

A Performance-Oriented Data Parallel Virtual Machine for GPUs

Mark Peercy Mark Segal Derek Gerstmann

AMD



Problem Statement

“...significant barriers still exist for the developer who wishes to use the inexpensive power of commodity graphics hardware, whether for in-game simulation of physics or for conventional computational science. These chips are designed for and driven by video game development; *the programming model is unusual, the programming environment is tightly constrained*, and the underlying architectures are largely secret. The GPU developer must be an expert in computer graphics and its computational idioms to make effective use of the hardware, and still pitfalls abound...”

» Course Description, SIGGRAPH 2005
GPGPU Course

Current Low-Level GPU Abstraction

Rendering Pipeline (OpenGL + Direct3D)

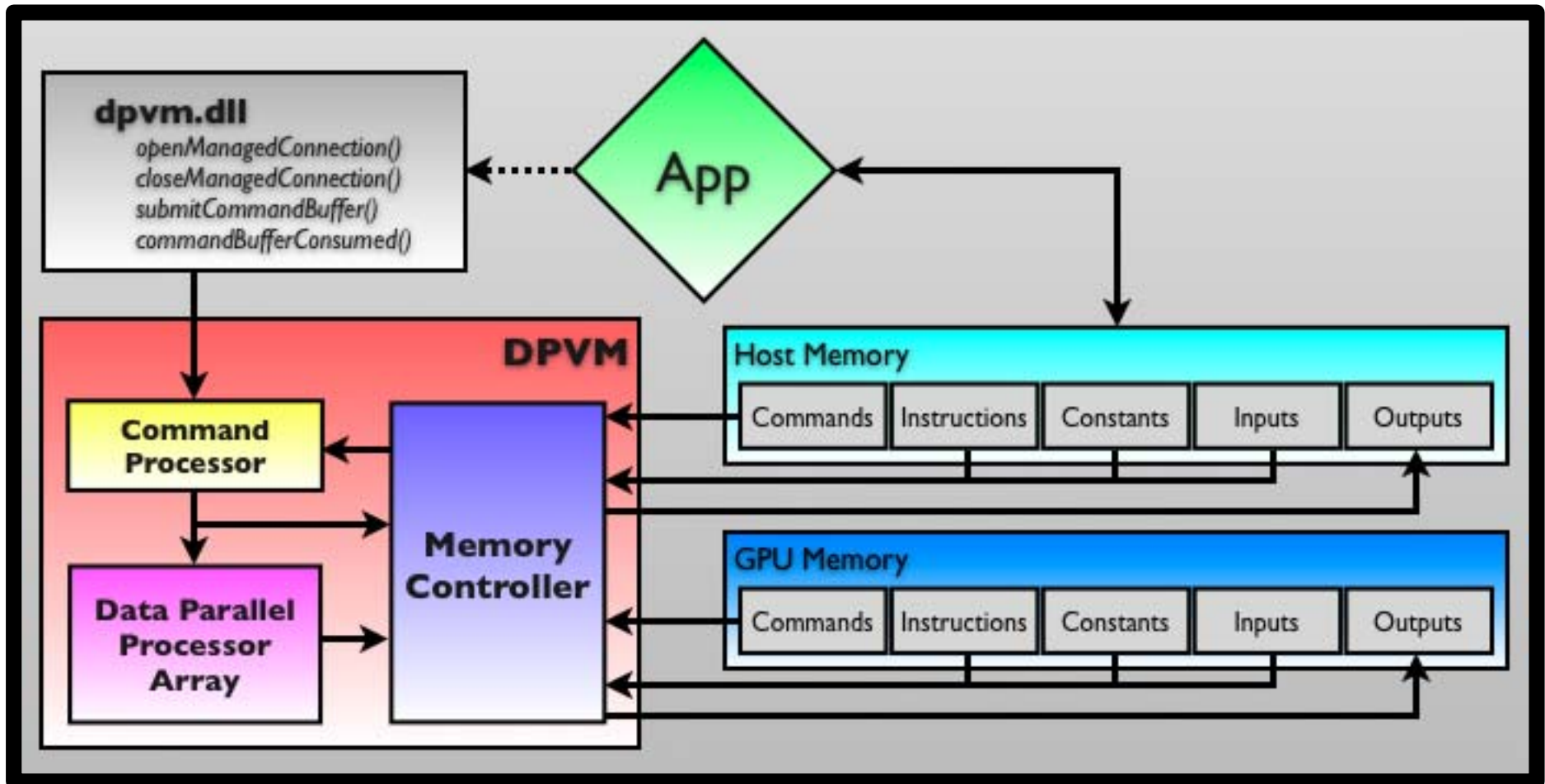
- Great for (existing) real-time graphics and games
- Cumbersome for other types of computation
 - Graphics-centric programming model
 - Forced to *manage graphics state*
- Implemented through graphics driver
 - Mechanism designed to *hide* hardware
 - Imposes *critical policy* decisions
 - How / when / where data resides
 - Updates + optimizations *driven by games...*

A Data Parallel Approach

The Data Parallel Virtual Machine (DPVM)

- Expose relevant parts of the GPU as they *really* are:
 - Command Processor
 - Data Parallel Processors
 - Memory Controller
- Hide all other graphics-specific features
- Provide direct communication to device
- Eliminate driver implemented procedural API
 - Push policy decisions back to *application*
 - *Remove constraints imposed by graphics APIs*

The Data Parallel VM



Command Processor

- Abstracts communication from architecture
 - Commands are architecturally *independent*
- Accepts command buffers (CBs) in memory
- Interprets commands in buffer
- Distributes work to processor array
- Application manages command buffers
 - Application fills and submits CBs
 - Application handles *synchronization*

Data Parallel Processors

- Performs floating-point computations
- Accepts binary executable (ELF)
 - Formal application binary interface (ABI)
 - Uses native instruction set architecture (ISA)
 - ISA is architecturally dependent
 - Only ISA needs to be updated for new architectures (ie. *recompile from high-level language*)
- Application submits compiled binary
 - ISA goes straight to the hardware
 - Executable is immune to driver changes

Memory Controller

- **Services GPU requests to read/write memory**
 - Exports graphics memory directly
 - GPU memory (accessible by GPU only)
 - Host memory (accessible by GPU + CPU)
- **Application manages memory resources**
 - Specifies locations and formats
 - Can cast between formats w/o copying data
 - Controls data submission + cache invalidation

ATI Close-to-the-Metal (CTM)

Additional Features (beyond SM3.0)

- Scatter (output `float1` values to arbitrary locations)
- Read + Modify + Write in a single program
- Fast tiled memory formats
 - Fetch4 (retrieve `x4 float1` in a single clock)
- ABI w/native ISA allows hand-tuned optimizations
- Ability to read/write *directly* to/from host memory
- Avoid undesirable floating-point optimizations
- Application controls CB submission (save to file)

CTM Usage

- **Initialize CTM device**
 - Obtain arena info: GPU & PCI-E memory
- **Allocate memory**
 - Inputs, outputs, constants, program, command buffer
- **Fill in a command buffer**
 - Set input/output formats, addresses; invalidate/flush caches
 - Load (binary) data-parallel array program
 - Specify computation domain
- **Submit command buffer; wait for completion**

CTM Example Applications

Runtime comparison (*Graphics API vs CTM*)

App	Benefit	Features
Matrix-Matrix Multiply	x7	CB, ISA, mem-formats, mem-offsets, interleaving, fetch4
FFT	x2	CB, ISA, interleaving
GPURay	x2	CB, mem-formats
QJulia	x2	CB, mem-formats

Measured on a single Radeon x1900

Conclusion

Benefits of the Data Parallel Approach

- Straight-forward programming model
 - Allows hand-tuned optimizations
- Exposes actual hardware device
 - Direct control over memory + processors
 - Application binary interface + native *ISA*
- Application is responsible for all *policy* decisions
- Allows *consistent* performance for compute

Future Work

Other things to explore...

- Open area for tool development
 - Low-level profilers + debuggers
- New opportunities for compiler research
 - ISA provides new target for code generation
 - Support for new high-level languages
 - Non-graphics based optimizations
 - Resource management for data parallel apps
- Extensions to expose more graphics functionality

QUESTIONS?

For availability and other information contact:

researcher@ati.com