Analyzing Program Flow within a Many-Kernel OpenCL Application

Perhaad Mistry, David Kaeli
Northeastern University

Chris Gregg, Kim Hazelwood
University of Virginia

Norman Rubin
Advanced Micro Devices

GPGPU-4 March 2011 – Newport Beach, CA
Topics

1. Motivation for OpenCL profiling
2. OpenCL events
3. Implementing an OpenCL Profiler using events
4. Speeded Up Robust Features (SURF)
5. Profiling SURF using OpenCL Events
6. Future Work
7. Conclusions
Motivation

- Heterogeneous hardware running increasingly complex algorithms
- The algorithm may only be a computational kernel within a high-level application
  - Algorithm developer does not always know the application his kernel will be used in
- Lots of algorithms exist where performance determined by factors other than “data size”
  - Performance analysis is required at runtime to learn about the application
OpenCL

- OpenCL is an API that enables portability in parallel computing on heterogeneous devices
  - Devices include CPUs, GPUs, Cell, DSPs, etc
    - Aim to optimally use all devices from within a single program

- Primary OpenCL Components
  - Platform Model – A context on a host which contains devices
  - Execution Model – An OpenCL application on a host submits work to the device through command queues
  - Memory and Programming Model - Similar explicit memory management to CUDA programming

- We will concentrate on a component of the execution model and its application to profiling on different devices
OpenCL Events

- OpenCL provides a platform to implement not only cross-platform applications but also tools for parallel computing.
- Events are an interface to understanding OpenCL performance.
  - Event objects (cl_event) used to determine command status.
- OpenCL enqueue methods return event objects.
  - Provide command level control and synchronization.

<table>
<thead>
<tr>
<th>Command State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_QUEUED</td>
<td>Command is in a queue</td>
</tr>
<tr>
<td>CL_SUBMITTED</td>
<td>Command has been submitted to device</td>
</tr>
<tr>
<td>CL_RUNNING</td>
<td>Command is currently executing on device</td>
</tr>
<tr>
<td>CL_COMPLETE</td>
<td>Command has finished execution</td>
</tr>
</tbody>
</table>

Command States as seen from OpenCL events

```
cl_int clEnqueueNDRangeKernel (  
    cl_command_queue  
    cl_kernel kernel, cl_uint work_dim,  
    const size_t *global_work_offset,  
    const size_t *global_work_size,  
    const size_t *local_work_size,  
    cl_uint num_events_in_wait_list,  
    const cl_event *event_wait_list,  
    cl_event *event)
```
Uses of OpenCL Events

- Scheduling – Events provides callbacks which can enqueue commands to the host or device
  - Scheduling asynchronous host device IO
  - Calling host based libraries when the device is finished

- Managing – Out of order or multiple command queues
  - Waitlists (an array of events) to indicate commands which must be complete before execution can proceed

- Timing and profiling using timestamps available at multiple stages of a command

- Events provided by every OpenCL conformant device
OpenCL Profiler Implementation

- Profiler maintains a list of events for each enqueue command passed to the runtime
- Events occur as kernel runs, queried at the end

Results and Analysis invoked at the end of the frame – off critical path
Speeded Up Robust Features (SURF)

- Motivating example to build an OpenCL based profiler
- "Summarize" an image into a number of "interest points"
  - Robust features - Simple to compute, small
  - More insensitive to changes in image like scale, rotation
- Applications: Object recognition, tracking, image stitching etc

I-point

- float2 Pixel Position
- float Orientation
- float Scale
- float Descriptor[64]

Speeded-Up Robust Features (SURF), Herbert Bay et. al.
OpenCL Profiler in SURF

Approximated Filters

Integral Image

Hessian Residuals

Non-Max Suppression

Orientation

SURF64 Descriptors

Event Table

Name Data

Results, Analysis, Feedback

References to event objects

OP = Features per frame

cl_event

cl_event

cl_event

cl_event

cl_event

cl_event

SURF

ocl_profiler
Performance Characteristics of SURF

- Performance is hard to predict because of variable feature counts
  - Feature count decides the workgroup sizes down the pipeline

- We aim to study SURF’s performance when embedded into applications
  - Not always as clean as embedding a spmv kernel in a solver

- Many OpenCL kernels of varying complexity
  - 10 kernels varying from 5 lines to 280 lines
  - Kernels called multiple times
    - Number of kernel calls unknown until runtime
Performance Analysis of SURF using Profiler

- Profiling applied on SURF for different OpenCL devices
  - Per frame “traces” saved as the application runs
- SURF is ONLY recompiled when switching devices
  - No platform specific tuning
  - AMD (5870, Phenom-II x6) and Nvidia (GTX 480) platforms
- Possible optimization spaces in SURF implementations
  - Different kernels which take a similar amount of time
    - Complicated kernel called once, simple kernels called often
  - Asynchronous host device interaction
Performance Variation

- Performance variation for videos of similar frame size
  - Use case for runtime performance analysis

- Same input parameters
  - Variation due to differing feature count
  - Cannot predict feature count

- Profiling helps study performance per data set
  - More than just “average time per frame”

Variation in time/frame across frames and number of i-points for 2 videos of similar resolution
Kernel Timeline in SURF

- Application view of SURF
  - Pipelined kernel events over data set
  - Averaged event time stamps for a data set
- Expose optimization space
  - Cumulative time of small kernel
  - High kernel call count
  - Device – host IO is not significant in pipeline
- Used to estimate host idle time once kernels enqueued

Similar data plotted on any compliant device
Individual Kernel Performance

- Optimization steps for kernels
  - Average timing of each kernel across frames
- Events show a consistent view across devices
- Individual timings may not be representative
- Surf64 kernel is the first candidate for optimization
- But BuildDet is called 12x more

Optimizing to reduce kernel calls may be as beneficial as platform specific optimization
Profiler Overhead

- Baseline of profiling disabled in command queue
  - Overhead for different videos
- Simple techniques to minimize overhead
- Grow event list once and reuse data structures
- Query events after frame
- Variable granularity of performance measurement

Consistent overhead seen - per platform

We show the worst case overhead for SURF- profiling all kernels for every frame
Future Work

- Profile more OpenCL applications especially those using multiple devices and command queues

- Capture information passed by user (data size, command queue details etc.) while still sticking to specification functions
  - We are intercepting calls to OpenCL runtime

- Interaction between profiling framework and the higher level applications like stabilization and searching
  - Using profiling results to drive optimizations of different components of SURF for computer vision applications
  - Our present work restricted to core SURF pipeline

- Will be open sourced very soon
Conclusions

☐ This work was motivated by an interesting case of data dependent parallelism performance

☐ The OpenCL specification provides a useful interface to understand application performance
  ■ Similar information provided for different devices

☐ Compliments existing tools like the Stream Profiler and Nvidia OpenCL Profiler
  ■ Switching tools for different devices used while developing and testing is really annoying
  ■ We get results either at the end or during an application’s run and can express profiling results back to the algorithm
Thank You, Questions?

Perhaad Mistry
pmistry@ece.neu.edu