CRISP: Concurrent Rendering and Compute Simulation Platform for GPUs

Junrui Pan
Elmore Family School of ECE
Purdue University
West Lafayette, USA
pan251@purdue.edu

Timothy G. Rogers
Elmore Family School of ECE
Purdue University
West Lafayette, USA
timrogers@purdue.edu

***Abstract*—Programmable graphics and compute shaders have blurred the lines between graphics processing and general-purpose computation. APIs such as Asynchronous Compute enable the concurrent execution of raster-based graphics shaders with more general, parallel computation, and emerging graphics and computation-heavy workloads in areas such as augmented and virtual reality can benefit from spatially sharing a GPU.**

Although concurrent execution of graphics rendering and compute kernels are widely used today, contemporary simulators primarily focus on either general compute kernels or rendering pipelines, lacking insight into the challenges and potential opportunities arising from concurrently executing both graphics and compute. To bridge this gap, we present CRISP, a validated cycle-level GPU simulator capable of running graphics rendering and compute kernels concurrently. CRISP extends Accel-Sim by incorporating a detailed programmable graphics pipeline to support Vulkan GLSL Shaders. We demonstrate that CRISP models the rendering pipeline to a high degree of accuracy and highlight the new research opportunities enabled by CRISP through case studies demonstrating the impact of mipmapping, L2 composition using advanced shading techniques, and how prior work on concurrent kernel execution behaves when mixing graphics workloads with compute.

I. Introduction

Graphics processors have significantly evolved from their initial roles as basic 2D display adapters with simple frame-buffer capabilities to sophisticated, high-performance devices. Today, even entry-level GPUs are equipped to perform general-purpose computations concurrently with graphics rendering. As a result, compute acceleration has become a key feature of GPU technology, with applications spanning various fields such as gaming, machine learning, digital content creation, and scientific research.

In contemporary graphics rendering, there are enormous opportunities (physics simulation [12], spatial audio [71], ray tracing [31], etc.) for parallel execution of rendering pipelines and compute kernels. However, adding tasks to the already heavily used GPU can often decrease the frame rate and create performance uncertainty. Therefore, GPUs must be carefully partitioned and tuned for each application to achieve optimal performance.

These advanced techniques result in graphics rendering demanding more general-purpose computation from the GPU. Thus, developers and system designers must consider the rendering pipeline and the intensive algorithms supporting them together. However, the tools available to the community are insufficient to support research on concurrent graphics and compute, and contemporaary raster-based simulators lack support for many of the latest improvements in the graphics pipeline. As shown in Table I, Emerald [40] aimed to provide a unified OpenGL shader simulator for SoC systems. However, it only supports a limited set of instructions and is incapable of studying complex scenes rendered by advanced rendering techniques [1], [26], such as Physically-Based Rendering (PBR) [47]. Other works [67] provide RTL implementations of graphics-enabled GPUs. Existing graphics-only GPU simulators also lack support for advances made in raster-based rendering architecture. For example, Teapot [21] uses a vertex cache to store post-transform vertex attributes to avoid redundant vertex shader invocations, while contemporary GPUs no longer use vertex cache. Instead, they use a batch-based approach where vertex reuse is only considered locally within the same batch [50]. Incorrect baseline assumptions can hide optimization opportunities and lead to potentially incorrect design decisions. Thus, a simulator that accurately models contemporary GPU architecture is needed.

In addition to supporting an updated microarchitecture model, the high-level graphics API used to produce raster-based scenes is evolving. Vulkan (Android's primary low-level graphics API) is increasingly replacing the OpenGL [15] interface Emerald is based upon. Recent work on Vulkan-Sim [65] enables the simulation of Vulkan's ray tracing pipeline; however, it does not support rasterization-based rendering, which is still the dominant rendering mechanism used in contemporary graphics, nor does it support the concurrent execution of rendering and compute kernels.

From a general-purpose computation perspective, compute shaders written in Vulkan have a higher level of abstraction than lower-level GPGPU APIs such as CUDA. As the general-purpose compute demanded by systems requiring real-time rendering evolves, supporting the co-execution of CUDA alongside raster-based rendering will allow developers to leverage the rich set of software written in CUDA to accelerate their general computation. With the support of

Simulator	Rendering Pipeline	Shader Model	GPGPU model	Workloads
Attila [30]	Yes	Unified	No	Rendering
Teapot [21]	Yes	non-Unified	No	Rendering
GLTraceSim [66]	Yes	Approximated	No	Rendering
Emerald [40]	Yes	Unified	No	Rendering
Skybox [67]	Yes	Unified	No	Rendering
Vulkan-Sim [65]	Ray-Tracing only	Ray Tracing	No	Ray Tracing
GPGPU-Sim [24]	No	N/A	Yes	CUDA
Accel-Sim [51]	No	N/A	Yes	CUDA
CRISP	***Yes***	Unified	***Yes***	***Rendering + CUDA***

TABLE I: Comparison of existing simulators. Only CRISP is capable of simultaneously simulating rendering and general computing.

CUDA, researchers can easily evaluate the algorithm and gain insights into the potential problems without rewriting the algorithm with shader language. However, state-of-the-art GPGPU simulators such as GPGPU-Sim [24] and Accel-Sim [51] lack support for any type of graphics rendering.

To enable research on the co-execution of contemporary raster-based graphics shaders and general-purpose computation, we introduce CRISP: a Concurrent Rendering and Compute Simulation Platform for GPUs. CRISP extends the Accel-Sim framework to enable the cycle-level simulation of raster-based Vulkan kernels using the framework's detailed GPU architecture model. With CRISP, Accel-Sim is able to simulate both graphics-only workloads and supports the co-execution of raster-based graphics with the full breadth of software written in CUDA. We demonstrate that our newly developed functional graphics model and timing simulation correlate with two contemporary NVIDIA Ampere GPUs (a Jetson Orin embedded GPU and an RTX 3070 discrete GPU). We demonstrate the research potential of CRISP and the importance of modeling advanced rendering techniques through a series of case studies on the impact of mipmapping, L2 composition using advanced shading techniques, and how prior work on concurrent kernel execution behaves when mixing graphics workloads with compute. To the best of our knowledge, CRISP is the first framework to support the co-execution of graphics shaders alongside CUDA compute kernels.

By supporting CUDA and Vulkan simultaneously, the simulator can also be used to study algorithms that can be offloaded to the GPUs but are not part of the rendering pipeline. Previous works [43], [75], [77], [78] have proposed task-specific custom accelerators for components/algorithms used in the Mixed-Reality (MR) systems. However, MR systems exhibit high computational diversity [42], making it inefficient and impractical to develop custom accelerators for each task. GPUs can be used to run these algorithms, but running the algorithms on the GPUs naively with the rendering workloads causes resource contention and hurts overall performance. We demonstrate that CRISP can be used to model this contention and enable new research on spatially sharing the GPU between graphics kernels and CUDA kernels.

In summary, this paper makes the following contributions:

1) A novel Vulkan unified rendering simulator that supports contemporary shading techniques (vertex batching, mipmapping, Physically-based rendering, etc.).
2) A framework to study concurrent execution of rendering pipeline and general compute kernels using the Accel-Sim architectural model.
3) Two case studies using CRISP that evaluate the cycle-level memory characteristics of graphics rendering workloads and demonstrate the importance of supporting contemporary shader techniques in simulation.
4) Two case studies using CRISP that demonstrate the effectiveness of prior GPGPU kernel mechanisms when applied to concurrent graphics and compute workloads.

II. Background

Graphics rendering in contemporary GPUs has evolved to resemble general computing closely. Developers write shaders in shader languages like GLSL and HLSL to create realistic scenes in real time.

In addition to traditional graphics shaders, compute shaders [18] have been integrated into contemporary graphics APIs to support general-purpose computing. For example, machine learning upscaling is being used to boost real-time rendering performance. In general, the rendering time of a scene scales with the number of pixels rendered. At 4K (3840x2160) resolution, roughly four times the number of pixels need to be rendered compared to the same scene at 2K (2560x1440). To reduce the total number of pixels that need to be shaded, Nvidia's Deep Learning Super Sampling (DLSS) [10] and Intel's Xe Super Sampling [2] renders the scene at a lower resolution and then super samples it to a higher resolution using a deep neural network. DLSS leverages Nvidia's Tensor Cores to accelerate the general matrix multiplication involved in the deep neural network. The latest version of DLSS supports frame generation, which uses spatial and temporal data to generate new frames from previous frames without the CPU submitting new frames. This feature is particularly useful

when a game is CPU-bound, meaning the CPU cannot keep up with the GPU's pace, causing the GPU to idle and wait for new commands from the CPU.

Even though dependencies typically exist between rendering and post-processing, meaning post-processing cannot start until rendering is complete, the rendering pipeline can begin processing the next frame while post-processing operates on the previously rendered image. Contemporary graphics APIs support Asynchronous Compute (async compute), allowing compute shaders to run concurrently with the fragment shaders on the same compute units to exploit parallelism.

For example, DLSS uses tensor cores extensively, and fragment shaders use floating-point units. This makes DLSS post-processing and the rendering pipeline suitable for async compute to maximize system throughput. Async compute is also beneficial when workloads have unbalanced usage. Running a compute-bound workload alongside a memory-bound one helps mitigate bottlenecks and increases overall system throughput.

A. *Concurrent Opportunities in Mixed Reality*

Besides traditional applications, MR (including virtual reality and augmented reality) has gained significant attention in recent years. It has the potential to revolutionize how humans interact with the world and how we communicate with each other. Despite the rendering being much heavier in MR compared to traditional graphics applications, system designers must consider the rendering and the system services required within the system.

In the MR rendering pipeline, asynchronous timewarp [59], [68] is a post-processing method used to reduce Motion-to-Photon (MTP) latency that has been adopted in virtually all state-of-the-art systems. After the scene is rendered, a compute shader is executed to warp the scene to reflect the user's latest position.

Holograms play a transformative role in augmented reality (AR) by seamlessly integrating digital content into the real world. In AR applications, holograms project three-dimensional images that appear to coexist with physical objects, enhancing the user's perception of reality with interactive, lifelike visuals.

System-wise, Visual Inertial Odometry (VIO) is used to track user movement. It differs from the examples mentioned, as VIO is not part of the rendering pipeline and does not operate on the framebuffer at all. The algorithms involved in VIO are classical computer vision algorithms, such as corner detection and feature extraction. These algorithms are highly parallelizable and suitable for GPU acceleration. However, this opportunity has not been exploited in contemporary systems because GPUs are reserved solely for rendering. Each task has its Quality of Service (QoS) requirements. Adding additional tasks without careful runtime management can significantly degrade system QoS.

III. CRISP FRAMEWORK

In this section, we describe the simulator in detail. During runtime, the CPU records commands (draw calls, state changes, resource bindings, etc) and saves them in a command buffer. The Mesa Driver forwards these commands to the simulator, and the simulator saves all parameters needed to render the frame. After all commands needed for one frame are saved, the CPU calls vkQueueSubmit to submit the command buffer to the GPU, which triggers the simulation of the frame.

Trace-driven simulation is most suitable for the scenario we are studying. Execution traces can be collected separately for each task and replayed together to achieve concurrent execution.

The framework overview is shown in Figure 1. Even though Accel-Sim [51] and GPGPU-Sim [24] share the same timing model, the frontend is different. GPGPU-Sim is execution-driven and uses the functional model to execute PTX kernels. Accel-Sim is trace-driven. It consumes SASS traces collected from GPUs using the CUDA tracer built with NVbit [70]. However, the tracer only supports CUDA. To obtain rendering traces, We extend GPGPU-Sim to functionally simulate the rendering pipeline and save the execution traces in SASS. The simulated shaders are obtained from the Vulkan-Sim's [65] translator. Vulkan-Sim introduces a NIR-to-PTX translator to obtain ray tracing shaders from the applications. We extend the translator to support vertex/fragment shaders.

At vkQueueSubmit, the rendering pipeline ❶ executes the drawcalls and saves the executed instruction traces for later use. The traces are saved in SASS which is compatible with Accel-Sim. Each executed PTX is mapped to a SASS instruction and saved with all additional information needed, such as data dependencies and memory addresses referenced by the instruction.

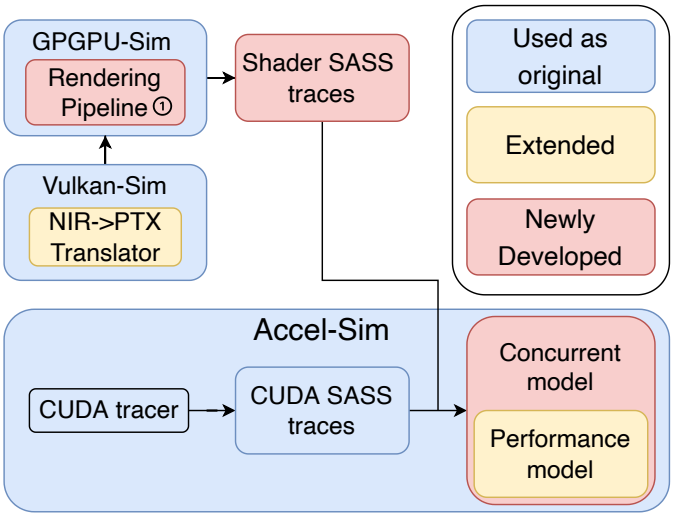


Fig. 1: CRISP Simulator.

In this study, we implement Nvidia's Immediate Tiled Rendering (ITR) [57], as observed in Nvidia's discrete and mobile GPUs. ITR divides the screen into a grid of tiles and then splits primitives into batches. ITR then bins and caches each batch's vertex shading results on-chip before using them immediately for fragment shading.

The pipeline ❶ in Figure III is detailed in Figure 2. Previous works [19], [40], [50], [57] have pointed out that contemporary GPUs use batch-based vertex shading to eliminate duplicate vertices within the batch.

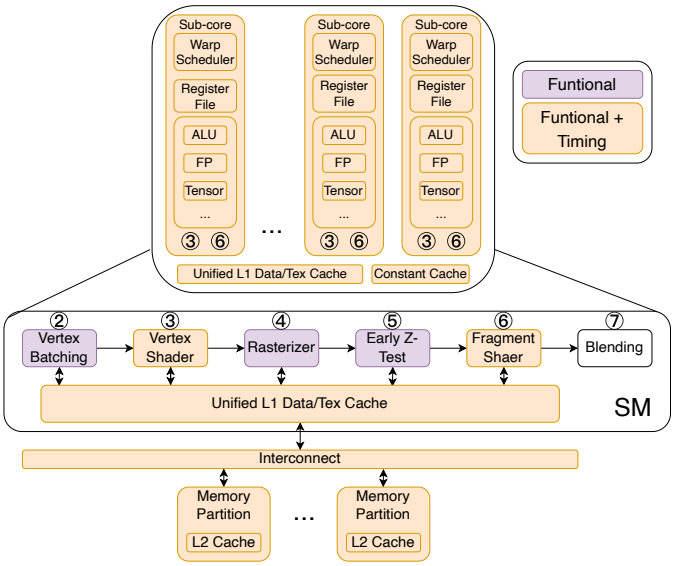


Fig. 2: Rendering pipeline modeled in CRISP.

We adopted vertex batching ❷ and tested the model with incrementing batch size. At *batchsize* = 96, we achieved the highest correlation on vertex shader invocation count, as pointed out by Kerbl et al. [50] as well. The correlation is shown in Figure 3. Drawcalls on the bottom-left of the graph have a slight error because the profiler reports thread count while the simulator's thread count is calculated from warp launched.

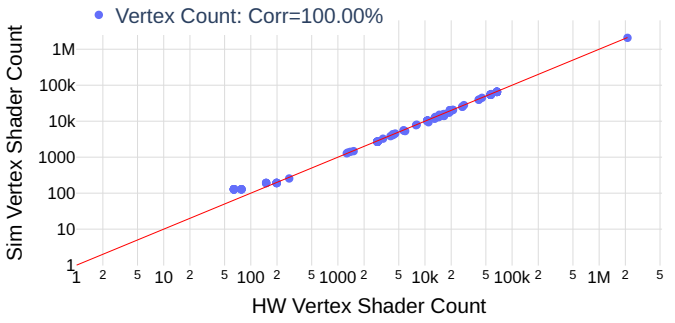


Fig. 3: Vertex shader invocations.

After the vertices are fetched and assembled into batches, each thread runs the geometry transformation defined in the vertex shader ❸ for each vertex on the Streaming Multiprocessors (SMs).

After the vertex shader commits, the results go through Primitive Assembly and Rasterization ❹. Here, clipping and culling are performed to remove the primitives that are not in view. They cannot be seen in the rendered image and do not need to be shaded. Until this point, all intermediate data are buffered on-core. After culling, survived primitives are redistributed based on their position on the screen. This is achieved with the interconnect. The origin SM writes the output attributes to the L2 cache. Then, the rasterizers at the destination SM can proceed.

At Rasterization ❹, primitives are transformed from 3D to 2D and filled with linear interpolation. However, not every pixel may be visible in the final rendered image. The early-Z test eliminates the pixels that are blocked to reduce the total number of pixels that need to be rendered.

Fragment shaders, are invoked for each fragment and executed on the SMs. The shading stage ❻ processes texture values and colors the fragments by executing the shader on the SMs. Mipmapping is a common technique used in computer graphics to reduce the appearance of texture moiré patterns and provide smoother transitions. In Section VI-B, we show that mipmapping support is crucial in modeling correct memory traffic.

Quads, 2x2 pixel squares, are the smallest unit in fragment shaders and are essential for LoD support. LoD is calculated using texture coordinates delta (*ddx*, *ddy*) within a quad. Even though we don't strictly enforce quads in the model, nearby pixels are sorted based on coordinates and grouped into warps based on screen location. Therefore, quads are naturally formed within the warp. Even though texture cache spatial locality is preserved with approximated quads, runtime LoD calculation cannot be done. To solve this, the LoD for each fragment is calculated during rasterization ❹. When a texel is sampled during shader execution ❻, the texture unit looks up the pre-calculated LoD and references the correct mipmap level.

Contemporary GPUs use unified L1 data cache instead of separate texture and data caches [3], [8], [9], [11]. We updated Accel-Sim's cache model to issue all texture load requests to the unified L1 data cache and removed the texture cache.

Accuracy is not the only factor in building simulators. To create a fast and accurate simulator, one must carefully select what to simulate in detail and what to skip [69]. In contemporary GPUs, programmable shaders ❸ ❻ are executed on the SMs, while other stages ❷ ❹ ❺ run on dedicated fixed-function hardware units. Detailed implementations of these units are not publicly documented and have limited effects on the scenarios we are studying. Therefore, we decided to model these stages functionally but not in the performance model. Only the shader stages are modeled in detail in the performance model, while other stages are modeled as black boxes that consume input and write corresponding outputs to the L2 cache to mimic the memory traffic pattern.

The shaders may execute out of order compared to the original API ordering of the triangles. The Render Output Unit (ROP) ❼ performs depth-testing and atomically blends the fragments with the frame buffer. The ROP primarily affects the rendered image visually but has very limited influence on the system we are studying. This motivated us to skip the ROP completely in the model.

A. Concurrent Execution

We used Accel-Sim [51] to collect CUDA traces on silicon with tools created with NVbit [70].

To execute the rendering pipeline concurrently with compute tasks, we leverage the stream support in Accel-Sim and GPGPU-SIM. A stream is a series of commands that execute in order. In Accel-Sim, statistics are aggregated between streams and misleading when concurrent execution is enabled. The performance model is updated to collect stats individually for each stream [62]. Each rendering batch is considered a stream, and the compute kernel's stream is defined in the program and captured from the Accel-Sim tracer.

We extended Accel-Sim and GPGPU-Sim to support advanced GPU partition methods [22], [44]–[46], [49], [72]. By default, the simulator supports concurrent kernel execution but launches thread blocks (CTAs) from one kernel exhaustively before switching to the next kernel. This means if a kernel is large enough and has enough warps to fill all the SMs, there is no concurrent execution. We updated the CTA scheduler and added the following partition methods: Multi-Process Service (MPS), Multi-instance GPU (MiG), and a Fine-grained intra-SM Partition (FG) similar to the async compute feature in Vulkan.

At a high level, MPS and MiG represent coarse-grained inter-SM partitioning methods where each SM is dedicated to either graphics rendering or general computing tasks. In the MPS model, only the SMs are partitioned, while the L2 cache and higher-level memory spaces remain shared across tasks. The MiG model partitions all resources, and each SM only accesses a designated subset of memory sub-partitions and controllers.

In fine-grained intra-SM partitioning (FG), each SM is partitioned to run both tasks. Instead of issuing as many CTAs as possible, the CTA scheduler only issues CTAs within the limits of partitioned resources. Partitioned resources include thread slots, shared memory, and registers. At the CTA issue stage, the CTA scheduler checks the CTA's resource requirements with the remaining resources on the SM. If all resource constraints are met, the CTA is issued. At CTA commit, resources occupied by the CTA are freed and can be used again for future CTAs. However, static partitioning leads to inefficiency since different kernel pairs exhibit divergent characteristics. For example, one kernel may be register-heavy while another uses a lot of shared memory. Therefore, the partition ratio can be changed dynamically to maximize resource utilization. When the partition ratio changes dynamically, and on-chip resources must be reassigned to reflect the updated ratio. For example, each CTA from kernel A has 128 threads, while CTAs from kernel B have 256 threads. Resources freed by one CTA from kernel A are not enough for a CTA from kernel B. In this case, the CTA scheduler stops issuing CTAs from kernel A and waits until two CTAs from kernel A commit, then starts issuing CTAs from kernel B.

IV. Limitations

In this work, we only study partitions of 2 tasks. However, the simulation framework can be easily extended to support more than 2 workloads. As mentioned in Section III, some parts of pipelines are not modeled in the performance model. This may be easily modeled as a FIFO queue with fixed latency. However, the system is usually not bounded by these stages, and the latency can be hidden. These stages execute on dedicated hardware and do not interference with the kernels running on the SMs. The memory traffic is recreated with Load/Stores.

Contemporary graphics shaders are compiled Just-In-Time (JIT), and the exact binary executed by the program depends on the vendor and the platforms. The shader obtained by the framework is from the Mesa Driver, while the shader executed by the GPU is compiled at runtime by Nvidia drivers. This mismatch causes the shaders we simulate to lack some device-specific optimizations. For example, the output variables are duplicated. Once the output is ready, the output is first stored to the duplicated variables' addresses. At the end of the program, the duplicated value is loaded again and stored to the correct outputs' addresses. The Mesa driver failed to identify that the two variables are effectively the same. We also observed variables are loaded without use. For example, three texture coordinates are loaded, but only two are used in the texture reference. We reverse-engineered many of them, but the framework cannot produce a shader that matches silicon exactly without proprietary driver optimization. Furthermore, the simulator runs PTX instead of SASS, which also introduces unavoidable errors. Rendering shaders use undocumented SASS instructions unrelated to CUDA compute kernels, which we can only model at the PTX level.

V. Evaluation Workloads

In this section, we explain in detail the applications and workloads we used in this paper.

A. Rendering Workloads

In theory, the model can support any arbitrary Vulkan application. However, it's hard to support every Vulkan API. We implemented enough APIs to support Godot V4.0, an open-source game engine. Note that we only support Godot V4.0+, as the older versions are OpenGL-based. We evaluated 3 Godot workloads from Godot and Monado [29] for MR: Sponza (SPH) [58], Platformer (PL) [37], and material (MT) [36].

Besides Godot, we also evaluated workloads from the Vulkan Samples repositories, including Sponza (SPL) [39], Pistol (PT) [74], and Planets (IT) [38]. Planets use instanced drawing, which duplicates objects on the scene by merging multiple instances into one draw call. As shown in Figure 5, each asteroid in the image is one instance of the object. The texture used for the object is a 3D texture with multiple layers of 2D texture. An index in the vertex attribute describes the layer of the texture to use. We included this workload because of the unique cache access pattern this workload presents.

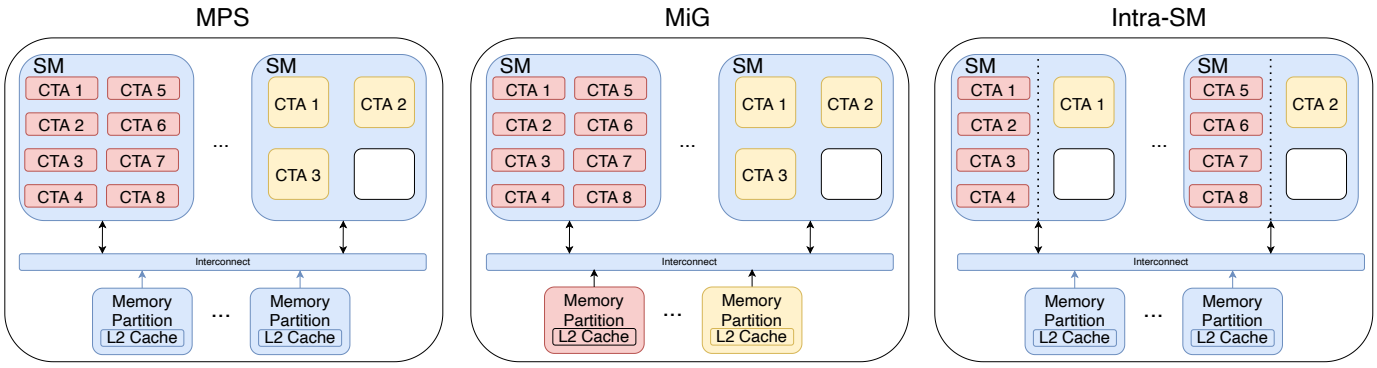


Fig. 4: Partitioning methods supported by the simulator.

The object is duplicated across the scene, and common vertex attributes are referenced repeatedly, which shows temporal locality. Other vertex attributes are unique to each instance, such as position coordinates, which create a streaming access pattern. In Pistol, an antique metallic pistol is rendered using PBR, and eight maps are referenced as textures.

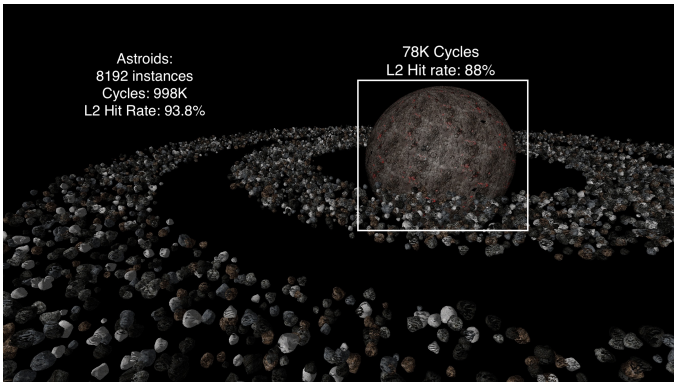


Fig. 5: Planets rendered by the model.

This paper examines two versions of the Sponza scene: one provided by the Godot engine and the other by the Khronos Group's Vulkan Samples. The Godot version uses PBR, whereas the Khronos version employs a simpler shader. For clarity in this paper, the version from Godot is designated as <Sponza PBR> and the Khronos version as <Sponza>.

B. General Compute Workloads

We evaluated system tasks involved in MR systems to determine real-world use scenarios of rendering-compute pairs. We prepared three XR system workloads that are suitable for GPU offloading: visual inertial odometry (VIO), hologram (HOLO), and neural network (NN).

VIO has been widely used on VR headsets [4]–[6], [13], [14]. It uses images captured by the camera and integrates linear acceleration and angular velocity data from IMUs to identify and track distinctive features and landmarks within the environment While in theory it can provide real-time, low latency positional information, recent research [42] shows that the VIO algorithm can only run at 15 Hz on mobile CPUs, which is significantly lower than the required 15-20 ms MTP [33] to prevent user sickness. The system depends on IMU to predict and estimate the user's movement between VIO updates to meet MTP latency requirements. This approach is not without its pitfalls. As the interval between VIO updates lengthens, the IMU integrators' estimation errors accumulate, leading to perceptible drifts in the user's position and orientation. Such inaccuracies also degrade the user experience, potentially causing motion sickness or misalignment between the virtual and physical environments [25].

We profiled state-of-the-art VIO pipelines [35], [64] and concluded that up to 60% of CPU time are computer vision algorithms. These tasks are inherently parallelizable and can be offloaded to GPU for acceleration. By offloading VIOs to the GPU, the system can achieve lower MTP latency and reduce overall system energy consumption, as GPUs are much more efficient in running these algorithms. The pipeline majorly consists of four algorithms: feature detection, distortion, corner detection, and optical flow. We compiled a VIO pipeline using these algorithms from Nvidia Vision Programming Interface [16] with inputs from commonly used datasets [27].

In Virtual reality (VR), eye segmentation provides services such as eye tracking and foveated rendering. Cameras mounted on the headset take photos of the user's eye, and the images are fed through a neural network to segment the eye. In this work, we used RITnet [28], [52], the winning model of OpenEDS Semantic Segmentation Challenge [34]. The network has only 248K parameters, and the model size is less than 1 MB. Most of the layers are CNNs, indicating that this network is memory-bounded. After profiling the network, we concluded that it suffers from small batch size and cannot maintain high occupancy on GPUs. This is caused by the fact that the batch size is fixed to two: one image per eye. Increasing batch size does not make sense, as only the latest image should be used for segmentation. This makes context-switch-based GPU sharing inefficient, as the neural network cannot fully exploit the massive parallelism of the GPU. Even though Accel-Sim is very fast in simulation, RITnet is still too big to be simulated in full. Instead, we used Principle Kernel

TABLE II: Simulation Configurations

	Jetson Orin	RTX 3070
# SMs	14	46
# Registers / SM	65536	
L1 Data Cache + Shared Memory	196KB	128 KB
# Warps / SM	Warps/SM = 64, Schedulers/SM = 4	
# Exec Units	4 FPs, 4 SFUs, 4 INTs, 4 TENSORs	
L2 Cache	4MB	
Compute Core Clock	1300 MHz	1132 MHz
Memory	LPDDR5, 200GB/s	GDDR6, 448GB/s

Selection [23] to select principle kernels that dominate the performance of the NN.

Augment reality (AR) affects our daily lives in content creation, gaming, and education. However, today's mobile devices cannot meet the heavy computing demands for real-time applications. Previous work [80] pointed out that holographic processing [60] is the major bottleneck in AR applications. This motivates us to study holograms using the concurrent model we proposed in this paper.

VI. Methodology

Accel-Sim supports a wide range of Nvidia GPU microarchitectures, from Kepler to Ampere. In this paper, we created two configs. We evaluated the RTX 3070 GPU as a desktop GPU and the Nvidia Jetson Orin as a mobile GPU. Table II shows the GPU configurations used. The hardware stats are collected with Nvidia Nsight Graphics and Nsight Perf SDK [7], [17], [48].

We present a total of 4 case studies, two targeting the rendering pipeline and two targeting concurrent execution.

In rendering pipeline studies, we first show that LoD is crucial in achieving high memory correlation. Then, we show that L2 data composition can vary drastically depending on the shader type and texture format.

In concurrent execution studies, we evaluated two previous works targeting GPU sharing and partitioning. We implemented these designs in the simulator. The L2 composition method used in the previous case study is also used here to evaluate L2 contentions between compute kernels and graphics rendering.

A. Validation

Validating the timing model with the pipelines in existing GPUs is challenging. Nevertheless, correlation versus Nvidia RTX 3070 is shown in Figure 6, and the simulator exhibits a correlation of 94.8% on the graphics applications we outlined in Section V. Each application is sampled at 2K and 4K resolution. In general, the simulated frame time is always longer than the actual hardware, which we suspect is because of the lack of driver optimizations for the shader. Comparing different resolutions, the framework correctly projects the slowdown introduced by extra rendered pixels. For example, IT is vertex-bounded, and only limited fragments are generated for each batch of vertices. Despite 4X more pixels needing to be shaded, scaling from 2K to 4K is only 20% slower.

For SPL, only 4K is shown here because we could not collect reliable frame time from the profiler. The frame time is almost the same at either 2K or 4K resolution, and it runs 2X faster on the Nvidia Jetson Orin than the Nvidia RTX 3070 (0.7ms vs 1.5ms) despite Orin being much smaller compared to the RTX 3070. We suspect the bottleneck is related to the PCI-E bus. The Jetson Orin's GPU is integrated, and CPU-GPU transfer is unnecessary as they share the same memory space, which explains why Jetson is faster despite being a much smaller GPU.

On top of the reasons mentioned in Section IV, we collected execution times on the RTX 3070 and multiplied them by the GPU clock to get cycles, which could introduce additional errors.

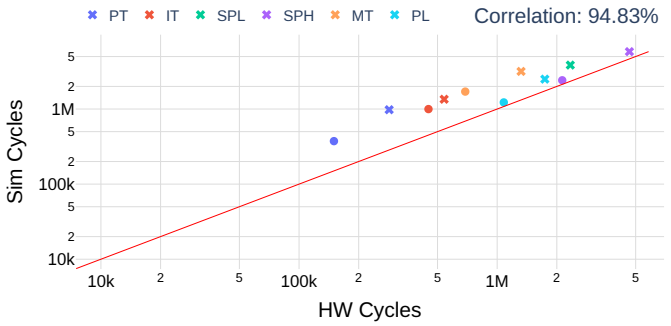


Fig. 6: Correlation between CRISP and Nvidia RTX 3070 GPU. Color represents workloads; circles are 2K, and crosses are 4K.

B. Rendering Pipeline

Level-Of-Detail (LoD): LoD technique decreases the detail and complexity of the texture as distance increases. Each level is down-sampled by half, and using a higher level LoD, more texture accesses collide onto the same texel, decreasing texture accesses and promoting spatial localities. The mipmaps are generated by the driver before execution, and the total number of levels is $\log_2 tex_{dim} + 1$ where tex_{dim} is the dimension of the texture.

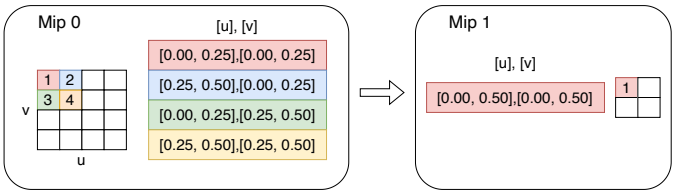


Fig. 7: An example 4x4 texture. Normalized coordinate ranges are shown in the figure. Four texture loads are reduced to one at a mip level 1.

At runtime, a normalized texel coordinate **UV** is provided to the texture unit. The absolute coordinate is obtained by multiplying the selected mipmap's resolution with the normalized coordinate, which is used to calculate the texel address. Then, the texture units filter the texture based on

the method defined by the software (such as linear, nearest, etc.), read the texels, apply the filters, and finally return the sampled texel color.

Fig. 8: Sponza frames comparing LoD effects.

Figure 7 shows how using mipmapping reduces traffic and improves localities with a 4x4 texture. At level 0, There are four requests, each with a **UV** within the range specified. these four requests are merged into one at level 1. Even though the normalized texture coordinates are the same, each texel at level 1 is 2X bigger than level 0.

Figure 9 shows the L1 texture accesses comparing with and without LoD. With LoD enabled, L1 texture access MAPE of the model reduced from 219% to 33%. This is because, with higher mipmaps, more texture instructions collide into the same texel and are merged by the texture unit, thus reducing total texture references. Without LoD, L1 Texture accesses can be off up to 6X, which significantly exaggerates L1 data port pressure. Potential designs targeting bandwidth optimization may be less effective and overlooked because of overestimated L1 bandwidth usage.

Figure 8 shows the zoomed-in Sponza rendered at 2K with LoD on and off. When LoD is disabled, the texture units always reference the original texture (Mip 0). Mipmapping also provides smoother transitions as anti-aliasing happens naturally during the downsampling.

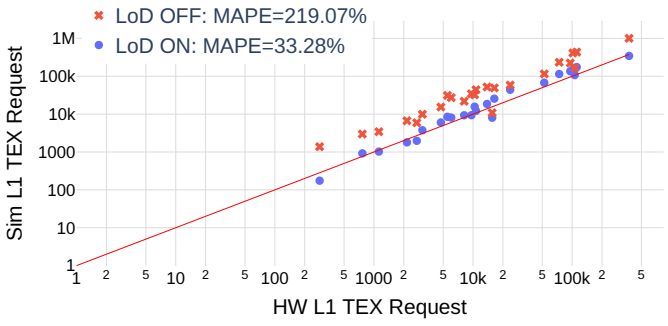


Fig. 9: Texture correlation comparing Lod On and Lod Off. With LoD enabled, the MAPE is reduced by 6.6X.

To further understand the memory characteristics of memory accesses, we analyzed the traces to calculate the number of cache lines (128B/line) referenced by texture instructions for each CTA. Each warp executes the same count of texture instructions, but the number of cache lines referenced in each instruction differs. Figure 10 shows most CTAs referenced 3 to 5 cache lines. This does not imply the working set because there could be accesses to the same cache line across CTAs. After examining all applications, we found that most drawcalls exhibit a distribution similar to Figure 10. The mean can vary from 2.54 to 21.191

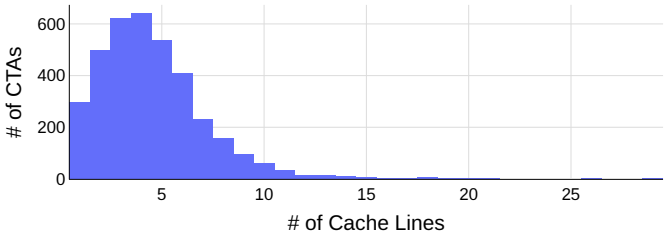


Fig. 10: Historgram showing the number of TEX cache lines per CTA in one drawcall from Sponza.

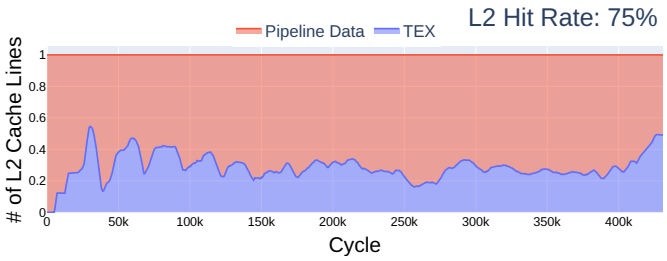
L2 Footprint: The rendering pipeline relies on the caches to communicate between stages. Public Documents [32], [56] show that the runtime scheduler may even limit occupancy intentionally if the intermediate data cannot fit in the caches. Figure 11a shows the L2 breakdown of cached data in Pistol[1], and up to 60% of cachelines are occupied by texture data. On average, 44% of the L2 cache lines are texture data. However, L2 composition varies significantly across applications. As shown in 11b, texture cache lines in Sponza are significantly less than in Pistol. This is because this application uses basic shading instead of PBR shading. In PBR shaders, textures are used as maps. These maps are sampled and used to calculate the lighting effects by determining how light reflects and diffuses physically. In this application, a total of 8 maps are used: irradiance, Bidirectional reflectance distribution function, albedo, normal, prefilter, normal, ambient occlusion, metallic, and roughness. Maps are saved in different formats, and all maps are used to shade the fragment. Compared to Sponza, only one texture is referenced per drawcall. Complexities of PBR shaders are also reflected in the L2 hit rate. In Sponza, the overall L2 hit rate is 90% compared to 75% in Pistol.

C. Graphics + Compute

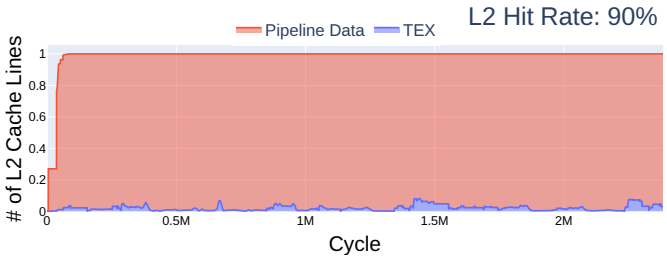
We evaluate two previous works proposed to improve GPU concurrent execution efficiency. TAP [53] uses cache utility score [63] to partition set-associative cache in a heterogeneous system. The key observation is that GPU usually has a much higher cache access rate than a CPU, and using the utility score naively usually favors GPU workloads. In this study, even though both workloads are running on the GPU, we observed a significant mismatch of access rates between the rendering and computing workloads. This observation motivates us to implement TAP in the L2 cache with coarse-grained inter-SM partitioning described in Section III-A.

Warped-Slicer [76] focuses on solving the resource under-utilization issue, which partitions each SM across multiple

[1]The PBR workload includes several draws that are not using PBR. Only PBR drawcalls are shown in the figure



(a) PBR shader.



(b) Basic shader.

Fig. 11: L2 composition comparing difference shading techniques.

kernels. At the beginning of the execution, parallel SMs are used to measure the performance impact of varying CTA counts for each kernel running concurrently in an SM. Then, it uses the water-filling algorithm to find the best partition ratio between two workloads.

Figure 12 shows the performance of the warped-slicer applied on top of the intra-SM partition on the Jetson Orin Platform. The dynamic partition is reset at the new kernel launch for compute kernels and at the new drawcall for rendering workloads. The baseline MPS and EVEN are partitioned evenly between the two workloads. EVEN and Dynamic are both intra-SM. Dynamic's ratio is determined by warped-slicer.

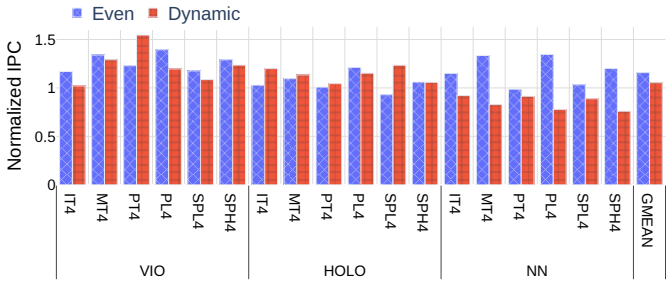


Fig. 12: Performance results of all workloads pairs evaluating warped-slicer. The results are normalized MPS evenly. For Dynamic, the partition ratio is determined by warped-slicer.

Even is the fastest among the three. VIO consists of many small kernels, and the sampling overhead cannot be justified because of that. HOLO is heavily compute-bounded. During the sampling phase, there is no contention as each SM only runs one workload, and HOLO can use all FP units and progress freely. However, running concurrently with the graphics workload causes FP bottlenecks. NN shows the highest speedup when running concurrently. MatMul kernels use shared memory extensively, while rendering uses the remaining L1 as texture cache. By exploring both shared memory and data cache throughput, the overall performance is improved significantly. In this study, most of the workload pairs are big enough to fill the Jetson Orin, and units are already in high usage; thus, the warped-slicer is under-ideal as the sampling mechanism cannot detect contentions.

Figure 13 shows the intra-SM dynamic partition ratio selected by the warped-slicer. Overall it favors rendering shaders over the compute kernels. The low occupancy regions are caused by insufficient registers.

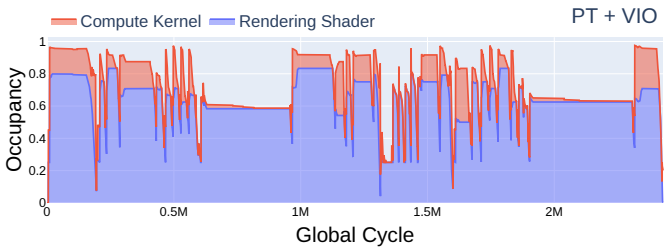


Fig. 13: Realtime occupancy of warped-slicer in PT and VIO. The low occupancy regions are limited by registers.

Figure 14 shows the performance of TAP applied on top of MPS inter-SM sharing on the Nvidia RTX 3070. In both MiG and TAP, SMs are partitioned evenly in coarse-grain (inter-SM). The difference is in how the L2 cache is partitioned. In MiG, each L2 bank is assigned to only one workload (bank-level partitioning). In TAP, L2 banks are all shared among both workloads, and each bank is partitioned by assigning sets to each workload. The ratio is determined by the TAP mechanism. Overall, TAP outperforms MiG and matches the baseline MPS. This indicates all of the workloads-pairs included are bandwidth-bounded, not capacity-bounded. The baseline cache replacement policy, LRU, is efficient enough. As mentioned in the warped-slicer case study, HOLO is extremely compute-bounded. This causes TAP to always favor rendering workloads and assign only 1 set to HOLO kernels. In VIO and NN, the slowdowns in the MiG configuration come from degraded L2 bandwidth. Because only a subset of L2 banks can be used by each workload, the total L2 bandwidth is limited.

To better understand L2 composition, we applied the L2 footprint used in Section VI-B with added compute cache lines. Figure 15 shows L2 composition with compute cache lines. As HOLO does not have much memory access, TAP allocates most cachelines to the rendering pipeline. There is no partition between pipeline data and texture data, as they are both part of the rendering pipeline.

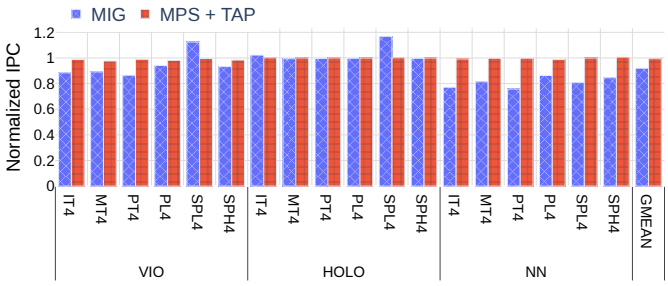


Fig. 14: Performance results of all workloads pairs evaluating TAP. Results are normalized to MPS even.

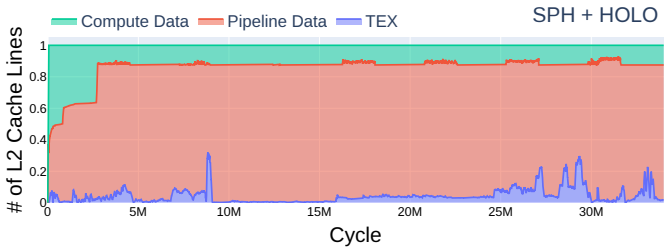


Fig. 15: Normalized L2 composition of TAP. Running workload is Sponza PBR (SPH) + Hologram (HOLO). TAP allocates most cache lines to rendering because HOLO is compute-bounded.

VII. Related work

As illustrated in Table I, even though substantial work has been done in GPU simulation, none of them studies the concurrent execution of graphics and compute kernels.

GPGPU Simulators: GPGPU-Sim [20], [24] is a cycle-level simulator in which we add the rasterization pipeline to generate execution traces. It captures CUDA functional calls to obtain kernels in PTX and accesses the data to perform functional simulation. Similarly, we capture Mesa driver Vulkan APIs to support graphics simulations. Accel-Sim [51] is a trace-driven simulator built on top of the GPGPU-Sim's timing model. Accel-Sim collects SASS execution traces from hardware and replays the traces in the simulator. GPGPU-Sim and Accel-Sim support multi-stream and concurrent execution of different streams. However, as described in Section III-A, neither of them supports GPU partitioning. gem5-gpu [61] incorporates gem5 and GPGPU-Sim to study heterogenous systems. Many other works [41], [54], [73], [79] aim to estimate GPU performance using analytic models. However, analytic models are too high level and not suitable for studying the contention between multiple workloads. More importantly, none of these simulators support graphics rendering.

Graphics Simulator: Attila [30] models the Immediate Mode Rendering (IMR) pipeline using the unified shader model. The custom driver is limited to OpenGL 2.0, which limits the functionality of the pipeline. Teapot [21] models Tile-based Rendering (TBR) and supports more recent OpenGL ES. However, the pipeline is obsolete as it models a non-unified rendering model separating vertex and fragment processors. GLTraceSim [66] does not simulate a detailed GPU model. Instead, it captures API calls and approximates GPU behavior using memory traces collected from the hardware. It cannot study unit contentions observed in concurrent execution. Emerald [40] is a cycle-level OpenGL simulator built on top of GPGPU-Sim and Mesa driver. However, the driver can only handle a limited set of APIs and supports simple shaders. It cannot run the workloads used in this paper. Vulkan-Sim [55], [65] incorporates ray tracing with GPGPU-Sim. It has a versatile translator that generates PTX codes from graphics shaders. Vulkan-Sim aims to study ray tracing and does not support rasterization.

VIII. Conclusion

This work presents CRISP, a framework to study concurrent execution of graphics rendering and compute kernels. CRISP integrates Mesa's Vulkan driver to support state-of-the-art graphics workloads. We incorporated the Immediate Tiled Rendering pipeline with GPGPU-Sim to obtain execution traces of the graphics rendering shaders. Combined with Accel-Sim's GPU instruction tracer, CRISP can simulate both rendering shaders and compute kernels and support flexible GPU partitioning methods as seen on hardware. To our best knowledge, this is the only simulator supporting both types of workloads. We did four case studies to demonstrate the new spaces this work enables. On the graphics rendering pipeline, we show that the Level of Detail affects memory traffic significantly, and memory charismatic differs with the type of shaders used. With concurrent execution, we evaluated two previously proposed GPU partition methods: TAP and warped-slicer. Even though warped-slicer still performs better than serial execution, the sampling mechanism is unable to detect on-chip resource contentions. We applied TAP to the shared L2 cache and compared it with MiG bank-level L2 partitioning. The TAP is faster than MiG while showing no speedup compared to the baseline MPS, which means the workloads are bandwidth-bounded, not compute-bounded.

The framework opens a board new space of research. For future work, we plan to use the model to study the microarchitecture details of concurrent execution on mobile XR platforms. In this work, we only explored the system throughput of concurrent execution. XR workloads have distinct quality-of-service requirements, which must be considered in the system design as well.

Acknowledgment

The authors thank the anonymous reviewers for their feedback. We would like to thank Cesar Avalos for his help in the project. We would also like to thank Shichen Qiao and Matthew D. Sinclair for their work on per-stream stat in GPGPU-Sim. This work was supported, in part, by NSF CCF #1910924 and CCF #1943379 (CAREER), and the Applications Driving Architectures (ADA) Research Center, a JUMP Center cosponsored by SRC and DARPA.

Appendix

A. Abstract

This artifact provides the entire environment to evaluate CRISP. The traces are included or can be collected. Scripts are included to validate the results presented in the paper.

B. Artifact check-list (meta-information)

- **Program:** Accel-Sim framework and Vulkan-Sim
- **Compilation:** gcc/g++, ninja, meson, cmake, cuda
- **Binary:**
- **Model:**
- **Run-time environment:** Ubuntu 20.04
- **Hardware:** Intel CPU with integrated graphics required only if generating Vulkan traces. There is no requirement to run the simulations.
- **Metrics:** Execution time, L2 cache breakdown, cache static analysis.
- **Output:** Simulation results and tables.
- **How much disk space required (approximately)?:** 250G
- **How much time is needed to prepare workflow (approximately)?:** 2 hours
- **How much time is needed to complete experiments (approximately)?:** 8 hours if running graphics only. Up to 7 days if running graphics and compute, depending on the config.
- **Publicly available?:** Yes
- **Workflow framework used?:** Accel-Sim and Vulkan-Sim
- **Archived (provide DOI)?:**

C. Description

This artifact includes the source code for the CRISP simulation framework. On the high level, the framework can be split into two parts: the tracer and the simulator. The tracer requires an intel CPU. All traces evaluated in the paper are provided. Since running all applications is impossible, we by default only enable a subset of the applications.

1) How to access: The source codes can be found here: https://zenodo.org/doi/10.5281/zenodo.12803387 and also here: https://github.com/JRPan/crisp-artifact. We recommend using the Github repo.

2) Hardware dependencies: There are no hardware requirements if only simulations are run with the provided traces. However, generating traces with the tracer requires an Intel CPU with integrated graphics.

3) Software dependencies: The framework is tested on Ubuntu 20.04. The host computer should have docker installed. CUDA is required to run the tracer.

- gcc/g++-9
- CUDA-11 (11.4 tested)
- Embree v3.12.0
- Vulkan SDK 1.2.162 or preferably newer
- docker
- Vulkan Samples.

For Vulkan Samples, you can checkout an earlier commit if you are experiencing CMake version issues. We used 2ce8856.

4) Data sets: All traces evaluated in the paper are provided. However, by default, only SPL paired with VIO is downloaded. If the user wishes to evaluate all workloads used in the paper, please follow the instructions accordingly.

D. Installation

After all software dependencies are met, please run the following to install the remaining dependencies:

```
$ sudo apt install -y build-essential git ninja-build meson libboost-all-dev xutils-dev bison zlib1g-dev flex libglu1-mesa-dev libxi-dev libxmu-dev libdrm-dev llvm libelf-dev libwayland-dev wayland-protocols libwayland-egl-backend-dev libxcb-glx0-dev libxcb-shm0-dev libx11-xcb-dev libxcb-dri2-0-dev libxcb-dri3-dev libxcb-present-dev libxshmfence-dev libxxf86vm-dev libxrandr-dev libglm-dev
```

Download and decompress the source codes: https://zenodo.org/records/12803388 The source code contains 3 folders:

- **accel-sim-framework:** the simulator.
- **vulkan-sim:** the tracer.
- **mesa-vulkan-sim:** the Mesa 3D driver.

Build the Simulator within the docker (from the crisp-framework folder):

```
$ docker run -it --rm -v $(pwd)/accel-sim-framework:/accel-sim/accel-sim-framework tgrogers/accel-sim_regress:Ubuntu-22.04-cuda-11.7
$ cd accel-sim-framework
$ source gpu_simulator/setup_environment
$ make -j -C ./gpu-simulator
$ exit
```

Finally, copy files gpgpusim.config and config_turing_islip.icnt from the crisp-framework folder to Vulkan-Samples folder.

E. Experiment workflow

To run the simulation,

```
$ docker run -it --rm -v $(pwd)/accel-sim-framework:/accel-sim/accel-sim-framework tgrogers/accel-sim_regress:Ubuntu-22.04-cuda-11.7
$ cd accel-sim-framework
$ . get_crisp_traces.sh
$ cd util/graphics
$ python3 ./setup_concurrent.py
$ cd ../../
$ . run.sh
```

The user can use ./util/job_launching/job_status.py to monitor the simulation. The simulations are expected to run for 8 hours. After simulations are finished, the results are included in folder sim_run_11.7

To collect the stats, simply run the following command the exit the docker image.

```
$ . collect.sh
$ exit
```

Several CSV files should be generated under the accel-sim-framework folder. These files contain simulation statistics such as execution cycles and cache hit rates.

(optional) To collect traces, first setup environments:

```
$ export CUDA_INSTALL_PATH=/usr/local/cuda
$ cd crisp-framework
$ source vulkan-sim/setup_environment
```

Build the tracer. Please ignore the error in the first ninja build:

```
$ cd mesa-vulkan-sim
$ meson --prefix="${PWD}/lib" build/
$ meson configure build/ -Dbuildtype=debug -D
b_lundef=false -Dgallium-drivers=swrast
$ ninja -C build/ install
$ cd ../vulkan-sim/
$ make -j
$ cd ../mesa-vulkan-sim
$ ninja -C build/ install
```

Execute the command from the Vulkan-Sample folder:

```
$ VULKAN_APP=render_passes ./build/linux/app/bin
/Release/x86_64/vulkan_samples sample
render_passes
```

Then wait for the tracer to finish. The resolution has been changed to 480P to speed up the process. After the process is finished, a file called complete.traceg should be generated in the working directory.

Then, from accel-sim-framework/util/graphics folder, edit line 7 to the compelete.g file, and optionally edit line 4 to change the output folder name.

```
$ python3 ./process-vulkan-traces.py
```

F. Evaluation and expected results

After completing the previous steps, you should have the following CSV files under accel-sim-framework: render_passes_2k.csv and render_passes_2k_lod0.csv.

The CSV files generated in the previous section contain all data used in this paper. The following scripts are used to generate figures in Section VI. However, the configurations and workload pairs are not the same, thus the generated figures may not look exactly the same as the paper.

A requirement.txt file is provided under accel-sim-framework's root folder. It's recommended to create a clean virtualenv and install the dependencies.

To generate the L1 texture plot similar to Figure 9,

```
$ python3 ./util/graphics/l1tex.py
```

To generate the L2 breakdown plot similar to Figure 11a, change ./util/graphics/l2breakdown.py::7 to match the visualizer log file. The log files are generated under sim_run* folders.

```
$ python3 ./util/graphics/l2breakdown.py
```

Depending on the workload, this figure may or may not include compute data. The generated figure may look different from the paper because of different simulation configurations.

To generate the concurrent ratio plot similar to Figure 13, change ./util/graphics/concurrent_ratio.py::7 to match the visualizer log. For this one, please choose the one under sim_run_11.7/render_passes_2k/all1/RTX3070-SASS-concurrent-fg-VISUAL.

```
$ python3 ./util/graphics/concurrent_ratio.py
```

Figure 13 used the warped-slicer to partition the GPU dynamically. However, this configuration is not included in the artifact due to extended simulation time. The partition ratio used here is fixed.

We provided a .ipynb notebook at util/graphics/working_set.ipynb to perform static analysis as seen in Figure 10. The figure may look different depending on the drawcall you choose.

G. Experiment customization

Customization can be done by adjusting the GPU configuration file. To do so, please follow the accel-sim framework's README, or modify the configs under accel-sim-framework/gpu-simulator/gpgpu-sim/configs/tested-cfgs directly. CRISP supports the simulation of Vulkan applications. Users can trace and simulate Vulkan applications that are not included in the artifact.

H. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae

References

[1] "Counter-Strike 2: Leveling Up The World - YouTube." [Online]. Available: https://www.youtube.com/watch?v=ExZtlSgOxEQ

[2] "Intel® Arc™- Xe Super Sampling." [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/arc/technology/xess.html

[3] "Introcuding RDNA Architecture." [Online]. Available: https://www.amd.com/system/files/documents/rdna-whitepaper.pdf

[4] "Meta Quest 2: Immersive All-In-One VR Headset | Meta Store | Meta Store." [Online]. Available: https://www.meta.com/quest/products/quest-2/

[5] "Meta Quest 3: New Mixed Reality VR Headset - Shop Now | Meta Store | Meta Store." [Online]. Available: https://www.meta.com/quest/quest-3/

[6] "Meta Quest Pro: Our Most Advanced New VR Headset | Meta Store." [Online]. Available: https://www.meta.com/quest/quest-pro/

[7] "Nsight Graphics." [Online]. Available: https://developer.nvidia.com/nsight-graphics

[8] "NVIDIA ADA GPU ARCHITECTURE." [Online]. Available: https://images.nvidia.com/aem-dam/Solutions/Data-Center/l4/nvidia-ada-gpu-architecture-whitepaper-v2.1.pdf

[9] "NVIDIA AMPERE GA102 GPU ARCHITECTURE." [Online]. Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

[10] "NVIDIA DLSS Technology." [Online]. Available: https://www.nvidia.com/en-us/geforce/technologies/dlss/

[11] "NVIDIA TURING GPU ARCHITECTURE." [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[12] "PhysX SDK - Latest Features & Libraries | NVIDIA | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/physx-sdk

[13] "PICO 4 All-In-One VR Headset | PICO Singapore." [Online]. Available: https://www.picoxr.com/sg/products/pico4

[14] "PlayStation®VR2 | The next generation of VR gaming on PS5." [Online]. Available: https://www.playstation.com/en-us/ps-vr2/

[15] "Use Vulkan for graphics | Android game development." [Online]. Available: https://developer.android.com/games/develop/vulkan/overview

[16] "VPI - Vision Programming Interface: Main Page." [Online]. Available: https://docs.nvidia.com/vpi/index.html

[17] "Nsight Perf SDK," Mar. 2021. [Online]. Available: https://developer.nvidia.com/nsight-perf-sdk

[18] "Getting Started with Vulkan Compute Acceleration," Feb. 2023, section: Blogs. [Online]. Available: https://www.khronos.org/blog/getting-started-with-vulkan-compute-acceleration

[19] N. Agrawal, A. Jain, D. Kirkland, K. Abdalla, Z. Hakura, and H. Kethareswaran, "Distributed index fetch, primitive assembly, and primitive batching," US Patent US20 170 178 401A1, Jun., 2017. [Online]. Available: https://patents.google.com/patent/US20170178401A1/en

[20] A. Ariel, W. W. L. Fung, A. E. Turner, and T. M. Aamodt, "Visualizing complex dynamics in many-core accelerator architectures," in *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, Mar. 2010, pp. 164–174.

[21] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis, "TEAPOT: a toolset for evaluating performance, power and image quality on mobile graphics systems," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. Eugene Oregon USA: ACM, Jun. 2013, pp. 37–46. [Online]. Available: https://dl.acm.org/doi/10.1145/2464996.2464999

[22] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. Williamsburg VA USA: ACM, Mar. 2018, pp. 503–518. [Online]. Available: https://dl.acm.org/doi/10.1145/3173162.3173169

[23] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal Kernel Analysis: A Tractable Methodology to Simulate Scaled GPU Workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Virtual Event Greece: ACM, Oct. 2021, pp. 724–737. [Online]. Available: https://dl.acm.org/doi/10.1145/3466752.3480100

[24] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, Apr. 2009, pp. 163–174. [Online]. Available: https://ieeexplore.ieee.org/document/4919648

[25] S. Baruah and J. Haritsa, "Scheduling for overload in real-time systems," *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1034–1039, Sep. 1997, conference Name: IEEE Transactions on Computers.

[26] B. Burley, "Physically-Based Shading at Disney."

[27] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, Sep. 2016, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364915620033

[28] A. K. Chaudhary, R. Kothari, M. Acharya, S. Dangi, N. Nair, R. Bailey, C. Kanan, G. Diaz, and J. B. Pelz, "RITnet: Real-time Semantic Segmentation of the Eye for Gaze Tracking," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct. 2019, pp. 3698–3702, arXiv:1910.00694 [cs, eess]. [Online]. Available: http://arxiv.org/abs/1910.00694

[29] Collabora, "Monado - Open Source XR Runtime." [Online]. Available: https://monado.dev

[30] V. Del Barrio, C. Gonzalez, J. Roca, and A. Fernandez, "ATTILA: a cycle-level execution-driven simulator for modern GPU architectures," in *2006 IEEE International Symposium on Performance Analysis of Systems and Software*. Austin, TX, USA: IEEE, 2006, pp. 231–241. [Online]. Available: http://ieeexplore.ieee.org/document/1620807/

[31] Y. Deng, Y. Ni, Z. Li, S. Mu, and W. Zhang, "Toward Real-Time Ray Tracing: A Survey on Hardware Acceleration and Microarchitecture Techniques," *ACM Computing Surveys*, vol. 50, no. 4, pp. 1–41, Jul. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3104067

[32] R. Dimitrov and Z. S. Hakura, "Hierarchical tiled caching," US Patent US9 779 533B2, Oct., 2017. [Online]. Available: https://patents.google.com/patent/US9779533B2/en?q=HIERARCHICAL+TILED+CACHING&oq=HIERARCHICAL+TILED+CACHING

[33] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward Low-Latency and Ultra-Reliable Virtual Reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, Mar. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8329628/

[34] S. J. Garbin, O. Komogortsev, R. Cavin, G. Hughes, Y. Shen, I. Schuetz, and S. S. Talathi, "Dataset for Eye Tracking on a Virtual Reality Platform," in *ACM Symposium on Eye Tracking Research and Applications*, ser. ETRA '20 Full Papers. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3379155.3391317

[35] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A Research Platform for Visual-Inertial Estimation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 4666–4672, iSSN: 2577-087X. [Online]. Available: https://ieeexplore.ieee.org/document/9196524

[36] Godot, "Material Testers." [Online]. Available: https://github.com/godotengine/godot-demo-projects/tree/master/3d/material_testers

[37] ——, "Platformer 3D." [Online]. Available: https://github.com/godotengine/godot-demo-projects/tree/master/3d/platformer

[38] K. Group, "Instancing." [Online]. Available: https://github.com/KhronosGroup/Vulkan-Samples/blob/main/samples/api/README.adoc

[39] ——, "Render Passes." [Online]. Available: https://github.com/KhronosGroup/Vulkan-Samples/blob/main/samples/performance/README.adoc

[40] A. A. Gubran and T. M. Aamodt, "Emerald: graphics modeling for SoC systems," in *Proceedings of the 46th International Symposium on Computer Architecture*. Phoenix Arizona: ACM, Jun. 2019, pp. 169–182. [Online]. Available: https://dl.acm.org/doi/10.1145/3307650.3322221

[41] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model."

[42] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "ILLIXR: Enabling End-to-End Extended Reality Research," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. Storrs, CT, USA: IEEE, Nov. 2021, pp. 24–38. [Online]. Available: https://ieeexplore.ieee.org/document/9668280/

[43] Y. Jing, Y. Sun, X. Wang, W. Zhao, M. Wu, F. Yan, Y. Ma, L. Ye, and T. Jia, "DCIM-3DRec: A 3D Reconstruction Accelerator with Digital Computing-in-Memory and Octree-Based Scheduler," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. Vienna, Austria: IEEE, Aug. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10244465/

[44] A. Jog, E. Bolotin, Z. Guz, M. Parker, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Application-aware Memory System for Fair and Efficient Execution of Concurrent GPGPU Applications," in *Proceedings of Workshop on General Purpose Processing Using GPUs*. Salt Lake City UT USA: ACM, Mar. 2014, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/2588768.2576780

[45] A. Jog, O. Kayiran, T. Kesten, A. Pattnaik, E. Bolotin, N. Chatterjee, S. W. Keckler, M. T. Kandemir, and C. R. Das, "Anatomy of GPU Memory System for Multi-Application Execution," in *Proceedings of the 2015 International Symposium on Memory Systems*. Washington DC DC USA: ACM, Oct. 2015, pp. 223–234. [Online]. Available: https://dl.acm.org/doi/10.1145/2818950.2818979

[46] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated Scheduling and Prefetching for GPGPUs."

[47] B. Karis and E. Games, "Real Shading in Unreal Engine 4," vol. 4, no. 3, p. 1, 2013.

[48] B. Karlsson, "RenderDoc," Jul. 2024, original-date: 2014-02-27T15:16:30Z. [Online]. Available: https://github.com/baldurk/renderdoc

[49] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing GPU concurrency in heterogeneous architectures," in *2014 47th annual IEEE/ACM international symposium on microarchitecture*. IEEE, 2014, pp. 114–126. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7011382/

[50] B. Kerbl, M. Kenzel, E. Ivanchenko, D. Schmalstieg, and M. Steinberger, "Revisiting The Vertex Cache: Understanding and Optimizing Vertex Processing on the modern GPU," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 1, no. 2, pp. 1–16, Aug. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3233302

[51] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, May 2020, pp. 473–486. [Online]. Available: https://ieeexplore.ieee.org/document/9138922/

[52] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda, C. Banbury, M. Mazumder, L. Lai, A. Sirasao, T. Krishna, H. Khaitan, V. Chandra, and V. J. Reddi, "XRBench: An Extended Reality (XR) Machine Learning Benchmark Suite for the Metaverse," May 2023, arXiv:2211.08675 [cs]. [Online]. Available: http://arxiv.org/abs/2211.08675

[53] J. Lee and H. Kim, "TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture," in *IEEE International Symposium on High-Performance Comp Architecture*, Feb. 2012, pp. 1–12, iSSN: 2378-203X.

[54] J. Lee, Y. Ha, S. Lee, J. Woo, J. Lee, H. Jang, and Y. Kim, "GCoM: a detailed GPU core model for accurate analytical modeling of modern GPUs," in *Proceedings of the 49th Annual International Symposium on Computer*

Architecture. New York New York: ACM, Jun. 2022, pp. 424–436. [Online]. Available: https://dl.acm.org/doi/10.1145/3470496.3527384

[55] L. Liu, M. Saed, Y. H. Chou, D. Grigoryan, T. Nowicki, and T. M. Aamodt, "LumiBench: A Benchmark Suite for Hardware Ray Tracing," in *2023 IEEE International Symposium on Workload Characterization (IISWC)*, Oct. 2023, pp. 1–14, iSSN: 2835-2238. [Online]. Available: https://ieeexplore.ieee.org/document/10289559

[56] M. Mantor, L. Lefebvre, M. Alho, M. Tuomi, and K. Kallio, "Hybrid render with preferred primitive batch binning and sorting," US Patent US20 210 209 831A1, Jul., 2021. [Online]. Available: https://patents.google.com/patent/US20210209831A1/en?q=depth+culling&q=early+z&q=culling&assignee=amd+inc,Advanced+Micro+Devices&language=ENGLISH&sort=new

[57] M. Mantor, L. Lefebvre, M. Fowler, T. Kelley, M. Alho, M. Tuomi, K. Kallio, P. K. R. Buss, J. A. Komppa, and K. Tuomi, "Hybrid render with deferred primitive batch binning," US Patent US20 220 277 508A1, Sep., 2022. [Online]. Available: https://patents.google.com/patent/US20220277508A1/en?q=depth+culling&q=early+z&q=culling&assignee=amd+inc,Advanced+Micro+Devices&language=ENGLISH&sort=new

[58] Monado, "Sponza demo for Godot 3," Dec. 2019. [Online]. Available: https://gitlab.freedesktop.org/monado/demos/godot-sponza-openxr

[59] T. C. Nguyen, S. Kim, J.-H. Son, and J.-H. Yun, "Selective Timewarp Based on Embedded Motion Vectors for Interactive Cloud Virtual Reality," *IEEE Access*, vol. 7, pp. 3031–3045, 2019, conference Name: IEEE Access.

[60] M. Persson, D. Engström, and M. Goksör, "Real-time generation of fully optimized holograms for optical trapping applications," in *Optical Trapping and Optical Micromanipulation VIII*, vol. 8097. SPIE, Sep. 2011, pp. 291–299. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/8097/80971H/Real-time-generation-of-fully-optimized-holograms-for-optical-trapping/10.1117/12.893599.full

[61] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A Heterogeneous CPU-GPU Simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, Jan. 2015, conference Name: IEEE Computer Architecture Letters. [Online]. Available: https://ieeexplore.ieee.org/document/6709764

[62] S. Qiao, X. Su, and M. D. Sinclair, "Integrating Per-Stream Stat Tracking into Accel-Sim," Sep. 2023, arXiv:2304.11136 [cs]. [Online]. Available: http://arxiv.org/abs/2304.11136

[63] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec. 2006, pp. 423–432, iSSN: 2379-3155.

[64] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," Mar. 2020, arXiv:1910.02490 [cs]. [Online]. Available: http://arxiv.org/abs/1910.02490

[65] M. Saed, Y. H. Chou, L. Liu, T. Nowicki, and T. M. Aamodt, "Vulkan-Sim: A GPU Architecture Simulator for Ray Tracing," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2022, pp. 263–281.

[66] A. Sembrant, T. E. Carlson, E. Hagersten, and D. Black-Schaffer, "A graphics tracing framework for exploring CPU+GPU memory systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*. Seattle, WA: IEEE, Oct. 2017, pp. 54–65. [Online]. Available: http://ieeexplore.ieee.org/document/8167756/

[67] B. Tine, V. Saxena, S. Srivatsan, J. R. Simpson, F. Alzammar, L. Cooper, and H. Kim, "Skybox: Open-Source Graphic Rendering on Programmable

[69] O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, N. Chatterjee, N. Jiang, and D. Nellans, "Need for Speed: Experiences Building a Trustworthy System-Level GPU Simulator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Seoul, Korea (South): IEEE, Feb. 2021, pp. 868–880. [Online]. Available: https://ieeexplore.ieee.org/document/9407154/

RISC-V GPUs," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. Vancouver BC Canada: ACM, Mar. 2023, pp. 616–630. [Online]. Available: https://dl.acm.org/doi/10.1145/3582016.3582024

[68] J. M. P. van Waveren, "The asynchronous time warp for virtual reality on consumer hardware," in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, ser. VRST '16. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 37–46. [Online]. Available: https://dl.acm.org/doi/10.1145/2993369.2993375

[70] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 372–383. [Online]. Available: https://doi.org/10.1145/3352460.3358307

[71] C. Wakeland and L. Antani, "POWERING SPATIAL AUDIO ON GPUS THROUGH HARDWARE, SOFTWARE, AND TOOLS," 2019.

[72] H. Wang, F. Luo, M. Ibrahim, O. Kayiran, and A. Jog, "Efficient and fair multi-programming in GPUs via effective bandwidth management," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 247–258. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8327013/

[73] X. Wang, K. Huang, A. Knoll, and X. Qian, "A Hybrid Framework for Fast and Accurate GPU Performance Estimation through Source-Level Analysis and Trace-Based Simulation," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2019, pp. 506–518, iSSN: 2378-203X. [Online]. Available: https://ieeexplore.ieee.org/document/8675207

[74] S. Willems, "pbrtexture." [Online]. Available: https://github.com/SaschaWillems/Vulkan/tree/master/examples/pbrtexture

[75] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "PIM-VR: Erasing Motion Anomalies In Highly-Interactive Virtual Reality World with Customized Memory Cube," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Washington, DC, USA: IEEE, Feb. 2019, pp. 609–622. [Online]. Available: https://ieeexplore.ieee.org/document/8675230/

[76] Q. Xu, H. Jeon, K. Kim, W. W. Ro, and M. Annavaram, "Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. Seoul, South Korea: IEEE, Jun. 2016, pp. 230–242. [Online]. Available: http://ieeexplore.ieee.org/document/7551396/

[77] Z. Ying, S. Bhuyan, Y. Kang, Y. Zhang, M. T. Kandemir, and C. R. Das, "EdgePC: Efficient Deep Learning Analytics for Point Clouds on Edge Devices," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. Orlando FL USA: ACM, Jun. 2023, pp. 1–14. [Online]. Available: https://dl.acm.org/doi/10.1145/3579371.3589113

[78] H. You, Y. Zhao, C. Wan, Z. Yu, Y. Fu, J. Yuan, S. Wu, S. Zhang, Y. Zhang, C. Li, V. Boominathan, A. Veeraraghavan, Z. Li, and Y. C. Lin, "EyeCoD: Eye Tracking System Acceleration via FlatCam-Based Algorithm and Hardware Co-Design," *IEEE Micro*, vol. 43, no. 4, pp. 88–97, Jul. 2023, conference Name: IEEE Micro. [Online]. Available: https://ieeexplore.ieee.org/document/10123119

[79] Y. Zamora, B. Lusch, M. Emani, V. Vishwanath, I. Foster, and H. Hoffmann, "Cross Architecture Performance Prediction."

[80] S. Zhao, H. Zhang, C. S. Mishra, S. Bhuyan, Z. Ying, M. T. Kandemir, A. Sivasubramaniam, and C. Das, "HoloAR: On-the-fly Optimization of 3D Holographic Processing for Augmented Reality," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Virtual Event Greece: ACM, Oct. 2021, pp. 494–506. [Online]. Available: https://dl.acm.org/doi/10.1145/3466752.3480056