Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum

Timo Aila Samuli Laine Tero Karras

NVIDIA Research

Abstract

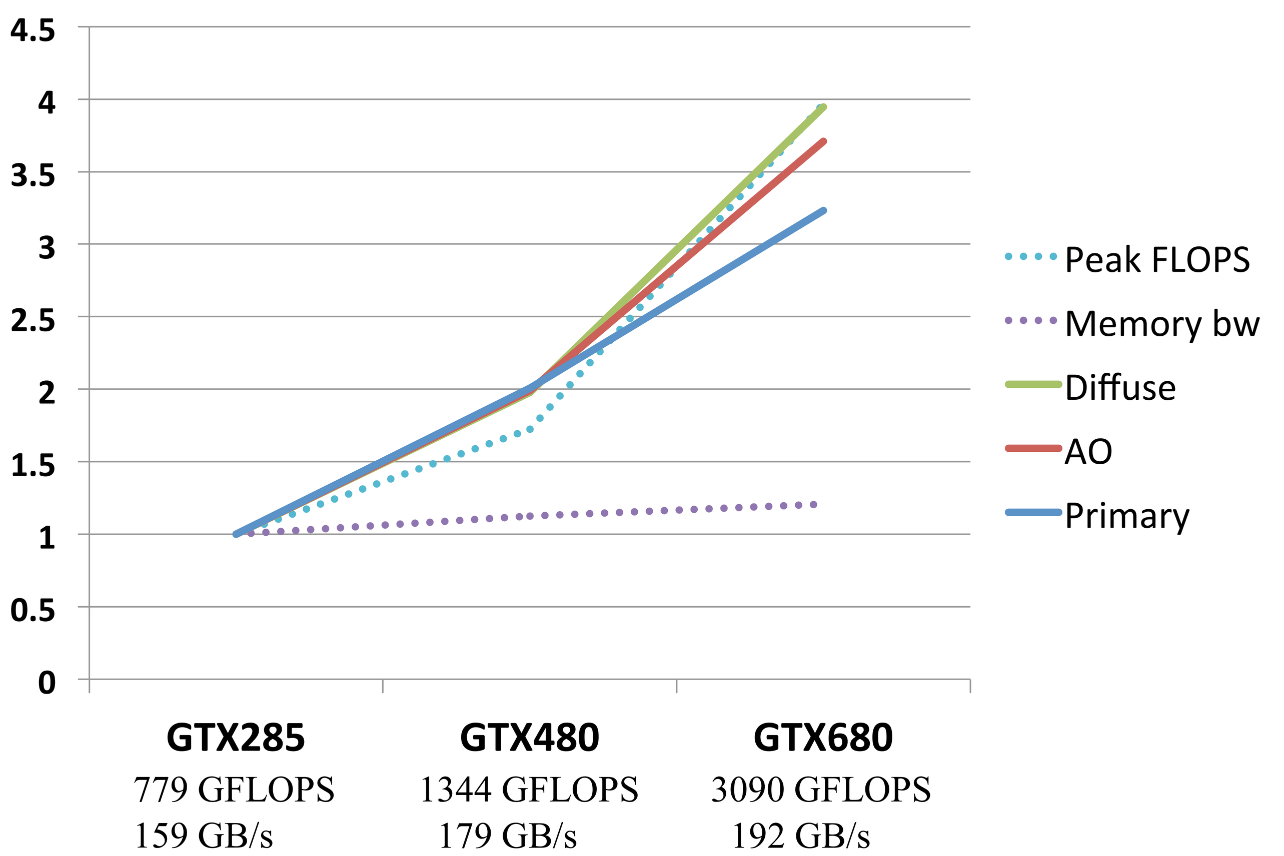
- This poster is an addendum to the HPG2009 paper "Understanding the Efficiency of Ray Traversal on GPUs" [AL09]
  - We cover the performance optimization of traversal and intersection kernels for Fermi and Kepler architectures [NVI10, NVI12a]
  - We demonstrate that ray tracing is still, even with incoherent rays and more complex scenes, almost entirely limited by the available FLOPS
  - We also discuss two esoteric instructions, present in both Fermi and Kepler, and show how they can be used for faster acceleration structure traversal

Implications of memory architecture

- Fermi
  - Has L1 and L2 caches
  - L1 services only one cache line per clock
    - Fetch instruction replayed until all threads of a warp serviced
  - Divergent accesses bottlenecked by L1 → SM
    - Even with high hit rate and abundant memory bandwidth
    - Speculative traversal is less useful than on Tesla
  - Texture units are not fast enough to handle all traffic
    - Split workload between L1 and texture
- Kepler
  - Significant upgrade to FLOPS and texture units
  - Same L1 and L2
  - L1 is useful only for coherent, low-priority accesses
    - Rays, traversal stacks
  - Fetches through texture cache are divergence-tolerant
    - Latency is high, dependent fetches should be avoided
    - Speculative traversal is useful again
  - It is beneficial to replace terminated rays when SIMD utilization drops below a threshold
    - As speculated in [AL09]
    - We use a threshold of 60%

Performance and scalability

- Relative average performance (MRays/sec) of primary, ambient occlusion, and diffuse rays in our four test scenes on GTX285, GTX480, and GTX680, plotted against the relative memory bandwidth and peak FLOPS
  - Ray tracing performance continues to follow peak flops very closely, while memory bandwidth has increased at a much slower rate
  - Interestingly, diffuse rays seem to scale even better than primary rays, but that is an artifact caused by our Kepler-specific optimizations that favor incoherent rays (See Table 2)



Conference, 283K tris

Fairy, 174K tris

Sibenik, 80K tris

San Miguel, 11M tris

		Conference, 283K tris			Fairy, 174K tris			Sibenik, 80K tris			San Miguel, 11M tris		
	Ray type	Tesla [AL09]	Fermi	Kepler	Tesla	Fermi	Kepler	Tesla	Fermi	Kepler	Tesla	Fermi	Kepler
Measured (MRays/s)	Primary	142.2	272.1	432.6	74.6	154.6	250.8	117.5	243.4	388.2	–	76.9	131.7
	AO	134.5	284.1	518.2	92.5	163.6	317.6	119.6	244.1	441.2	–	94.5	187.9
	Diffuse	60.9	126.1	245.4	40.8	73.2	156.6	46.8	94.7	192.5	–	33.3	58.8
× previous architecture	Primary		1.91	1.59		2.07	1.62		2.07	1.59			1.71
	AO		2.11	1.82		1.77	1.94		2.04	1.81			1.99
	Diffuse		2.07	1.95		1.79	2.14		2.02	2.03			1.77

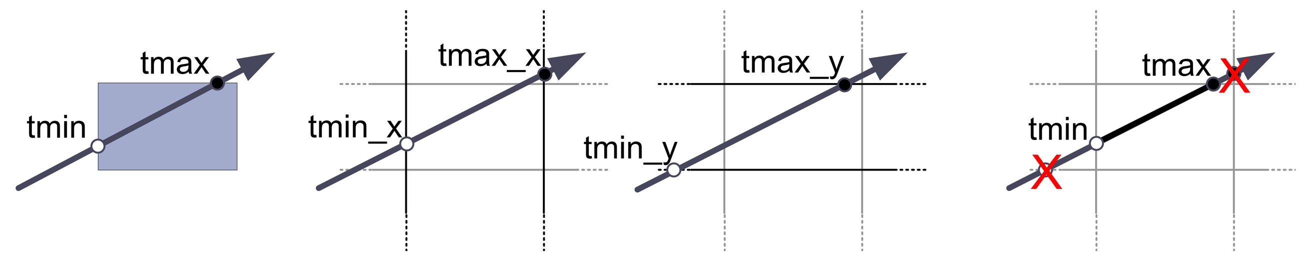
Table 1: *Performance measurements in MRays/sec for Tesla (GTX285), Fermi (GTX480) and Kepler (GTX680) using the setup from [AL09]. The San Miguel scene is from PBRT. The scaling between generations is visualized above.*

	Tesla	Fermi	Kepler
Data fetches	Fetch nodes through texture, triangles from (uncached) global memory.	Fetch nodes through L1, triangles via texture. L1 is a bottleneck with incoherent rays but texture is not fast enough to fetch everything.	Fetch all node and triangle data via texture, and avoid dependent fetches whenever possible. L1 is slow and beneficial only for traversal stacks and ray fetches.
Persistent threads	Doubles performance.	Not beneficial due to a better hardware work distributor.	Can replace all terminated rays once fewer than 60% of the warp's lanes have work. Favors incoherent rays, +10%.
Speculative traversal	Nearly always useful.	A small win for incoherent rays and large scenes.	One or two-slot postpone buffer is beneficial for incoherent rays.
VMIN, VMAX	Not available.	A trivial +10%.	Less useful than on Fermi because of lower throughput, +5%.

Table 2: *Summary of differences in traversal and intersection kernels for Tesla, Fermi, and Kepler.*

VMIN, VMAX

- Large portion of BVH traversal is ray-box tests
  - Most of ray-box is min and max instructions
- Fermi and Kepler have two esoteric instructions:
  - vmin.max(a,b,c) = max(min(a,b),c)
  - vmax.min(a,b,c) = min(max(a,b),c)
  - Exposed through PTX [NVI12b]
  - Only integer data types supported



Tesla

```
DEFINITIONS
B = Box (xmin,ymin,zmin,xmax,ymax,zmax);
O = ray origin (x,y,z);
D = ray direction (x,y,z);
invD = (1/D.x,1/D.y,1/D.z);
OoD  = (O.x/D.x,O.y/D.y,O.z/D.z);
tminray = ray segment's minimum t value; ≥ 0
tmaxray = ray segment's maximum t value; ≥ 0

RAY vs. AXIS-ALIGNED BOX
// Plane intersections (6 x FMA)
float x0 = B.xmin*invD[x] - OoD[x]; [-∞ ,∞ ]
float y0 = B.ymin*invD[y] - OoD[y]; [-∞ ,∞ ]
float z0 = B.zmin*invD[z] - OoD[z]; [-∞ ,∞ ]
float x1 = B.xmax*invD[x] - OoD[x]; [-∞ ,∞ ]
float y1 = B.ymax*invD[y] - OoD[y]; [-∞ ,∞ ]
float z1 = B.zmax*invD[z] - OoD[z]; [-∞ ,∞ ]

// Span intersection (12 x 2-way MIN/MAX)
float tminbox = max4(tminray, min2(x0,x1),
                     min2(y0,y1), min2(z0,z1));
float tmaxbox = min4(tmaxray, max2(x0,x1),
                     max2(y0,y1), max2(z0,z1));

// Overlap test (1 x SETP)
bool intersect = (tminbox<=tmaxbox);
```

Fermi, Kepler

- Can safely use integer VMIN and VMAX in ray-box test
  - Intermediate results often wrong
  - End result provably correct
- Traversal and intersection performance
  - Fermi: +10%
  - Kepler: +5% due to lower throughput of the instructions

```
// Span intersection (6 x VMIN/VMAX)
float tminbox = vmin.max(x0,x1,vmin.max(y0,y1,
                     vmin.max(z0,z1,tminray)));
float tmaxbox = vmax.min(x0,x1,vmax.min(y0,y1,
                     vmax.min(z0,z1,tmaxray)));
```

Summary

- Our recommendations are summarized in Table 2
- Extended version is available as technical report NVR-2012-02 at http://research.nvidia.com
- Optimized kernels are available at http://code.google.com/p/understanding-the-efficiency-of-ray-traversal-on-gpus/

References

[AL09] AILA T., LAINE S.: Understanding the efficiency of ray traversal on GPUs. In *Proc. High-Performance Graphics 2009* (2009), pp. 145–149.

[NVI10] NVIDIA: NVIDIA's next generation CUDA compute architecture: Fermi. Whitepaper, 2010.

[NVI12a] NVIDIA: NVIDIA's next generation CUDA compute architecture: Kepler GK110. Whitepaper, 2012.

[NVI12b] NVIDIA: Parallel thread execution ISA version 3.0. Whitepaper, 2012.