Tile-Based GPU Optimizations through ESL Full System Simulation

Hsu-Yao Huang
Information and Communications Research Labs
Industrial Technology Research Institute
Hsinchu, Taiwan

louis@itri.org.tw

Chi-Yuan Huang and Chung-Ho Chen
Dept. of Electrical Engineering and Inst. of Computer & Communication Engineering
National Cheng Kung University
Tainan, Taiwan

arks.huang@gmail.com, chchen@mail.ncku.edu.tw

***Abstract*—We present a tile-based GPU design which is modeled in a full system simulation platform. The full system simulation platform includes a functional Linux-based system on which the GPU is incorporated for design explorations. To accurately estimate the execution time of the application graphics software, an execution time synchronization mechanism for the virtual platform is developed. We extend the Ericsson Texture Compression (ETC) scheme in our GPU to support alpha compression. In this way, we are able to reduce the external memory accesses to about one sixth, and speed up the rasterization engine (RE) by 35%. We also optimize the hardware-and-software data flow through the full system design platform and obtain significant improvements.**

I. Introduction

Our early efforts [1-2] have resulted in a successful design and functional verification of a 3D rendering system prototype that consists of graphic accelerator hardware and the entire graphics software stack. This 3D rendering system was simulated in a full system simulation platform based on QEMU-SystemC simulator as shown in Figure 1. We have implemented the RTL design of the GPU in the CoWare environment which had been integrated into the functional QEMU-simulation system such that a full system simulation is possible. TABLE I. shows this verification environment. While the entire simulation system is able to verify the functional design from application down to the RTL module, nevertheless, the system is only able to give cycle times of the hardware modules. There is no indication on how well the software, such as the GPU device driver, performs, and also there is no ways to reveal the interaction efficiency of the graphics application and the hardware designs.

In this paper, we present an abstract level of the tile-based GPU model using SystemC with cycles obtained from the RTL design and develop an execution time synchronization mechanism that estimates the execution cycles of the application software run on the QEMU virtual platform. With this, we are able to further improve our rendering system through optimizations, such as data flow and architecture, and reduce the overall system overheads.

The rest of the paper is organized as follows. Section II discusses the performance model. Section III describes methods for optimizations. Section IV presents the evaluation results and compares the performance with other GPUs. Finally, we conclude the paper in Section V.

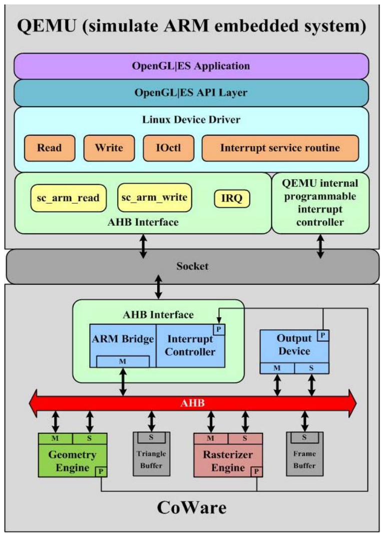


Figure 1. Full system simulation virtual platform

TABLE I. VERIFICATION ENVIRONMENT

Host machine	Cent OS：Kernel 2.6.9
QEMU	ARM Versatile Platform Baseboard for ARM926EJS
	Debian GNU/Linux 4.0
	Kernel Image 2.6.18-6 for Versatile
	GCC 4.1.2
CoWare	Platform Architecture v2009.1.2
ModelSim	6.3a

II. Performance evaluation Model

To truly reflect the software overheads in system performance, we have constructed a synchronization mechanism between the QEMU simulation model and the GPU hardware in CoWare through event synchronizations. In this way, we can obtain the total system cycle count from the instruction count running in the host CPU and the hardware execution cycles. The breakdown of the execution cycle count would in theory include the cycles of the hardware GPU module modeled in SystemC, and cycles of the App (3D application), DLL (driver, system call, etc), and the kernel overheads. To characterize the total execution cycles of the graphics application, we focus on the total cycle count of the former three parts.

We divide the cycle count into four components as follows:

- Context：Initialization of the GPU
- GE：Set up context + Do GE + Wait for polling
- Linklist：Refresh display list
- RE：Execution time of rasterization

Specifically, the Context and Linklist parts are software overheads. Context is the cycle count of that the ARM processor initializes the GPU while the linklist part accounts for the processing time of preparing the display list for the rasterizer engine (RE). In the next section, we show an optimization technique that offloads the linklist part to the tile divider unit in the geometry engine (GE).

III. Optimizations

In this section, we propose various optimizations for the 3D rendering system.

A. Data flow

1) Using vertex buffer instead of data register

Before the GE can start, the processor should write a 12-word vertex data into the registers of the GE, and then processor waits for interrupt from the GE. It turns to the next Drawarray until a Drawarray finishes. When only a limited number of data registers be used, significant communication overheads between the processor and the GPU would occur.

Instead of using more registers to store more vertices, we allocate a new memory called vertex buffer to store all the vertices, and thus greatly reduce the communication overheads between the geometry engine and the processor.

2) Display list offloading

The display list which includes a tile list and a state list is a list of sequential data that the RE needs for pixel rendering. A tile list is transformed from a triangle list which is computed by the tile divider in the GE. The tile divider stores triangles computed by the GE in a triangle buffer using a linklist. This linklist is referred to as the triangle list. Each triangle in the triangle list has 15-word triangle data, a 16-bit serial number that shows the order of the tile covered this triangle, and a 16-bit tile number.

In this way, the processor in QEMU can read a triangle list from the memory, and transform it into a tile list. However, this approach has produced huge communication traffic between the processor and the GPU module. We thus offload the above work by hardware-based solution which allows the RE to directly read the list of data without the intervention of the ARM processor.

Another part of a display list is the state list, which is a group of parameters set by the users for the control of depth test, alpha test, alpha blending et al. These parameters are used by the RE for computations in different invocations of the Drawarray command. Observing the communication overheads through the full system simulation, we resort to construct a state list data structure inside the hardware and store the required state information in front of the tile head after the tile divider has processed all triangles. If the RE needs to update the state data for the current Drawarray command, the GE will write the state data in the Display List ahead.

B. Bus architecture

Heavy bus contentions would happen under various conditions. For example, a master of GE reads data from the vertex buffer while another master of GE writes a display list to the triangle buffer. We analyzed the bus utilization and used the multi-layer AHB design. Specifically, the ARM processor connects the GE and RE through the main AHB while the GE and RE access buffers through the multi-layer AHB. This architecture helps in distributing accesses to memories and increasing the parallelism of GPU operations.

C. Texture compression

In [4], the work reports that 70 percent of power is consumed off-chip, and most of them are accessing textures from memory; furthermore, 75 percent of the bus utilization is used in transferring textures. To reduce this traffic, we develop a texture decompression unit for the GPU system since texture compression can be done offline in advance.

Ericsson Research and Lund University have developed a high-quality, low-complexity texture compression called Ericsson Texture Compression (ETC) [5]. They use a codebook generated by starting from random numbers and then optimizing them by minimizing the error for a set of training images. We extend the ETC approach in our GPU to support alpha compression.

IV. Evaluation Results

We have optimized the HW/SW data flow and the GPU architecture design. The functional accuracy of our SystemC GPU model has been verified through the golden C model. The verification indicates that the images generated by the SystemC model have only little variations compared with the output of the PowerVR counterpart. The GPU specification is shown in TABLE IV.

A. Performance

TABLE II. shows the performance improvement when the optimizations are used (column B). The results are for the first frame so that the initialization cycles can be revealed. The linklist cycle now is zero because the Display List operations are offloaded to the tile divider in GE while the context preparation increases due to the setup of more vertices. On the average, the GE performance is speeded up by 96%, the RE by 89%, and the whole system by 70%. Due to space limitation, we are unable to show results of more benchmarks.

In addition to multi-layer bus architecture, we also explore design of using two frame buffers so that they're accessed by the RE and output device alternately. We design our frame buffer with two slave ports, which can be accessed simultaneously, and allow non-blocking accesses of the output device. In this way, the output device execution time is hidden, overlapping in GE and RE. The operation of output device is merely a part of RE, so it guarantees that the output device time is less than that of the GE and RE.

Figure 2 shows the frame per second (FPS) for several benchmarks, including multi-layer bus and the ping pong frame buffer versions as mentioned above. The average FPS is 5.6 times that of the original un-optimized version.

The improvement of RE cycle count with texture compression is presented in Figure 3. The reduction ranges from 10 percent to 50 percent, and the average is about 35 percent. We reduce the usage of the external memory to one sixth in the RGB format, and one fourth in the RGBA format.

While the maximum throughput is 7.407 Mtriangle/sec at GE, 200 Mpixels/sec at RE in our GPU; however, it is understandable that the average throughput in reality is far from the maximum due to system overheads. By running the actual 3D graphics programs through our full system simulation platform, we evaluate the system performance with the valid triangles in GE and pixels in RE. Figure 4 shows the average throughput of GE and RE respectively. When system overheads are considered, the performance of GE is down to 0.92 Mtriangle/sec and RE down to 13.2 Mpixels/sec. This performance however when compared with those without optimizations, improves 48.3 times in GE, and 9.7 times in RE.

TABLE II. IMPROVEMENT OF TOTAL OPTIMIZATIONS

Venus	context	GE	linklist	RE	Total
A(cycles)	13,917	7,433,606	2,511,065	7,749,469	17,708,057
B(cycles)	3,976,336	114,522	0	677,912	4,768,770
(%)	+	-98.46	-100.0	-91.25	-73.07
Porsche	**context**	**GE**	**linklist**	**RE**	**Total**
A(cycles)	250,506	38,987,958	20,375,619	32,309,823	91,923,906
B(cycles)	25,505,542	600,351	0	940,799	27,046,692
(%)	+	-98.46	-100.0	-97.09	-70.58

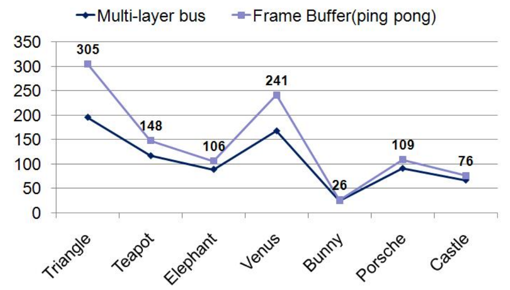


Figure 2. Frames per second

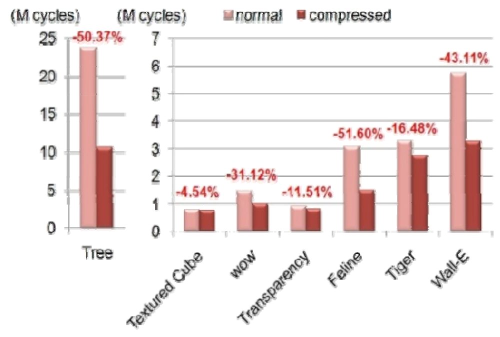


Figure 3. RE performance with texture compression

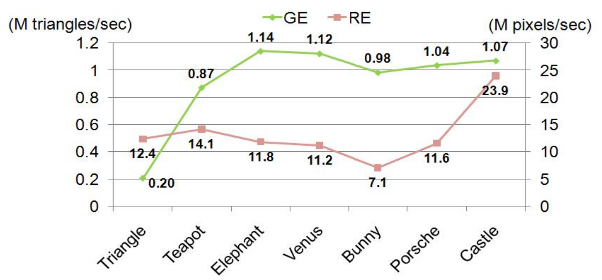
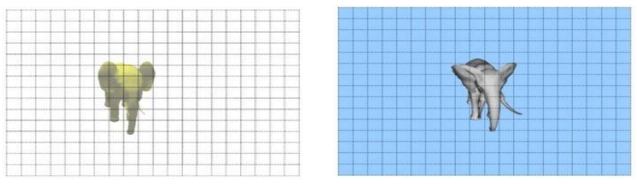


Figure 4. Throughput of GE and RE

B. Comparison

From the above, we can see that using the maximum throughput of GE and RE to indicate the performance of a GPU can be miss-leading. Through our full system simulation, we show that there is a great gap between the average and maximum throughput. And for this reason in performance comparisons, it's best to execute the same program on different GPUs with the equivalent benchmarks and parameters. Unfortunately, there are little details of the benchmarks run on other GPUs.

The work in [8] reports that their rendering system needs to clear the triangle buffer and frame buffer by a DMA on the bus; the size of the frame buffer is equal to ours, and the resolution is also 640x480. It will take millions of cycles to initialize such enormous memory before the RE starts rendering. We speculate that they simply write back the modified pixels from the tile buffer to the frame buffer in order to reduce transmissions. However, the penalty for doing this way is the extra clear time. Instead, our system doesn't have to clear any buffers since we write back all pixels to the frame buffer.



(a) [9] (b) Our benchmark

Figure 5. Elephant benchmark

Another work in [9] uses the ‘Elephant’ benchmark as shown in Figure 5(a), which has 87840 vertices covering 28 tiles on the screen, and the size of the tile is 32 by 32. We found a similar benchmark in Figure 5(b), which is a more complex ‘Elephant’ with 117870 vertices that covers 28 tiles on the screen. TABLE III illustrates their performance and ours, including cycle count of pipeline, DMA , frequency, et al.

TABLE III. COMPARISION OF THE SYSTEM PERFORMANCE

	GE pipeline	GE read	GE write	GE total	RE pipeline	RE read	RE write	RE total
	GE operation	GE wrapper	TD wrapper	GE function	RE operation	RE wrapper		RE function
Our design	2,930,028	2,278,322	425,640	3,106,458	816,141	598,977	614,401	1,368,486
[9]	1,991,537	1,704,047	669,992	2,268,062	2,068,062	1,134,435		2,842,855

(a) Cycle count of GE and RE

	Frequency (Hz)	GE total	# of Vertices	Vertices / sec	RE total	Second
		GE function			RE function	
Our design	200 M	3,106,458	117,870	7.59 M	1,368,486	6.84 m
[9]	139 M	2,268,062	87,840	5.38 M	2,842,855	20.45 m

(b) Throughput of GE and RE

	Frequency (Hz)	context	GE total	RE total	Total Frame	Second	FPS
		Clear Buffer	GE function	RE function			
Our design	200 M	91,918	3,106,458	1,368,486	4,566,862	22.83 m	44
[9]	139 M	18,316,580	2,268,062	2,842,855	21,159,435	152.23 m	7

(c) Total cycle count and FPS

TABLE III. (a) lists the detailed cycle count comparisons while in the two top rows, we showed the corresponding names of functions for clarity, for example, wrapper means read or write. Because of different numbers of vertices, TABLE III. (b) quantifies the GE and RE throughput. However, we don’t have the information about their valid pixels on the screen, so we just calculate the execution time of the RE. The whole system performance is shown in TABLE III. (c). Since the work in [9] needs to perform initializations before rendering each frame, the system spent about 80% of the execution time on this, and this results in a much worse FPS. Our 3D rendering system can achieve 44 FPS on this benchmark, instead. It can be seen that using the proposed full system simulation approach to help evaluating the software-hardware interaction overheads is crucial for overall system performance.

V. CONCLUSIONS

We use a full system simulation virtual platform to develop and optimize a 3D rendering system. First, we construct an abstract level of the tile-based GPU model using SystemC to assess the performance, including texture mapping unit, and estimate the execution cycles and the system overheads. Secondly, an execution time synchronization mechanism of virtual platform is proposed. Last, we optimize the HW/SW data flow and architecture through the simulation platform, which improves the geometry engine performance by 96%, the RE by 89%, and the whole system by 70%.

TABLE IV. SPECIFICATION

Module	GE	RE
Process Technology	TSMC 0.18 um	
Gate Count	537K (include Tile Sorting and Tile Head)	464 K (include texture unit and on-chip tile buffer)
Working Frequency	200 M Hz	
Maximum throughput	7.4 M triangles/sec、22 M vertices/sec	200 M pixels/sec
Supported API	OpenGL ES 1.1 compliant	
Resolution	Up to 640 x 480 (300 tiles、tile size：32 x 32)	
Communication Bus	AMBA 2.0 AHB	
Memory Module	SDRAM (CAS latency 3 cycles)	

REFERENCES

[1] S.-T. Shen, “Full System Design and Simulation of a Multi-view Graphics Processor using QEMU,” Master Thesis, Dept. of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, Jul. 2009.

[2] X.-Z. Shen, S.-Y. Lee, and C.-H. Chen, “Full System Simulation with QEMU: an Approach to Multi-View 3D GPU Design,” *IEEE Int’l Symp. Circuits and Systems (ISCAS ‘10)*, pp.3877-3880, May./Jun. 2010.

[3] C.-Y. Lin, “Performance Modeling for a 3D Graphics SoC,” Master Thesis, Dept. of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, Jul. 2009.

[4] C.-H. Sun, Y.-M. Tsao, and S.-Y. Chien, “High-Quality Mipmapping Texture Compression With Alpha Maps for Graphics Processing Units,” *IEEE Trans. Multimedia*, vol. 11, no. 4, Jun. 2009.

[5] J. Ström and T. Akenine-Möller, “iPACKMAN : High Quality, Low Complexity Texture Compression for Mobile Phones,” *Proc. ACM SIGGRAPH /EUROGRAPHICS Conf. Graphics hardware (HWWS ‘05)*, pp. 63–70, Jul. 2005.

[6] A. Munshi and J.Leech, “OpenGL ES Common/Common-Lite Profile Specification Version 1.1.12,” Khronos Group, Apr. 2008.

[7] D. C. Black, J. Donovan, B. Bunton, and A Keist, *SystemC: From the Ground Up*, 2nd edition, Springer, 2010.

[8] L.-B. Chen, C.-T. Yeh, H.-Y. Chen, et al., “A System-Level Model of Design Space Exploration for a Tile-Based 3D Graphics Soc Refinement,” *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 12, pp. 3193-3202, Dec. 2009.

[9] C.-T. Yeh, L.-B. Chen, C.-Y. Lin, et al., “A Bottom-Up Exploration Approach for 3D Graphics Hardware Accelerator in Consumer Electronics,” *Proc. SASIMI ‘09*, pp. 183-188, Mar. 2009.

[10] ARM Ltd., “Multi-layer AHB Overview,” 2004.

[11] Tom Olson, “ARM Mali-400 MP : A Scalable GPU for Mobile Devices,” http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, ARM Ltd., Jun. 2010.