“OpenCL supports a wide range of applications... through a low-level, high-performance, portable abstraction.”

Page 11: OpenCL 2.1 specification
“OpenCL supports a wide range of applications... through a low-level, high-performance, portable abstraction.”

Page 11: OpenCL 2.1 specification
“OpenCL supports a wide range of applications... through a low-level, high-performance, portable abstraction.”

Page 11: OpenCL 2.1 specification

We consider functional portability rather than performance portability
Example

• single source shortest path application
Example

• single source shortest path application
Example

• single source shortest path application

Quadro K5200 (Nvidia), Intel HD5500

[Capture of a computer screen showing a terminal with error messages and console output.]
An experience report on OpenCL portability

• How well is portability evaluated?

• Our experience running applications on 8 GPUs spanning 4 vendors

• Recommendations going forward
An experience report on OpenCL portability

• How well is portability evaluated?

• Our experience running applications on 8 GPUs spanning 4 vendors

• Recommendations going forward
Portability in research literature

• Reviewed the 50 most recent OpenCL papers on: http://hgpu.org/

  • Only considered papers including GPU targets
  • Only considered papers with some type of experimental evaluation

• How many different vendors did the study experiment with?
Portability in research literature

Results
(number of evaluated vendors)

58% (29)
Portability in research literature

Results
(number of evaluated vendors)

- 58% (29) for Vendor 2
- 36% (18) for Vendor 1
Portability in research literature

Results
(number of evaluated vendors)

- 58% (29)
- 36% (18)
- 6% (3)
Portability in research literature

Results (which vendor)

- Nvidia: 39
- AMD: 23
- Intel: 8
- ARM: 3
- Imagination: 1
Portability in research literature

Results (which vendor)

Portability is not well tested in research literature!

- **Nvidia**: 39
- **AMD**: 23
- **Intel**: 8
- **ARM**: 3
- **Imagination**: 1
An experience report on OpenCL portability

• How well is portability evaluated?

• Our experience running applications on 8 GPUs spanning 4 vendors

• Recommendations going forward
Chips we test

<table>
<thead>
<tr>
<th>Chip</th>
<th>Vendor</th>
<th>Compute Units</th>
<th>OpenCL Version</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>Nvidia</td>
<td>16</td>
<td>1.1</td>
<td>Discrete</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>Nvidia</td>
<td>12</td>
<td>1.1</td>
<td>Discrete</td>
</tr>
<tr>
<td>Iris 6100</td>
<td>Intel</td>
<td>47</td>
<td>2.0</td>
<td>Integrated</td>
</tr>
<tr>
<td>HD 5500</td>
<td>Intel</td>
<td>24</td>
<td>2.0</td>
<td>Integrated</td>
</tr>
<tr>
<td>Radeon R9</td>
<td>AMD</td>
<td>28</td>
<td>2.0</td>
<td>Discrete</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>AMD</td>
<td>8</td>
<td>2.0</td>
<td>Integrated</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>ARM</td>
<td>4</td>
<td>1.2</td>
<td>Integrated</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>ARM</td>
<td>2</td>
<td>1.2</td>
<td>integrated</td>
</tr>
</tbody>
</table>
Applications

• Part of a larger study on GPU irregular parallelism
Applications

Pannotia

• Target AMD Radeon HD 7000

• Written in OpenCL 1.x

• 4 graph algorithms applications

https://github.com/pannotia/pannotia
Applications

Pannotia

• Target AMD Radeon HD 7000

• Written in OpenCL 1.x

• 4 graph algorithms applications

Loop until a fixed point is reached.

https://github.com/pannotia/pannotia
Applications

LonestarGPU

• Target Nvidia Kepler and Fermi

• Written in CUDA

• 4 graph algorithms applications

http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu
Applications

LonestarGPU

• Target Nvidia Kepler and Fermi

• Written in CUDA

• 4 graph algorithms applications

http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu
Applications

• Total of 8 applications

• Experience report of:
  
  • Porting LonestarGPU to OpenCL
  
  • Running Pannotia cross platform
  
  • Experimenting with new synchronisation idioms via OpenCL 2.0 atomics
Portability Issues

12 issues encountered, grouped into categories

• 3 Framework bugs

• 6 Specification limitations

• 3 Programming bugs
Portability Issues

12 issues encountered, grouped into categories

• 3 Framework bugs

• 6 Specification limitations

• 3 Programming bugs
Framework bugs

#1 Compiler crash

Platforms: Intel
Framework bugs

#1 Compiler crash

Platforms: Intel
Framework bugs

#1 Compiler crash

Platforms: Intel

compiling several large kernels occasionally crashes compiler

Workaround: reduce the number of kernels in file
Framework bugs

#2 Non-terminating loops

Platforms: Nvidia and AMD
Framework bugs

#2 Non-terminating loops

Platforms: Nvidia and AMD

This looping idiom used in kernel code

```c
while(true) {
    more_work = false;

    // Do computation,
    // if more work, set more_work

    if (!more_work)
        break;
}
```
Framework bugs

#2 Non-terminating loops

Platforms: Nvidia and AMD

while(true) {
    more_work = false;

    .. // Do computation,
    .. // if more work, set more_work

    if (!more_work)
        break;
}

This looping idiom used in kernel code

Does not terminate on Nvidia and AMD platforms!!
Framework bugs

#2 Non-terminating loops

Platforms: Nvidia and AMD

Change while loop to for loop

End value of i is consistent across platforms

```c
while(true) {
    for (int i = 0; i < INT_MAX; i++) {
        more_work = false;

        .. // Do computation,
        .. // if more work, set more_work

        if (!more_work) 
            break;
    }
}
```

This looping idiom used in kernel code
Framework bugs

#3 AMD defunct processes

*Platforms*: AMD on Linux

Long running kernels become defunct and un-killable requiring a reboot.

*Workaround*: Switch to Windows OS
Portability Issues

12 issues encountered, grouped into categories

• 3 Framework bugs

• 6 Specification limitations

• 3 Programming bugs
Specification limitations

#1 GPU watchdogs
Platforms and operating systems handle watchdogs differently.
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

Controlled with registry

Watchdog kills entire OpenCL process

![Windows GPU](image1)

![Linux (Ubuntu) GPU](image2)

![Chrome OS GPU](image3)
# Specification limitations

**#1 GPU watchdogs**

Platforms and operating systems handle watchdogs differently.

- **Windows:** Controlled with registry. Watchdog kills entire OpenCL process.
- **Linux (Ubuntu):** Controlled in X server settings. Watchdog only kills kernel.
- **Chrome OS:**
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

- **Windows**
  - Controlled with registry
  - Watchdog kills entire OpenCL process

- **Linux (Ubuntu)**
  - Controlled in X server settings
  - Watchdog only kills kernel

- **Chrome OS**
  - Cannot control at all without recompiling the driver
Specification limitations

#2 Occupancy vs compute units

An OpenCL device has one or more compute units. A workgroup executes on a single compute unit.

Intel OpenCL Optimisation Guide
Specification limitations

#2 Occupancy vs compute units

An OpenCL device has one or more compute units. A workgroup executes on a single compute unit.

Intel OpenCL Optimisation Guide

Persistent thread model (Gupta et al. PIPC’12): forward progress between occupant workgroups
Specification limitations

#2 Occupancy vs compute units

An OpenCL device has one or more compute units. A workgroup executes on a single compute unit.

Intel OpenCL Optimisation Guide

Persistent thread model (Gupta et al. PIPC’12): forward progress between occupant workgroups

LonestarGPU applications depend on this
### #2 Occupancy vs compute units

<table>
<thead>
<tr>
<th>chip</th>
<th>compute units</th>
<th>PT occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Quadro K500</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Iris 6100</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>HD 5500</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Radeon R9</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Radeon R7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Mali-T628</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Mali-T628</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Specification limitations

#2 Occupancy vs compute units

<table>
<thead>
<tr>
<th>chip</th>
<th>compute units</th>
<th>PT occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Quadro K500</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Iris 6100</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>HD 5500</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Radeon R9</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Radeon R7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Mali-T628</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
#2 Occupancy vs compute units

<table>
<thead>
<tr>
<th>chip</th>
<th>compute units</th>
<th>PT occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Iris 6100</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>HD 5500</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Radeon R9</td>
<td>28</td>
<td>48</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Compute units are safe and optimal

Compute units are safe but not optimal
## Specification limitations

### #2 Occupancy vs compute units

<table>
<thead>
<tr>
<th>chip</th>
<th>compute units</th>
<th>PT occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Iris 6100</td>
<td>47</td>
<td>6</td>
</tr>
<tr>
<td>HD 5500</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>Radeon R9</td>
<td>28</td>
<td>48</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Compute units are safe and optimal
- Compute units are safe but not optimal
- Compute units are not safe
Portability Issues

12 issues encountered, grouped into categories

• 3 Framework bugs

• 6 Specification limitations

• 3 Programming bugs
Programming bugs

#1 Data-races

Application: LonestarGPU bfs and sssp
Fix: Add additional synchronisation barriers
Programming bugs

#1 Data-races

Application: LonestarGPU bfs and sssp
Fix: Add additional synchronisation barriers

Bug was dormant on Nvidia but caused crashes on Intel
Programming bugs

#2 Struct kernel arguments

How to represent a graph:
Programming bugs

#2 Struct kernel arguments

How to represent a graph:

- adjacency matrix
- array of edge weights
- number of nodes
- number of edges
Programming bugs

#2 Struct kernel arguments

Graphs are large and globally shared so they go into global memory.

Each struct member is a global memory pointer

How to represent a graph:

```
struct Graph

• adjacency matrix
• array of edge weights
• number of nodes
• number of edges
```
Programming bugs

#2 Struct kernel arguments

c1SetKernelArg (bfs_kernel, 0, sizeof(Graph), &graph1);

// Execute bfs kernel

<table>
<thead>
<tr>
<th>Chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
</tr>
<tr>
<td>Quadro K500</td>
</tr>
<tr>
<td>Iris 6100</td>
</tr>
<tr>
<td>HD 5500</td>
</tr>
<tr>
<td>Radeon R9</td>
</tr>
<tr>
<td>Radeon R7</td>
</tr>
<tr>
<td>Mali-T628</td>
</tr>
<tr>
<td>Mali-T628</td>
</tr>
</tbody>
</table>
Programming bugs

#2 Struct kernel arguments

c1SetKernelArg (bfs_kernel, 0, sizeof(Graph), &graph1);

// Execute bfs kernel

<table>
<thead>
<tr>
<th>Chip</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>✓</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>✓</td>
</tr>
<tr>
<td>Iris 6100</td>
<td>✓</td>
</tr>
<tr>
<td>HD 5500</td>
<td>✓</td>
</tr>
<tr>
<td>Radeon R9</td>
<td>✓</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>✓</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>×</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>×</td>
</tr>
</tbody>
</table>
Programming bugs

#2 Struct kernel arguments

“Arguments to kernel functions that are declared to be a struct or union do not allow OpenCL objects to be passed as elements of the struct or union”

Page 176: OpenCL 2.0 specification
An experience report on OpenCL portability

• How well is portability evaluated?

• Our experience running applications on 8 GPUs spanning 4 vendors

• Recommendations going forward
Going forward

• Conformance tests

• Compiler Fuzzing
  • “Many-Core Compiler Fuzzing” PLDI’16, Lidbury et al.

• Memory consistency
  • “GPU Concurrency: Weak Behaviours and Programming Assumptions” ASPLOS’15, Alglave et al.
Going forward

• Conformance tests

  • Compiler Fuzzing
    • “Many-Core Compiler Fuzzing” PLDI’16, Lidbury et al.

  • Memory consistency
    • “GPU Concurrency: Weak Behaviours and Programming Assumptions” ASPLOS’15, Alglave et al.

unofficial open source tests?
Going forward

• Specification clarifications

  • Inter-workgroup execution model
    • “A Study of Persistent Threads Style GPU Programming for GPGPU Workloads”, PIPC’12 Gupta et al.

• GPU watchdog
Going forward

• Programming tools

  • Data-race checkers
    • GPUVerify “The Design and Implementation of a Verification Technique for GPU Kernels”, TOPLAS’15, Betts et al.

  • Dynamic analysis tools
    • OCLGrind “Oclgrind: an extensible OpenCL device simulator”, IWOCL’15, Price and McIntosh-Smith
Conclusions

• Most applications were able to run cross-platform!

• Many portability challenges

• We believe that as a community we can overcome these challenges for a more portable OpenCL world!
Thank You

• Assessed the OpenCL portability evaluation in research
  • Surveyed 50 most recent OpenCL papers

• Found portability issues across 8 GPUs (4 Vendors)
  • 3 framework bugs, 6 specification limitations, 3 Programming Bugs

• Suggested ways to improve OpenCL portability
  • Conformance tests, specification clarifications, testing/verification tools

Tyler Sorensen  
http://www.doc.ic.ac.uk/~tsorensen/  

Alastair Donaldson  
http://multicore.doc.ic.ac.uk/
Specification limitations

**#4 Floating point accuracy**

*Application*: LonestarGPU DMR

32 bit floating point application *successful* on Intel
Specification limitations

#4 Floating point accuracy

**Application:** LonestarGPU DMR

32 bit floating point application **successful** on Intel

32 bit floating point application **fails** on Nvidia
## Specification limitations

### #5 OS portability

<table>
<thead>
<tr>
<th>Chip</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon R9</td>
<td>✓</td>
<td>🐜</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>✓</td>
<td>🐜</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
### #5 OS portability

<table>
<thead>
<tr>
<th>Chip</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon R9</td>
<td>✓</td>
<td>✟</td>
</tr>
<tr>
<td>Radeon R7</td>
<td>✓</td>
<td>✟</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Defunct process bug
## Specification limitations

### #5 OS portability

<table>
<thead>
<tr>
<th>Chip</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon R9</td>
<td><img src="checkmark.png" alt="Check" /></td>
<td><img src="bug.png" alt="Bug" /></td>
</tr>
<tr>
<td>Radeon R7</td>
<td><img src="checkmark.png" alt="Check" /></td>
<td><img src="bug.png" alt="Bug" /></td>
</tr>
<tr>
<td>Mali-T628</td>
<td><img src="x-mark.png" alt="X" /></td>
<td><img src="checkmark.png" alt="Check" /></td>
</tr>
<tr>
<td>Mali-T628</td>
<td><img src="x-mark.png" alt="X" /></td>
<td><img src="checkmark.png" alt="Check" /></td>
</tr>
</tbody>
</table>

Thus entire OpenCL application (device and host) must be cross platform.
Specification limitations

#1 Memory allocation failures

Platforms: Intel

Host memory allocations can cause device memory allocations to fail

Due to fragmentation
Specification limitations

#3 Memory consistency

OpenCL 2.0 atomics allow synchronisation idioms
# Specification limitations

## #3 Memory consistency

OpenCL 2.0 atomics allow synchronisation idioms

<table>
<thead>
<tr>
<th>Chip</th>
<th>OpenCL Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>1.1</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>1.1</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>1.2</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>1.2</td>
</tr>
</tbody>
</table>

No support for OpenCL 2.0!
Specification limitations

#3 Memory consistency

Implement our own atomic operations

```c
typedef int atomic_int;

void atomic_store(atomic_int *addr, int val) {
    mem_fence()
    *addr = val;
    mem_fence()
}
```
Specification limitations

#3 Memory consistency

These chips passed our memory consistency unit tests

<table>
<thead>
<tr>
<th>Chip</th>
<th>OpenCL Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980</td>
<td>1.1</td>
</tr>
<tr>
<td>Quadro K500</td>
<td>1.1</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>1.2</td>
</tr>
<tr>
<td>Mali-T628</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Specification limitations

#3 Memory consistency

Several other (older) chips did not

<table>
<thead>
<tr>
<th>Chip</th>
<th>Vendor</th>
<th>OpenCL Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 480</td>
<td>Nvidia</td>
<td>1.1</td>
</tr>
<tr>
<td>Tesla C2075</td>
<td>Nvidia</td>
<td>1.1</td>
</tr>
<tr>
<td>HD 4400</td>
<td>Intel</td>
<td>1.2</td>
</tr>
<tr>
<td>Radeon HD 7970</td>
<td>AMD</td>
<td>1.2</td>
</tr>
<tr>
<td>Radeon HD 6570</td>
<td>AMD</td>
<td>1.2</td>
</tr>
</tbody>
</table>
# Specification limitations

## #3 Memory consistency

Several other (older) chips did not

<table>
<thead>
<tr>
<th>Chip</th>
<th>Vendor</th>
<th>OpenCL Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 480</td>
<td>Nvidia</td>
<td>1.1</td>
</tr>
<tr>
<td>Tesla C2075</td>
<td>Nvidia</td>
<td>1.1</td>
</tr>
<tr>
<td>HD 4400</td>
<td>Intel</td>
<td>1.2</td>
</tr>
<tr>
<td>Radeon HD 7970</td>
<td>AMD</td>
<td>1.2</td>
</tr>
<tr>
<td>Radeon HD 6570</td>
<td>AMD</td>
<td>1.2</td>
</tr>
</tbody>
</table>

We did not consider these chips further
Programming bugs

#2 Stability

*Application*: LonestarGPU DMR

execute application repeatedly

DRM()

Quadro K5200 (Nvidia)
Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

DRM (

Quadro K5200 (Nvidia)

occasional failures (known by developer and deemed acceptable)

Due to floating point precision
Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

DRM ()

Radeon R9 (AMD)
Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

Fails nearly every iteration on AMD chips

Radeon R9 (AMD)