Studying Thermal Management for Graphics-Processor Architectures

Jeremy W. Sheaffer, Kevin Skadron, David P. Luebke
Department of Computer Science
The University of Virginia
{jws9c, skadron, luebke}@cs.virginia.edu

Abstract

We have previously presented Qsilver, a flexible simulation system for graphics architectures. In this paper we describe our extensions to this system, which we use—instrumented with a power model and HotSpot—to analyze the application of standard CPU static and runtime thermal management techniques on the GPU. We describe experiments implementing clock gating, fetch gating, dynamic voltage scaling, multiple clock domains and permuted floorplanning on the GPU using our simulation environment, and demonstrate that these techniques are beneficial in the GPU domain. Further, we show that the inherent parallelism of GPU workloads enables significant thermal gains on chips designed employing static floorplan repartitioning.

1 Introduction

Commodity graphics hardware is evolving at a tremendous rate, with each successive generation adding not only performance but fundamentally new functionality. Graphics processors (*GPUs*) sport sophisticated memory hierarchies, multiple issue, wide parallel SIMD and MIMD pipelines, and NVIDIA's current offering, the "NV40" architecture, implements out-of-order issue [15].

Graphics architectures are also becoming more programmable, with greater program sophistication possible with each generation. Early programmable GPUs had programmability available only in the vertex engine, they were limited with respect to size of programs, and these programs could not have branch instructions. More recent generations have added conditional expressions, the ability to execute much larger programs, and the ability to program the pixel engine. The current generation of hardware even supports fully general branching (loops, subroutines, etc.). As these processors continue to add programmability, they become more general. GPUs offer very high performance in their specialized domain; with massively parallel floating point arrays and the recent trends toward increased programmability, they are beginning to be applied towards scientific computing. Such applications of graphics hardware are known within the graphics community as *GPGPU*, short for General Purpose Graphics Processing Unit.

The rapid rate of innovation in graphics architecture, combined with the need for energy and thermal efficiency, creates a rich design space well-suited for study by the methods familiar to the general-purpose processor-architecture community. Yet the inherent parallelism of GPU workloads makes the design space much richer than for traditional CPUs, and the lack of a suitable publicly-available simulation infrastructure has hampered academic research in GPU architecture. The lack of infrastructure is particularly serious, since in the time required to build a complex simulation infrastructure, the simulated architecture can easily become obsolete. GPU architectures also span a wide range of aggressiveness, from high-end products intended for gaming and scientific visualization, where the emphasis is on performance, to low-end products for mobile applications, where the emphasis is on energy efficiency.

Across this entire spectrum, thermal considerations have become important, with cooling constraints already limiting performance. Furthermore, as evidenced by Figure 1, graphics processors display thermal behavior that is both localized to specific functional units, and bursty. Such activity is ripe for exploitation by dynamic thermal management techniques.

In this paper, we use Qsilver [17] to explore a series of thermal management techniques, ranging from classical dynamic voltage scaling (*DVS*) and clock gating, to techniques like multiple clock domains and temperature-aware floorplans, both of which specifically exploit the parallelism of graphics workloads. Though we also describe two energy efficiency experiments, one varying throughput in the vertex and fragment engines and the other using Multiple Clock Domains or *MCD* while varying leakage rate, we choose thermal management as a driving problem because we feel it is an area with rich rewards for graphics architecture and an area that can draw a great deal from past work in general

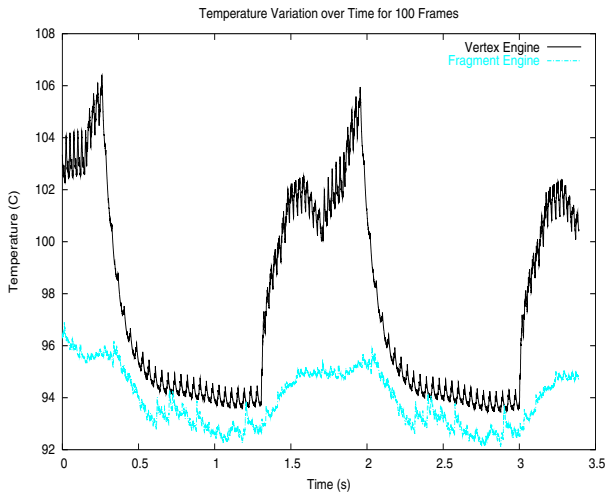

Figure 1. The temperature variation on the chip is both localized, at the granularity of the functional unit block, and bursty. These traces plot temperature over time processing an execution trace on a system based on the base low-resolution floorplan (see Figure 3). It is also evident that the GPU displays both inter- and intra-frame variations in activity and temperature fluctuation. The frequent periodic variations correlate to individual frames.

purpose processor design. Furthermore, and perhaps more importantly, we believe that these two communities—those of general purpose and of graphics processor architects—have much to offer each other. We hope that our work will spur interest that leads to cross fertilization of ideas between these largely disjoint groups.

2 Related Work: Architectural Simulation

The advent of detailed but flexible, configurable, cycle-accurate CPU simulators in the 1990s for complex, superscalar architectures served as the catalyst for an explosion of quantitative research in the computer architecture community. The most prevalent simulator in academic architectural research is SimpleScalar [2]; other simulators used in specific circumstances include Rsim [6] for multiprocessors, as well as Simics [9] and SimOS [14] for capturing operating-system and multi-programmed behavior. By describing instruction flow at the granularity of individual steps through the CPU pipeline, these simulators allowed research and design to move beyond good but imprecise analytical models or cumbersome, logic-level models. Instead, architects could analyze detailed tradeoffs under realistic workloads and estimate how various microarchitectural choices affected instruction throughput. Examples of problems that can now be studied at least partially thanks to academic architectural simulators include the impact of different cache and branch predictor algorithms, the impact of different superscalar out-of-order instruction-issue designs, and the effectiveness of a host of novel CPU organizations such as hyper-threading.

These simulation systems are not alone; new infrastructures continue to appear. One of these, ASIM [11], offers a high degree of flexibility in definition of the system being simulated. It is much like Qsilver in this respect. Unlike Qsilver, ASIM uses an asynchronous event model for communication between blocks, while QSilver, being designed with the GPU in mind, depends on a streaming infrastructure and more conventional intra-block communication.

More recently, power-modeling capability launched another round of innovation by allowing architects to estimate the energy efficiency of different processor organizations, verify that new microarchitecture innovations are justifiable from an energy-efficiency standpoint, and explore microarchitectural techniques for managing energy efficiency. The dominant power model today is Wattch [3], which uses calibrated analytical models to allow flexible and configurable estimation of power for a variety of structures, structure sizes, organizations, and semiconductor technology generations or *nodes*. Other power models that use circuit-extracted data have been described, but they are based on specific implementations and tend to be less flexible. These two approaches can be combined, using the circuit-extracted model as calibration for Wattch's analytical models; see for example a recent study of hyper-threading using IBM's circuit-extracted PowerTimer [8]. Most recently, the architecture community has begun to explore architectural techniques for thermal management, facilitated by the HotSpot [20] dynamic temperature model. This paper extends our earlier work in thermally aware graphics processor design [18].

Our goals with Qsilver are to stimulate the same kind of innovation in the GPU community that products such as SimpleScalar have stimulated in the general purpose architecture community, to stimulate greater cross-fertilization of ideas with the general-purpose CPU architecture community, and to enable new studies in power-aware and temperature-aware design.

3 Description and Modification of the Simulator

3.1 Description of Qsilver

There are two major components of the Qsilver system: the annotator and the simulator core. The annotator gener-

ates an input trace, which the simulator core traces through its cycle timer model. This section summarizes Qsilver as presented in [17].

3.1.1 The Annotator

The Qsilver annotator is built on *Chromium* [7], an OpenGL library interceptor which implements and exports the OpenGL API, an API for realtime 3D graphics. Chromium allows the OpenGL call stream of an application to be intercepted and transformed on the fly, without need of any modification or source code. Chromium is typically used for such applications as scientific simulation and visualization; for example, splitting output over multiple displays or balancing a graphics workload over many GPUs. By implementing several new Chromium stream processing units (*SPUs*), we transform the OpenGL stream in such a way that we can gather aggregate statistics about the activities of the hardware during rendering. These statistics form the input to our cycle-timer simulation.

An unfortunate fault of our Chromium based annotator is our inability to gather non-aggregate data, specifically screen-space positional information and texture coordinates. Many structures, such as the texture cache, cannot be accurately modeled with the current system. Another avenue worth exploring as a base system for the Qsilver annotator is the use of a software renderer like *Mesa* [12]. With Mesa, we would sacrifice some of the speed of the current system, and in exchange get all of the current functionality, as well as the non-aggregate data required for lower level simulation. We intend to move the annotator to Mesa, or some Chromium/Mesa hybrid, in our future work.

Video games drive the graphics hardware market; however many games will not run if they cannot maintain sufficient framerate. Since our annotation system significantly slows the annotated application, we also use Chromium to record and play back an OpenGL trace of the application. This trace is no longer the application itself, but only its calls into the OpenGL API. As the application logic has already been processed, framerate is no longer a concern. We pass this OpenGL trace to the annotator for data aggregation.

The annotation process consists of the following stages, implemented as Chromium SPUs:

1. Expansion of vertex arrays

   OpenGL allows vertices to be specified in two main ways: individually, with component parameters specified explicitly in the call, and as part of a set of vertices stored in an array on the GPU and indexed by an element number. For efficiency reasons, and because only a single call in to the API is required to render complex geometries, the latter, called a *vertex array* or *vertex buffer*, is favored by game developers. As described below, we need all geometry to be broken into individual triangles, so the first task of the annotator is to expand the vertex arrays to produce the triangles which will eventually pass through to the GPU. The most significant problem with complex geometries is that they can be self occluding. The first three SPUs in our SPU chain are designed to eliminate self occlusion. Also note that to correctly model such structures as the vertex cache, it is necessary to maintain an awareness of the original layout of the vertex array.

2. Unfolding of display lists

   Display lists are another efficiency construct of the OpenGL API. Like vertex arrays, display lists allows complex geometries to be rendered with a single API call. We use Chromium to store display lists as they are 'recorded', then monitor the stream for the API calls which play them back. When the call comes through, we play back the recorded source rather than passing the display list invocation though to the driver.

3. Triangulation of geometries

   The GPU deals with geometry in terms of individual triangles, but OpenGL allows specification of geometry in terms of triangle strips and fans, and polygons with an arbitrary number of sides. These complex geometries need to be *triangulated*—turned in to individual triangles—before they can be rasterized. This stage of the annotator further transforms the OpenGL call stream such that subsequent stages receive only triangulated geometry.

4. Query of state and activity

   Rasterization is the process of turning triangles into *fragments*, which can be thought of as a generalization of pixels. The transformations leading to this stage eliminate complex geometry—meaning that all geometry coming to this stage is organized into individual triangles—and with them self occlusion. In order to accurately model a GPU, we must know—among other statistics—how many fragments were generated from a triangle and how many of these fragments pass the depth test and get written to the framebuffer. We wrap each triangle in an *occlusion query* which returns a count of the number of non-occluded fragments that that triangle generates on the hardware. We then render the triangle again into another buffer with the depth test disabled, wrapped in another occlusion query, to determine the total number of fragments generated by the triangle.

   In addition to counting fragments, in this stage we also collect some information on OpenGL state and count

statistics such as the number of texture accesses in this stage. The latter is non-trivial, as magnification and minification texture sampling filters do not necessarily make the same number of texel accesses, and it requires some further manipulation of the OpenGL stream, another rendering of the triangle in question, another occlusion query, and use of the programmable features of modern GPUs. For full details, please see [17].

The annotator ultimately writes an annotated trace to disk. This trace describes what the hardware must do to render the OpenGL trace input. The simulator core takes this annotated trace as input for its timing simulation.

3.1.2 The Simulator Core

The Qsilver simulator core is based on the OpenGL pipeline. Figure 2 is a diagram of a simple GPU architecture. This is the machine we model in the Qsilver core. The simulated architecture has four main functional units: the vertex engine, the rasterizer, the fragment engine, and the framebuffer control unit. These are all decoupled by queues, with the fragment queue acting as a natural decoupler of vertex and fragment portions of the chip. This becomes important when we discuss our multiple clock domains experiment in section 4.2.5.

The Qsilver core traces the input through a timing model. A cycle counter is advanced each time it runs through the pipeline. As the simulation advances, events are counted and aggregated at a user defined granularity. There are currently dozens of events monitored. These include the number of vertices transformed, the number of vertices lit, the number of fragments generated, the number of texture accesses, and the number of fragments that fail the depth- or Z-test. A typical aggregation period is on the order of 25000 cycles. The user can make this as fine as a one cycle aggregation period for true micro-level output, however the shear quantity of data generally precludes such a practice.

3.1.3 The Power Model

Qsilver's power model is based on an industrial power model for a high performance, general purpose processor (CPU). We scale this power model to account for differences in voltage, frequency, process node and bit width. We acknowledge, of course, that differences in the microarchitecture and circuit design methodology impair our ability to apply a power model from the CPU realm to a graphics processor. For example, modern, high-performance general purpose processors tend to use a custom or semi-custom design, while industry sources tell us that GPUs primarily use standard cell technology. Unfortunately, no industry models are available for validation, but we believe our model provides relative accuracy for many coarse-grained architectural studies, even if absolute accuracy is not yet possible. Our future work includes developing an accurate GPU power model for use in Qsilver and by the community at large.

3.2 Changes to Qsilver

We have made significant modifications and extensions to Qsilver since its original publication. In addition to adding thermal modeling capabilities, we have also extended its configurability so that it now allows runtime definition of the architectural pipeline. The next two sections discuss these modifications in more detail.

3.2.1 Runtime Pipeline Configuration

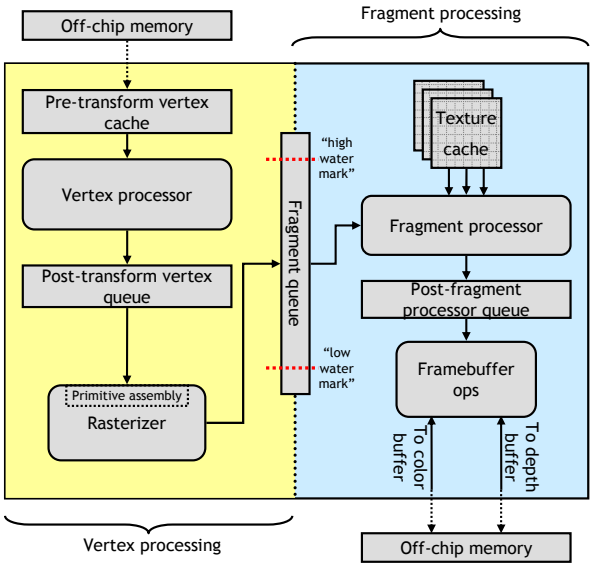

Figure 2. The stages of the rendering pipeline are joined by decoupling queues. The fragment queue in particular serves to decouple the vertex and fragment "halves" of the chip. This latter proves useful in implementing, for example, multiple clock domains on the QSilver framework. The vertex and fragment portions of the processor, clearly separated by the fragment queue, form logical domains for separate clocks. The high- and low-water marks in this figure are used in the MCD power experiment described in Section 4

At the time of original publication [17], Qsilver was highly flexible from the standpoint of runtime configuration

of state variables. We could, at runtime, define such parameters as the number of pipelines in the fragment core, the length of the vertex queue, or the bandwidth of the network between the fragment queue and fragment engine. A major failing was the inability to define the simulated pipeline itself at runtime.

We undertook a complete rewrite of the Qsilver system. The changes to the annotator are mostly cosmetic, but the changes to the simulator core itself are significant. The most important modification is our addition of runtime pipeline configuration.

Qsilver now is essentially a stream system. The user writes kernels, in the form of C functions, representing each stage of the pipeline they wish to simulate (of course they also may use an existing kernel). The stream infrastructure sees that the kernels receive input, and it is the responsibility of the pipeline builder to signal the infrastructure when work is complete on a particular datum. This is similar to the approach that was used in the *Liberty* system [21].

At runtime, the pipeline itself is organized into a list of function pointers. The simulation infrastructure traverses the list and calls each function or kernel with its associated input and a state data structure that is also stored in the list. With this organization, it is straightforward to run multiple simulations, each using a different pipeline configuration, without need for a recompile. Configuration of the pipeline, as well as all runtime state variables, is handled in a single, simple configuration file.

3.2.2 Thermal Modeling with HotSpot

We have augmented Qsilver with *HotSpot* [5, 19], a tool for architectural thermal modeling. Drawing from the power model described in section 3.1.3 and based on the floorplans in section 4.2.1, HotSpot is invoked at the end of every aggregation period to calculate the temperature of each functional unit.

HotSpot builds a system of differential equations which describe a complex RC circuit—conveniently thermal systems are modeled in exactly the same way as electrical RC circuits, with a thermal resistance and capacitance—and solves them numerically for temperatures.

4 Experiments and Results

Figure 1 is a trace of the thermal activity of the vertex engine and the framebuffer control unit over 100 frames of data. From this plot, we can see that thermal activity is both localized and bursty. The dynamic thermal management techniques discussed below capitalize on this characteristic.

Given the limitations of our power model combined with the dependence of temperature on power, and the fact that we have not fully explored the design space for any of these techniques, we must acknowledge that the 'error bars' on our results are large. Without a more accurate power model, we cannot precisely, quantitatively compare the techniques listed below. Nonetheless, we reiterate that we believe our results hold qualitatively, and that interesting insights can be gained from this work. As a case in point, below we show that DVS outperforms MCD with respect to performance in managing temperature on the GPU. This is consistent with the literature, but with our large and unquantified error, the small difference in performance of these techniques, and the fact that MCD does a better job of controlling temperature, we should probably consider these techniques to be in an equivalence class! Both present interesting room for further study.

4.1 Experiments

On top of the QSilver framework we implemented the following thermal management techniques:

- Global Clock Gating

  This technique stops the clock when any monitored unit on the chip exceeds a threshold temperature. The clock is restarted when the temperature of all the units is once again below the threshold. With this technique, the processor is effectively turned off whenever the chip experiences thermal stress, except that state is preserved and leakage currents are unaffected.

- Fetch Gating

  Fetch gating, in a general purpose processor, toggles on and off the instruction fetch stage of the pipeline, effectively reducing the throughput of the entire pipe. Due to the decoupled nature of the rendering pipeline, we believe that this technique can be used effectively in *any stage* of the pipeline. We have tested it, with positive results, on the vertex fetch and transform-and-light (another name for the vertex engine) stages.

- Dynamic Voltage Scaling

  In Dynamic Voltage Scaling, or DVS, voltage is reduced upon crossing the thermal threshold. With lower voltage, the switching speed of the transistors is decreased, so it is necessary to also scale frequency. Because $P \propto V^2 f$, this technique achieves a roughly cubic reduction in power relative to performance loss[1]. Employing this technique, the power saved with even a small change in voltage can be significant, with little detriment to performance. This is despite a switching penalty incurred while the clock resynchronizes.

[1] Note that we do correctly account for the non-linear dependence of f and V.

- Multiple Clock Domains

  Multiple Clock Domains, or MCD [16], is an architectural technique whereby separate functional units or sets of units on a chip are operated by different clocks—essentially DVS is employed at the granularity of the functional unit block, with mostly the same benefits and drawbacks as that solution. This technique improves efficiency when different portions of the chip have imbalanced workloads—in the GPU realm, for example, an architect may want to slow down the vertex engine to save power when the processor is fill bound [17]. Thermally, it may be practical to slow down only the hot stage of a pipeline rather than to penalize the entire chip for a single unit's poor behavior. Again the decoupling inherent in the graphics pipeline is advantageous to the architect here, making the implementation of this technique on the GPU far easier than its equivalent on a general purpose processor. In fact, existing GPUs already make heavy use of non-dynamic MCD, so little modification should be necessary to take advantage of these clocks for thermal management.

- Thermal-aware Floorplanning

  This static technique puts space between the hottest units of the chip, so there is more cool area surrounding them to help spread heat. We take floorplanning one step farther, by producing unique layouts where some functional units are broken into smaller constituent parts. This is possible on the GPU, because, for example, the fragment engine actually contains multiple fragment pipelines (e.g. NVIDIA's new NV40 architecture has 16) [15]. Many of the functional units on the GPU contain multiple pipelines, making this technique an especially rich avenue for exploration. Of course, breaking up units imposes the overhead of longer communication pathways, but due to the high prevalence of queues in the GPU, communication latency doesn't seem to play a major role in performance.

- Power Management

  Graphics hardware is moving into embedded devices like PDAs and cell phones [1, 13]. For example, it has been suggested that the most played electronic game of all time is the 'snake' cell phone game [13], which demonstrates that even in the embedded market, games are likely to become the driving application for graphics hardware. As embedded devices ship with more sophisticated graphics capabilities, battery life becomes a greater concern. Energy efficiency of these systems is of major importance to the vendors. We can use Qsilver to conduct energy efficiency studies on our simulated architectures.

  In [17], we presented several power management results on the GPU using Qsilver. Among these are studies of unit throughput and MCD for power management, which we reiterate here. In the throughput experiments, we vary the processing rates of our vertex and fragment engines to discover the highest performing and the most energy efficient design points. For the MCD experiment, we note that activity on the GPU tends to alternate between being vertex bound and fragment bound and take advantage of this by implementing two clock domains, one before the fragment queue, and one after, and scaling the voltage in one domain when the other has a workload above a certain threshold. Figure 2 illustrates high- and low-water marks on the fragment queue. We implement a simple state machine to avoid oscillations between MCD state. We require that the queue remain within 10% of maximum capacity or empty for at least 50000 cycles before turning on MCD, and use a similar test to decide when to turn MCD off.

4.2 Results

All thermal results are summarized in Table 1. In our experiments we assume that:

- Case ambient temperature is 45°C
- The cooling solution is under-designed, and consists only of a small, aluminum heatsink with no fan
- The vendor specifies a maximum safe operating temperature of 100°C
- The chip has one temperature sensor for each functional unit block
- Sensor precision is specified to be within ± 3°C

Furthermore, all DTM techniques are employed for multiples of 25000 cycles. HotSpot is only invoked at the end of an aggregation period, and As we are modeling 300MHz processors, this corresponds to an 85μs sampling interval. Our input traces are generated from an OpenGL stream we captured from Splash Damage's game *Wolfenstein: Enemy Territory*. We have two traces, both identical save resolution: one is 800×600 pixels, and the other is 1280×1024. These traces contain frames which are typical for this game, and which contain mixes of large and small triangles, texturing modes, etc. Both are 50 frames long. We 'play' traces twice in succession to get 100 frames worth of simulation data. Section 4.2.1 illustrates all four of the floorplans we use in these experiments.

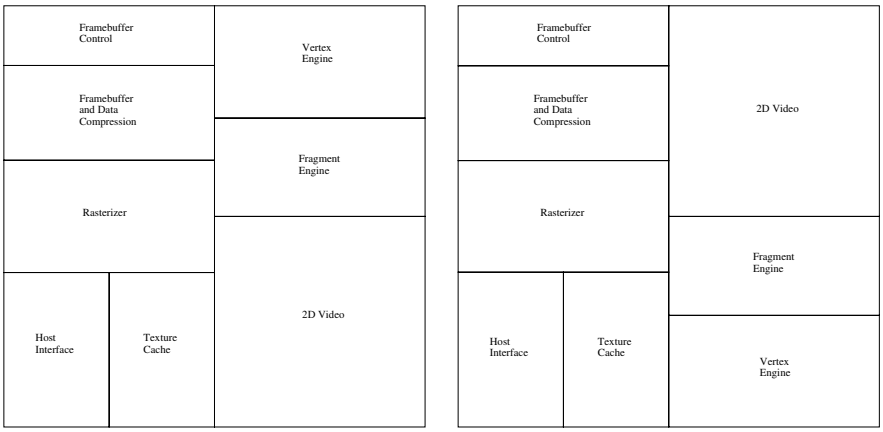

Figure 3. The floorplan on the left is based on an NVIDIA marketing photo for the GeForce4. The units labeled framebuffer_control and vertex_engine are the hotspots, and placed adjacently. The right floorplan moves the vertex unit to the opposite corner of the chip, reducing the maximum temperature of each unit and decreasing the thermal gradient across the chip. We use these two floorplans with our 800×600 resolution trace. We add a cycle of latency between pipeline stages when using the repartitioned floorplan to account for increased wire length.

4.2.1 Floorplans

We consider floorplan reorganization itself a thermal management technique, though a static one, and demonstrate that the parallelism inherent in GPU workloads allows big rewards to be garnered from careful, thermal-aware design of functional unit layout. Figure 3 shows two floorplans. The left layout is based on an NVIDIA marketing photo[2]. The right layout moves the vertex processor down to the bottom of the right side of the chip. This places the two hottest units—the vertex engine and the framebuffer operations unit—at opposite corners of the chip, reducing their respective maximum temperatures by about 1°C each. When attempting to maintain interactive framerates—30 to 60 frames per second—with higher resolution traces, the necessary addition of fragment processing power to this chip caused unrealistic power densities in some units. As a result, we produced the new floorplan on the right in Figure 4. This chip layout slightly increases the area of the framebuffer operations unit, nearly doubles the size of the fragment core, and increases the area of the texture cache by a factor of 2.5, but gives room for the addition of 4 more fragment pipes. The left layout in this image makes a more radical change than that used on the low resolution chip. We attempt to separate hot units with cooler ones, and in doing so, break up some of the units into constituent parts. We have 12 fragment pipes on this chip, so we separate the fragment engine into three blocks with four pipes each. We also separate the vertex engine, as it consists of two distinct pipelines. This layout preserves unit area, but does add some dead space to the silicon[3]. With both rearrangements, we add a cycle of latency between pipeline stages to account for the extra wire length.

Without any DTM technique engaged, and with our under-engineered cooling solution assumptions in place, all four of these chips exceeded the 100°C maximum operating temperature. In the case of the two low resolution floorplans, 100% of the cycles of the simulation executed at temperatures in excess of the safe operating temperature, and both chips topped 105°C. Figure 5 is a thermal map of the base low-resolution design at the hottest point in a 100 frame input trace, with no DTM enabled.

4.2.2 Global Clock Gating

Clock gating is a fairly primitive technique, but simple to implement. In exchange for its simplicity our results demonstrate a very high cost in performance. Our base floorplan runs 62% slower with clock gating turned on, which corresponds with a system in which the clock is gated 38% of the time! We see a behavior here in which the processor is gated after one temperature update, then ungated after the next, and then gated again after the following, etc. It is only during the cooler stages of the execution that the

[2]Industry sources have commented to us with doubts about the correctness of this photograph's labels.

[3]Note that this transformation does not maintain the aspect ratios of the split blocks. Increased border area of the separated units most likely contributes to the effectiveness of this technique by increasing lateral thermal conduction. This is an interesting space to explore in future work.

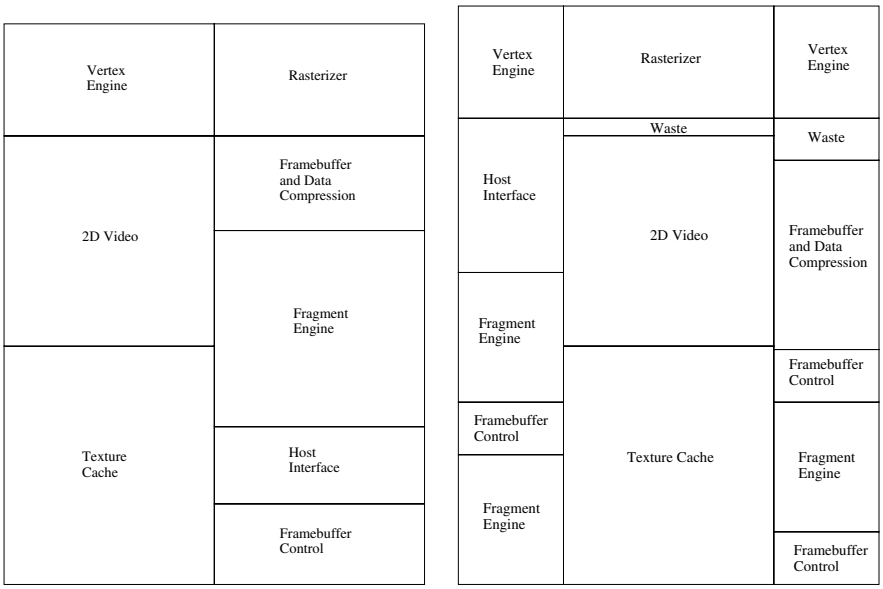

Figure 4. In order to maintain reasonable power densities with our 1280×1024 trace, it was necessary to increase the area of some of the units. The left hand floorplan is a conventional layout for these resized blocks. The right floorplan preserves unit area while spreading out the hotspots across the chip. We add a cycle of latency between pipeline stages when using the repartitioned floorplan to account for increased wire length.

processor can spend significant time running consistently.

4.2.3 Fetch Gating

We have employed fetch gating on two different stages of the pipeline:

- **Vertex Fetch**

  Canonically, fetch gating is employed in the instruction fetch stage of a general purpose processor. The logical extension of this idea to the GPU domain is to toggle vertex fetch. An instruction on a CPU, though, does not generate more work, while three vertices—a triangle—can potentially generate over a million fragments on a system with display resolution of 1280×1024. Even with more realistically sized triangles, it is not unreasonable for hundreds to thousands of fragments to be created from each triangle, and since vertices are queued between the fetch and transform-and-light stages, toggling vertex fetch is ineffective in controlling thermal behavior. In fact, both the low- and high-resolution base floorplans exceed the maximum safe operating temperature while also incurring not insignificant performance penalties. DTM is engaged in these two cases 100% and 67% of the time, respectively.

- **Rasterizer**

  The rasterizer draws input from the post transform-and-light queue and writes to the fragment queue. At some level, it can be thought of as the creator of the work that is done in the second half of the pipeline. 'Fetch gating' the rasterizer was able to keep the processors beneath the maximum operating temperature, but incurred unacceptable performance penalties. The base floorplans took performance reductions of 90% and 17% for low- and high-resolution designs respectively.

4.2.4 Dynamic Voltage Scaling

Global dynamic voltage scaling scales the voltage uniformly over the entire processor. In our experiments we scale voltage by 20%, which gives a nearly 50% savings in power. We choose 20% because this seems to be the upper limit on voltage scaling without adversely impacting caches and other memory structures. Additionally, we impose a 3000 cycle synchronization penalty whenever DVS state is changed. For all processors, save the one based on the base low-resolution floorplan, this technique was able to maintain safe operating characteristics for the processor, while incurring performance penalties of less than 3.5%. In the case of the base low-resolution design, the processor had

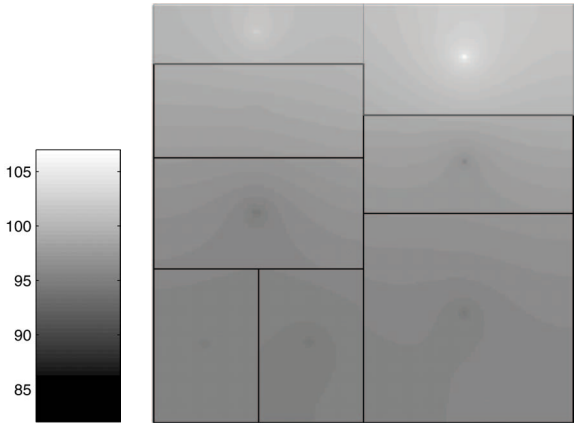

Figure 5. The hottest point in an execution trace on the base low-resolution design with no DTM. Note how the two hottest units are located adjacently, leading to a large thermal gradient across the chip. Such situations as this stress the cooling solution, which is insufficient in this case. See Figure 3 for unit labels.

DVS turned on 71% of the time and only incurred a 13% slowdown, or about 1 frame out of 8. This experiment only slightly exceeded 100°C, and an only slightly higher voltage scaling factor would have been sufficient to keep the temperature within the safe zone with similar performance. Figure 6 illustrates the thermal gradients across the high-resolution partitioned chip with DVS at the hottest point in the simulation.

4.2.5 Multiple Clock Domains

Multiple Clock Domains is the second voltage scaling technique we have employed. Unlike DVS, MCD employs voltage scaling at a sub-chip granularity. In this case, we've implemented MCD at the granularity of the individual functional unit block. Whenever the DTM state changes for any functional unit block, the entire chip must be stopped while clocks resynchronize. As with DVS, we impose a 3000 cycle, or 10μs, penalty for synchronization whenever MCD state changes. MCD is the only technique, under our assumptions, that is able to both keep all four processor designs within safe operating limits and maintain reasonable performance in all cases. In the worst case, the base low-resolution design, the processor ran with MCD engaged for 55% of all cycles, incuring less than a 17% performance penalty. Our best performing design here, the partitioned high-resolution chip, was only slowed by 0.5%.

4.2.6 Thermally-Aware Floorplanning

Independent of the above dynamic thermal management techniques, thermally-aware floorplanning deserves mention on its own. Without any active dynamic techniques, intelligent floorplanning was able to reduce the maximum temperature of the low resolution design by .9°C, and the high resolution design by 2.8°C. The lower interdependence between unit blocks on the GPU, as compared to traditional CPU architectures, makes the kind of radical changes we have employed possible.

It seems counter-intuitive that the high-resolution systems should perform so much better thermally. Among possible explanations for this are that the units were scaled to account for power density and not for thermal characteristics. The fragment engine was never a concern thermally, yet it has increased in size. The framebuffer operations unit, due to increased fragment activity has grown, but only by a small amount. The vertex engine, one of the hottest units, is doing exactly the same work, and so it has not been changed. And we were forced to add some dead silicon. All told, we have significantly increased the total area of the chip, while making only a small increase in the area of the hottest units, thereby decreasing the mean temperature of the chip, and increasing lateral cooling ability.

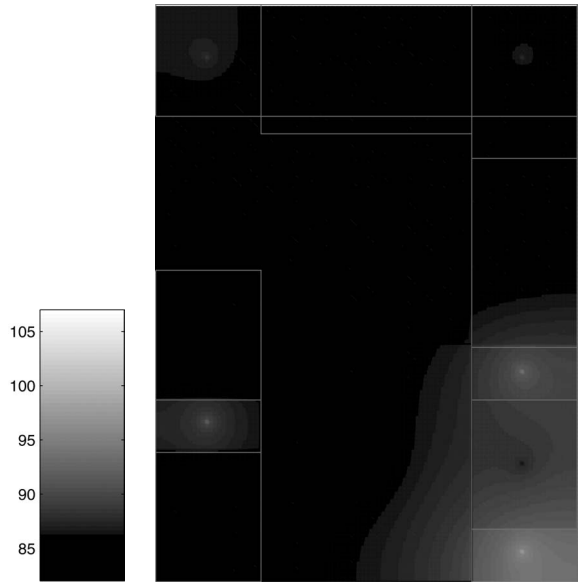

Figure 6. With DVS and static repartitioning of the floorplan, the chip has a much lower maximum temperature. Furthermore, the thermal gradients are smaller here so that the cooling solution is better able to do its job. See Figure 4 for unit labels.

Floorplan	Base		Permuted Base		High Resolution		*Partitioned High Res*	
DTM Technique	Performance Cost	Maximum Temperature	Performance Cost	Maximum Temperature	Performance Cost	Maximum Temperature	Performance Cost	Maximum Temperature
No DTM	0.0%	106.4	0.0%	105.5	0.0%	103.7	0.0%	100.9
Clock Gating	62.0%	97.0	13.6%	97.0	14.8%	97.0	0.7%	97.0
Vertex Fetch Gating	25.9%	102.9	10.2%	98.7	9.2%	101.3	0.5%	98.1
Rasterizer Fetch Gating	90.1%	98.1	17.7%	97.8	17.4%	97.0	0.7%	97.8
Dynamic Voltage Scaling	13.1%	100.7	*3.4%*	*98.2*	*3.4%*	*97.4*	*0.1%*	*97.0*
Multiple Clock Domains	*16.7%*	*98.4*	*4.1%*	*97.0*	*3.7%*	*97.0*	*0.5%*	*97.4*

Table 1. A summary of all thermal results. Notable results are marked in italics; particularly poor results in sans serif. As a general observation, note that the two voltage scaling techniques, DVS and MCD, performed very well, while the more primitive gating techniques tended to impose heavy performance penalties and exceed maximum operating temperature. Also note that static floorplan partitioning gives impressive results as well. All temperatures are in degrees centigrade.

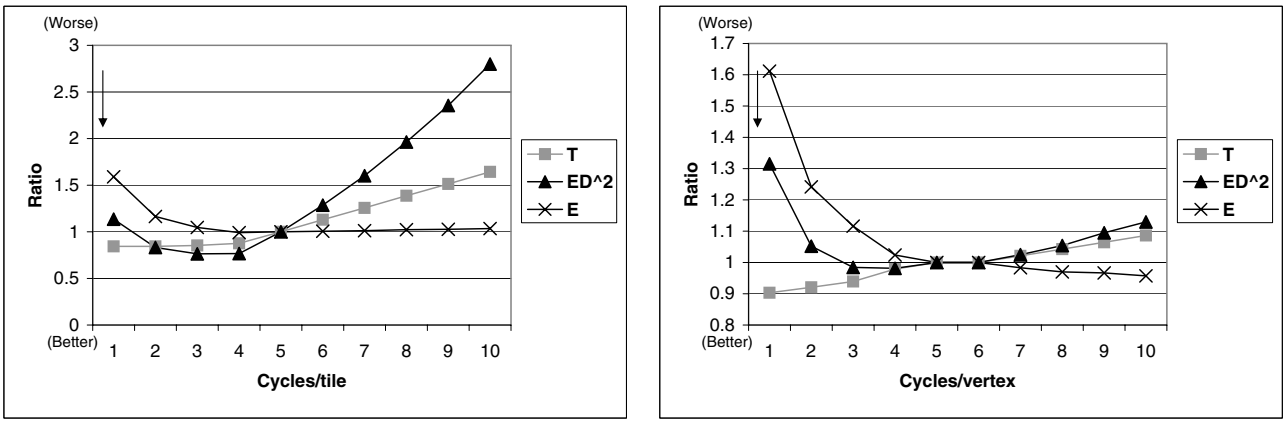

Figure 7. Performance (T) and energy-efficiency data (ED^2 and E) for different fragment- and vertex-processing rates. All results are normalized to our base cases, which are 5 cycles/tile and 18 cycles/vertex, respectively. Note also that in all cases, a smaller ratio is better.

4.2.7 Power Management

We implement two energy efficiency experiments, one varying unit throughput and one using MCD. In both of these experiments, we choose ED^2 as our energy efficiency metric. In the first experiment, we vary fragment and vertex engine throughput rates to locate the most energy efficient and the best performing design points—these are usually not the same point. Note that we have Qsilver configured to process fragments in SIMD tiles of 2×2 fragments. From the plots in Figure 7 we see that in the fragment engine, the optimal design point from an energy efficiency perspective is at 4 cycles per tile. In the case of the vertex engine, the energy efficiency optimum comes at 4 cycles per vertex. In both cases, as would be expected, performance increases with throughput.

The MCD power efficiency experiment measures the impact of varying the leakage ratio while using MCD for power management. Figure 8 presents a comparison of performance and energy efficiency for three different leakage rates, the only variable changed. In all cases the performance loss was only 1.5%; while for a leakage rate of 10% MCD achieved an 11% energy efficiency gain, a leakage rate of 50% corresponds with a 28% increase in efficiency!

Because leakage is exponentially dependent on voltage, results are even better for future technology nodes. This suggests that MCD will become increasingly effective in future.

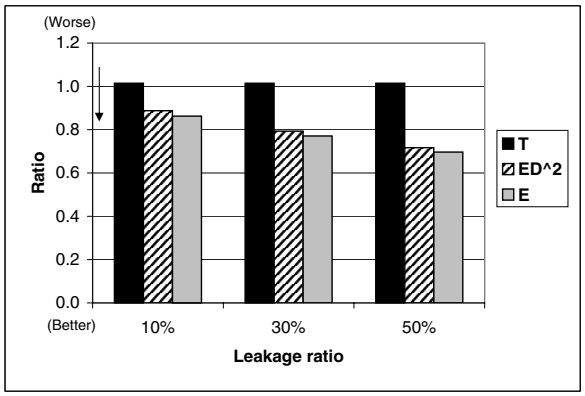

Figure 8. Performance and energy-efficiency data of MCD for different leakage ratios. All results are normalized to the base case (with no MCD) for the appropriate leakage ratio. Smaller y-axis values are better, representing better performance and better energy efficiency.

5 Conclusions

We have extended Qsilver, our graphics architectural simulation system, to allow runtime pipeline configuration and to enable studies of dynamic thermal management.

We have also demonstrated that standard CPU thermal management techniques map well to the GPU. Voltage scaling techniques—dynamic voltage scaling and multiple clock domains—give much better results in this domain than the more primitive gating techniques we have explored. DVS imposes less performance penalty than MCD, but MCD is the only dynamic technique that was able to keep our simulated processor within safe operating limits in all cases.

Our results indicate that static floorplan repartitioning, which separates hot units and splits up pipelines, may hold rich rewards for the GPU domain. This is because of the highly parallel nature of GPU workloads and design spaces. With one of our repartitioned floorplans, addition of DTM techniques imposed negligible performance penalties.

Lastly we presented two power management techniques: throughput variation and MCD. As graphics hardware continues to move into the embedded domain, such techniques will grow in importance. Indeed, vendors are already considering such ideas to extend battery life of handheld graphics-enabled devices.

6 Discussion and Future Work

The input trace fed to QSilver contains only aggregate information about the rendered primitives. In order to accurately model, for example, a texture cache, the input stream must also contain position information. As discussed in section 3.1.1, we intend to use Mesa to extend the trace generator (discussed in detail in [17]) to encode such non-aggregate information so that we can more fully model modern architectures. The graphics literature is rich with architectural innovations such as Z_{min} culling [1] and Z_{max} or *hierarchical Z-buffer* culling [4, 10]. These techniques cannot easily be implemented without positional information on fragments coming through the pipeline. We plan to implement these and other techniques, and study their efficiency and thermal characteristics.

We feel it is important to create a new, more precise power model to replace our scaled CPU model. We are developing a series of comprehensive micro-benchmarks, physically measuring the power used for each operation, much as was done by the Wattch group [3].

Our voltage scaling implementations are based on all-or-nothing approaches—voltage scaling is on or off, with a fixed scaling factor. We note that although DVS is consistently beating MCD thermal management in terms of performance, MCD maintains a lower maximum temperature in all explored cases. We believe that with a feedback mechanism that dynamically modifies the scaling factor, MCD could beat DVS in performance while still maintaining safe operating temperatures.

While we believe that Qsilver is qualitatively sound, we cannot hope for it to be quantitatively accurate without some means of validation and verification. We are not yet certain how to accomplish validation, and would like to issue a challenge to the community to solve the problem, and to the vendors to make the problem more tractable. Graphics hardware vendors are traditionally very closed-mouthed. Their unwillingness to talk about their products makes academic research in the graphics architecture domain very difficult. Just as the CPU industry continues to benefit from architectural innovations originating in academia, vendors could expect useful innovation and analysis from architects in academia if they were more open.

Even without a proper means of validation, due to its flexibility and extensibility, Qsilver is a good first step toward a cycle-accurate GPU simulator and a simulator accurate enough to be the validation engine for future endeavors in graphics architecture.

Qsilver source is available for download and use at `http://qsilver.cs.virginia.edu/`.

7 Acknowledgments

This work is supported in part by the National Science Foundation under grant numbers CCR-0133634, CCR-0306404, Department of Energy grant number DE-FG02-02ER25539, a grant from Intel MRL, and an Excellence Award from the University of Virginia Fund for Excellence in Science and Technology. The authors would also like to thank the anonymous reviewers for their helpful suggestions.

References

[1] T. Akenine-Möller and J. Ström. Graphics for the masses: a hardware rasterization architecture for mobile phones. *ACM Transactions on Graphics*, 22(3):801–808, 2003.

[2] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(4):59–67, Feb. 2002.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[4] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM Press, 1993.

[5] W. Huang, M. R. Stan, K. Skadron, S. Ghosh, K. Sankaranarayanan, and S. Velusamy. Compact thermal modeling for temperature-aware design. In *Proceedings of the 41st Design Automation Conference*, June 2004.

[6] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve. Rsim: Simulating shared-memory multiprocessors with ilp processors. *IEEE Computer*, 35(4):40–49, Feb. 2002.

[7] G. Humphreys, M. Houston, R. Ng, S. Ahern, R. Frank, P. Kirchner, and J. T. Klosowski. Chromium: A stream processing framework for interactive graphics on clusters of workstations. *ACM Transactions on Graphics*, 21(3):693–702, July 2002.

[8] Y. Li, K. Skadron, Z. Hu, and D. Brooks. Understanding the energy efficiency of simultaneous multithreading. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, Aug. 2004.

[9] P. S. Magnusson et al. Simics: A full system simulation platform. *IEEE Computer*, 35(4):50–58, Feb. 2002.

[10] S. Morein. ATI radeon HyperZ technology. Presentation at Workshop on Graphics Hardware, Hot3D Proceedings, ACM SIGGRAPH/Eurographics, 2000.

[11] D. Nellans, V. K. Kadaru, and E. Brunvand. ASIM-An asynchronous architectural level simulator. In *Proceedings of GLSVLSI*, April 2004.

[12] B. Paul et al. The mesa 3-d graphics library, 1993–2004. http://www.mesa3d.org/.

[13] K. Pulli. Graphics everywhere—pixels in your pocket, 2004. Keynote Talk, Graphics Hardware 2004, http://www.graphicshardware.org/Presentations/KariPulli.pdf.

[14] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34–43, Winter 1995.

[15] D. Salvator. Preview: Nvidia's geforce 6800 ultra, 2004. http://www.extremetech.com/article2/0,1558,1566805,00.asp.

[16] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 29–40, Feb. 2002.

[17] J. W. Sheaffer, D. P. Luebke, and K. Skadron. A flexible simulation framework for graphics architectures. In *Proceedings of Graphics Hardware 2004*, Aug. 2004.

[18] J. W. Sheaffer, K. Skadron, and D. P. Luebke. Temperature-aware GPU design. Poster at ACM SIGGRAPH, Aug 2004.

[19] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M. R. Stan, and W. Huang. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 2004.

[20] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13, Apr. 2003.

[21] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural exploration with liberty. In *Proceedings of the 35th International Symposium on Microarchitecture*, November 2002.