Expediting Peer-to-Peer Simulation using GPU

Di Niu, Zhengjun Feng

Apr. 14th, 2009

Outline

- An Introduction to P2P Content Distribution
- Overview of P2P Algorithms and Simulation
- GPU Algorithm Design
- Performance Evaluation
- Summary & Future Work
- Discussion

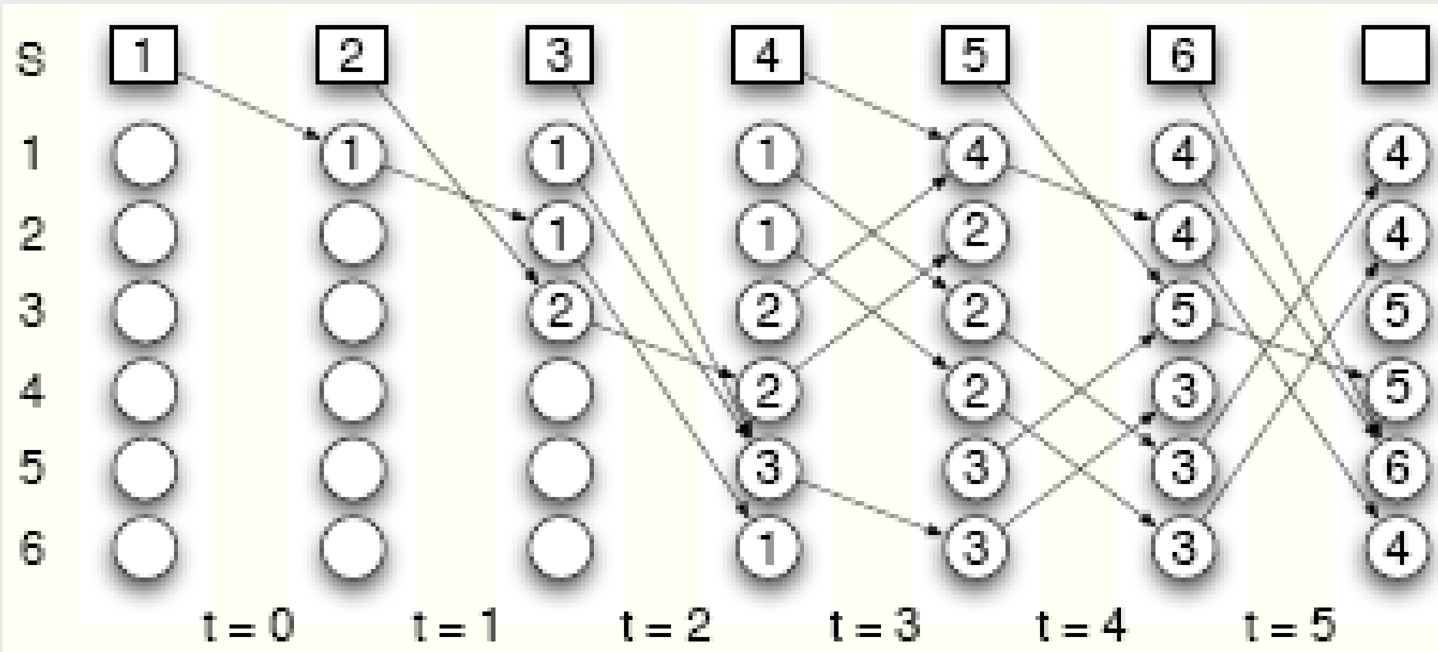
Introduction

- Current-generation P2P Content Distribution (BitTorrent)
  - A large file is broken into k blocks
  - N participating peers form a random topology
  - Neighboring Peers “gossip” the data blocks
  - Each peer has a buffer map to indicate which block it has
    - A binary array of size k.
    - B[x] = 1 means the peer has block x
  - Seed: Peers with all data blocks
  - Downloader: Peers with a subset of the data blocks.

P2P Algorithms: RUB & GRF

- Rand Useful Block (RUB): in each round, each peer selects a random neighbor and transmits a random useful block.
- Global Rarest First (GRF): in each round, each peer selects a random neighbor and transmits a random useful block that is the rarest within the neighborhood
  - Maintains a global counter for each block
  - Count[x] = 7 means there are 7 copies of block x in the network.

Simulating P2P Algorithms



- Simulation Methods: Synchronized, Unsynchronized
- Starts with one seed, ends with N seeds

RUB Design on GPU

- Without Shared Memory

  Each thread handles a peer:

  - Step 1: generate a random number
  - Step 2: get the random target peer
  - Step 3: compare data block matrix → count of difference
  - Step 4: generate another random number
  - Step 5: compare data block matrix again → data block index
  - Step 6: transmit the data block to target peer
  - Step 7: count the seeds

RUB Design on GPU

- With Shared Memory (store the buffer difference)

  Each thread handles a peer:

  - Step 1: generate a random number
  - Step 2: get the random target peer
  - Step 3: compare data block matrix, save different block index to SMEM
  - Step 4: generate another random number
  - Step 5: get the data block index from SMEM
  - Step 6: transmit the data block to target peer
  - Step 7: count the seeds

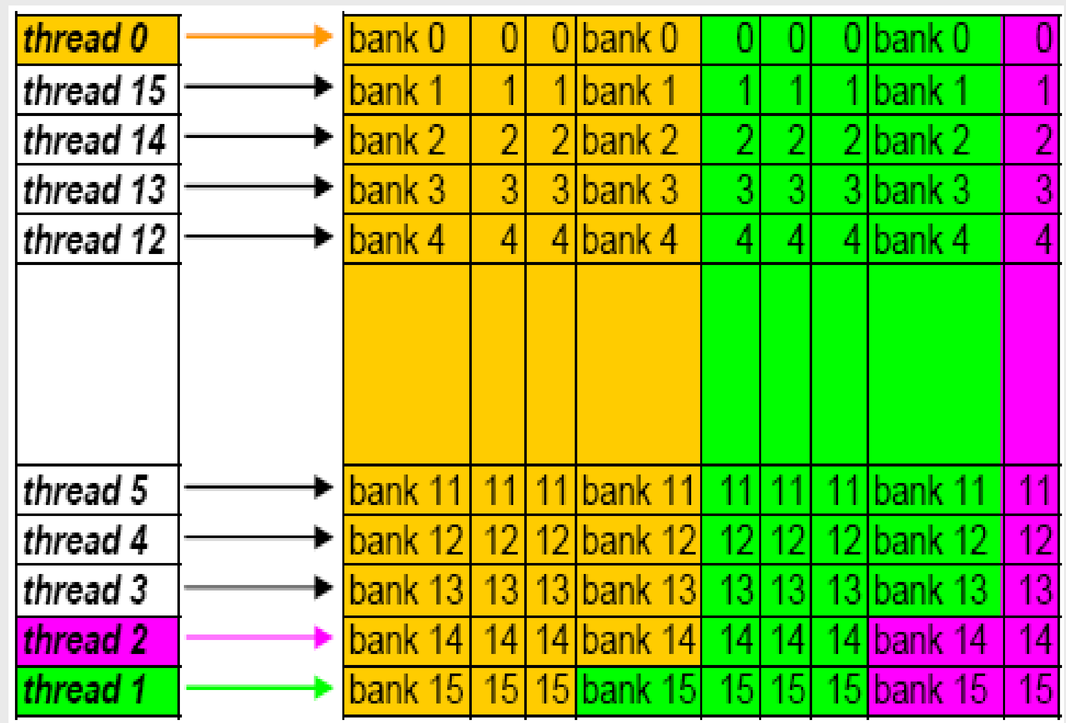
RUB Design on GPU

Shared Memory – Bank Conflict Avoidance

- Block size 64
- Data: USHORT
- 16KB/64=256B

  =128 data/thread

  (= 64 integers/thread)



To avoid bank conflict:

126 data / thread

(=63 integers/thread)

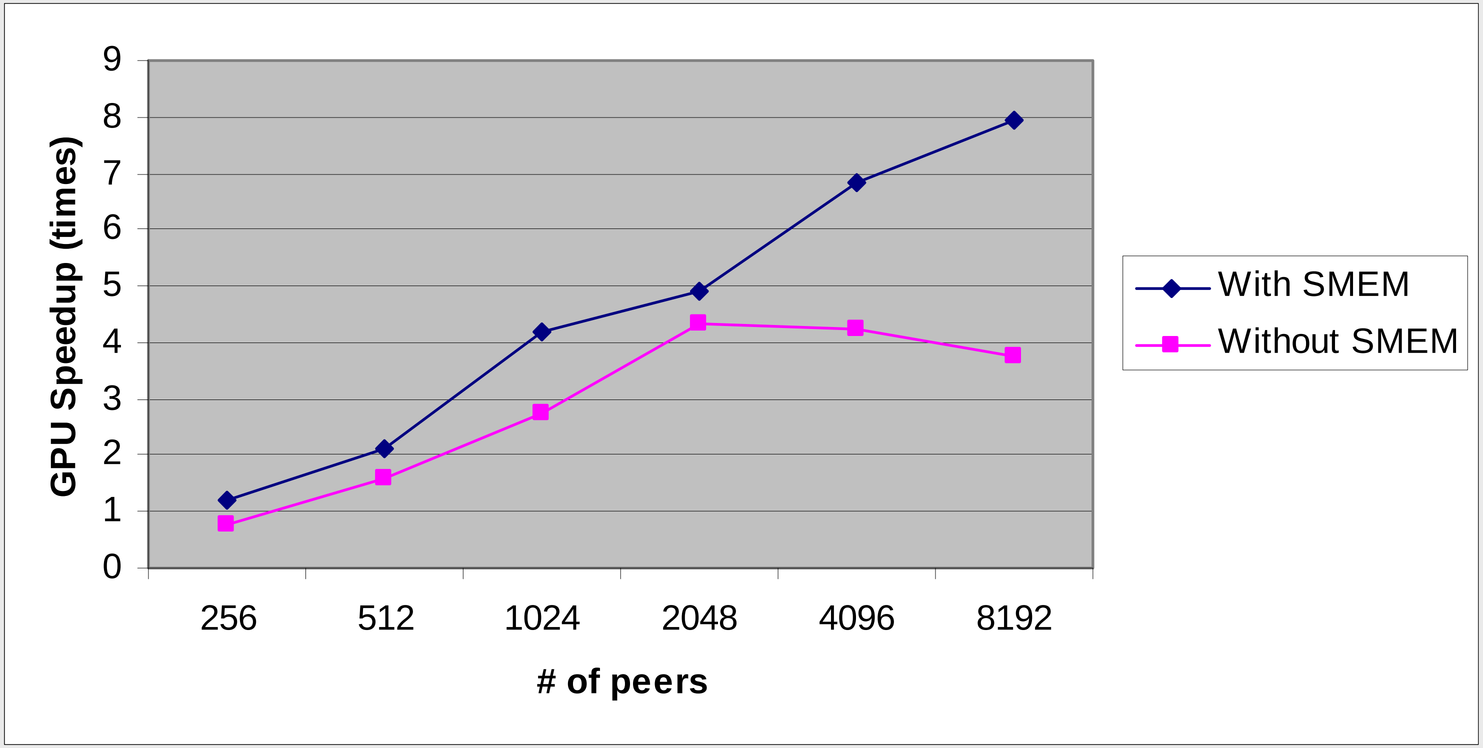
GRF on GPU: Algorithm 1

- Each thread handles one sender:
  - 1. Choose a random neighbor to upload to
  - 2. Compare the buffer difference between the sender and receiver
  - 3. Find the rarest block by checking Count (in global memory)
  - 4. Updating Count
  - 5. Updating the receiver buffer
- Use a separate kernel to update Count
  - This resolves the conflict of two peers transmitting the same block to the same peer.

GRF on GPU: Algorithm 2

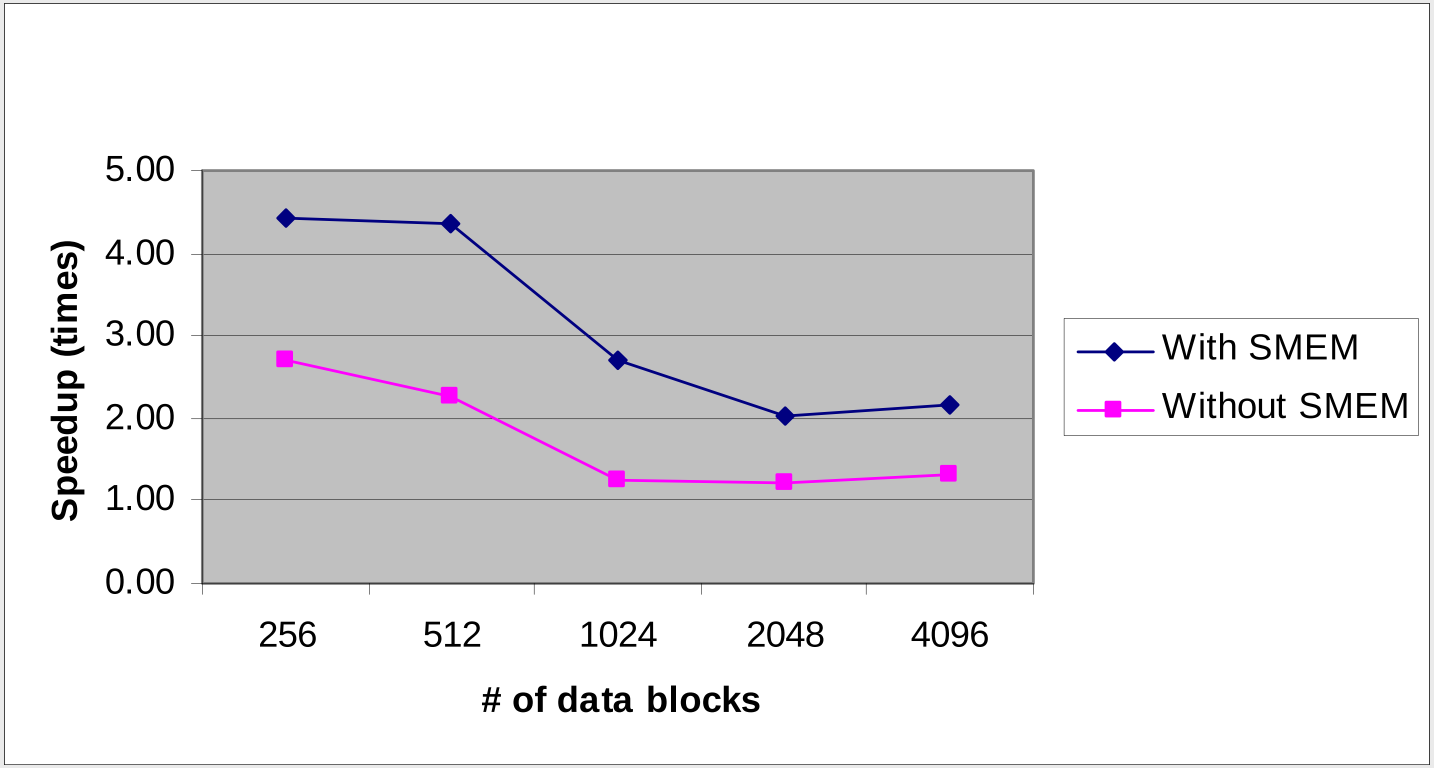
- Viewing each thread block as a sender
- Each thread handles one block in one sender
  - Step 1: Thread 0 in each sender selects a neighbor and stores receiver in the shared memory
  - Step 2: Buffer comparison. Each block in each sender is compared to the corresponding block in the receiver using one thread.
  - Step 3: Find the block that has the least Count
  - Step 4: Thread 0 in each sender transmits the rarest block to the receiver.

Evaluation – RUB Speedup



With SMEM: up to 8x speedup
W/o SMEM: up to 4.3x speedup

Evaluation – RUB Speedup

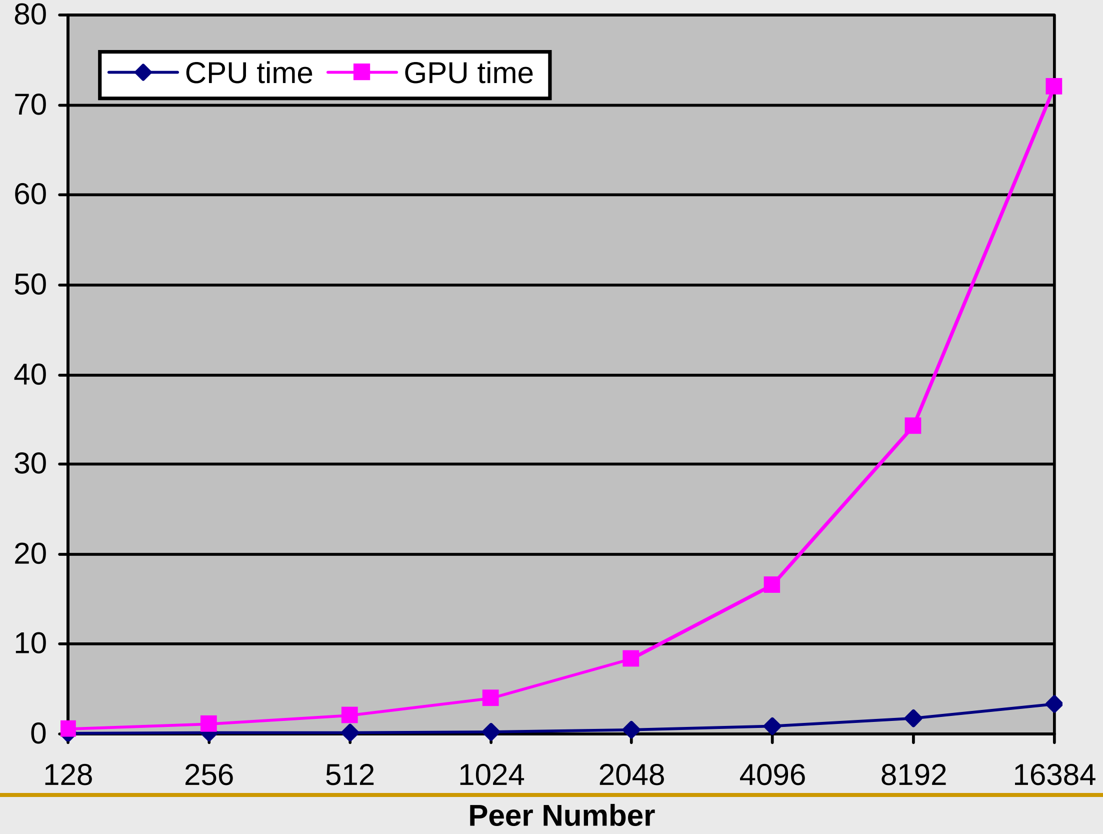


With SMEM: 2x ~ 4.5x speedup

W/O SMEM: 1.2x ~ 2.7 speedup

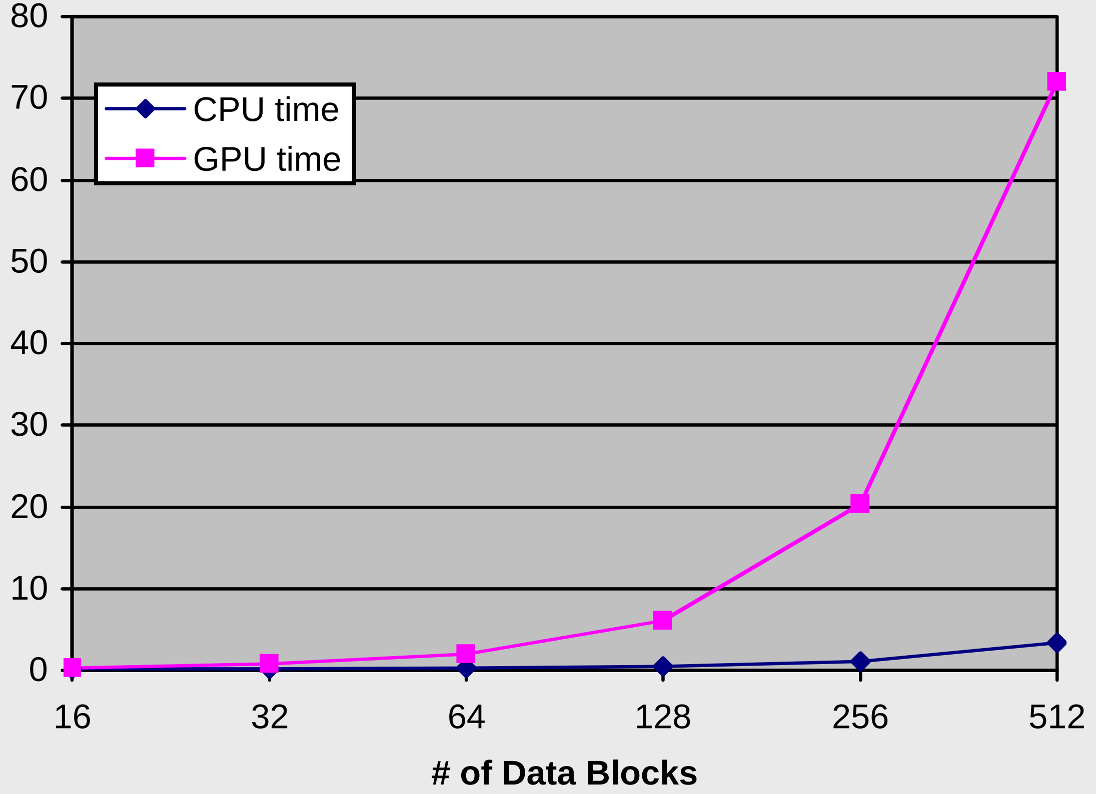
Evaluation - GRF

- Each thread handles a peer: 7x speedup
- Each thread handles a data block: 21x speedup

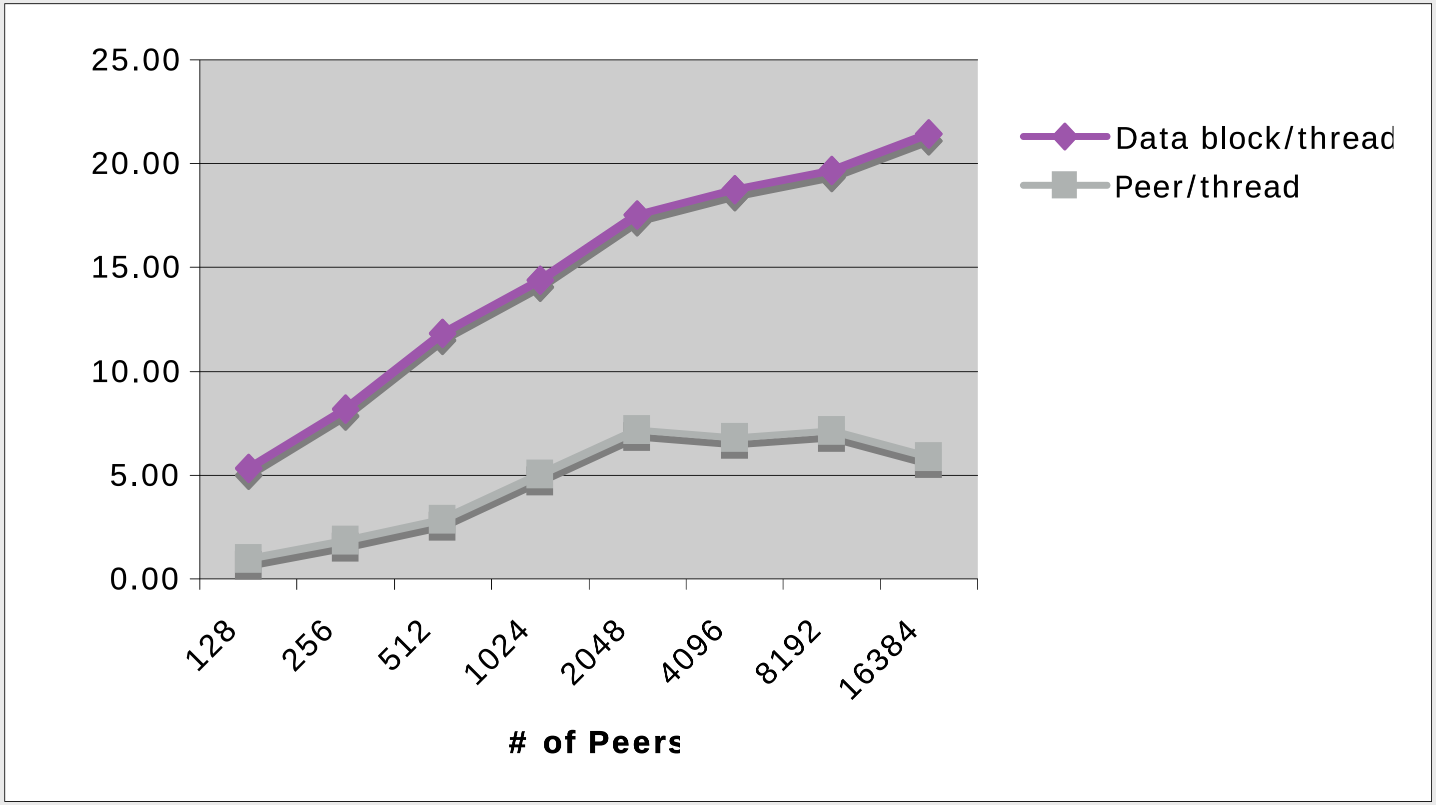


Evaluation - GRF

- Each thread handles a peer: 8x speedup
- Each thread handles a data block: 21x speedup



Evaluation - GRF Speedup

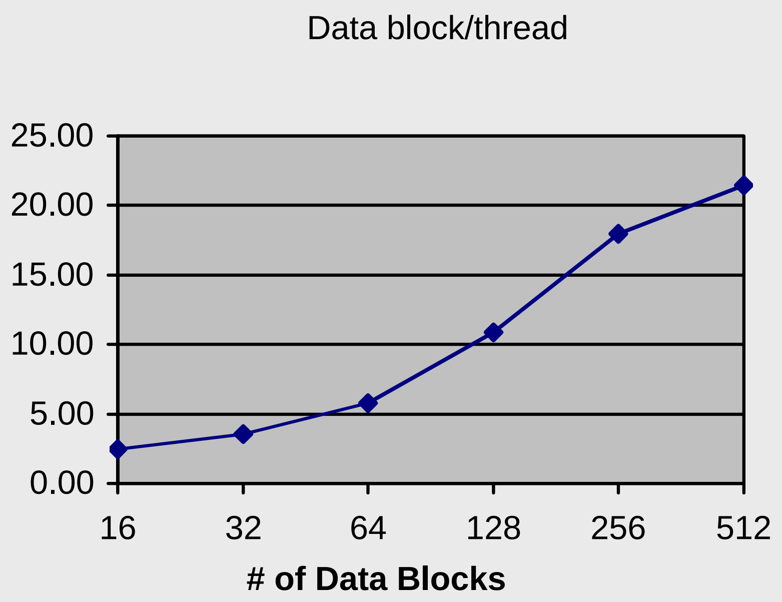


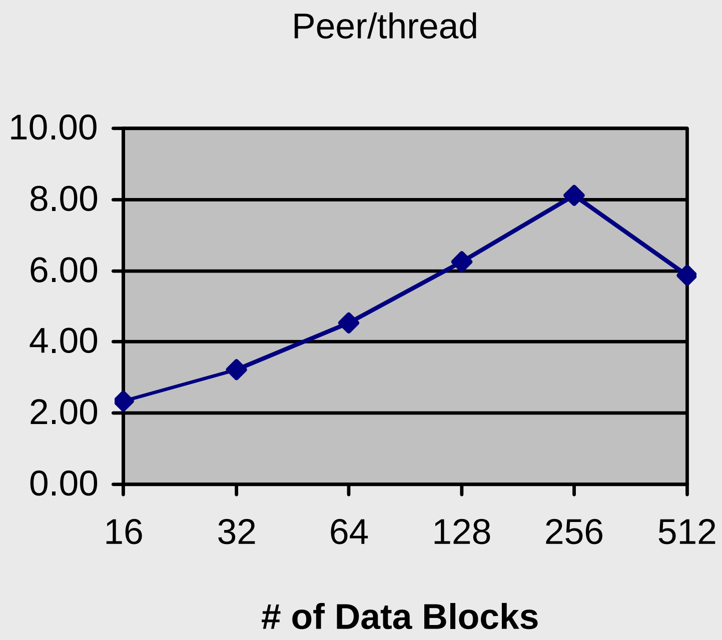
 # Data Blocks = 512

For one thread per block, grid size = # of Peers, blocksize = 512

For one thread per peer, grid size = # of Peers/512, blocksize = 512

Evaluation - GRF Speedup





 # Peers = 16384

For data block/thread: grid size = 16384, block size = # of data blocks.
For peer/thread: grid size = # of peers/512, block size = 512.

Summary

- RUB:
  - without SMEM: up to 4x speedup
  - Shared-memory-based: up to 8x speedup
- GRF:
  - Each thread handles a peer: up to 8x speedup
  - Each thread handles a block: up to 21x speedup

Future Work

- RUB
  - Design RUB with “one thread per data block”
  - Difficulty: randomly select a thread among a bunch of parallel threads
- GRF
  - Handles more data blocks (>512)
  - Let each thread handle multiple data blocks of a sender.

Discussion

Thanks

Any questions?