35.2

Signature-Based Workload Estimation for Mobile 3D Graphics

Bren C. Mochocki[†‡] Kanishka Lahiri[‡] Srihari Cadambi[‡] X. Sharon Hu[†]

[†]Dept. of CSE, University of Notre Dame, IN (bmochock,shu@cse.nd.edu)
[‡]NEC Laboratories America, Princeton, NJ (klahiri,cadambi@nec-labs.com)

ABSTRACT

Until recently, most 3D graphics applications had been regarded as too computationally intensive for devices other than desktop computers and gaming consoles. This notion is rapidly changing due to improving screen resolutions and computing capabilities of mass-market handheld devices such as cellular phones and PDAs. As the mobile 3D gaming industry is poised to expand, significant innovations are required to provide users with high-quality 3D experience under limited processing, memory and energy budgets that are characteristic of the mobile domain.

Energy saving schemes such as Dynamic Voltage and Frequency Scaling (DVFS), as well as system-level power and performance optimization methods for mobile devices require accurate and fast workload prediction. In this paper, we address the problem of workload prediction for mobile 3D graphics. We propose and describe a signature-based estimation technique for predicting 3D graphics workloads. By analyzing a gaming benchmark, we show that monitoring specific parameters of the 3D pipeline provides better prediction accuracy over conventional approaches. We describe how signatures capture such parameters concisely to make accurate workload predictions. Signature-based prediction is computationally efficient because first, signatures are compact, and second, they do not require elaborate model evaluations. Thus, they are amenable to efficient, real-time prediction. A fundamental difference between signatures and standard history-based predictors is that signatures capture previous outcomes as well as the cause that led to the outcome, and use both to predict future outcomes. We illustrate the utility of signature-based workload estimation technique by using it as a basis for DVFS in 3D graphics pipelines.

Categories and Subject Descriptors: C.3 [Special-Purpose and Application-Based Systems]

General Terms: Algorithms, Design, Performance

Keywords: 3D Graphics, embedded systems, workload estimation, dynamic voltage scaling.

1. INTRODUCTION

3D graphics applications, which have primarily been developed for desktop computers and dedicated gaming consoles, are being increasingly targeted to mobile devices such as cellular phones and PDAs. Mobile devices lend themselves well to 3D applications such as GPS-backed maps, networked games, interactive chats and animated messages. However, they have far fewer computing resources than their desktop counterparts, and short battery lifetimes.

Limited computing resources on mobile platforms have two implications that are of relevance to this paper. First, mobile devices that execute 3D applications meant for graphics PCs will have to dynamically adapt application parameters to reduce complexity and meet deadlines. For example, networked games often distribute the same 3D content to both graphics PC users and mobile users, requiring the mobile devices to reduce the level of detail (the detail with which each object is drawn) in order to provide acceptable frame rates [1]. Selecting suitable parameter values (such as the level of detail) is facilitated by a good workload prediction scheme. The second implication is that many mobile devices have either a single processor performing multiple tasks, or a few generic processors coupled with application-specific cores. These situations require resource allocation, sharing and job scheduling, all of which can benefit from a fast and accurate workload prediction scheme.

Another domain where workload prediction is useful is low-power computing. Battery technology improvements are not expected to keep pace with the expected increase in energy consumption of mobile 3D graphics [2], [3]. [4] reports that 3D graphics application workloads can vary across frames by over an order of magnitude. For example, games frequently have scenes with considerable detail followed by relatively empty scenes. For such a varying workload, dynamic voltage and frequency scaling (DVFS) is a natural choice as an energy-reducing scheme [3]. A key requirement for DVFS is a fast and accurate workload prediction model.

Analytical schemes for predicting mobile 3D graphics workloads exist [1]. However, analytical prediction models are computationally expensive. Evaluating them online (i.e., during frame execution) could mean missing frame deadlines, an effect that produces annoying visual distortions. Therefore, in practice, such models often compromise computation complexity for accuracy, resulting in frequent over- and under-estimations of the workload.

In this paper, we propose a specialized online prediction scheme for mobile 3D graphics that addresses the above drawbacks. Instead of estimating the workload using analytical models, we construct a compact signature for each frame. Each signature, comprising a subset of 3D graphics parameters, has a corresponding workload. Both signature and workload are stored in a table. For every future frame, we generate a signature and predict the workload using its closest matching signature from the table. When the prediction deviates from the actual workload, the table is updated using real measurements. As we show by experimental evaluation, the simplicity of generating and finding a matching signature eliminates the overhead associated with evaluating complex 3D graphics estimation models, while still making good predictions. Refining the predictions using real measurements makes the scheme accurate even for hard-to-predict, highly dynamic workloads. Note that measurement-driven, dynamic refinements cannot be applied to analytical estimation models, i.e., unlike signatures, it is not possible to refine complex equations dynamically. As an application, we use signatures to predict workloads for DVFS: with practically no hardware requirements except DVFS capability, the scheme can be easily integrated into existing 3D graphics systems.

It may be observed that standard history-based predictors use workloads of past frames to predict the workload of future frames. Thus, such predictors predict future outcomes using past outcomes only. In

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

contrast, the signature table captures both the outcome *as well as the cause that led to the outcome*. The outcome is the workload, and the cause is the signature which is a snapshot of the frame's properties. Both cause and outcome are used to predict future outcomes. This is more powerful than history-based approaches, and particularly suitable for specific applications (such as 3D graphics), where domain-specific knowledge can be used to clearly identify the cause.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we provide a brief background of 3D graphics pipelines. In Section 4, we present a motivational example, before describing our signature-based prediction scheme in detail in Section 5. We present experimental results, including energy saved by using signature-based prediction for DVFS, in Section 6, and conclude in Section 7.

2. RELATED WORK

Workload prediction has been studied for general-purpose processors running arbitrary workloads ([5], [6]) as well as real-time systems [7], motivated either by the appeal of power savings via DVFS [6] or the effective use of dynamically reconfigurable hardware [8]. The signature-based scheme described in [8], like most prior work on workload prediction, is based on run-time profiling of the usage of standard hardware resources such as the CPU or cache. Conventional workload prediction techniques do not exploit application-specific information such as the parameters of a 3D pipeline. As a result, many on-line prediction policies do not perform well for specific applications [9]. This has led to workload-specific techniques in which additional information about the executing application is used to better anticipate workload variations [10].

Prior work that addresses workload estimation for graphics pipelines includes analytical performance models that compute workload requirements from pipeline parameters [1], [11]. Such analytical models can require considerable effort in model development and calibration, and in addition, can incur significant run-time computation overhead. In contrast, our signature-based approach does not require a priori model calibration, or run-time computation. Other efforts that quantitatively analyze 3D graphics workloads are geared towards architectural optimization [12] rather than on-line workload prediction. Finally, performance and power modeling techniques for GPUs have been proposed [13]. More widespread availability of such models will spur research on graphics pipelines based on quantitative analysis of hardware performance and power consumption. Our experimental framework is conceptually similar to the one described in [13] in that both are based on analyzing traces of calls to a graphics library.

3. 3D GRAPHICS PIPELINE

In typical graphics processors all surfaces are fundamentally represented with triangular patches [14]. Each triangle is drawn in a series of three basic steps, which can be implemented as three stages in a pipeline architecture:

The **geometry** stage applies geometric transforms to each triangle, and computes its perspective projection onto the screen. It also *culls* (removes) triangles that will not be seen, *clips* triangles that are outside the frame, and computes lighting information for each vertex.

The **triangle setup** stage determines the set of pixels contained within the projection of each triangle. Along each triangle edge, this steps interpolates depth values, shading values, and texture indices.

The **rendering** stage computes the actual color for each pixel. The major sub-tasks in rendering are hidden-surface elimination (*z-buffering*), final interpolation of shading values, texture mapping and adding transparency information for translucent objects (*alpha blending*).

Key factors that effect the perceived quality of 3D graphics are resolution, frame rate, level of detail, lighting and textures. *Resolution* represents either the total number of pixels on the screen or the number of pixels per inch of screen. Note that handheld devices are typically held close to the eye, and consequently demand a higher number of pixels per inch than desktop monitors. *Frame rate* is the rate with which the scene is redrawn. *Level of detail* is the sampling used to represent curved shapes with triangular patches. This determines both how smooth the shapes' silhouettes appear and whether the individual triangles are visible on the surface due to shading. Possible *lighting models* include spot lighting (illumination only in specified areas of the scene), point lighting (illumination from nearby light sources) and parallel lighting (illumination from distant light sources). The *texture model* indicates how textures, usually in the form of 2D images, are applied to surfaces. There are several different methods of applying textures to surfaces, with different tradeoffs between computation and visual quality.

For our purposes, it is important to note that the computational cost of each stage in the pipeline is scene-dependent. The geometry stage visits each vertex of every triangle, possibly transforming it to a new position. Further, it also clips and culls triangles. Vertices that require lighting involve significantly more work. The triangle setup stage is usually implemented with the "scanline" algorithm [14]. Finally, the rendering stage processes every pixel within each triangle. However, it's computational requirement depends on the number of texture maps, hidden surfaces, transparent objects, and in general, the scene complexity. In summary, predicting the computational cost of a graphics pipeline is complex and requires non-trivial models.

4. MOTIVATIONAL EXAMPLE

In this section, we examine the dynamic variability of 3D graphics workloads using a representative benchmark, and discuss the implications on techniques for on-line workload prediction. For this study, we consider a visualization of a 3D racing game, targeted towards a 176x220 16-bit (65536) color display. The benchmark features twelve logical objects, including a dark background (a starlit night sky), three racers, three race tracks, and five billboards. Texturing is used to depict starlight. Since the object on which the texture is applied is larger than the texture itself, a simple filtering scheme is used. Point and parallel lights illuminate the scene, with varying direction and intensity. Alpha blending controls object transparency, which varies with time. For example, the transparency of the jetstream varies inversely with its speed. A version of the benchmark was obtained from [15], and was optimized for an open-source implementation of OpenGL-ES [16], the standard 3D graphics API for embedded platforms. We profiled the execution times of the application, geometry, setup and rendering stages with instruction-level simulation, using an experimental infrastructure described in Section 6. Figure 1(a) is an illustrative snapshot of the workload created by this benchmark in each of these stages. The figure plots the variation in the number of cycles spent in executing different graphics tasks over a sequence of 200 frames. It may be seen that frame-to-frame processing requirements vary significantly. For reasons described in Section 1, predicting such workload variations at run-time can help significantly optimize processing of graphics applications on a mobile device. We now analyze selected segments of the trace in greater detail, to better understand some of the factors that influence the workload and point out implications on the design of workload predictors.

Frames 1-50: During this segment, the camera is looking at the entire scene (Frame A), including all the logical objects mentioned earlier[1]. The geometry computation requirement is high due to the large number of objects, and the large number of vertices that are illuminated. However, the small size of the racers (located at the center of the screen, though some of them are culled by the billboard), leads to a low rendering workload.

[1]Note that, for the sake of print quality, the images in the paper have a white background instead of the night sky.

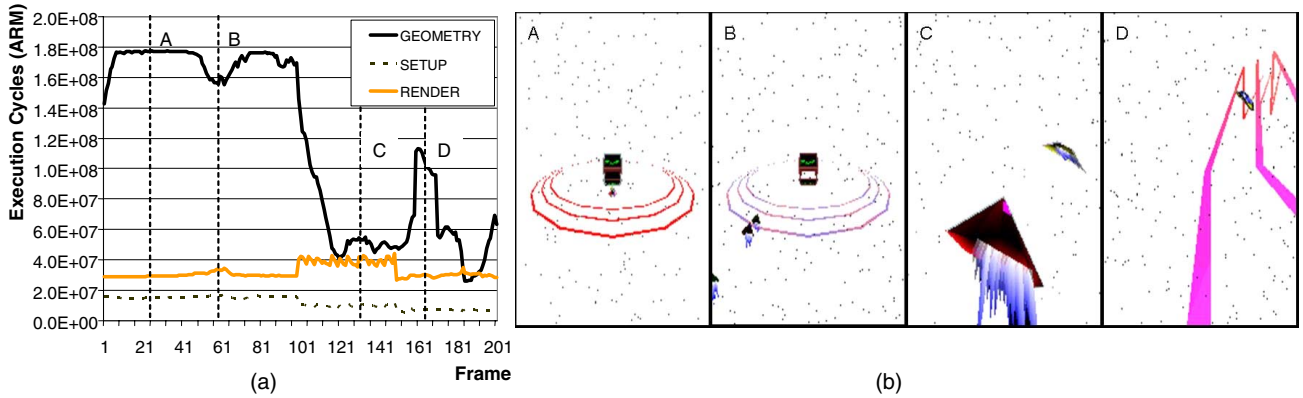

Figure 1: (a) 3D graphics workload variation for an OpenGL-ES based racing visualization benchmark: breakdown by pipeline stage; (b) Representative frames from illustrative segments of the benchmark.

Frames 50-70: During this interval, one of the racers flies off the screen (Frame B), and is clipped since it goes outside the angle of view. This leads to fewer vertex lighting calculations, and a dip in the geometry workload.

Frames 100-150: The viewing angle changes to one that is directly behind one of the racers (Frame C). This leads to a reduced geometry workload, mainly due to a drastic reduction in the number of in-view objects: a large portion of the race track, the billboards, and one of the racers are clipped. However, we observe that the rendering workload rises sharply. This is attributed to the increased complexity of rendering the racers, which are viewed in close-up, and the jetstream, which changes rapidly.

Frames 150-200: The camera is now positioned within one of the racers (Frame D), causing the track to appear in greater detail, along with intermittent appearance of another racer. Both lead to significant fluctuation in the geometry workload.

The above example illustrates some of the factors that influence workload characteristics in graphics pipelines. While we were able to intuitively explain the workload variation using high-level knowledge of the set of objects and their movements, on-line workload prediction techniques will not have this luxury. This is because in the graphics pipeline, the granularity of processed items is at the level of triangles, or scan lines, rather than racers, or billboards. A crucial challenge therefore, is to perform accurate and efficient workload prediction based on information that can be easily extracted from the graphics pipeline.

The above study motivates analytical models that compute workload requirements based on specific observable parameters in the graphics library [1]. Graphics libraries contain a large number of parameters that constitute the "state" of the graphics pipeline. Model parameters could include such variables and data structures related to visual effects, such as lighting, shading, texturing, fog and shadows, in addition to parameters related to the primitives being processed, such as triangles and scan lines. Challenges with this approach lie in the difficulty of choosing a form for the analytical model (given the large number of parameters), and subsequently calibrating it using "representative" workloads. Further, the state of the pipeline could vary on a *per triangle* basis, making it necessary to perform fine-grained, intra-frame monitoring of the pipeline. In such a scenario, on-line evaluation of these analytical models may prove computationally expensive, resulting in frames missing their deadlines, producing undesirable visual artifacts. Finally, analytical models are static and cannot adapt dynamically to different applications, or different scenes within an application.

In contrast, we propose the use of a far simpler, *signature-based* approach, which uses a limited amount of information gleaned from the graphics pipeline to make workload predictions, but does not require a priori model calibration or design.

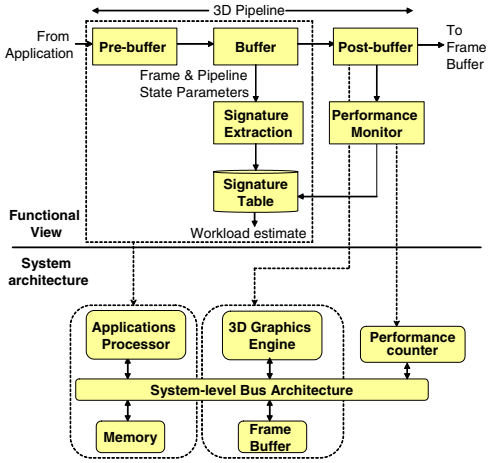

Figure 2: Signature-based workload prediction scheme for mobile 3D graphics, illustrating (a) a functional view of proposed scheme, and (b) a candidate mapping to a system architecture.

5. SIGNATURE-BASED TECHNIQUE

In this section, we describe our signature-based prediction strategy for mobile 3D graphics. Signature-based prediction is computationally simple, can dynamically adapt to different applications and workloads and uses both a frame's properties as well as historical performance (cause and outcome) in order to make a prediction. We first describe the different steps in signature-based workload prediction, and then explain how this strategy can be applied to a system architecture consisting of components typical of a mobile platform with 3D graphics processing capabilities.

5.1 Signature-based Prediction

The top portion of Figure 2 illustrates a functional view of the proposed signature-based workload prediction scheme. Relevant information that is readily available from 3D graphics pipeline through API data structures (e.g., the OpenGL-ES state), is used to construct a concise signature every frame. The very first time a signature is encountered, it is inserted into a *signature-table*, along with a conservative workload prediction. When the frame completes, an on-line measurement using a performance monitor is used to update the signature table with the actual observed workload. The signature table maps signatures to their predicted workloads. If a signature matches one that has been encountered in the past, the stored workload is used as the new prediction.

Signature-based prediction has two key advantages over analytical prediction models. First, signatures are simple to construct — there is little computation necessary; therefore, the method is suitable for ap-

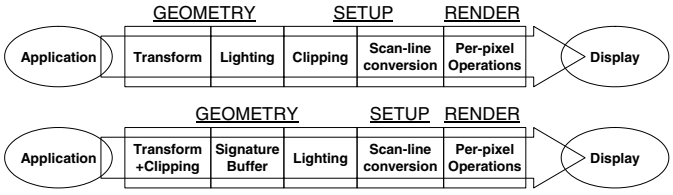

Figure 3: Original 3D graphics pipeline (top) and modified pipeline for signature-based prediction (bottom).

plications (such as 3D graphics) with stringent real-time constraints. Second, signatures allow measurement-driven refinement and therefore can adapt to different 3D graphics applications. This is not possible for analytical models since equations cannot be refined in real time. Thus, during the course of the application, the same signature can predict different workloads based on how the refinement proceeds.

5.1.1 Pipeline Modifications

The typical 3D graphics pipeline is shown in Figure 3. For every frame, the application stage creates triangles and sends them to the geometry stage. In order to construct a complete and meaningful signature, we need information about the entire frame (such as the total number of triangles in the frame). Hence we insert a buffer to collect all the triangles, their vertices and associated information after the application stage, prior to signature generation. This "signature buffer" inserts a tolerable delay of one frame.

The pipeline is now modified to facilitate prediction. We move vertex transformation, triangle culling and clipping (see Section 3) to before the signature buffer, leaving only lighting in the geometry stage. Thus, the buffer contains an image of the frame after all vertices have been transformed, and optimizations such as triangle clipping and culling have been applied. The transformed pipeline is shown in Figure 2, with a more detailed view in Figure 3. It broadly consists of the *pre-buffer portion*, the signature buffer and the *post-buffer portion*. Three points are note-worthy regarding the transformed pipeline. First, it is functionally equivalent to the original pipeline. Second, such a transformation can be made easily by graphics application developers (using say OpenGL-ES) without hardware changes. Third, we demonstrate that the running time of the pre-buffer portion is not only small, but also more deterministic and predictable than the running-time of the post-buffer portion. Thus, the signature-based workload predictor has to predict the workload of the post-buffer portion, which constitutes the bulk of the 3D graphics pipeline.

5.1.2 Signature Generation and Matching

3D graphics parameters required for a compact, yet meaningful and useful signature are readily available for each frame from OpenGL-ES data structures. We use the following parameters obtained from the triangles in the signature buffer to construct signatures. These specific parameters are straightforward, and are easy to extract and compute for each frame: (1) average triangle area, (2) triangle count, (3) average triangle height and (4) vertex count. The signature is constructed using actual values of these four parameters, concatenated into a string. We experimentally show that these are sufficient for even complex 3D graphics applications.

3D graphics applications frequently have frames with only minor differences and very similar workloads. A good predictor should recognize if the current frame is "similar" to, though not exactly the same as, a past frame, and predict the workload of that past frame. Therefore, for every generated signature, we find the *best matching signature* from the signature table. For a generated signature S, the best matching signature from the signature table is the one with the smallest *distance metric*. Specifically, for a generated signature S comprising parameters $s_1, s_2...s_d$ and a signature T in the signature table comprising parameters $t_1, t_2...t_d$, the distance metric $D(S,T)$ is:

$$D(S,T) = \sum_{i=1}^{i=d} \frac{|s_i - t_i|}{s_i} \quad (1)$$

The distance metric shown in Equation 1 is linear and normalized to the generated signature. The best matching signature is obtained from the signature table using a linear search. This is not a scalable solution; faster matching algorithms are available in the literature, but are not the focus of this paper. Any of these faster algorithms may be employed with our overall signature-based prediction scheme. We use the present study to motivate the evaluation of more complex distance metrics as well as faster matching algorithms and data structures in future work.

5.1.3 Prediction Algorithm

When the triangles of a frame are in the signature buffer, a signature S is generated using the OpenGL-ES parameters listed in Section 5.1.2. The closest matching signature to S is then found from the signature table. The corresponding workload is read from the table is the prediction. Note that if the signature table is empty, a conservative estimate for the workload is used.

We augment the 3D graphics library to tag the first triangle and the last scanline in every frame. Using these, hardware timers note the start and end times of every frame. These execution times represent workloads. The workload of a frame i (with signature S_i) that has just completed is measured using the difference between the start and end times. This is the actual, measured workload used to refine the workload of signature S_i in the signature table.

Note that hardware monitors need only record the start time, finish time and the frame number. While this measurement can be performed in real time, the operating system (OS) and other overheads often mean that the table cannot be updated immediately. We propose using "lazy updates" with which the OS can batch updates to the signature table and perform them when ready. This comes at the expense of a few inaccurate predictions.

5.2 System Architecture

Now we describe the system architecture in which signature-based prediction may be used. In most 3D graphics processing systems, a major part of the graphics pipeline is mapped to a dedicated, programmable graphics processor. The graphics processor is connected to the host or applications processor through a system-level bus architecture. The graphics processor may either be a discrete component [17], or may be integrated with the applications processor as a core on an on-chip bus [18]. Figure 2 illustrates such an architecture, and how the proposed signature-based workload prediction scheme is mapped to it. The architecture includes a hardware performance counter. Performance counters vary in complexity, depending on the nature of the information profiled (e.g., cache miss rates, bus utilization, *etc*). In our case, the counter measures elapsed time between the completion of specific events, namely, the detection of tagged first triangle and the tagged last scan line for each frame. Hence, the only hardware requirement is a standard timer, which is commonly available in most modern SoCs.

In the illustration, the pre-buffer portions of the graphics pipeline are mapped to the applications processor, while all the remaining tasks are mapped to software running on the graphics processor. The performance monitor task uses standard drivers to start, stop, and obtain measurements using memory-mapped timer registers. In our experiments we observed that signature processing in software is sufficient for frame-level workload prediction.

5.3 Application to DVFS

DVFS is an important energy-reducing technique requiring a fast and accurate workload prediction scheme. Computationally expensive

analytical models that frequently over- and under-estimate the workload may cause excessive energy dissipation and missed frame deadlines. Signature-based prediction can easily be applied to application-specific processors with DVFS to overcome these problems. Following signature analysis, DVFS is applied to the post-buffer portion only.

After signature generation, when a frame i is dispatched from the prediction buffer, several frames may be in flight within the graphics pipeline. Some frames in the pipeline may have large workloads, while others may have small workloads. In order to ensure that no frame misses its deadline, the voltage issued to the pipeline must be the maximum of the voltages required by all frames currently in flight.

We also note that because different frames in flight may have encountered different clock frequencies during their execution, clock cycles rather than raw execution time is the correct measure of workload. Therefore, hardware timers, which record the raw execution time, must work with the performance unit that is aware of the clock frequencies, thereby allowing the translation to clock cycles.

6. EXPERIMENTAL EVALUATION

We modeled the system architecture outlined in the previous section using an ARM processor for the pre-buffer portion, and an ARM processor for the post-buffer stage of the graphics pipeline. The different pipeline stages communicate with each other through shared buffers mapped to local memories. DMA engines transfer data between stages. For performance and power analysis, we use a combination of trace-driven simulation and instruction-level profiling [3]. The starting point of our framework is a 3D software library [16] that implements OpenGL/ES [19], a standard API for implementing 3D applications on embedded platforms. We modified this library to generate a trace of triggers indicating the start and completion points of each pipeline stage, and the triangles being processed. The applications and the modified library were cross-compiled and linked to the ARM architecture using the `arm-gcc` toolchain [20]. To drive the subsequent analysis, traces of executed instructions, triggers, and triangles were collected from fast, instruction-level simulation of the ARM architecture [21]. The resulting traces were provided to a simulator that we developed to analyze performance and energy consumption of the different 3D pipeline stages, using an energy model based on [22]. The graphics processor (modeled by an ARM) can be operated at 11 different voltage and frequency levels, with an associated overhead of 150μs each time the operating point is changed. Different voltage/frequency scaling policies and their impact on the 3D pipeline can easily be evaluated using this framework.

In all our experiments, we started with an empty signature table. We used the distance metric described in Section 5.1.2 to select the best matching signature from the table. The workload corresponding to the best matching signature was used as the prediction. When the actual measured workload differed from the prediction, and the minimum distance was non-zero, a new signature was introduced into the signature table. If the measured workload differed from the prediction, and the minimum distance was zero, the prediction corresponding to the matching signature in the table was simply updated.

Two representative 3D application benchmarks we used were the CUBE benchmark from [23], and the RACER benchmark from [15]. The RACER benchmark is described in Section 4. CUBE contains a 10x10x10 cube centered at the origin, formed using triangle fans (24 vertices and 12 triangles). It makes one complete revolution every 45 frames. A light source spins around the top of the cube in the opposite direction with the same period. The camera is situated on the +Z axis (0, 30, 30) and is looking at the origin. A perspective view is used to project the cube.

6.1 Impact of Pipeline Modification

Recall that we reordered the classical 3D graphics pipeline by moving transformation and clipping to before the prediction buffer, leaving only lighting in the geometry stage. Figure 4 shows the workloads for each of these portions. The pre-buffer portion workload comprising the application, state, transformation and clipping is shown on the left, while the post-buffer portion comprising lighting, setup and rendering is shown on the right. For the CUBE benchmark, we observe that the pre-buffer portion is less than 7% of the total workload, with a variation (indicated by the error bars) of less than 0.1%. For the more complex RACER benchmark, the pre-buffer portion is around 23% of the total workload, with less than a 6% variation. In contrast, the variation in the lighting, setup and rendering, which constitute three-quarters of the execution time, is 35%, 13% and 17% respectively. Thus, the post-buffer portion is computationally more significant, with considerably more variation. The pre-buffer portion, on the other hand, is more predictable and computationally smaller, justifying our reorganization of the graphics pipeline.

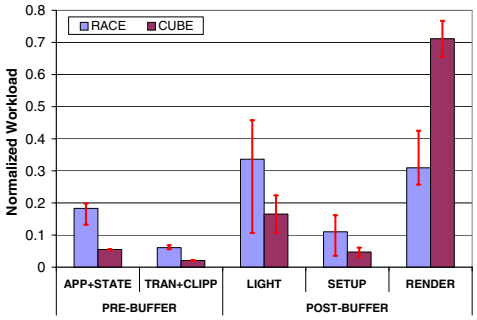

Figure 4: Workloads of the pre- and post-buffer portions.

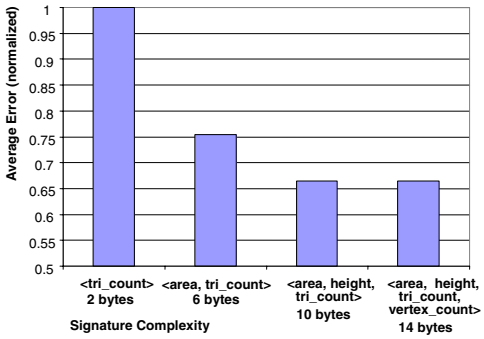

Figure 5: Prediction accuracies of signatures of various lengths.

6.2 Signature Complexity vs Accuracy

A major advantage of signature-based prediction is the compactness of the signature, leading to easy signature generation. We experimentally measured prediction errors for signatures of varying lengths comprising different 3D graphics parameters. Figure 5 shows these errors (averaged difference between predicted and actual workloads) normalized to the errors sustained by a signature that tracks only the total triangle area for the CUBE benchmark. We note that the error rate drops by a factor or 3 when the signature comprises just two parameters, and further when we use 3 and 4 parameters. Owing to dynamic refinement over several frames, signatures larger than 4 parameters had the same error rates indicating diminishing returns with increasing signature complexity. Thus, we obtain good prediction accuracy with relatively small signatures.

We measured the computational overhead of constructing signatures. The most expensive operation is computing the area of a triangle, which was found to be less than 1% of the time taken to perform geometry and lighting calculations for the triangle. Hence, taken together with the rendering (post-buffer) effort, signature computation represents negligible overhead. In addition, the space overhead of storing and managing signatures was found to be manageable. As

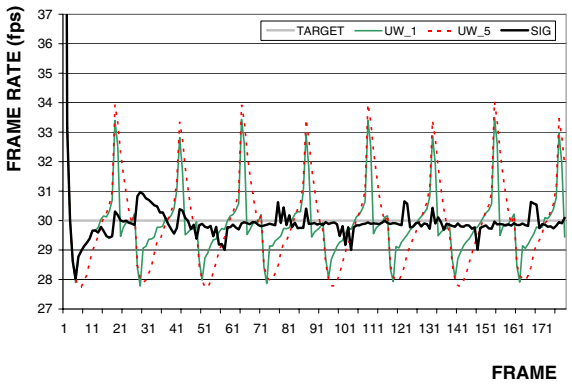

Figure 6: Prediction accuracy for the CUBE benchmark.

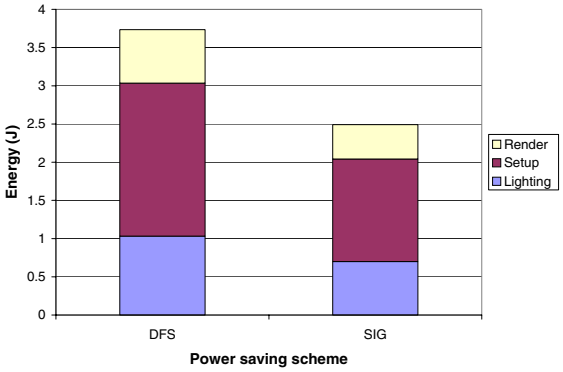

Figure 7: Energy savings with signature-based prediction for DVFS.

illustrated in Figure 5, the most complex signature is only 14 bytes. For the CUBE benchmark, a total of 50 unique signatures were generated, resulting in a signature table of 500 bytes, which can easily be accommodated within on-chip memory constraints.

6.3 Prediction Accuracy

Here we compare the accuracy of signature-based prediction with two history-based predictors, *Uniform Window-based Predictor 1 (UW_1)* and *Uniform Window-based Predictor 5 (UW_5)*. UW_1 predicts the workload of a frame to be exactly the same as the previous frame, while UW_5 predicts the workload of the current frame as the average of the workloads of 5 previous frames. Note that the table could be pre-populated by analyzing the application beforehand. For the CUBE benchmark, over almost 200 frames, we set the target frame rate to 30 fps and observed that the both UW_1 and UW_5 over-estimated by as much as 13% and under-estimated by around 7%, while the signature-based scheme did not stray by more than 3% in either direction. We used a signature consisting of the above 4 parameters for this experiment.

6.4 Application to DVFS

As an application, we used our signature-based predictor to estimate workloads for a DVFS strategy for mobile 3D graphics. We implemented a simulator to measure DVFS energy savings. As described in Section 6, the post-buffer portion of the 3D pipeline is mapped to a GPU and the pre-buffer portion to the generic applications processor. In our setup, both are modeled by an ARM processor, with DVFS affecting only the ARM representing the GPU. For the complex RACER gaming benchmark, we observed an overall energy reduction of 33% via voltage scaling using the signature based scheme compared to frequency scaling alone (Figure 7).

7. CONCLUSIONS

In this paper, we described a novel signature-based workload prediction scheme for mobile 3D graphics. Accurate and fast workload prediction finds application in dynamic voltage and frequency scaling (DVFS), dynamic 3D parameter adaptation and resource sharing and task allocation for mobile devices.

Signature-based prediction for mobile 3D graphics uses a concise signature constructed from readily available 3D graphics parameters, and uses the signature to predict the workload on a frame-by-frame basis. All generated signatures are stored in a signature table. When the prediction deviates from the actual workload, a device measures the actual workload and updates the signature table. This measurement-driven refinement provides signatures the ability to predict even dynamic, highly-variable workloads. Using representative 3D benchmarks, including a complex game, we demonstrated that compact signatures that track just a few 3D graphics parameters are sufficient for accurate prediction.

Our work also motivates further research in this direction. Signature similarity can be evaluated using different distance metrics, while signature matching can be performed with more efficient data structures.

8. REFERENCES

[1] N. Tack, F. Moran, G. Lafruit, and R. Lauwereins, "3D Graphics Rendering Time Modeling and Control for Mobile Terminals," in *Proceedings of Int. Conf. on 3D Web technology*, pp. 109–117, 2004.

[2] "International Technology Roadmap for Semiconductors (ITRS 2004)." `http://www.itrs.net/Common/2004Update/2004_000_ORTC.pdf`, 2004.

[3] B. Mochocki, K. Lahiri, and S. Cadambi, "Power Analysis of Mobile 3D Graphics," in *Proc. Design Automation & Test Europe (DATE) Conf.*, pp. 502–507, Mar. 2006.

[4] G. Lafruit, L. Nachtergaele, K. Denolf, and J. Bormans, "3D Computational Graceful Degradation," in *Proceedings of ISCAS - Workshop and Exhibition on MPEG-4*, vol. III, pp. 547–550, 2000.

[5] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," in *Proc. Symp. Foundations of Computer Science*, pp. 374–382, Oct. 1995.

[6] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 76–81, Aug. 1998.

[7] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," in *Proc. Symp. Operating Systems Principles*, pp. 89–102, Oct. 2001.

[8] A. S. Dhodapkar and J. E. Smith, "Managing Multi-Configuration Hardware via Dynamic Working Set Analysis," in *Proc. Int. Symp. Computer Architecture*, pp. 233–244, May 2002.

[9] D. Grunwald, P. Levis, K. Farkas, C. B. Morrey, and M. Neufeld, "Policies for Dynamic Clock Scheduling," in *Proc. Symp. Operating Systems Design and Implementation*, pp. 73–86, Oct. 2000.

[10] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," in *Proc. Symp. Operating Systems Principles*, pp. 149–163, Oct. 2003.

[11] M. Wimmer and P. Wonka, "Rendering Time Estimation for Real-Time Rendering," in *Eurographics Symposium on Rendering 2003*, pp. 118–128, 2003.

[12] T. Mitra and T. Z. Chiueh, "Dynamic 3D Graphics Workload Characterization and the Architectural Implications," in *Proc. Intl. Symp. on Microarchitecture (MICRO-32)*, pp. 62–71, Nov. 1999.

[13] J. W. Sheaffer and D. P. Luebke and K. Skadron, "A Flexible Simulation Framework for Graphics Architectures," in *Proc. of the Eurographics Conference on Graphics Hardware*, pp. 85–94, Aug. 2004.

[14] A. H. Watt, *3D Computer Graphics*. Addison-Wesley, 2000.

[15] B. Mochocki, "Desktop2Handheld: the Porting of an OpenGL Application to OpenGL/ES." `http://www.nd.edu/~bmochock/race/ComputerGraphicsFinal_Mochocki.pdf`.

[16] Hans-Martin Will, "A 3-D Rendering Library for Mobile Devices." `http://ogl-es.sourceforge.net/`, 2004.

[17] "ATI Imageon 2300 Media Co-Processor." `http://www.ati.com/products/imageon2300/`.

[18] "PowerVR MBX Product Sheet." `http://www.powervr.com`.

[19] Khronos Group, "OpenGL ES Overview." `http://www.khronos.org/opengles`, 2005.

[20] B. Gatliff `http://www.billgatliff.com`, 2005.

[21] W. Qin, "Simit-ARM: Very Fast Functional and Cycle-Accurate Simulators for ARM." `http://http://sourceforge.net/projects/simit-arm/`, 2004.

[22] A. Sinha and A. Chandrakasan, "JouleTrack: A Web Based Tool for Software Energy Profiling," in *Proceedings of the 38th Design Automation Conference*, pp. 220–225, 2001.

[23] J. R. Villar, "Typhoon Labs-OpenGL ES Tutorials." `http://www.khronos.org/devu/opengles_challenge/`, 2004.