Power Gating Strategies on GPUs

PO-HAN WANG, CHIA-LIN YANG, YEN-MING CHEN, and YU-JUNG CHENG,
National Taiwan University

As technology continues to shrink, reducing leakage is critical to achieving energy efficiency. Previous studies on low-power GPUs (Graphics Processing Units) focused on techniques for dynamic power reduction, such as DVFS (Dynamic Voltage and Frequency Scaling) and clock gating. In this paper, we explore the potential of adopting architecture-level power gating techniques for leakage reduction on GPUs. We propose three strategies for applying power gating on different modules in GPUs. The Predictive Shader Shutdown technique exploits workload variation across frames to eliminate leakage in shader clusters. Deferred Geometry Pipeline seeks to minimize leakage in fixed-function geometry units by utilizing an imbalance between geometry and fragment computation across batches. Finally, the simple time-out power gating method is applied to nonshader execution units to exploit a finer granularity of the idle time. Our results indicate that Predictive Shader Shutdown eliminates up to 60% of the leakage in shader clusters, Deferred Geometry Pipeline removes up to 57% of the leakage in the fixed-function geometry units, and the simple time-out power gating mechanism eliminates 83.3% of the leakage in nonshader execution units on average. All three schemes incur negligible performance degradation, less than 1%.

Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture—*Graphics processors*; B.m [Miscellaneous]: Design—*Management*

General Terms: Design, Performance

Additional Key Words and Phrases: GPU, power gating, leakage

ACM Reference Format:

Wang, P.-H., Yang, C.-L., Chen, Y.-M., and Cheng, Y.-J. 2011. Power gating strategies on GPUs. *ACM Trans. Architect. Code Optim.* 8, 3, Article 13 (October 2011), 25 pages.
DOI = 10.1145/2019608.2019612 <http://doi.acm.org/10.1145/2019608.2019612>

1. INTRODUCTION

Most of the modern 3D graphics processing units (GPUs) are implemented as programmable parallel processors with extremely high computing power. The high computing capability of GPUs comes at the cost of high power dissipation. NVIDIA GTX480

This research is supported in part by research grants from ROC National Science Council NSC 100-2220-E-002-015, NSC 100-2219-E-002-030, NSC 100-2219-E-002-027; Excellent Research Projects of National Taiwan University 99R80304; Macronix International Co., LTD. 99-S-C25; and Etron Technology Inc., 10R70152.

Part of the contents were published in Wang et al. [2009]. In addition to the Predictive Shader Shutdown (PSS) technique, this article proposes two new power gating strategies, Deferred Geometry Pipeline (DGP), and time-out power gating for nonshader execution units. We improve the PSS mechanism described in the IEEE CAL version to achieve more energy savings, and we also provide a more thorough literature survey. Authors' addresses: P. Wang, C. Yang, and Y. Chen, Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan (R.O.C); email: {f96922002, yangc, r95125}@csie.ntu.edu.tw; Y. Cheng, Graduate Institute of Networking and Multimedia, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan (R.O.C); email: d96944002@ntu.edu.tw.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2011 ACM 1544-3566/2011/10-ART13 \$10.00

DOI 10.1145/2019608.2019612 <http://doi.acm.org/10.1145/2019608.2019612>

can deliver 1.31T FLOPs (floating-point operations) at peak performance and consumes 376 W [Butler 2010], while AMD Radeon 5870 achieves 2.72T FLOPs at peak performance and consumes 276 W [Goodhead 2010]. The GPU has become one of the major power hogs in current computer systems.

Several low-power GPU designs have been proposed in recent years. These studies addressed the power issues of novel functional designs with lower complexity or by performing power management at the system level. Dynamic Voltage and Frequency Scaling (DVFS) and clock gating are two commonly used techniques for reducing dynamic power. One previous study [Nam et al. 2007] proposed a low-power GPU which supports DVFS for vertex shaders and rendering engines. Sohn et al. [2006] proposed a pixel-level clock gating technique that clock gates texture and blending units if a pixel fails to pass the depth test that is performed in the early stages of a rendering pipeline. Sheaffer et al. [2005] showed the effectiveness of clock gating, fetch gating, DVFS, and multiple clock domains on the thermal management of GPUs. Several studies aimed to reduce expensive external memory accesses through prefetching texture cache architecture [Igehy et al. 1998], novel rasterization architecture [Akenine-Möller and Ström 2003] or tile-based rendering [Cox and Bhandari 1997; Chen et al. 1998; Antochi et al. 2004].

Existing low-power GPU studies all focused on reducing dynamic power. However, as technology continues to shrink, leakage power will become a dominant factor [Borkar 1999]. Power gating [Chandrakasan et al. 2000] is a commonly used circuit technique to remove leakage by turning off the supply voltage of unused circuits. Power gating incurs energy overhead; therefore, unused circuits need to remain idle long enough to compensate this overheads. Several studies on CPU leakage reduction have proposed to adopt a power gating technique to shut down either the entire CPU or an individual micro-architecture component, such as caches [Kaxiras et al. 2001; Flautner et al. 2002], ALUs [Hu et al. 2004] or branch prediction units [Juang et al. 2004]. For an energy-efficient GPU design, power gating should be used jointly with DVFS for the greatest energy reduction.

This article is the first study to apply architectural-level power gating for leakage reduction on GPUs. We perform cycle-accurate analysis on the utilization of various GPU modules to understand the power gating potential of GPUs. Specifically, we look at three GPU modules: shader clusters, fixed-function geometry units, and nonshader execution units. Three architectural-level power gating policies are then proposed.

- In modern GPUs, shader clusters, which perform the computation-intensive vertex and pixel shading, are the most power-hungry components [Hong and Kim 2010]. Because of different scene complexities, such as with different numbers of objects, we find that the shader resources that are required to satisfy a user's visual perception quality, which are often measured in frames per second, vary across frames. We propose an efficient method to predict the required shader resources for the next frame, and shut off the extra shader clusters. This method is called Predictive Shader Shutdown (PSS) [Wang et al. 2009]. Our results indicate that the proposed PSS technique can eliminate up to 60% of the leakage in shader clusters with negligible performance degradation.
- We observe that the long stall time of the fixed-function geometry units results from an imbalance in the geometry and the fragment computation across batches. We propose the Deferred Geometry Pipeline mechanism (DGP) to shut off both the execution and memory circuits in the fixed-function geometry units during this idle period. Our results indicate that the proposed DGP method can remove up to 57% of the total leakage generated by the fixed-function geometry units.
- For the gaming workloads that we target in this article, we found that shader clusters are often a performance bottleneck because of the complicated shading effects

involved (e.g., per-pixel cubic reflection map); therefore, the execution units within nonshader pipeline stages are usually underutilized by anywhere from 0.5% to 50%. We propose to apply the time-out power gating approach to nonshader execution units to exploit the finer granularity of the idle cycles. Our results indicate that 83.3% of the leakage of the nonshader execution units is eliminated, on average. For the execution units with shorter idle periods, request batching could further reduce leakage by 10%.

The rest of this article is organized as follows. Section 2 introduces the GPU architecture and power gating fundamentals. Section 3 analyzes the power gating opportunities in GPUs. In Section 4, we introduce the proposed power gating strategies. The experimental results are provided in Section 5. Section 6 discusses the related work, and Section 7 concludes this work.

2. BACKGROUND

2.1. Overview of GPUs with Unified Shader Architecture

To render a 3D scene (for example, a battlefield in a First-Person Shooter game), primitives constituting the scene (usually triangles) accompanied with associated state settings, materials, and textures are issued to GPUs from applications via a 3D API, such as OpenGL. GPUs process these primitives in a pipeline manner. First is the geometry pipeline, which transforms vertex coordinates, calculates the lighting results, assembles vertices into triangles, and clips triangles outside the view port range. Next is the fragment pipeline, which generates fragments from the triangles, performs attribute interpolation, performs texturing and fog computations for each fragment, and then performs a depth test and draws blended visible pixels into the frame buffer. GPUs also employ programmable shader clusters for vertex, geometry, and fragment shader programs. Vertex shader programs perform transformation and lighting computations. Geometry shader programs generate new primitives based on the shaded ones, such as shadow volume extrusion, which create new shadow-volume primitives from the original ones. Pixel shader (also called fragment shader) programs are responsible for texturing, color sums and fog computations. Shader programs are usually written in a C-like language, for example, the OpenGL Shading Language (GLSL). Due to its programmability, shader clusters can support richer functions compared to the corresponding fixed-function pipeline stages. To maximize the utilization of shader clusters, modern GPUs implement unified shader architecture [Mantor 2007; Lindholm and Oberman 2007], where all of the shader units are shared by vertex and pixel computations.

Figure 1 shows an ATI-R600-family-like GPU with unified shader architecture. ATI R600 was announced in 2006 and was ATI's first unified shader processor design. Each unified shader cluster is composed of a group of small scalar processors, also known as shader cores, and a texture unit. The Command Processor is responsible for coordinating the execution of the different GPU modules. At the beginning of each frame, the rendering state of each pipeline stage is set according to the instructions from the driver. The arbitrator unit is in charge of dispatching jobs to shader clusters. The rest of the pipeline stages are fixed-function, which are divided into two parts: geometry and fragment. In the fixed-function geometry units, Streamer receives vertices from the application via the 3D API and sends them to Arbitrator. Primitive Assembly assembles vertices into triangles, and the assembled primitives are passed back to Arbitrator for geometry shader processing if necessary. Clipper clips the triangles that lie outside of the view frustum, and Triangle Setup calculates the line equations for each triangle. In the fixed-function fragment units, Fragment Generation generates the pixels that are inside the view frustum of each triangle. Hierarchical Z and Z/Stencil Tests perform

Fig. 1. Unified shader GPU architecture.

Fig. 2. Power gating circuits controlled by header transistor.

depth tests to remove the nonvisible pixels, while Hierarchical Z performs a rougher and faster comparison. The Interpolator unit interpolates each pixel's attributes from the vertex attributes of a triangle and sends them to the arbitrator again. Shaded fragments are sent to Raster Operations for blending and frame-buffer updating. Current GPUs generally have architectures that are similar to ATI R600 but are enhanced with more shader clusters and larger memory resources with higher bandwidth [Voicu 2008; Kanter 2008]. With the popularity of accelerating nongraphics applications on GPUs, that is, GPGPU applications, there are new APIs for GPUs such as CUDA [Nvidia 2010] and OpenCL [Khronos 2010]. Nonetheless, we believe that the architectural-level power gating mechanisms proposed in this work are also applicable to the latest GPUs.

2.2. Power Gating Fundamentals

Power gating [Chandrakasan et al. 2000] is a hardware technique that can turn off the supply voltage of a circuit block to eliminate leakage. Power gating is achieved via a header (Figure 2) or footer transistor to control the target circuit block. To power gate a circuit block, a “sleep” signal is applied to the gate of the header or footer transistor to turn off the supply voltage of the circuit block. Similarly, the “sleep” signal is deasserted to restore the voltage at the virtual V_{dd} when the target circuit block has

received a request. Power gating can remove almost all of the leakage energy of a target circuit block [Powell et al. 2000; Flautner et al. 2002; Kao and Chandrakasan 2000].

The size of the header or footer transistor determines the area overhead of power gating, which depends on the overall switching current of the target circuit block. Three times the switching capacitance is a rule of thumb for the size of the header/footer transistor [Iyer 2006], for instance, a 2.5% area overhead for an 8×8 multiplexer under a $0.25\mu\text{m}$ technology [Duarte et al. 2002]. Power gating also introduces energy overhead; therefore, a circuit block must stay in a power-gated mode (or a sleep mode) long enough to compensate for the energy overhead of the header-transistor switching. The minimum length of time needed for an idle period to save power is called the break-even time. The length of the break-even time ($T_{breakeven}$) is mainly determined by the circuit design limits [Hu et al. 2004]. Power gating also incurs the timing overhead of waking up a powered-down unit. This time is called the wake up delay (T_{wakeup}). As a result, it is necessary to determine power gating policies for when to wake up or power-down a unit. A commonly used power gating policy is to turn off a module after seeing a streak of idle cycles and to wake up the module when a new request arrives. This strategy is called the time-out policy in Hu et al. [2004] and Lu and Micheli [2001]. A more complicated policy is to adaptively change the time-out values or to predict the arrival time of the next request to wake up a sleep module in advance [Youssef et al. 2006; Lu and Micheli 2001].

3. OPPORTUNITIES FOR POWER GATING ON GPUS

In this section, we analyze the opportunities of applying power gating on GPUs. This analysis is based on the GPU architecture described in Section 2.1. Specifically, we assume that we have 6 shader clusters and that each of them contains 20 shader cores. The 3D games under evaluation are First-Person Shooter games, such as Doom 3. The details of the experimental setup are presented in Section 5.1. We first analyze the workload variation across frames. We observe that because of the different scene complexities, the shader resources required to meet the target frame rate vary. Next, we show that because of the imbalance in the geometry and the fragment computation, fixed-function geometry units stall frequently. Finally, we find that, for the First-Person Shooter games that we target in this paper, shader clusters are often the performance bottleneck because of complicated shading effects (e.g., per-pixel cubic reflection map); therefore, the execution units in those nonshader pipeline stages have low utilization.

3.1. Interframe Workload Variation

The frames per second (FPS) of a 3D game is an important factor that influences the game player's perception of smoothness and playability [Claypool and Claypool 2007]. Modern game designers usually set the target frame rate at 30 or 60 FPS, depending on the genre of the game. In general, games that requires a fast-moving pace, for example, First-Person Shooter games, need a higher FPS; other game genres, such as Role-Playing Games (RPG), require a lower moving pace and can therefore render scenes at a lower frame rate.

Because of variations in scene complexities, the shader resources that are required to satisfy the target frame rate varies across frames. Figure 3 shows the achieved FPS with the number of active shader clusters ranging from 1 to 6 for a trace from Doom 3. The FPS of each frame is the inverse of a frame's execution time, that is, how many current frames could be drawn within one second. Doom 3 is a First-Person Shooter game that requires a player to respond quickly to enemies. We can make two important observations from Figure 3. First, shader clusters are critical for performance. FPS scales with the number of active shader clusters. Second, with the same number of

Fig. 3. Performance delivered by different number of active shader clusters for Doom 3.

Fig. 4. Ideal batch pipeline.

active shader units, the achieved FPS varies across frames. For example, with 6 active shader clusters, the achieved FPS is above 240 (between the 485th and 500th frames) and below 80 (between the 200th and 250th frames). In the interval of the 485th to 500th frames, the scene becomes dark, and the player's field of view is toward the floor; as a result, there are fewer scene objects. Therefore, in this period, if the target frame rate is 60FPS, two shader clusters are sufficient to sustain the desired performance, and the rest of the shader clusters can be turned off. However, in the interval of the 200th to the 250th frame, the scene is more complicated, with several light sources and a wider field of view. Thus, it takes four shader clusters to deliver 60 FPS. In this period of time, two shader clusters can be power gated to save leakage. This analysis indicates that the workload variation among the frames can be exploited to shut down redundant shader clusters for the purpose of leakage reduction.

3.2. Interbatch Stall in the Geometry Pipeline

In OpenGL, a frame is composed of batches; a batch is a group of triangles that apply the same rendering configurations, such as the type of primitives (e.g., triangle strip and triangle fan) [Cebenoyan 2004]. Before processing a new batch, a GPU needs to update configuration registers for the new rendering setting. To achieve higher resource utilization, GPUs pipeline batch processing between the geometry and fragment computation, as shown in Figure 4. When the geometry units finish processing batch N, they could then be configured to process batch N+1 while the fragment units are still processing batch N, as shown in Figure 4.

Fig. 5. Fixed-function geometry units stall due to the imbalance in geometry and fragment computation.

Fig. 6. Cumulative distribution of interbatch stall percentage.

Because the fragment processing consists of Z testing, texturing, per-pixel shading, and color blending, which in general require more computation than geometry processing and generate a substantial amount of memory access, fixed-function geometry units often need to stall until fragment computation of the previous batch completes. This scenario is illustrated in Figure 5. We call these stalls interbatch stalls. To show the significance of interbatch stalls, we begin by quantifying the stall time as a percentage of a batch's execution cycles, and we plot the cumulative distribution in Figure 6. Figure 6 shows that approximately 50% of the batches have more than 70% of the interbatch stall cycles. We also observe that, in most cases, each stall period lasts for hundreds of cycles. As shown in Figure 7 which plots the cumulative distribution of the interbatch stall length, 80% of these stalls are longer than 600 cycles. Based on this analysis, we know that utilizing the interbatch stall for power gating has a high potential for reducing leakage power of the fixed-function geometry units.

3.3. Underutilization of Nonshader Execution Units

For the workloads tested in this work (i.e., First-Person Shooter games), we observe that we have a very low utilization of execution units in fixed-function geometry and fragment units (i.e., pipeline stages inside the dotted lines in Figure 1). To obtain the utilization of the nonshader execution units, we observe the status (busy or idle) of the execution units of a pipeline stage at each cycle. If a pipeline stage contains multiple execution units, then the pipeline stage is considered busy if at least one execution unit is active. The utilization of a pipeline stage is determined by dividing the busy cycles by the total execution cycles and is provided in Table I. Please note that we only report the utilization of execution units. There are other modules in these fixed-function pipeline

Fig. 7. Cumulative distribution of interbatch stall length.

Table I.

The utilization of nonshader execution units. STR: Streamer, PA: Primitive Assembly, CLIP: Clipper, TS: Triangle Setup, FG: Fragment Generation, HZ: Hierarchical Z, INT: Interpolator, ZST: Z/Stencil Test, ROP: Raster Operation

Pipeline Stage	STR	PA	CLIP	TS	FG	HZ	INT	ZST	ROP
Utilization(%)	0.24	0.63	1.71	1.20	10.81	11.48	39.90	49.44	2.33

stages, for instance, the Z-cache in the Z/Stencil Test (ZST) and the frame-buffer cache in the Raster Operation (ROP). Therefore, the utilizations of these stages are actually higher than the number reported in this table. From Table I, we can see that the utilizations of the geometry processing stages (i.e., STR, PA, CLIP, TS) are very low. As mentioned in Section 3.2, geometry processing usually requires less computation than fragment processing; as a result, the corresponding pipeline stages often stall. Execution units in the fragment pipeline stages have higher utilization compared to the geometry stages, but their utilizations are still low. The highest utilization is only 50%, which occurs in the ZST. The lowest utilization occurs in the ROP and is only 2.5%. As mentioned earlier, the First-Person Shooter games targeted in this paper implement complicated shading effects, such as the per-pixel cubic reflection map. As a result, each pixel usually takes more than one instruction for its shading computation; thus, the shader clusters are unable to deliver peak throughput, which would keep the ROP fully utilized.

Figure 8 shows the cumulative distribution of idle length in nonshader execution units in each fixed-function pipeline stage. We can see that long idle periods dominate in the stages that have very low utilization, such as STR, PA, CLIP, TS, and ROP. More than 95% of the idle periods are longer than 100 cycles. For those stages that have a higher utilization, there are more short idle periods. For example, in Fragment Generation (FG), Hierarchical Z (HZ), and ZST, 30% of the idle periods are less than 60 cycles. However, approximately half of the idle periods are still longer than 100 cycles. Note that part of the idle time of STR, PA, Clip and TS can be attributed to the interbatch stall mentioned earlier. According to our experiments, the interbatch stall accounts for 60% of the total execution cycles. Because the utilization of these four stages is below 2%, as shown in Table I, we know that, in addition to the interbatch stall, there are still a significant number of idle cycles that could be exploited on the execution units of these four stages.

Fig. 8. The cumulative distribution of idle length in non-shader execution units in each fixed-function pipeline stage. Y-axis is $\frac{\sum_{i=0}^n (\text{idle cycles of group } i)}{\text{total idle cycles}}$.

4. POWER GATING STRATEGIES

Based on the analysis in the previous section, we propose three power gating strategies. The Predictive Shader Shutdown technique exploits workload variation across frames to reduce leakage in the shader clusters. Deferred Geometry Pipeline seeks to minimize the leakage in fixed-function geometry units by utilizing the imbalance between geometry and fragment computation across batches. Finally, the simple time-out power gating method is applied to nonshader execution units to exploit finer granularity in the idle time.

4.1. Predictive Shader Shutdown (PSS)

The purpose of the proposed Predictive Shader Shutdown (PSS) technique is to predict the minimum required shader resources to meet the target frame rate for the upcoming frame and to shut down extra shader clusters. PSS adopts a history-based prediction method to predict the required shader clusters for the next frame. We estimate the delivered FPS of one shader cluster (FS_i) by assuming that there is a linear correlation between the FPS and the number of active shader clusters ($Shader_i$) in $frame_i$ for simplicity. Therefore, we observe FS_i in the past m frames, as follows:

$$FS_i = \frac{FPS_i}{Shader_i}. \quad (1)$$

We then predict the delivered FPS of one shader cluster for the next frame, FS_{n+1} , by the following formula:

$$FS_{n+1} = \begin{cases} FS_n - \frac{FS_{n-m+1} - FS_n}{m-1}, & \text{if } FS_n < FS_{n-1} < \dots < FS_{n-m+1}, \\ \min(FS_n, FS_{n-1}, \dots, FS_{n-m+1}), & \text{otherwise.} \end{cases} \quad (2)$$

We design the prediction function in favor of the performance because the since performance is critical for gaming. If FS_i in the past m frames are monotonically decreasing, then the game scenes are becoming complicated. Therefore, we predict FS_{n+1} by extrapolating FS_i from the past m frames (i.e., the first case in Formula (2)). However, in other cases where FS_i in the past m frames is monotonically increasing or in random, in favor of the performance, the minimal delivered FPS of one shader

Fig. 9. Ideal power gating situation applying Deferred Geometry Pipeline.

cluster in the past m frames is selected as the prediction of FS_{n+1} . Then the shader clusters that are required to meet the target frame rate for the next frame, $Shader_{n+1}$, can be obtained by the following:

$$Shader_{n+1} = \left\lceil \frac{Target\ FPS}{FS_{n+1}} \right\rceil. \quad (3)$$

The prediction function of PSS is simple and could be implemented in the GPU driver, which records that past frames' FPS and the number of active shader clusters. At the beginning of each frame, along with other render state settings, the driver also notifies the Command Processor with the number of shader clusters required for the upcoming frame. As a result, control of the sleep signals of the shader clusters can be performed by the Command Processor. Because setting the rendering states between frames usually takes approximately 200 to 500 cycles, according to our experiments,¹ this delay can be used to tolerate the wake-up delay of power-gated shader clusters.

4.2. Deferred Geometry Pipeline (DGP)

To take advantage of the interbatch stall cycles described in Section 3.2, we could shut down the execution units in fixed-function geometry units once a stall is detected. However, memory structures (such as configuration registers, post-transform vertex cache in the streamer unit, and pipeline latches) cannot be turned off. The leakage of these memory structures cannot be neglected. The post-transform vertex cache in the streamer unit stores the attributes of shaded vertices for the purpose of order not reshading the shared vertices within a batch [Sander et al. 2007]. We assume that there is a 512-entry post-transform vertex cache in the streamer unit. The number of triangles that constitute a batch in the tested workloads is usually up to one thousand. Because vertices are often organized in triangle meshes, there is a significant amount of reuse. We found that a 512-entry vertex cache can capture most of the vertices in a batch. We evaluated the effect of different vertex cache sizes ranging from 64 to 1024 entries. The results indicate that a cache size larger than 512 has little impact on performance. Because each vertex could have up to 8 attributes represented in 4 single-precision floating-point numbers, a 512-entry post-transform vertex cache is approximately 64 KBs. Latches between pipeline stages could be up to 12 KBs, assuming 32-entry input/output queues, and configuration registers are roughly 2 KBs. As a result, the leakage of memory resources is expected to be a significant part of the leakage of the fixed-function geometry units.

Therefore, to power gate the whole fixed-function geometry units including both execution and memory modules, we propose to defer the processing of the next batch, as shown in Figure 9. We call it Deferred Geometry Pipeline (DGP). The concept behind DGP is to turn off the fixed-function geometry unit completely once it finishes

¹The texture unit is the most complicated component which has up to 500 states.

processing batch N and to wake it up just in time to avoid delaying the processing of the next batch in the fragment pipeline. In this way, memory modules, such as the post-transform vertex cache and configuration registers, have not yet been configured for the next batch, therefore, they can be power gated as well. To predict the optimal wake-up point, higher-level application information may need to be exploited. In this article, we adopt a simple heuristic to trigger the wake-up signal, which occurs when the fragment shaders start processing the last pixel of the previous batch.

The assertion and deassertion of the sleeping signals in DGP are controlled by the fixed-function geometry stage itself and the Command Processor, respectively. In a GPU, the end of a batch is usually annotated with a dummy vertex/pixel to allow a GPU to setup rendering states between batches. As a result, the fixed-function geometry units can detect the end of a batch and assert a sleep signal immediately. The wake-up signal is controlled by the Command Processor. As mentioned above, the wake-up event in DGP is the last pixel of a batch passing through the fragment shaders. Because the Command Processor is responsible for coordinating the execution of different GPU modules, it monitors the states of every pipeline stage. Thus, it can detect the wake-up event and de-assert the sleep signals of the fixed-function geometry units.

4.3. Time-Out Power Gating for Nonshader Execution Units

To exploit the low utilization of nonshader execution units, we propose to use the time-out power gating policy with request batching. Time-out power gating turns off a circuit block after seeing a streak of idle cycles ($T_{idledetect}$), and wakes it up when a new request arrives [Lu and Micheli 2001; Hu et al. 2004]. A larger value for $T_{idledetect}$ sets a device to sleep mode more cautiously but results in energy waste during the time-out period. In general, workloads with longer idle periods are less sensitive to $T_{idledetect}$. Figure 8 indicates that most of the pipeline stages have very long idle periods; therefore, a simple fixed time-out policy is expected to be effective. For those pipeline stages that have shorter idle periods, such as FG, HZ, and ZST, request batching could be utilized to achieve greater reduction in leakage. Instead of activating a sleeping unit immediately after seeing a new request, request batching gathers short idle periods into a longer period by delaying the wake-up action until the input buffer of the next pipeline stage is below a predefined threshold. As mentioned earlier, the execution units of nonshader pipeline stages are usually not the performance limiter; therefore, the performance degradation incurred by request batching should be negligible.

Note that the interbatch stall mentioned previously could also be exploited by the time-out power gating policy. However, the time-out power gating method can only power gate the execution units of fixed-function geometry units, whereas DGP turns off both the execution and memory units. Furthermore, as described in Section 3.3, in addition to inter-batch stall, there still exists a significant amount of idle time. Therefore, using DGP and time-out power gating together could more fully utilize the idle time in the fixed-function geometry units to achieve leakage reduction.

The hardware support for the time-out power gating policy is simple. Each execution unit of the nonshader pipeline stages is associated with a counter and a comparator, to decide whether its idle cycles have exceeded the time-out threshold. The sleep signal is asserted when the time-out condition is met; and is de-asserted when a new request arrives.

5. EXPERIMENTAL RESULTS

5.1. Experimental Setup

We use ATTILA [del Barrio et al. 2006], a cycle-level execution-driven GPU simulator, to conduct the experiments of this study. The GPU core that we model is close to ATI

Table II. Bandwidth and Latency of GPU Pipeline Stages

Stage	Input Bandwidth	Output Bandwidth	Latency	Output-Buffer Size
STR	2 indices	2 vertexes	Mem	512 vertexes
PA	2 vertexes	2 triangles	1	32 vertexes
CLIP	2 triangles	2 triangles	6	32 vertexes
TS	2 triangles	2 triangles	10	32 vertexes
FG	2 triangles	4*64 fragments	1	4*512 fragments
HZ	4*64 fragments	4*64 fragments	1	4*512 fragments
ZST	4 fragments	4 fragments	2+Mem	4*32 fragments
INT	4*4 fragments	4*4 fragments	4 to 10	4*16 fragments
ROP	4 fragments		2+Mem	

Table III. Description of Game Traces

Game/Timedemo	Resolution	# of Frames	Engine	Release Date
Doom 3/trdemo2	1280 × 1024	2398	Doom 3	Aug-04
Quake 4/demo4	1280 × 1024	2270	Doom 3	Oct-05
UT2004/Primeval	1280 × 1024	2759	Unreal 2.5	Mar-04
Riddick/PrisonArea	1280 × 1024	2483	Starbreeze	Dec-04

RV635, a version of ATI's 1st unified-shader architecture R600 [Mantor 2007]. Our GPU has 6 shader clusters, each of which contains 20 shader cores. The operating frequency of our GPU is 600 MHz. Each pipeline stage is able to process multiple data(index/vertex/fragment) per cycle. The detailed architectural parameters assumed in this study are summarized in Table II. We evaluate four 3D game traces: Doom 3, Quake 4, Riddick, and Unreal Tournament 2004 (UT2004), which were released with the ATTLA simulator. These traces are generated from the official demos that are published with 3D games. The details of these game traces, for instance, the resolutions, are shown in Table III. We run only 700 consecutive frames for each trace for the results presented in this section because of the long simulation time. These 700 frames are carefully selected to ensure that all of the shader programs for the demo traces are executed at least once.² Because the interceptor of ATTLA, which captures the API call sequence and the data used by OpenGL, is not an open-source library and supports only a subset of OpenGL APIs, we are not able to evaluate the proposed schemes on game traces other than those released with ATTLA. However, we do observe workload variations among the frames for the latest 3D games from a test report provided by PC Games Hardware [Spille et al. 2010]. In the report, 11 3D games were tested and the differences between minimum FPS and average FPS are 18 frames on average. For example, the report plots the FPS of a two-minute demo of the game "Armed Assault 2" and finds that the frame-rate ranges from 10 to 55. As a result, we believe that the power gating mechanisms proposed in this work are still applicable to the latest 3D games.

To evaluate the leakage reduction effect on target modules of the proposed power gating policies, we adopt a previous approach [Hu et al. 2004] that uses the power-gated cycle ratio to approximate the leakage reduction ratio. For the results presented in this section, we assume that 99.7% of the leakage can be eliminated on a power-gated module [Kao and Chandrakasan 2000; Flautner et al. 2002]. Therefore, the leakage reduction ratio is given by:

$$\frac{\sum_{\text{intervals}}^{all \text{ power-gated}} (T_{\text{power-gated}} - T_{\text{breakeven}} - T_{\text{wakeup}})}{\text{total execution cycles}} \times 99.7\%. \quad (4)$$

²The shader programs used in each frame are published with the traces.

Table IV.
The leakage reduction and the average shader per cycle (ASPC)

	Doom 3	Quake 4	UT2004	Riddick
Potential (%)	45.5	14.8	62.0	15.3
ASPC	3.20	4.82	2.24	5.03

Without the actual GPU implementations, the break-even time and wake-up delay for the proposed power gating techniques could only be roughly estimated. Because the idle periods exploited by PSS and DGP are very long, these two mechanisms are less sensitive to the break-even time and the wake-up delay. Therefore, we assume fixed values when evaluating these two mechanisms. We set the break-even time for power gating a shader cluster to be 1/60 second, that is, 10 million cycles for a 600-MHz shader core. We assume that the associated wake-up delay is less than 200 cycles; thus, a power-gated shader cluster can be activated early enough for the next frame, as discussed in Section 4.1. Compared to a previous study [Duarte et al. 2002] that estimated the energy and performance overheads of power gating on various types of functional units, we believe that the break-even time and the wake-up delay assumed for the shader cluster are likely to be overestimated; as a result, the leakage reduction of PSS reported in the next section is not inflated. For DGP evaluation, because the largest component in those stages is the post-transform vertex cache (approximately 64KB), we assume that the break-even time of power gating the fixed-function geometry units is 80 cycles by scaling the values in Kalla et al. [2006] and that the wake-up delay is 100 cycles, which is the same as a shader cluster. Because the time-out power gating technique exploits finer granularity in the idle time, the break-even time and the wake-up delay could have a significant impact on the amount of leakage reduction achieved. Therefore, we perform detailed analysis on how different break-even times and wake-up delay settings affect the effectiveness of the time-out power gating method. The evaluated parameters are similar to those in [Hu et al. 2004], which targeted the ALUs of the GPP (General Purpose Processor).

5.2. Effects of Predictive Shader Shutdown

Before presenting the shader leakage reduction achieved by the proposed shader shutdown technique, we first show the potential leakage reduction while assuming that the minimal number of shader clusters per frame to meet the target frame rate is known a priori. For the results presented in this section, the target frame rate is set to 60 FPS, and the history of the last 3 frames³ is used in Formula (2). We evaluate the leakage reduction on shader clusters using the following metric:

$$1 - \frac{\sum_{i=1}^n PSS_shader_i}{\sum_{i=1}^n baseline_shader_i}, \quad (5)$$

where i is the cycle number and PSS_shader_i represents the number of active shader clusters of the i^{th} cycle using PSS. The baseline assumes that the GPU runs at full speed (i.e., with all shader clusters on).

We also show the average shader per cycle (ASPC) to quantify the workload. This average is given by the following metric:

$$\frac{\sum_{i=1}^n PSS_shader_i}{n}. \quad (6)$$

Table IV shows the potential leakage reduction on shader clusters and the ASPC for 4 game traces. UT2004 has the highest potential leakage reduction on shader clusters,

³A 3-frame history window is chosen through experiments on 5 different window sizes ranging from 1 to 5.

Fig. 10. Performance with various amount of active shader clusters for Quake 4.

Fig. 11. Prediction accuracy of PSS.

62.0%, and its average shader per cycle is 2.24. Quake 4 and Riddick are heavier workloads, so the potential leakage reduction is approximately 15%. We plot the FPS of each frame with different numbers of active shader clusters for Quake 4 in Figure 10. Compared to Figure 3, which shows the FPS variation of Doom 3, we observe that the frame rate corresponding to 6 shader clusters for Quake 4 is much lower than that for Doom 3. The FPS of the first 90 frames is even lower than the target frame rate with all of the shader clusters on. Although these traces are heavy in general, there are still opportunities to turn off the shader clusters, such as from the 105th to the 287th frame, where 4 shader clusters are sufficient to deliver the target frame rate.

Before presenting the actual leakage reduction achieved by PSS, we first discuss the accuracy of the prediction function in PSS. Figure 11 shows the frame distribution according to the differences between the predicted shader number and its oracle value. We can see that, in 64% of the frames, the PSS mechanism correctly predicts the required shader resources, and in 32% of the frames, PSS overestimates the resources by one. We can also see that underestimation rarely occurs (less than 1% of the frames). The misprediction of PSS arises from two sources: one source is the assumption about the linear correlation between FPS and the shader resources (Formulas (1) and (3)) and the other source is the workload prediction (Formula (2)). We plot the delivered FPS

Fig. 12. Average delivered performance of various amount of active shader clusters. (Normalized to one active shader cluster).

Table V.
The potential and the actual leakage reduction of predictive shader shutdown

	Doom 3	Quake 4	UT2004	Riddick
Potential (%)	45.5	14.8	62.0	15.3
PSS (%)	40.8	10.7	60.8	6.7

with different numbers of active shader clusters in Figure 12. We can see that, with fewer active shader clusters, their correlation is close to linear. However, as the number of active shader clusters increases, the system bottleneck shifts to other components, for instance, memory bandwidth. Therefore, increasing the number of shader clusters results in less throughput improvement. However, there is still a positive correlation. This scenario explains why PSS only overestimates the required shader clusters by one in most cases. Underestimation rarely occurs, mainly because of our workload prediction function. When game scenes are becoming complicated, PSS attempts to increase the active shader cores. If the linear correlation between the FPS and the shader resources does not hold, then the predicted FPS is higher than the actually delivered FPS. In this case, PSS may underestimate the required shader clusters for the target frame rate. However, PSS continues to increase the active shader clusters until the target frame rate is achieved. In the first equation of the workload prediction function, if FS_i in the past m frames is monotonically decreasing (i.e., the scenes are becoming more complicated), PSS predicts FS_{n+1} by extrapolating FS_i in the past m frames. The result is that PSS may add one shader cluster in advance to prevent possible upcoming underestimation. Therefore, underestimation rarely occurs.

Table V shows the actual leakage reduction of PSS. We also show the potential leakage reduction for a comparison. For Doom 3 and UT2004, PSS achieves a 40.8% and 60.8% leakage reduction, respectively. For heavier workloads, that is, Quake 4 and Riddick, the leakage reduction is less significant, less than 10%. To understand how much total GPU leakage can be eliminated by PSS, we assume that the leakage generated by shader clusters is roughly close to their area ratio.⁴ As mentioned in

⁴The shader clusters may yield higher leakage than other GPU modules due to more complex datapaths or stricter timing requirements. However, we do not have a detailed leakage breakdown for GPUs so we still assume uniform leakage across the logics to avoid overestimating the leakage gain obtained from our schemes.

Fig. 13. Die photo of ATI RV770.

Section 5.1, our target architecture is close to ATI RV635. ATI RV635 is implemented in 55nm, and its die size is 132mm^2 [Beyond3D 2008]. However, its die photo is not available. As a result, we refer to the die photo of ATI RV770 [Voicu 2008] (Figure 13), which has the shader and texture architecture that is similar to ATI RV635 and is implemented with the same process technology. The die size of ATI RV770 is 260mm^2 . We can see from the die photo that the shader cores (i.e., the SIMD Cores in Figure 13,) and the texture units of RV770 occupy approximately 30%, and 12.2% of the die area, respectively. Based on these ratios and on the number of shader cores and texture units of ATI RV770 (800 and 40), we can then derive that the size of a shader core is 0.0975mm^2 and that the size of a texture unit is 0.793mm^2 . Because the architectures of the shader core/texture units are close in RV770 and RV635, we assume that their sizes are also close in both architectures. Based on the number of shader cores/texture units (120/24) and the die size of RV635, we can derive that the area percentage of its shader clusters is roughly 23.3%. The result is that the leakage generated by the shader clusters is approximately 23.3% of the total chip leakage. With a reduction of 60% in the shader leakage, PSS can eliminate up to 14% of the total GPU leakage. If we assume that the leakage constitute approximately 50% of the overall energy consumption under 55nm technology [ITRS 2006], PSS can save up to 7% of the total energy consumption. We expect that the overall leakage reduction rate achieved by PSS will grow as the GPU vendors continue to put additional shader resources into a chip. For example, in the latest GPU of Nvidia, GT200 [Kanter 2008], the shader clusters occupy 43% of the chip area. Additionally, as the technology continues to shrink, leakage will contribute more to the total chip power. As a result, the leakage reduction effect of PSS is anticipated to increase for future GPUs.

Table VI. The Percentage of Frames Meeting the Performance Constraint (60 FPS)

	Doom 3	Quake 4	UT2004	Riddick
Baseline (%)	100.0	84.4	99.9	87.4
PSS (%)	99.5	82.4	99.0	86.9

Table VII. The Average Shader On/Off Frequency and the Power Gating Overheads

	Doom 3	Quake 4	UT2004	Riddick
Average Shader On/Off Frequency (frame)	24.4	14.4	26.2	37.9
Overhead (%)	0.31	0.53	0.51	0.19

Table VIII.

Potential, actual leakage reduction of the fixed-function geometry units and the performance degradations achieved by DGP

	Doom 3	Quake 4	UT2004	Riddick
Potential Leakage Reduction (%)	74.09	60.32	28.92	54.88
Actual Leakage Reduction (%)	55.80	57.34	27.19	52.89
Performance Degradation (%)	0.41	0.17	0.29	0.39

Table VI compares the performance (measured as the percentage of frames meeting the 60 FPS constraint) of the baseline (with all of the shader clusters on) with PSS. We can see that our mechanism causes almost no performance loss (approximately 1%). Table VII shows the average shader on/off frequency and the energy overhead incurred by turning on/off shader clusters. We can see that for UT2004 which has the highest leakage reduction from PSS, the shader number changes roughly every 26 frames, and the power gating overhead is only 0.5% of the baseline energy consumption. Hence, the effect of the break-even time of PSS is negligible.

5.3. Effects of Deferred Geometry Pipeline

Table VIII shows the potential/actual leakage reduction and the incurred performance degradation of DGP. The potential leakage reduction is obtained by assuming that the fixed-function geometry units are awoken sufficiently early such that no delay is incurred in the fragment pipeline. Our results indicate that the potential leakage reduction on the fixed-function geometry units that is achieved by DGP ranges from 28.92% to 74.09%. Doom 3 presents the largest potential for leakage reduction. Doom 3 has a highly complicated fragment computation (e.g., more texture access per fragment); thus, Doom 3 exhibits the largest computational imbalance between the geometry and the fragment computation.

From Table VIII, we can observe that the actual leakage reduction that is achieved by DGP is close to the potential leakage reduction discussed above, with less than a 0.5% performance degradation, with the exception of Doom 3. The DGP mechanism wakes up the fixed-function geometry units when the fragment shader starts processing the last pixel of the current batch. Because Doom 3 has the heaviest fragment computation among the game traces tested, the fixed-function geometry units are awoken too early in most cases. That feature explains why Doom 3 has the largest discrepancy between the potential and the actual leakage reduction. Higher level information, such as information related to the fragment shader program's complexity, may be exploited to determine the wake-up time of the fixed-function geometry units more accurately.

Fig. 14. Power gating potential of nonshader execution units with various $T_{overhead}$.

5.4. Effects of Time-Out Power Gating and Request Batching

Because the time-out power gating exploits the finer granularity of the idle time, to evaluate its effectiveness, we study in detail how $T_{idledetect}$, $T_{breakeven}$ and T_{wakeup} affect the achieved leakage reduction. We first show the leakage reduction potential, assuming that the length of the idle period is known a priori such that the circuit can enter the power-gated mode without waiting for a period of time and can also wake up in advance without incurring performance degradation. For power gating potential analysis, $T_{breakeven}$ and T_{wakeup} are considered together as power gating overhead ($T_{overhead}$). Figure 14 shows the power gating potential for each stage with different $T_{overhead}$ values for Quake 4.⁵ The stages that have a very low utilization (Table I), such as STR, PA, CLIP, TS, and ROP, have a high leakage reduction potential, between 90% and 100%. ZST has the highest utilization. As a result, ZST achieves the lowest leakage reduction, from 45% to 60%. The sensitivity to $T_{overhead}$ depends on the length of the idle periods at each stage. As indicated in Figure 8, long idle periods dominate in STR, PA, CLIP, and TS; thus, they are not sensitive to $T_{overhead}$. FG, HZ and ZST have more short idle periods, so $T_{overhead}$ has a larger effect on their potential leakage reduction.

After quantifying the power gating potentials, we now show the actual leakage reduction and the impact on performance achieved by time-out power gating with different $T_{idledetect}$. Figure 15 shows the leakage reduction of STR, FG, INT and ZST with various $T_{idledetect}$ and $T_{breakeven}$ (T_{wakeup} is fixed at 6 cycles). Other stages show similar results, so they are omitted from this paper. The results show that the leakage reduction achieved by STR and INT are insensitive to $T_{idledetect}$ because of extremely long idle cycles (97% > 100 cycles), as indicated in Figure 8. Therefore, energy waste during the time-out period is negligible. We also observe that the leakage reduction of FG decreases as $T_{idledetect}$ increases. FG has more short idle periods compared to other stages, but the majority of them are still longer than the break-even time. The idle periods shorter than 24 cycles (i.e., the longest break-even time tested in Figure 8) are less than 5%. Thus, the advantage of entering sleep mode more accurately with a larger $T_{idledetect}$ value is not able to compensate for the energy waste during the time-out period. In the case of ZST, the leakage reduction is similar or slightly increases (for a 24-cycle break-even time) as $T_{idledetect}$ increases. Compared to FG, ZST has more idle periods that are shorter than the break-even time (approximately 20% of the idle periods that

⁵Other game traces also show similar results. Therefore, we only present results for one trace in this paper.

Fig. 15. Leakage reduction of time-out power gating. The leakage reduction of STR, FG, INT and ZST are reported with different $T_{idledetect}$ and $T_{breakeven}$. T_{wakeup} is fixed at 6 cycles.

Table IX.
Performance degradation of time-out power gating with various T_{wakeup} . $T_{idledetect}$ and $T_{breakeven}$ are fixed at 9 cycles

T_{wakeup}	3	6	9
Normalized Performance (%)	99.9	99.6	99.3

are shorter than 24 cycles). Thus, the advantage of a larger $T_{idledetect}$ begins to become apparent.

Table IX shows the performance of time-out power gating that is normalized to the baseline GPU with different T_{wakeup} values. Because the time-out power gating method does not wake up a sleeping device in advance, a longer wake-up delay may cause more performance degradation. However, the results indicate that the performance degradation due to the wake-up delay is negligible. As mentioned earlier, the performance of the nonshader pipeline stages is limited by a predecessor and successor stage, as well as by possible memory access, rather than by the execution units. Therefore, the delay in the nonshader execution units could be overlooked in terms of the overall GPU performance.

Next, we show how request batching affects leakage reduction and performance.⁶ The leakage reduction of request batching is plotted in Figure 16 with the same

⁶In the following experiments, we assume request batching delays the wake-up action until the input buffer of the next pipeline stage is below 10%.

Fig. 16. Leakage reduction of time-out power gating plus request batching. The leakage reduction of STR, FG, INT and ZST are reported with different $T_{idledetect}$ and $T_{breakeven}$. T_{wakeup} is fixed at 6 cycles.

Table X.
Performance degradation of time-out power gating plus request batching with various T_{wakeup} . $T_{idledetect}$ and $T_{breakeven}$ are fixed at 9 cycles

T_{wakeup}	3	6	9
Normalized Performance (%)	99.2	98.9	98.4

parameter settings as in Figure 15. As expected, request batching does not increase the leakage reduction of STR and INT because the idle periods of these two stages are already very long even in the absence of request batching. Request batching increases the leakage reduction of FG and ZST because these two stages have more short idle periods. Furthermore, FG is less sensitive to $T_{idledetect}$ with request batching. Table X and Table IX shows the performance of time-out power gating normalized to the baseline GPU with different T_{wakeup} values with and without request batching, respectively. From Table X and Table IX, we can see that the performance degradation incurred by request batching is negligible.

Table XI summarizes the average leakage reduction on nonshader execution units by the time-out policy with/without request batching for the 4 game traces. Without request batching, the leakage reduction ranges from 79.08% to 97.91%. Request batching reduces leakage further by 7% for FG and HZ and by 5% for ZST.

Table XI.
Average leakage reduction of time-out power gating plus request batching across four traces. $T_{idle\ detect} = 9$, $T_{break\ even} = 9$ and $T_{wake\ up} = 6$

Pipeline Stage	time-out (%)	time-out + request batching (%)
STR	97.91	97.98
PA	96.77	96.88
CLIP	96.70	96.82
TS	97.74	97.83
FG	85.23	92.70
HZ	85.26	92.72
INT	56.55	58.77
ZST	61.08	65.52
ROP	79.08	80.12

Table XII.
Leakage reduction of shader clusters while three power gating policies are applied together

	Doom 3	Quake 4	UT2004	Riddick
PSS Leakage Reduction (%)	40.3	10.0	60.7	6.2

Table XIII.
Leakage reduction of fixed-function geometry units while three power gating policies are applied together

	Doom 3	Quake 4	UT2004	Riddick
DGP Leakage Reduction (%)	63.7	60.9	32.2	52.3

5.5. Joint Effect of Three Power Gating Policies

In this section, we present the joint effects of PSS, DGP and time-out power gating with request batching. $T_{idle\ detect}$, $T_{break\ even}$, and $T_{wake\ up}$ are fixed at 9, 9, and 6 cycles, respectively, for the results presented in this section.

Table XII shows the leakage reduction on the shader clusters that is achieved by PSS. These results are the same as those presented in Table V. According to the performance analysis of DGP and the time-out power gating, neither of these two mechanisms causes performance degradation. Therefore, PSS performs the same with/without DGP and time-out power gating. However, because PSS reduces the active number of shader clusters, the leakage reduction achieved by DGP (shown in Table XIII) is more than that when DGP is used alone. Recall that DGP exploits the imbalance between the geometry and the fragment computations. Having fewer shader clusters exacerbates this imbalance. Figure 17 shows the leakage reduction of the time-out power gating on the nonshader execution units. Compared to Table XI, except for STR, PA, CLIP and TS, time-out power gating achieves greater leakage reduction. With PSS, less computation is performed by the GPU per unit time. This relationship leads to even lower utilization in fixed-function pipeline stages. In contrast, STR, PA, CLIP and TS show much lower leakage reduction compared to the results presented in Section 5.4. STR, PA, CLIP and TS are the fixed-function geometry units; thus, parts of the idle time are exploited by DGP. However, as discussed in Section 4.3, in addition to inter-batch stalls, there are still some idle times on these three stages that could be further utilized by time-out power gating. According to our results shown in Figure 17, time-out power gating could still achieve a 35% leakage reduction on STR, PA, CLIP, and TS.

6. RELATED WORK

Prior research has studied the workload variation of interactive computer games on general purpose processor platforms, including the use DVFS to reduce dynamic energy dissipation. Gu et al. [2006] studied the workload variations of a First-Person Shooter game to motivate the use of DVFS. They used a frame structure-based predictor to

Fig. 17. Average leakage reduction of time-out power gating and request batching across four traces, while applying PSS and DGP together.

predict future workloads for the DVFS. They also proposed a control theory-based workload predictor that periodically adjusts the game workload prediction based on the feedback from recent prediction errors [Gu and Chakraborty 2008a]. To utilize the advantages of two workload predictors, Gu and Chakraborty [2008b] proposed a hybrid approach that uses the frame structure-based predictor to compute the frame's workload with high variation and switches to a control-theory-based predictor when the workload goes flat.

In recent low-power GPU designs, DVFS is also a commonly used technique. Sheaffer et al. [2004] proposed monitoring the fragment queue between the vertex and the fragment engines to perform DVFS on the engines. When the queue is full, the GPU is fragment-bound, and the vertex engine could reduce its voltage and frequency, and vice versa. Mochocki et al. [2006b] observed imbalances among Geometry, Triangle Setup, and Rendering stages to motivate the use of DVFS, and they proposed a signature-based workload prediction scheme that estimates the next frame's workload from the frame history and attributes of the current frame (e.g., the triangle count and the triangle size) in [Mochocki et al. 2006a]. Note that these observations [Mochocki et al. 2006a, 2006b] are made on ARM embedded processors. Nam et al. [2007] proposed a GPU architecture for hand-held devices with three power domains, and they adjusted the frequency and supply voltage of each power domain with a history-based predictor.

Several studies focused on novel architectural designs that reduce expensive external memory accesses. Igehy et al. [1998] introduced a prefetching texture cache architecture for texture mapping which used a fragment FIFO to hide texture memory access latency. Akenine-Möller and Ström [2003] proposed new hardware architecture for rasterizing textured triangles that was able to reduce the required memory bandwidth. The architecture includes a low-cost high-quality multisampling scheme, a texture minification/compression system to achieve trilinear mipmapping quality with 1/6 memory accesses on average, and a scanline-based culling scheme to reduce z-buffer reads. Tile-based rendering (also known as chunk rendering or bucket rendering) decomposes the screen into small regions called tiles and renders them independently. Tile-based rendering reduces the power dissipation of the frame buffer and z buffer accessing but introduces overheads of primitive sorting and redundant primitive processing of overlapped primitives. Cox and Bhandari [1997] and Chen et al. [1998] used a bounding box test method to test a primitive-to-tile overlap and to evaluate the impact

of overlap, whereas Antochi et al. [2004] evaluated several primitive sorting algorithms' computational complexity and the memory requirements for GPU designers.

Many studies have looked at the applicability of power gating on different components of the CPU. Powell et al. [2000] applied power gating, which is called Gated- V_{dd} , on a dynamic resizable instruction-cache (DRI i-cache). The DRI i-cache exploits the varying i-cache utilization within and across applications and chooses the required i-cache size accordingly. The supply voltage of unused cache cells could be gated to eliminate leakage. Kaxiras et al. [2001] observed that the cache lines are frequently used when first brought into the cache and then have a period of "dead time" before they are replaced. Therefore, they propose a time-based policy called cache decay, which turns off a cache line after seeing a pre-set number of inactive cycles since its last access. Flautner et al. [2002] periodically turned the cache lines into a low-supply-voltage drowsy mode while retaining the data in the cache. This scheme achieves significant cache leakage reduction with only negligible performance degradation. Juang et al. [2004] studied how to apply the decay method for leakage reduction on the branch prediction unit. Hu et al. [2004] analyzed the power gating potential on the execution units of out-of-order superscalar processors, based either on a simple time-out policy or on the branch-prediction-guided policy that utilizes the idle cycles after a branch misprediction to turn the execution units off. Youssef et al. [2006] proposed a predictive time-out method that predicts the length of the idle period dynamically, for the purpose of not entering the power-gated mode when encountering short idle intervals.

7. CONCLUSION AND FUTURE WORK

In this article, we demonstrated that there are power gating opportunities in GPUs by analyzing the utilization of various GPU modules. To make use of their available pieces of slack time, we propose three architectural-level power gating mechanisms that save leakage power at three different levels of granularity. These three mechanisms are orthogonal to one other because they control different parts of the GPU. They can be used either together or separately. First, we show how to use a simple history-based prediction approach to determine the shader resource requirements of the next frame and to shut down the unnecessary shader clusters. PSS eliminates up to 60% of the leakage consumption of shader clusters, with approximately 1% in performance degradation. To address the imbalance in the geometry and the fragment computations, we can shut off the fixed-function geometry units completely by delaying their render-configuration settings to eliminate up to 57% of the leakage of the fixed-function geometry units, with a performance degradation that is less than 0.5%. Finally, we observe a low utilization of the execution units in the non-shader pipeline stages because of the complex shader programs used in First-Person Shooter games. We find that a simple time-out power gating method can remove the leakage of the execution units in those pipeline stages by 83.3% on average and has little impact (0.4%) on the overall performance. For the execution units with shorter idle periods, request batching could further remove the leakage by 10%, at an expense of 0.7% in performance degradation.

We are currently developing the DVFS technique for the unified shader GPU architecture, and will study how to use DVFS and power gating jointly to achieve greater energy savings compared to adopting either one of these technologies alone. Because of the different execution characteristics for multiple rendering passes, there is also workload variation among the batches in a frame. Therefore, we plan to extend the proposed PSS by exploiting this opportunity for power gating shader clusters.

REFERENCES

- AKENINE-MOLLER, T. AND STROM, J. 2003. Graphics for the masses: a hardware rasterization architecture for mobile phones. In *Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'03)*. ACM, New York, NY, 801–808.
- ANTOCHI, I., JUURLINK, B., VASSILIADIS, S., AND LIUHA, P. 2004. Scene management models and overlap tests for tile-based rendering. In *Proceedings of the EUROMICRO Symposium on Digital System Design (DSD'04)*. IEEE Computer Society, 424–431.
- BEYOND3D. 2008. Ati rv635 chip details. <http://www.beyond3d.com/resources/chip/127>.
- BORKAR, S. 1999. Design challenges of technology scaling. *IEEE Micro* 19, 4, 23–29.
- BUTLER, H. 2010. Nvidia geforce gtx 480 1,536mb review. <http://www.bittech.net/hardware/2010/03/27/nvidia-geforce-gtx-480-1-5gb-review/10>.
- CEBENOYAN, C. 2004. Graphics pipeline performance. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, R. Fernando, Ed., Pearson Higher Education, Chapter 28, 473–486.
- CHANDRAKASAN, A., BOWHILL, W. J., AND FOX, F. 2000. *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE Press.
- CHEN, M., STOLL, G., IGEHY, H., PROUDFOOT, K., AND HANRAHAN, P. 1998. Simple models of the impact of overlap in bucket rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware (HWWS'98)*. ACM, New York, NY, 105–112.
- CLAYPOOL, K. AND CLAYPOOL, M. 2007. On frame rate and player performance in first person shooter games. *Springer Multimedia Syst. J.* 13, 1, 3–17.
- COX, M. AND BHANDARI, N. 1997. Architectural implications of hardware-accelerated bucket rendering on the pc. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*. ACM, New York, NY, 25–34.
- DEL BARRIO, V. M., GONZALEZ, C., ROCA, J., FERNANDEZ, A., AND R., E. 2006. Attila: a cycle-level execution-driven simulator for modern GPU architectures. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'06)*. IEEE Computer Society, 231–241.
- DUARTE, D., TSAI, Y.-F., VIJAYKRISHNAN, N., AND IRWIN, M. J. 2002. Evaluating run-time techniques for leakage power reduction. In *Proceedings of the Conference on Asia South Pacific Design Automation / VLSI Design (ASPDAC'02)*. IEEE Computer Society, 31–38.
- FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., AND MUDGE, T. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. IEEE Computer Society, 148–157.
- GOODHEAD, P. 2010. Matrix hd 5870 power consumption and thermals. <http://www.bittech.net/hardware/graphics/2010/07/15/asus-matrix-hd-5870-graphics-card-review/7>.
- GU, Y. AND CHAKRABORTY, S. 2008a. Control theory-based DVS for interactive 3d games. In *Proceedings of the 45th Annual Conference on Design Automation (DAC'08)*. ACM, New York, NY, 740–745.
- GU, Y. AND CHAKRABORTY, S. 2008b. A hybrid DVS scheme for interactive 3d games. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*. IEEE Computer Society, 3–12.
- GU, Y., CHAKRABORTY, S., AND OOI, W. T. 2006. Games are up for DVFs. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC'06)*. ACM, New York, NY, 598–603.
- HONG, S. AND KIM, H. 2010. An integrated gpu power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY, 280–289.
- HU, Z., BUYUKTOSUNOGLU, A., SRINIVASAN, V., ZYUBAN, V., JACOBSON, H., AND BOSE, P. 2004. Microarchitectural techniques for power gating of execution units. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'04)*. ACM, New York, NY, 32–37.
- IGEHY, H., ELDRIDGE, M., AND PROUDFOOT, K. 1998. Prefetching in a texture cache architecture. In *HWWS'98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*. ACM, New York, NY, 133–ff.
- ITRS. 2006. International technology roadmap for semiconductors.
- IYER, A. 2006. Demystify power gating and stop leakage cold. <http://www.eetimes.com/design/automotive-design/4012054/Demystify-power-gating-and-stop-leakage-cold>.
- JUANG, P., SKADRON, K., MARTONOSI, M., HU, Z., CLARK, D. W., DIODATO, P. W., AND KAXIRAS, S. 2004. Implementing branch-predictor decay using quasi-static memory cells. *ACM Trans. Archit. Code Optim.* 1, 2, 180–219.
- KALLA, P., HU, X. S., AND HENKEL, J. 2006. Distance-based recent use (dru): an enhancement to instruction cache replacement policies for transition energy reduction. *IEEE Trans. VLSI Syst.* 14, 1, 69–80.

- KANTER, D. 2008. Nvidia's gt200: Inside a parallel processor. <http://www.realworldtech.com/page.cfm?ArticleID=RWT090808195242>.
- KAO, J. AND CHANDRAKASAN, A. 2000. Dual-threshold voltage techniques for low-power digital circuits. *IEEE J. Solid-State Circ.* 35, 7, 1009–1018.
- KAXIRAS, S., HU, Z., AND MARTONOSI, M. 2001. Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA'01)*. ACM, New York, NY, 240–251.
- KHRONOS. 2010. Opencl overview. <http://www.khronos.org/opencl/>.
- LINDHOLM, E. AND OBERMAN, S. 2007. Nvidia geforce 8800 gpu. In *Hot Chips 19: A Symposium on High Performance Chips*.
- LU, Y.-H. AND MICHELI, G. D. 2001. Comparing system-level power management policies. *IEEE Des. Test* 18, 2, 10–19.
- MANTOR, M. 2007. Radeon r600, a 2nd generation unified shader architecture. In *Hot Chips 19: A Symposium on High Performance Chips*.
- MOCHOCKI, B., LAHIRI, K. AND CADAMBI, S. 2006a. Power analysis of mobile 3d graphics. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'06)*. European Design and Automation Association, Belgium, 502–507.
- MOCHOCKI, B., LAHIRI, K., CADAMBI, S., AND HU, X. S. 2006b. Signature-based workload estimation for mobile 3d graphics. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC'06)*. ACM, New York, NY, 592–597.
- NAM, B.-G., LEE, J., KIM, K., LEE, S. J., AND YOO, H.-J. 2007. A low-power handheld GPU using logarithmic arithmetic and triple DVFS power domains. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics hardware (GH'07)*. 73–80.
- NVIDIA. 2010. Cuda zone. <http://www.nvidia.com/object/cuda.home.new.html>.
- POWELL, M., YANG, S.-H., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2000. Gated-vdd: A circuit technique to reduce leakage in deep-sub micron cache memories. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'00)*. ACM, New York, NY, 90–95.
- SANDER, P. V., NEHAB, D., AND BARCZAK, J. 2007. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph.* 26, 3, 89.
- SHEAFFER, J. W., LUEBKE, D., AND SKADRON, K. 2004. A flexible simulation framework for graphics architectures. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWs'04)*. ACM, New York, NY, 85–94.
- SHEAFFER, J. W., SKADRON, K., AND LUEBKE, D. P. 2005. Studying thermal management for graphics processor architectures. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05)*. IEEE Computer Society, 54–65.
- SOHN, J.-H., WOO, J.-H., LEE, M.-W., KIM, H.-J., WOO, R., AND YOO, H.-J. 2006. A 155-mw 50-m vertices/s graphics processor with fixed-point programmable vertex shader for mobile applications. *IEEE J. Solid-State Circ.* 41, 5, 1081–1091.
- SPILE, C., VÖTTER, R., AND SAUTER, M. 2010. Geforce gtx 480 and gtx 470 reviewed: Fermi performance benchmarks. <http://www.pcgameshardware.com/laid/743498/Geforce-GTX-480-and-GTX-470-reviewed-Fermi-performance-benchmarks/Reviews/>.
- VOICU, A. 2008. Ati rv770 - architecture overview. <http://www.rage3d.com/reviews/video/atirv770/architecture/>.
- WANG, P.-H., CHEN, Y.-M., YANG, C.-L., AND CHENG, Y.-J. 2009. A predictive shutdown technique for gpu shader processors. *IEEE Comput. Archit. Lett.* 8, 9–12.
- YOUSSEF, A., ANIS, M., AND ELMASRY, M. 2006. Dynamic standby prediction for leakage tolerant microprocessor functional units. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 371–384.

Received January 2009; revised January 2010; accepted November 2010