GPU accelerated application tracing

David Couturier
B. Eng.

Updated in May 2015
CONTENT

Page 03 / Context
Page 10 / Objectives
Page 12 / Solution
Page 19 / Methodology
Page 22 / Results
Page 26 / Use Cases
Page 30 / Future Work
CONTEXT: HETEROGENEOUS HARDWARE

CPU
- 1 to 8 physical cores
- High frequency (< ~ 4 GHz)
- Serial computation
- 100s of GFLOPS

GPU
- Up to 3072 computation cores
- Moderated frequency (< ~ 1.2 GHz)
- Parallel computations (SIMD)
- 1000s of GFLOPS

CONTEXT: HETEROGENEOUS HARDWARE

- NVIDIA: Tegra
- Intel: Intel HD Graphics
- Qualcomm: Snapdragon
- AMD: « APU » (Accelerated processing unit)

- Zynq (CPU + FPGA)
- XeonPhi

Snapdragon 810 (Qualcomm)

Source: qualcomm.com
CONTEXT: ARCHITECTURE

Source: nvidia.com
CONTEXT: OPENCL

User Space

OpenCL Application → OpenCL

Kernel Space

Linux Kernel → Drivers (Graphic, Audio, ...)

Hardware Layer

CPU → Sound card → GPU → ...

20/05/2015
CONTEXT: TRACING

Trace CPU

Trace GPU
Trace Compass:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>11393</td>
<td>11256</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ltng</td>
<td>11423</td>
<td>11393</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sinoscope</td>
<td>11456</td>
<td>11393</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sinoscope</td>
<td>11457</td>
<td>11456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sinoscope</td>
<td>11458</td>
<td>11456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ltng</td>
<td>11460</td>
<td>11393</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unity-scope-hom</td>
<td>3301</td>
<td>1982</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CodeXL (AMD’s GPU tracing tool)

Limitations:
- OpenCL application has to be launched by CodeXL
- Recording trace performance for large traces
- AMD GPU only
OBJECTIVE: UNIFIED TRACE

Unified Trace
SECONDARY OBJECTIVES

- Performance
  - Minimal overhead on the system
- Problem solving
SOLUTION: OPENCL PROFILING

clGetEventProfilingInfo
- Task’s enqueued time
- Task’s submission time
- Task’s start time
- Task’s end time

Host Th.2

(1) (2) clFinish(); ...

Device

OpenCL Kernel Exec.

(3) (4) (5)

Host Th.1
SOLUTION: NON MONOTONIC PROFILING

```
(1) clFinish();
(2) ...
```

```
Host Th.1  Device  Host Th.2
(3)       (4)    (5)
OpenCL Kernel Exec.
```
SOLUTION: HOST-DEVICE SYNCHRONIZATION

Host Th.1 ➜ (1) ➜ (2) ➜ `clFinish();` ➜ ... ➜ Device ➜ OpenCL Kernel Exec. ➜ (3) ➜ (4) ➜ (5) ➜ Host Th.2
SOLUTION: DYNAMIC SYMBOL OVERLOADING

User Space
- OpenCL Application
- CLUST
- OpenCL

Kernel Space
- Linux Kernel
- Drivers (Graphic, Audio, ...)

Hardware Layer
- CPU
- Sound card
- GPU
- ...
SOLUTION: SYNCHRONOUS API CALL

c_int clGetPlatformIDs(cl_uint num_entries, cl_platform_id * platforms, cl_uint * num_platforms) {
    tracepoint(clust_provider, cl_clGetPlatformIDs_start);
    c_int ret = reallib_clGetPlatformIDs(num_entries, platforms, num_platforms);
    tracepoint(clust_provider, cl_clGetPlatformIDs_end);
    return ret;
}
SOLUTION: ASYNCHRONOUS API CALL

```c
cl_int clEnqueueReadBuffer(cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_event *event, cl_int num_events_in_wait_list)
{
    const bool trace = __tracepoint_clust_provider__clust_device_event.state;
    bool toDelete = false;

    if(caa_unlikely(trace)) {
        if(event == NULL) {
            event = malloc(sizeof(cl_event));
            toDelete = true;
        }
    }

    tracepoint(clust_provider, cl_clEnqueueReadBuffer_start);
    cl_int ret = reallib_clEnqueueReadBuffer(command_queue, buffer, blocking_read, offset, cb, ptr, num_events_in_wait_list);
    tracepoint(clust_provider, cl_clEnqueueReadBuffer_end);

    if(caa_unlikely(trace)) {
        int r = reallib_clSetEventCallback(event, CL_COMPLETE, &eventCompleted, (toDelete)?&ev_delete:&ev_keep);
        if(r != CL_SUCCESS) fprintf(stderr, "CLUST::clEnqueueReadBuffer->clSetEventCallback: error->%d\n", r);
    }

    return ret;
}
```
// Get event start time
cl_int ret = clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &start, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL PROFILING COMMAND_START returned %d\n", ret);

// Get event end time
ret = clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &end, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL PROFILING COMMAND_END returned %d\n", ret);

// Get event enqueue time
ret = clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_QUEUE, sizeof(cl_ulong), &queued, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL PROFILING COMMAND_QUEUE returned %d\n", ret);

// Get event submit time
ret = clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_SUBMIT, sizeof(cl_ulong), &submit, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL PROFILING COMMAND_SUBMIT returned %d\n", ret);

// Get event command name (CL COMMAND NDRANGE_KERNEL, CL_COMMAND_WRITE_BUFFER, ...)
ret = clGetEventInfo(event,CL_EVENT_COMMAND_TYPE,sizeof(cl_command_type), &command, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL EVENT_COMMAND_TYPE returned %d\n", ret);

// Get event queue id
ret = clGetEventInfo(event,CL_EVENT_COMMAND_QUEUE,sizeof(cl_command_queue), &queue, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUT::eventCompleted:error->CL EVENT COMMAND QUEUE returned %d\n", ret);

// Record with UST tracepoint
tracepoint(clust provider, clust device event, (ulong)queue, command, queued, submit, start, end);
if(*releaseEvent == ev_delete) {
    releaseEvent = NULL;
}
19/05/2015
METHODOLOGY
CONFIGURATION

• Intel i7-4770 with HD Graphics 4600 integrated graphics
• 32 GB DDR3 RAM
• Ubuntu 14.04 (Kernel 3.18.4 + patch)
• Beignet v1.0.2 OpenCL drivers + patch
• LTTng v2.6.0-rc1

• Monotonic clock

• 1000s of iterations measured many times to acquire statistics
METHODOLOGY
MEASURING SYNCHRONOUS FUNCTION OVERHEAD

c1GetPlatformIDs(num_entries, platforms, num_platforms);
METHODOLOGY
MEASURING ASYNCHRONOUS FUNCTION OVERHEAD

clEnqueueReadBuffer(command_queue, buffer, blocking_read, offset, cb, ptr, num_events_in_wait_list)
## RESULTS: CLUST OVERHEAD SYNCHRONOUS FUNCTIONS

<table>
<thead>
<tr>
<th>Qty. (iterations)</th>
<th>Reference (ns)</th>
<th>Preload (ns)</th>
<th>UST Tracing (ns)</th>
<th>Preload Overhead (ns)</th>
<th>UST Tracing Overhead (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>16</td>
<td>18</td>
<td>383</td>
<td>2</td>
<td>367</td>
</tr>
<tr>
<td>$10^1$</td>
<td>5.2</td>
<td>7.8</td>
<td>366.5</td>
<td>2.6</td>
<td>361.3</td>
</tr>
<tr>
<td>$10^2$</td>
<td>4.64</td>
<td>6.66</td>
<td>365.68</td>
<td>2.02</td>
<td>361.04</td>
</tr>
<tr>
<td>$10^3$</td>
<td>4.291</td>
<td>6.058</td>
<td>365.168</td>
<td>1.767</td>
<td>360.877</td>
</tr>
<tr>
<td>$10^4$</td>
<td>4.277</td>
<td>6.283</td>
<td>359.780</td>
<td>2.006</td>
<td>355.503</td>
</tr>
<tr>
<td>$10^5$</td>
<td>4.526</td>
<td>6.484</td>
<td>359.379</td>
<td>1.958</td>
<td>354.853</td>
</tr>
<tr>
<td>$10^6$</td>
<td>4.531</td>
<td>6.467</td>
<td>363.313</td>
<td>1.936</td>
<td>358.782</td>
</tr>
<tr>
<td>$10^7$</td>
<td>4.537</td>
<td>6.499</td>
<td>361.145</td>
<td>1.962</td>
<td>356.608</td>
</tr>
<tr>
<td>$10^8$</td>
<td>4.535</td>
<td>6.460</td>
<td>361.108</td>
<td>1.925</td>
<td>356.573</td>
</tr>
</tbody>
</table>

→ ~ 1 ns overhead per inactive UST tracepoint
→ ~ 180 ns overhead per active UST tracepoint
## RESULTS: CLUST OVERHEAD
### ASYNCHRONOUS FUNCTIONS

<table>
<thead>
<tr>
<th>Buffer size (byte)</th>
<th>Reference (ns)</th>
<th>Preload (ns)</th>
<th>UST Tracing (ns)</th>
<th>Preload Overhead (ns)</th>
<th>Tracing overhead (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4 \times 10^0$</td>
<td>149.51</td>
<td>164.7</td>
<td>7000.6</td>
<td>15.2</td>
<td>6851.1</td>
</tr>
<tr>
<td>$4 \times 10^1$</td>
<td>158.99</td>
<td>168.7</td>
<td>7026.8</td>
<td>9.7</td>
<td>6867.8</td>
</tr>
<tr>
<td>$4 \times 10^2$</td>
<td>156.15</td>
<td>174.7</td>
<td>7269.3</td>
<td>18.5</td>
<td>7113.2</td>
</tr>
<tr>
<td>$4 \times 10^3$</td>
<td>188.44</td>
<td>226.7</td>
<td>7043.6</td>
<td>38.3</td>
<td>6855.2</td>
</tr>
<tr>
<td>$4 \times 10^4$</td>
<td>1499.76</td>
<td>1503.3</td>
<td>8393.0</td>
<td>3.6</td>
<td>6893.3</td>
</tr>
<tr>
<td>$4 \times 10^5$</td>
<td>17805.67</td>
<td>17862.1</td>
<td>25404.7</td>
<td>56.4</td>
<td>7599.0</td>
</tr>
</tbody>
</table>

→ [3.6, 56.4] ns overhead per « preloaded » asynchronous call*
→ ~ 7030 ns overhead per traced asynchronous call
## RESULTS: CLUST OVERHEAD
### REAL OPENCL APPLICATION

<table>
<thead>
<tr>
<th>Buffer width (pixels)</th>
<th>Buffer height (pixels)</th>
<th>Reference (ns)</th>
<th>Preload (ns)</th>
<th>UST Tracing (ns)</th>
<th>UST + Kernel Tracing (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>35198</td>
<td>36399</td>
<td>50890</td>
<td>58838</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>35183</td>
<td>35702</td>
<td>51883</td>
<td>58265</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>36031</td>
<td>36890</td>
<td>50758</td>
<td>59619</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>37937</td>
<td>39067</td>
<td>55820</td>
<td>61108</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>56770</td>
<td>59709</td>
<td>75073</td>
<td>84746</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>250694</td>
<td>251165</td>
<td>268726</td>
<td>280299</td>
</tr>
<tr>
<td>1280</td>
<td>720</td>
<td>1951826</td>
<td>1951965</td>
<td>1976916</td>
<td>1988445</td>
</tr>
<tr>
<td>1920</td>
<td>1080</td>
<td>4466096</td>
<td>4466777</td>
<td>4491589</td>
<td>4511394</td>
</tr>
</tbody>
</table>
RESULTS: CLUST OVERHEAD
REAL OPENCL APPLICATION (…)

Overhead vs workload

<table>
<thead>
<tr>
<th>Buffer width (pixels)</th>
<th>Buffer height (pixels)</th>
<th>Preload Overhead (%)</th>
<th>Tracing Overhead (%)</th>
<th>UST Tracing Overhead (%)</th>
<th>UST + Kernel Tracing Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3.41%</td>
<td>44.58%</td>
<td>67.16%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1.48%</td>
<td>47.47%</td>
<td>65.61%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2.38%</td>
<td>40.87%</td>
<td>65.47%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>2.98%</td>
<td>47.14%</td>
<td>61.08%</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>5.18%</td>
<td>32.24%</td>
<td>49.28%</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>0.19%</td>
<td>7.19%</td>
<td>11.81%</td>
<td></td>
</tr>
<tr>
<td>1280</td>
<td>720</td>
<td>0.01%</td>
<td>1.29%</td>
<td>1.88%</td>
<td></td>
</tr>
<tr>
<td>1920</td>
<td>1080</td>
<td>0.02%</td>
<td>0.57%</td>
<td>1.01%</td>
<td></td>
</tr>
</tbody>
</table>

Relative overhead (%) vs Workload (Megapixels)
USE CASES

- System-wide unified tracing
- Flight recording mode
  - 24/7 volatile recording
  - Hard drive dump when required
- OpenCL application optimization
- OpenCL application debugging
USE CASE: RESSOURCE SHARING

Legend

Process States:
- UNKNOWN
- WAIT_BLOCKED
- WAIT_FOR_CPU
- USERMODE
- SYSCALL
- INTERRUPTED

Control Flow

Resources

Statistics

Process | TID   | PTID | Time Stamps
--------|-------|------|-----------------
bash    | 11393 | 11256| 18:43:10.120
lltn    | 11423 | 11393| 18:43:10.130
sinoscope| 11456| 11393| 18:43:10.140
sinoscope| 11456| 11393| 18:43:10.150
sinoscope| 11456| 11393| 18:43:10.150
ltng    | 11460 | 11393|
unity-scope-hom | 3301 | 1982 |
**USE CASE: CPU PREEMPTION**

![Control Flow Diagram](image)

<table>
<thead>
<tr>
<th>Process</th>
<th>TID</th>
<th>PTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>11393</td>
<td>11256</td>
</tr>
<tr>
<td>ltng</td>
<td>11423</td>
<td>11393</td>
</tr>
<tr>
<td>sinoscope</td>
<td>11456</td>
<td>11393</td>
</tr>
<tr>
<td>sinoscope OCL API</td>
<td>11456</td>
<td></td>
</tr>
<tr>
<td>sinoscope GPU Accl</td>
<td>11456</td>
<td></td>
</tr>
<tr>
<td>ltng</td>
<td>11460</td>
<td>11393</td>
</tr>
<tr>
<td>unity-scope-hom</td>
<td>3301</td>
<td>1982</td>
</tr>
</tbody>
</table>

**Legend**

- **UNKNOWN**
- **WAIT_BLOCKED**
- **WAIT_FOR_CPU**
- **USERMODE**
- **SYSCALL**
- **INTERRUPTED**
USE CASE: OPENCL PIPELINE USAGE MAXIMISATION
FUTURE WORK

• Trace analysis utilities

• Hardware performance counter data acquisition

• OpenCL 2.0 support

• OpenGL tracing
  • Vulkan API

• Heterogeneous computing framework
  • CPU-GPU
  • CPU-DSP
  • ...
Questions?
Thank you!