Berkeley Winter School

Advanced Algorithmic Techniques for GPUs

Lecture 8: Privatization and Further Studies

Objective

- To understand the nature of privatization of highly contended output data structures
  - When the output cannot be statically determined
  - Queues, histograms, etc.

- To learn efficient queue structures that support massively parallel extraction of input data from bulk data structures

Dynamic Data Extraction

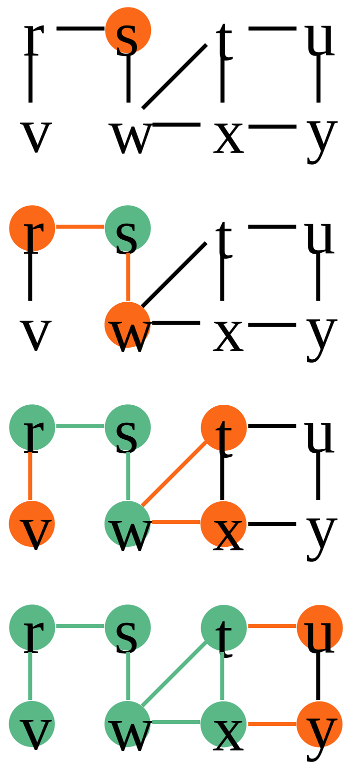
- The data to be processed in each phase of computation need to be dynamically determined and extracted from a bulk data structure
  - Harder when the bulk data structure is not organized for massively parallel access, such as graphs.
- Graph algorithms are popular examples that deal with dynamic data
  - Widely used in EDA and large scale optimization applications
  - We will use Breadth-First Search (BFS) as an example

Main Challenges of Dynamic Data

- Input data need to be organized for locality, coalescing, and contention avoidance as they are extracted during execution

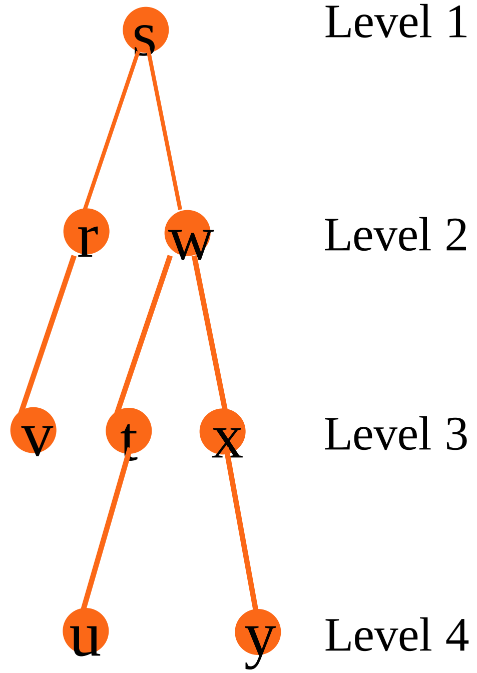
- The amount of work and level of parallelism often grow and shrink during execution
  - As more or less data is extracted during each phase
  - Hard to efficiently fit into one CUDA/OPenCL kernel configuration, which cannot be changed once launched
  - Different kernel strategies fit different data sizes

Breadth-First Search (BFS)



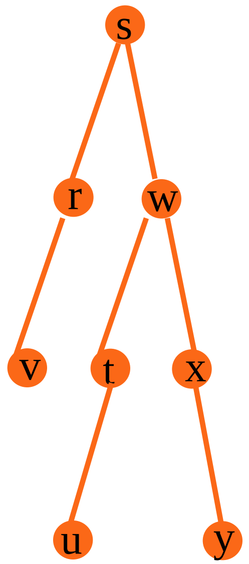
● Frontier vertex

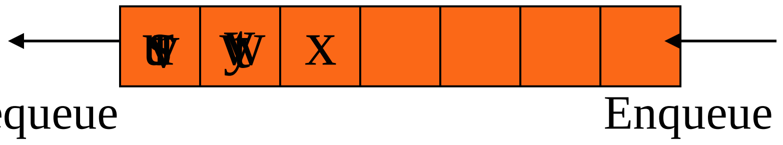
● Visited vertex



Sequential BFS

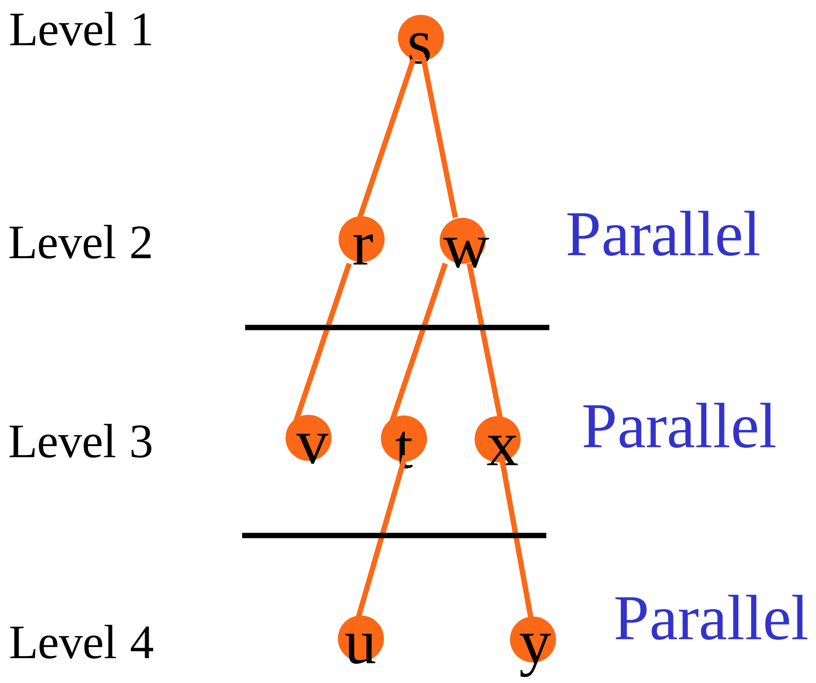
- Store the frontier in a queue
- Complexity (*O(V+E)*)

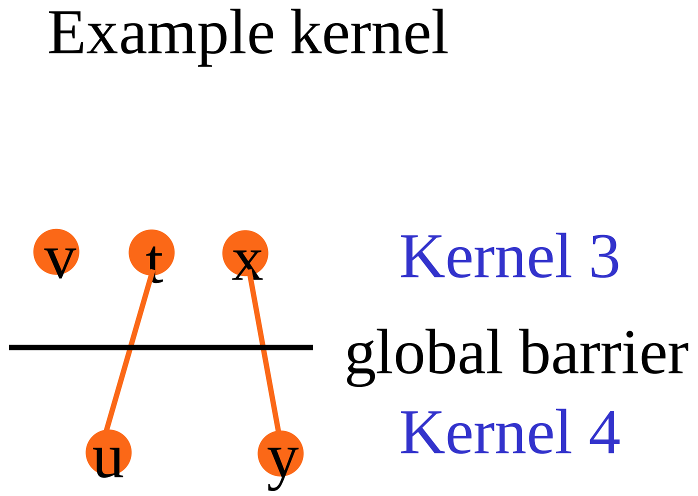




Parallelism in BFS

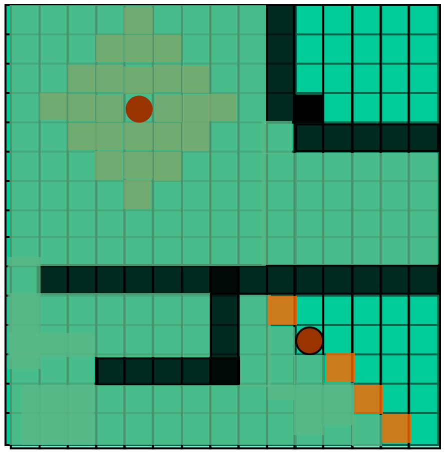
- Parallel Propagation in each level
- One GPU kernel per level





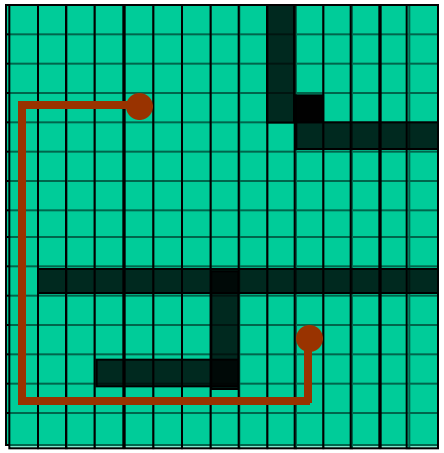
BFS in VLSI CAD

- Maze Routing



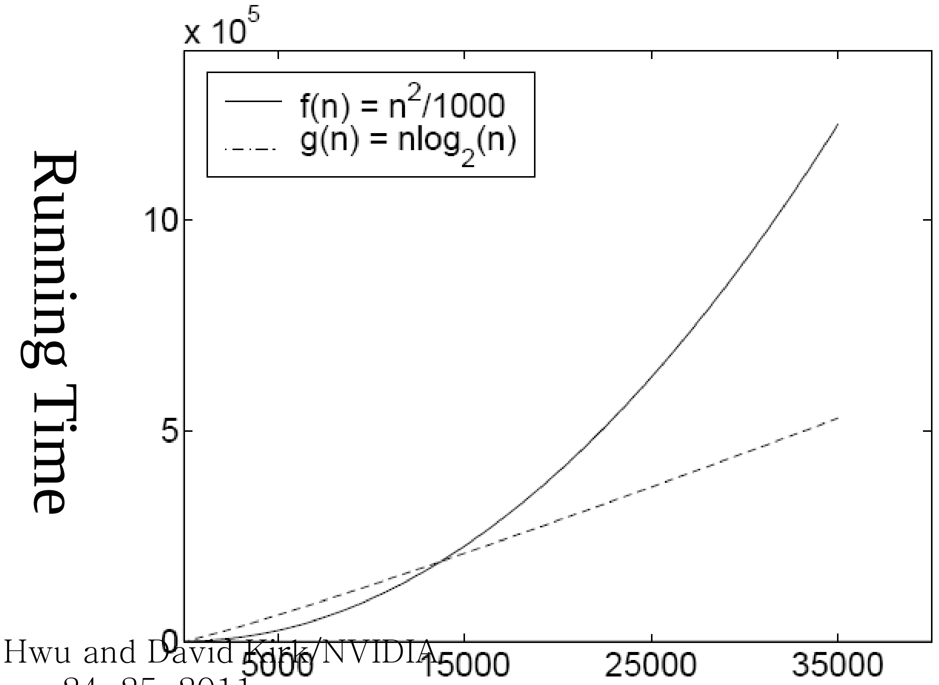
● net terminal

■ blockage



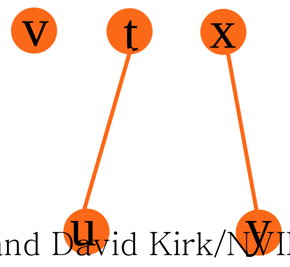
Potential Pitfall of Parallel Algorithms

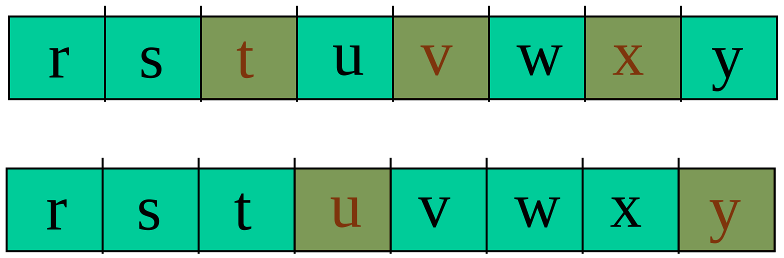
- Greatly accelerated n^2 algorithm is still slower than an $n\log n$ algorithm.
- Always need to keep an eye on fast sequential algorithm as the baseline.



Node Oriented Parallelization

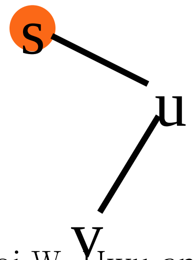
- IIIT-BFS
  - P. Harish et. al. “Accelerating large graph algorithms on the GPU using CUDA”
  - Each thread is dedicated to one node
  - Every thread examines neighbor nodes to determine if its node will be a frontier node in the next phase
  - Complexity *O(VL+E)* (Compared with *O(V+E)*)
  - Slower than the sequential version for large graphs
    - Especially for sparsely connect graphs



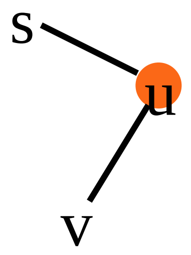


Matrix-based Parallelization

- Yangdong Deng et. al. “Taming Irregular EDA applications on GPUs”
- Propagation is done through matrix-vector multiplication
  - For sparsely connected graphs, the connectivity matrix will be a sparse matrix
- Complexity O(V+EL) (compared with O(V+E))
  - Slower than sequential for large graphs



$$\begin{array}{c|ccc|}
 & & & \\
s & 0 & 1 & 0 \\
u & 1 & 0 & 1 \\
v & 0 & 1 & 0 \\
 & s & u & v
\end{array}
\times
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
=
\begin{array}{|c|c}
0 & s \\
1 & u \\
0 & v
\end{array}$$



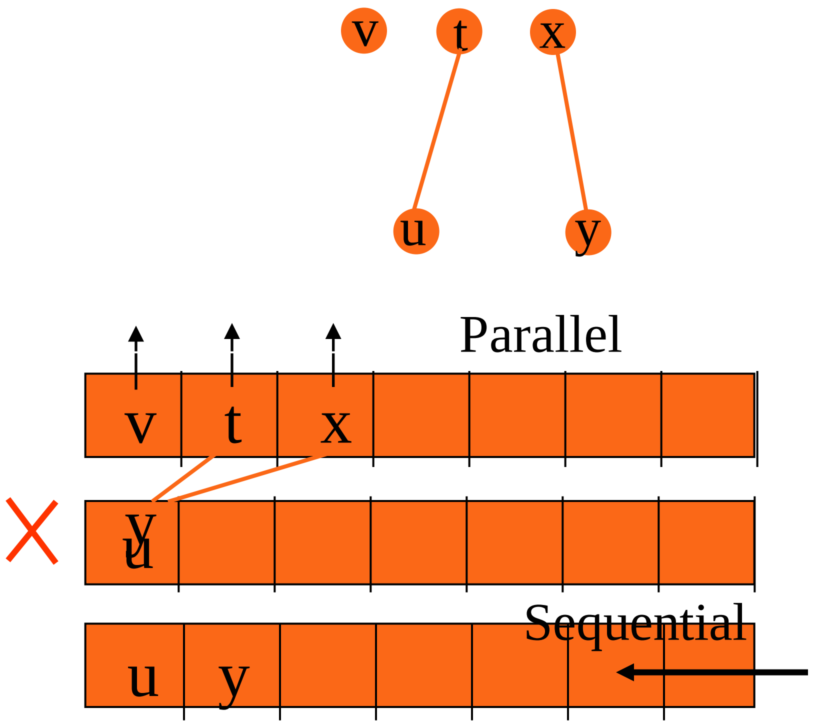
Need a More General Technique

- To efficiently handle most graph types
- Use more specialized formulation when appropriate as an optimization

- Efficient queue-based parallel algorithms
  – Hierarchical scalable queue implementation
  – Hierarchical kernel arrangements

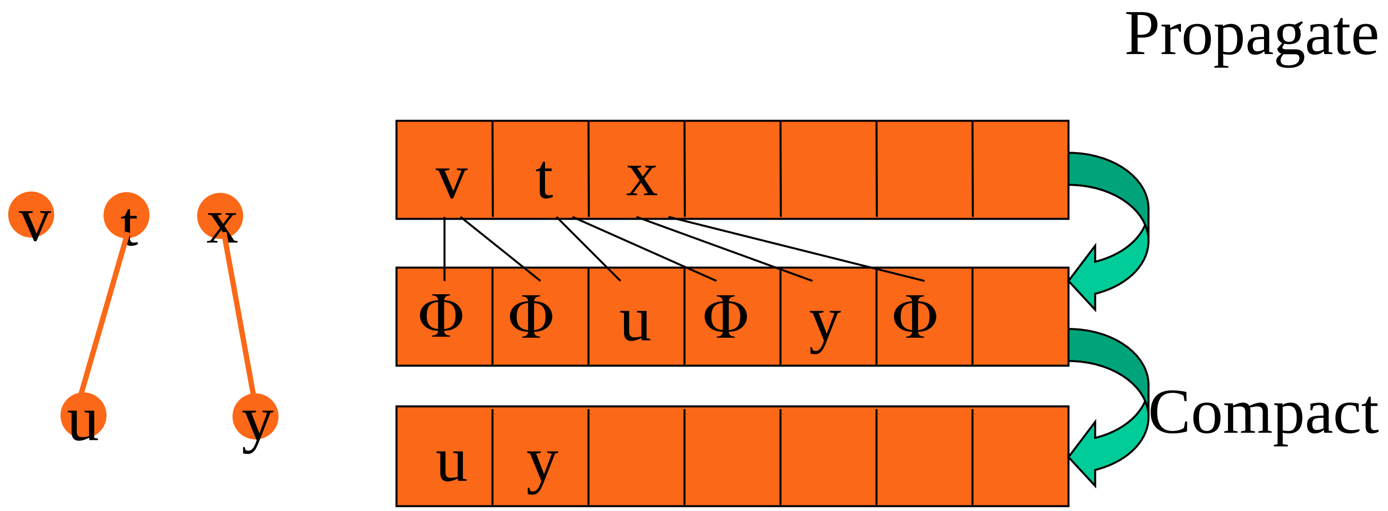
An Initial Attempt

- Manage the queue structure
  - Complexity: *O(V+E)*
  - Dequeue in parallel
  - Each frontier node is a thread
  - Enqueue in sequence.
    - Poor coalescing
    - Poor scalability
  - No speedup



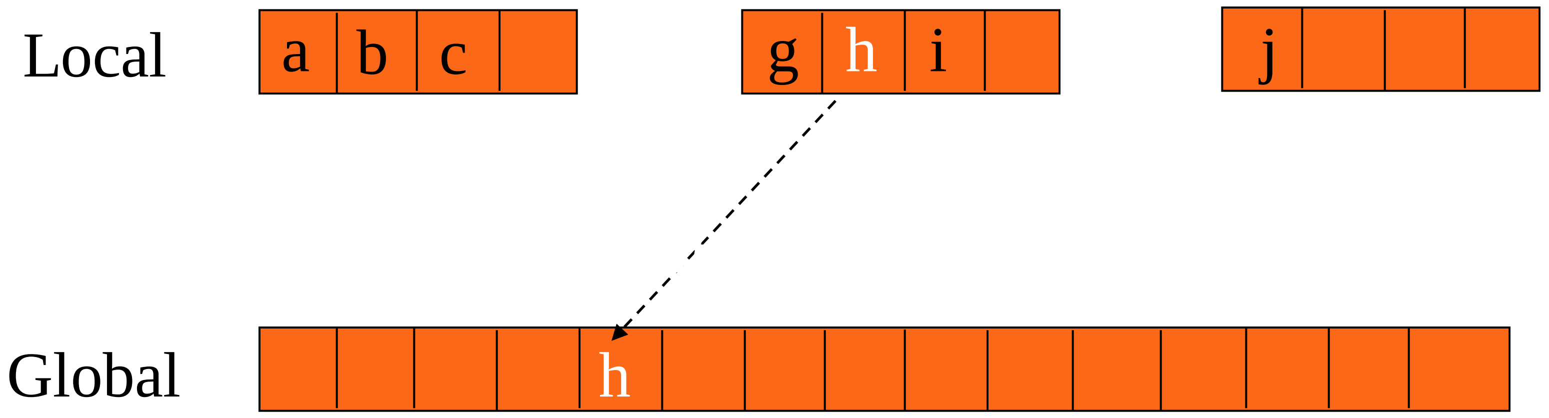
Parallel Insert-Compact Queues

- C.Lauterbach et.al."Fast BVH Construction on GPUs"
- Parallel enqueue with compaction cost
- Not suitable for light-node problems

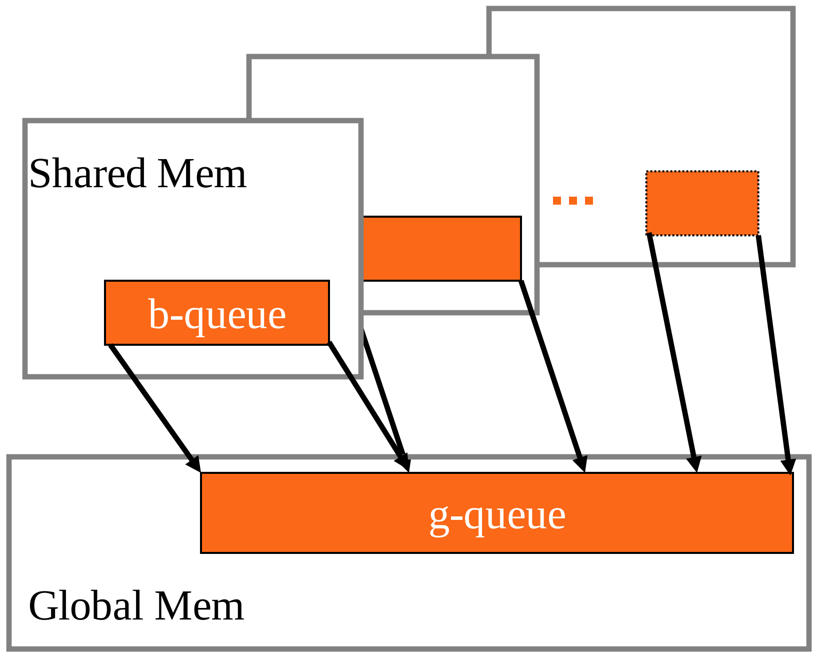


Basic ideas

- Each thread processes one or more frontier nodes
- Find the index of each new frontier node
- Build queue of next frontier hierarchically



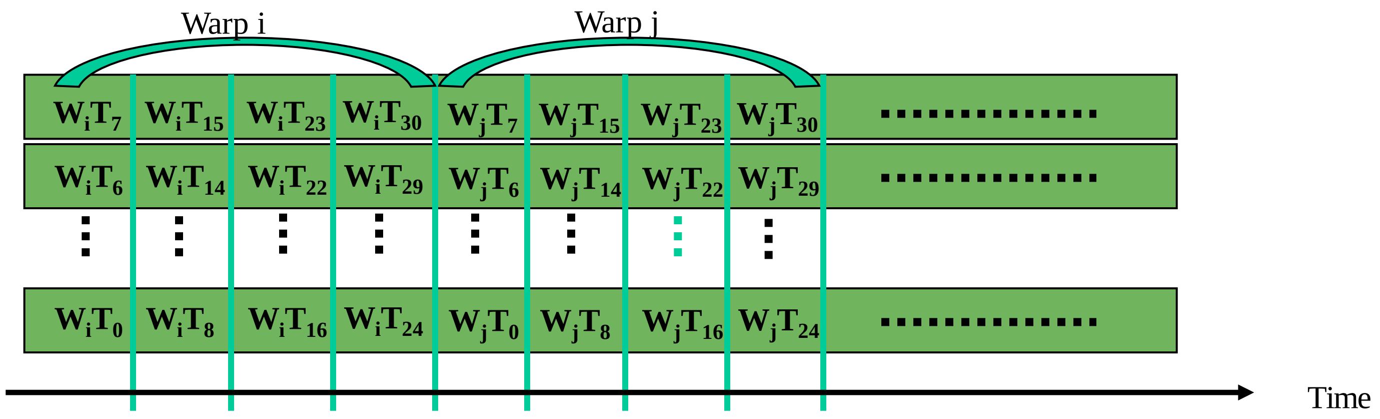
Two-level Hierarchy



- Block queue (b-queue)
  - Inserted by all threads in a block
  - Reside in Shared Memory
- Global queue (g-queue)
  - Inserted only when a block completes
- Problem:
  - Collision on b-queues
  - Threads in the same block can cause heavy contention

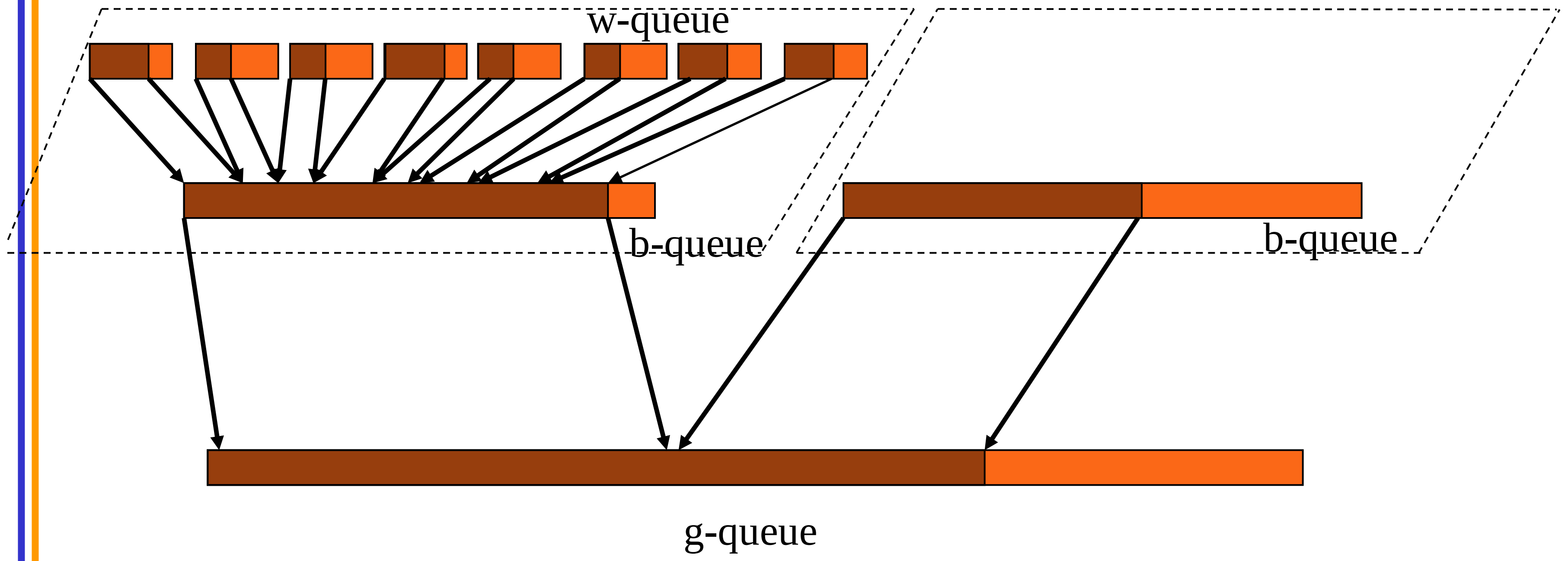
Warp-level Queue

- Thread Scheduling


Warp i
Warp j
W_iT_7 W_iT_{15} W_iT_{23} W_iT_{30} W_jT_7 W_jT_{15} W_jT_{23} W_jT_{30} ....
W_iT_6 W_iT_{14} W_iT_{22} W_iT_{29} W_jT_6 W_jT_{14} W_jT_{22} W_jT_{29} ....
W_iT_0 W_iT_8 W_iT_{16} W_iT_{24} W_jT_0 W_jT_8 W_jT_{16} W_jT_{24} ....
Time

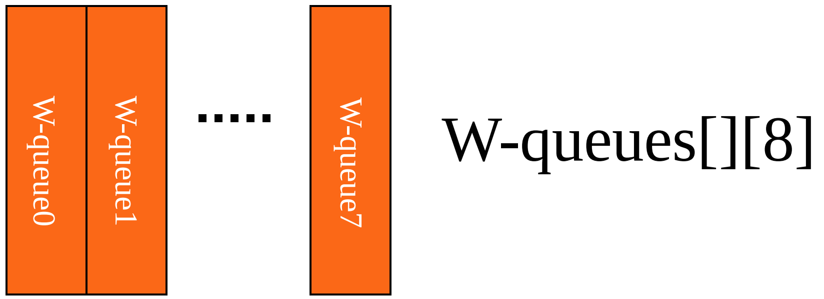
- Divide threads into 8 groups (for GTX280)
  - Number of SP's in each SM in general
  - One queue to be written by each SP in the SM
- Each group writes to one warp-level queue
- Still should use atomic operation
  - But much lower level of contention

Three-level hierarchy



Hierarchical Queue Management

- Shared Memory:
  - Interleaved queue layout, no bank conflict

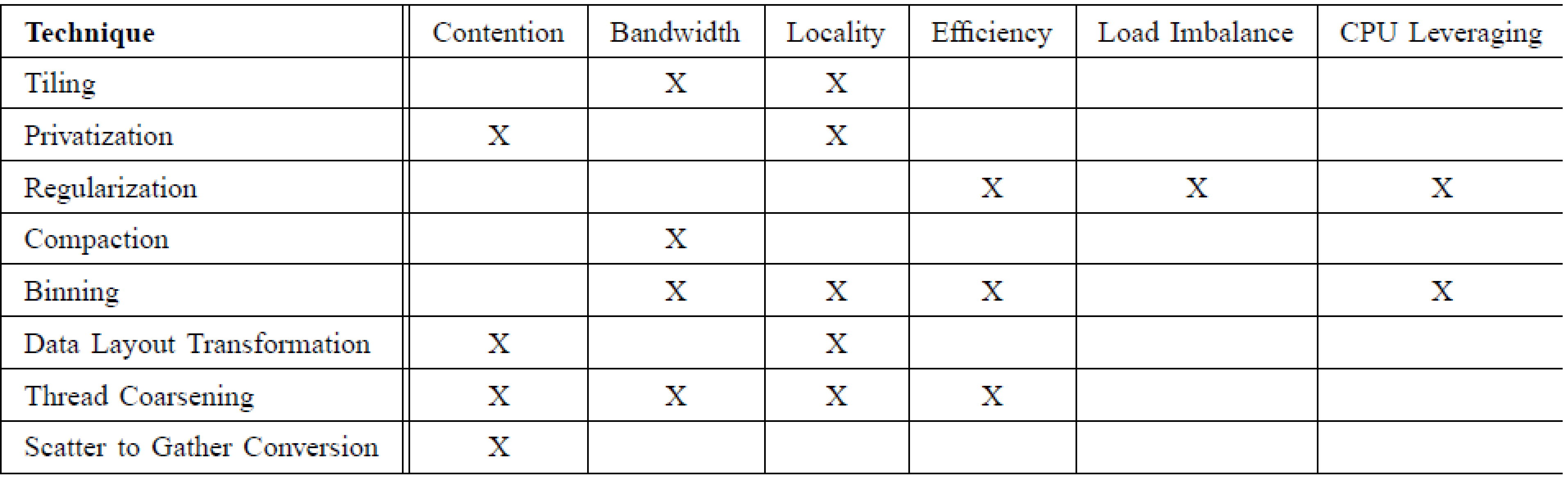


- Global Memory:
  - Coalescing when releasing a b-queue to g-queue
  - Moderate contention across blocks
- Texture memory :
  - Store graph structure (random access, no coalescing)
  - Fermi cache may help.

Hierarchical Queue Management

- Advantage and limitation
  - The technique can be applied to any inherently sequential data structure
  - The w-queues and b-queues are limited by the capacity of shared memory. If we know the upper limit of the degree, we can adjust the number of threads per block accordingly.

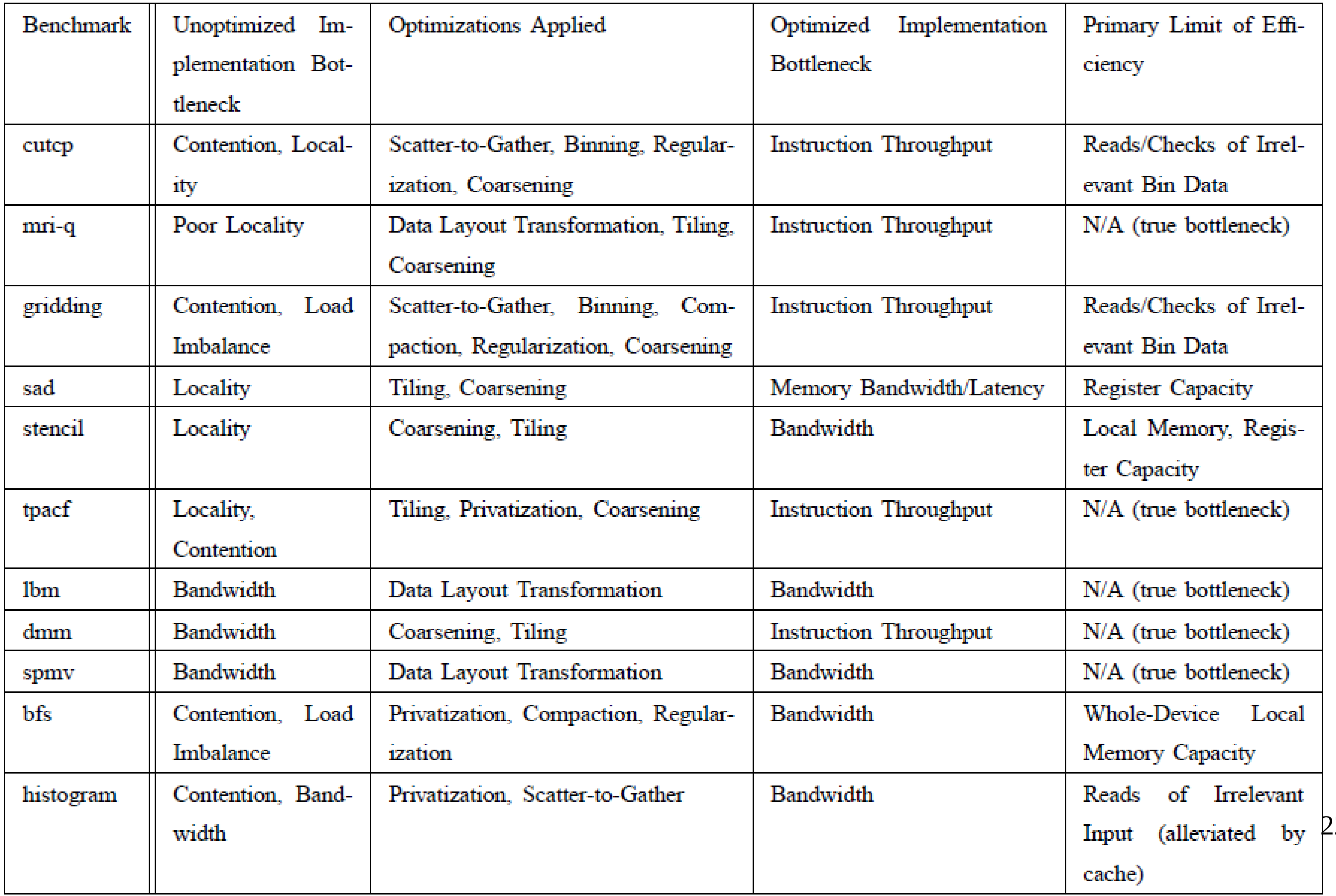
Eight Optimization Patterns for Algorithms (so far)



Technique	Contention	Bandwidth	Locality	Efficiency	Load Imbalance	CPU Leveraging
Tiling		X	X			
Privatization	X		X			
Regularization				X	X	X
Compaction		X				
Binning		X	X	X		X
Data Layout Transformation	X		X			
Thread Coarsening	X	X	X	X		
Scatter to Gather Conversion	X					

http://courses.engr.illinois.edu/ece598/hk/

Impact of Techniques on Apps



Benchmark	Unoptimized Implementation Bottleneck	Optimizations Applied	Optimized Implementation Bottleneck	Primary Limit of Efficiency
cutcp	Contention, Locality	Scatter-to-Gather, Binning, Regularization, Coarsening	Instruction Throughput	Reads/Checks of Irrelevant Bin Data
mri-q	Poor Locality	Data Layout Transformation, Tiling, Coarsening	Instruction Throughput	N/A (true bottleneck)
gridding	Contention, Load Imbalance	Scatter-to-Gather, Binning, Compaction, Regularization, Coarsening	Instruction Throughput	Reads/Checks of Irrelevant Bin Data
sad	Locality	Tiling, Coarsening	Memory Bandwidth/Latency	Register Capacity
stencil	Locality	Coarsening, Tiling	Bandwidth	Local Memory, Register Capacity
tpacf	Locality, Contention	Tiling, Privatization, Coarsening	Instruction Throughput	N/A (true bottleneck)
lbm	Bandwidth	Data Layout Transformation	Bandwidth	N/A (true bottleneck)
dmm	Bandwidth	Coarsening, Tiling	Instruction Throughput	N/A (true bottleneck)
spmv	Bandwidth	Data Layout Transformation	Bandwidth	N/A (true bottleneck)
bfs	Contention, Load Imbalance	Privatization, Compaction, Regularization	Bandwidth	Whole-Device Local Memory Capacity
histogram	Contention, Bandwidth	Privatization, Scatter-to-Gather	Bandwidth	Reads of Irrelevant Input (alleviated by cache)

Challenges of Parallel Programming

- Computations with no known scalable parallel algorithms
  - Shortest path, Delaunay triangulation, …
- Data distributions that cause catastrophical load imbalance in parallel algorithms
  - Free-form graphs, MRI spiral scan
- Computations that do not have data reuse
  - Matrix vector multiplication, …
- Algorithm optimizations that are require expertise
  - Locality and regularization transformations

Benefit from other people's experience

- GPU Computing Gems Vol 1
  - Coming January 2011
  - 50 gems in 10 applications areas
  - Scientific simulation, life sciences, statistical modeling, emerging data-intensive applications, electronic design automation, computer vision, ray tracing and rendering, video and imaging processing, signal and audio processing, medical imaging
- GPU Computing Gems Vol 2
  - Coming in May 2011
  - 50+ gems in more application areas, tools, environments

THANK YOU!