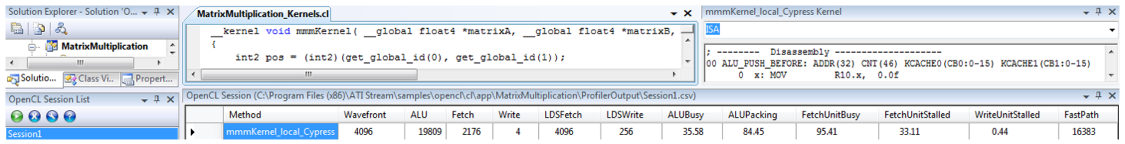
ATI Stream Profiler: a Tool to Optimize an OpenCL Kernel on ATI Radeon GPUs

Budirijanto Purnomo* Norman Rubin† Michael Houston‡
Advanced Micro Devices, Inc.


Solution Explorer - Solution 'O...
MatrixMultiplication
Solutio...
Class Vi...
Propert...
OpenCL Session List
Session1
MatrixMultiplication_Kernels.cl
mmmKernel_local_Cypress Kernel
ISA

```
__kernel void mmmKernel( __global float4 *matrixA, __global float4 *matrixB,
{
    int2 pos = (int2)(get_global_id(0), get_global_id(1));
```

```
; --------  Disassembly --------------------
00 ALU_PUSH_BEFORE: ADDR(32) CNT(46) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15)
      0  x: MOV        R10.x,  0.0f
```

OpenCL Session (C:\Program Files (x86)\ATI Stream\samples\opencl\cl\app\MatrixMultiplication\ProfilerOutput\Session1.csv)

Method	Wavefront	ALU	Fetch	Write	LDSFetch	LDSWrite	ALUBusy	ALUPacking	FetchUnitBusy	FetchUnitStalled	WriteUnitStalled	FastPath
mmmKernel_local_Cypress	4096	19809	2176	4	4096	256	35.58	84.45	95.41	33.11	0.44	16383

Figure 1: *ATI Stream Profiler is a run-time OpenCL^{TM} profiler developed as a Microsoft® Visual Studio 2008 plugin.*

1 Introduction

Modern GPUs have been shown to be highly efficient machines for data-parallel applications such as graphics, image, video processing, or physical simulation applications. For example, a single ATI Radeon^{TM} HD 5870 GPU has a theoretical peak of 2.72 teraflops (10^{12} floating-point operations per second) with a video memory bandwidth of 153.6 GB/s. While it is not difficult to port CPU algorithms to run on GPUs, it is extremely challenging to optimize the algorithms to achieve teraflops performance on GPUs. Only a select few expert engineers with the application domain expertise, a deep understanding of the modern GPU architecture, and an intimate knowledge of shader compiler optimization can program GPUs close to their optimal capabilities. Many developers are content with several folds of improvements rather than one or several orders of magnitude acceleration compared to their optimized CPU implementations.

In this work, we share several important lessons we learned in the process of developing ATI Stream Profiler (shown in Figure 1) for OpenCL^{TM}, an open standard tool for programming parallel applications on many-core architectures.

2 ATI Stream Profiler

The core functionality of ATI Stream Profiler is its ability to present a minimal set of meaningful and relevant performance counters derived from thousands of hardware raw signals supported by ATI GPUs. We highlight several important lessons for optimizing OpenCL^{TM} kernel on ATI Radeon^{TM} 5000 series GPUs.

2.1 ALU Optimization

One important measure of kernel performance is the effective rate of floating-point computation (*ALU*) compared to the peak theoretical rate of the GPU.

We show the utilization of the single-instruction multiple-data (SIMD) units in these two performance counters: *ALUBusy* and *ALUPacking*. The former is the rate of instruction processed by the SIMD units and the latter is the utilization of the five-issue architecture in the SIMD. Multiplying the values from these two counters gives you the percentage of the SIMD utilization. For example, 70% *ALUBusy* and 50% *ALUPacking* indicate the kernel is performing at 35% of the peak theoretical rate. Low *ALUBusy* indicates either not enough work is scheduled or ALU units are stalled due to data latency. To improve *ALUPacking*, developers can structure their codes to use more vector operations and/or reducing long dependency computation chains.

2.2 Global Memory Optimization

Since most kernels are memory-bound, it is important to optimize accesses to the global memory (video memory). We show the amount of data fetched from the global memory (*FetchMem*) and the amount of data written to the global memory (*FastPath* and *CompletePath*). *FastPath* and *CompletePath* denote the two memory paths in the hardware for writing data. The former is an optimized hardware path but supports only simple operations. The latter supports additional advanced operations including atomics and sub-32-bit (byte/short) data transfers. In our experiments, we observed an effective bandwidth of 20 GB/s for the *CompletePath* compared to 100+ GB/s for the *FastPath* when moving a block of data. Two reasons for the performance difference: (1) additional atomics data transferred for the *CompletePath*, and (2) the maximum bus utilization between the shader unit and the memory unit for the *CompletePath* is 25% compared to the 100% for the *FastPath*. To improve performance, we suggest performing atomics as partial reductions in the local memory and running a final kernel pass to combine the results.

When working with the image objects, developers need to adapt the access pattern to the data layout of the objects: tiled for the image objects versus linear for the buffer objects. We show the percentage of fetches that hit the L1 cache for the image objects (*L1CacheHit*).

2.3 Local Memory Optimization

You can also reduce the data transfer to the global memory by utilizing the local memory (local data share, or LDS). This memory unit has higher bandwidth (2 terabytes/s on the ATI Radeon^{TM} HD 5870) but limited size (32 KB per SIMD) compared to global memory bandwidth and size.

To utilize the local memory efficiently, developers need to minimize bank conflict accesses that will be processed serially by the units. We show the impact of the bank conflicts in terms of time spent in the GPU in the *LDSBankConflict* counter.

3 Future Work

Currently, we are working on expanding the counter sets and investigating the counter usage for automatic bottleneck detection.

*e-mail: Budirijanto.Purnomo@amd.com
†e-mail:Norman.Rubin@amd.com
‡e-mail:Michael.Houston@amd.com

Copyright is held by the author / owner(s).
SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007