A Performance-Oriented Data Parallel Virtual Machine for GPUs (sketches_0451)

Mark Peercy
ATI Research, Inc.

Mark Segal
ATI Research, Inc.

Derek Gerstmann
ATI Research, Inc.

Abstract

Existing GPU programming interfaces require applications to adopt a graphics-centric programming model exported by a device driver tuned for real-time graphics and games. However, this programming model hinders the development, and performance, of non-graphics applications by imposing a graphics policy for program execution and hiding hardware resources. We present a new virtual machine abstraction for GPUs that provides policy-free, low-level access to the hardware and is designed for high-performance, data-parallel applications.

1 Overview

Several non-graphics applications, including physics, numerical analysis, and simulation, have been implemented on graphics processors in order to take advantage of their inexpensive raw compute power, and high-bandwidth latency-tolerant memory systems [GPGPU]. Unfortunately, such programs must rely on OpenGL and Direct3D to access the hardware. These APIs are simultaneously over-specified (one must set state and manipulate data that is not directly relevant) and underspecified (the inner workings of the hardware are, by design, suppressed) for this class of computation. In addition, the drivers that implement these APIs make critical policy decisions, such as where data resides in memory and when it is copied, that may be suboptimal.

This mismatch between interface and intent can compromise performance, undermining the very motivation for porting applications to the GPU. We present a new virtual machine abstraction for graphics processors that hides all non-relevant graphics components and exposes the GPU as a data parallel processor array and memory controller, fed by a simple command processor. The exported components are specified sufficiently close to the actual hardware such that an application appears to be speaking directly to a device without intervention. Having control of relevant policy and resources ensures that applications can maximize performance.

2 The Data Parallel Virtual Machine (DPVM)

The DPVM consists of three major components: the command processor, the data parallel array, and the memory controller (see Figure 1). All non-critical GPU features are hidden and managed by the virtual machine, enabling the DPVM to support multiple architectures with a simplified interface. Furthermore, the DPVM model allows the GPU to coexist with other applications. For example, a game may use a single graphics device to perform a physics simulation at one point and graphics rendering at another. A particular DPVM implementation can seamlessly integrate with video display drivers, graphics APIs, or other DPVM instantiations.

Command Processor: An application sends commands (such as set memory addresses and formats, invalidate and flush caches, and start program) to the DPVM by writing them into command buffers in memory, and then sending them to the DPVM. Methods to open and close a device, submit a command buffer, and wait for a command buffer are exported via a shared library. Global information, such as memory pool sizes and addresses are returned upon device open. This interface simplifies device communication, and eliminates unwanted policy.

Data Parallel Array: Computation is performed by a data-parallel processor array. The DPVM specifies an application binary interface that exposes the native instruction set architecture of the processors. Developers can program the GPU either in its native machine language, or through binary executables generated by a compiler from a higher level language. Once a program is compiled (or written directly), it is immune to driver changes that might affect its performance.

Memory Controller: In contrast to textures, render targets, and opaque program constructs, the DPVM presents graphics memory directly to the application. Program instructions, constants, inputs, outputs, and command buffers are stored in GPU (video) or PCI-Express (shared between GPU and CPU) memory locations specified by the application. The application also specifies the input and output data formats to the memory controller, prior to reading or writing memory. Memory can also be cast: values written in one format (e.g. a 4-channel array) can be read in another format (e.g. a 1-channel array of 4 times the length) without moving or copying data. This gives the application complete control over how it manages memory.

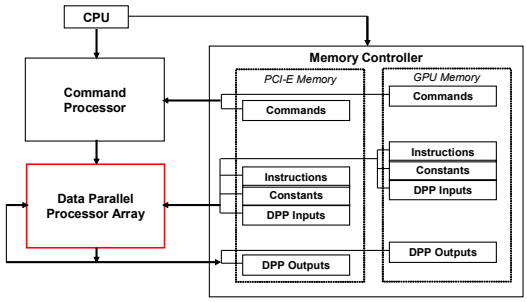

Figure 1. Simplified Block Diagram of the ATI DPVM.

Implementation: We have implemented the DPVM on ATI's X1k architecture [ATI] using components from ATI's graphics driver, as well as other custom support libraries. We implemented several data-parallel applications, including matrix-matrix multiply, FFT, and Julia set computation. Because the DPVM presents the processors and memory directly, we have been able to achieve performance not possible with the graphics APIs.

3 Conclusion

The DPVM provides a straightforward programming model for data parallel applications. It gives access to essential low-level functionality, yet presents a simplified target more palatable for tool development than the full hardware specification. As a result, developers can focus on compilers, debuggers, and libraries that target the data-parallel array of fragment processors without the burden of a graphics-centric driver. This is essential for developing high-performance data parallel applications that use the GPU for computation.

References

[GPGPU] General-Purpose Computation Using Graphics Hardware.<http://www.gpgpu.org>

[ATI] ATI, Inc. 2006. Radeon X1K Family Tech Overview.