Mach-RT: A Many Chip Architecture for High Performance Ray Tracing

Elena Vasiou, Konstantin Shkurko, Erik Brunvand, *Senior Member, IEEE*, and Cem Yuksel

Abstract—Data movement, particularly access to the main memory, has been the bottleneck of most computing problems. Ray tracing is no exception. We propose an unconventional solution 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. This paper extends our previous work [1] with design space exploration of the L3 cache size, more detailed evaluation of energy and memory performance, a discussion of energy delay product, and a brief exploration of boards with 16 chips. We also introduce new treelet enqueueing logic for the predictive scheduler.

Index Terms—Raytracing hardware, Graphics Accelerators

1 Introduction

The tremendous rendering performance of commercial GPUs is due to two important properties of the rasterization algorithm. The first one is its high level of parallelism, which provides a great fit for GPUs with many parallel cores. The second one is 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. The caching and prefetching mechanisms of GPUs are structured to exploit this behavior, allowing the performance of rasterization to scale well with the increasing number of computation cores.

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. Not only does this increase data access latency but it also makes it highly challenging to hide this latency. Therefore, most recent work on accelerating ray tracing concentrates on improving its memory behavior, such as coalescing memory accesses [2], [3], [4] or compressing scene data [5], [6], [7]. The ray tracing performance of NVIDIA's recent Turing architecture [8] is a testament to how far such optimizations can go, combined with dedicated computation units.

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. Dual Streaming [9] 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. 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 its ray stream by storing ray data in on-chip buffers. This brings the main memory traffic to its absolute minimum, but it also limits the number of rays that can be in flight simultaneously to the size of these 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 to reduce the workload on the main memory, a single memory system can 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 different instructions, unlike

• *E. Vasiou, K. Shkurko, E. Brunvand and C. Yuksel are with the School of Computing, University of Utah, Salt Lake City, UT, 84112. E-mail: {elvasiou, kshkurko, elb}@cs.utah.edu, cem@cemyuksel.com*

Manuscript received April 19, 2005; revised August 26, 2015.

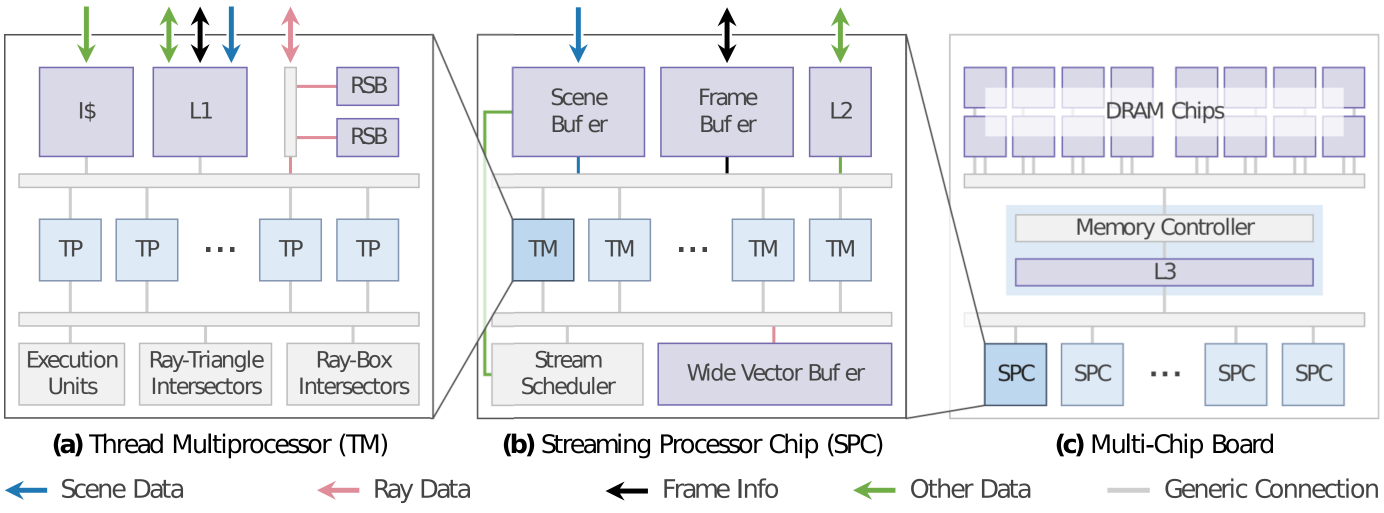

Fig. 1. Overview of Mach-RT – our multiple chip architecture for high-performance ray tracing. We propose a solution that integrates many chips on a board sharing main (scene) memory, while keeping all rays stored in on-chip memory distributed across those chips. Each chip contains a number of Thread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

existing GPUs that rely on the SIMD (single instruction multiple data) paradigm. 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 [10], [11]. 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 because the main consumer of energy in modern architectures has been the memory system and optimizations to the memory access patterns can have a profound impact on energy use, as well as performance. This is an important contribution as in fact, energy consumption is a growing concern for real-time rendering, especially considering recent commercial efforts in cloud computing for online gaming.

We demonstrate the capabilities of the new Mach-RT architecture by comparing against other leading ray tracing hardware architectures like STRaTA [12] and Dual Streaming [9]. We show that our proposed configuration scales better with the increasing number of threads. In addition, we provide comparisons to a single unrealistically-large chip that contains the same amount of local storage and computational resources as all of our processor chips put together, to demonstrate the ability of our multi-chip solution to share off-chip memory resources.

We also provide comparisons to highly-optimized ray tracing software/hardware products from the industry, including Intel's Embree [13] and Microsoft's DXR [14] running on NVIDIA's recent Turing architecture with hardware support for ray tracing [8], showing that our system can provide substantially improved performance.

This paper extends our prior work [1] by providing a design space exploration of the L3 cache size with expanded evaluation of energy consumption and memory behavior, a discussion on energy delay product, and an exploration of the scaling of our system with boards of 16 chips.

Furthermore, we introduce a new scheme for the predictive scheduling and prefetching of treelets that operates in a depth-first manner. Our results show that depth-first treelet traversal does not always provide a performance improvement as compared to scheduling the treelets in a breadth-first order [1], [9].

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 [15], [16], [17], [18], [19], [20], [21], [22]. While effective at accelerating the issuing of memory requests, the fixed-function units typically restrict the acceleration structures to one exact type. 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 [23], [24], [25], [26], or applying the SIMD processing paradigm to the algorithm [24], [27], [28]. Collecting ray and scene data requests through software means is also a widely studied solution [2], [3], [4], [29], [30], [31], [32]. None of those methods, however, focus on increasing the utilization of larger computation thread counts.

Other work has focused on using the acceleration structure to control rays by utilizing batching and packets during scheduling [26], [33], [34], [35], ordered ray generation [13], [36] or sorting [37], [38], [39], [40]. While these methods decrease the resource requirements to process rays, they do not completely eliminate ray traffic to and from main memory DRAM. One can reduce the memory traffic by exploiting the BVH structure [41], [42] or by modifying it to get better performance [43], [44], [45], [46]. Even in cases where the proposed architecture is designed to keep rays on chip [12], [47], it is still limited in the number of rays it can process simultaneously.

A large pool of work is focused on compressing the acceleration structure [**?**], [5], [6], [7], [48], [49], [50]. These methods are effective at reducing the amount of data that needs to be communicated between the processors and memory thus improving performance. Even the most aggressive compression schemes cannot realistically enable storing all rays on chip because rendering a single frame can require tens of millions of rays. However, the idea of compression is fundamentally orthogonal to the solution we are presenting in this paper and it could be used in conjunction with our Mach-RT multi-chip architecture to reduce the size of the on-chip memories and allow more threads to be implemented per chip.

Dual Streaming Algorithm: Our proposed system is inspired by the Dual Streaming architecture's approach to restructuring and minimizing the scene data stream [9].

In that approach the scene stream consists of BVH treelets [42], 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. Once a treelet's ray queue is drained, the child treelets are marked available for processing.If a treelet has no rays for processing, then the entire scene subtree is skipped. Treelets with non-empty ray queues can be processed in parallel by distributing their corresponding rays between threads. Traversal continues until all ray queues are empty.

Rays rely on a local stack when traversing an individual treelet. 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. Ray duplication prevents rays from revisiting and reloading parent treelets on their way to sibling treelets.

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

At a first glance, the system proposed in this paper seems counter-intuitive. With the increasing size and power of GPU chips, why propose a multiple chip solution? The answer is, again, related to memory. 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 [47], [51], [52], [53], [54], [55]. That is specifically the case for ray tracing as well [56].

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. Furthermore, this would minimize the DRAM traffic so much that it is conceivable to connect multiple processor chips to the same DRAM.

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.

In the near term future,a chip could easily fit a few million 32 byte sized rays. Storing the frame buffer on chip, which uses less memory than the ray data, also helps alleviate pressure on DRAM. Thus during ray traversal, DRAM needs to service only scene data read requests.

While using a collection of chips is unconventional considering today's GPU designs, such architectures were explored and built in the distant past. A good example is the Pixel Planes architecture from the 1980's [57], [58], [59], which essentially allocated one (bit-serial) processor per pixel in a custom VLSI "enhanced memory" chip. The Pixel Planes 4 prototype was envisioned to be scalable to a 512×512 pixel display with 128 pixels on each of the 2048 enhanced-memory Pxpl4 chips. While not envisioning a system with 2048 chips, we are inspired by the spirit of the Pixel Planes to imagine accelerating ray tracing through a modern implementation of multiple chips with enhanced memory.

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 [60], [61]. 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. It is worth noting that while NVIDIA integrates DRAM chips directly on their boards, AMD already uses silicon interposers to assemble processor systems [62].

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 such as [63], 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.

Each SPC contains an on-chip wide vector storage buffer that stores unique ray data for the entire wavefront. The ray stream consists of indexes into the wide vector storage buffer to reduce the storage requirements of ray duplicates. Thus we reduce the on-chip area cost of the ray stream.

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.

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 without incurring additional latency from a more complex memory model. 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

Given the general architecture description, we can explore the design space of possible Mach-RT configurations to select the one that performs optimally.

We use Cacti 7.0 [64], [65] 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.

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

For comparisons, single large chip incarnations of each architecture are configured such that their compute and memory capacities match the values of our proposed architecture as noted in Table 1. For the direct comparison with architectures published previously, we simulate running at 2.0 GHz. To compare against the newest NVIDIA Turing GPU, we configure our architecture with the same number of threads and running at the matching frequency of 1.8 GHz.

Each SPC in our simulations 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.

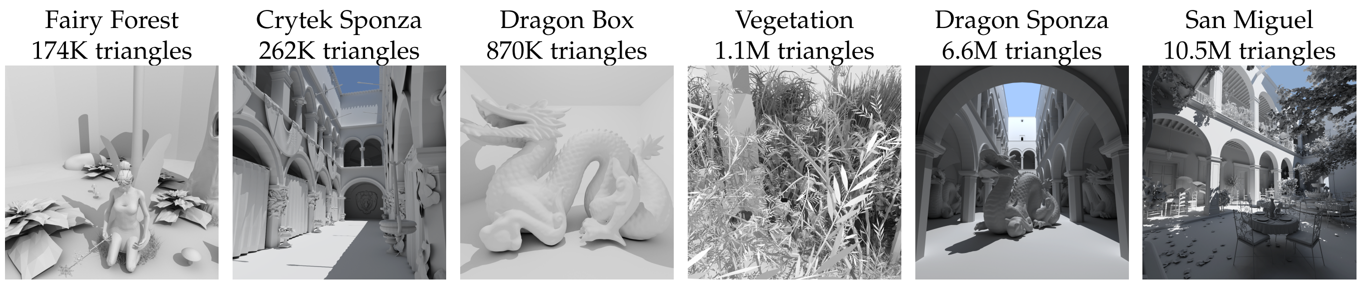

Fig. 2. Benchmark scenes used in our performance tests, ordered according to increasing triangle count from left to right.

3.3 Scheduling

Since the dual streaming algorithm maximizes scene reuse to reduce memory accesses and constrains the tracing to occur as a wavefront, the traversal order follows a fixed predetermined path. In the original dual streaming architecture [9] and our prior work [1] the partitioning of the scene BHV into treelets dictates a rigid hierarchy, where rays move strictly from a parent treelet to its children. In the hardware architecture, the scheduler is responsible for maintaining this top-down traversal order by keeping track of which treelets are currently being processed (i.e the treelets that are in the *working set*) and which are to be processed next (i.e the eligible children).

When the working set has a vacant slot, the scheduler searches the eligible children list until it finds a treelet that has been assigned rays and whose parent has fully drained its ray queue.

The eligible children list is implemented as a queue and treelets are placed within when a ray crosses the boundary of the treelet root node. While some treelets might be skipped completely if no rays enter them, the overall traversal of treelets follows a breath-first ordering.

On the other hand, most other ray tracing applications favor depth-first traversal because it allows reaching leaf nodes earlier than traversing breadth-first. Additionally, early ray termination can skip parts of the tree traversal, providing superior performance when combined with depth-first traversal.

We introduce a new mode of scheduling treelets that resembles a depth-first traversal in an attempt to benefit from early ray termination.

The depth-first scheduler replaces the queue for eligible children with a stack. Under this mode of operation, when the working set needs to be replenished, the scheduler looks through the treelets in the list of eligible children and pops the first treelet whose parent has been evicted from the working set. Once the newly added treelet is fetched and begins processing, its children are placed on the top of the eligible children stack. This process continues until all treelets with non-empty ray queues are processed.

While this ordering prioritizes nodes deeper into the tree, it is not a pure depth-first traversal. Replacing the original queue with a stack implies that for a given treelet its children begin processing before sibling nodes. However, the scheduling still needs to ensure that rays do not traverse up the tree. Therefore, we still have the constraint that a treelet is placed into the working set only after its parent has finished processing. This can lead to situations where a node higher up the tree will be preferred, but it occurs considerably less often, as compared to the breadth-first scheduler.

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 our knowledge this is the first proposed architecture for ray tracing with these properties, which are critical as scene sizes and screen resolutions continue to increase.

To demonstrate this, we evaluate our proposed hardware architecture on a set of test scenes using a cycle-accurate simulator [66]. The simulator integrates a detailed memory simulator [54] that accurately models DRAM, a vital component of our experiments that ensures proper system evaluation. Because of the detailed nature of system simulation and the requirement to simulate image generation to completion to accurately model all data interrelations, each simulation can take from a few hours to several days to finish, while the simulated time is only a fraction of a second. Given this simulation overhead, we use a relatively small 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 [67] 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. We use a breadth-first scheduler, like the original dual streaming method [9], unless otherwise specified.

4.1 L3 Cache Size Optimization

First, we model how multiple chips could reasonably share access to a large DRAM-based main memory system. Even though the traversal order we use minimizes the scene traffic for a single chip, connecting multiple chips (each with their own memory controller) is not a trivial task that brings up a variety of issues related to resource contention, bandwidth optimization, and data locality. One well-understood mechanism for dealing with this sort of an issue is to add a level of caching, with a multi-banked cache feeding the

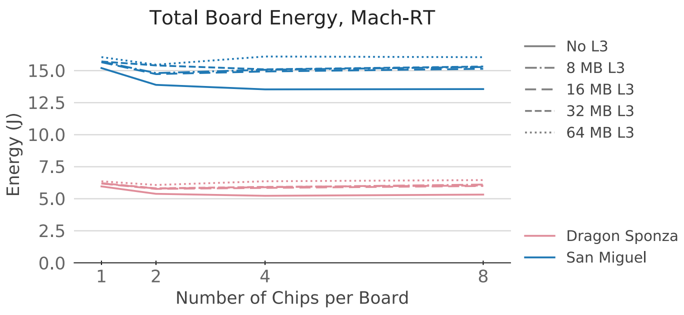

Fig. 3. Effect of increasing L3 cache size on total energy per frame for the Mach-RT architecture.

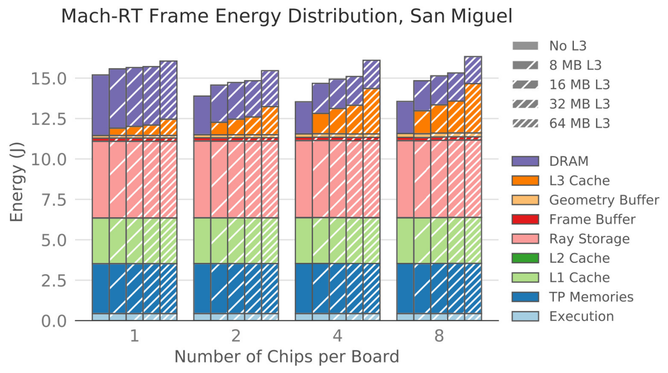

Fig. 4. Energy distribution for the San Miguel scene with varying L3 cache sizes of the Mach-RT architecture with 1-8 chips. The energy profile for other tested scenes is similar.

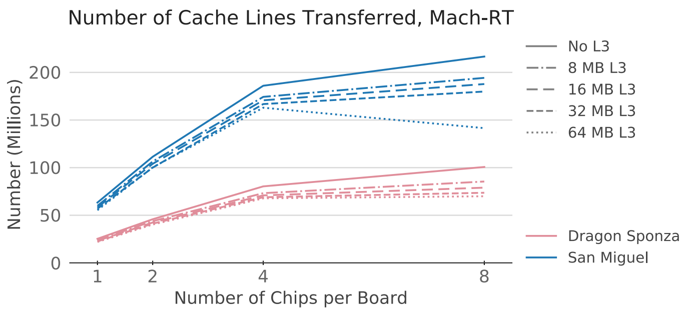

Fig. 5. Effect of increasing L3 cache size on number of cache lines transferred from DRAM for the Mach-RT architecture. Larger L3 caches lead to a reduction.

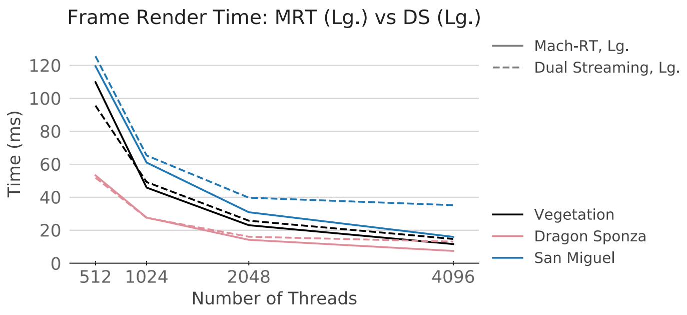

Fig. 6. Frame render time for Mach-RT and Dual Streaming architectures, both as individual unrealistically-large chips.

processors and a sophisticated memory controller refilling that cache from DRAM.

We test the effects on performance between no off-chip cache and a single L3 cache of varied sizes at 8 MB, 16 MB, 32 MB and 64 MB. The effect on frame rendering time is minimal, with differences of less than 1 ms/frame for the larger scenes (Dragon Sponza and San Miguel) even at 8 chips. This is not surprising because the scene bandwidth has been reduced so much that it is no longer the limiting factor on frame time. However, we expect to see differences in energy and bandwidth used because the SRAM L3 cache has different energy behavior than DRAM, and the larger cache sizes qualitatively increase the size of the DRAM row buffer.

We observe small increases in energy used per scene when increasing the size of the L3 cache. Although accessing SRAM uses less energy than accessing DRAM, SRAM energy increases as its size increases. Also, while DRAM energy does decrease when the L3 cache is used, that decrease does not completely outweigh the extra energy used in the L3 cache SRAM (Figure 3). Figure 4 shows the distribution of energy between architectural components for the San Miguel scene; the other tested scenes follow a similar trend.

The biggest positive impact of the L3 cache is the reduction of the total bandwidth to DRAM - a precious resource. While the energy used per access increases with larger cache sizes, at 64 MB we observe fewer total accesses to the L3 cache and the highest hit rates in all scenes for all chip counts.

Placing an off-chip cache between the main memory and the processor chips reduces the number of cache lines transferred (Figure 5) and DRAM bandwidth. For most tested scenes, the bandwidth reduction reaches almost 80 GB/s at 8 chips for the large scenes. The reduction in the number of cache lines transferred ranges from approximately 5 million for Crytek Sponza and Fairy Forest, to an average of 50 million for San Miguel and Dragon Sponza. These are consistent with the cache hit rates increasing from 8% at 8 MB to 20% at 64 MB for San Miguel. Thus we select the 64MB configuration for comparison with other methods.

4.2 Single Large Chip

We evaluate a single large chip version of our proposed architecture and compare it with the Dual Streaming architecture [9] 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. Figure 6 shows a subset of the tested scenes. 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. Although Dual Streaming reduces scene traffic to its

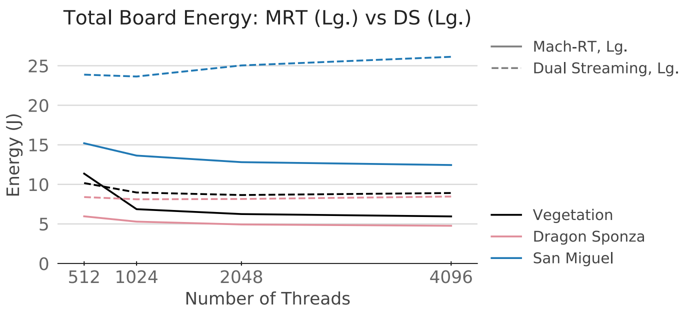

Fig. 7. Total board energy for Mach-RT and Dual Streaming architectures, both as individual unrealistically-large chips.

minimum, our proposed architecture also completely eliminates the off-chip ray traffic. 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 (Figure 7). Because the chip has more of the required data available, it can utilize its resources more effectively. This direct comparison shows the dramatic effect of on-chip availability of all rays and most of the scene data. However, such configurations require chips with unrealistically-large on-chip buffers: over 100 MB, most of which would be allocated for ray storage.

4.3 Multiple-Chip Ray Tracing

While the single large chip simulation is interesting, it is unrealistic. 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 [9] and STRaTA chip [12], [47] with hardware resources scaled to a combination of all of our chips. Figure 8 demonstrates how multiple chips can benefit performance. 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 (Figure 9). 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 as seen in Figure 8. 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 extends our analysis for the energy and performance of our 8 chip sytem and also 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. Additionally, since the L3 cache is designed to serve overlapping scene requests from multiple chips the pressure on DRAM is greatly reduced leading to much lower total energy. 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.

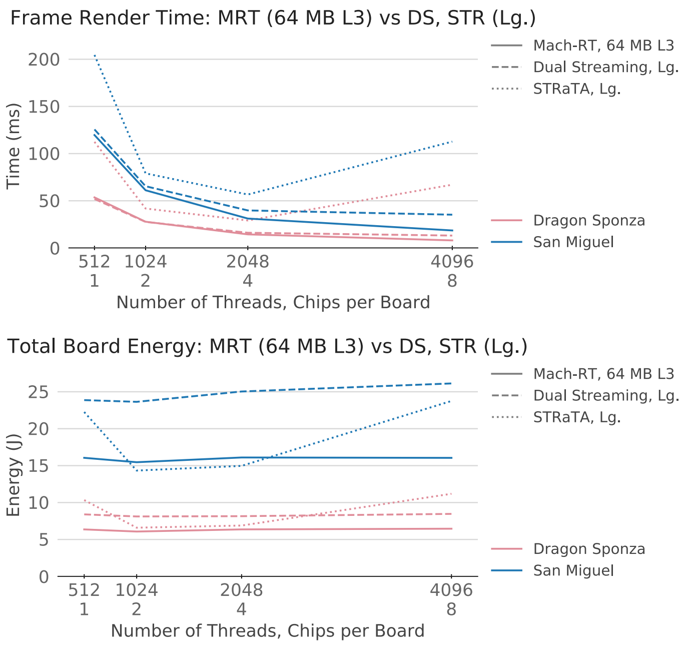

Fig. 8. Frame render time and total board energy for Mach-RT with multiple chips, and Dual Streaming and STRaTA architectures, both as individual unrealistically-large chips.

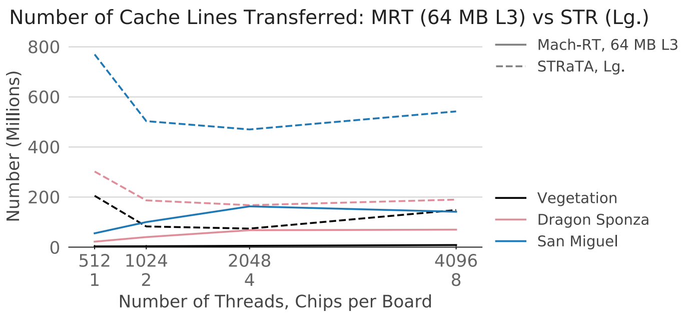

Fig. 9. Number of cache lines transferred for Mach-RT with multiple chips and STRaTA as individual unrealistically-large chip.

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
	Compute Energy	0.08 J	0.28 J	0.14 J	0.21 J	0.19 J	0.44 J
	On-Chip Memory Energy	1.81 J	6.88 J	3.03 J	5.39 J	4.19 J	11.17 J
	L3 Cache Energy	0.13 J	0.23 J	0.08 J	0.37 J	1.29 J	3.05 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
	Cache Lines Transferred	6 M	7 M	22 M	8 M	70 M	171 M
	Bytes / Ray	62	51	116	98	466	1201
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
	Compute Energy	0.08 J	0.29 J	0.15 J	0.23 J	0.20 J	0.49 J
	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
	Cache Lines Transferred	46 M	142 M	845 M	71 M	123 M	293 M
	Bytes / Ray	498	1,001	460	823	819	2,053
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
	Compute Energy	0.09 J	0.23 J	0.33 J	0.16 J	0.23 J	0.36 J
	On-Chip Memory Energy	1.60 J	4.36 J	9.24 J	3.02 J	6.43 J	10.78 J
	DRAM Energy	0.47 J	1.69 J	11.61 J	0.47 J	4.53 J	12.62 J
	Total Board Energy	2.15 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
	Cache Lines Transferred	41 M	136 M	640 M	148 M	190 M	542 M
	Bytes / Ray	445	955	3,484	1,707	1,265	3,806

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.

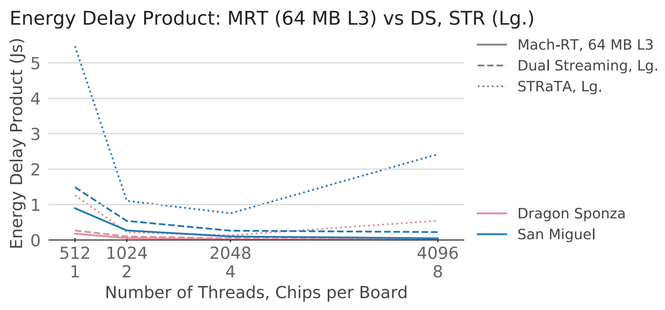

Fig. 10. Energy Delay Product for Mach-RT, Dual Streaming and STRaTA architectures

4.4 Energy Delay Product

The Energy-Delay product (EDP) is widely used as a metric capable of coupling both the energy consumption and performance of alternative architecture design choices [68]. It is used both in low-power system design where there is often an interest in trading off delay for increased energy efficiency, and in high-performance system design where energy dissipation is a limiting factor. To better measure the efficiency of our system against the previously developed ones, we investigate the effects of increasing the number of chips on the energy-delay product. As seen from Figure 10, for the two largest scenes of Dragon Sponza and San Miguel our architecture has consistently lower EDP than both Dual Streaming and STRaTA, with the latter experiencing a large spike at 4096 threads. While the Dual Streaming EDP is continuously improving as ours, at 8 chips of the equivalent 4096 thread chip, our architecture achieves an EDP of 0.0108 and 0.0478 for Dragon Sponza and San Miguel respectively, compared to 0.332 and 0.223 for the large Dual Streaming.

4.5 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 [14] and Intel's Embree [13] 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, 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. For a fair comparison with 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.

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 (6 chips @ 1.8 GHz)	Frame Render Time	4.82 ms	14.81 ms	7.86 ms	20.64 ms	10.89 ms	23.50 ms
	Rays Traced per sec	1,218 M	615 M	1,495 M	284 M	882 M	388 M
DXR (Nvidia RTX 2080)	Frame Render Time	11.13 ms	16.92 ms	16.89 ms	22.51 ms	24.90 ms	37.98 ms
	Rays Traced per sec	546 M	698 M	745 M	289 M	478 M	288 M
Embree (Intel Core i7-5960X)	Frame Render Time	83.6 ms	150.63 ms	103.81 ms	178.99 ms	118.05 ms	143.64 ms
	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.

TABLE 4
Render times of our Mach-RT architecture with breadth-first and depth-first scheduling orders tested using 8 chips

	Crytek Sponza	Vegetation	Dragon Sponza	San Miguel
breadth-first	10.27 ms	11.75 ms	7.98 ms	18.52 ms
depth-first	10.26 ms	11.56 ms	8.29 ms	17.18 ms

4.6 Depth-First Scheduling

All results presented above have been generated using a breadth-first scheduler. We present a performance comparison of the breadth-first scheduler to our depth-first scheduler in Table 4, using our Mach-RT architecture with 8 chips. Our tests show about 7% speed up for the San Miguel scene (18.51 ms to 17.18 ms) but about 4% slow down for the Dragon Sponza scene (7.98 ms to 8.29 ms). In our tests the render times for the other scenes are almost identical between the two schedulers.

While one might expect that the depth-first scheduler, which prioritizes a traversal that gets to the leaf nodes and finds triangle hits earlier, would have a clear performance advantage, our tests show that this is not the case for our Mach-RT architecture. This is mainly because the performance advantage of depth-first traversal is directly tied to early ray termination. With early ray termination, finding hits earlier allows skipping the traversal of a portion of the tree that would intersect with a ray. However, this is effective only if the hit that is found earlier during traversal is closer to the ray origin. This requires picking the right traversal order, based on the ray direction. Dual streaming, however, uses the same treelet traversal order for all rays in the entire wavefront, which cannot be modified for each ray based on its direction. As a result, simply using a depth-first treelet traversal order is not sufficient to effectively skip a relatively large portion of the ray traversal.

5 Discussion

While the increased on-chip memory needs require us to keep the thread count per chip relatively low, we are able to connect enough chips together to use high total thread counts.

We have shown the performance of up to 8 chips and have explored systems with 16 chips (8192 threads).

As seen in Table 5, increasing to 16 chips, the frame time decreases less. At larger chip counts, there are fewer rays per chip because each is assigned a smaller pixel workload. With less work per thread on a chip, we observe work starvation and wasted resource utilization through a drop in the average issue rate per thread from 70% to 40%. Additionally, the impact of fetching a treelet is more significant per ray because there are fewer rays per chip to amortize the cost of scene data transfer. For Mach-RT configurations with more than 16 chips, we observe that prefetching entire 64 KB treelets starts hurting performance. Increasing the image resolution could provide the necessary work to use more chips effectively; however, since each simulation can take a day or more to complete, we kept the resolution at 1024×1024.

Although we have reduced the impact of ray duplication by representing each duplicate as an index into ray data store, the increase in ray counts still leads to significant drawbacks. Without the extra rays, the on-chip ray storage could be reduced, thus freeing on-chip area to place more threads within a chip. The extra rays also introduce additional computation that could otherwise be avoided.

The extra computation stems not only from traversal of ray duplicates but also from the ineffectiveness of the implemented early ray termination. Although Mach-RT keeps hit records on chip and checks each ray for a closer hit before traversal, the performance benefits of that check are minimal. It accounts to less than 10 milliseconds per frame, even for the Vegetation scene whose high depth complexity would greatly benefit from early ray termination.

The area and energy estimates provided in this paper use a 65nm manufacturing process, since we currently have no access to synthesis/simulation models at the cutting-edge 14nm and 7nm nodes that are used commercially. Conservative scaling could give a rough idea of implementing our proposed hardware in those nodes and make it comparable to currently commerically available architectures.

6 Future Work

While the proposed architecture handles scene data traffic efficiently, compressing both off-chip scene and on-chip ray data would substantially improve both the rendering speed and the total energy use. Scene compression would allow rendering larger scenes while maintaining low bandwidth requirements even for boards with many chips. It would also allow packing more information within each treelet.

Reducing the size of the ray storage has not been investigated extensively. These improvements would directly translate to reductions in the area demand from the on-chip memories, allowing each chip to contain more threads and reducing the size of the board.

TABLE 5
Frame render time of our Mach-RT architecture configured with different chip counts running at 2 GHz.

		Fairy Forest	Crytek Sponza	Dragon Box	Vegetation	Dragon Sponza	San Miguel
Mach-RT, Ours (@ 2 GHz)	**4 Chips**	6.43 ms	20.18 ms	10.83 ms	23.25 ms	14.36 ms	31.14 ms
	8 Chips	3.42 ms	10.27 ms	5.17 ms	11.75 ms	7.98 ms	18.52 ms
	16 Chips	1.88 ms	5.30 ms	3.48 ms	6.02 ms	6.39 ms	12.21 ms

Although the render time decreases for all tested scenes as the number of chips increases, the performance gains are reduced.

As the memory technology improves, it would be beneficial to explore different memory types for the off-chip L3 cache and the main memory. Bandwidth is still an important concern at higher chip counts, even with the scene streaming to multiple chips. Utilizing other forms of memory that trade off lower energy cost per access for higher latency such as cached DRAM [69] or HBM [70] would further contribute to lowering the total energy cost per frame without performance penalties. The more experimental HMC [71] would open possibilities for further optimization in data management towards a multitude of chips.

This paper focuses on a linear chip layout, where all chips are connected to the main memory in parallel through a single L3 cache. Organizing chips hierarchically would be an interesting future direction. Dedicated cache replacement policies at each tier could also enable further data reuse, thus reducing the memory impact further and allowing to connect more chips together.

7 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 and assessed its realizability using architecturally-sound approaches. 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.

Acknowledgments

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

[1] E. Vasiou, K. Shkurko, E. Brunvand, and C. Yuksel, "Mach-rt: A many chip architecture for high-performance ray tracing," in *High-Performance Graphics (HPG 2019)*. New York, NY, USA: ACM, 2019.
[2] J. Spjut, A. Kensler, D. Kopta, and E. Brunvand, "TRaX: A multicore hardware architecture for real-time ray tracing," *IEEE Trans. on CAD*, vol. 28, no. 12, pp. 1802 – 1815, 2009.
[3] D. Kopta, J. Spjut, E. Brunvand, and A. Davis, "Efficient MIMD architectures for high-performance ray tracing," in *IEEE ICCD '10*, Oct. 2010.
[4] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on GPUs," in *HPG '09*, 2009, pp. 145–149.
[5] G. Liktor and K. Vaidyanathan, "Bandwidth-efficient BHV layout for incremental hardware traversal," in *HPG '16*, 2016.
[6] H. Ylitie, T. Karras, and S. Laine, "Efficient incoherent ray traversal on GPUs through compressed wide BHVs," in *HPG '17*, 2017.
[7] C. Benthin, I. Wald, S. Woop, and A. T. Áfra, "Compressed-leaf bounding volume hierarchies," in *HPG '18*, 2018.
[8] NVIDIA, "Turing GPU arch." 2018, wP-09183-001_v01.
[9] K. Shkurko, T. Grant, D. Kopta, I. Mallett, C. Yuksel, and E. Brunvand, "Dual streaming for hardware-accelerated ray tracing," in *HPG '17*. ACM/EG, 2017.
[10] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, Mar. 1997.
[11] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: a modular reconfigurable architecture," in *IEEE ISCA '00*, 2000.
[12] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand, and A. Davis, "An energy and bandwidth efficient ray tracing architecture," in *HPG '13*, 2013.
[13] I. Wald, S. Woop, C. Benthin, G. Johnson, and M. Ernst, "Embree - a kernel framework for efficient CPU ray tracing," in *SIGGRAPH '14*, 2014.
[14] C. Wyman, S. Hargreaves, P. Shirley, and C. Barré-Brisebois, "Introduction to directx raytracing," in *ACM SIGGRAPH 2018 Courses*, Aug. 2018.
[15] J. Schmittler, I. Wald, and P. Slusallek, "SaarCOR – a hardware architecture for realtime ray-tracing," in *EUROGRAPHICS Workshop on Graphics Hardware*, Sep. 2002.
[16] J. Schmittler, S. Woop, D. Wagner, W. Paul, and P. Slusallek, "Realtime ray tracing of dynamic scenes on an FPGA chip," in *Graphics Hardware Conference*, Aug. 2004, pp. 95–106.
[17] S. Woop, J. Schmittler, and P. Slusallek, "RPU: A programmable ray processing unit for realtime ray tracing," *ACM Trans. on Graphics*, vol. 24, no. 3, Jul. 2005.
[18] S. Woop, E. Brunvand, and P. Slusallak, "Estimating performance of a ray tracing ASIC design," in *IRT '06*, Sep. 2006.
[19] H. Kim, Y. Kim, and L. Kim, "Reconfigurable mobile stream processor for ray tracing," in *Custom Int. Circ. Conf. (CICC)*, 2010.
[20] H.-Y. Kim, Y.-J. Kim, and L.-S. Kim, "MRTP: Mobile ray tracing processor with reconfigurable stream multi-processors for high datapath utilization," *IEEE JSSC*, vol. 47, no. 2, pp. 518–535, 2012.
[21] W. Lee, Y. Shin, J. Lee, J. Kim, J. Nah, S. Jung, S. Lee, H. Park, and T. Han, "Sgrt: A mobile GPU architecture for real-time ray tracing," in *HPG '13*, 2013.
[22] J. Nah, H. Kwon, D. Kim, C. Jeong, J. Park, T. Han, D. Manocha, and W. Park, "Raycore: A ray-tracing hardware architecture for mobile devices," *ACM TOG*, vol. 33, no. 5, 2014.
[23] I. Wald, C. P. Gribble, S. Boulos, and A. Kensler, "Simd ray stream tracing-simd ray traversal with generalized ray packets and on-the-fly re-ordering," SCI Institute, U. of Utah, Tech. Rep. UUSCI-2007-012, 2007.
[24] C. Gribble and K. Ramani, "Coherent ray tracing via stream filtering," in *IRT '08*, 2008.
[25] J. Bikker, "Improving data locality for efficient in-core path tracing," in *Computer Graphics Forum*, vol. 31, no. 6, 2012.
[26] R. Barringer and T. Akenine-Möller, "Dynamic ray stream traversal," *ACM Trans. Graph.*, vol. 33, no. 4, Jul. 2014.

[27] K. Ramani and C. Gribble, "StreamRay: A stream filtering architecture for coherent ray tracing," in *ASPLOS '09*, 2009.

[28] A. Lier, M. Stamminger, and K. Selgrad, "CPU-style SIMD ray traversal on GPUs," in *HPG '18*, 2018.

[29] V. Govindaraju, P. Djeu, K. Sankaralingam, M. Vernon, and W. R. Mark, "Toward a multicore architecture for real-time ray-tracing," in *IEEE/ACM Micro '08*, October 2008.

[30] J. Kelm, D. Johnson, M. Johnson, N. Crago, W. Tuohy, A. Mahesri, S. Lumetta, M. Frank, and S. Patel, "Rigel: an architecture and scalable programming interface for a 1000-core accelerator," in *ISCA '09*, 2009.

[31] J. Spjut, D. Kopta, S. Boulos, S. Kellis, and E. Brunvand, "TRaX: A multi-threaded architecture for real-time ray tracing," in *IEEE SASP*, 2008.

[32] T. Aila, S. Laine, and T. Karras, "Understanding the efficiency of ray traversal on GPUs – Kepler and Fermi addendum," NVIDIA Technical Report NVR-2012-02, Jun. 2012.

[33] W. Lee, Y. Shin, S. Hwang, S. Kang, J. Yoo, and S. Ryu, "Reorder buffer: an energy-efficient multithreading architecture for hardware mimd ray traversal," in *HPG '15*, 2015.

[34] M. Son and S.-E. Yoon, "Timeline scheduling for out-of-core ray batching," in *HPG '17*, 2017.

[35] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald, "Packet-based Whitted and Distribution Ray Tracing," in *GI '07*, May 2007.

[36] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Trans. on Graphics*, vol. 21, no. 3, 2002.

[37] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan, "Rendering complex scenes with memory-coherent ray tracing," in *Proc. of SIGGRAPH*, ser. SIGGRAPH '97, 1997.

[38] J. Bigler, A. Stephens, and S. G. Parker, "Design for parallel interactive ray tracing systems," in *IRT '06*, 2006.

[39] B. Moon, Y. Byun, T.-J. Kim, P. Claudio, H.-S. Kim, Y.-J. Ban, S. W. Nam, and S.-E. Yoon, "Cache-oblivious ray reordering," *ACM Trans. Graph.*, vol. 29, no. 3, 2010.

[40] C. Eisenacher, G. Nichols, A. Selle, and B. Burley, "Sorted deferred shading for production path tracing," *Computer Graphics Forum*, vol. 32, no. 4, 2013.

[41] P. Navrátil, D. Fussell, C. Lin, and W. Mark, "Dynamic ray scheduling to improve ray coherence and bandwidth utilization," ser. IRT '07, 2007.

[42] T. Aila and T. Karras, "Architecture considerations for tracing incoherent rays," ser. HPG '10, 2010.

[43] T. Viitanen, M. Koskela, P. Jääskeläinen, H. Kultala, and J. Takala, "Mergetree: A fast hardware HLBVH constructor for animated ray tracing," *ACM Trans. Graph.*, vol. 36, no. 5, Oct. 2017.

[44] D. Meister and J. Bittner, "Parallel reinsertion for bounding volume hierarchy optimization," in *CGF*, vol. 37, no. 2, 2018.

[45] J. A. Tsakok, "Faster incoherent rays: Multi-BVH ray stream tracing," in *HPG '09*, 2009.

[46] M. Hapala, T. Davidovič, I. Wald, V. Havran, and P. Slusallek, "Efficient stack-less BHV traversal for ray tracing," in *SCCG '11*, Apr. 2013.

[47] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand, and A. Davis, "Memory considerations for low energy ray tracing," *CGF*, vol. 34, no. 1, 2015.

[48] S. Keely, "Reduced precision for hardware ray tracing in GPUs," in *HPG '14*, 2014.

[49] T. Kim, B. Moon, D. Kim, and S. Yoon, "RACBVHs: Random-accessible compressed bounding volume hierarchies," *IEEE TVCG*, vol. 16 2, 2010.

[50] K. Selgrad, A. Lier, M. Martinek, C. Buchenau, M. Guthe, F. Kranz, H. Schäfer, and M. Stamminger, "A compressed representation for ray tracing parametric surfaces," *ACM Trans. Graph.*, vol. 36, no. 1, Nov. 2016.

[51] W. A. Wulf and S. McKee, "Hitting the Memory Wall: Implications of the Obvious," *Comp. Arch. News*, vol. 23, no. 1, March 1995.

[52] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," in *MICRO '00*, 2000.

[53] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems - Cache, DRAM, Disk*. Elsevier, 2008.

[54] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," University of Utah, Tech. Rep. UUCS-12-02, 2012.

[55] E. Brunvand, D. Kopta, and N. Chatterjee, "Why graphics programmers need to know about DRAM," in *SIGGRAPH 2014 Courses*, 2014.

[56] E. Vasiou, K. Shkurko, I. Mallett, E. Brunvand, and C. Yuksel, "A detailed study of ray tracing performance: render time and energy cost," *The Visual Computer*, vol. 34, no. 6, Jun. 2018.

[57] J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heineche, C. Hsieh, J. Goldfeather, J. P. Hultquist, and S. Spach, "PIXEL-PLANES: Building a VLSI based raster graphics system," in *Chapel Hill Conference on VLSI*, 1985.

[58] H. Fuchs, J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, J. F. P. Brooks, J. G. Eyles, and J. Poulton, "Fast spheres, shadows, textures, transparencies, and imgage enhancements in pixel-planes," in *SIGGRAPH '85*, 1985.

[59] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories," in *SIGGRAPH '89*, 1989.

[60] Y. Thonnart and M. Zid, "Technology assessment of silicon interposers for manycore socs: Active, passive, or optical?" in *IEEE Networks on Chip confrence*, ser. NoCs '14, Sep. 2014.

[61] A. Usman, E. Shah, N. B. Satishprasad, J. Chen, S. A. Bohlemann, S. H. Shami, A. A. Eftekhar, and A. Adibi, "Interposer technologies for high-performance applications," *IEEE Trans. on Components, Packaging and Mfc. Tech.*, vol. 7, no. 6, Jun. 2017.

[62] C. Lee, C. Hung, C. Cheung, P. Yang, C. Kao, D. Chen, M. Shih, C. C. Chien, Y. Hsiao, L. Chen, M. Su, M. Alfano, J. Siegel, J. Din, and B. Black, "An overview of the development of a GPU with integrated HBM on silicon interposer," in *IEEE ECTC '16*, May 2016.

[63] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," in *ISCA '17*, 2017.

[64] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Jun. 2017.

[65] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "Cacti-io: Cacti with off-chip power-area-timing models," in *ICCAD '12*, 2012.

[66] K. Shkurko, T. Grant, D. K. E. Brunvand, J. Spjut, E. Vasiou, I. Mallett, and C. Yuksel, "Simtrax: Simulation infrastructure for exploring thousands of cores," in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2018.

[67] J. Kajiya, "The rendering equation," in *SIGGRAPH '86*, 1986.

[68] R. Gonzales and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, 1996.

[69] Z. Zhang, Z. Zhu, and X. Zhang, "Design and optimization of large size and low overhead off-chip caches," *IEEE Trans. on Computers*, vol. 53, no. 7, 2004.

[70] JDEC Standard, "High bandwidth memory (HBM) DRAM," JDEC Solid State Tech. Assn., Tech. Rep. JESD325A, Nov. 2015.

[71] R. Hadidi, B. Asgari, B. A. Mudassar, S. Mukhopadhyay, S. Yalamanchili, and H. Kim, "Demystifying the characteristics of 3d-stacked memories: A case study for hybrid memory cube," in *IISWC*, 2017.

Elena Vasiou received BS degree in mathematics from Davidson College in 2015. She is currently working toward a PhD in computer graphics at the School of Computing at the University of Utah. Her research focuses on ray tracing hardware, energy-efficient graphics accelerators, graphics architectures, and real-time graphics.

Konstantin Shkurko received his PhD in computer graphics from the School of Computing at the University of Utah in 2019. His research interests focus mainly on ray tracing hardware, but also include acceleration structures, rendering algorithms, and scientific visualization.

Erik Brunvand received his PhD from Carnegie Mellon University in 1990. Since then he has been a faculty member in the School of Computing at the University of Utah where his interests include the design of application-specific computers, graphics processors, physical computing, asynchronous systems, VLSI integrated circuit design, and arts/technology collaboration and integration in both research and education.

Cem Yuksel is a faculty member in the School of Computing at the University of Utah. Previously, he was a postdoctoral fellow at Cornell University, after receiving his PhD in Computer Science from Texas A&M University in 2010. His research interests are in computer graphics and related fields, including physically-based simulations, rendering techniques, global illumination, sampling, GPU algorithms, graphics hardware, knitted structures, and hair modeling, animation, and rendering.