Hindawi Publishing Corporation
Journal of Sensors
Volume 2016, Article ID 7281031, 9 pages
http://dx.doi.org/10.1155/2016/7281031

Research Article

A Tile-Based EGPU with a Fused Universal Processing Engine and Graphics Coprocessor Cluster

Yang Wang,[1] Li Zhou,[1] Tao Sun,[2] Yanhu Chen,[1] Lei Wang,[2] and Shaotao Sun[3]

[1]*School of Information Science and Engineering, Shandong University, No. 27, South Shanda Road, Jinan 250100, China*
[2]*Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, No. 336, West Nan Xinzhuang Road, Jinan 250022, China*
[3]*Administration Center, Shandong Academy of Information and Communication Technology, Jinan 250101, China*

Correspondence should be addressed to Li Zhou; zhou_li@sdu.edu.cn

Received 16 March 2015; Revised 15 May 2015; Accepted 18 May 2015

Academic Editor: Gwanggil Jeon

As various applied sensors have been integrated into embedded devices, the Embedded Graphics Processing Unit (EGPU) has assumed more processing tasks, which requires an EGPU with higher performance. A tile-based EGPU is proposed that can be used in both general-purpose computing and 3D graphics rendering. With fused, scalable, and hierarchical parallelism architecture, the EGPU has the ability to address nearly 100 million vertices or fragments and achieves 1 GFLOPS per second at a clock frequency of 200 MHz. A fused and scalable architecture, constituted by Universal Processing Engine (UPE) and Graphics Coprocessor Cluster (GCC), ensures that the EGPU can adapt to various graphic processing scenes and situations, achieving more efficient rendering. Moreover, hierarchical parallelism is implemented via the UPE. Additionally, tiling brings a significant reduction in both system memory bandwidth and power consumption. A 0.18 μm technology library is used for timing and power analysis. The area of the proposed EGPU is 6.5 mm $*$ 6.5 mm, and its power consumption is approximately 349.318 mW. Experimental results demonstrate that the proposed EGPU can be used in a System on Chip (SoC) configuration connected to sensors to accelerate its processing and create a proper balance between performance and cost.

1. Introduction

With the development of embedded applications, various embedded platforms and devices have become an essential part of people's daily lives [1]. Mobile phones, hand-held electronics, and automobile electronics have greatly changed the way people live. As the most significant part of an embedded platform, the Graphics Processing Unit (GPU) plays an important role in an embedded system [2].

The concept of GPU was first put forward by NVIDIA in 1999 [3]. Its powerful performance in both 3D graphic processing acceleration and general-purpose computing has attracted considerable attention from researchers in different fields [4, 5]. The hardware design of GPUs has made dramatic progress during the last decade. The modern GPU has evolved from a traditional fixed-function graphics pipeline to a programmable parallel processor. Traditional graphics processing pipelines consisted of fixed-function stages without programmability. With the advent of vertex and fragment shaders, vertices, and pixels could be processed, respectively. However, when these shaders were applied in different processing cases, typical workloads of vertex and fragment shaders were usually not well balanced, which led to inefficiency. Unified shader architecture was then introduced to enable dynamic load balancing of mutative vertex- and pixel-processing workloads [6].

Mobile applications have achieved great success in recent years [7]. The performance of the Embedded GPU (EGPU) has become one of the most crucial factors in evaluating embedded platforms. Compared with a desktop GPU, an EGPU requires equivalent processing performance, reduced energy consumption, better portable APIs, low cost, and more efficient use of memory bandwidth. These critical factors relate to each other dependently and tightly, which determines the optimization strategies in EGPU hardware design. The Immediate Mode Renderer (IMR) and Tile-Based

Renderer (TBR) are two of the most popular renderers in modern EGPU hardware design. Whereas a traditional IMR renders all objects within the screen's boundaries, a TBR renders each screen tile one after the other until the full image is rendered. Furthermore, an IMR relies on the depth values in the Z-buffer to sort the final results. As a result, obscured fragments are still processed, and the amount of overdraw has increased by between 4 and 5 times as modern games have become more realistic [8], causing a great waste in memory bandwidth. On the contrary, TBR determines what is visible and only renders what is necessary to avoid overdraw.

Tile-based methods split the display screen into tiles, which are independent of each other. Because each tile region occupies a small subset of the entire scene, reduced memory access can easily be implemented based on on-chip memory. There are several on-chip memories to support tile-based graphic processing, including the Z-buffer, color buffer, and tile frame buffer. A Z-test is performed based on the on-chip Z-buffer. Pixel processing and blending use the color buffer and the on-chip "tile frame buffer" is used as a local storage area. Compared with a conventional 3D EGPU, a tile-based EGPU reduces memory bandwidth costs and enhances the system performance. All the on-chip processing is performed at high depth and pixel accuracy at the full clock rate without external memory access latency. This approach greatly saves memory bandwidth and, thus, enables modern games and other graphics applications to run with optimized performance [9].

In this paper, a fused EGPU architecture is proposed. The Universal Processing Engine (UPE) cooperates with the Graphics Coprocessor Cluster (GCC), completing graphics processing tasks and general-purpose computation efficiently. This design is a flexible combination that can operate in both tile mode and blending mode to handle most scenes. The proposed tile-based EGPU platform renders opaque objects based on a fused and parallel architecture. Considering that tiles are independent from each other, more processing elements mean more powerful performance and higher parallelism. Two Universal Processors (UPs) are designed in this paper to address tiles rendering under the management of the Universal Processor Controller (UPC). Based on a SIMT (Single Instruction Multiple Threads) [10] architecture, different threads are concurrently distributed to Streaming Processors (SP) in the UPs. Both specific graphics processing and General-Purpose GPU (GPGPU) tasks can be executed and accelerated.

In this work, the following contributions have been made:

(1) The UPE and GCC constitute fused and scalable architecture, allowing the whole platform to operate in different modes to meet various scenes.

(2) Different tiles are assigned to different UPs in the rendering stage, making the system more efficient. SIMT architecture is applied in the UP, whose processing units are allocated with different threads. The entire UPE is implemented with hierarchical parallelism.

(3) A flexible tiling mechanism is designed to finish the Z-test earlier, reducing overdraw and improving the efficiency.

(4) A Memory Hub (MH) is introduced to solve the conflicts in memory access.

The rest of the paper is organized as follows. Section 2 describes the architecture of the proposed EGPU. The experimental results are provided in Section 3. Section 4 presents the conclusion and direction of further consideration for future works.

2. Tile-Based EGPU

In this section, a tile-based EGPU is presented in detail, as shown in Figure 1. The proposed EGPU architecture contains a UPE and GCC for processing acceleration. The UPE's processing operations are based on a unified programmable model, which can be used in both graphics processing and general-purpose computation. The GCC, including tiling, blending, texturing, and interpolating coprocessors, cooperates with the UPE to achieve enhanced graphics processing performance. The UPE and GCC are fused together in the EGPU to constitute a fused hardware platform for different processing situations and application cases. The other components, including command processing, data preparation, data preprocessing, and the raster operations, cooperate with the UPE and GCC to complete the entire graphics processing flow. All components communicate with each other via an internal network and buffers, working as a whole.

A command processor (CP) responds to commands from the host CPU and coordinates with other EGPU components to work harmoniously. The CP analyzes instructions, sends control signals to the data preparation (DP) unit, dispatches instructions to the UPE, and changes the rendering mode according to the current application scenes. The DP unit collects geometric primitives, such as points, lines, triangles, and fetches associated vertex attribute data from the MH. The fetched data are stored in an input buffer so that the UPE can access them directly. After the UPE completes the processing of vertex shading programs, the result data are written to the on-chip internal buffer and are then further processed by the Primitive Assembler (PA)/clip/viewport/setup/raster unit under the CP's control until realization of the final pixel fragments.

The PA assembles related vertices into triangles to build up basic geometry primitives. Then, the viewport and clip units clip the primitives into the standard view frustum. They transform the postclipping vertices into screen (pixel) space and reject primitives outside the view volume as well as back-facing primitives. Surviving primitives are then processed by the setup unit to generate edge equations for the rasterizer. Attribute plane equations are also generated for the linear attribute interpolation of pixels in the pixel shading stage. A coarse rasterization stage generates all pixel tiles that are at least partially covered by the primitive. The UPE reads the relative pixel fragment data from the internal buffer to complete pixel-fragment shading. Shaded pixel-fragments are sent across the interconnection network to the Raster Operation Processor (ROP) unit. Data in the output buffer are finally written to system memory via the MH.

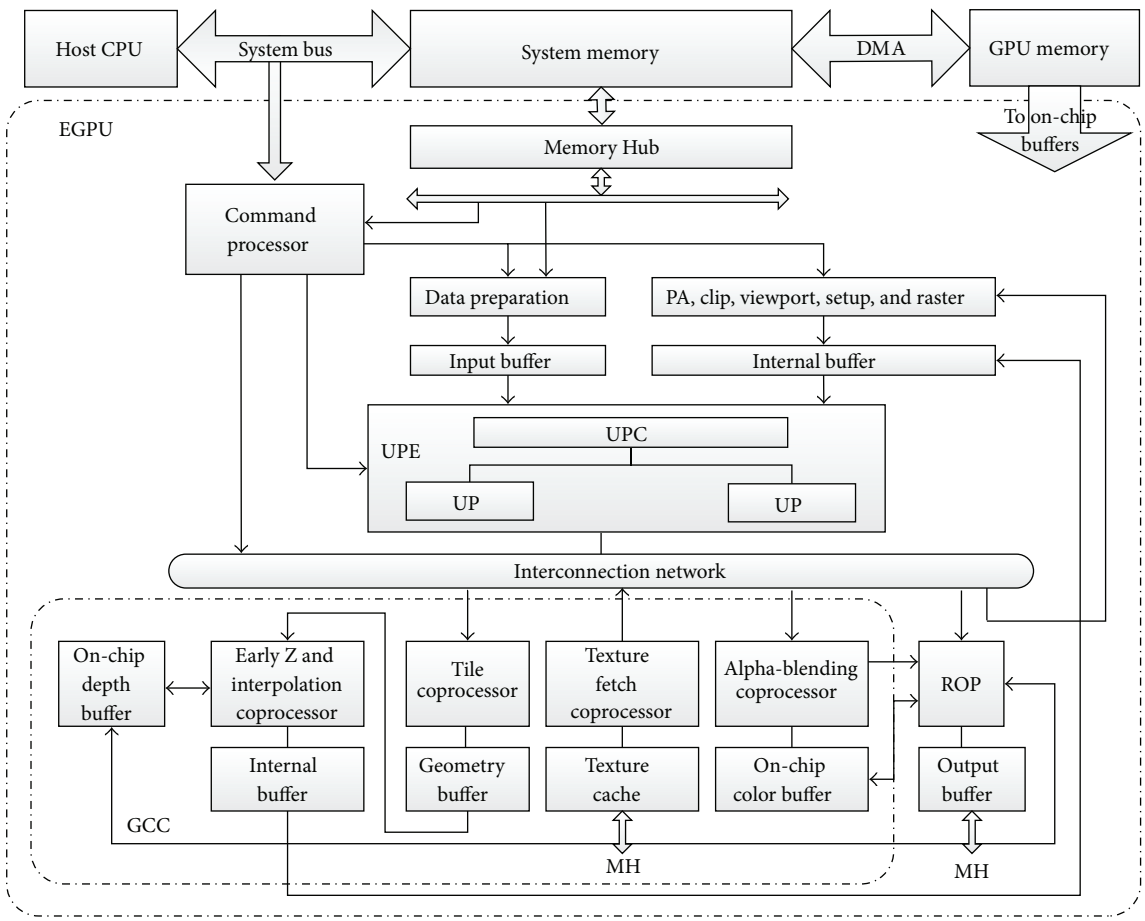

Figure 1: Architecture overview of the EGPU.

2.1. Universal Processing Engine. The UPE, as shown in Figure 2, is the processing core of the entire EGPU, undertaking the majority of the workload in both graphics processing and scientific computation. The UPE is designed as a parallel and hierarchical architecture, containing 2 unified UPs that work together under the management of the UPC. The UPC sends instructions and corresponding data from the input buffer or internal buffer to each UP, which executes vertex, geometry, and pixel shading programs and general computing programs. As the central control unit in the UPE, the UPC distributes various types of tasks to the UPs and balances the workload of each UP dynamically. Two independent UPs can work in parallel and improve the efficiency of the EGPU. In particular, in a tile-based EGPU architecture, each UP can process different tiles simultaneously in the fragment shading stage.

UP, based on a hierarchical architecture, is the basic and vital processing unit of the UPE. A UP is composed of an instruction pool (IP), a thread distributer (TD), a Hierarchical Processing Group (HPG), and internal memory. The TD reads instructions from the IP, which stores 32 instructions from the CP, and allocates them to the HPG. The HPG, including SPs and a Special Function Unit (SFU), is the execution unit of the UP. Once all data have been processed, they are written to the output buffer. General algorithms, logic, and memory access can be processed in the HPG, which supports 16 specific operations for each type. Most EGPU operations are covered.

The memory unit in each UP consists of a constant buffer, shared memory, and relative memory. The constant buffer stores constants used in the EGPU shading stage. The shared memory belongs to the HPG and is divided into 5 banks for the 4 SPs and 1 SFU. Intermediate results are written into shared memory for data exchange in the HPG. Relative memory stores data corresponding to the current shading stage. An anticonflict mechanism is proposed to ensure ordered memory access. SPs and the SFU write data to their own bank to avoid data conflicts. Their reading and writing access is also given specific priority to ensure that the correct data are accessed in order.

Instruction coissue is implemented in the UP with the help of the TD. Instructions are tagged from 1D to 4D according to their dimension. All processors in the HPG, SPs, and SFU are 1D scalar ones that can only address 32-bit floating operations. The dimension reflects the width of the data that are waiting to be processed. Thus, the operations on 64-bit, 96-bit, and 128-bit data are defined as 2D, 3D, and 4D operations, respectively. While 128-bit operations can fully utilize the 4 SP cores, other operations must be combined to occupy all SP cores simultaneously. The

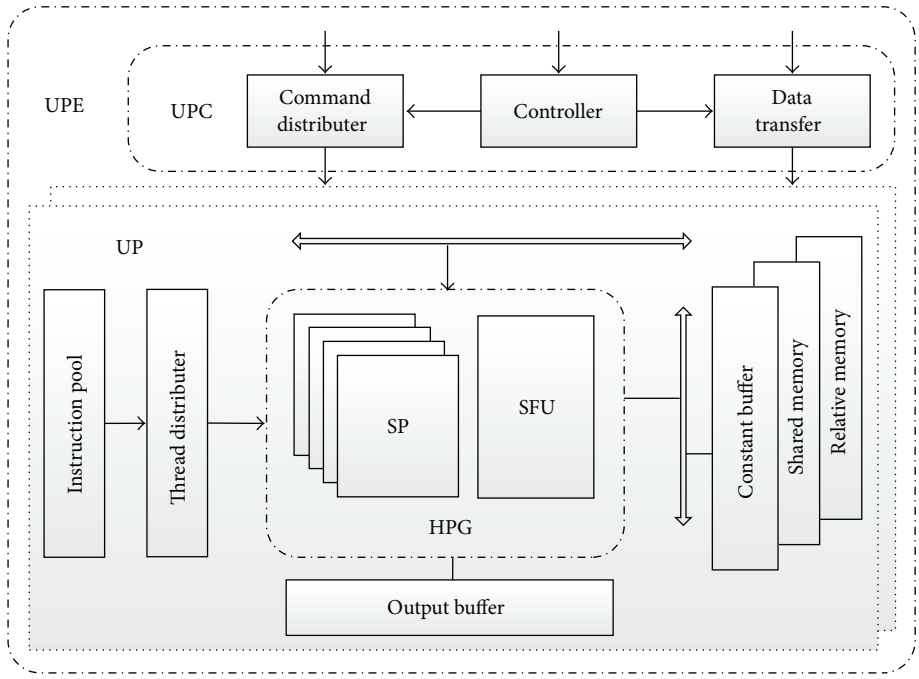

Figure 2: Architecture of the UPE.

combinations of 3D and 1D operations, 2D and 2D operations are used in this paper. For the most intricate occasion, the TD launches 4 independent 1D instructions simultaneously. With the coissue architecture, all the SP cores are occupied to make the instruction execution more efficient. Moreover, the TD distinguishes between basic operations and complex ones and distributes them to SPs and the SFU as different threads. By communicating with the HPG, the TD balances the workload dynamically. Additionally, a flexible memory access mechanism is designed to solve the memory access conflicts that arise with instruction coissue. Two AXI buses are valid for SPs, and another one is reserved for the SFU. Consequently, 4D, 3D and 1D, and 2D and 2D instructions can read corresponding data in one cycle. Four 1D instructions spend 2 cycles to finish a memory access.

Processing element (PE) and computing unit (CU) are the basic parallel processing units of the SP and SFU, respectively. The HPG is a hierarchical processing system. SPs and the SFU compose a scalable architecture to address basic and complex operations, respectively, which constitutes top-level parallelism. Additionally, the parallel processing among PEs and CUs is an underlying parallelism. Hierarchical parallelism makes it efficient to address those threads allocated by the TD.

The architecture of an SP and an SFU is shown in Figure 3. The SP and the SFU have similar architectures, and the SFU is more complicated to address those complex operations. The basic operations, such as adding, multiplying, and comparison, can be executed in one cycle. However, complex ones, such as trigonometric functions and logarithm, may need multiple cycles because of their complexities in accessing lookup tables (LUTs). Recognizing the fact that most operations are basic ones, the SFU is separated and shared by the SPs to reduce the consumption of area on the chip [14].

An LUT is designed to yield a complex calculation result by indexing a predefined array to reduce processing time because retrieving a value from memory is usually faster than undergoing a complex computation. The width of the input operand influences the hardware area and accuracy of the LUT. The input of an LUT in this paper is 6 bits in width, with a 3-bit function control. Functions implemented in an LUT include reciprocal, sine, cosine, exponent, and binary logarithm.

Considering that each SP or SFU can only obtain one memory bus, PEs and CUs work in a pipeline to ensure that only one PE or CU occupies the memory bus in each clock cycle. In each PE, operations are divided into 5 substages: instruction fetch (IF), instruction decode (ID), read data (RD), execution (EXE), and write data (WD). An instruction is first fetched from the instruction FIFO, and then it is decoded in the ID stage. After data are prepared in the RD stage, the instruction is executed in the EXE stage. Finally, corresponding data are written to memory in the WD stage. The general-purpose registers (GPR) array is the interface for PEs and CUs to interact with shared memory. A CU is designed with a similar pipeline. Three extra cycles are added to address the requests of complex operations, and these cycles are defined as EXE0, EXE1, EXE2, and EXE3. Compared with PE cores, which can only address basic operations, CU cores contain more algorithm and logic resources for complex operations.

2.2. Graphics Coprocessor Cluster. Graphics coprocessors are integrated into the EGPU system to accelerate

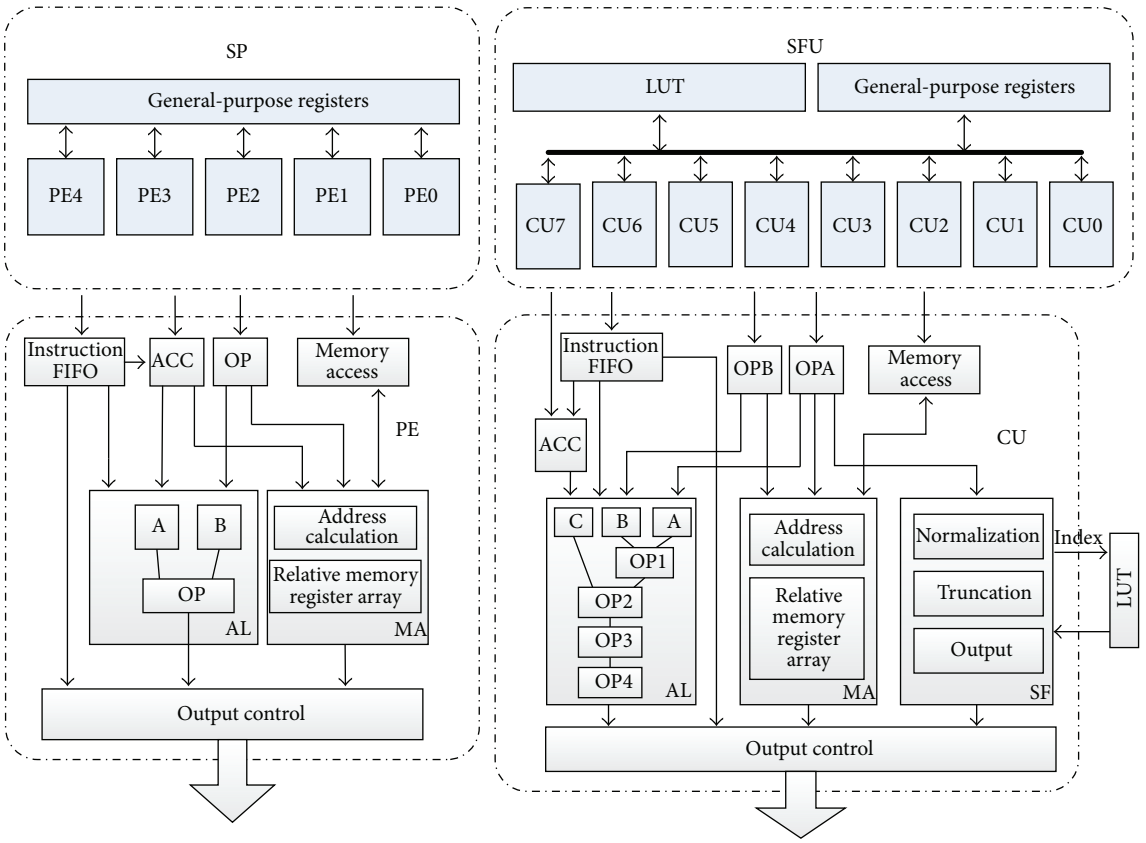

Figure 3: Architecture of an SP and SFU.

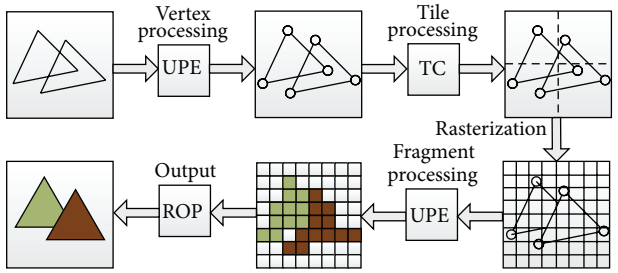

Figure 4: Processing flow with the tile architecture.

graphics-related calculations. In this design, four graphics coprocessors, including a tile coprocessor (TC), early Z and interpolation coprocessor (EZIC), texture fetch coprocessor (TFC), and alpha-blending coprocessor (ABC), are integrated to undertake specific graphics tasks. Various coprocessors are used in different modes under the control of the CP. Two modes are provided in this design, tile mode and blending mode. The first one is used for opaque scenes, as shown in Figure 4. The latter is applicable for transparent or blended scenes.

The TC, as shown in Figure 5, performs clip, project, and overlap test operations for geometric data that have been transformed by the UPE. In a unified shader-based system, the UPE executes vertex shading programs on geometric data for coordinates transformation and other per-vertex operations, such as lighting. Result data are then given to the TC. The TC divides the screen into several tiles, judges the relationships between triangles within the screen and tiles, and writes relative information to the tile list. A Bounding Box Test (BBT) is used to decide the overlap between triangles and tiles. As shown in Figure 5 [8], if the triangle is judged as belonging to a specific tile, information of that triangle is written into the corresponding position of the tile list. After all triangles have passed the BBT, the TC updates all the tiles covered by objects and writes out the transformed data to the geometry buffer.

The number of tiles required to complete the render is determined by the resolution of the tiles. Larger tile size does improve performance, leading to fewer tiles to process and fewer tile lists to update. However, it also causes an increase of the on-chip memory requirements in the graphics core. The choice of tile size is a balance between graphics processing performance and the cost of additional resources. In this paper, the size of each tile is fixed at 32×32 to handle different scenes.

After tiling, object data are tested and processed by the EZIC. In general, the TC and EZIC are enabled simultaneously. The EZIC, as shown in Figure 6, compares the calculated depth information of each fragment with the values stored in the on-chip tile depth buffer to determine if the current fragment is visible. Only those visible fragments

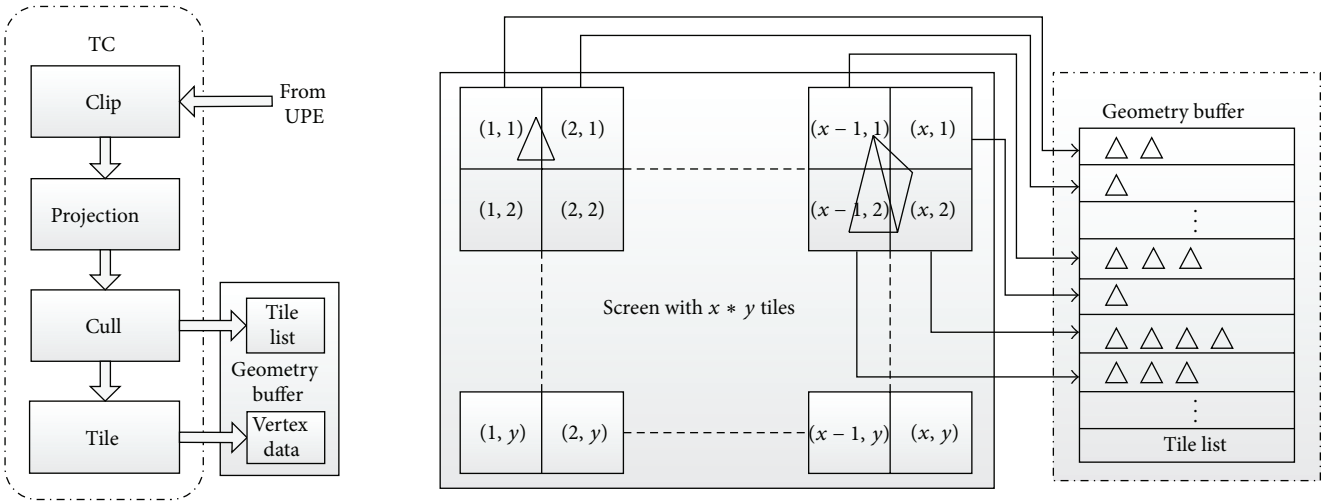

Figure 5: Tile mechanism. Left: architecture of the TC; right: TC tiling.

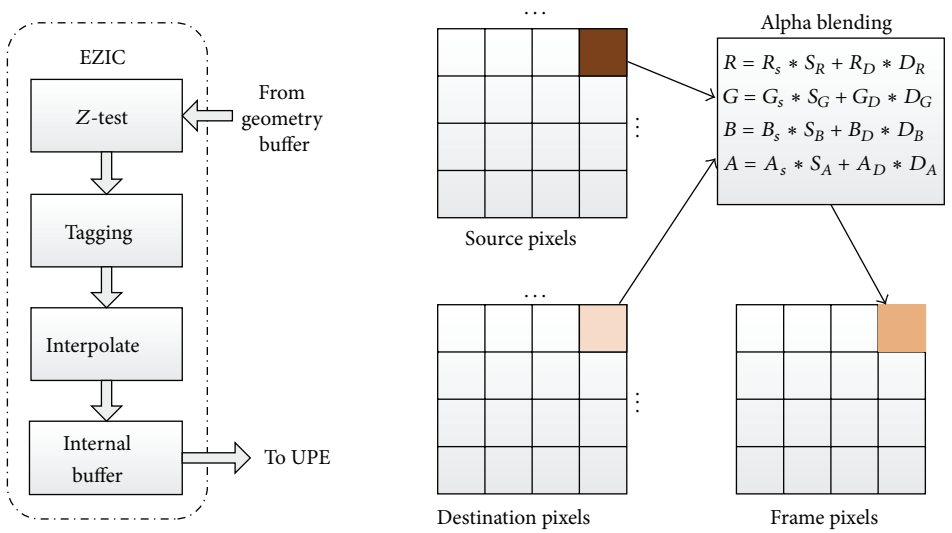

Figure 6: Architecture of the EZIC and alpha blending in the ABC.

are then interpolated and stored in the internal buffer. Fragments of different primitives are distinguished by tags to identify the primitive properties that should be used to texture the fragments later on. When all of the primitives in the tile list have been processed, the EZIC submits the remaining fragments back to the UPE for texturing and other operations. Additionally, the fragments belonging to the same primitive are organized in the same group with the help of their tags, to improve efficiency in memory access and fragment processing.

The TFC prefetches texture data based on calculated texture coordinates before the UPE begins to process fragments. The TFC calculates the corresponding address according to the texture coordinates and then sends reading requests to the MH. The prepared texture data are stored in the on-chip buffer for ease of access by the UPE.

The ABC is designed to draw transparent objects and implement graphics blending effects. To render blended objects accurately, the hardware has to process each object individually as they may all contribute to the frame buffer's color. Unfortunately, with the tile architecture, the depth test is performed earlier, and the obscured objects are all discarded. As a result, alpha and blending cannot be simultaneously realized. To overcome this drawback, the ABC method is proposed. When the current scene requires transparency or a blending effect, the CP halts the TC and EZIC. The entire system exits tile mode and enters blending mode. Then, the CP dispatches the rendering task to the UPE and ABC. The UPE completes the vertex and fragment stages without tiling acceleration. At the end of rendering, the ABC calculates the new color according to the blending coefficients and the original color in the color buffer and then replaces the original color in the frame buffer, as shown in Figure 6. The ABC is designed for transparent or translucent scenes, which require blending operations. All blending operations are performed by accessing the on-chip color buffer so that

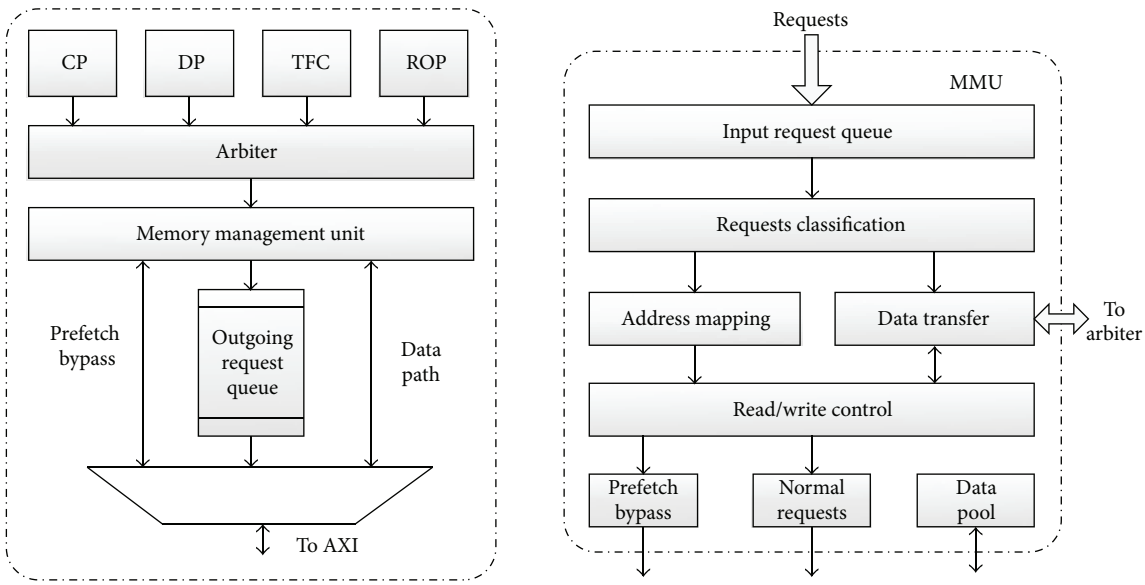

Figure 7: Architecture of the MH. Left: arbitration in the MH; right: architecture of the MMU.

they can be executed very quickly and not waste additional system memory bandwidth. The architecture of the EGPU with an ABC can adapt to increased types of scenes and obtain further flexibility.

2.3. Memory Hub. The MH provides arbitration for outgoing requests to system memory and also buffers data simultaneously. The MH functions as an arbiter to handle conflicts among related parts during memory access. As shown in Figure 7, the MH interacts with the following EGPU modules: the CP, DP, TFC, and ROP. The CP fetches instructions from system memory via the MH. The DP reads vertex data from the MH and then sends them to the input buffer, which consists of a list of buffers, including the vertex buffer, color buffer, and normal buffer. Additionally, all of these buffers are used to store the corresponding data. The TFC fetches textures via the MH, and the ROP writes data back to the system with the help of the MH. Moreover, the CP and DP support multiple read targets, which are identified with different master IDs. The MH supports a total of 4 unique read masters and 2 write masters. The MH responds to those requests and gives them permission to read or write. When several memory requests are sent to the MH simultaneously, the MH gives turn-around access to one of these blocks based on a predefined priority. Then, the memory bus will be occupied and busy. The memory management unit (MMU) will map the reading or writing address to the input address and number, and the following requests will enter the queue to wait for a response. Only when the memory bus is released by the current client will the MH continue to address the remaining outstanding requests. The fixed priority is designed according to the following order of graphics processing for increased efficiency: CP → DP → TFC → ROP.

The MMU handles the reading and writing requests of the MH, as shown in Figure 7. Because different requests have been arbitrated, the MMU responds to the current request, calculates the memory address, and establishes the data path. Requests are firstly classified. Then, their access addresses are calculated according to their index and number. A prefetching bypass mechanism is designed in this paper, ensuring that high priority or emergency requests can be responded to in time. The order of reading and writing requests is controlled to avoid potential conflicts. Data are transferred through the data transfer block.

3. Experimental Results

In this section, several experimental results are provided to evaluate the performance of the EGPU system.

A system verification platform is established to verify the performance of the EGPU design proposed in this paper. A 3D graphics scene has been developed by transferring related data to a test case manually to meet the requirements of the EGPU's input. This graphics scene includes approximately 1000 vertices and 800 triangles. No point and line primitives are included. As a result, the EGPU processes the data successfully and produces a 2D image in the frame buffer.

The speed of the processor is regarded as one of the most important evaluation factors. The clock cycles used by each instruction can reflect the speed directly. Moreover, the hardware design cost is also considered by means of the logic design area and power consumption. The balance between the performance and hardware cost has been studied during the design of this work. For the proposed UP, it can enhance the speed of processing at a low hardware resource cost. With a frequency of 200 MHz, an average of approximately 50 million vertices or fragments can be processed per second

Table 1: Performance analysis of a UP.

Case	Instruction package type: Number		Clock cycles used	Execution unit
1	1D ∗ 4:	X	$X + 5$	SPs
2	1D + 3D:	X	$X + 4$	SPs
3	2D + 2D:	X	$X + 4$	SPs
4	4D:	X	$X + 4$	SPs
5	Complex operation:	X	$X + 7$	SFU

Table 2: Performance analysis of other modules.

Target: number	Processing time (Clock cycles)	Processing module
Triangles: X	$[X, 7X]$	Clipper
Primitive: X	$[X, 3X]$	PA
Primitive: X	Depends on the size of the primitives	TC

Table 3: Technology condition and results.

Processing technology	Voltage	Temperature
0.18 μm CMOS	1.68 V	125°C
Area	Power consumption	Frequency
6.5 mm ∗ 6.5 mm	349.318 mW	200 MHz

on each UP. Table 1 presents the processing ability of a UP. Take case 1 as an example. For four 1D instructions packed together, the SPs take $X + 5$ cycles to process X packages.

In Table 2, a series of design indicators of corresponding modules are presented. Taking triangles as an example, the processing time is between X and $7X$ clock cycles to process X triangles. For some operations, the clock cycle consumption depends on objects' sizes, positions, and also attributes in the 3D application. For example, triangles are usually of different sizes, which indicate the different numbers of fragments they contain, and this will eventually lead to difficulty in evaluating the clock cycle consumption. Moreover, in the Clipper, primitives will firstly be classified into three types, and operations will be different according to these types. The first type is the primitive that is totally inside the view frustum, and such a primitive will not be clipped. The second type is the primitive that is totally outside the view frustum, and such a primitive will be discarded. The final type is the primitive that is partially inside the view frustum, and such a primitive will be clipped. Different types have different associated clock cycle consumptions.

Now that the clock cycles used in tiling are dependent on the size of primitives and the overlap conditions between the primitives and tiles, the performance of the TC is measured by the number of triangles tiled per second. With a frequency of 200 MHz, the TC can process 5.7 million triangles per second in the worst case and 18.4 million in the best case.

Moreover, a 0.18 μm CMOS technology library is applied to analyze the timing and power consumption in the worst case. The results of the PVT corner, area, and power are demonstrated in Table 3.

Comparisons have been made as shown in Table 4. The performance of the UPE in this work is compared with that of other designs. The area and energy performance have been improved by varying degrees as compared with conventional works. The UPE also provides a floating-point capacity of 1 GFLOPS.

4. Conclusion

In this paper, a tile-based EGPU is proposed with fused, scalable, and hierarchical parallelism architecture. A UPE is proposed to improve the computing efficiency via SIMT architecture, a coissue mechanism, and high parallelism. Different modes are provided to meet the demands of different scenes with the help of the GCC, leading to a wider range of adaptability. Additionally, the tile-based mechanism dramatically reduces the consumption of memory bandwidth by tiling the screen and using on-chip buffers. The design of the MH avoids memory access conflicts efficiently. With the innovation of the architecture and processing mechanism, the proposed EGPU has achieved a proper balance between performance and hardware costs. The UPE has the ability to address nearly 100 million vertices or fragments per second at a 200 MHz clock frequency. Additionally, the TC can process 18.4 million triangles in its best case in the tiling mode. Additionally, a 0.18 μm technology library is used for timing and power analysis. The area of the EGPU is approximately 6.5 mm ∗ 6.5 mm, and the power consumption is approximately 349.318 mW. The entire EGPU can complete graphics processing tasks successfully. Moreover, it is possible to further improve the current design. For example, the TC cannot operate at full speed because of the speed limitations of the Clipper and the PA. Simultaneously, the GCC can also be improved to match the UPE in performance and processing quality. The next stage of research will be focused on ways to enhance the efficiency of the UPE via a more advanced architecture.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Natural Science Foundation of Shandong Province (ZR2013FQ006), the Key Innovation Project of Shandong Province (2013CXB3020),

Table 4: Compared results and conditions.

Architecture	[11]	[12]	[13]	This work
Feature	Vertex processor	Unified shader	Graphics processor	UPE
Acceleration range	Geometry	Geometry, rendering	Geometry, rendering	GP-GPU, Geometry, rendering
Performance	186 M vertices/s	9.1 M vertices/s 100 M fragments/s	50 M vertices/s 50 M fragments/s	1 GFLOPS 100 M vertices/s 100 M fragments/s
Processing technology	0.18 μm CMOS	0.13 μm CMOS	0.18 μm CMOS	0.18 μm CMOS
Frequency	110 MHz	100 MHz	200 MHz	200 MHz
Power consumption	160.7 mW	195 mW	155 mW	198.228 mW
Area	4 mm $*$ 4.8 mm	3.3 mm $*$ 3 mm	22 mm^2	~27 mm^2

the State Key Laboratory of Digital Multimedia Technology (2013-1-2569), Shandong Post-Doctor Innovation Foundation Grant (201002029), Grant of the China Post-Doctor Innovation Foundation (20110491601), and National Natural Science Foundation of China (NSFC) Grant (61302063). The authors would like to thank all research partners for their significant contributions to this work. The authors also thank University of Jinan for their support of the hardware platform.

References

[1] L. Garber, "GPUs go mobile," *Computer*, vol. 46, no. 2, Article ID 6457381, pp. 16–19, 2013.

[2] Imagination Technologies, *PowerVR MBX Technology Overview*, Revision 1, Imagination Technologies, 2009.

[3] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011.

[4] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[5] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[6] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: a unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[7] B.-G. Nam, H. Kim, and H.-J. Yoo, "A low-power unified arithmetic unit for programmable handheld 3-D graphics systems," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 8, pp. 1767–1778, 2007.

[8] Imagination Technologies, *POWERVR Series5 Graphics SGX Architecture Guide for Developers*, Version 1.0.8, Imagination Technologies, 2011.

[9] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley, New York, NY, USA, 2005.

[10] H.-Y. Kim, Y.-J. Kim, J.-H. Oh, and L.-S. Kim, "A reconfigurable SIMT processor for mobile ray tracing with contention reduction in shared memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 4, pp. 938–950, 2013.

[11] C.-H. Yu, K. Chung, D. Kim, S.-H. Kim, and L.-S. Kim, "A 186-Mvertices/s 161-mW floating-point vertex processor with optimized datapath and vertex caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 10, pp. 1369–1382, 2009.

[12] J. H. Woo, J. H. Sohn, H. Kim, and H. J. Yoo, "A 195 mW/152 mW mobile multimedia SoC with fully programmable 3-D graphics and MPEG4/H.264/JPEG," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 9, pp. 2047–2056, 2008.

[13] J.-H. Sohn, J.-H. Woo, M.-W. Lee, H.-J. Kim, R. Woo, and H.-J. Yoo, "A 155-mW 50-m vertices/s graphics processor with fixed-point programmable vertex shader for mobile applications," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 5, pp. 1081–1091, 2006.

[14] Y.-J. Kim, H.-E. Kim, S.-H. Kim, J.-S. Park, S. Paek, and L.-S. Kim, "Homogeneous stream processors with embedded special function units for high-utilization programmable shaders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1691–1704, 2012.