Extended Data Collection: Analysis of Cache Behavior and Performance of Different BVH Memory Layouts for Tracing Incoherent Rays

Technical Report 13rp003-GRIS

A. Schulz[1], D. Wodniok[2], S. Widmer[2] and M. Goesele[2]

[1]TU Darmstadt, Germany
[2]Graduate School Computational Engineering, TU Darmstadt, Germany

1 Introduction

This technical report complements the paper "Analysis of Cache Behavior and Performance of Different BVH Memory Layouts for Tracing Incoherent Rays" by Wodniok et al. published in the proceedings of the "Eurographics Symposium on Parallel Graphics and Visualization" [WSWG13]. Please see this main paper for details. The purpose of this report is to publish the complete data collection the paper is based on using the NVIDIA Kepler architecture plus additional data collected on the NVIDIA Fermi architecture.

2 Metrics

Several different metrics are computed using the event counters from CUPTI. Some of these can be found in the CUPTI User's Guide [NVIb] others were deduced from values in Parallel Nsight and reconstructing them with events from CUPTI. The formulas in this section use the event names as defined by CUPTI (refer to [NVIb] for more information). A short explanation of each metric follows:

Runtime
Trace kernel runtime in milliseconds, measured using CUPTI's activity API.

L1 global load hit rate
Percentage of global memory loads that hit in L1. Higher is better.

$$\frac{l1\_global\_load\_hit}{l1\_global\_load\_hit + l1\_global\_load\_miss} \times 100$$

L1 global load size
Amount of data transferred by global memory loads. Lower is better.

$$(l1\_global\_load\_hit + l1\_global\_load\_miss) \times 128\ bytes$$

L1 global load transactions per request

Number of cache lines read per global memory load request of a warp. Range is $[1,32]$. If a whole warp executes a 16-byte load it always results in at least 4 transactions because the warp is requesting 512 bytes but the hardware can only load up to 128 bytes in a single transaction. When all threads in a warp access a single 16-byte value, the value is broadcast to up to 8 threads at once resulting in 4 transactions. The number of transactions can also be lower than 4 in the aforementioned cases if a warp contains fewer active threads. Lower is better.

$$\frac{l1\_global\_load\_hit + l1\_global\_load\_miss}{gld\_request}$$

L1←L2 load hit rate

Percentage of global memory loads that missed in L1 but hit in L2. Higher is better.

$$\frac{l2\_subp0\_read\_hit\_sectors + l2\_subp1\_read\_hit\_sectors}{l2\_subp0\_read\_sector\_queries + l2\_subp1\_read\_sector\_queries} \times 100$$

L1←L2 load size

Amount of data transferred from L2 cache by global memory loads. Lower is better.

$$(l2\_subp0\_read\_sector\_queries + l2\_subp1\_read\_sector\_queries) \times 32\ bytes$$

Tex cache hit rate

Percentage of texture memory loads that hit in the texture memory cache. Higher is better.

$$\frac{tex0\_cache\_sector\_queries - tex0\_cache\_sector\_misses}{tex0\_cache\_sector\_queries} \times 100$$

Tex load size

Amount of data transferred by texture memory loads from the texture memory cache. Lower is better.

$$tex0\_cache\_sector\_queries \times 32\ bytes$$

Tex←L2 load hit rate

Percentage of texture memory loads that missed in the L1 cache but hit in the L2 cache. Higher is better.

$$\frac{l2\_subp0\_read\_tex\_hit\_sectors + l2\_subp1\_read\_tex\_hit\_sectors}{l2\_subp0\_read\_tex\_sector\_queries + l2\_subp1\_read\_tex\_sector\_queries} \times 100$$

Tex←L2 load size

Amount of data transferred from the L2 cache by texture memory loads. Lower is better.

$$(l2\_subp0\_read\_tex\_sector\_queries + l2\_subp1\_read\_tex\_sector\_queries) \times 32\ bytes$$

Shared memory load size

Amount of data transferred by shared memory loads. This is actually impossible to compute without explicit knowledge of the kernel's code because the *shared_load* event increments by 1 regardless of the size of the load instruction used. The reason why we can compute this metric is because all loads are guaranteed to be 8 bytes. Thus the shared memory load size is: $shared\_load \times 8$. Lower is better.

Shared memory bank conflicts per request

Shared memory is divided into 32 banks. If threads in a warp access the same bank but with different addresses a bank conflict happens and access is serialized. Lower is better.

$$\frac{l1\_shared\_bank\_conflict}{shared\_load}$$

Device memory load size

Amount of data transferred from global/device memory. Lower is better.

$$(fb\_subp0\_read\_sectors + fb\_subp1\_read\_sectors) \times 32\ bytes$$

Instruction replay overhead

Percentage of instructions that were issued due to replaying memory accesses, such as cache misses. Lower is better.

$$\frac{instructions\_issued - instructions\_executed}{instructions\_issued} \times 100$$

IPC

Instructions executed per cycle. The Fermi GPU can issue up to 2 instructions per cycle which means the range for this metrics is $[0,2]$. Higher is better.

$$\frac{instructions\_executed}{num\_multiprocessors \times elapsed\_clocks}$$

SIMD efficiency

Also called *warp execution efficiency*. Percentage of average active threads per warp to total number of threads in a warp. Higher is better.

$$\frac{thread\_inst\_executed\_0 + thread\_inst\_executed\_1}{instructions\_executed \times warp\_size}$$

Branch efficiency

Measures SIMD divergence. Percentage of coherent branches to all branches. Higher is better.

$$\frac{branch - divergent\_branch}{branch} \times 100$$

Achieved occupancy

Percentage of average number of active warps to maximum number of warps supported on a multiprocessor. Higher is better.

$$\frac{active\_warps}{48 \times active\_cycles} \times 100$$

Crytek Sponza	**Kitchen**
≈262K Triangles	≈426K Triangles
99114 BVH Nodes	296521 BVH Nodes
6.05 BVH Size(MB)	18.10 BVH Size(MB)
Hairball	**San - Miguel**
≈2,880K Triangles	≈7,880K Triangles
1900973 BVH Nodes	3513623 BVH Nodes
116.03 BVH Size(MB)	214.46 BVH Size(MB)

Table 1: Scenes used for benchmarking.

3 Scenes

For testing the performance of the different BVH and node layouts four different scenes of varying complexity and with different materials were used (Table 1). The Cornell Box scene contains two spheres, one with a glass material and one with translucent material. The crytek-sponza scene is the improved version by Frank Meinl at Crytek [Sm]. Both the crytek-sponza and san-miguel scenes contain only diffuse materials. The kitchen scene contains a number of objects with glass material.

4 Evaluation - Fermi architecture

All experiments were run on a computer equipped with an Intel Core i7-960 CPU clocked at 3.2 GHz, an Nvidia Tesla C2070 GPU, Ubuntu 12.04.1 LTS running Linux kernel version 3.2.0-36-generic as the operating system, GCC 4.6.3, NVIDIA display driver 304.64 and CUDA toolkit 5.0. The Tesla C2070 consists of 15 multi-processors which in turn consist of 32 processors each. Memory-wise it has 6144 MB of global/-texture memory, 16 or 48 KB of shared memory or global memory L1 cache depending on its runtime configuration and 32768 registers per multi-processor. Size of the L2 cache is 768KB.

4.1 Memory access properties

We used micro-benchmarking ([WPSAM10]) to derive memory access properties. Fetch latency for a global memory load of 4 bytes that hits in L1 is ≈ 32 cycles, a hit in L2 is ≈ 395 cycles while missing both costs ≈ 523 cycles. L1 texture memory cache size is 12KB with a cache line size of 128B. L1 hit latency for reading 4 bytes is ≈ 220 cycles, L2 hit latency is ≈ 524 cycles and missing both L1 and L2 incurs a latency of ≈ 647 cycles.

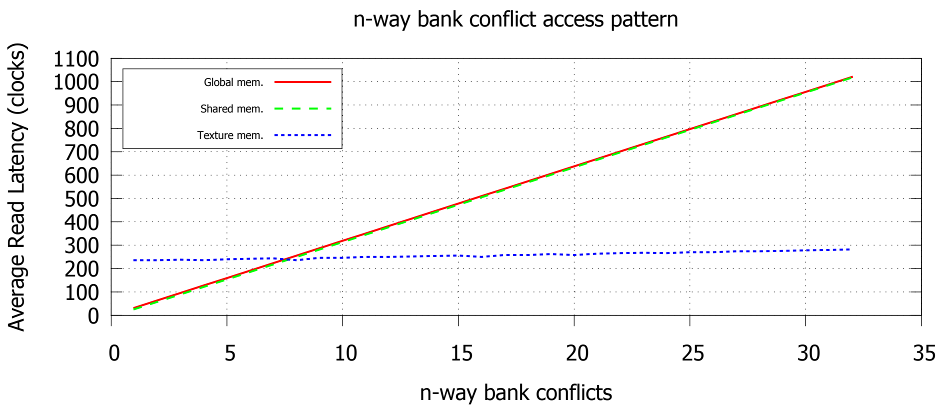
Figure 1 shows the latency of letting an increasing number of threads in a warp access cached memory locations with stride threadID $* 128B$ and stride threadID $* 132B$. The first access pattern results in n-way bank conflicts for shared memory and is a worst case for global memory, as each thread reads from a different memory segment leading to complete serialization of the request. Latency of both is the same, as L1 cache and shared memory are the same hardware. Texture memory latency stays constant and starts to be lower than L1 latency as soon as at least 8 different L1 cache lines are accessed. The second access pattern results in no bank conflicts for shared memory but again is a worst case for global memory, as each thread reads from a different memory segment. Texture memory latency behaves the same as before. Thus we can see that texture memory performs equally well for access patterns which are worst for either global or shared memory. Figure 2 shows the broadcasting capabilities of global and texture memory when several threads in a warp read the same 4 byte word. We can see, that global memory needs less transactions than texture memory.

4.2 Baseline

The baseline BVH is laid out in DFS order and stores nodes in AoS format. The AoS node format was chosen because Aila et al. [AL09] are using it in their GPU ray traversal routines which are one of the fastest. Tree nodes are accessed via global memory and geometry via texture memory. The trace kernel was profiled using a path tracer, 1024x768 pixel, 32spp, DFS BVH layout, AoS node layout. Figures 3, 4, 5 and 6 show the runtime behavior (in ms) and GPU metrics (percentage) over all render loop iterations for our test scenes with BVH nodes stored either in global memory (left) or texture memory (right).

4.3 BVH and node layouts

Tables 2 and 3 show a ranking of all BVH and node layout combinations which were accessed via global memory or texture memory. The ranking is performed w.r.t. the average achieved speedup compared to the DFS layout in the respective memory area. The SWST, TDFS and TBFS layouts require a threshold probability. We have tested a number of different values to find the best performing one. The best threshold is required to perform well for all scenes in our data set so that its performance extends to unknown data sets. We use the sum of the scene runtimes to measure the performance of a threshold and choose the best performing ones. The determined thresholds are stated next to the respective BVH layout names in the tables. Following, we will compare the best performing combinations of threshold, BVH and node layout in each memory area to the other introduced BVH layouts.



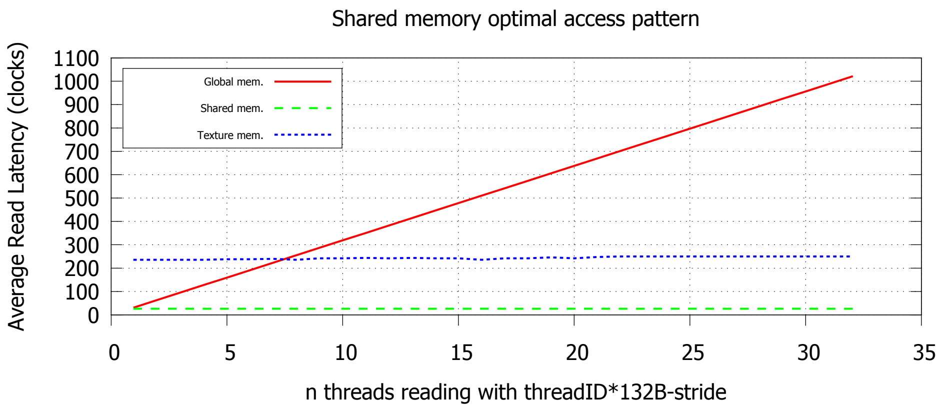


Figure 1: **Tesla C2070** - Latency plots of two access patterns which are either the worst case (top) or optimal (bottom) for shared memory compared with the latency of directly hitting in global or texture memory with the same pattern. Both patterns are worst case access patterns for global memory as access has to be serialized, though they hit in cache. In both cases texture memory latency stays constant and performs better than global memory, as soon as at least 8 different memory segments are accessed.

4.4 Best performing layout

The best performing BVH layout for nodes stored in global memory is the TDFS layout with a threshold of 0.4 in combination with the SoA32_24 node layout, shown on the left side of the figures 7, 8, 9 and 10. For nodes stored in texture memory a TDFS layout with a threshold of 0.5 and AoS node layout is most beneficial. In global memory we have achieved runtime reduction by 2.8 – 6.7%. In texture memory, we gained $\approx$ 5.0 – 17.6% runtime reduction compared to the baseline in global memory. Thus, contrary to [ALK12] our path tracer benefited from using texture memory for loading nodes when run on a Fermi GPU. Also accessing the baseline in texture memory, an improvement of only $\approx$ 2.3% was observable in the san-miguel scene for treelet based layouts. We attribute the smaller amount of data transferred when using global memory to superior broadcast capabilities (see Section 4.1).

We have also tried to leverage the unused shared memory by using it as a static cache for a part of the BVH but were unable to achieve any advantages over using only

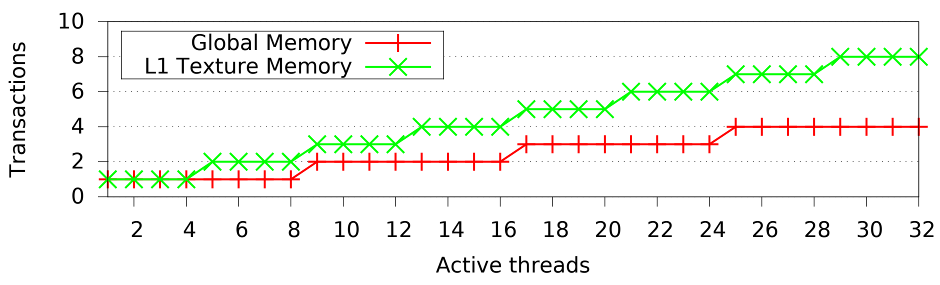


Figure 2: **Tesla C2070** - Number of transactions for a broadcast in global and texture memory for an increasing number of threads in a warp.

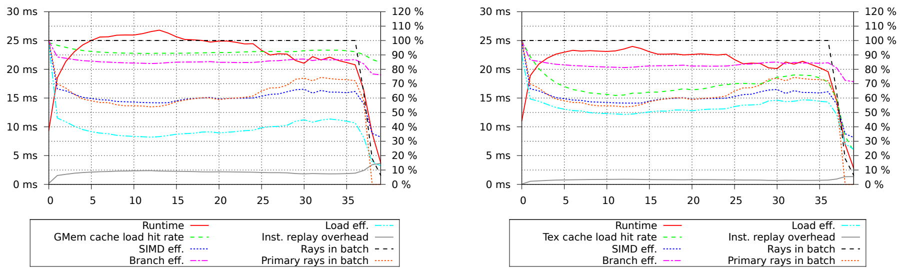


Figure 3: **Tesla C2070** - **Crytek Sponza** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

a single memory area.

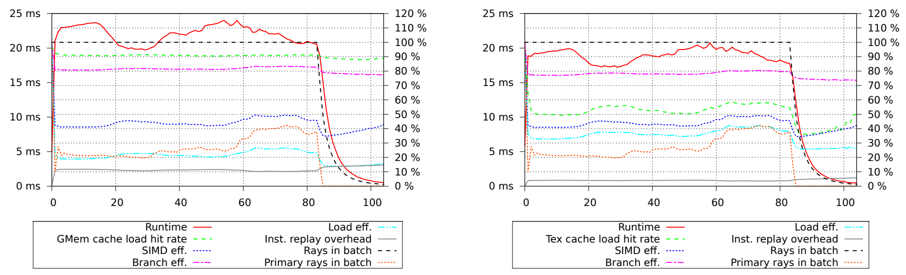


Figure 4: **Tesla C2070 - Kitchen** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

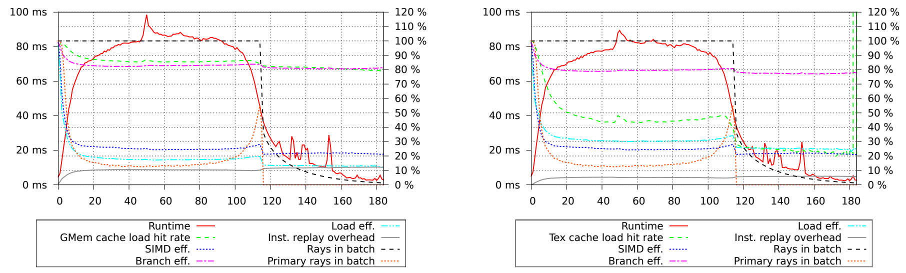


Figure 5: **Tesla C2070 - Hairball - Glass** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

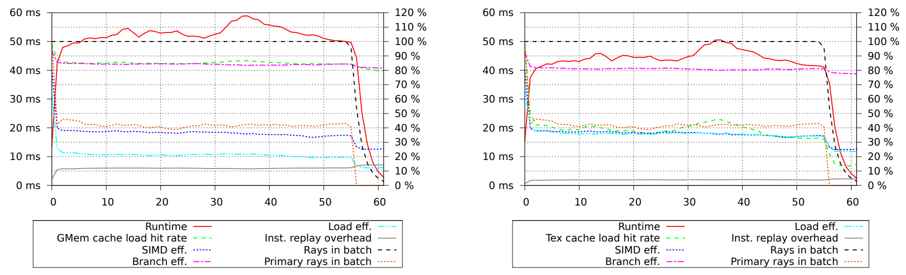


Figure 6: **Tesla C2070 - San Miguel** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

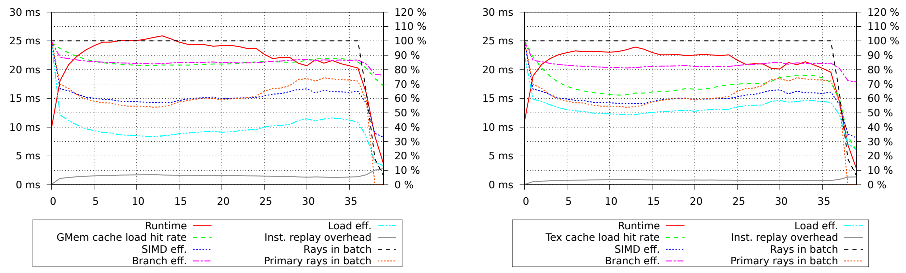


Figure 7: **Tesla C2070 - Crytek Sponza** - Trace kernel profiling graph for the best layouts for BVH nodes stored in global memory (left, TDFS 0.4, SoA32_24) and stored in texture memory (right, TDFS 0.5, AoS).

GMem													
BVH lay.	node lay.	crytek-sponza			kitchen			hairball-glass			san-miguel		
		R	H	SZ	R	H	SZ	R	H	SZ	R	H	SZ
TDFS 0.4	Soa32_24	877.7	85.0	310.4	1868.5	84.0	698.6	9492.3	68.7	2534.7	2872.7	68.9	886.0
SWST 0.5	Soa32_24	882.5	84.6	311.3	1900.2	82.6	700.0	9489.5	68.8	2534.1	2898.1	68.8	886.3
COL	Soa32_24	885.9	84.3	311.1	1886.5	82.8	693.3	9597.6	67.2	2532.1	2912.0	68.0	884.6
TBFS 0.5	Soa32_24	886.5	84.2	312.2	1873.4	83.0	688.7	9786.7	65.6	2536.9	2910.4	67.5	883.9
vEB	Soa32_24	884.9	84.3	311.4	1908.4	81.5	693.0	9608.1	66.9	2529.9	2920.0	67.2	882.2
DFS	Soa32_24	897.6	83.3	313.2	1951.4	80.4	704.4	9606.3	67.3	2535.1	2980.4	66.3	889.1
BFS	Soa32_24	902.2	82.7	311.9	1944.9	78.5	678.8	10077.6	62.0	2525.8	3039.1	63.3	880.2
DFS	Aos	903.1	92.1	315.8	2004.4	90.6	708.2	9888.7	84.7	2554.5	3021.7	85.0	893.6
COL	Aos	905.6	92.0	315.7	1999.8	90.7	704.7	10037.6	83.8	2554.7	3043.8	84.4	892.4
vEB	Aos	909.0	91.8	316.0	2009.0	90.4	704.4	10068.9	83.4	2556.1	3053.4	83.9	892.3
BFS	Aos	917.8	91.4	316.3	2026.1	89.7	698.2	10278.6	82.2	2554.5	3099.9	82.9	890.4
vEB	Soa16_8	967.1	72.8	304.4	2106.7	67.3	674.3	11270.4	45.1	2498.0	3546.9	44.4	870.0
COL	Soa16_8	969.3	72.9	304.8	2094.6	69.0	676.1	11360.6	44.7	2511.0	3567.8	44.2	875.0
BFS	Soa16_8	992.0	70.7	305.9	2137.9	63.2	653.7	12235.8	36.2	2486.0	3759.5	39.5	867.0
DFS	Soa16_8	1022.5	68.2	310.1	2267.3	61.8	699.0	11743.0	43.2	2524.3	3883.9	40.0	886.1

Table 2: **Ranking Tesla C2070** - BVH (global memory) and node layout combinations sorted ascending by average speedup over all scenes accessed via global memory. **R**untime in ms, **H**itrate in percent and **SZ** denotes the total amount of data transferred (in GB).

TMem													
BVH lay.	node lay.	crytek-sponza			kitchen			hairball-glass			san-miguel		
		R	H	SZ	R	H	SZ	R	H	SZ	R	H	SZ
TDFS 0.5	Aos	832.1	68.0	270.3	1726.9	52.1	559.3	9391.0	40.1	2200.9	2489.6	37.7	663.3
TBFS 0.1	Aos	832.5	67.8	270.2	1726.9	52.1	559.2	9392.4	40.4	2200.6	2494.2	37.8	663.2
BFS	Aos	832.6	67.6	270.2	1728.2	52.1	559.3	9386.1	40.4	2200.4	2494.2	37.8	663.2
vEB	Aos	832.4	67.8	270.3	1727.8	51.7	559.3	9377.7	40.1	2200.6	2499.4	37.7	663.3
COL	Aos	833.0	67.8	270.2	1729.0	51.9	559.3	9391.2	39.9	2200.7	2514.7	37.6	663.4
DFS	Aos	833.6	67.7	270.2	1730.8	51.5	559.3	9393.3	40.1	2200.3	2549.0	37.5	664.0
BFS	Soa32_24	848.1	62.5	270.4	1754.3	44.9	559.5	9630.3	33.4	2201.0	2667.9	32.1	663.6
vEB	Soa32_24	847.7	62.6	270.3	1756.1	44.3	559.5	9621.6	33.0	2201.2	2680.6	31.9	663.7
COL	Soa32_24	848.8	62.6	270.4	1757.2	44.4	559.6	9642.7	33.2	2201.4	2712.9	31.8	664.0
DFS	Soa32_24	852.6	62.3	270.3	1771.3	43.6	559.6	9658.5	32.9	2201.6	2768.0	31.6	664.7
vEB	Soa16_8	884.9	48.7	269.0	1846.6	20.9	556.6	10423.1	13.3	2192.7	3162.2	7.7	660.5
COL	Soa16_8	886.5	48.9	269.0	1844.3	21.5	556.8	10479.7	14.0	2192.3	3190.4	7.7	661.0
BFS	Soa16_8	889.2	48.0	269.1	1849.8	21.3	553.1	10603.3	13.9	2191.7	3186.7	7.3	659.7
DFS	Soa16_8	904.3	47.6	269.2	1904.7	19.9	559.0	10540.0	12.1	2194.3	3358.6	7.1	662.6

Table 3: **Ranking Tesla C2070** - BVH (texture memory) and node layout combinations sorted ascending by average speedup over all scenes accessed via global memory. **R**untime in ms, **H**itrate in percent and **SZ** denotes the total amount of data transferred (in GB).

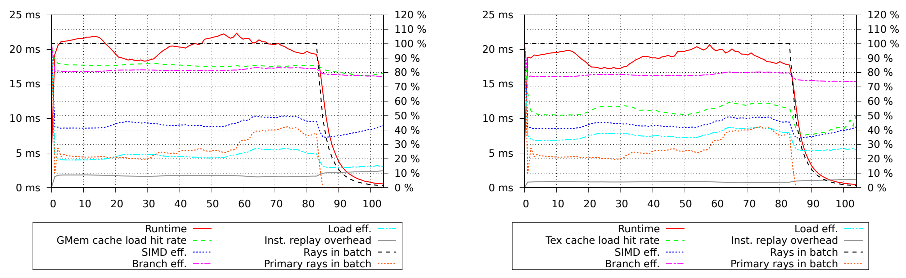


Figure 8: **Tesla C2070 - Kitchen** - Trace kernel profiling graph for the best layouts for BVH nodes stored in global memory (left, TDFS 0.4, SoA32_24) and stored in texture memory (right, TDFS 0.5, AoS).

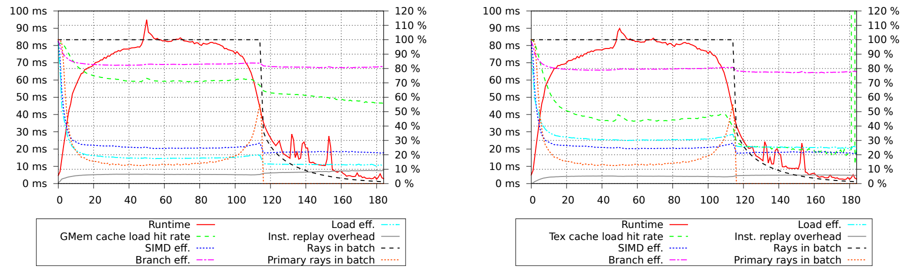


Figure 9: **Tesla C2070 - Hairball - Glass** - Trace kernel profiling graph for the best layouts for BVH nodes stored in global memory (left, TDFS 0.4, SoA32_24) and stored in texture memory (right, TDFS 0.5, AoS).

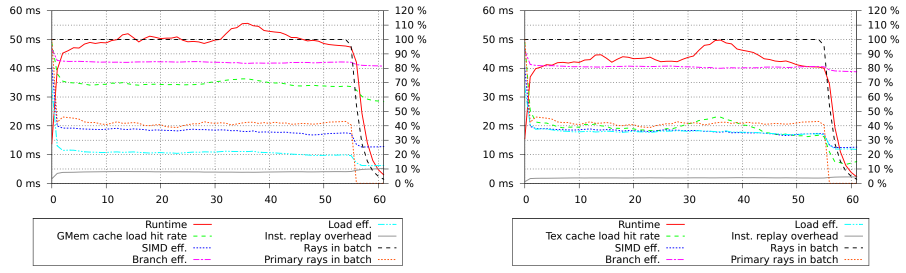


Figure 10: **Tesla C2070 - San Miguel** - Trace kernel profiling graph for the best layouts for BVH nodes stored in global memory (left, TDFS 0.4, SoA32_24) and stored in texture memory (right, TDFS 0.5, AoS).

5 Evaluation - Kepler architecture

All experiments were performed using the system described in Section 4 but equipped with a Geforce GTX 680 GPU instead of a Tesla C2070. The Geforce GTX 680 consists of eight Streaming Multiprocessors (SMX) with 192 CUDA cores each. It provides 2048 MB of global/texture memory, 16, 32 or 48 KB of shared memory or L1 cache for local memory (depending on the runtime configuration) and 65536 registers per SMX. Accesses to global memory bypass the L1 cache and directly go through the 512KB L2 cache.

5.1 Memory access properties

Again we used micro-benchmarking ([WPSAM10]) to derive memory access properties. L2 hit latency is $\approx$ 160 cycles while a miss costs 290 cycles. L1 texture memory cache size is 12KB with a cache line size of 128B (see 11). L1 hit latency for reading 4 bytes is $\approx$ 105 cycles, L2 hit latency is $\approx$ 266 cycles and missing both L1 and L2 incurs a latency of $\approx$ 350 cycles.

Figure 12 shows the latency of letting an increasing number of threads in a warp access cached memory locations with stride threadID $*$ $128B$ and stride threadID $*$ $132B$. The first access pattern results in n-way bank conflicts for shared memory and is a worst case for global memory, as each thread reads from a different memory segment leading to complete serialization of the request. Texture memory latency stays constant and is lower than global memory access latency. The second access pattern results in no bank conflicts for shared memory but again is a worst case for global memory, as each thread reads from a different memory segment. Texture memory latency behaves the same as before. Thus we can see that texture memory performs equally well for access patterns which are worst for either global or shared memory. Figure 13 shows the broadcasting capabilities of global and texture memory when several threads in a warp read the same 4 byte word. We can see, that global memory needs less transactions than texture memory.

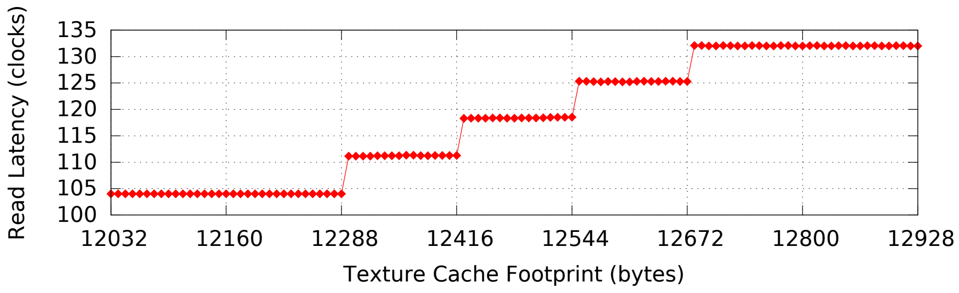


Figure 11: **Geforce GTX 680** - L1 texture cache latency plot indicating that cache size is 12KB with a cache line size of 128B. There are 4 cache sets with 24-way set associativity.

13

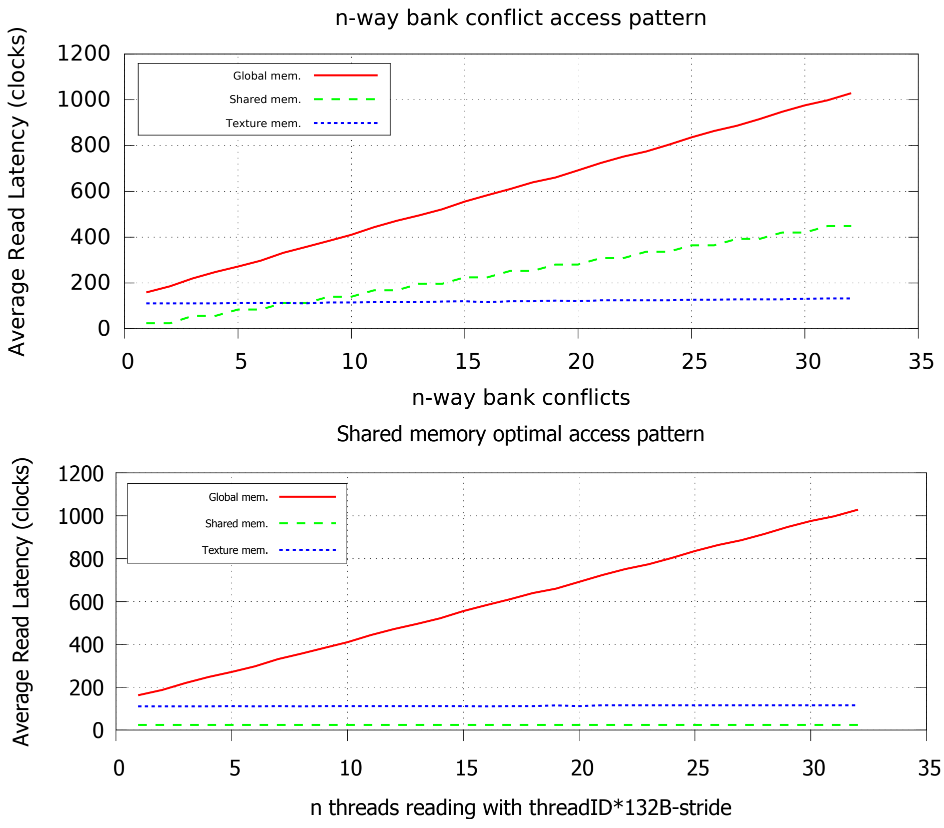


Figure 12: **Geforce GTX 680** - Latency plots of two access patterns which are either the worst case (top) or optimal (bottom) for shared memory compared with the latency of directly hitting in global or texture memory with the same pattern. Both patterns are worst case access patterns for global memory as access has to be serialized, though they hit in cache. Texture memory performs equally well in both cases.

5.2 Baseline

We chose the same baseline setup as in Section 4.2. Figures 14, 15, 16 and 17 show the runtime behavior (in ms) and GPU metrics (percentage) over all iterations for our test scenes with BVH nodes stored in global memory (left) and stored in texture memory (right).

5.3 BVH and node layouts

Tables 4 and 5 show a ranking of all BVH and node layout combinations which were accessed via global memory or texture memory. Thresholds for the SWST, TDFS and TBFS layouts were determined in the same manner as described in Section 4.3.

5.4 Profiling stats

Tables 6, 7, 8 and 9 illustrate the changes of the GPU metrics from the baseline measurements with DFS layout to the measurements with the best performing layout com-

GMem													
BVH lay.	node lay.	crytek-sponza			kitchen			hairball-glass			san-miguel		
		R	H	SZ	R	H	SZ	R	H	SZ	R	H	SZ
TDFS 0.6	Aos	581.7	86.7	94.3	1386.2	89.1	218.6	7504.9	60.5	948.8	2071.4	71.4	287.4
BFS	Aos	583.3	86.5	94.3	1364.6	89.0	218.6	7462.2	60.4	948.8	2131.1	70.7	287.6
TBFS 0.3	Aos	582.6	86.6	94.3	1371.8	89.1	218.6	7469.5	60.4	948.8	2142.8	70.8	287.7
vEB	Aos	582.5	86.6	94.3	1374.6	89.0	218.6	7469.4	60.6	948.6	2165.4	70.7	287.7
COL	Aos	582.5	86.7	94.3	1385.5	89.1	218.6	7539.5	60.6	949.1	2166.9	70.9	287.7
DFS	Aos	583.5	86.6	94.3	1394.9	89.0	218.6	7576.0	60.7	949.1	2205.3	70.6	287.9
SWST 0.5	Aos	582.2	86.8	94.3	1391.9	89.1	218.6	7638.8	60.8	949.3	2267.9	71.0	288.0
BFS	Soa32_24	581.0	85.1	94.1	1310.1	88.4	217.8	9099.2	55.3	950.3	2683.3	64.4	287.4
vEB	Soa32_24	583.9	85.3	94.2	1355.1	88.7	217.9	9059.4	55.7	950.1	2824.0	64.6	287.6
COL	Soa32_24	585.1	85.4	94.2	1335.2	88.8	217.9	9269.1	55.4	950.7	2859.4	64.9	287.7
DFS	Soa32_24	595.3	84.6	94.2	1357.9	88.3	217.9	9320.0	54.8	950.7	2932.9	63.3	288.2
BFS	Soa16_8	637.4	77.2	93.4	1346.1	81.7	213.4	10747.6	38.1	942.2	3270.6	48.3	285.3
vEB	Soa16_8	641.3	77.9	93.3	1364.4	83.4	215.3	10555.1	40.0	942.8	3376.6	48.6	286.1
COL	Soa16_8	651.3	77.8	93.2	1371.4	84.0	215.3	10780.1	39.6	943.4	3437.5	49.1	286.3
DFS	Soa16_8	680.1	75.8	93.6	1436.5	82.0	217.1	10969.3	38.3	943.0	3667.2	45.4	287.1

Table 4: **Ranking Geforce GTX 680** - BVH (global memory) and node layout combinations sorted ascending by average speedup over all scenes accessed via global memory. **R**untime in ms, **H**itrate in percent and **SZ** denotes the total amount of data transferred (in GB).

TMem													
BVH lay.	node lay.	crytek-sponza			kitchen			hairball-glass			san-miguel		
		R	H	SZ	R	H	SZ	R	H	SZ	R	H	SZ
TDFS 0.6	Aos	372.4	76.5	136.3	812.6	65.6	268.1	5369.4	59.8	1053.8	1300.4	61.0	337.3
BFS	Aos	373.1	76.4	136.3	807.7	65.6	268.1	5356.7	59.9	1053.8	1315.0	61.2	337.3
TBFS 0.2	Aos	373.1	76.5	136.3	810.2	65.7	268.1	5359.3	59.9	1053.8	1315.0	61.2	337.3
vEB	Aos	373.0	76.5	136.3	806.7	65.5	268.1	5357.3	59.8	1053.8	1326.4	61.1	337.3
COL	Aos	373.4	76.5	136.3	804.2	65.6	268.1	5386.1	59.8	1053.8	1334.8	61.0	337.3
SWST 0.4	Aos	373.8	76.5	136.3	805.1	65.5	268.1	5394.3	59.9	1053.8	1353.1	60.9	337.2
DFS	Aos	374.2	76.4	136.3	806.2	65.4	268.1	5394.9	59.8	1053.8	1356.4	60.9	337.3
BFS	Soa32_24	412.9	73.0	136.4	845.6	61.5	268.3	6868.8	56.6	1056.5	1877.0	56.4	337.3
vEB	Soa32_24	417.0	73.0	136.4	837.4	61.1	268.3	6839.6	56.4	1056.6	1955.8	56.2	337.4
COL	Soa32_24	417.1	73.0	136.4	852.9	61.2	268.3	6956.5	56.3	1056.5	1978.8	56.2	337.4
DFS	Soa32_24	423.4	72.7	136.4	852.7	60.7	268.3	6971.4	56.3	1056.5	2023.4	55.9	337.3
BFS	Soa16_8	497.0	60.2	135.7	988.0	43.9	264.8	9570.8	36.7	1048.7	2837.8	34.7	335.0
vEB	Soa16_8	495.3	61.1	135.6	981.6	43.9	266.5	9261.5	37.8	1048.9	2932.9	35.2	335.3
COL	Soa16_8	506.1	61.2	135.6	973.1	44.2	266.6	9515.5	37.7	1048.6	2999.3	35.3	335.4
DFS	Soa16_8	535.7	60.5	135.7	1042.8	42.9	267.6	9663.3	38.4	1049.5	3229.9	35.1	335.9

Table 5: **Ranking Geforce GTX 680** - BVH (texture memory) and node layout combinations sorted ascending by average speedup over all scenes accessed via global memory. **R**untime in ms, **H**itrate in percent and **SZ** denotes the total amount of data transferred (in GB).

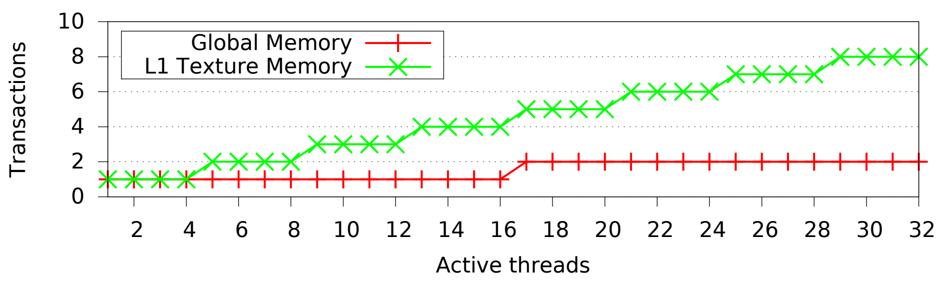


Figure 13: **Geforce GTX 680** - Number of transactions for a broadcast in global and texture memory for an increasing number of threads in a warp.

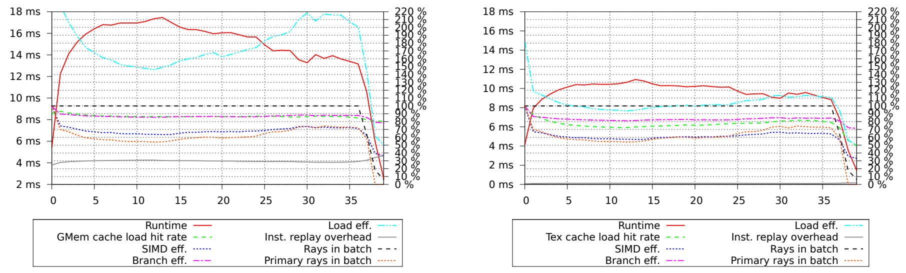


Figure 14: **Geforce GTX 680 - Crytek Sponza** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

bination. Cells with four values separated by slashes represent minimum, average, maximum and average absolute deviation of the respective metric over profiled iterations.

5.5 Best performing layout

For both, storing nodes in global and texture memory the best performing BVH layout is the TDFS layout with a threshold of 0.6 in combination with the AoS node layout. Agreeing with [ALK12] storing nodes in texture memory is most beneficial for Kepler GPUs. Comparing the runtime of the best layout combinations in texture and global memory, we get the same qualitative behavior as for the Fermi GPU in Section 4.4. In global memory we have achieved runtime reduction by 1% – 6%. In texture memory, we gained $\approx 30.0\%$ – 40% runtime reduction compared to the baseline in global memory. Also accessing the baseline in texture memory, an improvement of only 0.5% – 4.0% was observable for TDFS. We attribute the smaller amount of data transferred when using global memory to superior broadcast capabilities (see Section 5.1).

5.6 Tesla K20C Addendum

According to [NVIa] NVIDIA Kepler GPUs with compute capability 3.5 feature a 48KB read-only data cache per SMX, which is the same as the texture cache. In order to see the effects of a much larger texture cache we also performed our experiments with a Tesla K20c GPU. Figures 22, 23, 24 and 25 show results for our baseline layouts accessing geometry and nodes via the read-only data cache compared with the baseline results for the GTX 680. ECC has been turned on for these experiments. On average

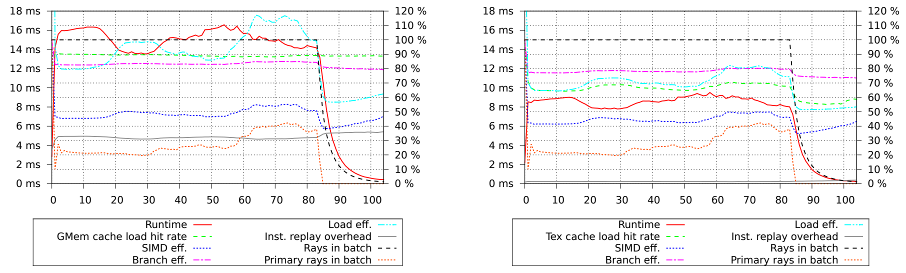


Figure 15: **Geforce GTX 680** - **Kitchen** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

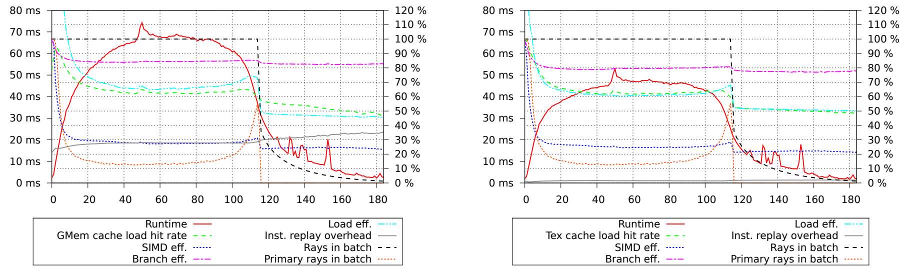


Figure 16: **Geforce GTX 680** - **Hairball** - **Glass** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

runtime is only about 8% better despite the 4 times larger cache and the 62.5% higher number of cores. In fact there is also a slight hit of 3 percentage points to cache hit rate. Using the microbenchmarking code from [WPSAM10] we were able to deduce that the size of the read-only data cache is seemingly only 12KB and not 48KB (assuming there is no mistake on our side), which might explain why there is no improvement in cache hit rate. Turning ECC off resulted in less than one percent runtime improvement. Thus bandwidth from device memory to L2 cache does not seem to be the main bottleneck.

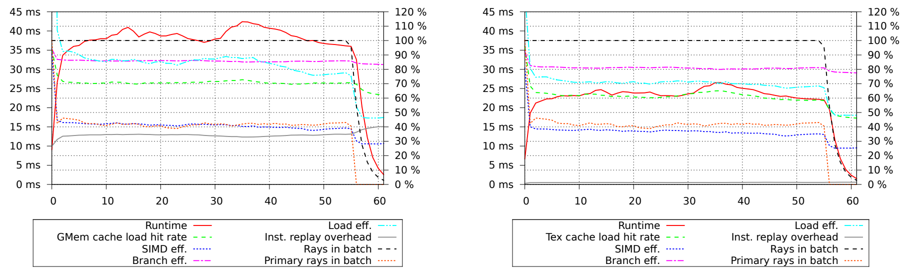


Figure 17: **Geforce GTX 680 - San Miguel** - Baseline trace kernel profiling graph. Nodes are either stored in global (left) or texture memory (right).

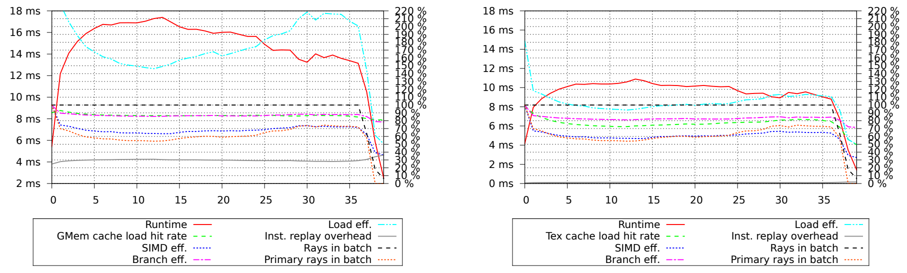


Figure 18: **Geforce GTX 680 - Crytek Sponza** - Trace kernel profiling graph for the best layout in global memory (left, TDFS 0.6, AoS) and texture memory (right, TDFS 0.6, AoS).

Sponza, GMem, TDFS 0.6, AoS	
Runtime (ms)	2.5 (−0.1) / 14.4 (−0.1) / 17.4 (−0.1) / 2.2 (+0.0)
Global load transact./req.	2.0 (+0.0) / 5.3 (+0.0) / 9.6 (+0.0) / 0.7 (+0.0)
L2 load hit rate (%)	81.0 (+0.5) / 86.7 (+0.1) / 93.0 (−0.1) / 1.1 (+0.0)
L2 load size (GB)	55.6 (−0.0)
L2 load bandwidth (GB/s)	72.5 (+0.1) / 101.9 (+0.4) / 112.5 (+1.0) / 3.3 (+0.1)
L2 load size (GB)	61.6 (−0.0)
Dev. mem. load size (GB)	11.2 (−0.0)
Inst. replay overhead (%)	24.8 (+0.0) / 29.7 (−0.1) / 36.2 (−0.3) / 1.1 (+0.0)

Sponza, TMem, TDFS 0.6, AoS	
Runtime (ms)	1.4 (−0.1) / 9.2 (−0.1) / 10.9 (+0.0) / 1.2 (+0.0)
Tex cache hit rate (%)	49.4 (+0.1) / 76.5 (+0.1) / 99.4 (+0.0) / 4.4 (−0.1)
Tex load size (GB)	111.2 (+0.0)
Tex load bandwidth (GB/s)	231.0 (+4.8) / 299.4 (+1.5) / 369.3 (+0.7) / 9.3 (−0.4)
Tex←L2 load hit rate (%)	62.1 (+0.2) / 69.5 (+0.0) / 95.7 (+0.1) / 2.8 (−0.1)
Tex←L2 load size (GB)	25.2 (−0.1)
Tex←L2 load bandwidth (GB/s)	2.1 (−0.1) / 68.3 (+0.1) / 115.4 (+2.2) / 11.1 (+0.1)
L2 load size (GB)	26.9 (−0.1)
Dev. mem. load size (GB)	9.4 (−0.0)
Inst. replay overhead (%)	0.7 (+0.0) / 1.3 (+0.0) / 2.0 (+0.0) / 0.1 (+0.0)

Table 6: **Geforce GTX 680 - Crytek Sponza** - Trace kernel profiling data totals for TDFS 0.6, AoS in global memory (top) and in texture memory (bottom).

Kitchen, GMem, TDFS 0.6, AoS	
Runtime (ms)	0.4 (+0.0) / 12.5 (−0.1) / 16.5 (−0.1) / 4.0 (+0.0)
Global load transact./req.	2.1 (+0.0) / 7.5 (+0.0) / 10.9 (+0.0) / 0.9 (+0.0)
L2 load hit rate (%)	82.5 (+0.0) / 89.1 (+0.1) / 90.6 (+0.2) / 0.6 (+0.1)
L2 load size (GB)	134.8 (−0.0)
L2 load bandwidth (GB/s)	73.7 (+2.1) / 105.7 (+0.6) / 112.8 (+0.4) / 4.5 (+0.1)
L2 load size (GB)	152.2 (−0.0)
Dev. mem. load size (GB)	21.9 (−0.2)
Inst. replay overhead (%)	23.4 (+0.0) / 32.5 (−0.1) / 36.2 (+0.0) / 1.2 (+0.0)

Kitchen, TMem, TDFS 0.6, AoS	
Runtime (ms)	0.2 (+0.0) / 7.2 (+0.1) / 9.6 (+0.1) / 2.3 (+0.0)
Tex cache hit rate (%)	55.2 (+0.3) / 65.6 (+0.2) / 100.0 (+0.0) / 3.7 (+0.0)
Tex load size (GB)	217.5 (−0.0)
Tex load bandwidth (GB/s)	188.6 (+3.8) / 281.4 (−2.2) / 356.1 (−6.5) / 12.0 (−0.6)
Tex←L2 load hit rate (%)	71.9 (−0.6) / 78.3 (+0.1) / 80.7 (−0.1) / 1.4 (+0.0)
Tex←L2 load size (GB)	70.8 (−0.4)
Tex←L2 load bandwidth (GB/s)	0.1 (+0.0) / 94.8 (−1.2) / 122.2 (−1.3) / 8.5 (−0.2)
L2 load size (GB)	75.3 (−0.4)
Dev. mem. load size (GB)	18.9 (−0.2)
Inst. replay overhead (%)	1.2 (+0.0) / 1.6 (+0.0) / 2.5 (+0.0) / 0.2 (+0.0)

Table 7: **Geforce GTX 680** - **Kitchen** - Trace kernel profiling data totals for TDFS 0.6, AoS in global memory (top) and in texture memory (bottom).

Hairball - Glass, GMem, TDFS 0.6, AoS	
Runtime (ms)	2.3 (+0.1) / 39.6 (−0.4) / 73.2 (−1.1) / 23.8 (−0.3)
Global load transact./req.	2.3 (+0.0) / 5.1 (+0.0) / 6.2 (+0.0) / 0.7 (+0.0)
L2 load hit rate (%)	45.8 (−0.4) / 60.5 (−0.2) / 94.0 (−0.1) / 6.6 (+0.0)
L2 load size (GB)	481.0 (+0.2)
L2 load bandwidth (GB/s)	15.4 (+0.1) / 63.4 (+0.6) / 84.3 (−0.5) / 13.1 (+0.1)
L2 load size (GB)	573.5 (+0.2)
Dev. mem. load size (GB)	258.9 (+1.4)
Inst. replay overhead (%)	21.1 (+0.0) / 29.3 (−0.1) / 35.4 (−0.3) / 2.4 (−0.1)

hairball-glass, TMem, TDFS 0.6, AoS	
Runtime (ms)	1.7 (+0.0) / 28.2 (−0.1) / 52.7 (−0.2) / 16.6 (−0.1)
Tex cache hit rate (%)	48.3 (+0.1) / 59.8 (+0.0) / 100.0 (+0.0) / 7.0 (−0.1)
Tex load size (GB)	777.5 (+0.0)
Tex load bandwidth (GB/s)	22.2 (+0.0) / 132.2 (+0.6) / 355.5 (+0.2) / 37.9 (−0.2)
Tex←L2 load hit rate (%)	23.6 (−0.2) / 31.6 (−0.4) / 92.1 (−0.4) / 5.3 (+0.0)
Tex←L2 load size (GB)	283.2 (+0.1)
Tex←L2 load bandwidth (GB/s)	0.1 (+0.0) / 48.6 (+0.3) / 62.1 (+0.4) / 10.7 (+0.0)
L2 load size (GB)	305.0 (+0.1)
Dev. mem. load size (GB)	211.2 (+1.2)
Inst. replay overhead (%)	0.8 (+0.0) / 1.7 (+0.0) / 2.1 (+0.0) / 0.2 (+0.0)

Table 8: **Geforce GTX 680** - **Hairball - Glass** - Trace kernel profiling data totals for TDFS 0.6, AoS in global memory (top) and in texture memory (bottom).

San Miguel, GMem, TDFS 0.6, AoS	
Runtime (ms)	2.3 (−0.2) / 33.3 (−2.1) / 40.2 (−2.2) / 5.1 (−0.4)
Global load transact./req.	3.2 (+0.0) / 6.2 (+0.0) / 7.6 (+0.0) / 0.3 (+0.0)
L2 load hit rate (%)	63.3 (+0.8) / 71.4 (+0.8) / 94.0 (+0.0) / 1.4 (−0.1)
L2 load size (GB)	186.1 (−0.3)
L2 load bandwidth (GB/s)	70.8 (+5.3) / 94.5 (+5.6) / 100.9 (+1.1) / 2.6 (+0.0)
L2 load size (GB)	204.7 (−0.3)
Dev. mem. load size (GB)	69.4 (−1.5)
Inst. replay overhead (%)	26.6 (+0.1) / 33.8 (−0.7) / 39.2 (−1.1) / 0.9 (−0.1)

San Miguel, TMem, TDFS 0.6, AoS	
Runtime (ms)	1.4 (−0.1) / 20.9 (−0.9) / 25.7 (−0.8) / 3.2 (−0.1)
Tex cache hit rate (%)	46.2 (+0.1) / 61.0 (+0.1) / 96.0 (+0.0) / 3.3 (−0.1)
Tex load size (GB)	262.3 (+0.0)
Tex load bandwidth (GB/s)	135.8 (+12.0) / 201.1 (+8.4) / 337.3 (−1.1) / 8.9 (−0.6)
Tex←L2 load hit rate (%)	43.2 (+0.4) / 46.2 (+0.2) / 85.7 (+0.0) / 1.4 (−0.1)
Tex←L2 load size (GB)	99.8 (−0.2)
Tex←L2 load bandwidth (GB/s)	13.2 (+0.0) / 76.7 (+3.2) / 90.1 (+5.5) / 4.0 (+0.4)
L2 load size (GB)	103.9 (−0.2)
Dev. mem. load size (GB)	57.8 (−0.4)
Inst. replay overhead (%)	0.8 (+0.0) / 1.4 (+0.0) / 1.6 (+0.0) / 0.1 (+0.0)

Table 9: **Geforce GTX 680 - San Miguel** - Trace kernel profiling data totals for TDFS 0.6, AoS in global memory (top) and in texture memory (bottom).

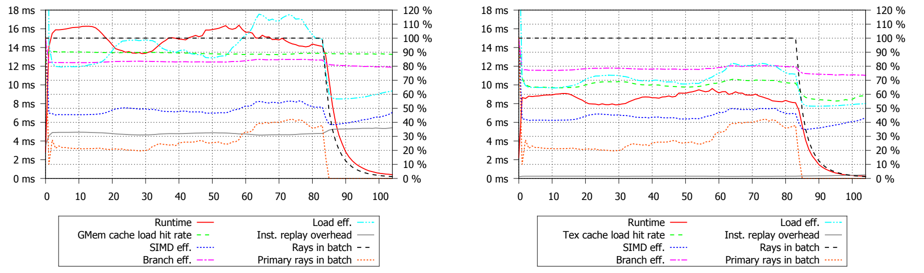


Figure 19: **Geforce GTX 680 - Kitchen** - Trace kernel profiling graph for the best layout in global memory (left, TDFS 0.6, AoS) and texture memory (right, TDFS 0.6, AoS).

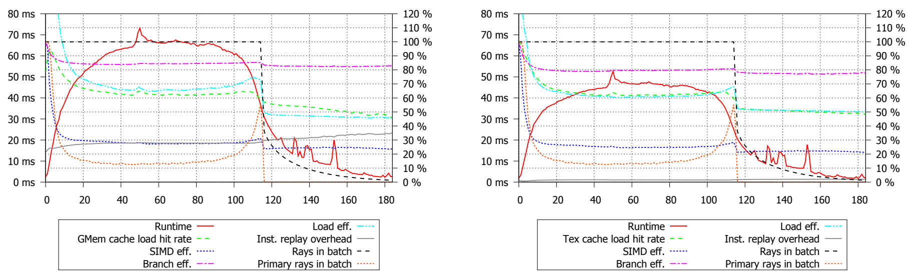


Figure 20: **Geforce GTX 680 - Hairball - Glass** - Trace kernel profiling graph for the best layout in global memory (left, TDFS 0.6, AoS) and texture memory (right, TDFS 0.6, AoS).

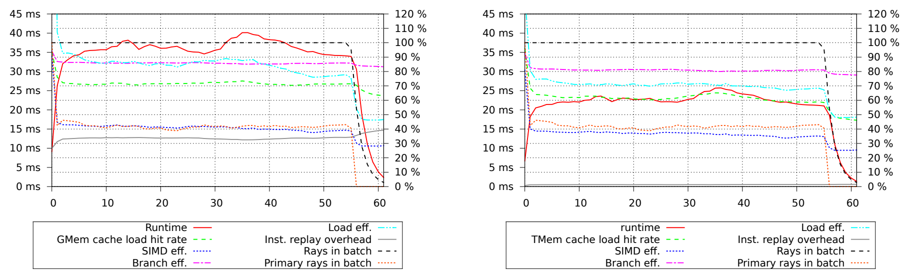


Figure 21: **Geforce GTX 680 - San Miguel** - Trace kernel profiling graph for the best layout in global memory (left, TDFS 0.6, AoS) and texture memory (right, TDFS 0.6, AoS).

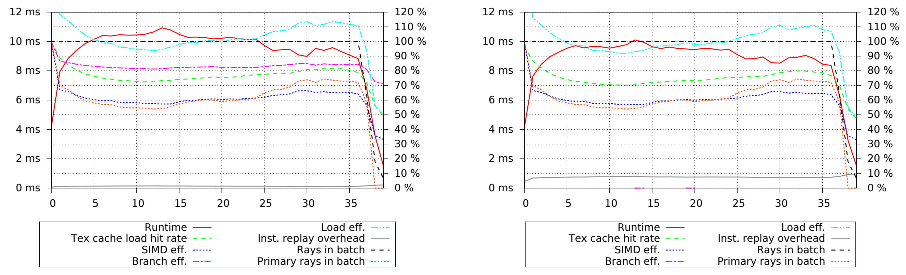


Figure 22: **Tesla K20c/Geforce GTX 680 - Crytek Sponza** - Trace kernel profiling graph for the best layout in global memory (left, TDFS 0.6, AoS) and texture memory (right, TDFS 0.6, AoS).

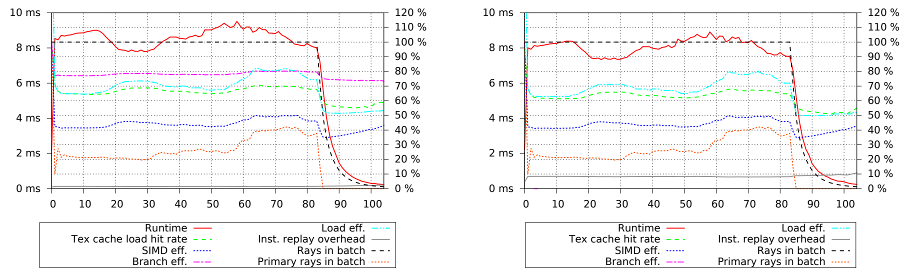


Figure 23: **Tesla K20c/Geforce GTX 680 - Kitchen** - Baseline trace kernel profiling graphs for the GTX 680 (left) and Tesla K20c (right). The GTX 680 accesses nodes via texture memory, while the Tesla K20c accesses nodes via the read-only data cache.

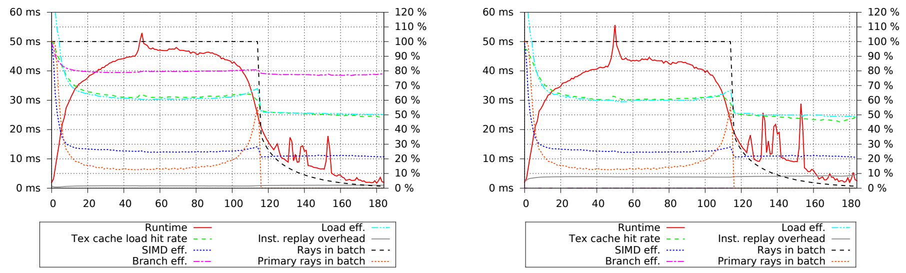


Figure 24: **Tesla K20c/Geforce GTX 680 - Hairball - Glass** - Baseline trace kernel profiling graphs for the GTX 680 (left) and Tesla K20c (right). The GTX 680 accesses nodes via texture memory, while the Tesla K20c accesses nodes via the read-only data cache.

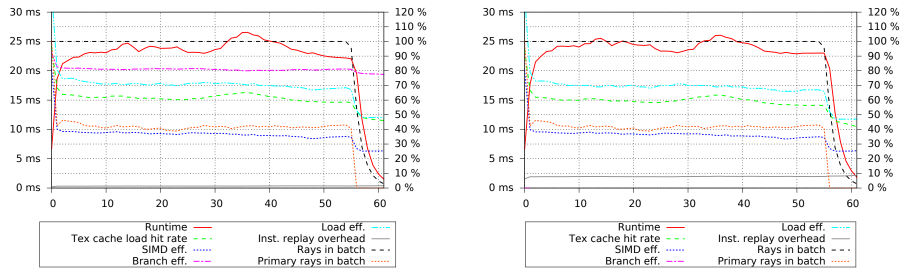


Figure 25: **Tesla K20c/Geforce GTX 680 - San Miguel** - Baseline trace kernel profiling graphs for the GTX 680 (left) and Tesla K20c (right). The GTX 680 accesses nodes via texture memory, while the Tesla K20c accesses nodes via the read-only data cache.

6 Overall Comparison

An overall observation we can make is, that the node layout has the largest impact on performance for both GPU architectures. The AoS layout performed best in both memory areas, except for Fermi and global memory, where SoA32_24 performed best. Our treelet based layout managed to achieve the best performance gains for both architectures though they are only moderate. On average the common DFS layout performed worst for all node layouts in both memory areas and architectures. Excluding layouts that use statistics the equally simple to construct BFS layout on average performed best and similar to the TDFS layout.

Acknowledgments

The work of S. Widmer and D. Wodniok is supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt. Hairball scene courtesy of Samuli Laine. Crytek-sponza scene courtesy of Frank Meinl. San-miguel scene courtesy of Guillermo M. Leal Llaguno.

References

[AL09] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on GPUs. In *Proc. HPG*, 2009.

[ALK12] Timo Aila, Samuli Laine, and Tero Karras. Understanding the efficiency of ray traversal on GPUs – Kepler and Fermi addendum. Technical Report NVR-2012-02, 2012.

[NVIa] NVIDIA. CUDA C Programming Guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

[NVIb] NVIDIA. Cuda Profiling Tools Interface. `developer.nvidia.com/cuda-profiling-tools-interface`.

[Sm] Crytek Sponza model. `http://www.crytek.com/cryengine/cryengine3/downloads`.

[WPSAM10] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos. Demystifying GPU microarchitecture through microbenchmarking. In *Proc. ISPASS*, 2010.

[WSWG13] D. Wodniok, A. Schulz, S. Widmer, and M. Goesele. Analysis of cache behavior and performance of different BVH memory layouts for tracing incoherent rays. In *Proc. EGPGV*, 2013.