OpenCL Path Tracing Best Practices

BRUNO STEFANIZZI
DMITRY KOZLOV

MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

AMD

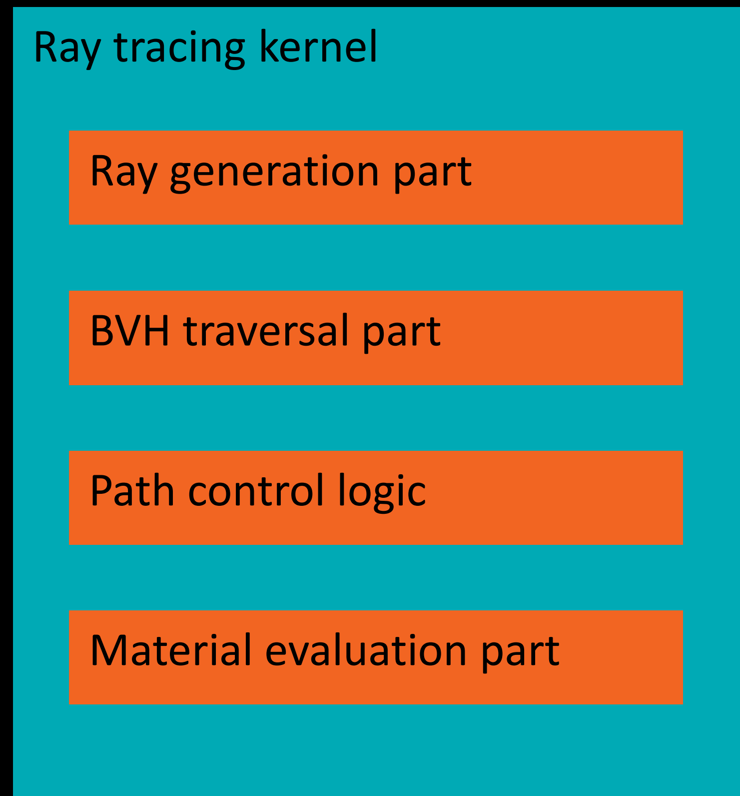
- Existing CPU codes are not for GPUs
- GPU CUs are optimized for higher throughput
- GPU is doing fine when arithmetic/memory ops ratio is high
- GPU has fair latency hiding capabilities
  - Make sure you have enough work for GPU
  - Make sure your code is fine-grained enough to exploit latency hiding
  - Make sure you have enough math work to cover memory access latency
  - **This means mega kernels are evil**

MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

Typical path tracing app kernel:

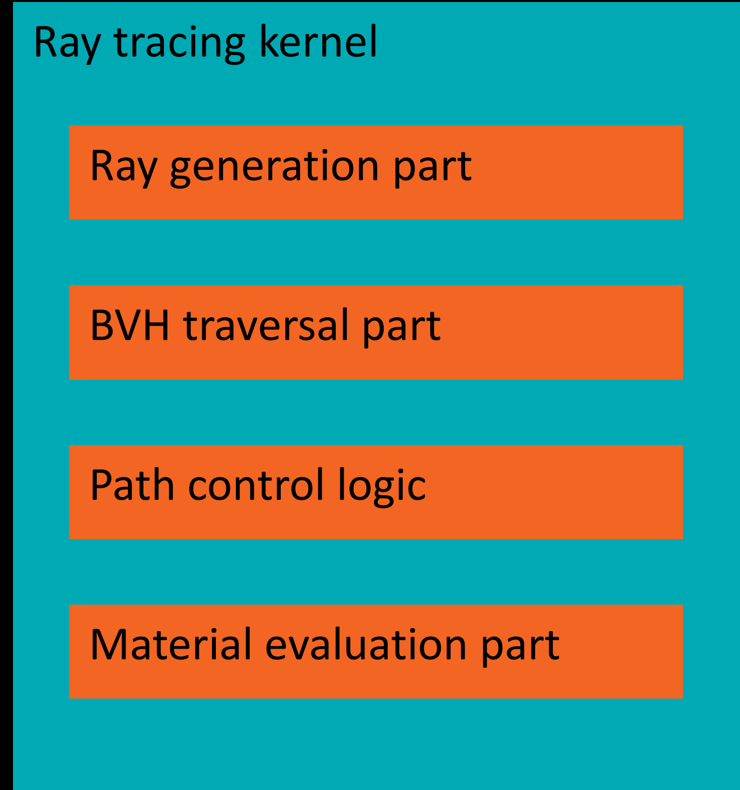
1. Establish ray state using some camera model
2. Cast ray against BVH
   - First bounce
     A. Get object’s material
     B. Sample direct illumination and evaluate BxDF closure for material
     C. Advance ray according to some closure sampling strategy
   - Second bounce
     A. …

Typically the stack of bounces in global memory for each ray is used



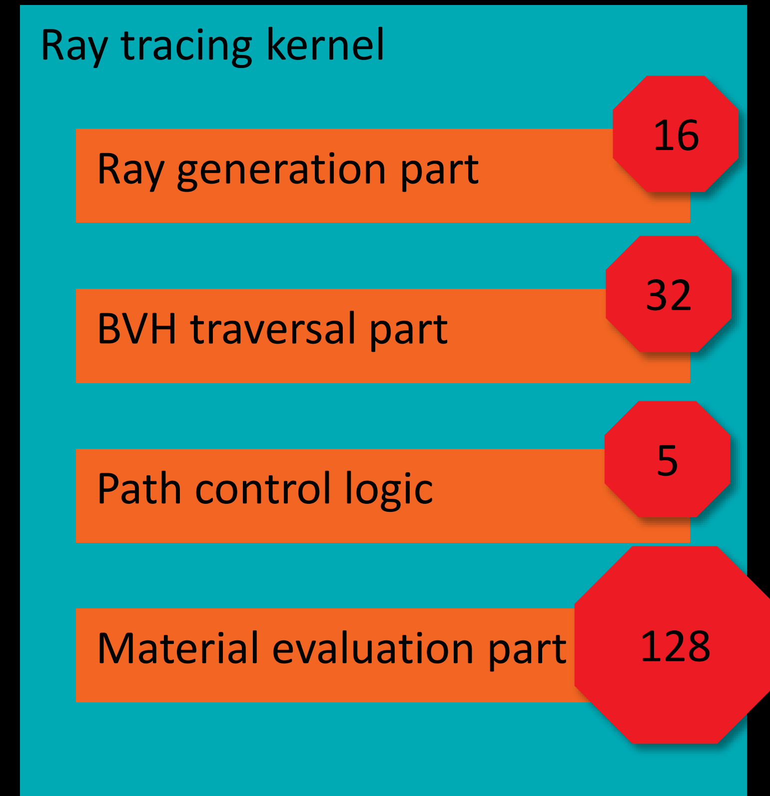
MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

- Let's track the occupancy
- GCN CU can have 2560 threads in flight
- Some limited resources
  - 256Kb VGPR file per CU
  - 8Kb SGPR file per CU



MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

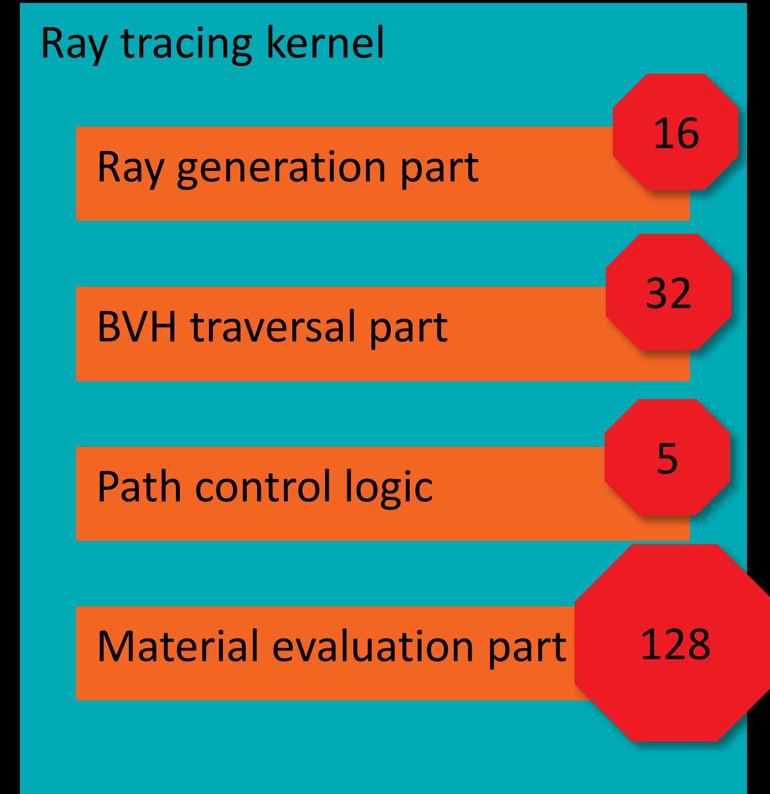
- Watch your GPR usage
- Ray generation part
  - Relatively lightweight
- BVH traversal
  - Memory intensive, low VGPR pressure
- Path control logic
  - High SGPR pressure, low VGPR usage
- Material part
  - High arithmetic intensity, extremely high VGPR usage



MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

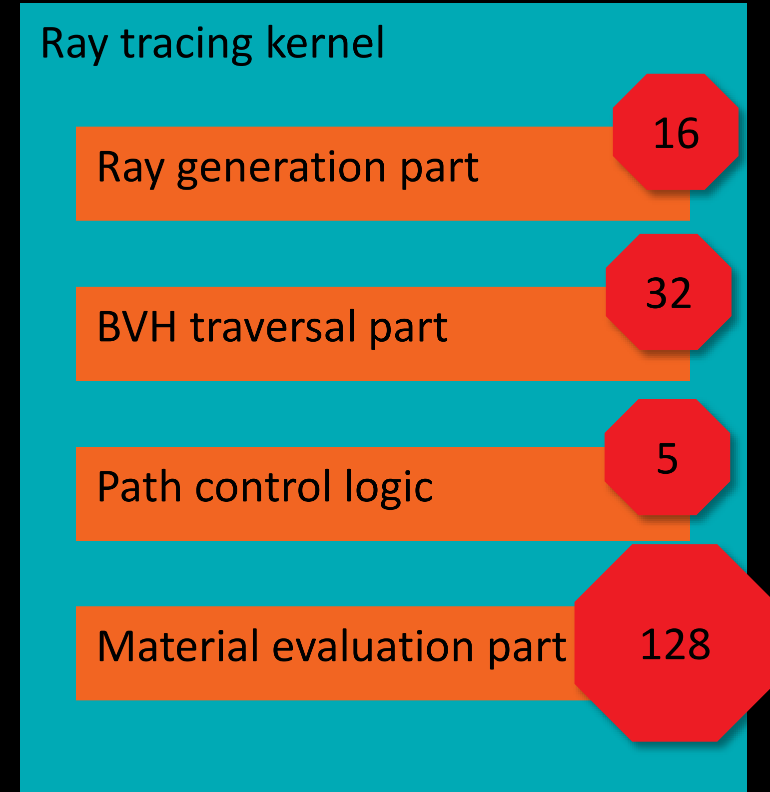
- 16 VGPRs = 40 waves / CU (100% occupancy)
- 32 VGPRs = 32 waves / CU (80% occupancy)
- 5 VGPRs = 40 waves / CU (100% occupancy)
- 128 VGRRs = 8 waves / CU (20% occupancy)

- The whole kernel occupancy is 20%☹
- Fully limited by material evaluation part



MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

- Another source of inefficiency is high thread divergence
- While one thread is still traversing BVH another evaluates materials resulting in serialization inside a wavefront ☹



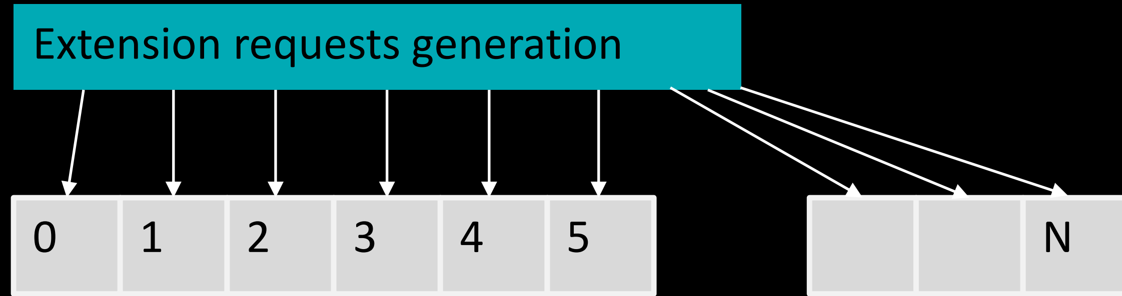
MAJOR SOURCES OF INEFFICIENCY IN GPU RAYTRACING ENGINES

- Split your code into multiple independent kernels
  - Separate material evaluation from intersection engine
  - Maintain the queue of work items for each material kernel
  - Try to sort material threads by material id to avoid divergence
- Sort and compact tasks where necessary
- Don’t waste the processing power on ray misses
  - Compact and sort tasks
- Avoid using recursion
- Keep your registers usage under control

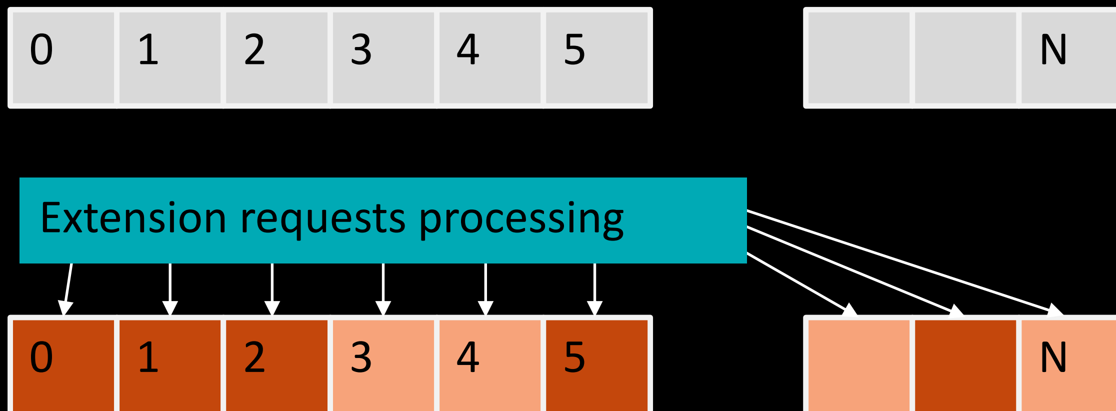
THE RIGHT WAY

AMD

- Consider using a short kernel to generate rays (path extension requests) into some buffer in global memory



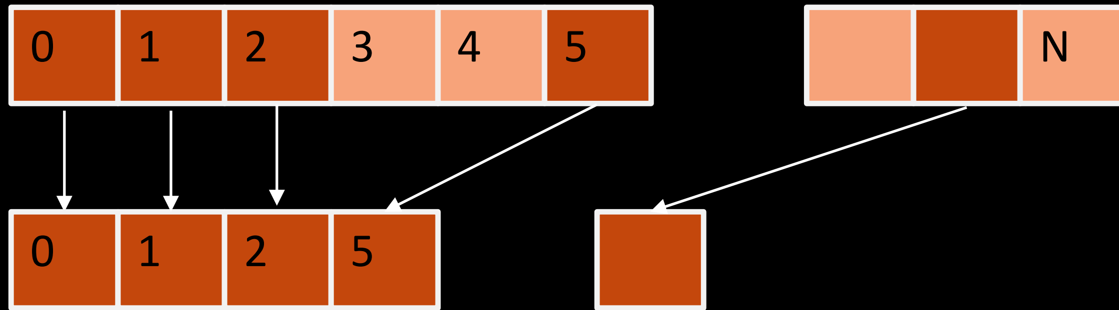
- Then use intersection kernel to transform requests buffer into actual hit information



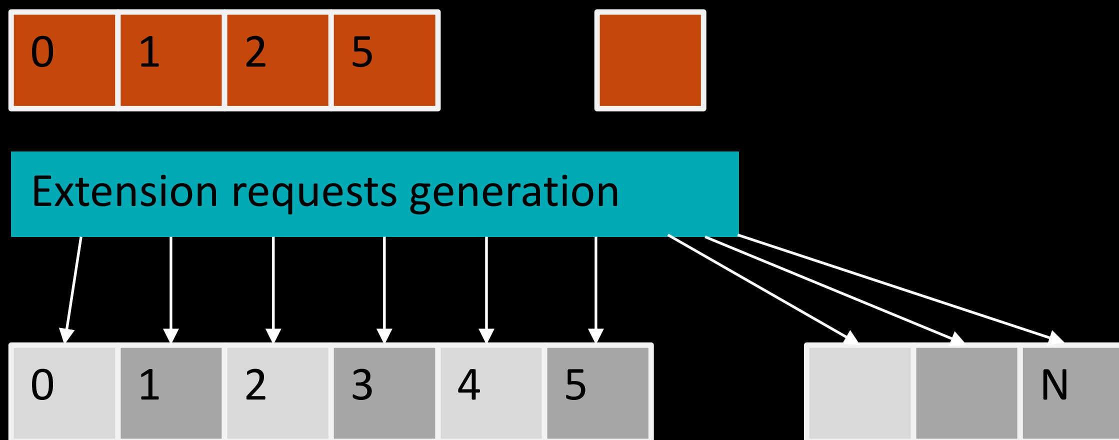
THE RIGHT WAY

AMD

- Compact the hits not to waste power on misses and minimize divergence

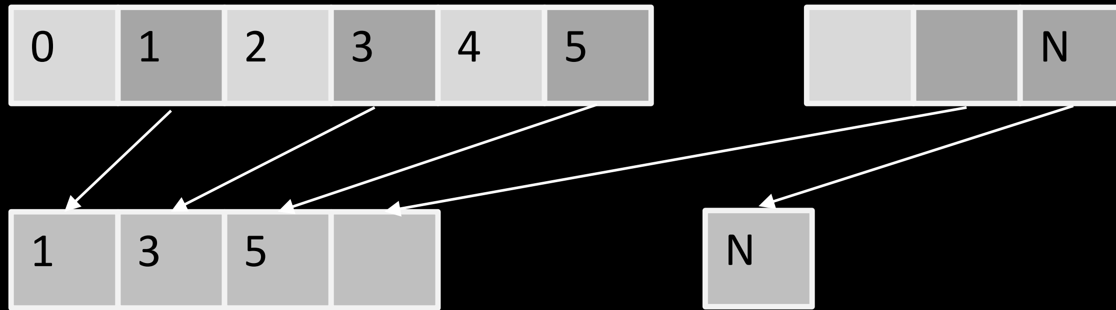


- Evaluate direct illumination on hits and produce extension requests to cast secondary rays



THE RIGHT WAY

- Compact secondary requests to throw away terminated rays



- Resort to recursion in case of small number of remaining items to avoid kernel launch overhead
- Resolve back the radiance

DEMO

AMD