VSCSE Summer School

Proven Algorithmic Techniques for Many-core Processors

Lecture 8: Dealing with Dynamic Data

Dynamic Data

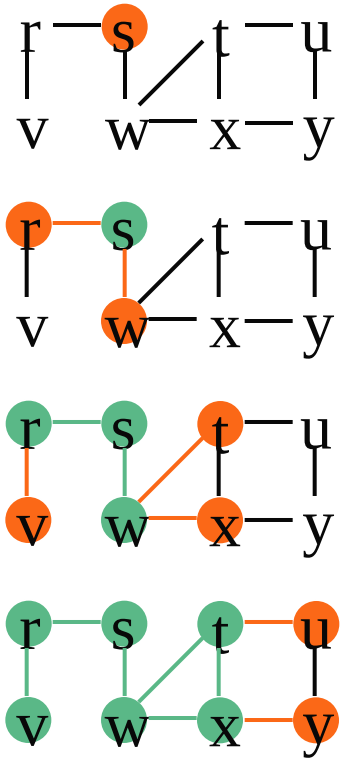
- The data to be processed in each phase of computation need to be dynamically determined and extracted from a bulk data structure
  - Harder when the bulk data structure is not organized for massively parallel access, such as graphs.
- Graph algorithms are popular examples that deal with dynamic data
  - 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

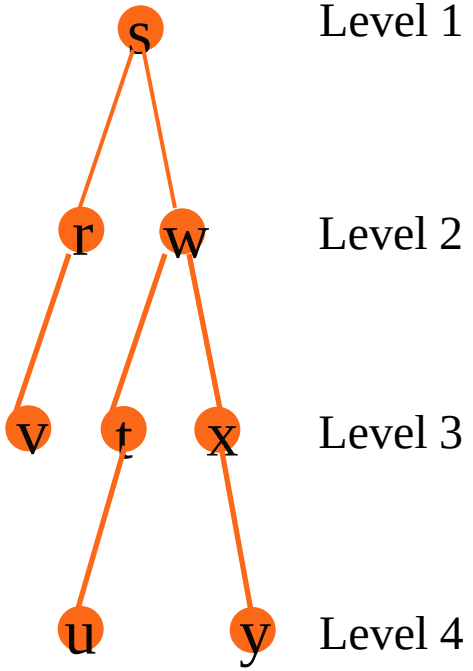
- The amount of work and level of parallelism often grow and shrink during execution
  - As more or less data is extracted during each phase
  - Hard to efficiently fit into one CUDA kernel configuration, which cannot be changed once launched
  - Different kernel strategies fit different data sizes

Breadth-First Search (BFS)



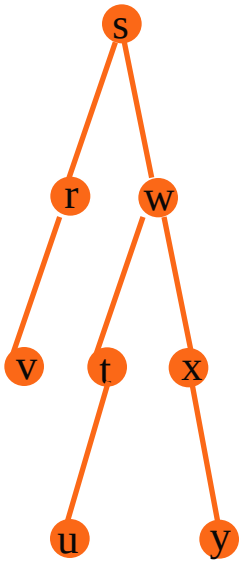
Frontier vertex

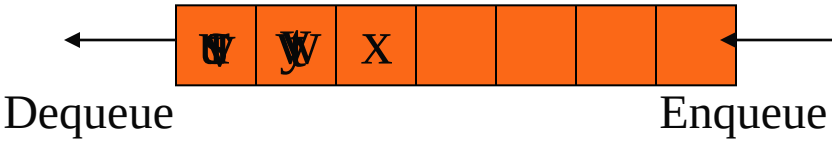
Visited vertex



Sequential BFS

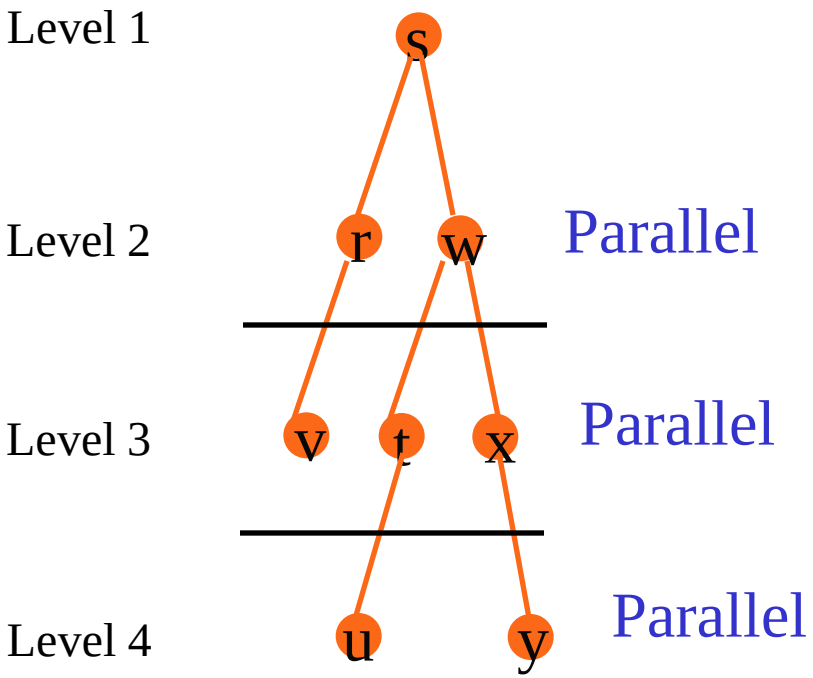
- Store the frontier in a queue
- Complexity (*O(V+E)*)

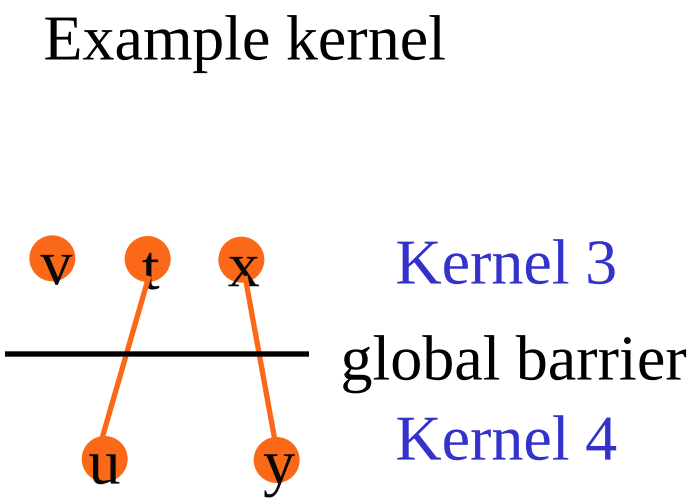




Parallelism in BFS

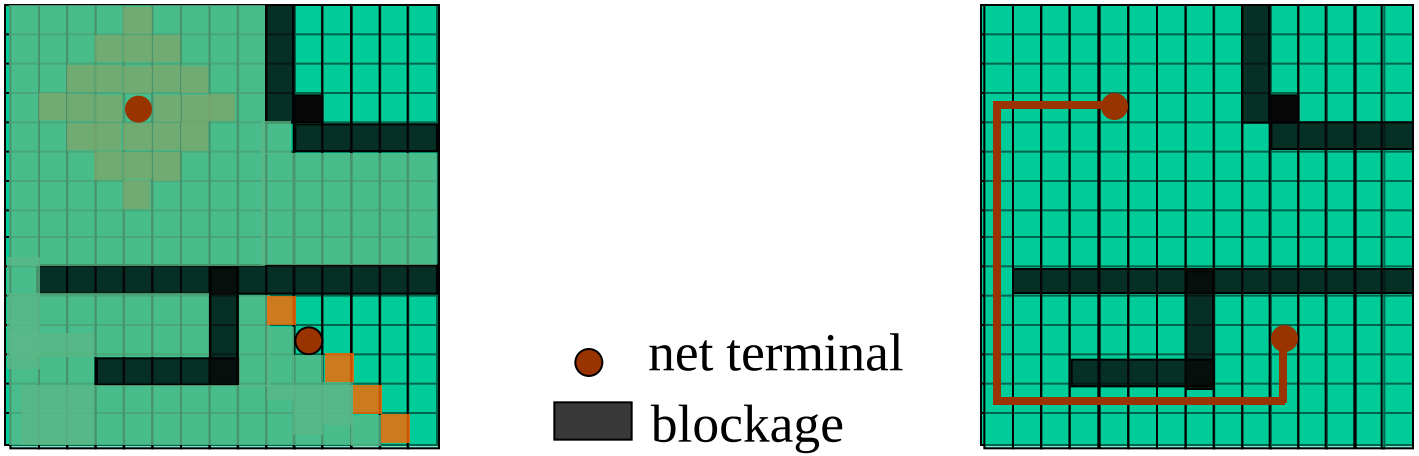
- Parallel Propagation in each level
- One GPU kernel per level





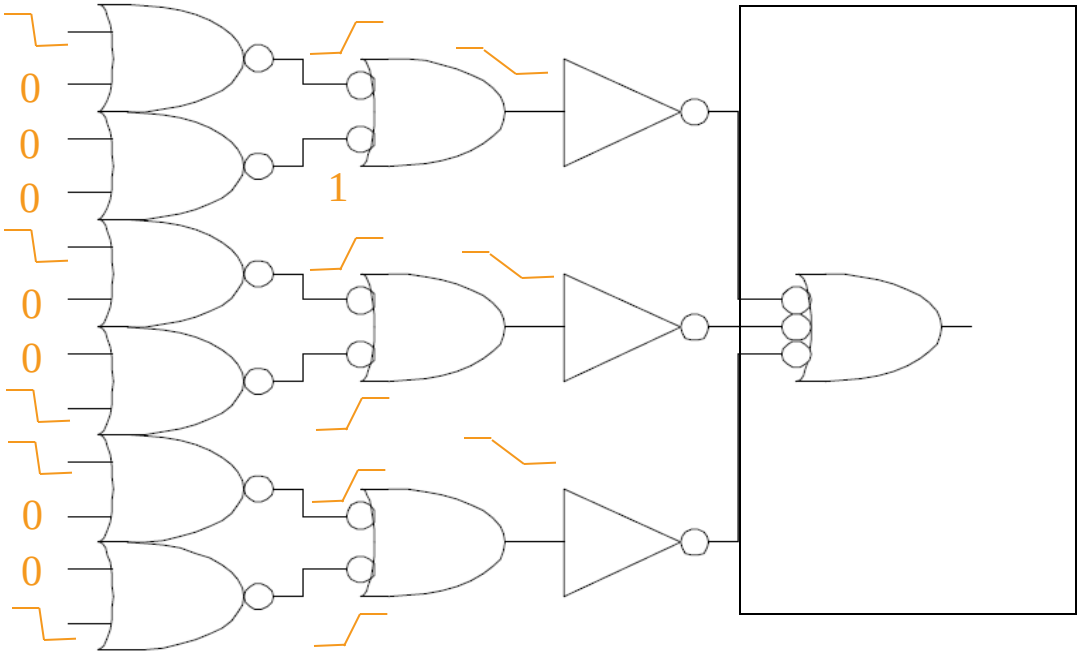
BFS in VLSI CAD

- Maze Routing



BFS in VLSI CAD

- Logic Simulation/Timing Analysis

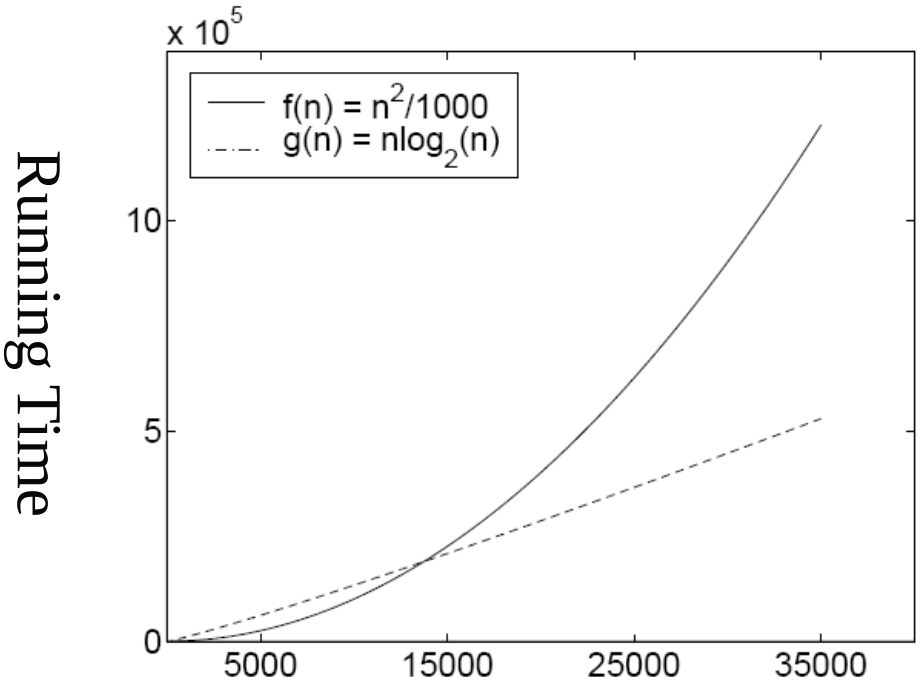


BFS in VLSI CAD

- In formal verification for reachabiliy analysis.
- In clustering for finding connected components.
- In logic synthesis
- ……..

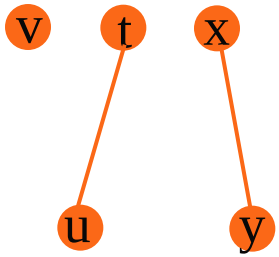
Potential Pitfall of Parallel Algorithms

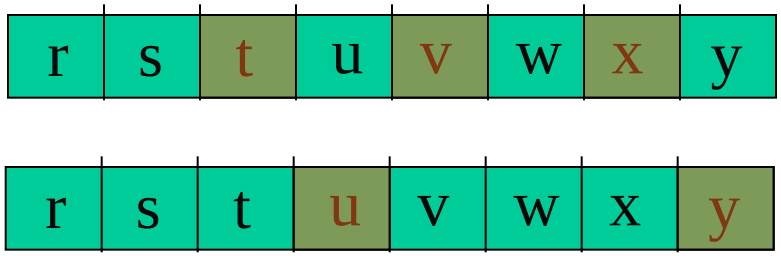
- Greatly accelerated n^2 algorithm is still slower than an $n\log n$ algorithm.
- Always need to keep an eye on fast sequential algorithm as the baseline.



Node Oriented Parallelization

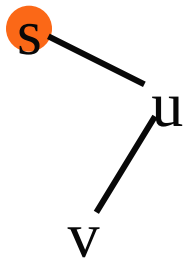
- IIIT-BFS
  - P. Harish et. al. “Accelerating large graph algorithms on the GPU using CUDA”
  - Each thread is dedicated to one node
  - Every thread examines neighbor nodes to determine if its node will be a frontier node in the next phase
  - Complexity *O(VL+E)* (Compared with *O(V+E)*)
  - Slower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





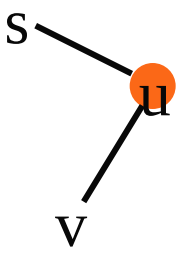
Matrix-based Parallelization

- 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



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

$$\begin{array}{ccc} s & u & v \end{array}$$



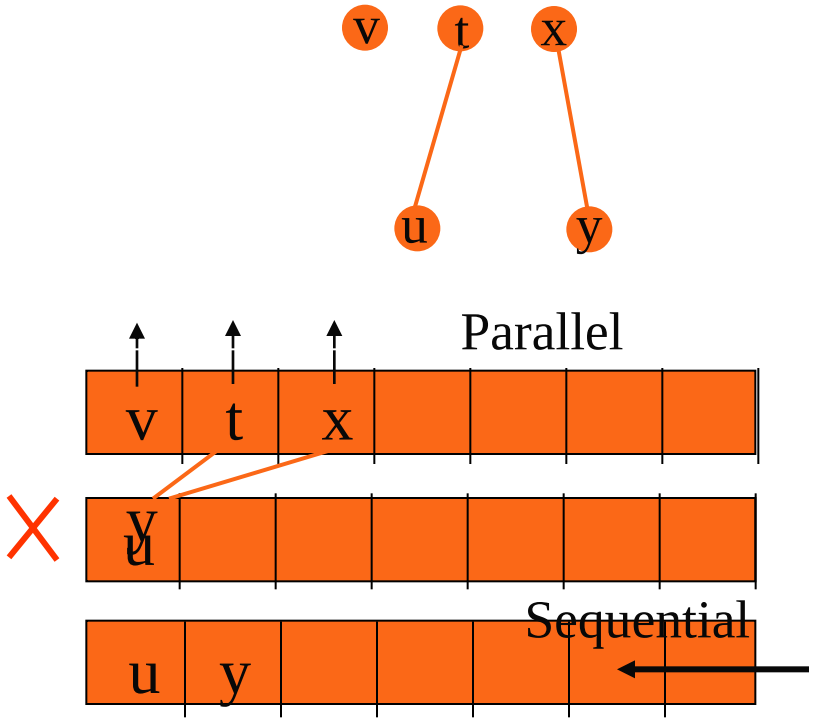
Need a More General Technique

- 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

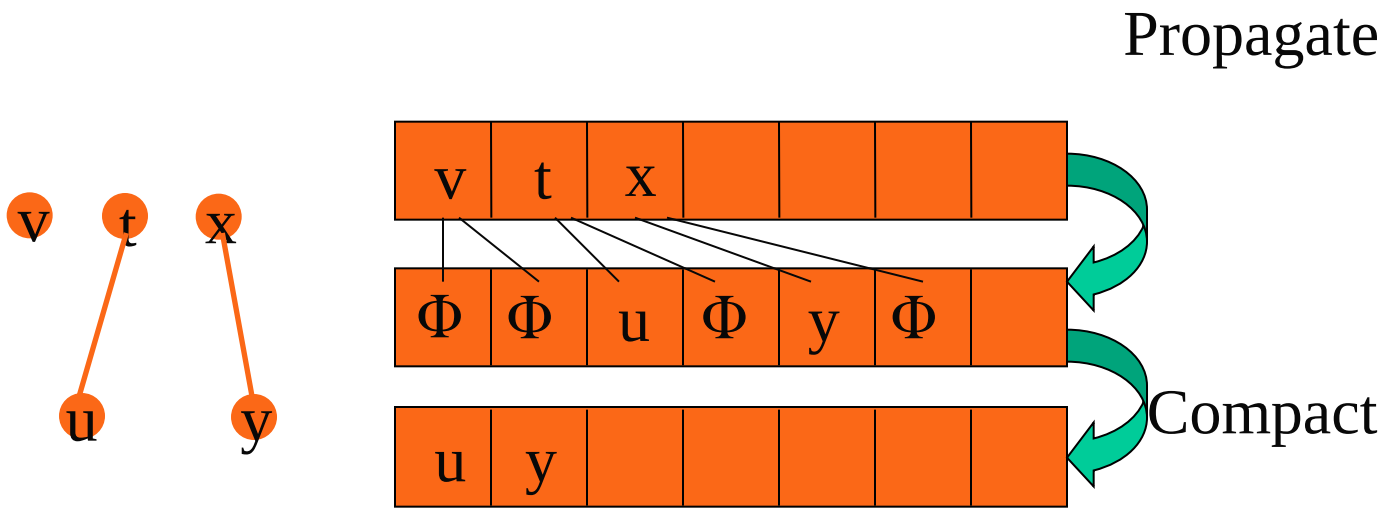
An Initial Attempt

- 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



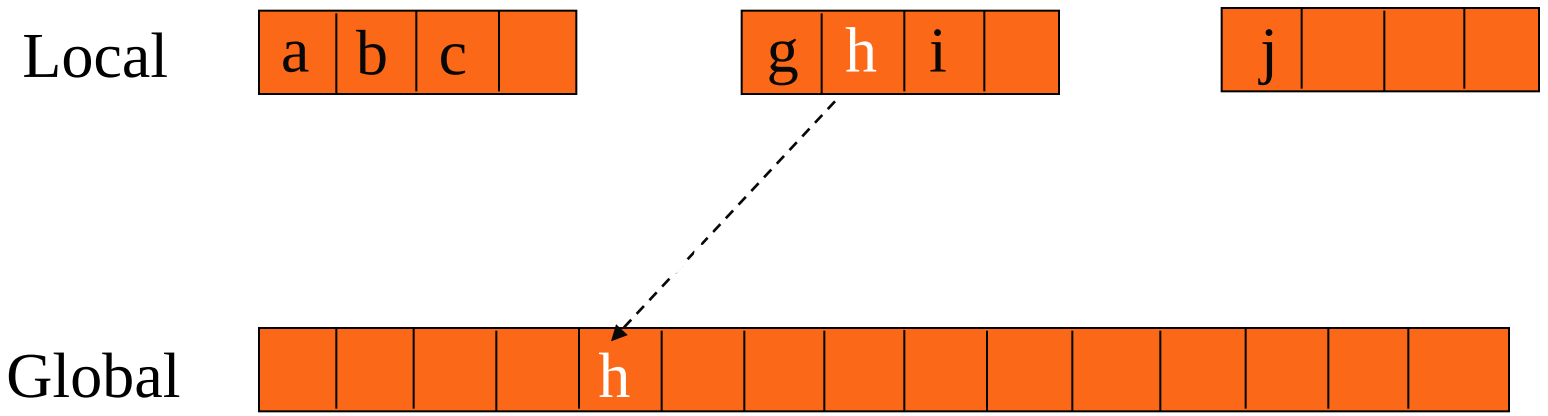
Parallel Insert-Compact Queues

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

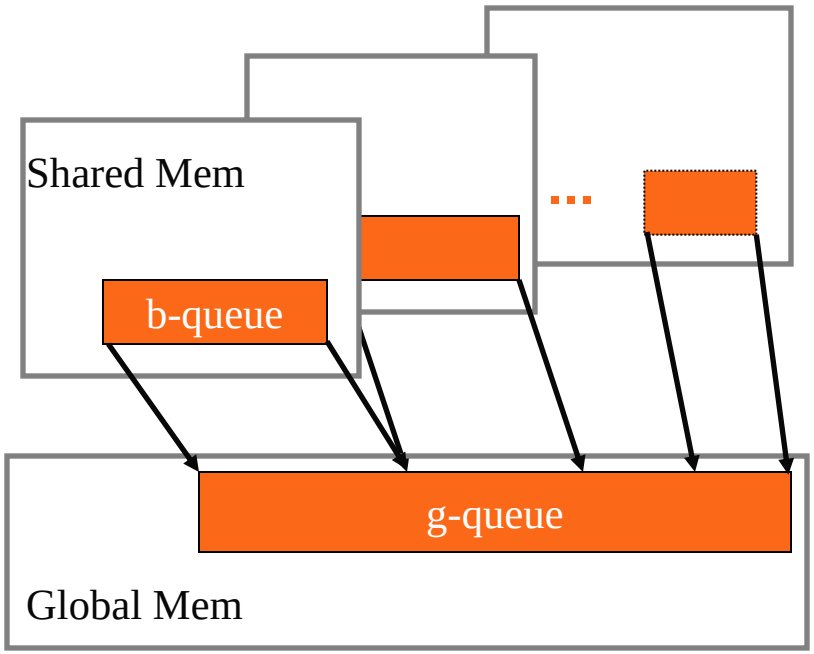


Basic ideas

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



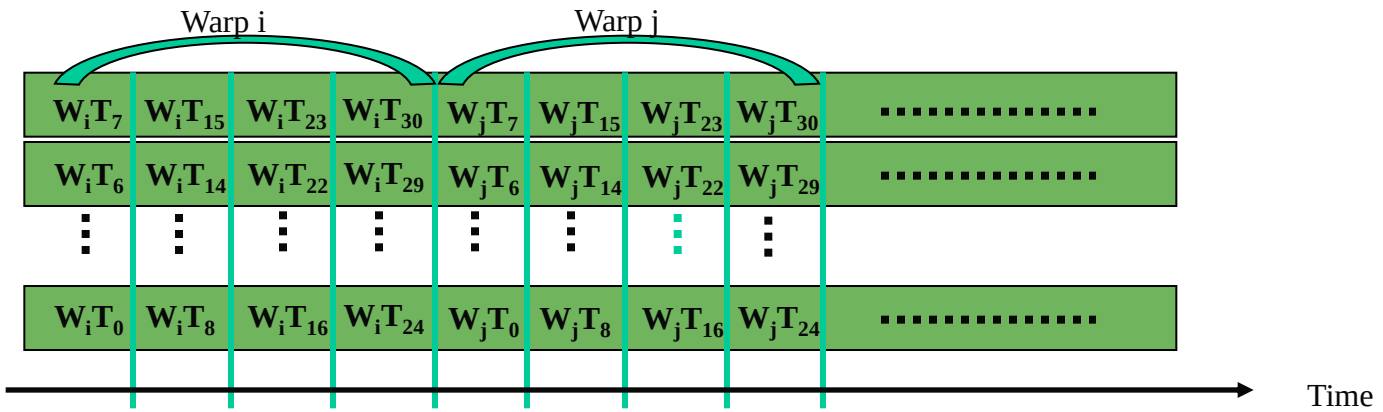
Two-level Hierarchy



- 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

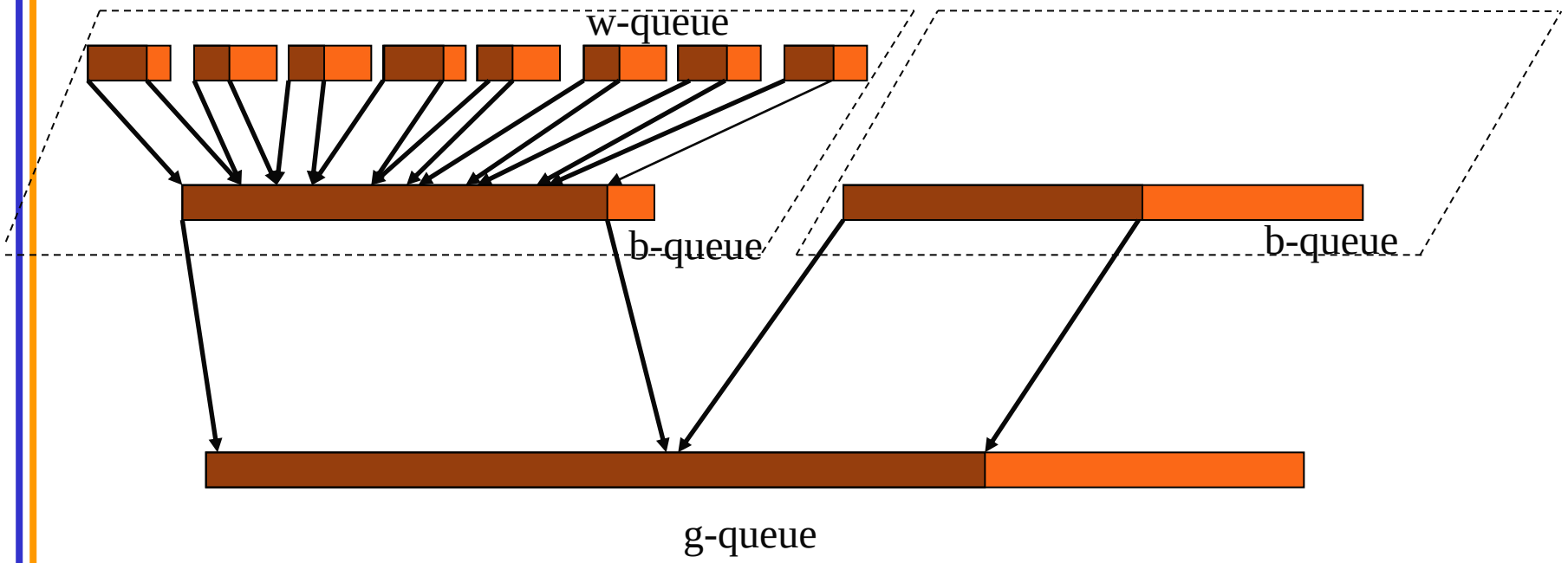
Warp-level Queue

- Thread Scheduling



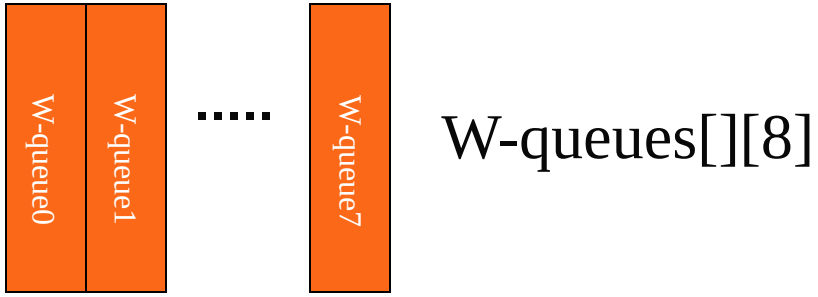
- Divide threads into 8 groups (for GTX280)
  - Number of SP's in each SM in general
  - One queue to be written by each SP in the SM
- Each group writes to one warp-level queue
- Still should use atomic operation
  - But much lower level of contention

Three-level hierarchy



Hierarchical Queue Management

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

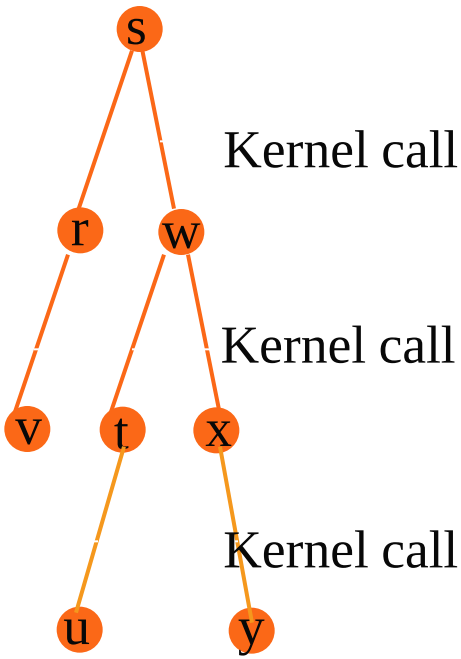


- Global Memory:
  - Coalescing when releasing a b-queue to g-queue
  - Moderate contention across blocks
- Texture memory :
  - Store graph structure (random access, no coalescing)
  - Fermi cache may help.

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.

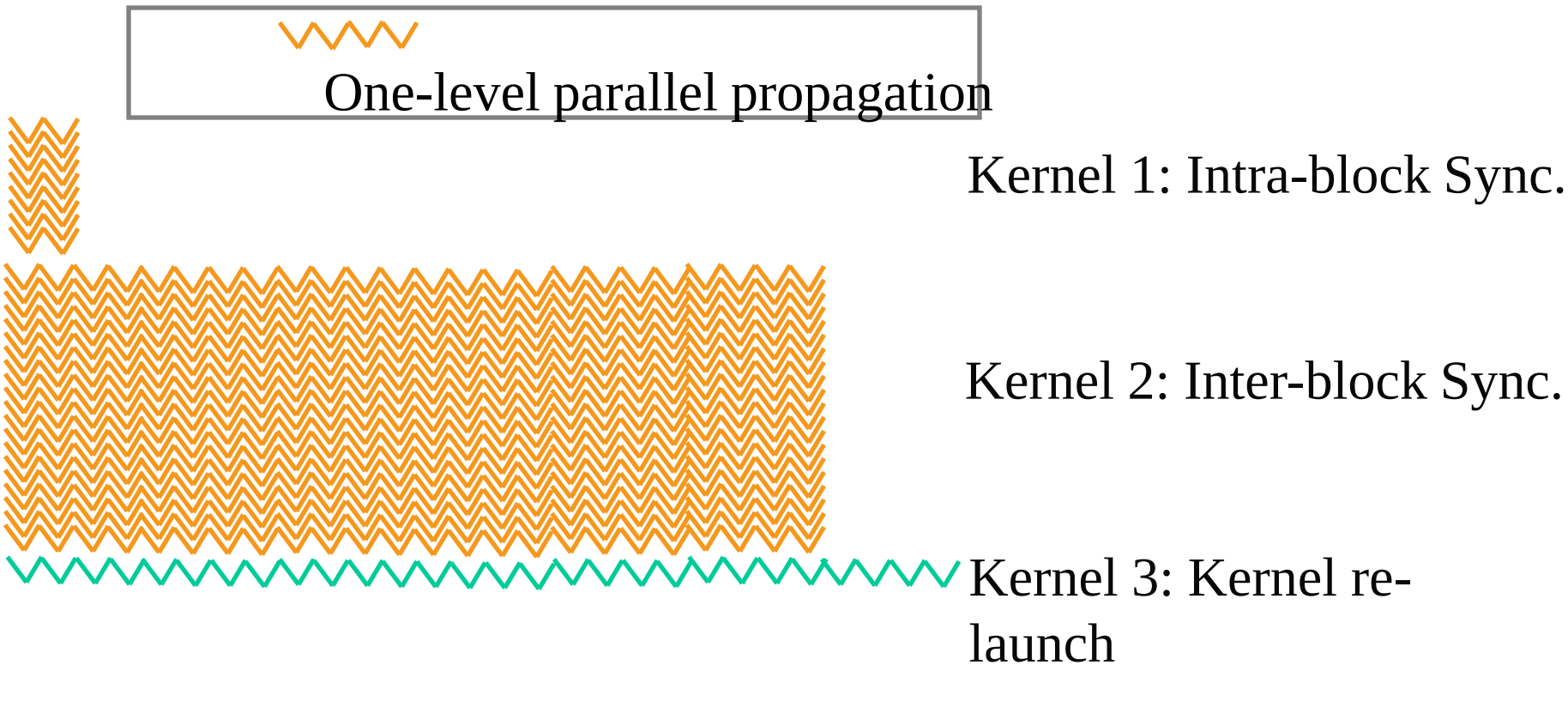
Kernel Arrangement



- To create global barriers needs frequent kernel launches
- Too much overhead
- Solution:
  - Partially use GPU-synchronization
  - Three-layer Kernel Arrangement

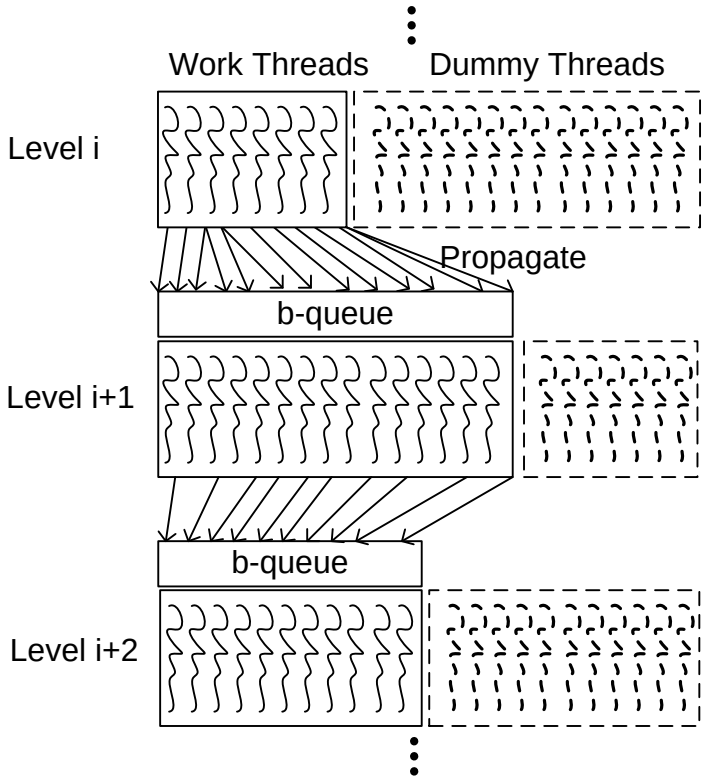
Hierarchical Kernel Arrangement

- Customize kernels based on the size of frontiers.
- Use GPU synchronization when the frontier is small.



Kernel Arrangement

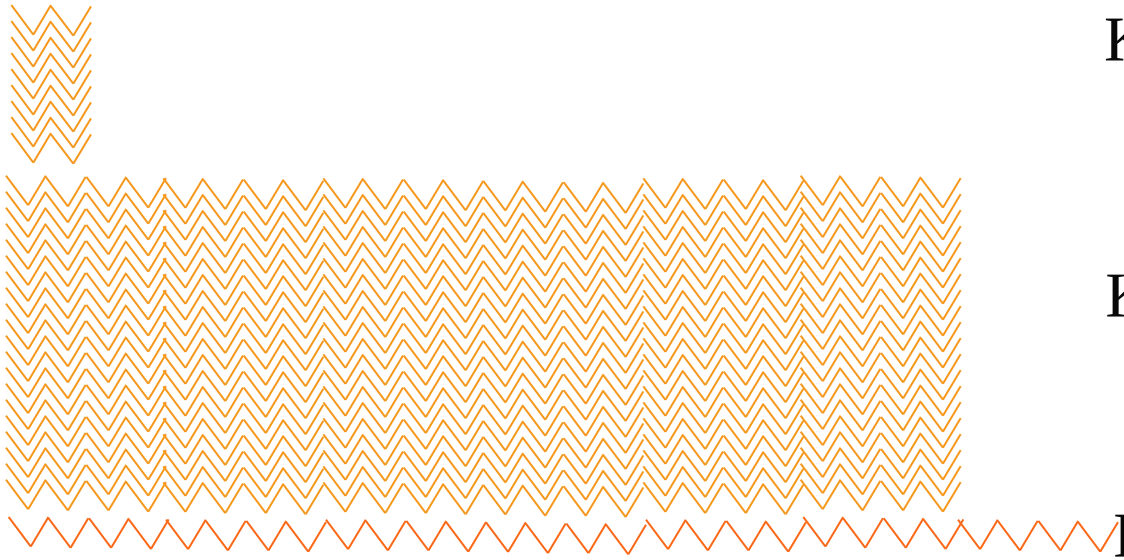
- Kernel 1: small-sized frontiers
  - Only launch one block
  - Use CUDA barrier function
  - Propagate through multiple levels
  - Save global memory access



Kernel Arrangement

- Kernel 2: mid-sized frontiers
  - Launch m blocks (m = #SM)
  - Each block is assigned to one SM and stays active
  - Use global memory to implement inter-block synchronization
  - Global synch across blocks is allowed in CUDA when all there is only one block per SM
  - Propagate through multiple levels
- Kernel 3: big-sized frontiers
  - Use kernel re-launch to implement synchronization
  - The kernel launch overhead is acceptable considering the time to propagate a huge frontier

Kernel Arrangement for GTX280



Kernel 1: Intra-block Sync.
≤ 512 nodes

Kernel 2: Inter-block Sync.
≤ 15360 (30*512)

Kernel 3: kernel termination Sync.
> 15360

Assumption: #SM = 30
#Thread/block = 512

Experimental setup

- CPU implementation
  - The classical BFS algorithm (O(V+E))
  - dual socket dual core 2.4 Ghz Opteron processor with 8GB memory.
- GPU: NVIDIA GeForce GTX280
- Benchmarks
  - Near-regular graphs (degree = 6)
  - Real world graphs (avg. degree = 2, max degree = 9)
  - Scale free graphs
    - 0.1% of the vertices: degree = 1000
    - The remaining vertices: degree = 6

Results on near-regular graphs

#Vertex (M)	IIIT-BFS (ms)	CPU-BFS (ms)	UIUC-BFS (ms)	Speedup (CPU/UIUC)
1	462.8	146.7	67.8	2.2
2	1129.2	311.8	121.0	2.6
5	4092.2	1402.2	266.0	5.3
7	6597.5	2831.4	509.5	5.6
9	9170.1	4388.3	449.3	9.8
10	11019.8	5023.0	488.0	10.3

Results on real-world graphs

#Vertex	IIIT-BFS (ms)	CPU-BFS (ms)	UIUC-BFS (ms)	Speedup (CPU/UIUC)
264,346	79.9	41.6	19.4	2.1
1,070,376	372.0	120.7	61.7	2.0
3,598,623	1471.1	581.4	158.5	3.7
6,262,104	2579.4	1323.0	236.6	5.6

Results on scale-free graphs

#Vertex (M)	IIIT-BFS (ms)	CPU-BFS (ms)	UIUC-BFS (ms)
1	161.5	52.8	100.7
5	1015.4	284.0	302.0
10	2252.8	506.9	483.6

- Parallelism on the imbalanced problems does not work as well as of today.

Concluding Remarks

- Effectively accelerated BFS, considering the upper limit of such memory-bound applications
  - Still need to address non-uniform data distribution
- Hierarchical queue management and multi-layer kernel arrangement are potentially applicable to other types of algorithms with dynamic data (work)
- To fully exploit the power of GPU computation, we need more efforts at the data structure and algorithmic level.

References

- Lijuan Luo, Martin Wong, Wen-mei Hwu, “An effective GPU implementation of breadth-first search”, accepted by Design Automation Conference, 2010.
- Pawan Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA", in IEEE High Performance Computing, 2007. LNCS 4873, pp 197-208.
- C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, “Fast BVH Construction on GPUs”, Computer Graphics Forum, Vol. 28, No. 2., pp. 375-384.
- Yangdong(Steve) Deng, Bo David Wang and Shuai Mu, "Taming Irregular EDA Applications on GPUs", ICCAD'09, page 539-546, 2009.
- Shucai Xiao and Wu-chun Feng, "Inter-Block GPU Communication via Fast Barrier Synchronization", Technical Report TR-09-19, Computer Science, Virginia Tech.

ANY FURTHER QUESTIONS?