PIM-VR: Erasing Motion Anomalies In Highly-Interactive Virtual Reality World With Customized Memory Cube

Chenhao Xie*, Xingyao Zhang*, Ang Li†, Xin Fu*, Shuaiwen Leon Song†
*ECMOS Lab, ECE Department, University of Houston
†HPC group, Pacific Northwest National Lab (PNNL)
*cxie@uh.edu, *xzhang55@uh.edu, †ang.li@pnnl.gov, *xfu8@central.uh.edu, †Shuaiwen.Song@pnnl.gov

***Abstract*—With the revolutionary innovations emerging in the computer graphics domain, virtual reality (VR) has become increasingly popular and shown great potential for entertainment, medical simulation and education. In the highly interactive VR world, the motion-to-photon delay (MPD) which represents the delay from users' head motion to the responded image displayed on their head devices, is the most critical factor for a successful VR experience. Long MPD may cause users to experience significant motion anomalies: judder, lagging and sickness. In order to achieve the short MPD and alleviate the motion anomalies, asynchronous time warp (ATW) which is known as an image re-projection technique, has been proposed by VR vendors to map the previously rendered frame to the correct position using the latest head-motion information. However, after a careful investigation on the efficiency of the current GPU-accelerated ATW through executing real VR applications on modern VR hardware, we observe that the state-of-the-art ATW technique cannot deliver the ideal MPD and often misses the refresh deadline, resulting in reduced frame rate and motion anomalies. This is caused by two major challenges: inefficient VR execution model and intensive off-chip memory accesses. To tackle these, we propose a preemption-free Processing-In-Memory based ATW design which asynchronously executes ATW within a 3D-stacked memory, without interrupting the rendering tasks on the host GPU. We also identify a redundancy reduction mechanism to further simplify and accelerate the ATW operation. A comprehensive evaluation of our proposed design demonstrates that our PIM-based ATW can achieve the ideal MPD and provide superior user experience. Finally, we provide a design space exploration to showcase different design choices for the PIM-based ATW design, and the results show that our design scales well in future VR scenarios with higher frame resolution and even lower ideal MPD.**

I. INTRODUCTION

Driven by the significant performance improvement of computing hardware and the revolution of graphics and display technologies, virtual reality (VR) which was invented back in 1990's is now experiencing a rapid growth. It is becoming a popular product to be deployed in many fields, including entertainment, medical simulation, industrial design, education, etc [1], [2], [3]. Currently, several major technology vendors have already published their own VR devices such as Oculus Rift [4], HTC vive [5], PlayStation VR [6]; and numerous cooperations and start-ups have invested specifically for VR related technologies [7]. The global VR market size is expected to grow exponentially from 3.9 billion in 2017 to 33.9 billion by 2022 [8].

Comparing with the traditional PC and mobile graphics applications, VR technologies provide users a fully immersive environment by displaying images directly in front of their eyes. To "cheat" human visual system, the display images must cover a broad field of view (FoV) which requires a high display resolution [9]. Meanwhile, the images with high resolutions must be delivered within an extremely short period so that users cannot perceive high *motion-to-photon delay (MPD)*, which represents the duration from users' head motion to the responded image displayed on their head devices. Typically, long MPD causes users to experience significant motion anomalies: judder, lagging and sickness [10]. However, due to the current hardware limitations, the state-of-the-art VR systems fail to achieve unnoticeable MPD even with significantly sacrificed image resolution, resulting in unsatisfied user experience [11].

To help reduce MPD in modern VR systems, asynchronous time warp (ATW) [12] which is known as an image re-projec-tion technique has been proposed and employed by VR vendors to *map* the previously rendered frame to the correct position using the latest head mounted device (HMD) information instead of *regenerating* a new frame. The state-of-the-art ATW is typically implemented on GPU cores to gain the maximum performance [13]; and to update images, ATW requires frequent sampling in the frame buffer located in graphics processing units' (GPUs) off-chip memory. The current GPU-accelerated ATW still faces two major challenges. **(1) Inefficient VR execution model.** Since GPU is commonly used for rendering frames, inserting ATW operations to GPU workloads causes inevitable resource competition and long unpredictable preemption overheads, thus further increasing MPD. **(2) Intensive off-chip memory accesses.** The frame size is much larger than the GPU's on-chip storage, and frequently loading the sizable frames from the off-chip memory causes substantial performance and energy costs on data movements. In summary, although ATW is considered as an effective method to reduce visual anomalies in VR systems [14], [15], [16], [17], [18], we observe that the current ATW design on commercial hardware cannot achieve the ideal MPD and often causes ATW to miss refresh deadlines (Section II).

In recent years, the emerging 3D stacked memory technologies such as hybrid memory cube (HMC) offer great opportunities to conduct processing-in-memory (PIM) which offloads data intensive tasks to the embedded logic layer of the memory. PIM-based architecture designs have recently been explored to reduce the communication overhead between the host and the memory, achieving significant performance improvement [19], [20], [21], [22], [23]. Additionally, HMC also provides a potential opportunity to execute the task concurrently with the host processor, demonstrating great potential to solve the asynchronous challenge in the current GPU-accelerated ATW design.

2378-203X/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCA.2019.00013

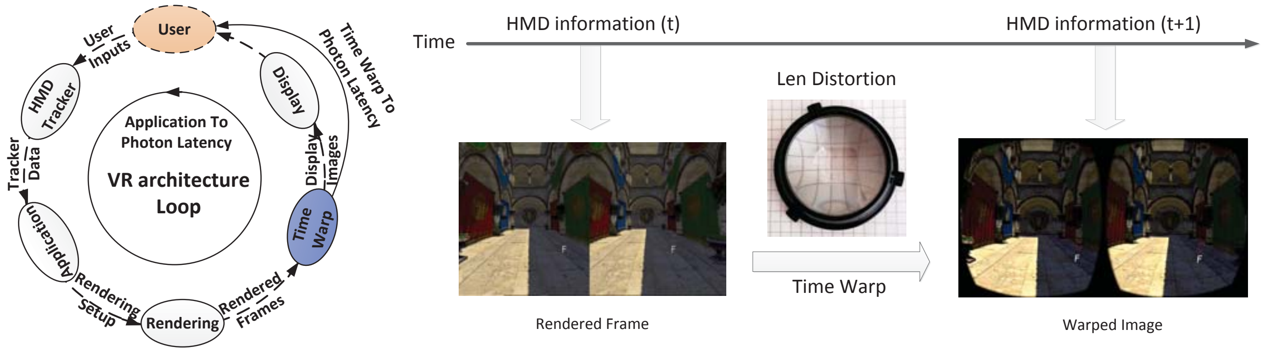


Figure 1: *Left*: Overview of the state-of-the-art VR workflow. *Right*: How time warp helps correct lens distortion and predict the view position of the displayed image using the latest HMD information.

Motivated by the above two features of PIM-based memory, we propose to conduct the ATW operations on the logic layer of 3D stacked memory, thus addressing the two ATW challenges in the current GPU design and dramatically shrinking the MPD for superior user experience. The PIM-based ATW needs to fulfill an important design constraint: the working power of the HMC design must be lower than the maximum power density that can be tolerated by the 3D stacked technology. This makes it infeasible to simply executing ATW as its original openGL kernel in HMC which requires to integrate the GPU SMs and other related function units (e.g., texture and rasterization units) into the HMC logic layer. On the other hand, a small core (e.g., an in-order CPU core) design in the HMC logical layer does not provide enough computation throughput required by the ATW operation under the refresh deadline constraint. The main design target in this study is how to effectively leverage the 3D stacked architecture to design a customized memory cube for efficient ATW execution, thus, achieving unnoticeable MPD without violating the thermal constraint.

Towards this design goal, we first investigate the state-of-the-art ATW algorithm. Based on the insights, we then propose three PIM architecture components to specifically execute the three ATW phases without interrupting the normal rendering tasks on the host GPU. Finally, we propose a redundancy reduction mechanism to simplify the overall ATW operation by leveraging the special two-eye characteristic of VR applications. To evaluate the proposed architecture, we design a physical platform incorporating HMC simulator framework, and estimate the impact of the proposed technology on a set of real VR applications. The results show that our design reduces MPD by an average of 175% comparing to the GPU-accelerated ATW, resulting in ideal MPD and significant enhancement of user experience. Additionally, by moving the ATW operation from GPU cores to power efficient PIM-enabled logic units, our design reduces the overall energy consumption of the GPU by 16% on average. We also provide a design space exploration to demonstrate multiple design choices for PIM-based ATW. We find that the performance of PIM-based ATW architecture can well scale up with the increasing frame resolution and the decreasing ideal MPD without exceeding the thermal budget of the PIM design. Thus, it can potentially benefit the future VR scenarios with larger frames, higher ATW trigger frequency, and even lower ideal MPD.

II. Background and Motivation

A. Modern VR system and Time Warp

Figure 1 (left) illustrates the common workflow of modern VR systems. The application loop starts from the time when users interact with their *Head-Mounted Device (HMD)* and ends when the display images for the two eyes are shown on the near-eye screen. Within this loop, the VR system first collects user's pose information through HMD, sensors and controllers. Then the application will set up the rendering process and copy the data to GPU. During the rendering process, two frames (i.e., *stereoscopic frames*) are generated simultaneously for two eyes to enhance the perceived reality. And then, *time warp* is executed that transacts the stereoscopic frames to the correct optical distortion, thus, warping the rendered frames to the current view position which changes after rendering starts [12]. Finally, the warped images will be sent to the display buffer and new images are displayed on the HMD when screen refreshes. The time duration from user's inputs to the displayed images is called *application to photon latency*, which equal to the *Motion-to-Photon* (MPD) in the basic VR workflow.

Different from normal PC games, VR applications tend to deliver true immersive environment around users. In order to help users perceive the display image as part of the world as they move their heads, the MPD is the most important factor for a successful VR experience [12], [17], [24], [25]. To avoid simulation sickness and discomfort, the MPD should be shorter than 7ms [25]. Unfortunately, while modern VR display screen typically refreshes as 90-120Hz which is capable to delivering images within 10 ms, the application to photon latency is often too long that the updated images are not ready before the screen refresh, which is the most significant challenge in today's VR system. Although rendering latency reduction seems to be a direct way to help address this challenge, it has been proven difficult to reduce the application to photon latency to achieve ideal MPD due to the limitation of hardware performance [12]. Moreover, several approaches were proposed to compensate for latency via head movement prediction [18], [26], [27]. However, the prediction error is often very high due to the initiation and changes of the head movement, causing even worse user experience

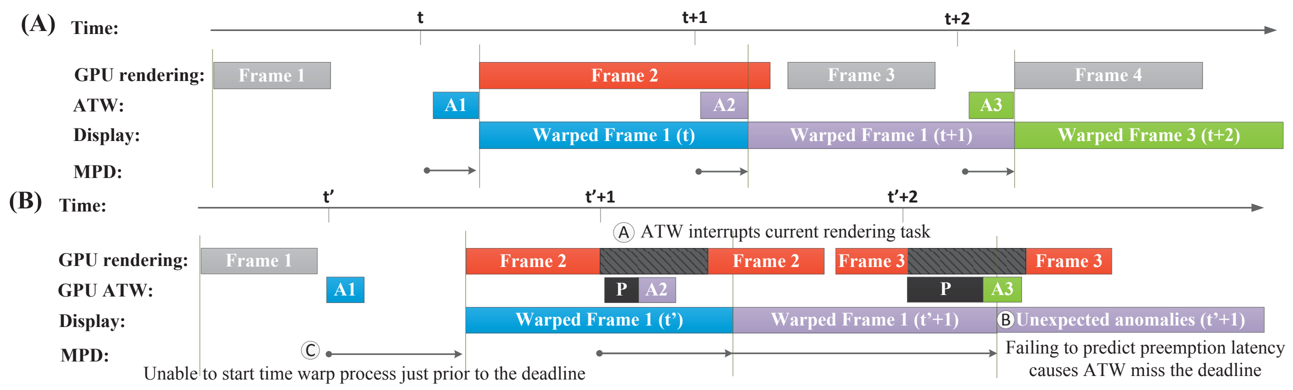


Figure 2: Ideal ATW (A) VS. GPU-accelerated ATW (B). Where Frame i represents the rendering latency, Ai represents the ATW latency, Warped Frame i(t) represents displaying the warped Frame i based on the HMD information at time t, and P represents the preemption latency for GPU-accelerated ATW. The MPD is measured from the time point that HMD information is gather to the screen refresh. The GPU-accelerated ATW (B) requires to wait for the long and unpredictable preemption which interrupts current rendering task as shown in Ⓐ, and the wrong prediction of preemption latency causes ATW missing the deadline as shown in Ⓑ and long MPD as shown in Ⓒ (Details in Section 2.3-(II)).

comparing to the perceivable long latency. Therefore, recent industrial solutions led by vendors like Oculus employ *time warp* to tackle the long MPD challenge, which has become the widely-adopted industry standard [12], [17], [24]. This emerging technology is proposed not only to correct the lens distortion but also to predict the view position of the displayed image using the latest HMD information, shown in Figure 1(right). For example, to process the time warp, the HMD information will be gathered at time $t+1$. Then, the frame rendered at time t is mapped to the 3D len distortion plane using the HMD information at time $t+1$ during the time warp. In this way, the MPD is significantly reduced from application to photon latency to time warp to photon latency (as shown in figure 1(left)) and users can perceive the image at the position as close to their head as possible.

B. Ideal Asynchronous Time Warp (ATW)

Although the time warp could serially operate after the rendering process completes, as shown in Figure 1, in reality it is better to perform asynchronously with the entire VR application loop, which is called *asynchronous time warp* (or **ATW**). Figure 2 (A) shows the ideal ATW pipeline. There are two major advantages to apply ATW. First, asynchronous operation can start the time warp before the rendering finishes. Therefore, if the rendering process is unable to deliver the updated frame in time for the next screen refresh (e.g., Frame 2 in Figure 2(A) misses the refresh deadline), the ATW can still complete in time (e.g., A2 in Figure 2(A)) that warps the latest rendered frame (Frame 1) based on the current HMD information (i.e., at time $t+1$) to ensure the MPD equal to $T(t+1, refresh\_2\_deadline)$ (i.e., the period between $t+1$ and refresh deadline for Frame 2). Second, if the time warp runs asynchronously, the rendering process is not required to stay synchronous with the screen refresh. For example, Frame 3 in Figure 2 (A) can start immediately after Frame 2 without waiting for the screen refresh, thus significantly improving the overall graphics hardware utilization.

Table I: PLATFORM CONFIGURATION

Item	Value
VR system	Oculus Rift
Display Resolution	2160x1200
CPUs	Intel Core i7 7th Gen 7700HQ 2.8GHz 4 Physical core with 16 GB Memory
GPU	Nvidia GeForce GTX 1060 256KB register file per SM 96KB share memory per SM 48KB unified data cache per SM 6G GDDR5 with 192GB/s Bandwidth 120W max working power

C. GPU-Accelerated ATW

While the ATW theory is promising to significantly improve VR experience by reducing MPD, it is still challenging to actually implement such high quality time warp with low latency on the commercial hardware. Oculus [12] has estimated the ATW latency on commercial GPUs equipped with specialized hardware (e.g., texture units and frame buffers) and proposed the state-of-the-art implementation of time warp as an OpenGL kernel executing on GPUs, shown in Figure 2 (B). To reduce the inefficiency of concurrently executing rendering and ATW (e.g., Frame 2 and A2 in Figure 2 (B)) due to the reduced parallelism and high memory-level contention, tile or pixel level preemption (i.e., P in Figure 2 (B)) is proposed for today's VR systems when the current frame cannot finish its rendering before the deadline [28], [29], [30], [31]. Under this design, ATW will be process in GPU after preemption completes.

(I) Overall Impact of GPU-Accelerated ATW on User's Perceptive Latency. In order to demonstrate the impact of ATW on real-world VR systems, we execute several VR applications on our evaluation platform. These VR applications are randomly selected from Nvidia VRwork [32], steamVR [33] and Oculus application store [34], and they are executed under different rendering resolutions. The configuration of our evaluation platform is shown in Table I. The experimental PC laptop equips VR-ready GTX 1060 GPU as its rendering processor which is enabled with preem-

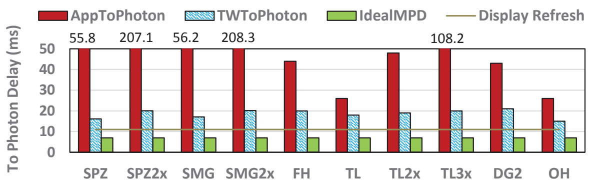


Figure 3: Perceived latency across different VR applications.

ption-based ATW, and Oculus Rift [4] is chosen as our VR system. The rendering drivers and analysis tools on our platform are all up-to-date. We leverage the Oculus Debug Tool [35] to configure VR systems and collect VR application information such as Motion-to-Photon Delay (MPD), App-To-Photon Delay, frame drop, ATW miss and so on. We collect these values by running 1000 frames and calculating their averaged value.

Figure 3 shows the perceived latency under the impact of ATW across different VR applications. We estimate it by measuring the averaged value of application to photon latency (**AppToPhoton**) and time warp to photon latency (**TWTo-Photon**), both of which were discussed in Section II.1 and Figure 1. AppToPhoton describes the MPD without applying ATW, while TWToPhoton describes the MPD after applying ATW. The green bar represents the ideal MPD, below which users will not feel motion-triggered discomfort [24], [25]. We have several observations: (i) by comparing AppToPhton and TWTo-Photon, ATW can effectively reduce the perceivable latency from the time when the last motion info is fed to the VR system to the point when the targeted frame is projected on the HMD display; (ii) the variation of blue bar values is much less than that of the red bars, indicating that ATW provides relatively static latency so that users can receive much smoother animation in their VR experience; and (iii) the results also indicate that the state-of-the-art solution of GPU-accelerated ATW (blue bars) cannot satisfy the high latency-sensitive requirement of today's VR applications (i.e., ideally less than **7ms**) and even longer than the display refresh interval (i.e., **11ms**). Although current VR industry commonly consider 20ms is a acceptable MPD [12], [24], they are trying to reduce this "tolerable" MPD to be as close to "ideal" as possible so users can stay in the VR world longer and experience highly-interactive environment without severe motion sickness [24], [25].

(II) Root causes of the inefficiency of the state-of-the-art GPU-acclerated ATW. We further explore the reason why GPU-accelerated ATW cannot achieve the targeted ideal MPD. Figure 2 (B) illustrates the rendering pipeline with GPU-accelerated ATW. Ideally, ATW could reduce MPD by transforming the stereoscopic frames from the view at the time they were rendered, to the correct view at the time they are displayed. However, under GPU-accelerated ATW design, in order to execute ATW in time, the rendering process for Frame 2 will be preempted and the runtime information on GPU registers needs to be stored to the off-chip memory so that it can be resumed later. We observe that the preemption and ATW execution interrupt the rendering task, resulting in significant frame rate reduction (Ⓐ in Figure 2(B)).

Figure 4 shows the performance and energy overheads when applying GPU-accelerated ATW. We collect GPU

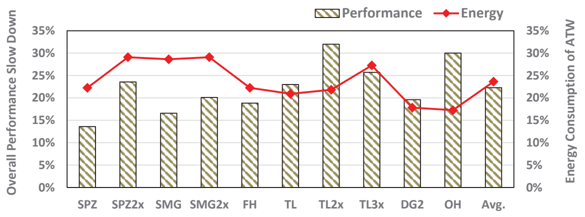


Figure 4: Performance and energy penalty from GPU-accelerated ATW.

execution time separately for the rendering task and ATW. The performance penalty is normalized to the GPU rendering time for each pair of frames. From the results, we observe that ATW operation significantly slows down the GPU rendering task by **22%** on average. Because ATW is triggered for each screen refresh instead of each frame, the performance degradation extends the frame delivery time and thus highly reduces the overall frame rate, resulting in motion lagging sensation in highly interactive VR environment. For instance, for SPZ2x and SMG2x applications, the rendering scenes are too complex to be finished within few screen refresh intervals, the ATW interferes the rendering process many times so that the performance penalty for each frame is even longer than 16ms. The red line in Figure 4 shows the percentage of energy consumption over the entire GPU by ATW within each screen refresh interval. On average, GPU costs **23.6%** additional energy to process ATW and preemption. In summary, GPU-accelerated ATW causes significant inefficiency due to the frequent context switch between normal rendering and ATW operation.

We also observe that the preemption latency highly depends on the concurrent workload which is very difficult to predict at runtime. In order to process ATW command on GPU, the CPU requires to issue a context switch command at first to suspend the current task in graphics queue. The task suspension can not be triggered until its current states can be saved for future task resume, resulting in long context switch overhead. Then, GPU preempts the runtime data to off-chip memory which incurs preemption penalty. The context switch overhead and preemption penalty are determined by many factors such as scene complexity, queue depth, OS factors and available memory bandwidth. These factors make the prediction on when to issue the ATW command the biggest challenge for modern preemption-based VR systems. Issuing the ATW command too late makes it missing the deadline (Ⓑ) and significantly extends MPD which becomes $T(t'+1, refresh\_ 3\_deadline)$; while issuing the ATW command too early leads to a long MPD as well (Ⓒ) because MPD becomes $T(t'+1, refresh\_2\_deadline)$, and it is irrelevant to the overall latency of preemption and ATW. As can be seen, Techniques (e.g., increasing memory bandwidth) that boost the performance of preemption and ATW have little impact on reducing the MPD.

Figure 5 quantifies the latency breakdown of MPD, from the point when the preemption command is issued, to that image is displayed in front of user's eyes. We leveraged Microsoft GPUView [36] and Oculus Debug Tool [35] for the measurement. We observe that even through the GPU can usually finish ATW within 2ms (green), the CPU context switch overhead (orange) and preemption latency (gray)

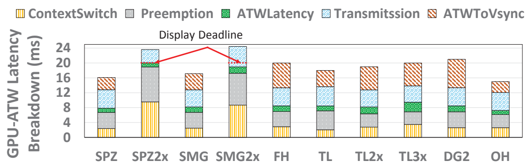


Figure 5: The latency breakdown of MPD across different VR applications.

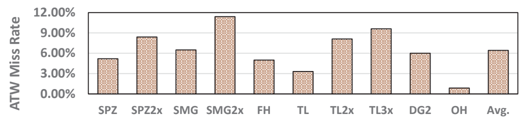


Figure 6: ATW's refresh deadline miss rate.

significantly delay the start time of ATW. In addition, we observe that this latency varies across different frames and applications, ranging between 6ms to 22ms. As a result, there is a high risk for the GPU to miss the ATW deadlines (e.g., display refresh interval being 11ms), which delivers outdated images to users and causes motion anomalies (Figure 2(B)-Ⓑ). Figure 6 shows the refresh deadline miss rate for ATW. We observe that ATW miss commonly occurs on GPU-accel-erated ATW design. On average, 6.44% of ATW operations miss the refresh deadline, indicating that users may receive anomalous images every 0.2 second which may cause severe discomfort and simulation sickness. Additionally, we find that the miss rate becomes even higher when increasing resolution and application workload. In order to avoid the worst case, the state-of-the-art VR systems (e.g., Oculus) tend to trade off ATW quality by initiating ATW at a much earlier time point than the display deadline (e.g., Ⓒ in Figure 2(B)), which unfortunately increases the MPD.

In summary, using asynchronous execution of ATW to entirely replace such preemption-based approach seems to be a more effective method to address this prediction challenge. This motivates us to find a new way to execute ATW asynchronously with GPU cores to directly eliminate the task switching latency and preemption penalty.

III. Processing-In-Memory Opportunities

In order to allow fully asynchronous execution of the regular rendering task and ATW, previous work [17] suggests to separate them across multiple GPUs. However, as shown in [17], copying the pair of frames to the second GPU through PCIe links incurs significant transfer latency. Similar situation also occurs to a more heterogeneous architecture (e.g., CPU+GPU or DSPs+GPU) [12]. Since VR system is latency-sensitive, a major challenge we face is: *how to design a new architecture to satisfy both performance and asynchronous requirement of ATW?*

Understanding the ATW Algorithm. The state-of-the-art time warp is generally implemented as rotating the rendered frames based on the latest head orientation information which maps the frames on the lens distortion mesh. Algorithm 1 shows the pseudo code of the current ATW algorithm, which includes three major execution phases: Lens Distortion, Matrix Transform and Texture Filtering. Before the ATW operation, the lens distortion (*LD*) is pre-calculated which depicts how the pixel coordinate of the display image will be projected on the len, and it is then stored in the memory. During ATW, the lens distortion is loaded from memory and separated into 32x32-pixel tiles (line 3-4) which serve as the basic workload unit. For each tile, the time warp transform (Δ_{xy}) is then calculated as a 4x4 homogeneous transformation matrix and applied on the distortion coordinate of the tile corners to get the texture coordinates of the corners ($texCorner_{display}$) (line 9-11). In this phase, the distortion coordinates of the tile corners are transformed into [0,1] texture coordinate which can be used directly to sample the rendered frames. Note the matrix multiplication is only processed on the corners to reduce the computing and storage requirements. Finally, for each pixel within the tile, a linear interpolation approximation is employed to calculate the warped texture coordinate for each pixel within the tile ($texCoor_j$), and the rendered frame is sampled to filter the final color of the pixel during texture filtering (line 15-16). To produce acceptable quality and performance, time warp uses bilinear texture filters as its default filtering method which interpolates between four texels surrounding the sample texture coordinate. As a result, the rendered frame which is stored in GPU off-chip memory need to be accessed multiple times. This causes high memory access overheads and becomes the major performance bottleneck for ATW. In summary, the performance of time warp is determined by the linear interpolation approximation and texture filtering from the texture unit.

Table II: Basic Comparison Between HMC and HBM

	HMC Gen3	HBM2
Density	8GB	8GB
Off-chip bandwidth	320 GB/s	256 GB/s
IO	4 SerDes links	Parallel links
Memory partition	32 vaults	8 Channels
Expansion Capability	Yes	No
Memory Access	Packet based	DDR
PIM module	3D	2.5D

Algorithm 1 Asynchronous Time Warp

```
 1: procedure TIMEWARPTHREAD
 2: Lens Distortion:
 3:     $i \leftarrow$ number of tiles
 4:     $LD_i \leftarrow$ distortion coordinate from memory
 5:     $V_{render} \leftarrow$ Get HMD matrix from rendering time
 6:     $V_{display} \leftarrow$ Get HMD matrix from display time
 7:     for all $i$ do
 8: Calculate Matrix Transform:
 9:         $texCorner_{render} \leftarrow \text{Convert}(LD_i)$
10:         $\Delta_{xy} \leftarrow V_{display} \times V_{render}^{-1}$
11:         $texCorner_{display} \leftarrow texCorner_{render} \times \Delta_{xy}^{-1}$
12:         $j \leftarrow$ number of pixel
13:         for all $j$ do
14: Texture Filtering:
15:             $texCoor_j \leftarrow \text{Interpolation}(LD_i, texCorner_{display})$
16:             $pixelColor \leftarrow \text{TextureFiltering}(Frames, texCoor_j)$
```

Basic Idea. Due to the frequent off-chip memory accesses induced by ATW, we consider designing a new hardware architecture which could effectively execute ATW near the frame buffer and prevent it from negatively affecting the normal rendering pipeline on GPU. In recent years, emerging 3D stacked technology, e.g., hybrid memory cube (HMC), provides a suitable platform to implement our idea. The fundamental design concept of HMC is to stack multiple DRAM dies upon the base logic layers to build a shortest connective path between data and processing. For example, according to the HMC 2.0/2.1 specification [37] from Micron, several DRAM dies stack on the CMOS logic layers, grouping as a cube. The memory cube connects with the host processor using a series of full-duplex links which provide up to 320GB/s of external memory bandwidth. Additionally, the logic layer is partitioned into 32 sub-memory controllers and communicate with their local DRAM banks through the Through-Silicon Via (TSV). The memory controller and its local memory group together to form as the *vault architecture*, providing 512 GB/s internal memory bandwidth per cube. A crossbar is integrated in the logic layer to support the communication between SerDes links and vaults.

Another popular commercial 3D stacked memory option is high bandwidth memory (HBM). Table II shows the basic differences between HMC and HBM. Both HMC and HBM can provide very high external bandwidth and capacity to serve as GPU resident memory. However, the embedded logic layer and packet-based memory access model from HMC offers good opportunities to execute customized memory command independently and asynchronously with the host core. As a result, HMC has been considered as a mainstream PIM platform in the recent works [19], [20], [21], [22], [23]. On the other hand, HBM is commonly used as a high-speed off-chip memory in high-end GPUs, but it is not often considered as a typical PIM platform due to its lack of logic layer. Additionally, as we discussed in Section II.3-(II) on memory bandwidth impact, only providing high bandwidth will not effectively address the current VR challenge from preemption-based solutions.

The major benefits of a PIM-based architecture are not only enabling efficient processing of memory-intensive ATW within GPU memory which is the closest location to access the frame buffer, but also moving the ATW out of GPU cores to its memory to avoid interrupting the normal rendering task and achieve asynchronous execution. In other words, letting HMC handle ATW prediction and execution releases the pressure of needing the preemption mechanism (Figure 2 (B)) and converting the current GPU-accelerated ATW design to the ideal preemption-free ATW scenario shown in Figure 2 (A); moreover, VR devices can directly read the output of ATW from the frame buffers in HMC at runtime. Hence, we naturally consider the HMC as the suitable platform for ATW. We also discuss the alternative design choices in Section VIII.

Design challenge. Introducing a new architecture design brings a new set of challenges. The state-of-the-art ATW is implemented as an OpenGL kernel for best performance [12]. Due to the thermal challenge of 3D stacked technology [19], we cannot directly integrate entire GPU SMs along with other graphics units (e.g., texture and rasterization units, etc.) onto HMC which is too complex and expensive. Meanwhile, since ATW is latency-sensitive, the-state-of-the-art PIM architecture such as [21] which integrates small in-order CPU cores cannot provide required performance throughput. How to design an architecture that fully unitizes the advantages of the 3D stacked technology to achieve the best performance for ATW is the major research challenge to address in this work.

As we discussed in Section II, ATW should be triggered as close to the refresh deadline as possible to obtain the newest

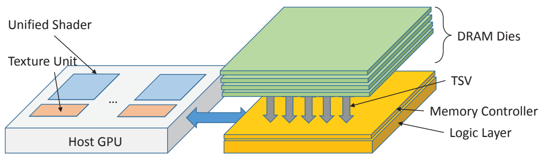


Figure 7: Overview of an GPU-HMC system.

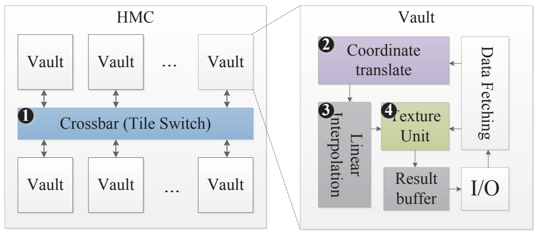


Figure 8: Architecture diagram of the PIM-ATW design.

HMD information. Additionally, the ideal MPD shown in Figure 3 is 7ms which means ideally the ATW command should be issued by the VR system within 7ms before the display refresh. Note that MPD is not only equal to the ATW processing time but also includes the position tracking latency, data transmit delay and display delay. MPD can be modeled using equation 1:

$$MPD = D_{ATW} + D_{sensor} + D_{trans} + D_{display}. \tag{1}$$

Among these parameters, the display delay can be compensated by interpolating the lens distortion mesh during ATW. By referencing to previous studies [38], [39], [40], we estimate that the position tracking latency and the data transmit delay together around 5ms. As a result, to provide the ideal MPD, the new ATW operation must be completed within 2ms and the execution time should remain relatively steady across intervals.

IV. PIM-ATW Architecture Design

Figure 7 shows the overview of the proposed GPU-HMC system. In this system, the host GPU directly connects with an off-chip HMC which acts as its default memory. To implement the design of PIM-based ATW, we propose an extension to the exiting vault architecture as shown in Figure 8. Since the ATW operation is nonprogrammable (supported by a special ISA) and requires a predictable execution time, the proposed PIM-ATW architecture consists of four fixed functional components: ❶ the tile switch to schedule tiles to each vault; ❷ the coordinate translate unit to conduct matrix transform; ❸ the linear interpolation to calculate the texture coordinate for each pixel inside a tile; and ❹ the texture unit to conduct texture filtering. In this Section, we will introduce each of the four components and explain how they help complete the major tasks of ATW algorithm (Algorithm 1): lens distortion, matrix transform and texture filtering.

A. PIM-ATW Architecture Structure

The tile switch ❶ is a simple workload scheduler which sends the tile to target vault if the it is idle. Note that the host GPU could access the entire HMC memory while a vault can only access its own local DRAM partition by default. Previous work [21] proposed to add an additional

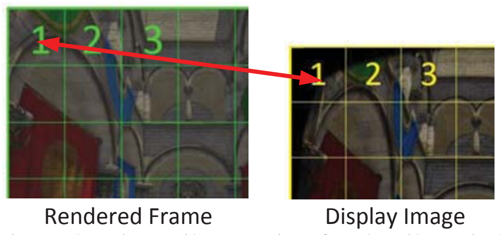


Figure 9: Direct tile mapping for the tile switch.

interface to help pass the required data to a remote vault, but it incurs extra overhead for the crossbar which may affect the rendering process. To avoid remote access and improve workload balance and utilization across vaults, we propose a direct tile mapping mechanism as shown in Figure 9. First, we divide the rendered frame into a number of tiles which have the same size as the display image. Then we mark these tiles using a number from 1 to $sizeof(tile)$ and pair the tiles of display image with the rendered frame. In this case, the sampled area of the rendered frame for each tile is limited to avoid remote access. This tile mapping policy leverages the default format of GPU's frame buffer, which is configured as tiled format instead of linearly left-to-right or top-to-bottom layout [41]. In this way, the frame buffer's memory space is fairly distributed across vaults in order to increase memory bandwidth. As a result, the tile switch distributes the same number of tiles to each vault to fully utilize the computing resources.

The coordinate translate unit❷ is responsible for calculating the warped transform coordination for the lens distribution. It performs 4x4 homogeneous transformation matrix calculation. We implement the matrix multiplier as a MAC unit [42]. It first calculates the delta matrix between the view matrix used for rendering and the updated view matrix based on the new sensor input. After that, it computes the transform by multiplying the inversed delta matrix with the rendering projection matrix. Finally, the transform is applied to the coordination of the tile corners so that they can be used directly to sample the stereoscopic images. During this phase, only the texture coordination of corners will be calculated which reduces the compute loop in Algorithm 1 by a factor of 2^8 (4 out of 1024).

The linear interpolation component ❸ is used to generate the texture coordinate for each pixel inside a tile. It performs as a SIMD4 floating point unit so that 4 pixels can be processed in parallel. Because a texture coordinate is formated as a 8.8 fixed-point value, a 32-bits register is used to hold the temporary result. The output coordinate and its source pixels will be stored in a buffer for texture filtering.

The texture unit ❹ has similar functions as the normal texture unit in GPU [43]. Unlike GPU cores, texture unit only processes fixed functions which can be easily integrated into the logic layer of HMC [23]. The texture unit acquires filtering tasks from the coordinate buffer and performs bilinear filtering by sampling four closest texels to the texture coordinate in the rendered frame. The performance of texture filtering can be significantly improved by the high internal bandwidth of the vaults and the tiled format of the rendered frame. Note that 4 pixels are filtered in parallel in the texture unit.

B. Detailed PIM-ATW Operations

In order to trigger PIM-ATW operation in HMC, we propose a customized memory operation which can be directly sent to HMC by leveraging the native memory access interface (e.g. Direct Memory Access (DMA)) between CPU and GPU [44]. This interface bypasses the graphics queue and the GPU shader cores through the asynchronous memory copy engine which is widely adopted in most GPU architectures (e.g. Nvidia Pascal [30] and AMD Radeon [45]). In other words, CPU offloads the ATW command which includes the latest HMD information directly to HMC to avoid interrupting the normal on-chip rendering tasks.

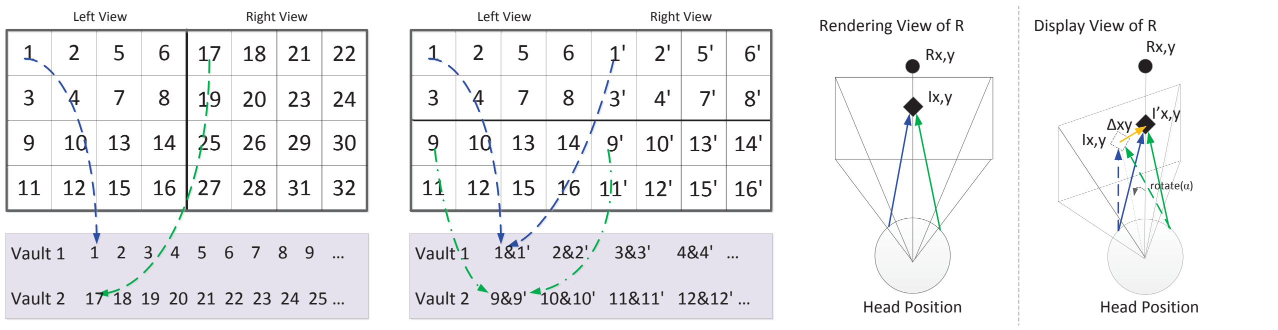
After receiving the ATW command from CPU, the view matrix will be broadcasted to each vault and stored in its local DRAM. The ATW tasks are then divided into tiles, each of which covers 32x32 pixels of the displayed image. The number of tiles is determined by the display resolution. For instance, modern VR systems typically adapt 2160x1200 resolution for display image (1080x1200 for each eye) which is then divided into 2560 tiles. Each tile will be scheduled to the targeted vault via the tile switch to perform time warp operation on 32x32 pixels. As introduced in Section III, the ATW algorithm or time warp operation consists of three phases (lens distortion, matrix transform and texture filtering), all of which are processed inside a vault to avoid any remote data fetching.

Next, the coordinate translate ❷ will calculate the texture coordinates for the corners of the tile. After the coordinates of the four corners are calculated, the linear interpolation ❸ generates the texture coordinate for each pixel. The linear interpolation delivers 4 pixels each time to the texture unit ❹. The 4 pixels are grouped as a quad which is the smallest workload for the texture unit. Finally, the texture unit applies bilinear filtering on the quad to get the final color for the pixel. The benefit for filtering the pixels as a quad is the texture unit can use one 64B-read request to fetch the entire 4x4 texels (corresponding to the 4 pixels) from the rendered frame. After all the pixels within the tile are colored, the result will be stored in the memory for display and the vault is marked as available for processing new tiles.

The linear interpolation and texture unit in the vault are fully pipelined in order to reduce performance latency. Their performance is determined by the texture fetching, and in this PIM-ATW design the pipeline could finish 4 bilinear filtering per cycle due to the high internal throughput from TSV.

C. Overhead and Thermal Analysis

We evaluate the area and power overhead of our proposed PIM-ATW design by modeling the logic units within a vault as a small in-order CPU core using McPAT [46] under 24 nm process technology. For area, the major overhead of PIM-ATW comes from the fixed function units inside the vault. We model the matrix multiplier as 4 MULs, and the linear interpolation and texture unit as 8 SIMD4 FPUs by referencing the logic circle design from the previous works [42], [43]. For the storage overhead, we model the texture cache as a 16KB read-only data cache and add 8KB 32-bits register file per vault to hold the intermediate data. Based on McPAT, the area overhead of PIM-ATW for each



(a) Original policy (b) Redundancy reducing policy (c) Time warping for object R

Figure 10: Although the images for two eyes have different contexts, they share the same warped transform matrix (δ_{xy}) because time warp only updates the newest head orientation (same for both eyes). Thus, we can map the tiles with the same coordinate from the two eyes to the same vault so that the transform matrix is only calculated once.

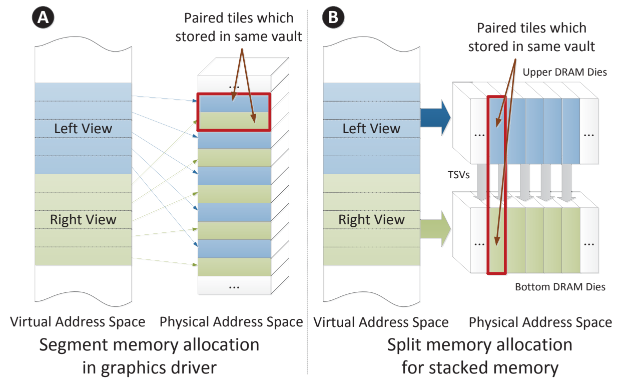

A
Paired tiles which stored in same vault
Left View
Right View
Virtual Address Space
Physical Address Space
Segment memory allocation in graphics driver
B
Paired tiles which stored in same vault
Upper DRAM Dies
TSVs
Bottom DRAM Dies
Split memory allocation for stacked memory

Figure 11: Memory allocation mechanisms for the left and right views.

vault is $1.6mm^2$, which is much smaller than the thermal and area budget of a fixed-function unit group in each vault ($3.25mm^2$) [47].

Since PIM-ATW design raises the power consumption which may incur thermal challenge in DRAM, we further evaluate its power and thermal effects. For power overhead, the modeled logic units incur about 106mW under the default frequency (500MHz) which translates to 66 mW/mm^2 power density. Prior works demonstrated that the maximum power density under the thermal budget of a 3D-stacked DRAM is around 133 mW/mm^2 [21], [48], and the maximum power overhead of in-memory processors is 10W TDP [19]. According to the evaluation, our PIM-ATW design's power density is within the thermal design constraints and acceptable for die-stacked processors. We conduct further thermal feasibility analysis in Section VII.2.

V. Reducing two eyes redundancy for PIM-ATW

After proposing PIM-ATW design, we continue to explore other optimization opportunities to further reduce MPD. VR systems are designed to deliver real immersion environment with high-quality images for both eyes. The images from the left and right views in VR typically exhibit particular similarity, resulting in redundancy. For instance, the same objects in left and right eyes share the same 3D space scene. Thus, some technologies such as single pass rendering [49] and instancing rendering [50], [51] try to process the rendering tasks for both eyes at the same time, but no discussion on ATW operation under this scenario. The state-of-the-art ATW operation considers the stereoscopic frame as one single texture input which ignores this special characteristic of VR applications. Figure 10a describes an example of the original tile distribution policy applied in our PIM-ATW design. In this example, the display image is divided into two sets of tiles which are directly mapped to the frame buffer using tile-based format. In addition, the HMC system only consists of 2 vaults so that the frame buffer is fairly distributed into the local DRAM of the two vaults. During the default ATW, the two tiles with the same position (e.g., 1 and 17) from the two eyes are scheduled into different vaults as shown in Figure 10a.

With further investigations, we observe that the ATW operation for left and right eyes share the same warped transform matrix. Figure 10c shows the rendering view and the display view of the object R. $R_{x,y}$ is the 3D space scene of R and $I_{x,y}$ is the projected coordinate on the 2D screen of R, where the projection from $R_{x,y}$ to $I_{x,y}$ satisfies the rendering projection matrix. Before the display refresh, head rotation causes both left (blue) and right (green) views to rotate. However, the way to calculate the transform matrix Δ_{xy} from $I_{x,y}$ to $I'_{x,y}$ only depends on the latest HMD rotation, not the individual eye movement [12]. In other words, the transform matrix Δ_{xy} generated by ATW rotation on 2D image is shared by both eyes. Motivated by this characteristic, we process ATW for the two tiles with the same coordinate from the left and right eyes simultaneously which is shown in Figure 10b. In contrast to the original tile scheduling policy, the new redundancy reduction policy schedules two tiles with the same coordinate to the same vault. By doing this, the transform matrix for the two tiles only needs to be calculated once to reduce the computing and fetching overhead, and processing two tiles in one vault enhances the vault-level parallelism. Note that our redundancy reduction mechanism for ATW is different from the rendering redundancy reduction on the shared contents/objects between two eyes' frames that has been widely used in the state-of-the-art VR systems.

To implement this idea, we need to group the original tiles for the two eyes as a new tile so that the required data for the left view and right view can be directly mapped within a vault. A straightforward method is to allocate continuous

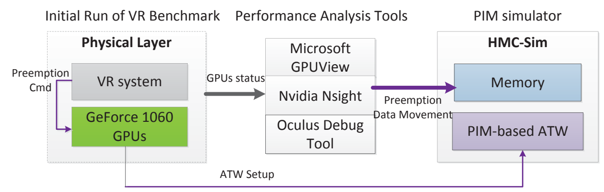


Figure 12: Evaluation Infrastructure.

physical address space for the memory segments of the tiles from the left and right views, as shown in Figure 11(A). However, aligning two memory segments will reduce the bank-level parallelism within the vault which causes contention at the memory controller level. Fortunately, the 3D stacked technology of HMC enables parallel access to DRAM dies within the same vault, providing an opportunity to design a more efficient mechanism to pair memory segments. Figure 11(B) describes the split memory allocation mechanism. In this method, the frame buffer is divided into two parts which are directly mapped to the upper and lower DRAM dies. As a result, the paired tiles for the left view and right view are stored in different floors of the same vault which split the physical address space, increasing internal bandwidth utilization.

VI. Evaluation Methodology

We evaluate our proposed PIM-ATW design using a physical platform incorporating HMC simulator framework as shown in Figure 12. The basic physical platform configuration is described in Section II and Table I. In order to evaluate the effectiveness of our PIM-ATW design, we compare the following design scenarios: (1) GPU-ATW (baseline): the state-of-the-art GPU-accelerated ATW which is widely adopted in the current VR systems; (2) GPU-ATW+HMC: the HMC equipped GPU which achieves 320GB/s memory bandwidth and the ATW is implemented on GPU cores, same as the baseline case; (3) GPU-ATW+exLogic: the naive asynchronous design which directly integrates ATW logic on the GPU chip and uses HMC as the default memory; (4) PIM-ATW: the basic processing-in-memory design for ATW as described in Section IV; (5) PIM-ATW-RR: the optimized PIM-ATW design which enables redundancy reduction scheduling policy (Section V). The rest of this section describes our simulation environment.

GPU-ATW. The basic setup has been introduced in Section II.3 and Table I. We leverage Oculus debugging tool [35] to enable ATW operation on real VR applications and collect their MPDs on this baseline design. To acquire the GPU utilization and energy consumption, we profile the runtime status of GPU using Nvidia Nsight [52] and GPUView [36]. Note that this setup is equipped with a typical GDDR5 memory with 192GB/s bandwidth.

GPU-ATW+HMC. This design simply replaces baseline G-PU's memory (GDDR5) with a high-bandwidth HMC without enabling PIM. In order to evaluate the preemption latency, we first model the GPU preemption by estimating the data copy latency using HMC-sim [53], [54]. Table III shows the configuration of our HMC model. Based on the memory frequency (1GHz), we observe that the data copy latency on HMC-sim is reduced by 35% on average comparing to GDDR5 (192GB/s) which is similar to the bandwidth scaling results. Then, we scale down the execution time of ATW by multiplying it with the ratio of memory bandwidth between GDDR5 and HMC. We believe this scaling method is accurate based on the prior works on 3D rendering [23]. Finally, we estimate the MPD by combining the context switch overhead, preemption latency, ATW execution time, transmit delay and display delay. However, we need to note that because it is the VR engine to determine when to issue ATW from CPU, this design scenario only reduce the ATW miss rate to avoid extra long MPD.

GPU-ATW+exLogic. This design naively integrates ATW logic on the HMC-equipped GPU chip so that ATW can be executed concurrently with the rendering process to avoid preemption latency. However, the newly added on-chip ATW units need to share the on-chip storage resources and the off-chip memory bandwidth with the rendering process. We model the sequential data loads access pattern for on-chip ATW and the random memory access pattern for rendering (i.e., worst-case rendering access scenario) on HMC-sim, we then estimate the off-chip memory contention overhead between the additional on-chip ATW units and the rendering pipeline. Each cycle, one memory packet which is randomly selected either from ATW or rendering is sent to the HMC through the available SerDes links. We collect the contention overheads and combine it in the total MPD.

PIM-ATW. We construct the PIM-ATW operation using H-MC-sim [53], [54]. The simulator has a hybrid memory cube micro-architecture model that provides cycle-accurate performance analysis. It also permits users to design customized memory cube (**CMC**) and its operations. We further expand this CMC capability to provide cycle-accurate simulation for the added hardware structures, including coordinate translation, linear interpolation, texture filtering and tile scheduling. The CMC operations of ATW are modeled as an instruction set by referencing the open-source library provided by Oculus VR [12], [55]. Similar to the GPU-ATW+exLogic scenario, we also inject random memory accesses into HMC during the ATW execution to estimate the internal memory contention. The logic unit configuration is shown in Table III. In order to estimate the execution time of PIM-ATW, we convert the simulated execution cycles into milliseconds by multiplying the cycles with the working frequency of the proposed computing units. When we calculate the MPD, we add extra 5ms to the execution time for data transmit and sensor delay as shown in Equation (1). Note that **PIM-ATW-RR**'s setup is the same as PIM-ATW design.

Energy Consumption Estimation. Similar to the previous works for modeling GPU power [56], [57], we first estimate the energy consumption of the baseline GPU-accelerated ATW based on GPU working power and utilization: $E_{ATW} = T_{ATW} \cdot U_{ATW} \cdot P_{GPU}$, where T_{ATW} is the GPU execution time for ATW (including the preemption latency), U_{ATW} is the GPU utilization during this time and P_{GPU} is the average working power of the GPU. We also apply a similar method to evaluate the GPU energy consumption of the rendering tasks.

To evaluate the energy consumption of the PIM-ATW, we separate its energy into two parts: memory access energy and in-memory computation energy. We apply a bit-based memory energy model used in the previous works [19],

Table III: SIMULATOR CONFIGURATION

HMC-Based ATW	
Off-chip bandwidth	320 GB/s total for 4 links
HMC configuration	32 vaults, 8 banks/vault 8GB 4-stacked DRAM Dies 1.25Gb/s TSV signaling rate 64 TSV per Vault
Memory frequency	1 GHz
Default framebuffer size	512M for default resolution
Logic unit in memory	1 for each vault
Logic unit frequency	500 MHz as default
Process tile size	32x32 pixels per tile
Coordination translate	4x4 matrix MUL
Linear interpolation	one SIMD4 ALU 4 pixels per cycle
Texture unit throughput	4 pixel per cycle
Texture cache	16KB, 4-way
Filtering method	bilinear filtering

Table IV: VR APPLICATION BENCHMARKS

Abbr.	Names	Platform	Engine	FPS(Hz)
SPZ SPZ2x	Sponza	VRwork	DirectX12	1xRes:56 2xRes:14
SMG SMG2x	SanMigue	VRwork	DirectX12	1xRes:42 2xRes:11
FH	FunHouse	SteamVR	UE 4	73
TL TL2x TL3x	TheLab	SteamVR	Unity	1xRes:90 2xRes:73 3xRes:45
DG2	DG2	Oculus	Hidden Path	74
OH	Home Page	Oculus	unknown	90

[58], [59] to calculate the energy consumption of data movement through TSVs, which is 4pJ/bit for the internal memory access within the vault. We then collect the total data transmission size (bits) through HMC-sim. The in-memory computation energy is estimated by multiplying the execution time of ATW with the average power of the added functional units in each vault via McPAT [46]. The working power of the baseline GDDR5 memory and HMC are estimated by the micron power model [60]. For example, the working power of HMC is estimated by scaling the number of memory channels, bandwidth and TSVs' configuration based on Table III.

Benchmarks. Shown in Table IV, the set of benchmarks employed to evaluate our design includes six real VR applications, covering different rendering libraries and vendor platforms. All the applications are rendered with our default rendering resolution (1920x1080). For better evaluating our techniques to reflect the impact of workload size, we also render some of them (*Sponza, SanMigue and The Lab*) with different resolutions. To obtain GPU runtime status, we leverage the Oculus debugging tool to capture 1000 frames for each application and calculate the their average result. The default display resolution is set to 2160x1200 which is equal to that on the state-of-the-art VR systems (e.g., HTC Vive [5] and Oculus Rift [4]). Additionally, considering various quality requirements for different VR systems (current and future), we warp the stereoscopic frames in different display resolutions (i.e., 2880x1600, 2160x1200 and 1280x640).

VII. RESULTS AND ANALYSIS

A. PIM-ATW's Impact on MPD, Energy and User Experience

(1) Impact on MPD. As we discussed in Section II, MPD is the most important factor determining user VR experience which is defined as the time from the ATW command is issued to the frame is displayed. Figure 13 shows the normalized MPD values of different VR applications under different design scenarios. As discussed previously, we set the "ideal" MPD as 7ms, below which users will not experience motion anomalies. All the other design scenarios are also normalized to the ideal MPD. We have several observations from this figure.

First, the high external memory bandwidth of HMC only slightly improves the MPD over the baseline (around 3%) because it only improves the ATW miss rate when ATW is issued too late by the VR system, but fails to reduce the long MPD when ATW is issued too early, not to mention completely addressing the complex prediction problem on the ATW issue time which is determined by a mix of factors including rendering scenes complexity, graphics queues, OS factors, etc, as discussed in Section II.3-(II). Second, the GPU-ATW+exLogic reduces the MPD by 114% comparing to the baseline case (i.e.,GPU-ATW) by removing the context switch and preemption overhead. However, it cannot achieve the ideal MPD due to the communication resource (e.g., SerDes links between HMC and host GPU, crossbar in the HMC logic layer) contentions between ATW and rendering tasks.Third, the PIM-ATW reduces the MPD by an average of 169%, 166%, and 47% comparing to GPU-ATW, GPU-ATW+HMC, and GPU-ATW +exLogic, respectively. The performance improvement of PIM-ATW is mainly achieved through preemption and context switch free asynchronous ATW execution, as well as leveraging the high internal bandwidth in HMC. We also observe that there are only few contentions on the communication resources between PIM-ATW and rendering process. This is because our direct tile mapping design avoids remote vault accesses so that PIM-ATW only utilizes the crossbar in HMC when issuing new tiles which is quite rare. In this case the off-chip communication resource such as SerDes links and crossbar in HMC can be fully utilized by the rendering process. In addition, the PIM-ATW and rendering process will access different memory blocks in different banks so that they have negligible contentions for the internal bandwidth within the vault. As a result, the performance of PIM-ATW is predictable so that the ATW can be issued as close to the display refresh time as possible for the shortest MPD. Finally, the PIM-ATW-RR outperforms all the other designs and achieves an average of 175% MPD reduction over GPU-ATW. The average MPD of PIM-ATW-RR is even 9.8% shorter than the "ideal" MPD which leaves enough space for prediction error on ATW issue time in HMC. Furthermore, we observe that the MPDs of GPU-ATW and GPU-ATW+exLogic across different refresh intervals exhibit high variations while the MPDs of PIM-ATW remain quite stable and only vary when altering the rendered frame resolution. Therefore, our designs can effectively avoid frequent ATW deadline missing.

Figure 14 shows the comparison for the average MPD variance under different applications. This variance is estimated by calculating the standard deviation of the ATW latency across 1000 frames for each application and normalizing the value to the average. From this figure, we observe that the variance caused by GPU-ATW design is intolerable

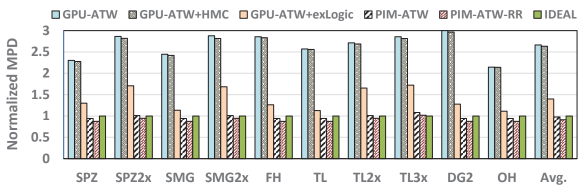


Figure 13: Normalized MPD under different design scenarios.

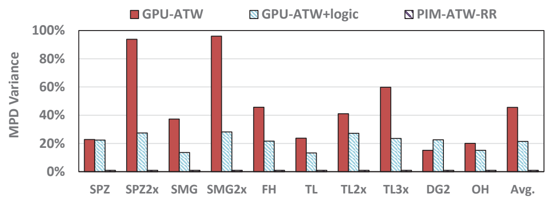


Figure 14: MPD variance under GPU-ATW and PIM-ATW-RR.

for certain applications. For example, the average MPD variance of SMG2x is 97% which means if the VR system wants to avoid ATW deadline missing, it needs to issue the ATW operation before the display refresh deadline about 2x of the average execution time of GPU-ATW. The large variances of GPU-ATW are due to the long and unpredictable context switching in CPU and preemption in GPU. The complexity of different rendering workloads and the depth of graphics queue result in different preemption latencies which highly affect the predictability of MPD. As a result, the measured MPD is usually much longer than the actual execution time of ATW under the GPU-ATW design. We also observe that the variance is more significant under higher image resolutions in more complex VR scenarios (e.g., SPZ2x, SMG2x). For GPU-ATW+exLogic, the variance is mainly caused by the off-chip memory contention between ATW and rendering process. For PIM-ATW-RR, it executes ATW operation asynchronously with the rendering tasks and it is preemption free. In our experiments, the average MPD variance of PIM-ATW-RR is less than 1%. In summary, PIM-ATW-RR not only achieves the best MPD value under different design scenarios but also provides the best average MPD variance.

(2) Impact on Energy Consumption. Figure 15 shows the normalized energy consumption of PIM-ATW and PIM-ATW-RR (bars) against that on GPU-ATW. The results show that PIM-ATW and PIM-ATW-RR achieve an average of 93.1% and 96.7% energy saving, respectively, comparing to the baseline GPU-ATW. The significant energy saving from PIM-ATW is mainly due to two reasons. First, the entire working power of executing ATW in our proposed PIM-ATW is only 3.424W, which is much more efficient than executing ATW in the high-end GPU shader cores(120W). Second, PIM-ATW is able to process ATW in HMC and significantly reduces the data movements between the host GPUs and HMC. Additionally, our design also benefits the overall GPU energy consumption. As the purple line in Figure 15 demonstrates, moving ATW operation from GPU to HMC significantly reduces the overall energy consumption of GPU is by 16% on average.

(3) Impact on User Experience. To understand how our

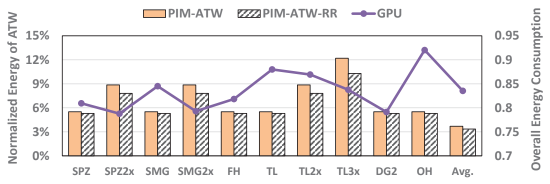


Figure 15: Energy consumption reduction of PIM-based ATW.

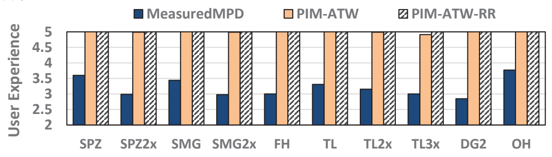


Figure 16: User Experience Study.

design impacts user VR experience, we build a linear-based user satisfaction estimation model, similar to that in [61]. Although the users' VR experience can also relate to image quality, energy and application latency, we only focus on MPD here since it is the most important factor that determines user motion-related sensation in their VR experience. We set the user satisfaction degree towards MPD to three levels: unnoticeable (0,7ms], tolerable (7,T_i], and unacceptable (T_i, ∞). If the MPD is within the unnoticeable range, we give it the highest satisfaction score (5) while the lowest score (0) will be given when MPD is longer than T_i. For tolerable range, we assume the satisfaction score linearly decreases with MPD increasing and 20ms MPD has an acceptable satisfaction score (3) based on industry standard [10]. Figure 16 shows the average user satisfaction scores for different design scenarios. MeasuredMPD represents satisfaction score for the average MPD measured on each VR application. Comparing with the MeasuredMPD, our proposed architecture designs achieve the highest satisfaction score for most cases.

B. *Sensitivity to Display Resolution*

After demonstrating that our proposed design can satisfy the requirement of the current state-of-the-art VR systems, we want to further explore whether it also work for future VR platforms. For example, head devices with higher resolutions (e.g., 2880x1600) are likely to be released into the VR market soon to provide even better image quality [62]. They will increase the number of pixel tiles for ATW processing. Meanwhile, lower display resolution (e.g., 1280 x640) is also possible to be applied in mobile VR systems to trade image quality for performance and energy efficiency. To understand how our design impacts MPD on different types of VR platforms, we warp the stereoscopic frame in different display resolutions including 2880x1600, 2160x1200 and 1280x640. Figure 17(left) shows the execution time of ATW for different display resolutions under PIM-ATW and PIM-ATW-RR. The results are normalized to the latency requirement to achieve ideal MPD (2ms). From this figure, we observe that the PIM-ATW is more sensitive to the display resolution than PIM-ATW-RR because the higher display resolution increases the performance impact of the redundancy reduction policy in PIM-ATW-RR (larger shared transform matrix). For a lower resolution (1280x640), the

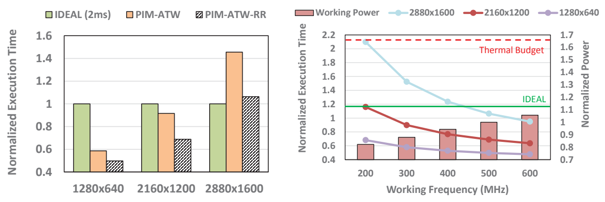


Figure 17: (Left) Normalized execution time under different display resolutions. (Right) Normalized execution time and working power over different working frequencies

PIM-ATW-RR is 51% faster than the ideal case which leaves enough room for miss prediction in ATW initiation point. On the other hand, for higher resolution (2880x1600), although PIM-ATW-RR can still outperform PIM-ATW by an average of 29%, it slightly exceeds the ideal ATW execution time.

We also observe that we can satisfy the requirement for different display resolutions by managing the working frequency of the added computing units in our design. Figure 17(right) shows the execution time and working power trade-offs across different frequencies under PIM-ATW-RR. The execution time is also normalized to the latency requirement to achieve ideal MPD (2ms), and the power is normalized to the working power at default frequency (500MHz). The red dashed line shows the maximum working power under the thermal budget of a 3D-stacked DRAM [21]. In general, we observe that increasing the frequency can help achieve better performance but also linearly increases the working power of PIM-ATW-RR. Additionally, the performance improvement is slower than the frequency scaling because when the working frequency increases, the frame fetching speed will determine the execution time of PIM-ATW-RR which is limited by the internal bandwidth. If energy efficiency becomes the first-order constraint, it is highly preferable to find the lowest working frequency that satisfies the performance requirement of ATW. For instance, the best working frequency for achieving the best energy efficiency under different display resolutions (2880x1600, 2160x1200 and 1280x640) are 600MHz, 300MHz and 100MHz, respectively. Therefore, DVFS can be a useful tuning knob to balance performance and energy efficiency in our PIM-ATW design. Meanwhile, following the linear projection of the orange dashed line, the working power may reach closer to the thermal budget if in future we find an extremely high resolution that requires up to 1500 MHz to reach max energy efficiency while still satisfying ATW's performance demand. However, even based on future screen resolution projection, this is an extreme scenario (e.g., 2880x1600 only requires 600 MHz). In summary, we conclude that the PIM-ATW-RR design is thermal feasible and shows strong scalability in future VR scenarios.

VIII. Discussions on Alternative Designs

Software technologies which rely on computing resource visualization [63] are considered as a useful method to eliminate the asynchronous bottleneck. It is able to execute ATW asynchronously by splitting the entire GPU computing resources into rendering and ATW cores. However, modern GPU is already fully utilized by rendering workloads, allocating cores for ATW incurs performance overhead in VR applications. In addition, ATW is only processed at the end of each refresh interval. Frequently switching the core status between rendering and ATW also causes extra latency. We believe future software optimization is able to provide more efficient solution to split the GPU resources for rendering and ATW. We leave this consideration as the future work.

Building On-chip ATW units is another choice to execute ATW asynchronously with the rendering tasks. Comparing with PIM-based design, it takes lower implementation overhead to integrate extra fix-function units into the 2D GPU chip. However, as the results shown in Section VII, this design causes significant off-chip memory contentions since both ATW and VR rendering are memory intensive. *Large on-chip storages* such as eDRAM [64] and large last-level cache (LLC) [65] are possible solutions to reduce the memory contentions. Because the rendering tasks are designed to directly write the pixel color into the GPU *off-chip* memory, how to leverage the large on-chip storage to speedup ATW is out of the scope of this work.

IX. Related work

Asynchronous Time Warp. Since the MPD in VR system highly affects user's experience, many works target on reducing the response latency as much as possible by warping the rendered frame using the latest head motion position. Some works [18], [27] tend to predict head movements so that the frames can be mapped to the head position correctly while other works [12], [17], [24] try to implement ATW in different hardware platform. Comparing with previous works, our work directly erases motion anomalies by moving ATW operation from processor to PIM capacity memory cube which can efficiently executes ATW without interrupting the rendering tasks.

Virtual Reality Technology. Emerging VR technologies have been published in order to improve the rendering performance [15], [16], [66], [67], [68]. Foveated rendering technology [15], [16] which render the frames using less quality outside the eye fixation region is proposed to reduce the rendering tasks to speed up the performance of VR system. This technology trades framerate using unnoticeable image quality degradation and is getting popular in VR market. Meanwhile, pre-rendering the required frames in 3D area [66] also is introduced to reduce rendering latency. This technology represents the real-time rendering tasks as video steam. Our work also improves the frame rate by running time warp asynchronously to avoid frequently context switching. In addition, our work focuses on ATW which is orthogonal to these rendering technology.

Processing-In-Memory. Motivated by the processing in memory capability of HMC, several proposals [19], [20], [21], [22], [23] also integrate computing units in memory. Some researches in this area [19], [20] proposed to integrate GPGPU streaming multiprocessors (SMs) into 3D stacked memory. Their works offload the memory intensive GPGPU kernel into the memory to employ the high internal bandwidth of HMC. However, their architecture design still need to fetch the kernel to the host GPUs which contrasts with our idea. Meanwhile their PIM design cannot be used for ATW because ATW need to go through entire GPU pipeline not only the SM. other works [21], [22] propose HMC-based accelerators for parallel graph applications to enable fast near-data processing. However, these customized accelerators are not effective for ATW, which depends on high throughput texture unit. Xie et al. [23] design a HMC architecture which enables PIM capacity for 3D rendering. Their work could significantly improve the texture filtering performance for graphics applications. Comparing with their architecture, we not only leverage the high internal memory bandwidth to speed up ATW but also efficiency execute ATW operation in customized memory cube without interrupting the normal tasks on host GPU. To the best of our knowledge, our work is the first PIM enabled VR architecture design.

X. Conclusion

In this paper, we propose a PIM based ATW design for VR system to reduce motion-to-photon delay and avoid visual anomalies. First, we observe the state of art GPU-accelerated ATW operation causes significant preemption overhead and cannot achieve the ideal MPD of VR application. Then we investigate the ATW operation and find that moving ATW to memory not only provides high memory bandwidth but also avoids preemption overhead. Based on this insight, we construct new architecture within HMC to execute ATW operation without going through the host GPUs. Finally, we propose a redundancy reduced mechanism to accelerate and simplify the ATW operation by exploring the special two-eye characteristic of VR applications. In order to evaluate the proposed architecture, we conduct a physical platform incorporating HMC simulator framework and estimate the impact of PIM-based ATW on real VR applications. The results show that our design could reduce MPD by 175% comparing to the baseline case and achieve best user experience. In addition, by moving the ATW operation from GPUs to power efficiency logic unit, our design reduces the overall energy consumption of GPU by 16% on average. We also provide a design space exploration to illustrate multiple design choices for PIM-based ATW.

Acknowledgment

This research is supported by U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under the CENATE project (award No. 466150), The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830. This research is partially supported by National Science Foundation grants CCF-1619243, CCF-1537085(CAREER), CCF-1537062.

References

[1] C. Youngblut, “Educational uses of virtual reality technology.,” tech. rep., IDA, 1998.

[2] R. M. Satava, “Virtual reality surgical simulator,” *Surgical Endoscopy*, 1993.

[3] R. McCloy and R. Stone, “Science, medicine, and the future: Virtual reality in surgery,” *BMJ*, 2001.

[4] “Oculus rift.” https://www.oculus.com/.

[5] “Htc vive.” https://www.vive.com/us/.

[6] “Playstationvr.” https://www.playstation.com/en-us/explore/playstation-vr/.

[7] “The global virtual reality market.” http://www.strategyr.com/MarketResearch/Virtual_Reality_VR_Market_Trends.asp.

[8] “Virtual reality (vr) market analysis.” https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market.

[9] G. C. Burdea and P. Coiffet, *Virtual reality technology*. 2003.

[10] D. Zhang and Y. Luo, “Single-trial erps elicited by visual stimuli at two contrast levels: Analysis of ongoing eeg and latency/amplitude jitters,” in *ISRA*, 2012.

[11] C. Anthes, R. J. Garcia-Hernandez, M. Wiedemann, and D. Kranzlmuller, “State of the art of virtual reality technology,” in *Aerospace*, 2016.

[12] J. Van Waveren, “The asynchronous time warp for virtual reality on consumer hardware,” in *VRST*, 2016.

[13] “Asynchronous time warp on oculus rift.” https://developer.oculus.com/blog/asynchronous-timewarp-on-oculus-rift/.

[14] B. A. Watson and L. F. Hodges, “Using texture maps to correct for optical distortion in head-mounted displays,” in *VR*, 1995.

[15] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, “Towards foveated rendering for gaze-tracked virtual reality,” *ACM Trans. Graph.*, 2016.

[16] A. Patney, J. Kim, M. Salvi, A. Kaplanyan, C. Wyman, N. Benty, A. Lefohn, and D. Luebke, “Perceptually-based foveated virtual reality,” in *SIGGRAPH*, 2016.

[17] F. Smit, R. van Liere, S. Beck, and B. Froehlich, “An image-warping architecture for vr: Low latency versus image quality,” in *IEEE VR*, 2009.

[18] R. Azuma and G. Bishop, “A frequency-domain analysis of head-motion prediction,” in *SIGGRAPH*, 1995.

[19] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, “Top-pim: Throughput-oriented programmable processing in memory,” in *HPDC*, 2014.

[20] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. OConnor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, “Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems,” in *ISCA*, 2016.

[21] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, “A scalable processing-in-memory accelerator for parallel graph processing,” in *ISCA*, 2015.

[22] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, “Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules,” in *HPCA*, 2015.

[23] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, “Processing-in-memory enabled graphics processors for 3d rendering,” in *HPCA*, 2017.

[24] D. Evangelakos and M. Mara, “Extended timewarp latency compensation for virtual reality,” in *I3D*, 2016.

[25] “Latency the sine qua non of ar and vr.” http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/.

[26] U. H. List, “Nonlinear prediction of head movements for helmet-mounted displays,” tech. rep., AIR FORCE HUMAN RESOURCES LAB BROOKS AFB TX, 1983.

[27] H. Himberg, Y. Motai, and A. Bradley, “A multiple model approach to track head orientation with delta quaternions,” *SMC*, 2013.

[28] Z. Lin, L. Nyland, and H. Zhou, “Enabling efficient preemption for simt architectures with lightweight context switching,” in *SC*, 2016.

[29] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, “Enabling preemptive multiprogramming on gpus,” in *ISCA*, 2014.

[30] “Whitepaper:nvidia geforce gtx 1080.” https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf.

[31] “The importance of fine-grained gpu preemption support for vr.” https://www.imgtec.com/blog/importance-fine-grained-gpu-preemption-support-vr/.

[32] “Nvidia vrworks.” https://developer.nvidia.com/vrworks.

[33] “steamvr store.” http://store.steampowered.com/vr/.

[34] “Oculus rift store.” https://www.oculus.com/experiences/rift/.

[35] “Oculus debug tool.” https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-debug-tool/.

[36] “Gpuview.” http://systemscenter.ru/gpuview.en/.

[37] “Hybrid memory cube specification 2.0,” 2014.

[38] “Building a sensor for low latency vr.” https://www.oculus.com/blog/building-a-sensor-for-low-latency-vr/.

[39] M. Di Luca, “New method to measure end-to-end delay of virtual reality,” 2010.

[40] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, “Towards low-latency and ultra-reliable virtual reality,” *IEEE Network*, 2018.

[41] K. Akeley and P. Hanrahan, “Real-time graphics architecture,”

[42] T. SegFault, “Matrix-matrix multiplication using systolic array architecture in bluespec,” 2015.

[43] H.-J. Yoo, J.-H. Woo, J.-H. Sohn, and B.-G. Nam, *Mobile 3D graphics SoC: From algorithm to chip*. 2010.

[44] A. E. Lefohn, “Gpu memory model overview.,”

[45] “Whitepaper:asychronous shaders.” http://developer.amd.com/wordpress/media/2012/10/Asynchronous-Shaders-White-Paper-FINAL.pdf.

[46] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures,” in *MICRO*, 2009.

[47] Y. Zhu, B. Wang, D. Li, and J. Zhao, “Integrated thermal analysis for processing in die-stacking memory,” in *MEMSYS*, 2016.

[48] Y. Eckert, N. Jayasena, and G. H. Loh, “Thermal feasibility of die-stacked processing in memory,” 2014.

[49] T. Hübner, “Single-pass multi-view rendering,”

[50] T. Wilson, “High performance stereo rendering for vr,”

[51] F. De Sorbier, V. Nozick, and H. Saito, “Gpu-based multi-view rendering,” in *CGAT*, 2010.

[52] “Nvidia-nsight.” https://developer.nvidia.com/nvidia-nsight-visual-studio-edition.

[53] J. D. Leidel and Y. Chen, “Hmc-sim-2.0: A simulation platform for exploring custom memory cube operations,” in *IPDPSW*, 2016.

[54] “Hmc-sim.” http://gc64.org/?page_id=56.

[55] “Khronosgroup-vulkan-samples-timewarp.” https://github.com/KhronosGroup/Vulkan-Samples/tree/master/samples/apps/atw.

[56] S. Hong and H. Kim, “An integrated gpu power and performance model,” in *SIGARCH*, 2010.

[57] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, “Understanding the characteristics of mobile augmented reality applications,” in *ISPASS*, 2018.

[58] “Initial hybrid memory cube short-reach interconnect specification issued to consortium adopters,” 2012. Denali Memory Report.

[59] C. Zhang, T. Meng, and G. Sun, “Pm3: Power modeling and power management for processing-in-memory,” in *HPCA*, IEEE, 2018.

[60] “Tn-41-01: Calculating memory system power for ddr3.” https://www.micron.com/resource-details/3465e69a-3616-4a69-b24d-ae459b295aae. Accessed: 2016-6-24.

[61] M. Song, Y. Hu, H. Chen, and T. Li, “Towards pervasive and user satisfactory cnn across gpu microarchitectures,” in *HPCA*, 2017.

[62] “Htc vive pro.” https://www.vive.com/us/product/vive-pro/.

[63] “virtual gpu technology.” https://www.nvidia.com/en-us/design-visualization/technologies/virtual-gpu/.

[64] N. Chatterjee, M. OConnor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, “Architecting an energy-efficient dram system for gpus,” in *HPCA*, 2017.

[65] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, “Technology comparison for large last-level caches (l 3 cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram,” in *HPCA*, 2013.

[66] K. Boos, D. Chu, and E. Cuervo, “Flashback: Immersive virtual reality on mobile devices via rendering memoization,” in *MobiSys*, 2016.

[67] W. J. Lee, S. J. Hwang, Y. Shin, J. J. Yoo, and S. Ryu, “Fast stereoscopic rendering on mobile ray tracing gpu for virtual reality applications,” in *ICCE*, 2017.

[68] R. Toth, J. Nilsson, and T. Akenine-Möller, “Comparison of projection methods for rendering virtual reality,” in *HPG*, 2016.