Research on Multi-CPU and Multi-GPU Scalable Parallel Rendering on Shared Memory Architecture

Huahai Liu, Pan Wang, Sikun Li ,Xun Cai, Liang Zeng
National University Of Defense Technology

***Abstract*-As the performance-price ratio of a GPU becoming high, lots of systems may have more than one GPU in a node. Every GPU in a node has the strong ability to render, and it is very important to effectively organize the parallel rendering pipeline to fully exploit the compute units of the system. However lots of parallel rendering systems couple the hardware rendering stage with the composition stage in the display thread and this will make GPUs stall. In this paper, we describe a parallel rendering approach which enables the two stages to execute in parallel. With the frame buffer in the main memory, the full image rendering time totally lies on the GPU rendering ability when the rendering task is large enough. The experiments show that our method performance are much better than that of the existing methods. And we also test the scalable ability and get a linear performance speedup with the GPU number when the rendering task is large enough. Any parallel rendering application can benefit from the decoupled parallel rendering approach.**

Keywords-Multi-GPU, Parallel Rendering, Scalable Rendering, Composition,

I. INTRODUCTION

Parallel rendering systems are essential to cope with the rapid growth of data sets to achieve interactive visualization. As the performance-price ratio of a GPU becoming high, lots of systems may be able to have more than one GPU in the node. Many research and application show that it is able to render the complex model with a single multi-GPU node by improving the system performance and the multi-GPU node is a great foundational 'building block' to compose the larger systems capable of rendering very large data [1]. Every GPU in the node can afford a powerful rendering ability, and it is very important to effectively organize the parallel rendering pipeline to fully exploit the compute units of the system. Because the input of the composite stage depends on the rendering stage output, lots of parallel rendering systems couple the two stages together in the display thread. So a full image rendering time becomes the sum of the rendering time and the composite time. The execution in serial makes the GPUs stall to wait for each other and descents the system performance.

In this paper, we introduce an efficient pipeline method to decouple the two stages and enable them execute in parallel. First, we separate rendering thread from the application logical event thread with multithread method. And we can organize the system's rendering resources conveniently by creating multiple threads. Second, we decouple the hardware rendering thread from the composite thread with a hybrid rendering method, which use both hardware-based rendering and software-based rendering. Then the rendering stage and composite stage can execute in parallel by creating a frame buffer in main memory. With overlapping the composite time, a full image rendering time can be only determined by the rendering time for large complex rendering task.

II. RELATED WORK

Current vendors support for multi-GPU configurations includes NVIDIA's SLI [2] and AMD's Crossfire [3]. Both of them are the sort-first approach in a node and have two primary rendering modes. In Split Frame Rendering (SFR) mode, every GPU renders a disjoint portion of the screen frame. In Alternate Frame Rendering (AFR), all the GPUs work in parallel on different consecutively assigned frames. Mark el. introduce a novel SFR-AFR combined mode and get a better performance [4]. All of these technologies make multiple GPUs look like a single entity in the logical and are highly optimized for game applications. This can simplify the programming model, but it is not allowed to run a different command stream on different GPU. NVIDIA's Tesla system can also be used to make up of multi-GPU workstation. For lacking of DVI outing port, it is not fit for compositing of large parallel rendering system.

Hanrahan el. design a graphics system named WireGL, which is driven by multiple simultaneous streams of graphics command in a node or cluster [5]. Igehy el. propose a method for creating barriers and semaphores that operates at the graphics context level by extending existing ideas found in OpenGL and X11 that allow multiple graphics contexts to simultaneously draw into the same image [5]. And later they integrate the parallel rendering interface into the WireGL. Chromium developed from WireGL supports both sort-first and sort-last approaches [7]. Both of the systems send the OpenGL command over interconnect network to the destination GPU though an OpenGL API abstract layer. When the static geometry or texture data is holding in the texture memory, the OpenGL command stream is only composed of rendering command and the system performance is good. But for the dynamic data, the OpenGL command stream contains not only rendering command but also rendering data. And the interconnect network becomes the bottle-neck of the system.

Bhaniramk el. design OpenGL Multipipe SDK (MPK), a toolkit for scalable parallel rendering based on OpenGL [8]. By separating the system's resource management and physical environment from the application, MPK is able to provide applications with run time configurability and scalability. It handles multipipe rendering by a lean abstraction layer via a conceptual callback mechanism, and that it runs different application tasks in parallel. MPK-

Published by Atlantis Press, Paris, France.

based applications can run seamlessly from single-processor, single-pipe desktop systems to large multi-processor, multipipe scalable graphics systems. Equalizer is designed on the same mechanism as the MPK, and it is the most advanced and powerful framework on multi-GPU parallel rendering [10]. Compared to MPK, it supports a fully distributed parallel rendering paradigm and features a more flexible task decomposition approach. But both of the MPK and the Equalizer couple the hardware rendering stage with the composition stage in the display thread, and this makes the two stages execute serially. When there are several GPUs in one node, the composition time seriously descents the system performance.

III. PIPELINE ORGANIZATION

To fully exploit the capabilities of multi-GPU, multi-core systems, the application has to maximize the usage of the GPU and CPU cores. The ideal way for OpenGL-based rendering is to use: one rendering thread with one off-screen buffer for each GPU, one on-screen window to display the result which is not bound to a specific GPU, and a task decomposition thread assigning a part of the rendering to each off-screen window. Limited by the operation system, every application has to assign a GPU for the system display. The users are allowed to assign some task on the display GPU thread to fully exploit the compute unit. Because the input of the composite stage depends on the rendering stage output, it makes the two stages couple together and execute serially. So the other GPU threads have to stall to wait for the display GPU thread and this descents the system performance. we decouple the hardware rendering thread from the composite thread with a hybrid rendering method, which use both hardware-based rendering and software-based rendering. This makes it possible to overlap the composite time with the GPU rendering time. With creating a frame buffer in main memory, the rendering stage and the composite stage can execute in parallel.

The parallel rendering approach is shown in fig 1. The off-screen threads render the task with the hardware. After that, they write back the partial result to the frame buffer in main memory. Then, the on-screen thread composites the partial results of the same frame in the frame queue. At last, it renders the image with software and sends the result to the display device. With decoupling the rendering stage and the composition stage, the rendering time and the composition time can overlap each other. In this way, the decoupled approach reduces the GPU stall, and improves the system's performance.

IV. THREAD MODEL

The parallel rendering thread model is shown in fig 1. There are three types of thread in the model: application thread, off-screen thread and on-screen thread. Every off-screen thread is assigned to a GPU. They execute the rendering task and read back intermediate result. The on-screen thread runs on CPU. It creates the display window, composites the partial result, renders the image and sends the final result to the display device. The application thread controls the application event logic. It reacts on events, updates its data and controls the rendering. Through separating the rendering code from the main event loop, we are able to create a thread for the GPU in the system to extend the model.

V. MEMORY ORGANIZATION

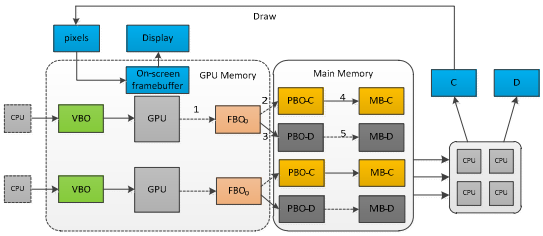


Figure 2. Memory organization architecture

Every GPU thread creates two PBO (pixel buffer object) and an off-screen framebuffer with a texture object. When the GPU thread finishes the rendering, it transfers the impartial results to the PBO in asynchronous. We firstly transfer the color to the PBO-C and the depth to the PBO-D. Then we map and copy the PBO-C to the MB-C and PBO-D to the MB-D, both of which are in the main memory. The composition thread composites the partial results in the frame buffer when all the partial results of the same frame are read back. The display thread renders the final result into on-screen framebuffer with software-based rendering and sends the pixels to the display device.

VI. RESULTS

We conducted our experiments on a multi-GPU, multi-core node with the following characteristics: quad-core 2.8GHz Intel Core i7 CPU with hyper-threading, 6GB of RAM, dual Geforce 250 GTS PCI-e graphics and a high-resolution 1920x1080 pixel LCD panel. The experiments tested and analyzed the system performance with our approach and the existing approach in sort-last at a variety of the viewports. With the combination of data size (Table I.) and the viewport, the radio of the rendering time to the composition time is at a variety.

In order to fully exploit the parallel rendering, we did not optimize its composition operation in the experiments. The reason is our approach has a high tolerance to the composition time. And the composition time do not contribute to the full image rendering time, when the sum time of the rendering and readback is larger than the composition time. The existing parallel rendering approach was implemented with Equalizer parallel rendering framework and its composition operation was well optimized with multithread method.

TABLE I. THE GEOMETRY MODEL USED IN THE EXPERIMENT.

Model	Vetexes	Trangles
Welsh-dragon	1,105,352	2,210,673
Asian Dragon	3,609,455	7,218,906
Thai Statue	5,000,000	10,000,000
Lucy	14,027,872	28,055,742

A. *Performance*

The composition time has different significant impact on the full image rendering time with the different viewport size in the coupled approach. In this experiment, we choose three different viewports: little (400x400), middle (800x800) and large (1024x1024). And the rendering time of all the data is always larger than the composition time at the little viewport. As viewport becoming larger, the composition time will have more significant impact on the full image rendering time with the existing approach. So the rendering time at the middle and large viewport will become larger than the composition time as data size changing larger.

The full image composition time is very small at the little viewport. As shown in the fig3.a, our approach performance is about 1.6 times to the existing one, when they render the welsh-dragon model. This is because the rendering time of the model is nearly equal the composition time and our approach overlaps the composition stage with the rendering stage. The rendering time becomes larger as the model size increasing and almost decides the full image rendering time. The speedup of the Lucy model is about 1.35.

The composition time at the middle viewport is 4 times to the one at the little viewport. And our approach performance becomes higher than the existing one again, and the results show that the speedup of the Lucy model is about 1.32. This also proves that the decoupling approach has a high tolerance to the composition time when the GPU number of the system increases.

The composition time at the large viewport is about 6.55 times to the one at the little viewport. The existing approach performance of all models is dropping as the viewport becomes big, though the composition operation has been well optimized. But our approach performance is almost the same when the rendering time is larger than the composition time. And the viewport size only affects the full image readback time. The speedup of the Lucy model is about 1.30, as shown in fig3.c.

From the above analysis，we can see that our parallel rendering approach overlaps the rendering and the composition stage and its performance is better than the existing one when the rendering task is large enough. We also can conclude that our approach is totally better than the existing one with any rendering task with the same composition optimization.

B. *Scalable Rendering*

We exploited our approach scalable with different model at little, middle and large viewport. As shown in the fig4.a, we first experimented with the welsh-dragon model. And the approach with dual GPU is no better than the one with single GPU at the large viewport because of lacking the composition optimization. But its speedup is about 1.71 at the little viewport. From above analysis, we can conclude that the approach can upgrade the speedup with the composition optimization. Then, we experimented with the Lucy model. Because the rendering time is always larger than the composition time, the speed up is nearly 2 at any viewport (fig4.b).

VII. CONCLUTION

In this paper, we present a parallel rendering approach in a multi-GPU, multi-core node which decouples the rendering and composition stage in display thread. The experiments show that our approach is better than the existing approach implemented with the Equalizer, which is the most powerful multi-GPU framework. We also test the approach's scalable rendering and get a linear performance speedup with the GPU number when the rendering task is large enough. Since our motherboard only has 2 PCI-E slots, a pity in our experiment is that we do not fully test GPU threads scalability, we will practically test GPU threads scalability in future.

ACKNOWLEDGMENT

This work is supported by the National Basic Research Program (No. 2009CB723803).

REFERENCES

[1] T. Fogal, H. Childs, S. Shankar, J. Krueger, R. Bergeron, P. Hatcher, large data visualization on distributed memory multi-GPU clusters, Proc. of High Performance Graphics,2010, pp. 57--66.

[2] NVIDIA Hybrid_SLI, http://www.nvidia.com/object/ hybrid_sli.html, 2011.

[3] AMD CrossFire, http://www.amd.com/us/PRODUCTS/ WORKSTATION/GRAPHICS/CROSSFIRE-PRO/Pages/crossfire-pro.aspx, 2011.

[4] M. Jordi, and G. Mark, Scaling of 3D game engine workloads on modern multi-GPU systems, Proc. of the High-Performance Graphics ,2009), pp. 37~46.

[5] G, Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan, WireGL: A scalable graphics system for clusters, Proc. of SIGGRAPH ,2001 , pp.129~140.

[6] H. Igehy, G. Stoll, and P. Hanrahan, The Design of a Parallel Graphics Interface, Proc. of SIGGRAPH,1998, pp. 141~150.

[7] G, Humphreys, H. Mike, T. James , Chromium: A stream-processing framework for interactive rendering on clusters , Proc. of SIGGRAPH ,2002 , pp. 693-702, 2002.

[8] P. Bhaniramka, P. C.D. Robert, S. Eilemann, OpenGL Multipipe SDK: A Toolkit for Scalable Parallel Rendering. Proc. of IEEE Visualization ,2005, pp.119~126.

[9] S. Eilemann, M. Makhinya, R. Pajarola, Equalizer: A Scalable Parallel Rendering Framework Visualization and Computer Graphics, IEEE Transactions on Graphics 15 ,2009, pp.436~452.

1

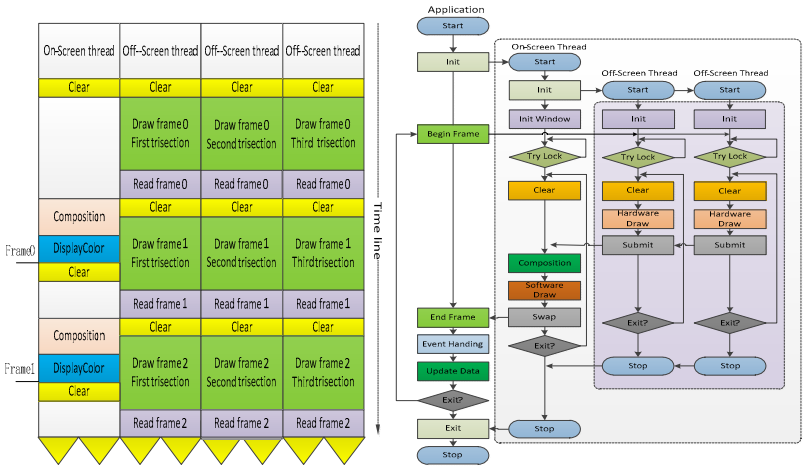
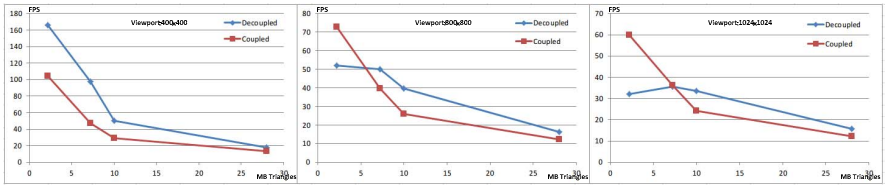
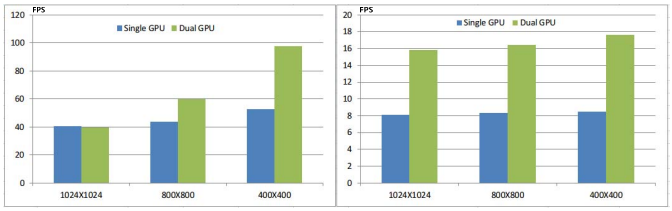


Figure 1. Multi-CPU, Multi-GPU parallel rendering pipeline and thread model



(a) viewport at 400x400, (b) viewport at 800x800, (c) viewport at 1024x1024.

Figure 3. Performance of the existing and our parallel rendering.



(a) Welsh-dragon, (b) Lucy.

Figure 4. Decoupling parallel rendering scalable with different model.