High-Performance Graphics (2019)
T. Foley and M. Steinberger (Editors)

Mach-RT: A Many Chip Architecture for Ray Tracing

E. Vasiou, K. Shkurko, E. Brunvand, and C. Yuksel
School of Computing, University of Utah

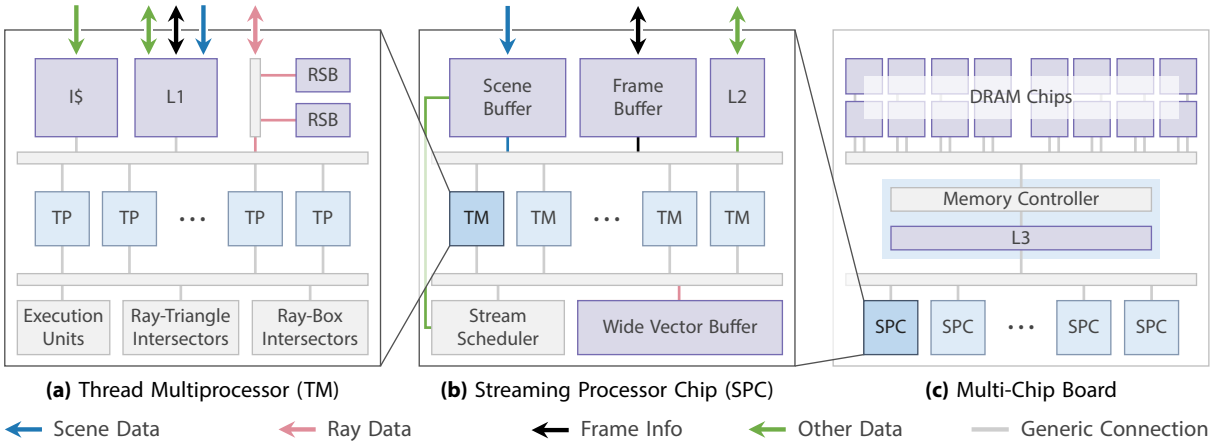

Figure 1: *Overview of Mach-RT – our multiple chip architecture for high-performance ray tracing. We propose a solution that integrates many chips on a board sharing main (scene) memory, while keeping all rays stored in on-chip memory distributed across those chips. Each chip contains a number of Thread Multiprocessors which in turn are comprised of many small Thread Processors operating in parallel.*

Abstract

*We propose an unconventional solution to high-performance ray tracing that combines a ray ordering scheme that minimizes access to the scene data with a large on-chip buffer acting as near-compute storage that is spread over multiple chips. We demonstrate the effectiveness of our approach by introducing Mach-RT (**Ma**ny **ch**ip - **R**ay **T**racing), a new hardware architecture for accelerating ray tracing. Extending the concept of dual streaming, we optimize the main memory accesses to a level that allows the same memory system to service multiple processor chips at the same time. While a multiple chip solution might seem to imply increased energy consumption as well, because of the reduced memory traffic we are able to demonstrate, performance increases while maintaining reasonable energy usage compared to academic and commercial architectures.*

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Graphics processors;*** • ***Computer systems org.*** → *Parallel architectures;*

1. Introduction

The tremendous rendering performance of commercial GPUs is due to two important properties of the rasterization algorithm: its high level of parallelism, which provides a great fit for GPUs with many parallel cores, and the perfectly predictable manner in which rasterization accesses the scene data. The second property is as important as the first one, if not more so, since it allows GPUs to completely hide memory latency. Indeed, memory operations are the bottleneck of most computing problems today and rasterization circumvents this performance penalty by streaming data from memory in a sequential order.

Ray tracing, on the other hand, does not provide such a predictable streaming access to the scene data. Instead, the spatial partitioning structures used for accelerating ray tracing lead to effectively random, non-sequential accesses to the scene data that prevent prefetching. Therefore, most recent work on accelerating ray tracing concentrates on improving memory behavior, by coalescing memory accesses [SKKB09; KSBD10; AL09] or compressing scene data [LV16; YKL17; BWWÁ18]. Nonetheless, closing the performance gap between rasterization and ray tracing requires fundamentally restructuring ray tracing such that it can produce the predictable streaming access to scene data that rasterization enjoys. The Dual Streaming method [SGK*17] is one such method that restructures the memory access pattern of ray tracing into two perfectly predictable streams: a scene stream and a ray stream.

DOI: 10.2312/hpg.20191188

While the scene stream reduces the scene data transferred from main memory to its absolute minimum, the additional ray stream dominates the memory traffic and hinders the performance of this approach.

In this paper, we propose an unconventional solution to the memory access problem. We begin with the ray ordering scheme of the dual streaming approach that minimizes the memory traffic for the scene data. We then eliminate the ray stream by storing ray data in on-chip buffers. Unfortunately, such buffers on a single chip cannot be large enough to store all rays needed for achieving high rendering performance. Indeed, a large buffer would lead to overheads not only in area and manufacturing costs, but also in performance, as each access to the allocated memory would be progressively more costly in latency and energy the larger the memory block is. Our solution is spreading this cost over multiple chips, all of which are connected to the same main memory system. This leads to much smaller chips that can be manufactured cost-effectively, and buffers on these chips that can operate with reduced latency and energy consumption. Adding an off-chip L3 cache shared among all chips reduces the workload on the main memory, allowing a single memory system to effectively service multiple chips.

We introduce Mach-RT (**Ma**ny **ch**ip - **R**ay **T**racing), a new hardware architecture for accelerating ray tracing, as a proof of concept to evaluate our approach. An important advantage of Mach-RT is that it uses general-purpose compute cores that can execute diverging instructions Moreover, our system is entirely programmable and the test renderer is written entirely in standard C++ with custom instructions for ray-triangle and ray-box intersections. One inspiration for this approach are the so-called processor in memory (PIM) or near-data processing (NDP) architectures that focus on large memories while moving processing closer to that memory. By focusing on moving the processing closer to the required memory, data movement can be minimized and the entire system can be made more efficient.

One might falsely expect a multi-chip architecture to consume significantly more energy than a single processor chip. On the contrary, we observe in our results that this assumption is incorrect and indeed a multi-chip architecture can improve performance while reducing the total energy consumption. This is an important contribution as in fact, energy consumption is a growing concern for real-time rendering, especially considering recent commercial efforts on cloud computing for online gaming.

We demonstrate the capabilities of the new Mach-RT architecture by comparing against academic architectures such as STRaTA [KSS*13] and Dual Streaming [SGK*17] and to highly-optimized ray tracing software/hardware products from the industry, including Intel's Embree [WWB*14] and Microsoft's DXR [WHSB18] running on NVIDIA's recent Turing architecture with hardware support for ray tracing [NVI18], showing that our system can provide substantially improved performance.

2. Background

It is not a new observation that accessing memory is a primary concern when accelerating ray tracing. One approach has been to propose specialized fixed-function traversal units integrated into processors [WBS06; KKK12; LSL*13; NKK*14]. However, fixed-function units do not necessarily optimize data requests from the memory system hierarchy.

Some researchers have focused on reordering how ray and scene data is accessed by utilizing a form of streaming [WGBK07; GR08; Bik12; BA14], or collecting ray and scene data requests through software means [GDS*08; KJJ*09; SKKB09; ALK12].

Memory traffic can be reduced by exploiting the BVH structure [NFLM07; AK10] or by modifying it to get better performance [MB18; Tsa09]. Even in cases where the proposed architecture is designed to keep rays on chip [KSS*13; KSS*15], it is still limited in the number of rays it can process simultaneously. While a pool of work is focused on compressing the acceleration structure [Kee14; LV16; YKL17; LMSS18] even the most aggressive of those schemes cannot realistically enable storing all rays on chip as rendering a single frame can require tens of millions of rays.

Dual Streaming Algorithm Our proposed system is inspired by the Dual Streaming architecture's approach to restructuring and minimizing the scene data stream [SGK*17]. In that approach the scene stream consists of BVH treelets [AK10], each containing both nodes and geometry (e.g. triangles). Each treelet fits into several DRAM row buffers for efficient streaming from main memory, and can be processed separately in parallel.

Dual Streaming processes rays as wavefronts, each of which contains all rays in flight at the same depth, making up the ray stream. The ray stream is split into ray queues, one per treelet, and stores only basic ray data (i.e. origin and direction), but not the traversal stack per ray. During traversal, the ray queue associated with a given treelet is drained fully before a new treelet is acquired.

Rays visit treelets in a fixed traversal order, strictly from a parent node into its children without returning up. This ensures that treelets are loaded from the main memory at most once per ray wavefront. Both scene treelet data and its corresponding ray queue are prefetched onto the chip ahead of traversal. If a treelet has no rays for processing, then the entire scene subtree is skipped. Traversal continues until all ray queues are empty. When a ray encounters an exit from the current treelet into one of its children, the ray is duplicated into the child queue and continues traversing the current treelet until all exit points are found.

Dual Streaming relies on a single shared hit record for each set of ray duplicates, which requires atomic updates whenever any ray in the set finds a hit. One disadvantage of this approach is that using the hit record for early ray termination becomes non-trivial because a duplicate could be traversing a different treelet with a closer hit.

3. Mach-RT: A Multi-Chip Architecture

It is well known in the computer architecture community that memory traffic, especially DRAM traffic, is the largest contributor to latency and energy increases in computing systems [WM95; BABD00; BKC14; KSS*15]. That is specifically the case for ray tracing as well [VSM*18]. Dual Streaming minimizes DRAM traffic for scene data at the cost of adding ray data traffic, which is significantly larger. Therefore, if one can eliminate the ray data traffic,

DRAM could handle the predictable scene data requests efficiently. One way to achieve this is by storing ray data in on-chip buffers. While it is not reasonable to expect that a single processor chip would have enough storage for all ray data, multiple chips can collectively store all rays.

One might think that connecting multiple chips to a single DRAM might be disastrous for performance. We find the contrary because our proposed architecture generates minimal scene traffic and absolutely no ray traffic to DRAM. On a fabrication level, it is easy to imagine that building a board that contains multiple Mach-RT chips is no different from a traditional circuit board. Yet, one could also imagine that multiple chips can be assembled on a silicon interposer with the off-chip and main memory also integrated in a tightly coupled system [TZ14; USS*17]. Bare die assembled on an interposer substrate would result in a denser, higher speed system with a smaller footprint and higher bandwidth links between the various components.

Although we focus on solving the memory bottlenecks associated with ray tracing with our architecture, the system we have designed, unlike the above approaches, resembles more a contemporary GPU that is fully programmable with only a handful of specialized units such as the ray-box/triangle intersection pipelines and stream scheduler. All chips utilize the Single Program Multiple Data (SPMD) programming paradigm and since each chip is independent they can be programmed to carry out differing tasks. Additionally, Mach-RT does not assume a fixed function rendering pipeline. The rendering programs are written in C++, making it accessible even to those without knowledge of intricate specialized languages and assembly.

3.1. Chip Architecture

Our architecture is designed with multiple chips on a board, all connected to main memory through an off-chip L3 cache. Each chip is assigned a portion of the final image to be computed in parallel. We assume a homogeneous model for all chips across the board, as shown in Figure 1. Each Streaming Processor Chip (SPC) consists of a large number of Thread Multiprocessors (TMs), each of which contains a number of Thread Processors (TPs). TPs are grouped in such a way to allow for shared utilization of energy and area expensive units. Chips contain an L2 data cache, used primarily for shading data. The scene buffer and the stream scheduler are implemented similarly to the Dual Streaming architecture. SPCs also contain an on-chip wide vector storage buffer that stores unique ray data for the entire wavefront.

The on-chip frame buffer stores the hit records and other data necessary to generate new rays in each wavefront. Unlike Dual Streaming that must utilize a Global Hit Record Updater to atomically update hit records stored in DRAM, our frame buffer processes hit records locally. Since chips are assigned a non-overlapping set of pixels from the entire image hit records shared by ray duplicates remain on chip and thus always available for updates and queries.This enables checking the current closest hit for each ray before traversing it through a treelet. The frame buffer also stores the colors for pixels assigned to the SPC.

Table 1: *The default configuration of our Mach-RT architecture.*

Board		**TM Configuration**	
Clock Rate	2.0 GHz	TMs / chip	32
DRAM Memory	4 GB GDDR5	TPs / TM	16
L3 Cache	64 MB	L1 Cache	32 KB, 8 banks
Total Threads	512 / chip	Ray Staging Buffer	2×2KB
Chips	1 - 8		
On-Chip Memory (per Chip)		**Area per Chip**	
L2 Cache	512 KB, 32 banks	Scheduler	negligible
Scene Buffer	4 MB	Caches / Buffers	542 mm^2
Frame Buffer	6.5 MB	Compute	52 mm^2
Wide Vector Buffer	6.75 MB	Total	594 mm^2

Even though scene traffic is minimized per chip, because multiple chips need to access the scene data, read requests can stress the memory interface. To ease the pressure on DRAM, we implement an off-chip L3 cache that facilitates communication between all the chips on the board and the main memory.The scene data transfer uses a small portion of the total available bandwidth, so a simple SRAM cache is sufficient to manage traffic to all chips. A single off-chip L3 cache also allows to interface with DRAM via a single memory controller, simplifying the memory hierarchy.

3.2. Mach-RT Configuration

We use Cacti 7.0 [BKM*17] for the area and energy estimates of all our SRAM buffers and on-chip caches. Computation units are estimated using Synopsys DesignWare/Design Compiler at the 65 nm process technology. Table 1 shows the configuration and the area estimates for our multi-chip system and each chip within. The memory sizes and area estimates assume that there are 8 chips on a board. For comparisons, single large chip instances of each architecture are configured such that their compute and memory capacities match the values of our proposed architecture.

Each SPC in our simulations runs at 2.0 GHz and includes 512 Thread Processors (TPs), split evenly between 32 Thread Multiprocessors (TMs) (see Figure 1). TPs contain their own floating point add and integer units, and rely on their own program counter for control flow. Each TP has 32 registers for a total of 128B scratchpad memory. We provision for 4 MB of on-chip memory for the Scene Buffer handling treelets 64 KB in size. TPs in each TM share 1 ray-box and 2 ray-triangle intersection pipelines. Other shared large-area units consist of L1 data cache, instruction cache, Ray Staging Buffer and large execution units such as floating point division.

Besides the scene buffer, on-chip memory is split between the L2 cache, the frame buffer and the wide vector buffer. The direct-mapped L2 cache holds 512 KB and is used strictly for shading data. While all other units remain constant in size when more chips are placed on a board, the wide vector buffer (ray storage) and the frame buffer can be smaller in size because the workload per chip is naturally reduced. We allocate 6.5 MB to the frame buffer to store the ray hit records and pixel colors. We assume each ray holds the most basic information 28B in size. Because rays can be duplicated during traversal (Section 2), we store each duplicate as 4B indexes that reference unique ray data. We provision for two rays per pixel (shadow and non-shadow) per wavefront with a generous duplication rate of 20. For the given image resolution, this configuration

requires approximately 54 MB of ray storage distributed across all chips on the Mach-RT multi-chip board.

The off-chip direct-mapped L3 cache is also modeled as a basic SRAM using Cacti's off-chip memory options. The access latency is estimated by adjusting Cacti output to account for the wire delay between the L3 cache and the on-board chips assuming all chips are equidistant from the L3. We test several L3 sizes, choosing the 64 MB configuration shown in Table 1 as the primary one. The multi-chip board has 4 GB of GDDR5 DRAM configured with 16 32-bit channels for a total of 512 GB/s maximum bandwidth.

4. Experimentation and Results

Our primary goal in this work is to show that a novel multi-chip architecture, such as Mach-RT, can enable ray tracing to exhibit completely streamed memory access behavior, similar to rasterization. The result is a high-performance ray tracing engine that can efficiently share memory resources across multiple chips.

To demonstrate this, we evaluate our proposed hardware architecture on a set of test scenes using a cycle-accurate simulator [SGE*18]. The simulator integrates a detailed memory simulator [CBS*12] that accurately models DRAM, a vital component of our experiments that ensures proper system evaluation. We use a number of test scenes with varying triangle counts that represent different types of rendering effort, shown in Figure 2.

Each scene is rendered using path tracing with the maximum ray depth of five at the image resolution of 1024×1024. This workload produces a set of rays which access scene data incoherently, stressing the memory systems of the dedicated ray tracing architectures. To keep focus on traversal and intersection performance, we rely on simple Lambertian shading.

4.1. Single Large Chip

To probe the limits of our architecture we evaluate a single large chip version of our proposed architecture and compare it with the Dual Streaming architecture [SGK*17] configured to use a large number of threads on a single large chip to understand the effects of keeping ray data on chip. Although both configurations are unrealistic, because they require much more on-chip memory than is plausible to allocate on a single chip, this comparison is useful to understand the best-case behavior. Then we can evaluate the impact of having multiple, more reasonable, chips accessing memory simultaneously.

For all tested scenes, our proposed Mach-RT as a large chip outperforms the large Dual Streaming chip for all thread counts. Note that while the Dual Streaming performance tapers off for larger thread counts, our Mach-RT architecture keeps improving. For example, at 4096 threads, the Mach-RT single large chip renders the frame in half the time: 8 instead of 16 ms/frame for Dragon Sponza and 16 instead of 35 ms/frame for San Miguel. Similarly, the energy costs are lower for our Mach-RT system. While Dual Streaming is bound by DRAM energy, our proposed architecture shows a trade off between DRAM and on-chip energies for larger thread counts Because the chip has more of the required data available, it can utilize its resources more effectively. However, such configurations require chips with unrealistically-large on-chip buffers: over 100 MB, most of which would be allocated for ray storage.

4.2. Multiple-Chip Ray Tracing

The crux of the Mach-RT architecture is that it distributes the unrealistically-large on-chip memory between multiple chips without incurring a drop in performance. This is primarily due to the ability to predictably stream scene data from main memory while keeping all ray traffic on-chip.

We compare our multi-chip design to a single unrealistically-large Dual Streaming chip [SGK*17] and STRaTA [KSS*13; KSS*15] chip with hardware resources scaled to a combination of all of our chips. While the energy requirements for our proposed system can be slightly higher for some scenes, such as Vegetation and San Miguel, for all tested scenes and chip/thread counts, the issue rate of our cores is higher, averaging at high 60% for Mach-RT and 40% for Dual Streaming and STRaTA. Except for Vegetation, for which STRaTA is considerably faster because it can utilize early ray termination, our method renders frames faster for all tested scenes at all chip/thread counts. Most interestingly, while our method keeps improving past four chips (2048 threads), STRaTA begins to slow down at those thread counts. Notably at 2048 threads for Dragon Sponza is at 29 ms/frame it slows down to 67 ms/frame for twice the size of the operating chip (scaled for both threads and memory) at 4096 threads.

The comparison with STRaTA is especially interesting since it is an architecture optimized to reduce DRAM energy while keeping some rays in specialized on-chip memory. Despite STRaTA being provisioned to hold an increasing number of rays as the chip scales, it requests a lot of data from main memory, transferring a larger number of cache lines Our Mach-RT multi-chip architecture transfers at most 180 million cache-lines at 4096 threads (8 chips) for San Miguel, while STRaTA is close to 542 million. Similarly, because STRaTA requests more data from DRAM, it consumes more energy than the Mach-RT architecture. The total energy decreases until 2048 threads (4 chips) and begins to increase after that. While our proposed architecture behaves similarly but at a smaller scale, the render times are considerably lower making the increase in energy an acceptable side effect.

While Mach-RT has no ray traffic to DRAM, each chip still streams the scene once per ray wavefront. Table 2 shows despite that artificially inflated scene traffic, Mach-RT bandwidth and bytes/ray is always the lowest compared to the other architectures and traces more rays per second. Given the presence of the on-chip buffers, on-chip memory energy is higher overall but not enough to outweigh the savings from not interfacing with main memory.

4.3. Comparisons to Existing Systems

For completeness, we also include comparisons to existing software/hardware systems optimized for high-performance ray tracing, including a Microsoft DXR implementation of path tracing [WHSB18] and Intel's Embree [WWB*14] CPU ray tracer both running on commercially-available hardware. The DXR results use an NVIDIA RTX 2080 GPU with 2688 cores running at 1.8 GHz,

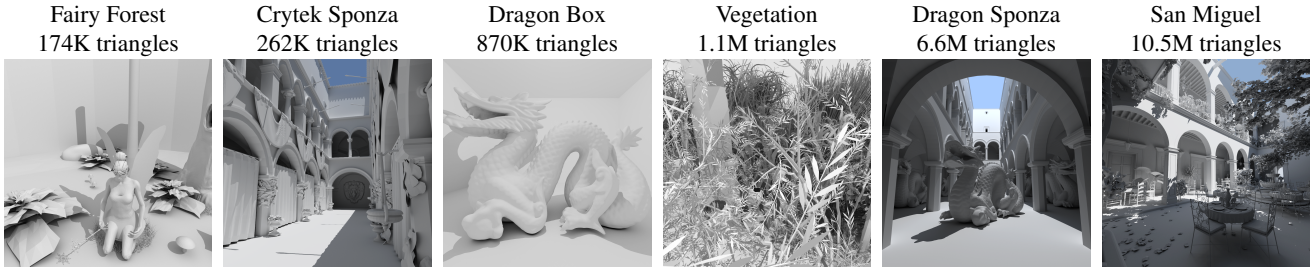

Figure 2: *Benchmark scenes used in our performance tests, ordered according to increasing triangle count.*

Table 2: *Overall performance evaluation of the dedicated ray tracing architectures.*

		Fairy Forest	Crytek Sponza	Dragon Box	Vegetation	Dragon Sponza	San Miguel
Mach-RT, Ours *8 chips*	Frame Render Time	3.42 ms	10.27 ms	5.17 ms	11.75 ms	7.98 ms	16.57 ms
	Rays Traced per sec	1719 M	886 M	2274 M	471 M	1203 M	550 M
	On-Chip Memory Energy	1.81 J	6.88 J	3.03 J	5.39 J	4.19 J	11.17 J
	DRAM Energy	0.14 J	0.33 J	0.28 J	0.38 J	0.79 J	1.68 J
	Total Energy	2.16 J	7.52 J	3.96 J	6.22 J	6.46 J	16.33 J
	Avg. Bandwidth	53 GB/s	22 GB/s	133 GB/s	23 GB/s	280 GB/s	330 GB/s
Dual Streaming *single chip (unrealistically-large)*	Frame Render Time	5.47 ms	17.20 ms	9.54 ms	14.72 ms	13.10 ms	35.23 ms
	Rays Traced per sec	1074 M	529 M	1231 M	376 M	732 M	259 M
	On-Chip Memory Energy	3.11 J	11.38 J	4.83 J	7.65 J	6.80 J	21.96 J
	DRAM Energy	0.57 J	1.76 J	1.12 J	1.05 J	1.49 J	3.71 J
	Total Board Energy	3.74 J	13.40 J	6.10 J	8.91 J	8.46 J	26.13 J
	Avg. Bandwidth	267 GB/s	265 GB/s	283 GB/s	155 GB/s	300 GB/s	266 GB/s
STRaTA *single chip (unrealistically-large)*	Frame Render Time	11.93 ms	29.43 ms	234.68 ms	28.76 ms	140.80 ms	226.70 ms
	Rays Traced per sec	1154 M	680 M	101 M	400 M	143 M	81 M
	On-Chip Memory Energy	1.60 J	4.36 J	9.24 J	3.02 J	6.43 J	10.78 J
	DRAM Energy	11.61 J	1.69 J	11.61 J	0.47 J	4.53 J	12.62 J
	Total Board Energy	21.17 J	6.28 J	21.17 J	2.16 J	11.19 J	23.75 J
	Avg. Bandwidth	257 GB/s	325 GB/s	175 GB/s	341 GB/s	91 GB/s	154 GB/s

Each system has 4096 threads running at 2 GHz frequency. The rendering performance is evaluated with frame render time and the millions (M) of rays traced per second. The component-wise distribution of total energy per frame is also shown. Higher is better for rays traced; otherwise, lower is better.

and 8192 MB GDDR6 memory with 448 GB/s peak bandwidth. The Embree (v2.10) results use the example path tracer (v2.3.2) running on an Intel Core i7-5960X processor with 20 MB L3 cache and 8 cores (16 threads) running at 3.8 GHz. To compare with Microsoft's DXR, we configured our Mach-RT architecture with 6 chips (3072 threads) running at 1.8 GHz. Table 3 shows the results of our tests. Although Embree performs slower than the custom ray tracing hardware, it can run on commodity general-purpose computation hardware. In our tests, the Mach-RT system with 6 chips provides faster render times than DXR running on NVIDIA RTX 2080 for all tested scenes. Notice that the scene data format we use in our system is not extensively compressed. This is particularly important because data movement is the bottleneck of most rendering operations.

5. Conclusion

We introduced a new many-chip architecture Mach-RT that leverages several independent chips co-located on a single board or interposer. This approach implements the Dual Streaming ray traversal but completely removes the ray stream traffic to DRAM and further reduces both DRAM and total system energy while increasing scalable performance up to 4096 threads.

We evaluated our proposed architecture using a cycle-accurate simulator Our proposed architecture is shown to exceed the performance capabilities of the Dual Streaming architecture. For the more direct applicability of the method, we also compared to the implementations of the traditional path tracing algorithm using DXR and Embree libraries running on current hardware, finding that the Mach-RT can significantly outperform them.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant no. 1409129. Scene data: Fairy Forest: U. Utah, Crytek Sponza: F. Meinl at Crytek and M. Dabrovic, Dragon: Stanford CG Lab., Vegetation: S. Laine, and San Miguel: G. Leal Laguno.

References

[AK10] AILA, T. and KARRAS, T. "Architecture Considerations for Tracing Incoherent Rays". HPG '10. 2010 2.

Table 3: *Performance comparison between our Mach-RT architecture and Existing Software/Hardware Systems*

		Fairy Forest	Crytek Sponza	Dragon Box	Vegetation	Dragon Sponza	San Miguel
Mach-RT, Ours	Frame Render Time	4.82 ms	14.81 ms	7.86 ms	20.64 ms	10.89 ms	23.50 ms
(6 chips @ 1.8 GHz)	Rays Traced per sec	1,218 M	615 M	1,495 M	284 M	882 M	388 M
DXR	Frame Render Time	11.13 ms	16.92 ms	16.89 ms	22.51 ms	24.90 ms	37.98 ms
(Nvidia RTX 2080)	Rays Traced per sec	546 M	698 M	745 M	289 M	478 M	288 M
Embree	Frame Render Time	83.6 ms	150.63 ms	103.81 ms	178.99 ms	118.05 ms	143.64 ms
(Intel Core i7-5960X)	Rays Traced per sec	96 M	62 M	89 M	42 M	71 M	50 M

Mach-RT simulated with 6 chips (3072 threads). Embree and DXR run on commercially-available hardware. The clock frequency of the Mach-RT architecture is set to 1.8 GHz, matching the NVIDIA RTX 2080 running DXR. M means millions.

[AL09] AILA, T. and LAINE, S. "Understanding the Efficiency of Ray Traversal on GPUs". *HPG '09*. 2009 1.

[ALK12] AILA, T., LAINE, S., and KARRAS, T. *Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum*. NVIDIA Technical Report NVR-2012-02. June 2012 2.

[BA14] BARRINGER, R. and AKENINE-MÖLLER, T. "Dynamic Ray Stream Traversal". *ACM Trans. Graph.* 33.4 (July 2014) 2.

[BABD00] BALASUBRAMONIAN, R., ALBONESI, D.H., BUYUKTOSUNOGLU, A., and DWARKADAS, S. "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures". *MICRO '00*. 2000 2.

[Bik12] BIKKER, J. "Improving Data Locality for Efficient In-Core Path Tracing". *Computer Graphics Forum*. Vol. 31. 6. 2012 2.

[BKC14] BRUNVAND, E., KOPTA, D., and CHATTERJEE, N. "Why Graphics Programmers Need to Know About DRAM". *SIGGRAPH 2014 Courses*. 2014 2.

[BKM*17] BALASUBRAMONIAN, R., KAHNG, A. B., MURALIMANOHAR, N., et al. "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories". *ACM Trans. Archit. Code Optim.* 14.2 (June 2017) 3.

[BWWÁ18] BENTHIN, C., WALD, I., WOOP, S., and ÁFRA, A. T. "Compressed-leaf Bounding Volume Hierarchies". *HPG '18*. 2018 1.

[CBS*12] CHATTERJEE, N., BALASUBRAMONIAN, R., SHEVGOOR, M., et al. *USIMM: the Utah SImulated Memory Module*. Tech. rep. UUCS-12-02. University of Utah, 2012 4.

[GDS*08] GOVINDARAJU, V., DJEU, P., SANKARALINGAM, K., et al. "Toward A Multicore Architecture for Real-time Ray-tracing". *IEEE/ACM Micro '08*. Oct. 2008 2.

[GR08] GRIBBLE, C. and RAMANI, K. "Coherent Ray Tracing via Stream Filtering". *IRT '08*. 2008 2.

[Kee14] KEELY, S. "Reduced Precision for Hardware Ray Tracing in GPUs". *HPG '14*. 2014 2.

[KJJ*09] KELM, J., JOHNSON, D., JOHNSON, M., et al. "Rigel: an architecture and scalable programming interface for a 1000-core accelerator". *ISCA '09*. 2009 2.

[KKK12] KIM, H., KIM, Y., and KIM, L. "MRTP: Mobile Ray Tracing Processor With Reconfigurable Stream Multi-Processors for High Datapath Utilization". *IEEE JSSC* 47.2 (2012), 518–535 2.

[KSBD10] KOPTA, D., SPJUT, J., BRUNVAND, E., and DAVIS, A. "Efficient MIMD architectures for high-performance ray tracing". *IEEE ICCD '10*. 2010 1.

[KSS*13] KOPTA, D., SHKURKO, K., SPJUT, J., et al. "An energy and bandwidth efficient ray tracing architecture". *HPG '13*. 2013 2, 4.

[KSS*15] KOPTA, D., SHKURKO, K., SPJUT, J., et al. "Memory Considerations for Low Energy Ray Tracing". *CGF* 34.1 (2015) 2, 4.

[LMSS18] LIER, A., MARTINEK, M., STAMMINGER, M., and SELGRAD, K. "A High-Resolution Compression Scheme for Ray Tracing Subdivision Surfaces with Displacement". *Proc. ACM Comput. Graph. Interact. Tech.* 1.2 (Aug. 2018) 2.

[LSL*13] LEE, W., SHIN, Y., LEE, J., et al. "SGRT: A mobile GPU architecture for real-time ray tracing". *HPG '13*. 2013 2.

[LV16] LIKTOR, G. and VAIDYANATHAN, K. "Bandwidth-efficient BVH Layout for Incremental Hardware Traversal". *HPG '16*. 2016 1, 2.

[MB18] MEISTER, D. and BITTNER, J. "Parallel Reinsertion for Bounding Volume Hierarchy Optimization". *CGF*. Vol. 37. 2. 2018 2.

[NFLM07] NAVRÁTIL, P., FUSSELL, D., LIN, C., and MARK, W. "Dynamic ray scheduling to improve ray coherence and bandwidth utilization". IRT '07. 2007 2.

[NKK*14] NAH, J., KWON, H., KIM, D., et al. "RayCore: A Ray-Tracing Hardware Architecture for Mobile Devices". *ACM TOG* 33.5 (2014) 2.

[NVI18] NVIDIA. *Turing GPU Arch.* WP-09183-001_v01. 2018 2.

[SGE*18] SHKURKO, K., GRANT, T., E. BRUNVAND, D. KOPTA, et al. "SimTRaX: Simulation Infrastructure for Exploring Thousands of Cores". *Great Lakes Symposium on VLSI (GLSVLSI)*. 2018 4.

[SGK*17] SHKURKO, K., GRANT, T., KOPTA, D., et al. "Dual Streaming for Hardware-accelerated Ray Tracing". *HPG '17*. 2017 1, 2, 4.

[SKKB09] SPJUT, J., KENSLER, A., KOPTA, D., and BRUNVAND, E. "TRaX: A Multicore Hardware Architecture for Real-Time Ray Tracing". *IEEE Trans. on CAD* 28.12 (2009) 1, 2.

[Tsa09] TSAKOK, JOHN A. "Faster incoherent rays: Multi-BVH ray stream tracing". *HPG '09*. 2009 2.

[TZ14] THONNART, Y. and ZID, M. "Technology assessment of silicon interposers for manycore SoCs: Active, passive, or optical?": *IEEE Networks on Chip confrence*. NoCs '14. Sept. 2014 3.

[USS*17] USMAN, A., SHAH, E., SATISHPRASAD, N. B., et al. "Interposer Technologies for High-Performance Applications". *IEEE Trans. on Components, Packaging and Mfc. Tech.* 7.6 (June 2017) 3.

[VSM*18] VASIOU, E., SHKURKO, K., MALLETT, I., et al. "A detailed study of ray tracing performance: render time and energy cost". *The Visual Computer* 34.6 (June 2018) 2.

[WBS06] WOOP, S., BRUNVAND, E., and SLUSALLAK, P. "Estimating Performance of a Ray Tracing ASIC Design". *IRT '06*. Sept. 2006 2.

[WGBK07] WALD, I., GRIBBLE, C. P., BOULOS, S., and KENSLER, A. *SIMD Ray Stream Tracing-SIMD Ray Traversal with Generalized Ray Packets and On-the-fly Re-Ordering*. Tech. rep. UUSCI-2007-012. SCI Institute, U. of Utah, 2007 2.

[WHSB18] WYMAN, C., HARGREAVES, S., SHIRLEY, P., and BARRÉ-BRISEBOIS, C. "Introduction to DirectX Raytracing". *ACM SIGGRAPH 2018 Courses*. Aug. 2018 2, 4.

[WM95] WULF, WM. A. and MCKEE, S.A. "Hitting the Memory Wall: Implications of the Obvious". *Comp. Arch. News* 23.1 (Mar. 1995) 2.

[WWB*14] WALD, I., WOOP, S., BENTHIN, C., et al. "Embree - A Kernel Framework for Efficient CPU Ray Tracing". *SIGGRAPH '14*. 2014 2, 4.

[YKL17] YLITIE, H., KARRAS, T., and LAINE, S. "Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs". *HPG '17*. 2017 1, 2.