPU shared memory, we can only process $\frac{C}{d}$ vertices per thread
- In reality, number items placed in the waiting queue reduces drastically with rounds
- More vertices can be processed per thread
- Start with smaller number of vertices, increase with rounds

OAK RIDGE National Laboratory

cuPPA: Tradeoff between Hardware Concurrency and Waiting Queue

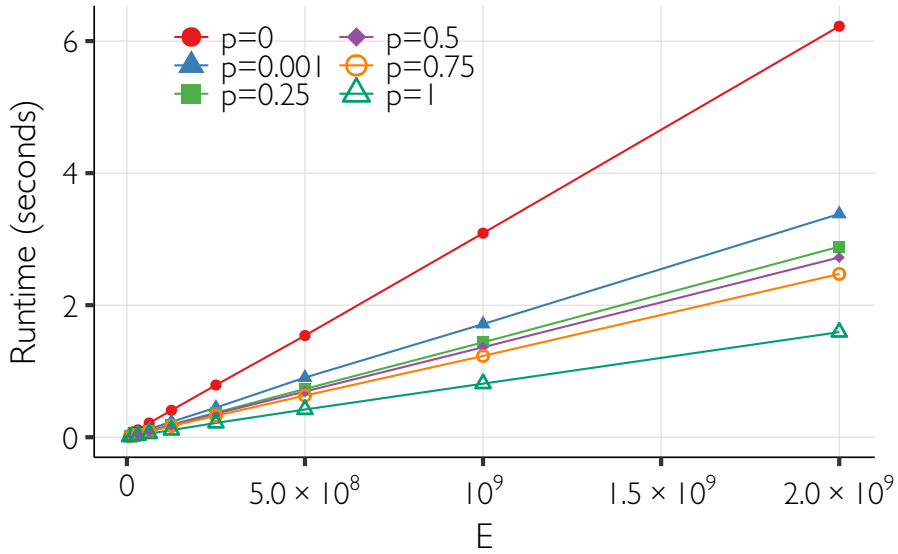


Runtime vs. Number of Threads

- Increasing the number of threads increases relative gain in speed
- Most gain is observed with 512 threads per block

OAK RIDGE National Laboratory

cuPPA: Runtime vs. Number of Vertices with varying p

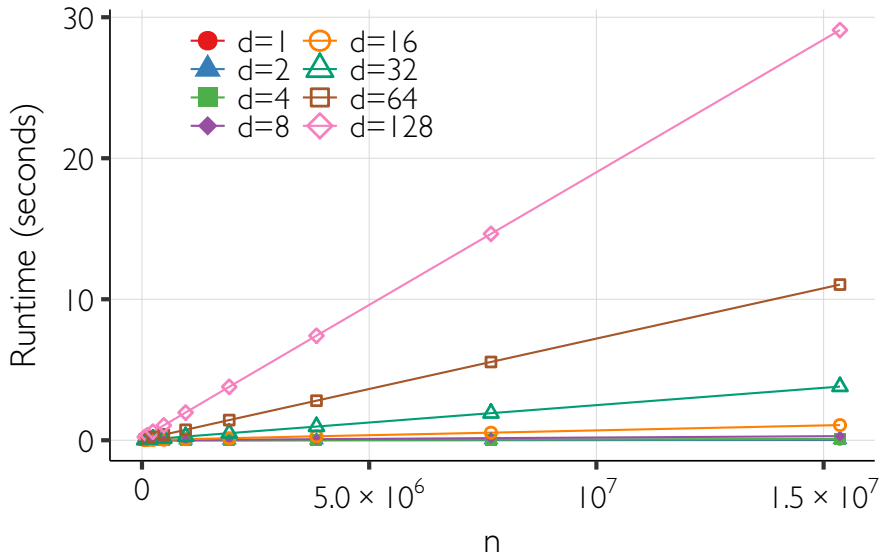


Runtime vs. Number of Vertices (varying p)

- Runtime varies linearly with increasing number of vertices
- Scales to a large number of vertices

OAK RIDGE National Laboratory

cuPPA: Runtime vs. Number of Vertices with varying d

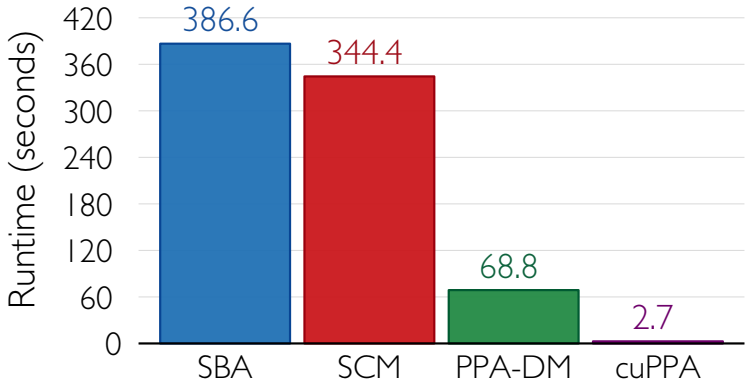


Runtime vs. Number of Vertices (varying d)

- Runtime varies linearly with increasing number of vertices
- Scales to a large number of vertices

OAK RIDGE National Laboratory

cuPPA: Runtime Comparison



Runtime of for generating 2B edges ($n = 500M, d = 4$)

Systems

	CPU	GPU
Make	AMD Opteron	NVIDIA GeForce
Model	6174	1080
Clock	800 MHz	1607 MHz
Memory	64 GB	8 GB
Compilation	g++ -O3	CUDA 8 nvcc -O3

Generators Compared

	Generator	Hardware
SBA	Sequential Barabási-Albert	1 CPU core
SCM	Sequential Copy-Model	1 CPU core
PPA-DM	Distributed–memory based	24 CPU cores, MPI
cuPPA	GPU-based	1 GPU

OAK RIDGE National Laboratory

cuPPA: Summary of Results

- **Contributions**
  - Designed and implemented GPU-based parallel algorithm
    - Efficient and scalable
    - Generates massive networks: 2 billion edges in 2.7 seconds
  - Performed theoretical and experimental analysis

- **Future Work**
  - Code profiling for further optimizations
  - Using multiple-GPU to generate large networks
  - Algorithms for hybrid CPU-GPU architechtures
  - Network conversion to other popular formats

OAK RIDGE National Laboratory

cuPPA: Summary of Results

- **Contributions**
  - Designed and implemented GPU-based parallel algorithm
    - Efficient and scalable
    - Generates massive networks: 2 billion edges in 2.7 seconds
  - Performed theoretical and experimental analysis

- **Future Work**
  - Code profiling for further optimizations
  - Using multiple-GPU to generate large networks
  - Algorithms for hybrid CPU-GPU architechtures
  - Network conversion to other popular formats

OAK RIDGE National Laboratory