Simpler and Faster HLBVH with Work Queues

Kirill Garanzha NVIDIA

Jacopo Pantaleoni NVIDIA Research

David McAllister NVIDIA

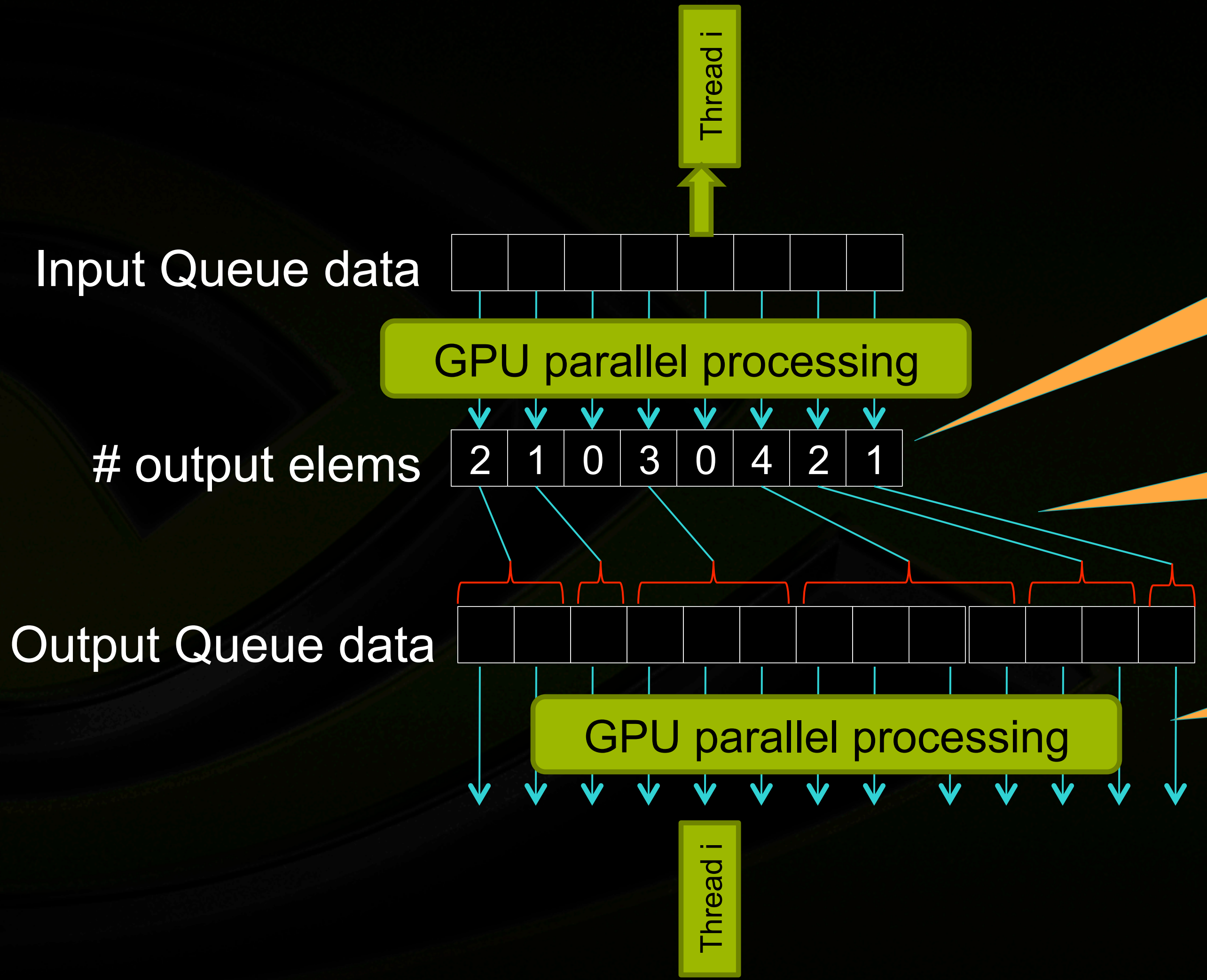
Short Summary

- Full GPU implementation
- Simple work queues generation
- Simple middle split hierarchy emission
- Efficient and straightforward top-level SAH tree emission
- Enabled by fast atomic instructions

3 improvements over HLBVH 2010

- Fast work queues
- Spatial-median BVH splits based on binary search
- Top-level SAH BVH build on GPU

What is work queue for GPU



Problem: Generates variable # of work items

Target: Generate the output queue with no holes

And get full GPU utilization

work queue generation

- Option 1: output item address = atomic increment on global counter for each element of input queue
  - Pros: no need to store temp results
  - Cons: many conflicting writes

work queue generation

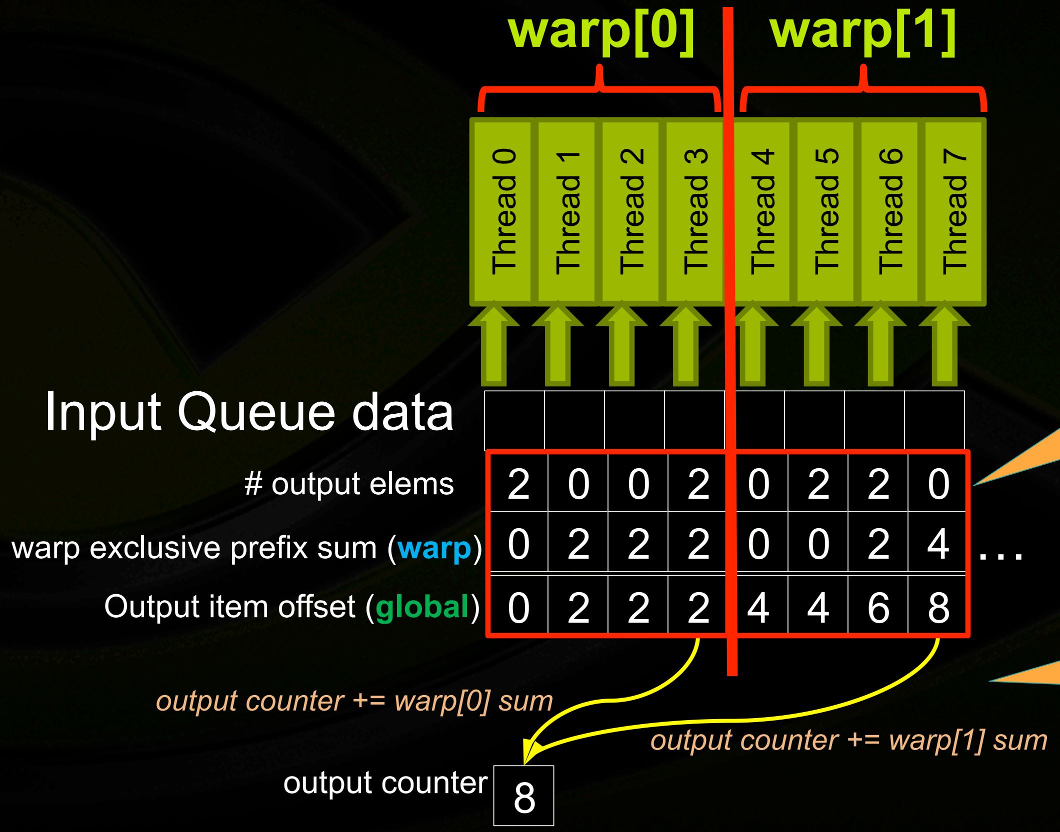
- Option 2: output item address = **prefix sum**
  - Pros: no conflicting writes
  - Cons: need to store results in temp memory and propagate them to output queue (global memory is limited on GPU)

work queue generation

- Option 3: hybrid approach
  (local prefix sum + atomic / warp)
  - Pros: unites the pros of both methods and remove their cons;
  - Pros: fastest!

work queue generation

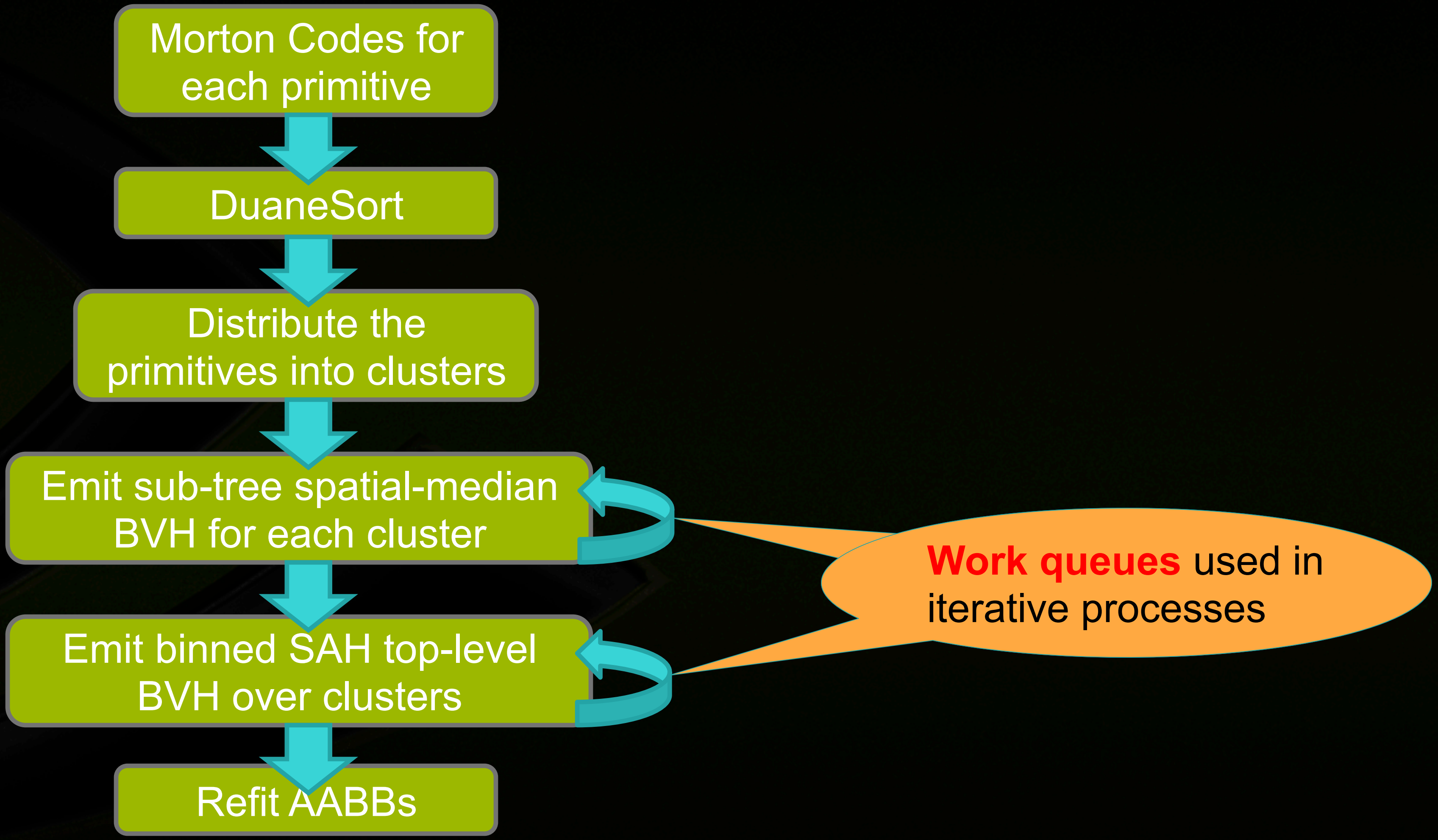
- Option 3: hybrid approach



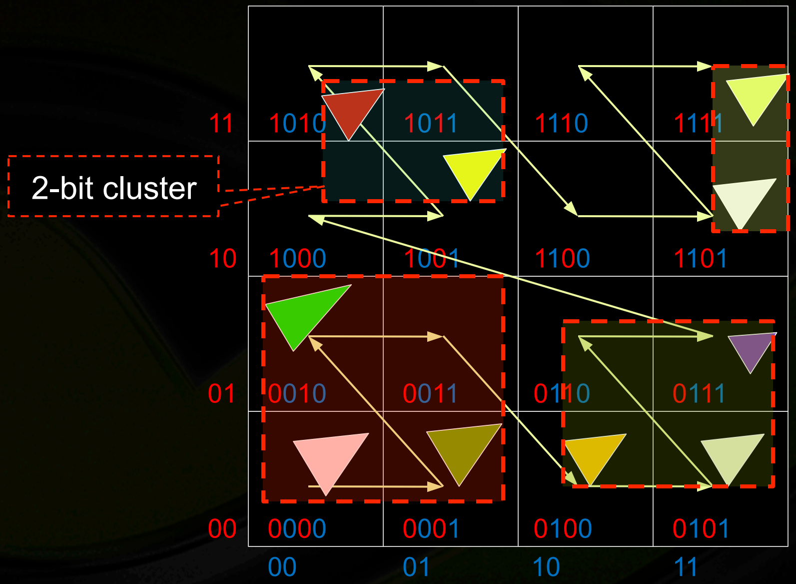
- Every prefix sum is computed in shared memory.
- No need to spend limited global memory.
- Temp results stored in each thread memory

-Atomic increments are fast on Fermi
- Reduce conflicting writes when used once per warp

WQ application: HLBVH pipeline



Distribute primitives into clusters



- Sample 4bit morton codes
- 2bit clusters are determined with 2 most significant bits of each prim morton code
- Compression from CSD is used to extract the segments of clusters
- 15bit clusters are good for high-quality BVH

Spatial median BVH emission

Each morton code expanded into bits shown as row per bit level

	0	1	2	3	4	5	6	7	8	9
mortoncode[bit3]	0	0	0	0	0	0	1	1	1	1
mortoncode[bit2]	0	0	0	1	1	1	0	0	1	1
mortoncode[bit1]	0	0	1	0	0	1	0	1	0	1
mortoncode[bit0]	0	1	0	0	1	1	1	0	1	1
Primitives										

1 thread searches for 1 split

2 threads search for 2 splits

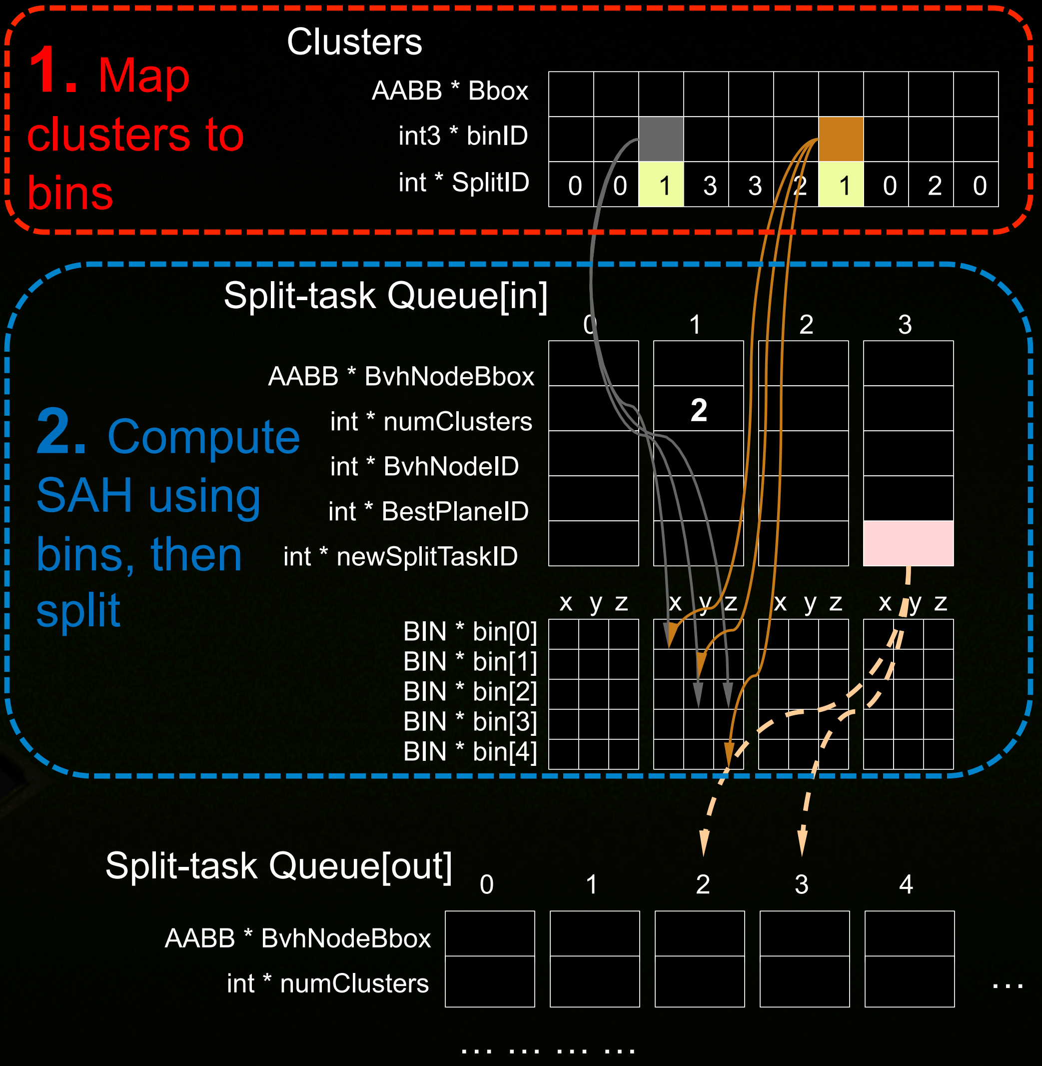
4 threads search for 4 splits

...

- Primitives sorted according to Morton Codes
- Find the split in the range of primitives using trivial binary search per thread: log2(n) memory fetches and comparison
- Very simple implementation of binary search merges well with simple work queues

Binned SAH top-level BVH

- 15bit clusters are good for high-quality BVH…
- Up to 32K clusters, each represent median-split BVH with a bounding box
- We have implemented [Wald IRT 2007] binned BVH builder using CUDA
- Atomic instructions of Fermi provide with good results and simple implementation
- Pros: Everything stays in GPU memory, no transfer costs



Results

- 10x faster than original HLBVH 2010
- Uses 4x less GPU device memory for BVH emission
- Quality of HLBVH is 10-15% lower than the quality of full SAH BVH
- Can build PowerPlant (12.7M triangles) in-core on GTX480 in 62ms

Results

- GTX 480:
  - Fairy Forest BVH (174K triangles) – 4.8ms
  - Turbine Blade BVH (2M triangles) – 10.5ms
  - Power Plant (12M triangles) – 62ms

Results

- Fast work queues enable a very high speed GPU based BVH build
- We believe they will enable a large class of parallel algorithms to be more straightforward on GPUs