The Hitchhiker's Guide to Cross-Platform OpenCL Application Development

Tyler Sorensen and Alastair F. Donaldson

Imperial College London, UK

IWOCL

April 2016

“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

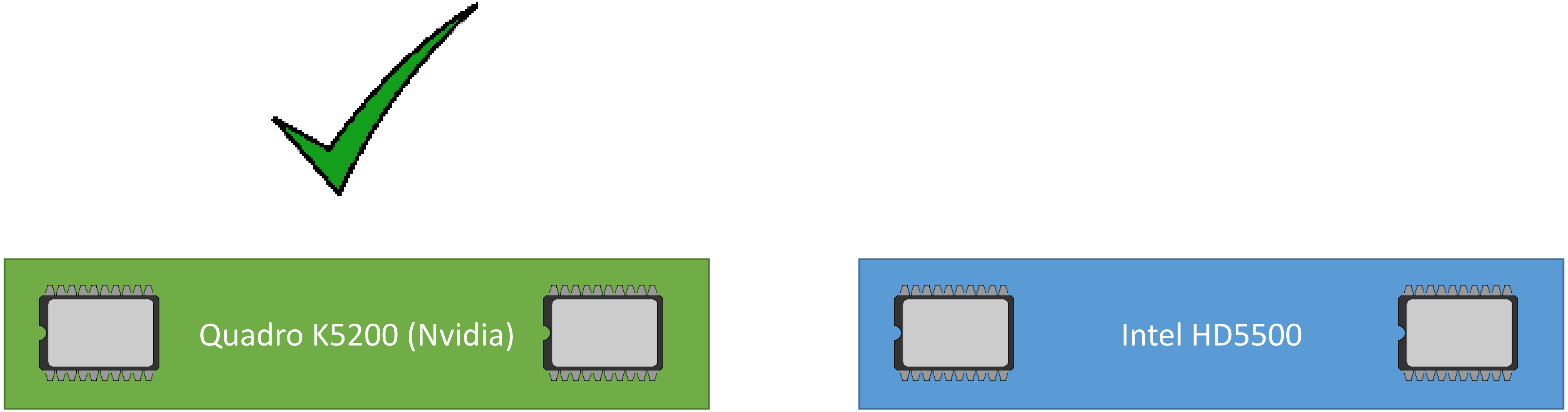
We consider functional portability rather than performance portability

Example

- single source shortest path application

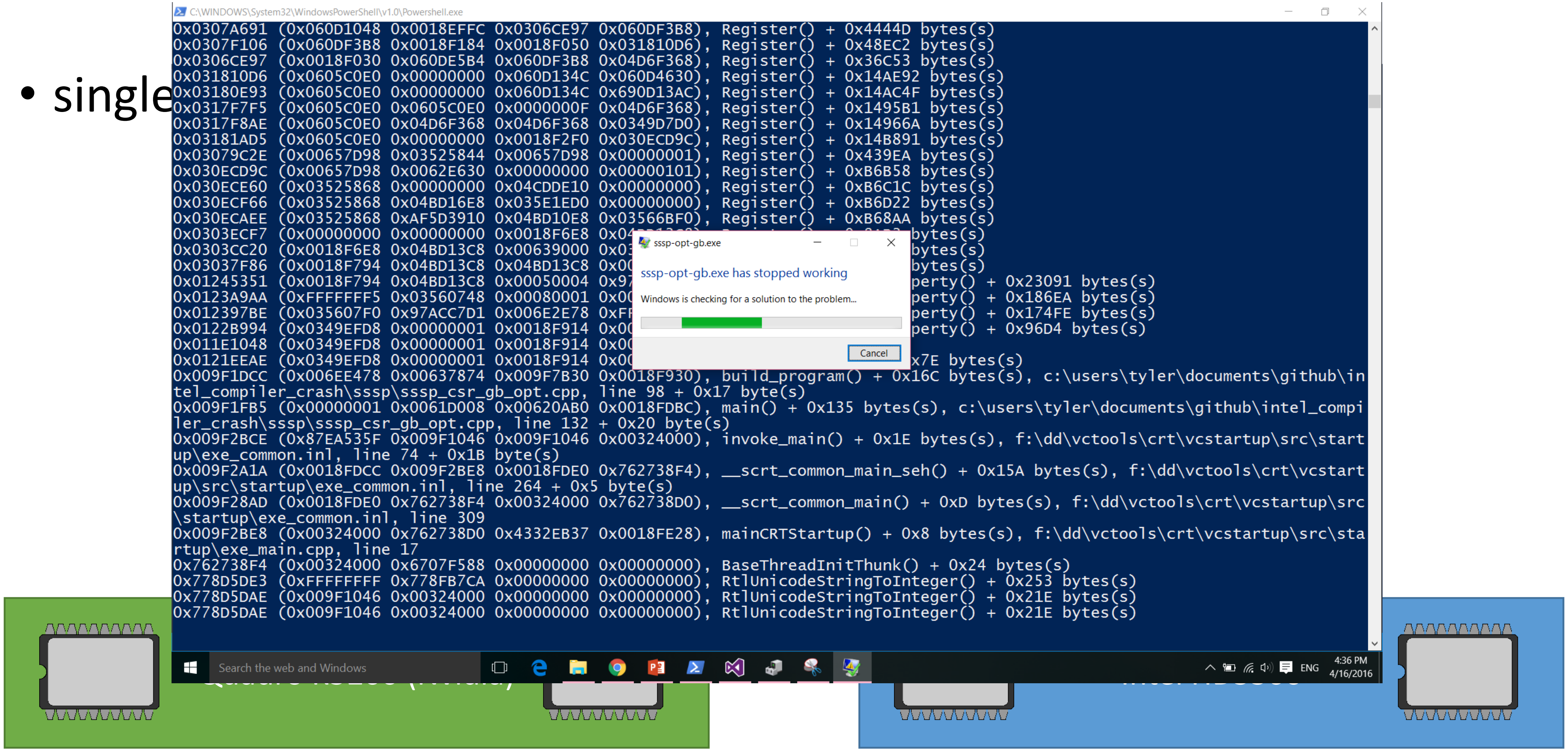
Example

- single source shortest path application



Example

- single



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?

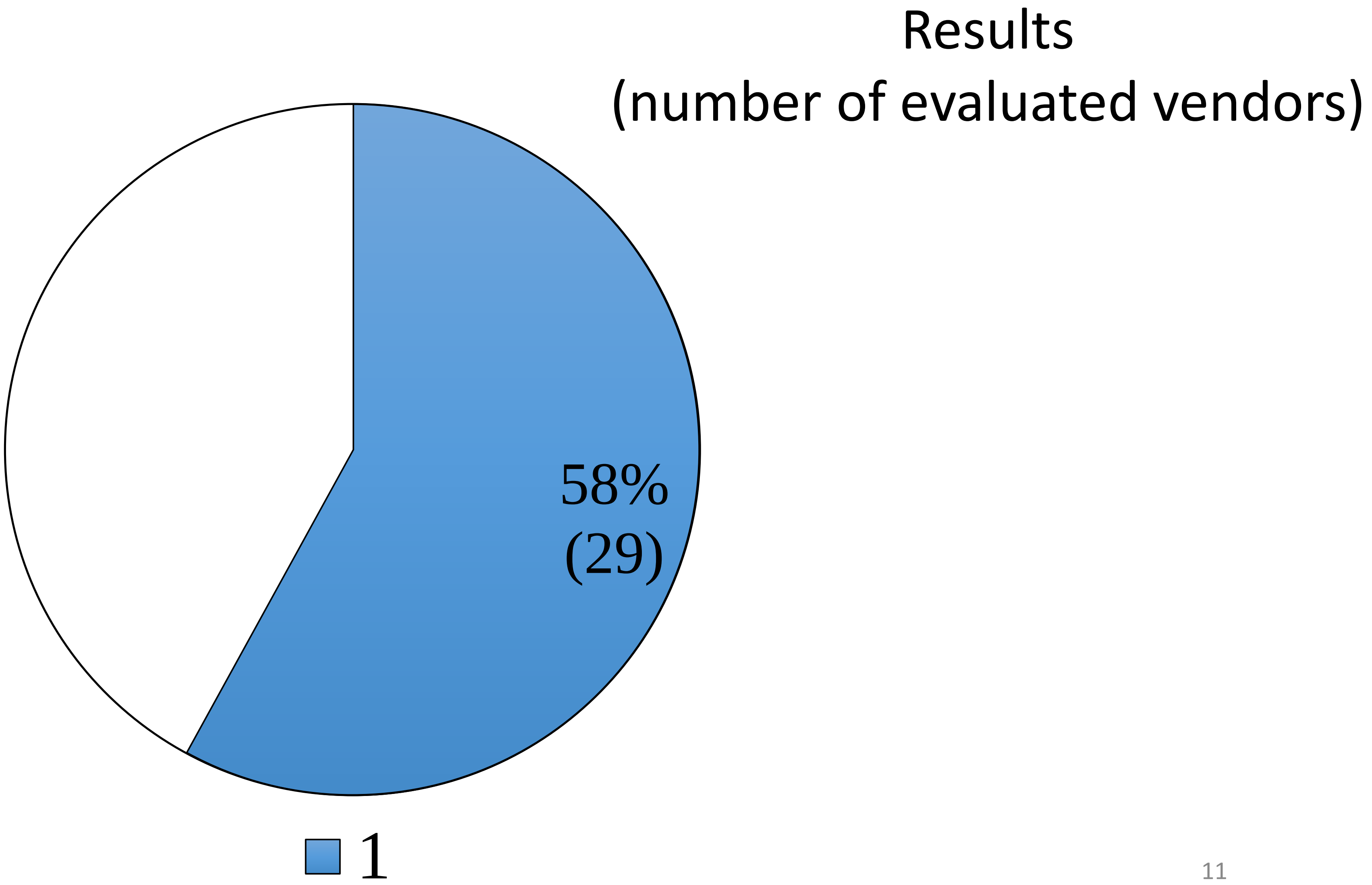
- Our experience running applications on 8 GPUs spanning 4 vendors

- Recommendations going forward

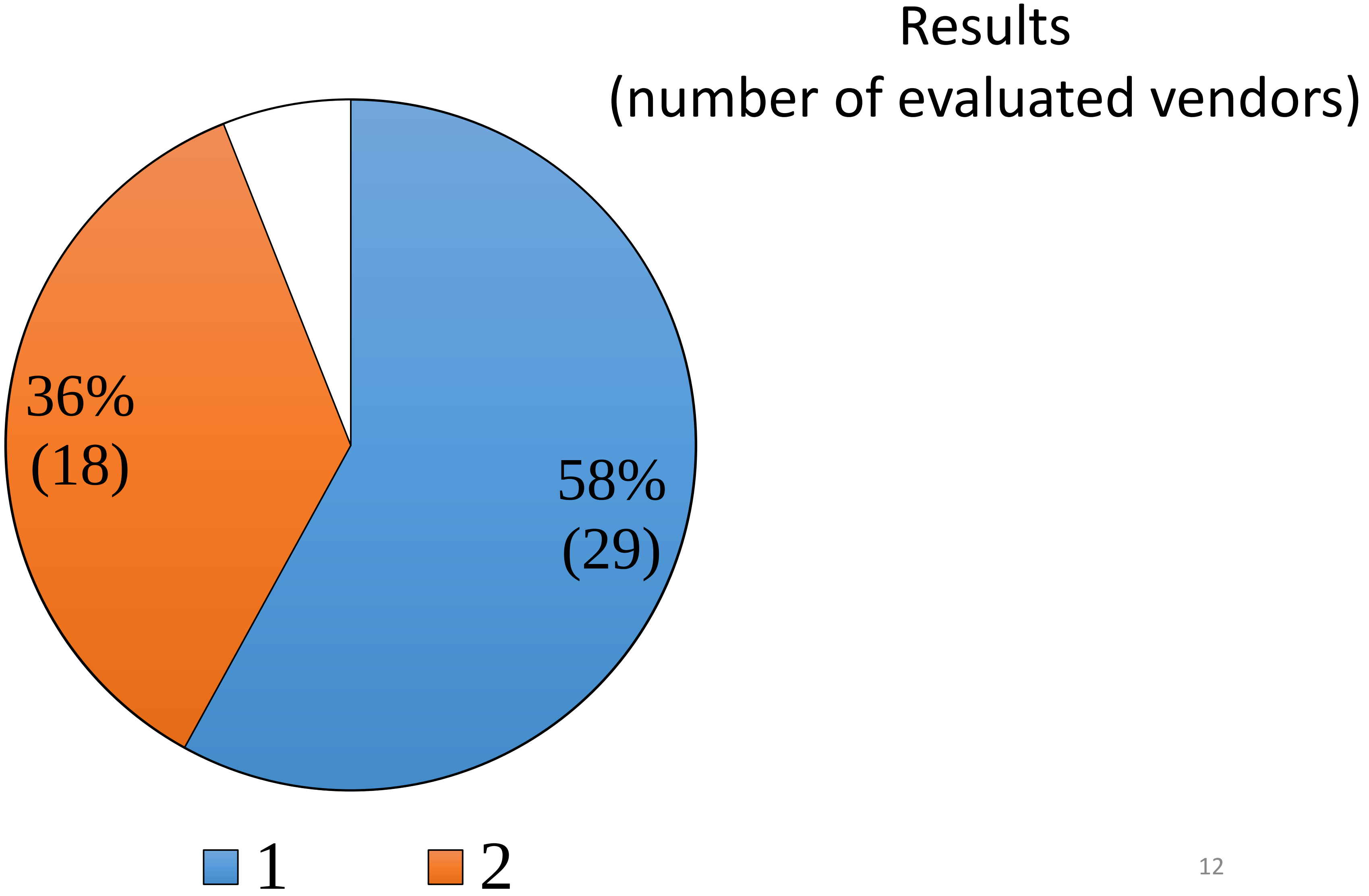
Portability in research literature

- 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?

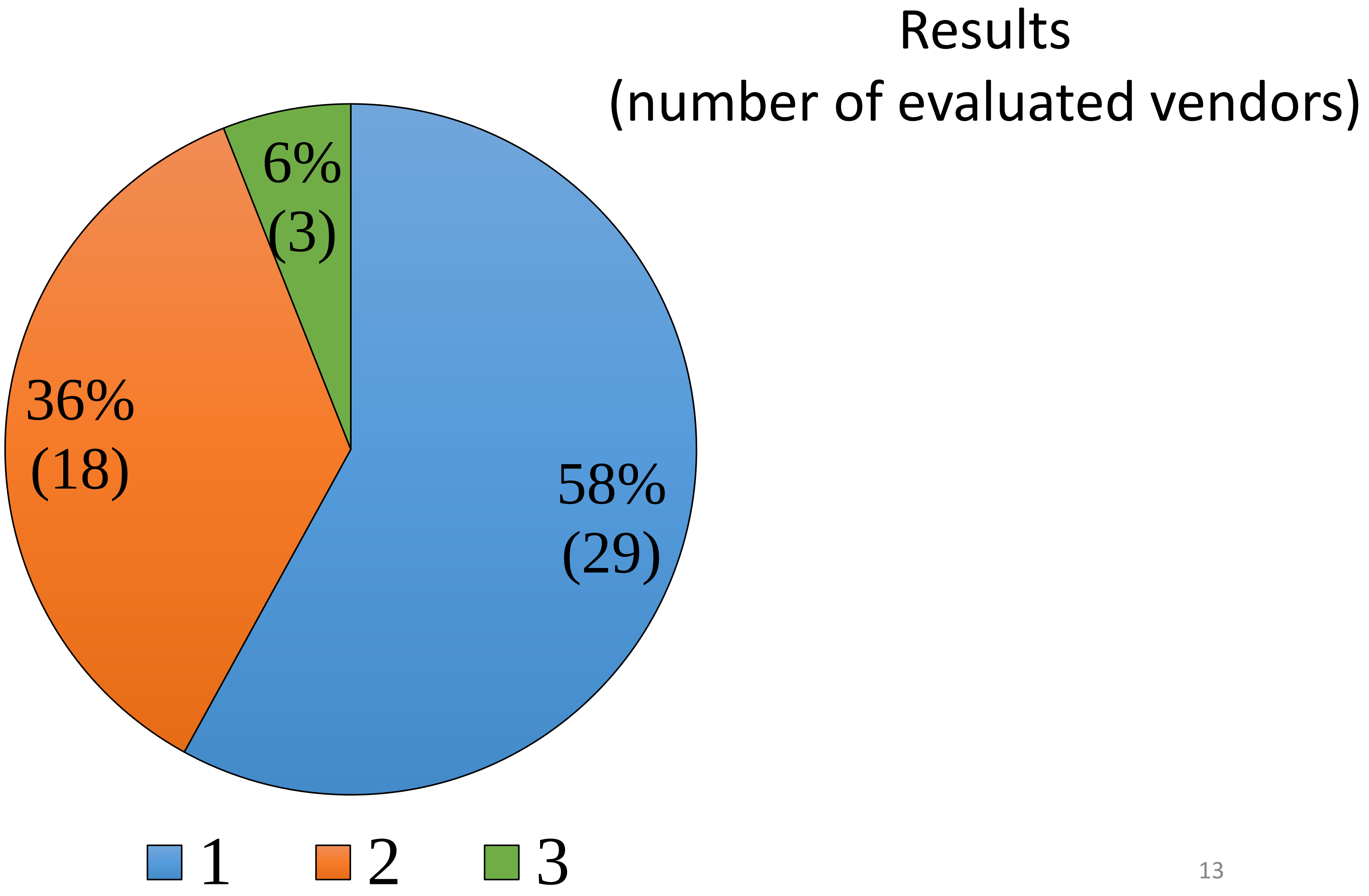
Portability in research literature



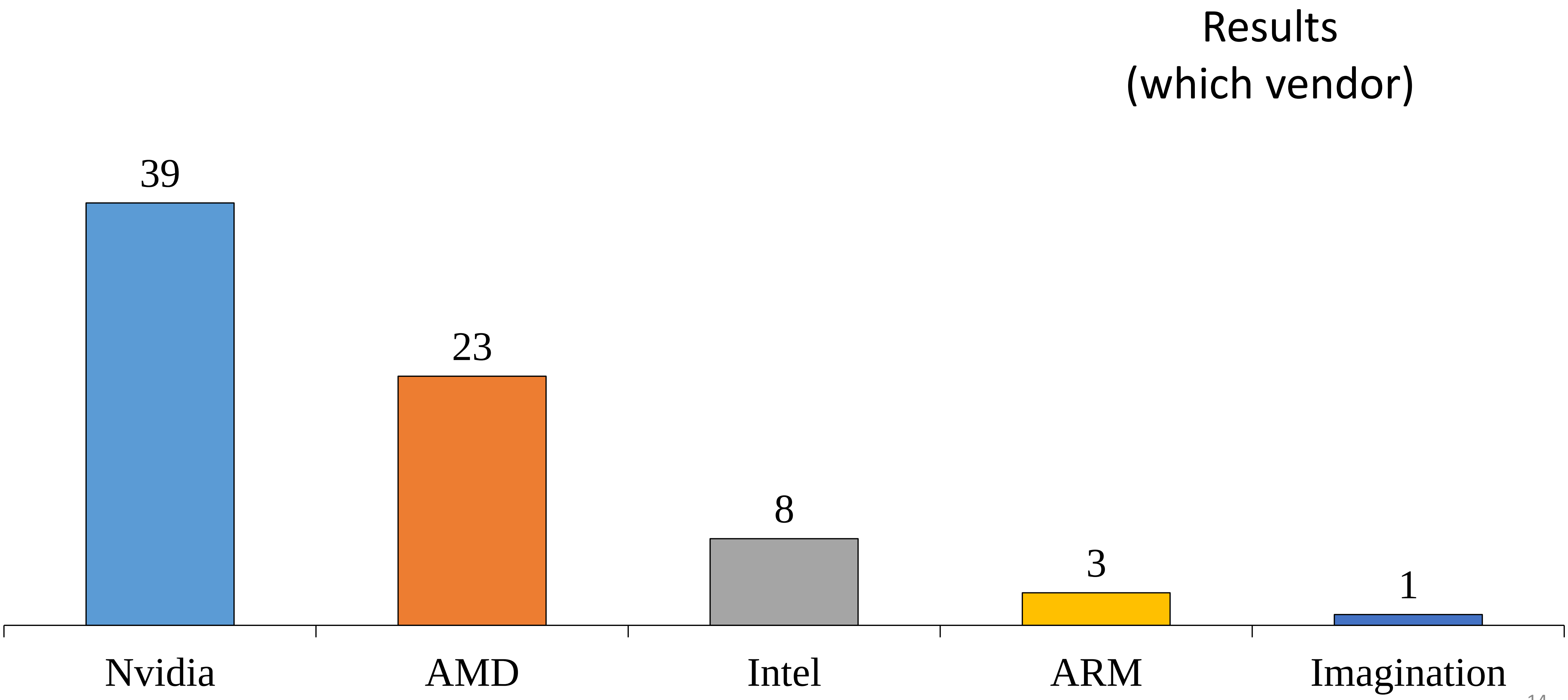
Portability in research literature



Portability in research literature

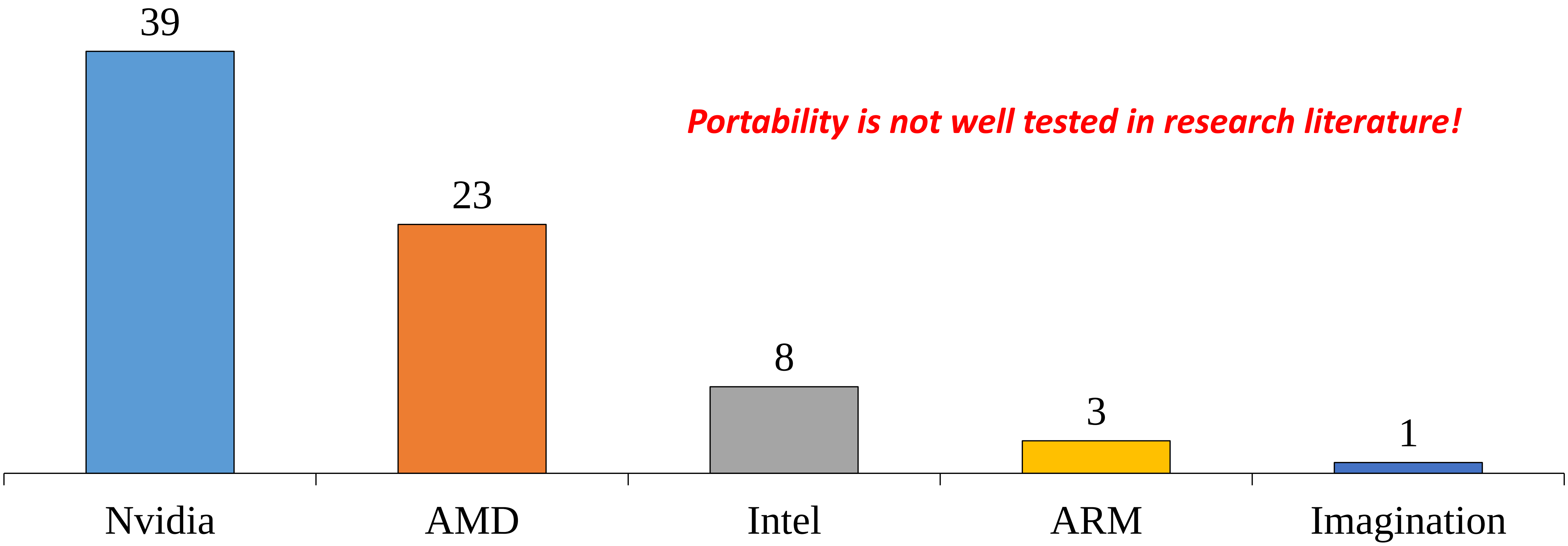


Portability in research literature



Portability in research literature

Results
(which vendor)



Portability is not well tested in research literature!

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

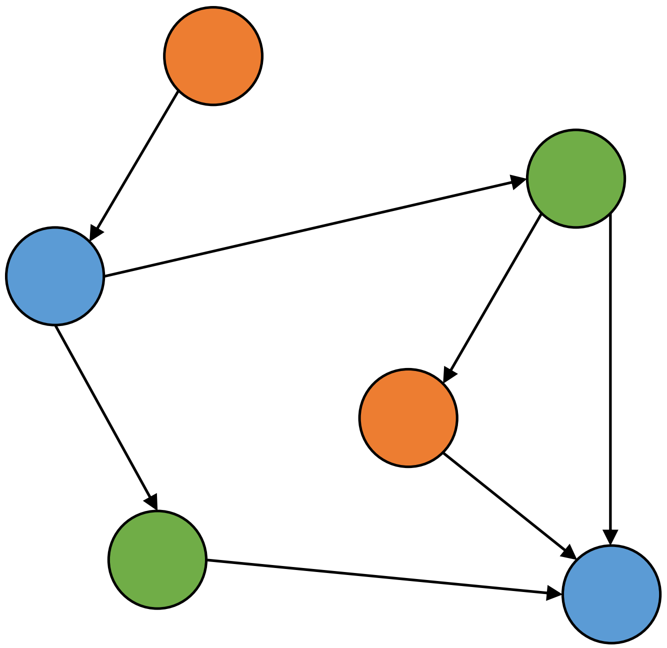
Chip	Vendor	Compute Units	OpenCL Version	Type
GTX 980	Nvidia	16	1.1	Discrete
Quadro K500	Nvidia	12	1.1	Discrete
Iris 6100	Intel	47	2.0	Integrated
HD 5500	Intel	24	2.0	Integrated
Radeon R9	AMD	28	2.0	Discrete
Radeon R7	AMD	8	2.0	Integrated
Mali-T628	ARM	4	1.2	Integrated
Mali-T628	ARM	2	1.2	integrated

Applications

- Part of a larger study on GPU irregular parallelism

https://github.com/pannotia/pannotia

Applications



Pannotia

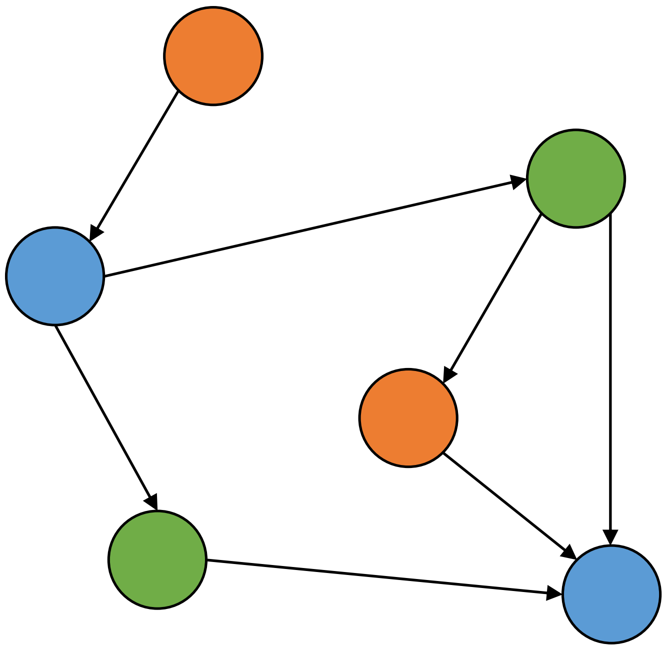
- Target AMD Radeon HD 7000
- Written in OpenCL 1.x
- 4 graph algorithms applications

https://github.com/pannotia/pannotia

Applications

Pannotia

- Target AMD Radeon HD 7000
- Written in OpenCL 1.x
- 4 graph algorithms applications

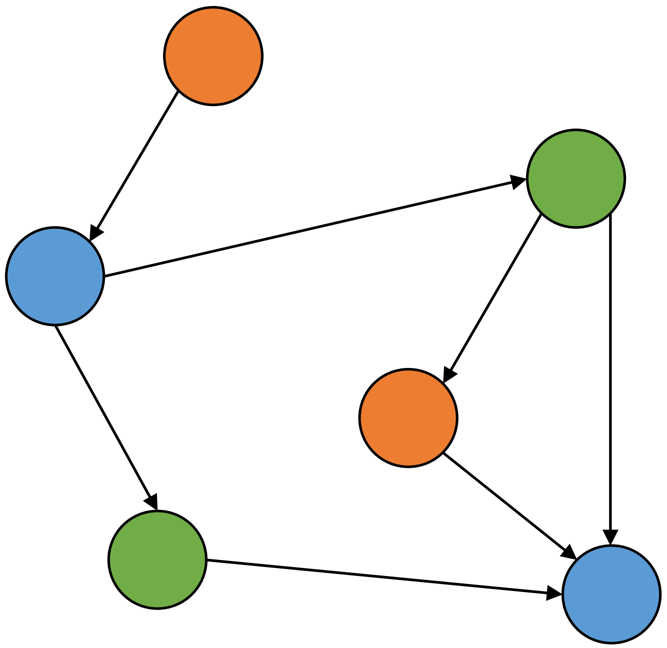


```
GPU_linear_algebra_routine1;
GPU_linear_algebra_routine2;
GPU_linear_algebra_routine3;
```

Loop until a fixed point is reached.

https://github.com/pannotia/pannotia

Applications



LonestarGPU

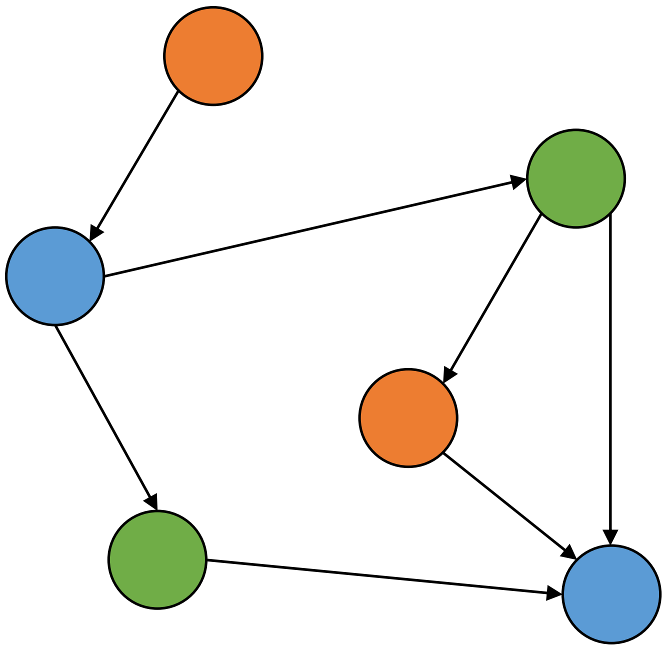
- Target Nvidia Kepler and Fermi
- Written in CUDA
- 4 graph algorithms applications

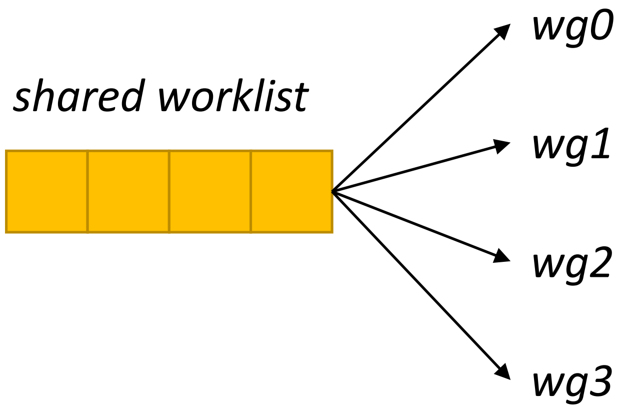
http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu

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

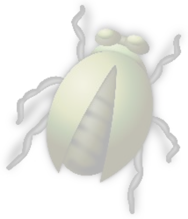
- 3 Framework bugs

- 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

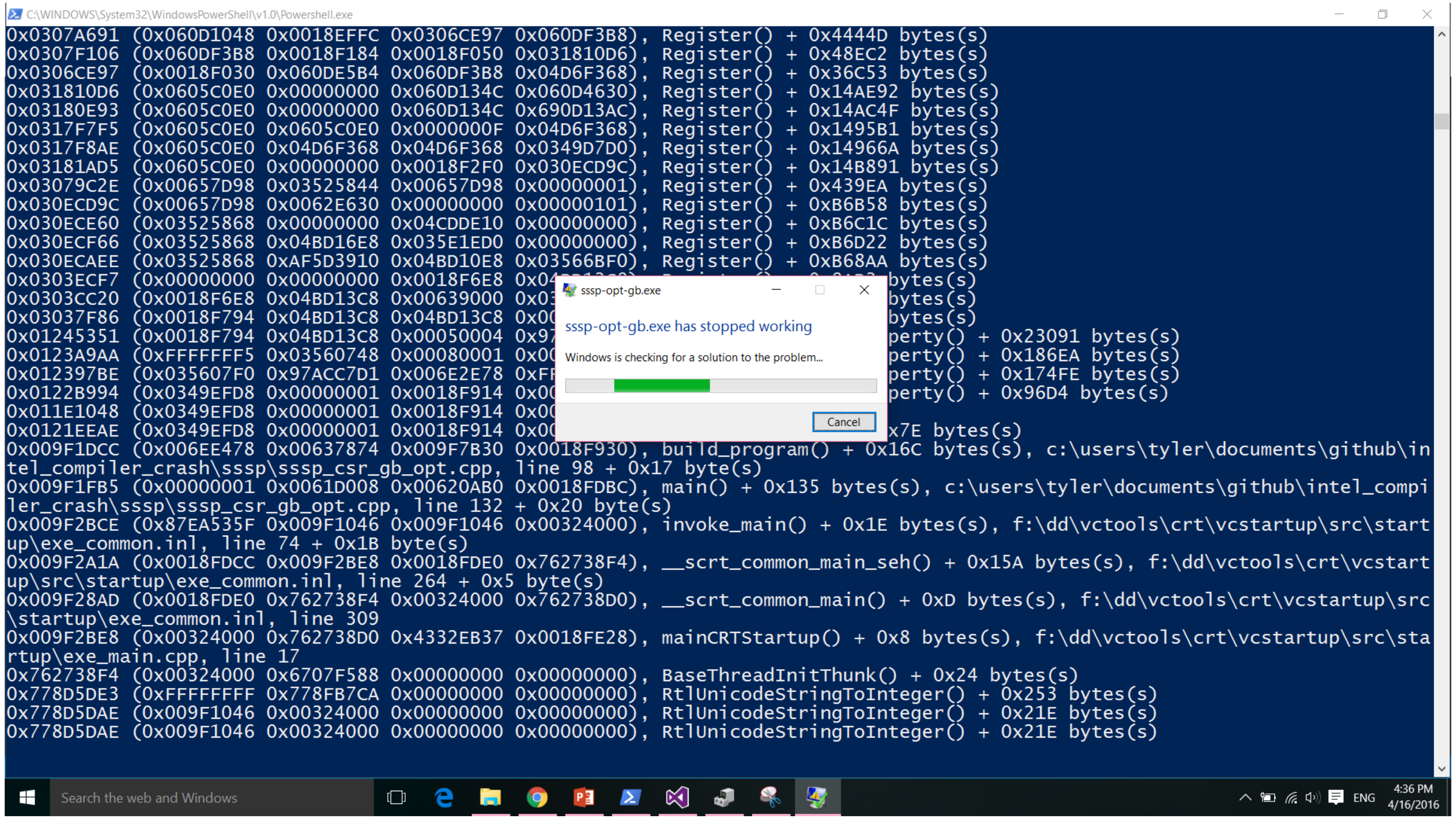
#1 Compiler crash

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

This looping idiom used in kernel code

Platforms: Nvidia and AMD

```
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

This looping idiom used in kernel code

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

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

    if (!more_work)
        break;
}
```

Does not terminate on Nvidia and AMD platforms!!

Framework bugs

This looping idiom used in kernel code

#2 Non-terminating loops

Platforms: Nvidia and AMD

```
~~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;
}
```

Change `while` loop to `for` loop

End value of `i` is consistent across platforms

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

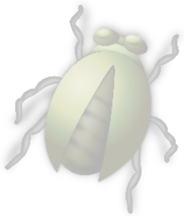
Portability Issues

12 issues encountered, grouped into categories

- 3 Framework bugs

- 6 Specification limitations

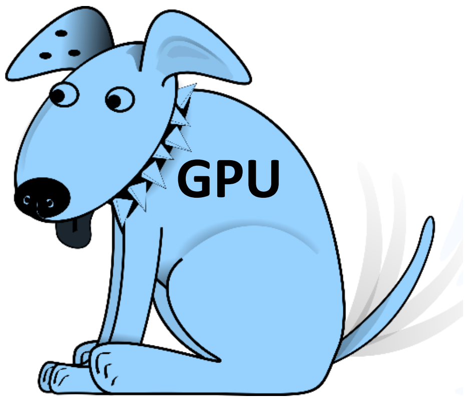
- 3 Programming bugs



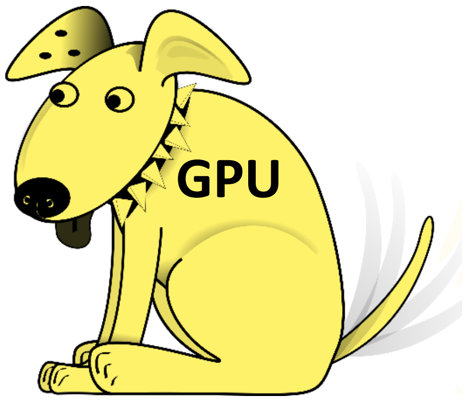
Specification limitations

#1 GPU watchdogs

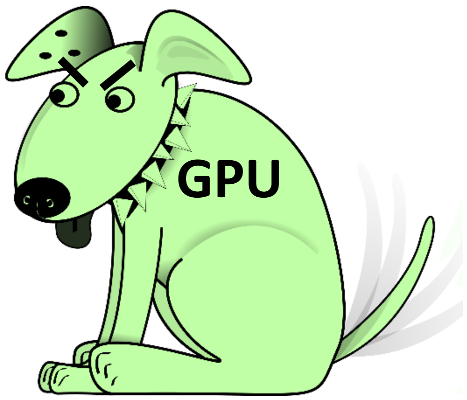
Platforms and operating systems handle watchdogs differently.



Windows



Linux (Ubuntu)



Chrome OS

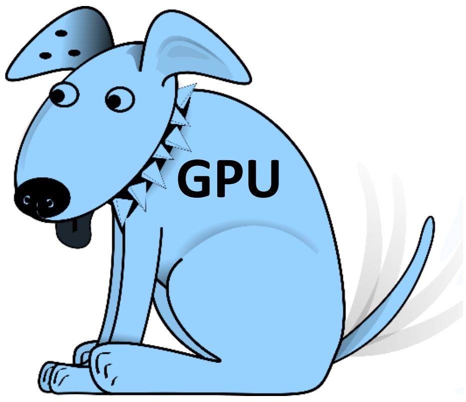
Specification limitations

#1 GPU watchdogs

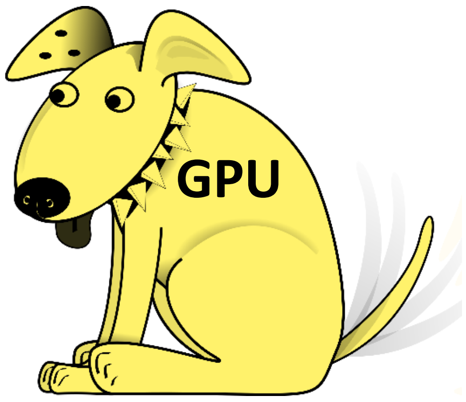
Platforms and operating systems handle watchdogs differently.

Controlled with registry

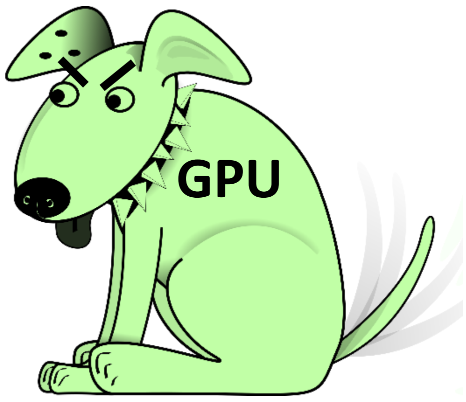
Watchdog kills entire
OpenCL process



Windows



Linux (Ubuntu)



Chrome OS

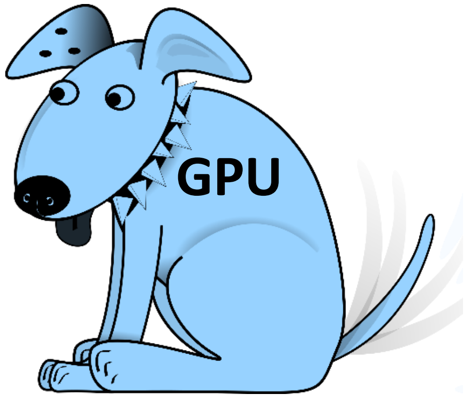
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

Controlled with registry

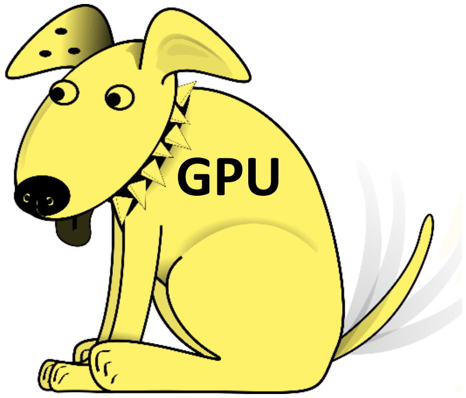
Watchdog kills entire
OpenCL process



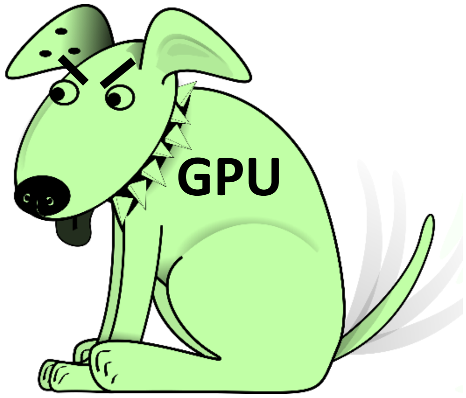
Windows

Controlled in X server settings

Watchdog only kills kernel



Linux (Ubuntu)



Chrome OS

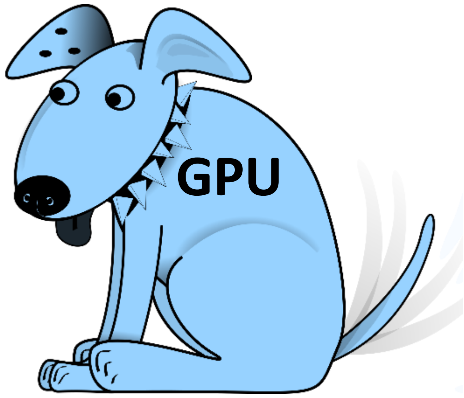
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

Controlled with registry

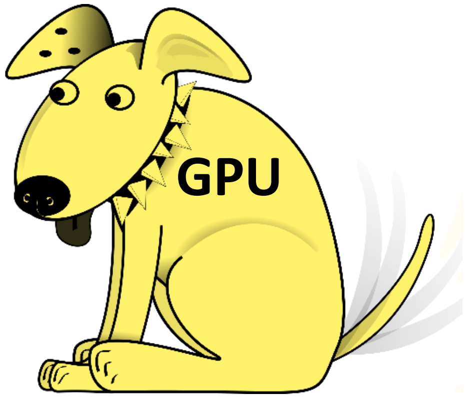
Watchdog kills entire OpenCL process



Windows

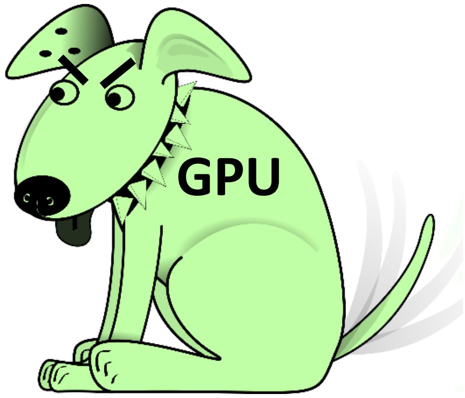
Controlled in X server settings

Watchdog only kills kernel



Linux (Ubuntu)

Cannot control at all without recompiling the driver



Chrome OS

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

Specification limitations

#2 Occupancy vs compute units

chip	compute units	PT occupancy
GTX 980	16	
Quadro K500	12	
Iris 6100	47	
HD 5500	24	
Radeon R9	28	
Radeon R7	8	
Mali-T628	4	
Mali-T628	2	

Compute units are safe and optimal

Specification limitations

#2 Occupancy vs compute units

chip	compute units	PT occupancy
GTX 980	16	
Quadro K500	12	12
Iris 6100	47	
HD 5500	24	
Radeon R9	28	
Radeon R7	8	
Mali-T628	4	4
Mali-T628	2	2

Specification limitations

Compute units are safe and optimal

Compute units are safe but not optimal

#2 Occupancy vs compute units

chip	compute units	PT occupancy
GTX 980	16	32
Quadro K500	12	12
Iris 6100	47	
HD 5500	24	
Radeon R9	28	48
Radeon R7	8	16
Mali-T628	4	4
Mali-T628	2	2

Specification limitations

Compute units are safe and optimal

Compute units are safe but not optimal

Compute units are not safe

#2 Occupancy vs compute units

chip	compute units	PT occupancy
GTX 980	16	32
Quadro K500	12	12
Iris 6100	47	6
HD 5500	24	3
Radeon R9	28	48
Radeon R7	8	16
Mali-T628	4	4
Mali-T628	2	2

Portability Issues

12 issues encountered, grouped into categories

- 3 Framework bugs

- 6 Specification limitations

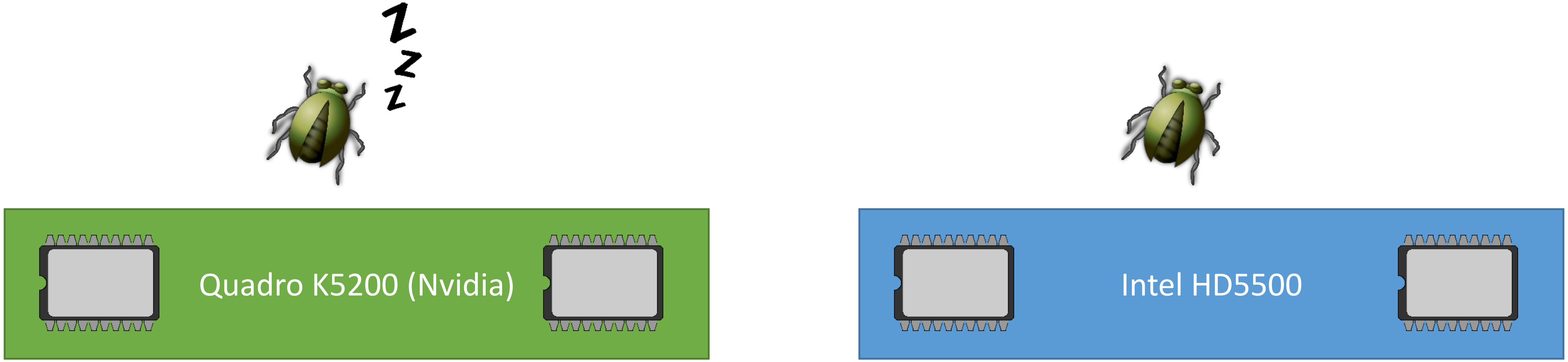
- 3 Programming bugs

Programming bugs

#1 Data-races

Application: LonestarGPU bfs and sssp

Fix: Add additional synchronisation barriers



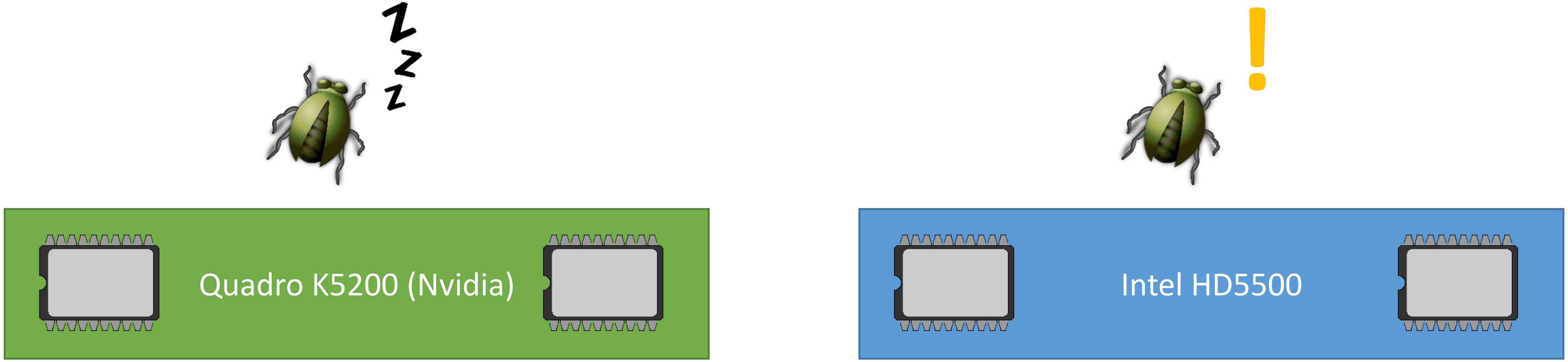
Programming bugs

#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

```
clSetKernelArg (bfs_kernel, 0,
        sizeof(Graph), &graph1);
 // Execute bfs kernel
```

Chip
GTX 980
Quadro K500
Iris 6100
HD 5500
Radeon R9
Radeon R7
Mali-T628
Mali-T628

Programming bugs

#2 Struct kernel arguments

```
clSetKernelArg (bfs_kernel, 0,
          sizeof(Graph), &graph1);
 // Execute bfs kernel
```

Chip	
GTX 980	✓
Quadro K500	✓
Iris 6100	✓
HD 5500	✓
Radeon R9	✓
Radeon R7	✓
Mali-T628	✗
Mali-T628	✗

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

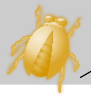
#5 OS portability

Chip	Windows	Linux
Radeon R9	✓	
Radeon R7	✓	
Mali-T628	✗	✓
Mali-T628	✗	✓

Specification limitations

#5 OS portability

Defunct process bug

Chip	Windows	Linux
Radeon R9	✓	
Radeon R7	✓	
Mali-T628	✗	✓
Mali-T628	✗	✓

Specification limitations

#5 OS portability

Defunct process bug

Chip	Windows	Linux
Radeon R9	✓	🐛
Radeon R7	✓	🐛
Mali-T628	✗	✓
Mali-T628	✗	✓

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

Chip	OpenCL Version
GTX 980	1.1
Quadro K500	1.1
Mali-T628	1.2
Mali-T628	1.2

No support for OpenCL 2.0!

Specification limitations

#3 Memory consistency

Implement our own atomic operations

```
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

Chip	OpenCL Version
GTX 980	1.1
Quadro K500	1.1
Mali-T628	1.2
Mali-T628	1.2

Specification limitations

#3 Memory consistency

Several other (older) chips did not

Chip	Vendor	OpenCL Version
GTX 480	Nvidia	1.1
Tesla C2075	Nvidia	1.1
HD 4400	Intel	1.2
Radeon HD 7970	AMD	1.2
Radeon HD 6570	AMD	1.2

Specification limitations

#3 Memory consistency

We did not consider these chips further

Several other (older) chips did not

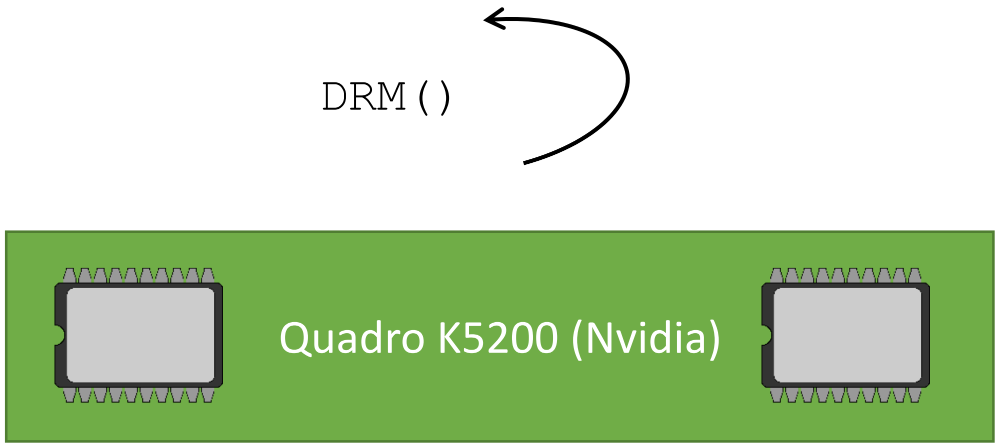
Chip	Vendor	OpenCL Version
GTX 480	Nvidia	1.1
Tesla C2075	Nvidia	1.1
HD 4400	Intel	1.2
Radeon HD 7970	AMD	1.2
Radeon HD 6570	AMD	1.2

Programming bugs

#2 Stability

Application: LonestarGPU DMR

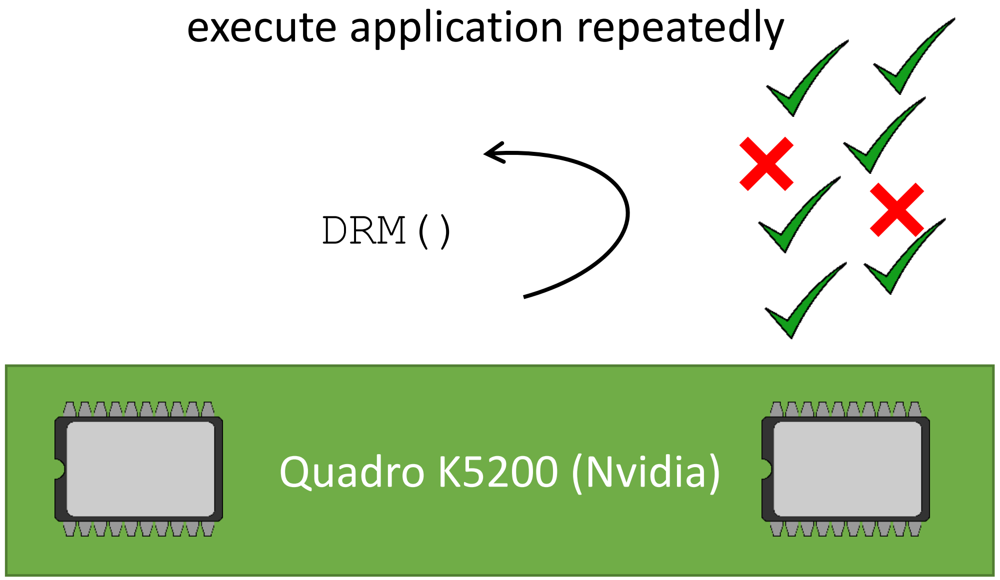
execute application repeatedly



Programming bugs

#2 Stability

Application: LonestarGPU DMR



occasional failures
(known by developer
and deemed acceptable)

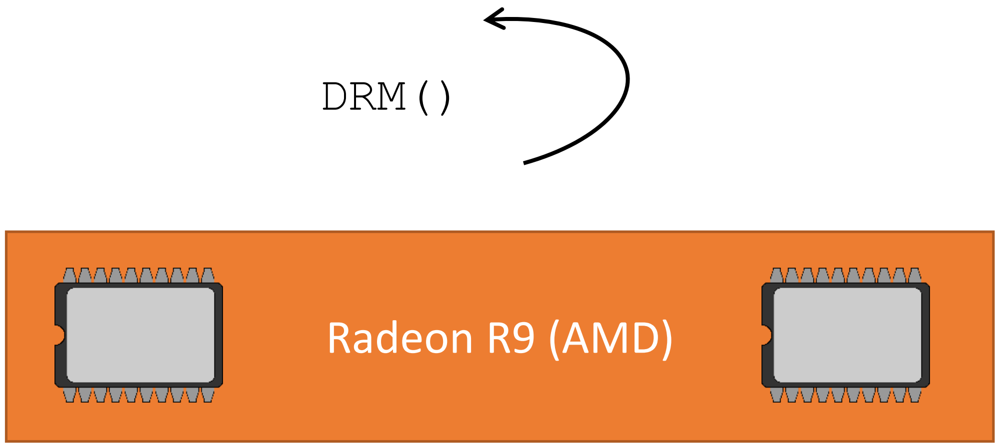
Due to floating point precision

Programming bugs

#2 Stability

Application: LonestarGPU DMR

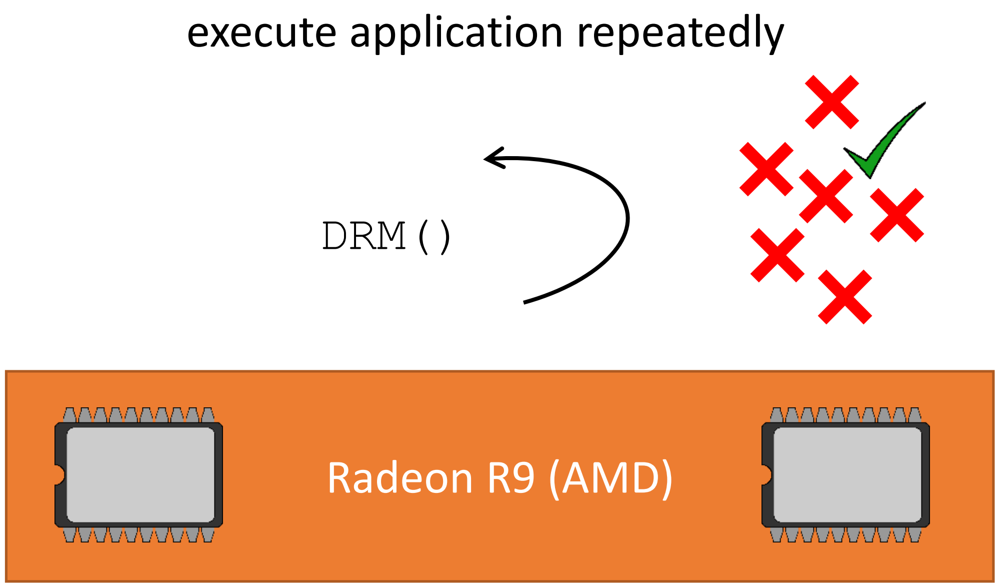
execute application repeatedly



Programming bugs

#2 Stability

Application: LonestarGPU DMR



Fails nearly every iteration on AMD chips