Unified Rendering Array Driver Runtime Optimization for Graphic Application

Chengcheng Ma1,2, Ze Tian1,2, Honghong Liu3 and Zhao Nie1,2

1. China Aeronautical Computing Research Institute
2. Aviation key Laboratory of Science and Technology on integrated Circuit and Micro-System Design, P.O. Box 710068, China
3. Xiangteng microelectronics technology co. LTD, P.O. Box 710068, China
Email: jiujuecheng@163.com; tarmz@126.com; xyzhongly@163.com; kaykaykay333@163.com

Abstract. As the core of GPU processing, the performance of Unified rendering array is directly affected by the shader driver. On the basis of research the structure of Unified rendering array processing, this paper proposes a runtime Unified rendering array driver optimization method for graphical application, can according to different graphics application scenario runtime to generate optimized Unified rendering array driver, testing on graphics processors FPGA verification environment, driven by the runtime optimization compared with the traditional full function driver, performance increased by 55% - 270%.

1. Introduction
As the core and key of visualization equipment, GPU is widely used in many fields such as audio-visual entertainment, industrial display and medical imaging. With the continuous development of graphics application, there is an urgent need for realistic sense and real-time graphics drawing ability, and higher requirements are put forward for the function and performance of GPU. Most modern GPU are actualized by large-scale programmable rendering array. As the "soul" of rendering array, the quality of the staining device directly determines the function and performance of the graphics processor. It is of great significance to study the development and application of graphics processor by studying the optimization of rendering array driver.

At present, the research on shader performance optimization has focused on the optimization of compiler and link environment, instruction scheduling and task allocation optimization, high-performance programming model and so on. For example, in the study of cpu-gpu heterogeneous parallel system programming model and compilation optimization, Mr. Tang et al. analyzed the CPU-GPU running software from the aspects of storage level, program locality, cache invalidation and so on, and proposes an openstream compiling framework for stream level optimization for shader software; The GPGPU quantization performance model for OpenGL architecture was built by Zhu Junfeng et al. to establish an abstract GPU architecture and execution model. The running structure of the program was analyzed in terms of GPGPU program performance factors, computational overhead, all the above optimizations are for application level static development optimization or compilation optimization for specific staining structure, although the application software can be well combined with the staining array structure to a certain extent, it cannot adjust
the staining device driver in real time according to the application characteristics, optimize the operation of the staining device, and will be affected by the GPU driving software.

This paper proposes a method for the runtime optimization for Graphic function shader, According to the graphics application software running the required shader array configuration information, dynamic cutting unused functions in the application software is running, optimization of branch control, avoid unnecessary operations, optimize software arrangement and so on, to improve the performance of shader runtime.

2. Shader Working Principle

Since 2001, NVIDIA launched Geforce3 for the first time since the concept of programmable shader, programmable shader has become the most popular GPU implementation, academia and industry have done a lot of research, the most representative GPU leader is the NVIDIA and AMD. R700 is released with a large scale SIMD shader array of graphics processor architecture by AMD company in 2008, the structure shown in Figure 1. The storage system consists of two parts, one part is in the system memory space, the other part is located on the GPU memory space. The data contained in the storage system consists of command queues, instructions constants, and input and output streams, the command queues specifies the tasks that the GPU needs to handle, the instructions give the details of the execution units, constants, input and output stream calculation of the required data, these elements constitute the basic elements of GPU operation.

A complete R700 application is divided into two parts, the host program running on the main processor and the kernel program running on the GPU DPP(data parallel processor) array. When the application in the running the host program generates the data required for the shader array, the shader program, and the executed commands, and controls the shader program's operation.
NVIDIA’s open product information on the shader array operation is not found on the instructions, but NVIDIA released CUDA (computer unified device architecture) system structure defines the GPU internal parallel processing unit running. As shown in Figure 2, GPU is seen as a host CPU with a large number of parallel computing power of the coprocessor. Thread is the most basic unit of parallel computing. Application in the process, host CPU will be sent to the compute-intensive part of the GPU for processing. That is to generate the shader array required procedures, data and task commands, and control its operation.

Thus, GPU internal programmable shader array running entirely by the host control, include the shader array required procedures, data, task commands, and operation control and so on. Graphics processing is a typical application of a programmable shader array, support is provided to the user through a graphical interface, such as OpenGL, Direct 3D, whereby the programmable shader array processing task is generated by the graphical interface.

![CUDA system structure diagram](image)
In a typical graphics processor, the shader array mainly performs vertex processing and pixel processing. For example, R700 uses shader array to process the coordinate transformation and light processing of vertex stage, processing texture and fog of pixel stage; The 3D graphics processor for mobile devices developed by Yang Yi and others from the China University of Science and Technology, uses programmable shader array to process the coordinate transformation and light processing of vertex stage. The shader software as a direct control of the shader array, complete the specific graphics processing algorithms. There are a large number of processing branches in the graphics algorithm, as shown in Figure 3, Typical Texture processing includes texture enable judgment, texture type processing, texture coordinate wrap mode processing, sampling mode processing, texture environment processing, texture multiplicity judgment and so on, and each process contains a variety of texture types. However, in the specific graphics rendering will only use one or a few types of processing. The traditional full-function shader requires all branches to be processed, Process control takes a lot of processing, seriously affecting the driver software performance.

In this paper, for the graphics interface portion of the graphics processor host-driven, add shader configurations that are included in graphics rendering, run shader management collections, generates an optimized driver for the current drawing operation when the shader array needs to be
run, improve graphics rendering capabilities.

3. **The Design of the Runtime Optimization Shader**

The design of the runtime optimization shader is based on the shader producer-consumer work, the graphic application generates parameter information, data information and runtime manage information that requires the processing of the shader array during operation. Before the shader runtime management does not start the shader, the data information has not been processed by the shader array, and the parameter information is only the makeup state, and is not really involved in the shader calculation. So the shader driver generation can be delayed, when the shader is really needed for processing in the graphics application running, it generates the optimal driver for this shader.

The runtime optimization shader architecture shown in Figure 4, include graphical application shader runtime configuration extraction and shader driver automatically generated two parts. The graphical application shader runtime configuration extraction is responsible for statistics and records the configuration information associated with shader runtime during application execution; The shader driver automatically generated according to the application configuration information extracted automatically generate optimized shader driver software.

![Figure 4: Shader intelligent optimization tool architecture.](image)

During the shader intelligent optimization runtime, the graphical application shader runtime configuration extraction unit first records and statistics of graphics applications referenced by the graphical interface contains configuration information, The configuration information includes configuration parameters, configuration parameter runtime frequency, and the order of configuration parameters. Start shader runtime configuration generation when the shader operation management unit receives the shader operation data quantity satisfies the certain condition or the shader is forced to execute, and decide whether the current shader configuration needs to reoptimize the shader driver, if necessary, the shader runtime software selection unit chooses the necessary blocks from the shader function software library according to the configuration parameters in the shader runtime configuration information, the shader runtime software arrangement unit optimizes, combines, and arranges the blocks according to the runtime frequency and action order information in the shader runtime configuration information, generates the shader driver for the current configuration.

A detailed description of the shader runtime configuration extraction and the shader driver automatically generated is given below.

### 3.1 The design of the graphic application shader runtime configuration extraction

Graphics applications on the shader reference are directly through the graphical interface driver to
complete, graphical interface for the shader control can be divided into four categories by the graphical interface functions and shader runtime tasks.

(1) shader irrelevant processing: this part of the graphics processing interface function in the graphics processing without shader participation in the processing;
(2) shader runtime parameters: the parameters included in this part of the graphics processing interface function directly affect the processing of the shader, such as lighting enable, texture enable, texture coordinate wraparound, texture sampling, etc., but do not cause the shader to run;
(3) shader runtime data: this part of the graphics processing interface function to provide data for the shader, the shader processing data in the graphics process, including vertex data and fragment data;
(4) shader runtime control: this part of the graphics processing interface function will directly or indirectly trigger the shader run, such as glFinish, glFlush in the graphical interface.

The design of the graphic application shader runtime configuration extraction shown in Figure 4, shader runtime configuration has two trigger conditions, shader runtime data volume satisfies the certain condition and the shader is forced to execute. Shader runtime data volume satisfies the certain condition refers to the time required for the shader array to process the shader runtime data in the current shader driver situation than the time replaces the shader driver and processes all the shader runtime data. Suppose that the time of using the optimized shader to process data volume is accelerated to T as compared with the existing shader, T is calculated as shown in Equation 1:

\[
T = \left(\sum T_o - \sum T_n\right) \times \frac{C_d}{C_w} - T_c
\]

\[
= TN \times \frac{C_d}{C_w} - T_c
\]

To: the processing time required to process an existing shader driver configuration;
Tn: the processing time required to process an optimized shader driver configuration;
Cw: the maximum amount of tasks that can be handled at the same time by shader array;
Cd: the total number of tasks this time;
Tc: the time required to generate and load the shader driver to the runnable state.

When \( T > 0 \) indicates that the data processing needs to generate runtime configuration optimization shader driver, When \( T < 0 \) indicates that this configuration does not require shader generation.

Shader forced runs are directly generate the shader runtime configuration information, after the shader run time configuration is generated, shader runtime parameters information will be emptied and then wait for the next processing, the generated shader runtime configuration is empty or the same as the previous configuration, this operation is not required shader processing and the driver is not generated, if not empty or not the same as the previous configuration, then start shader driver generation.

### 3.2 The shader driver automatically generated

The shader driver automatically generated optimized, generates, loads shader driver based on the shader runtime configuration, its operation is based on the shader runtime configuration and the shader function software library; the shader runtime configuration information is the bridge between graphics applications and shader drivers; the shader function software library is the basic element of the shader driver software. Shader runtime software selection and shader runtime software arrangement extract the necessary runtime software in the shader function library and compiles the executable file based on the shader runtime configuration, the following describes the shader function software library construction, shader software selection and arrangement process.
The shader function software library construction. In this paper, the design of the shader function software executable binary or assembly file as a software file, attach the administrative information for the executable, the management information includes function flags and performance parameters.

There are two methods of the shader function software library construction:

Method one: according to the different configuration independent development of driver software, the design of the driver software in the particular shader configuration can achieve optimal performance, but the number of shader configuration types is huge, as shown in Figure 3 of the texture processing, there are $A^3 \times A^2 \times A^2 \times A$ shader configurations, cannot achieve.

Method two: the driver is divided into a small functional software segment, according to the different configuration assembly software segment to complete the overall driver software, this method enables shader driving with a small number of combination, but the processing performance is inferior to the method one outstanding.

In this paper, a combination of two methods to achieve the driver software library, through the statistical application of the current understanding of the commonly used scene configuration, as shown in Table 1, the current application areas is commonly used texture configuration statistics, using the method one for the development of the dedicated driver, improve processing performance, the rest are implemented using method two.

| Application | Texture Target | Level | Data Type | Texture Format | Texture Wrap | Texture Filter | Texture Env |
|-------------|----------------|-------|-----------|----------------|--------------|---------------|-------------|
| scene 1     | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: REPEAT  | MAG,MN: LINEAR |             |
| scene 2     | 2D             | 1     | UNSIGNE D_BYTE | LUMINANCE E    | S.T: REPEAT  | MAG,MN: LINEAR |             |
| scene 3     | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: REPEAT  | MAG,MN: LINEAR |             |
| scene 4     | 2D             | 1     | UNSIGNE D_BYTE | LUMINANCE E_ALPHA | S.T: CLAMP TO EDGE | REPLACE  |
| scene 5     | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: CLAMP TO EDGE | MODULATE REPLACE |
| scene 6     | 2D,CU          | 1     | UNSIGNE D_BYTE | RGBA           | S.T: CLAMP TO EDGE | MODULATE REPLACE |
| scene 7     | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: REPEAT  | MAG,MN: LINEAR |             |
| scene 8     | 2D,CU          | 1     | UNSIGNE D_BYTE | RGBA           | S.T: CLAMP TO EDGE | MODULATE REPLACE |
| scene 9     | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: REPEAT  | MAG,MN: LINEAR |             |
| scene 10    | 2D             | 1     | UNSIGNE D_BYTE | RGB            | S.T: REPEAT  | MAG,MN: LINEAR |             |

The implementation of method two is divided into three steps:

Step one: divide the function blocks, build the function software composition framework

For the implementation process of the full-featured software, it is divided into several atomic blocks, atomic block reference has a certain order of relationship, this sequence of relationship is the composition of functional software framework;

Step two: set the function block interface

Extract the minimum functional set of the atomic block, statistic the position of the minimum feature set in the functional software composition framework, the storage resources are distributed to each atomic block in an orderly manner;

Step three: optimization function block and build function software library

To achieve the optimization of functional blocks, and generate executable files and administrative information, component function software library.

Take the texture type processing shown in Figure 3 as an example, first, the process is divided into atomic lock, as shown in Figure 5, the composition of the software frame order for the atomic

3.2.1 The shader function software library construction. In this paper, the design of the shader function software executable binary or assembly file as a software file, attach the administrative information for the executable, the management information includes function flags and performance parameters.
bock 1, 2, 3, 4, 5, 6, 7, 8. The atomic block 1, 3, 5, 7 are marked to judge, the contents of the remaining atomic blocks are different, therefore, the minimum functional set of the atomic software segment generated by the texture type processing is the atomic block 1, 2, 4, 6, 8 and the output of the atomic block 1 is the input of the atomic block 2, 4, 6, 8. The output of the atomic block 1 is marked to the texture type processing, Bit0 indicates cube map, Bit1 indicates 3D, Bit2 indicates 2D, Bit3 indicates 1D.

3.2.2 The shader runtime software selection and arrangement. The shader runtime software selection and arrangement generates an optimized shader driver based on the shader runtime configuration information in conjunction with the shader function software library. In the shader driver generation process, first of all, match in the method one described in section 4.1 of the function library build, if successful, the shader driver is generated directly; otherwise select the appropriate atomic block in the method two described of the function library build, then optimize, arrange and generate the shader driver that is current running optimally.
The selection and arrangement of atomic blocks in the function library build in the method two follows the following rules:

1. Keep the main frame of software running
   As shown in Figure 3, the texture processing is followed by texture enable judgment, texture type processing, texture coordinate wrap mode processing, sampling mode processing, texture environment processing, texture multiplicity judgment;

2. In a plurality of parallel processing, only one shader runtime configuration information is selected, it can be directly processed without judgment
   As shown in Figure 5, only matches 2D in the texture type processing, only atomic block 6 is selected;

3. In multiple parallel processing, there are multiple staining devices running configuration information and the operation frequency is inconsistent, then the staining device should be arranged according to the operation frequency of each configuration. The configuration with high operation frequency should ensure the minimum operation matching cost.
   As shown in Figure 5, if the texture type process includes 2D and 1D, and 1D runs 2000 times and 2D runs 3000, only atomic program segments 1, 6 and 8 are selected, and the final generation program is atomic program segments 1, 6, 1 and 8.

4. In the multiple parallel processing, there are various operation configuration information of the steller, and the running frequency is consistent. Then, it is arranged according to the order of calls to improve the cache hit rate of the software running instruction.
   As shown in Figure 5, if the texture type processing includes 2D, 1D, 1D and 2D operations for 2000 times, and 1D runs before 2D, only atomic program segments 1, 6 and 8 are selected, and the final generation program is atomic program segments 1, 8, 1 and 6. When 1D is first run during operation, instruction cache may also load 2D processed software, thus improving the efficiency of 2D run time.

4. The Test Contrast
The method described herein is implemented in our own research and development of the GPU host drive, our own research and development of the GPU shader array consists of four vertex shaders and eight pixel shaders. On the verification platform consisting of the 400 MHZ MPC8270 host and the 50 MHZ GPU of our own research core based on FPGA, for the airborne comprehend commonly used in the five representative of the common scene to test, verification results and the realization of the mapping effect is completely correct, the average drawing frame rate for full-function shader driver operations, the average drawing frame rate for optimized shader driver operations and The number of shader drive software replacement as shown in Table 2.

Airborne scene 1, 2, 3, vertex processing is dominant, a small number of texture map in the segments, model view transformation that requires shader processing with less computation of vertex, the drawing process uses the same shader configuration, the shader processing is less work, so performance speed ratio is relatively low.

Airborne scene 3, the shader is required for complex processing, but using a single graphic configuration, and no shader driver switching, performance speed ratio is relatively high.

Airborne scene 3 and common scene 4, 5, vertex stage, pixel stage are carried out a large number of complex calculations, different shader configuration are used to draw different elements, shader driver switching has some effect on the performance speedup.

Test results show that, after optimization of the shader driver processing performance generally improved more than 50%, the most significant acceleration is achieved with the same shader and the need for a shader to handle a large number of complex situations. the acceleration that scene that require a small amount of shader operations is general. there are shader driver switching scenarios, shader driver switching causes a certain consumption, affect the acceleration effect.

| Table 2. The results of the scenario test |
|------------------------------------------|
| Airborne scene 1 | Airborne scene 2 | Airborne scene 3 | Airborne scene 4 | Airborne scene 5 |
|------------------|------------------|------------------|------------------|------------------|

9
Conclusions

In this paper, based on the study of shader working principle, proposes a method for the runtime optimization for shader software, can generated the runtime-optimized shader for different application scenarios, improves the performance of graphics processing. Successful application in our own research and development of the GPU, through to the scenes within the airborne field and the public scenes within the general test, performance is generally improved by more than 50%. However, the current research is only for the software graphical interface driver optimization, and the hardware aspects cannot be fully optimized. In the future, the software and hardware implementation of rendering array will be studied from the perspective of the system to further improve the processing performance.

References

[1] Liu Xiangdong, Research and application of high-density crowd counting method, Zhejiang: Zhejiang University of Technology, 2012.
[2] Bay Herbert, Ess A, Tuytelaars T, et al. SURF: speeded up robust features, Computer Vision and Image Understanding, 2008, pp. 346-359.
[3] Wang Yalin, Research on Algorithm of Crowd Density Estimation Based on Gray Level Co-occurrence Matrix, Xi-an: Xi-an University of Technology and Science, 2013.
[4] Proceedings of Eleventh IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS2009), IEEE Computer Science, 2009, pp. 1-8.
[5] Conte D, Foggia P, Percannella G, et al. A Method for Counting Moving People in Video Surveillance Videos, EURASIP Journal on Advances in Signal Processing, 2010, pp. 1-8.
[6] Ya Lihou, Pang G KH. People Counting and Human Detection in a Challenging Situation, IEEE Transactions on Systems, Man and Cybernetics (Part A: Systems and Humans), 2011, pp. 24 - 33.
[7] Ryan D, Denman S, Fookes C, et al. Crowd Counting Using Group Tracking and Local Features, 7th IEEE International Conference on Advanced Video and Signal Based Surveillance, 2010, pp. 218 - 224.
[8] Antoni B, Chan, Nuno Vasconcelos. Counting People With Low-Level Features and Bayesian Regression, IEEE Transactions on Image Processing, 2012, pp. 21.
[9] Ya Huang, Su Hang, Zheng Shibao. Large-scale Crowd Density Estimation, Video Application and Project, 2010, pp. 113-116.
[10] Sun Wenchang, Song Jianshe, Yang Meng, et al. Corner Detection Algorithm Based on Entropy and Uniqueness, Journal of Computer Applications, 2009, 29(S2): 225-227.
[11] X. Liu, B. Dai, H. G. He. Real-time object segmentation for visual object detection in dynamic scenes, Proceedings of International Conference of soft Computing and Pattern Recognition, 2011, 423-428.
[12] K. Zhang, T. F. Xu, P. Wang, L. Feng. Real-time full-frame digital image stabilization system By SURF, Optics and Precision Engineering, Vol. 19, No. 8, 2011, pp. 1964-1972.