

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

Tyler Sorensen (led the work and made the slides)

Alastair F. Donaldson (delivered this version of the talk)

Imperial College London, UK

UKMAC

May 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

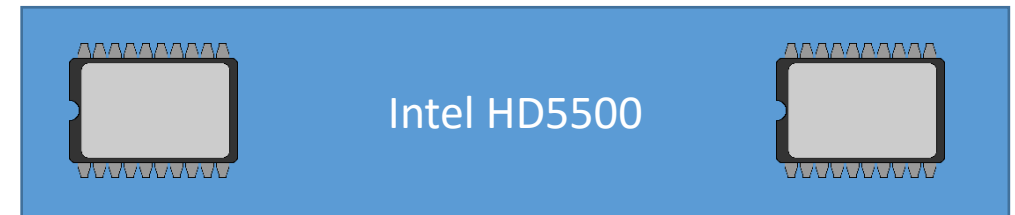
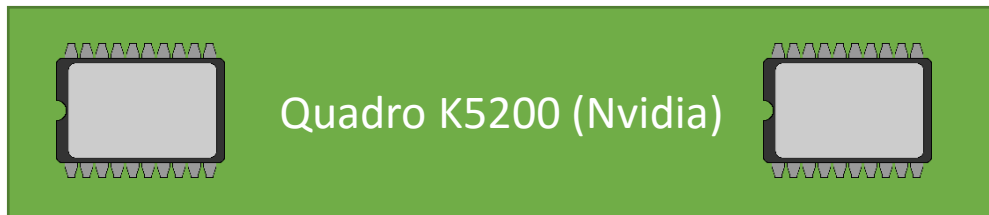
*“OpenCL supports a wide range of applications... through a low-level, high-performance, **portable** abstraction.”*

Page 11: OpenCL 2.1 specification

We consider functional portability rather than performance portability

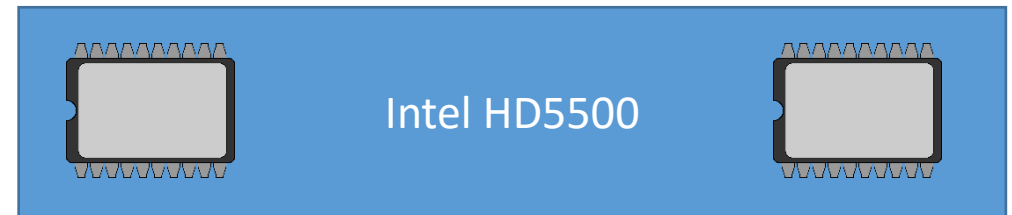
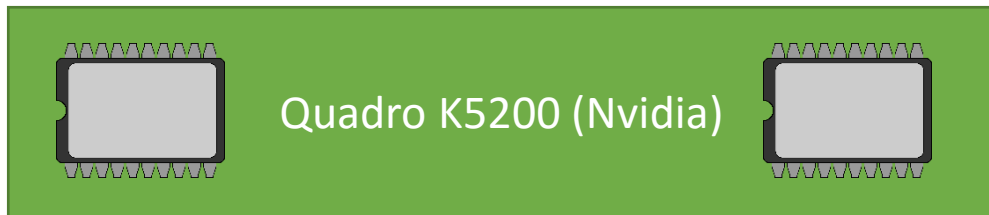
Example

- single source shortest path application



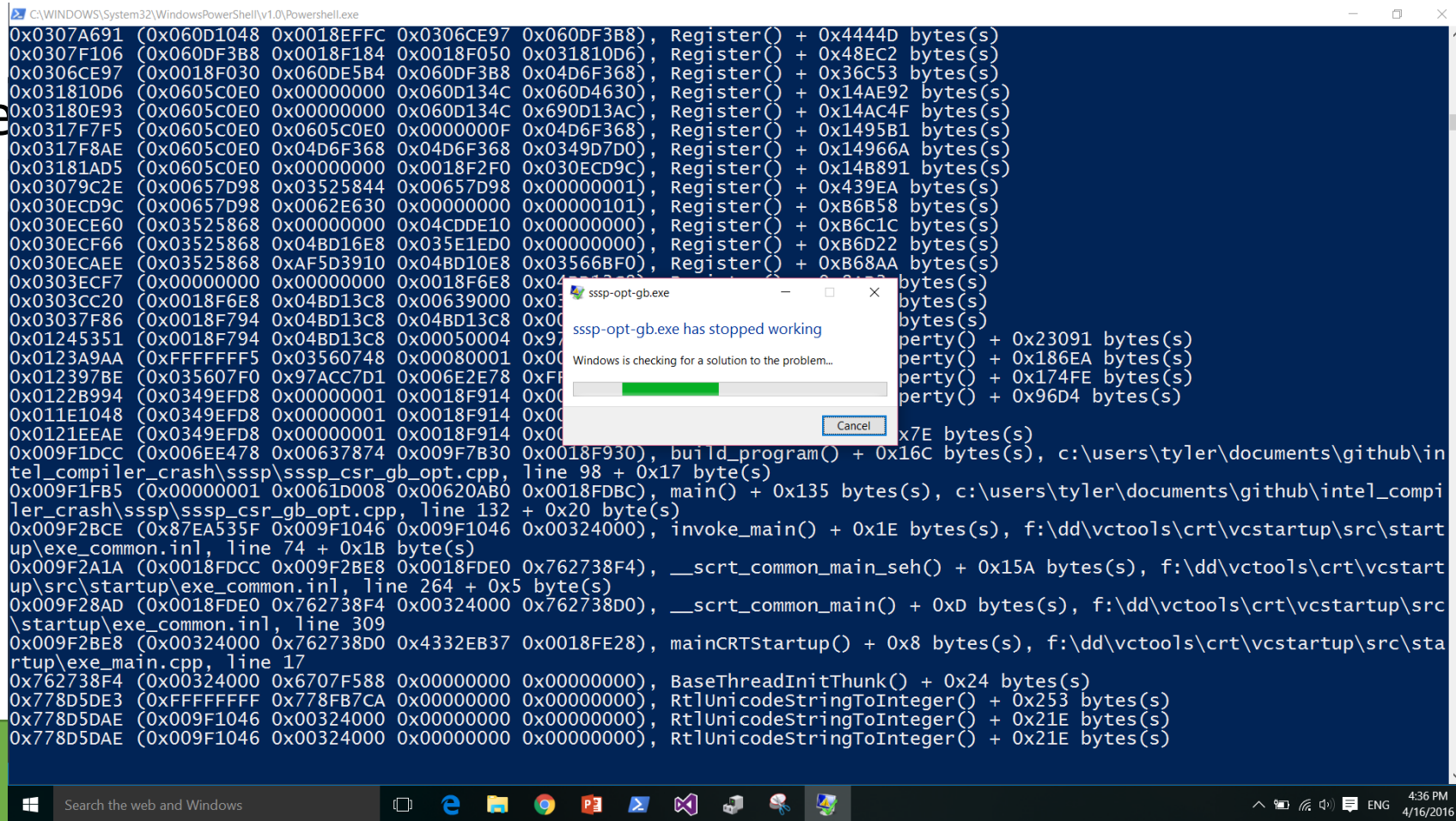
Example

- single source shortest path application



Example

- single



```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\Powershell.exe
0x0307A691 (0x060D1048 0x0018E0FC 0x0306CE97 0x060DF3B8), Register() + 0x4444D bytes(s)
0x0307F106 (0x060DF3B8 0x0018F184 0x0018F050 0x031810D6), Register() + 0x48EC2 bytes(s)
0x0306CE97 (0x0018F030 0x060DE5B4 0x060DF3B8 0x04D6F368), Register() + 0x36C53 bytes(s)
0x031810D6 (0x0605C0E0 0x00000000 0x060D134C 0x060D4630), Register() + 0x14AE92 bytes(s)
0x03180E93 (0x0605C0E0 0x00000000 0x060D134C 0x690D13AC), Register() + 0x14AC4F bytes(s)
0x0317F7F5 (0x0605C0E0 0x0605C0E0 0x0000000F 0x04D6F368), Register() + 0x1495B1 bytes(s)
0x0317F8AE (0x0605C0E0 0x04D6F368 0x04D6F368 0x0349D7D0), Register() + 0x14966A bytes(s)
0x03181AD5 (0x0605C0E0 0x00000000 0x0018F2F0 0x030ECD9C), Register() + 0x14B891 bytes(s)
0x03079C2E (0x00657D98 0x03525844 0x00657D98 0x00000001), Register() + 0x439EA bytes(s)
0x030ECD9C (0x00657D98 0x0062E630 0x00000000 0x00000101), Register() + 0xB6B58 bytes(s)
0x030ECE60 (0x03525868 0x00000000 0x04CDE10 0x00000000), Register() + 0xB6C1C bytes(s)
0x030ECF66 (0x03525868 0x04BD16E8 0x035E1ED0 0x00000000), Register() + 0xB6D22 bytes(s)
0x030ECAEE (0x03525868 0xAF5D3910 0x04BD10E8 0x03566BF0), Register() + 0xB68AA bytes(s)
0x0303ECF7 (0x00000000 0x00000000 0x0018F6E8 0x04D6F368), Register() + 0x1495B1 bytes(s)
0x0303CC20 (0x0018F6E8 0x04BD13C8 0x00639000 0x0018F6E8), Register() + 0x1495B1 bytes(s)
0x03037F86 (0x0018F794 0x04BD13C8 0x04BD13C8 0x0018F794), Register() + 0x1495B1 bytes(s)
0x01245351 (0x0018F794 0x04BD13C8 0x00050004 0x97ACC7D1), Register() + 0x1495B1 bytes(s)
0x0123A9AA (0FFFFFFF5 0x03560748 0x00080001 0x00080001), Register() + 0x1495B1 bytes(s)
0x012397BE (0x035607F0 0x97ACC7D1 0x006E2E78 0xFF), Register() + 0x1495B1 bytes(s)
0x0122B994 (0x0349EFD8 0x00000001 0x0018F914 0x0018F914), Register() + 0x1495B1 bytes(s)
0x011E1048 (0x0349EFD8 0x00000001 0x0018F914 0x0018F914), Register() + 0x1495B1 bytes(s)
0x0121EEAE (0x0349EFD8 0x00000001 0x0018F914 0x0018F914), Register() + 0x1495B1 bytes(s)
0x009F1DCC (0x006EE478 0x00637874 0x009F7B30 0x0018F930), build_program() + 0x16C bytes(s), c:\users\tyler\documents\github\intel_compiler_crash\sssp\sssp_csr_gb_opt.cpp, line 98 + 0x17 byte(s)
0x009F1FB5 (0x00000001 0x0061D008 0x00620AB0 0x0018FDBC), main() + 0x135 bytes(s), c:\users\tyler\documents\github\intel_compiler_crash\sssp\sssp_csr_gb_opt.cpp, line 132 + 0x20 byte(s)
0x009F2BCE (0x87EA535F 0x009F1046 0x009F1046 0x00324000), invoke_main() + 0x1E bytes(s), f:\dd\vctools\crt\vcstartup\src\startup\exe_common.inl, line 74 + 0x1B byte(s)
0x009F2A1A (0x0018FDCC 0x009F2BE8 0x0018FDE0 0x762738F4), __scrt_common_main_seh() + 0x15A bytes(s), f:\dd\vctools\crt\vcstartup\src\startup\exe_common.inl, line 264 + 0x5 byte(s)
0x009F28AD (0x0018FDE0 0x762738F4 0x00324000 0x762738D0), __scrt_common_main() + 0xD bytes(s), f:\dd\vctools\crt\vcstartup\src\startup\exe_common.inl, line 309
0x009F2BE8 (0x00324000 0x762738D0 0x4332EB37 0x0018FE28), mainCRTStartup() + 0x8 bytes(s), f:\dd\vctools\crt\vcstartup\src\startup\exe_main.cpp, line 17
0x762738F4 (0x00324000 0x6707F588 0x00000000 0x00000000), BaseThreadInitThunk() + 0x24 bytes(s)
0x778D5DE3 (0FFFFFFF 0x778FB7CA 0x00000000 0x00000000), RtlUnicodeStringToInteger() + 0x253 bytes(s)
0x778D5DAE (0x009F1046 0x00324000 0x00000000 0x00000000), RtlUnicodeStringToInteger() + 0x21E bytes(s)
0x778D5DAE (0x009F1046 0x00324000 0x00000000 0x00000000), RtlUnicodeStringToInteger() + 0x21E bytes(s)
```

An experience report on OpenCL portability

- How well is portability evaluated?
- Our experience running applications on 8 GPUs spanning 4 vendors
- Recommendations going forward

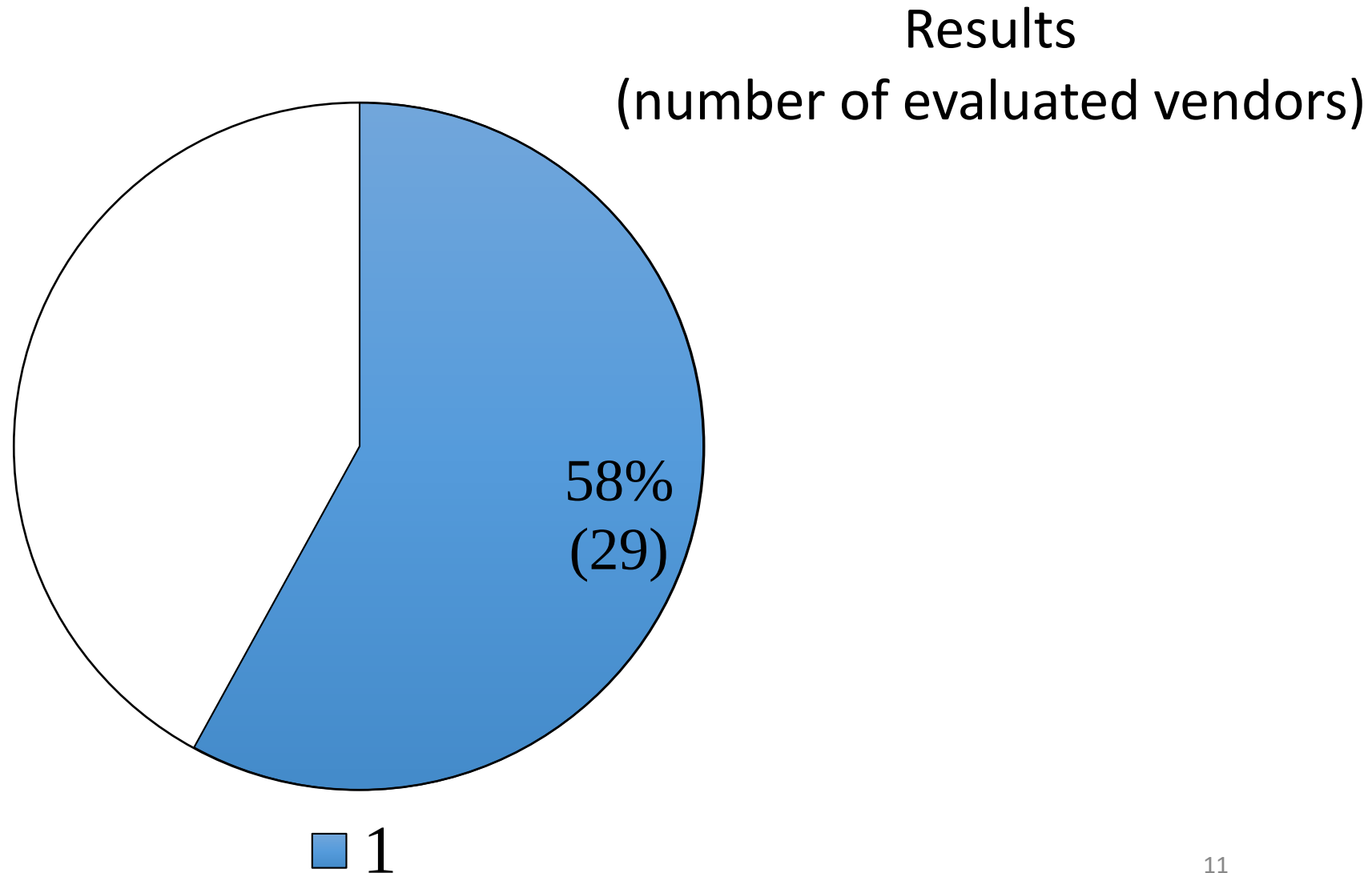
An experience report on OpenCL portability

- How well is portability evaluated?
- Our experience running applications on 8 GPUs spanning 4 vendors
- Recommendations going forward

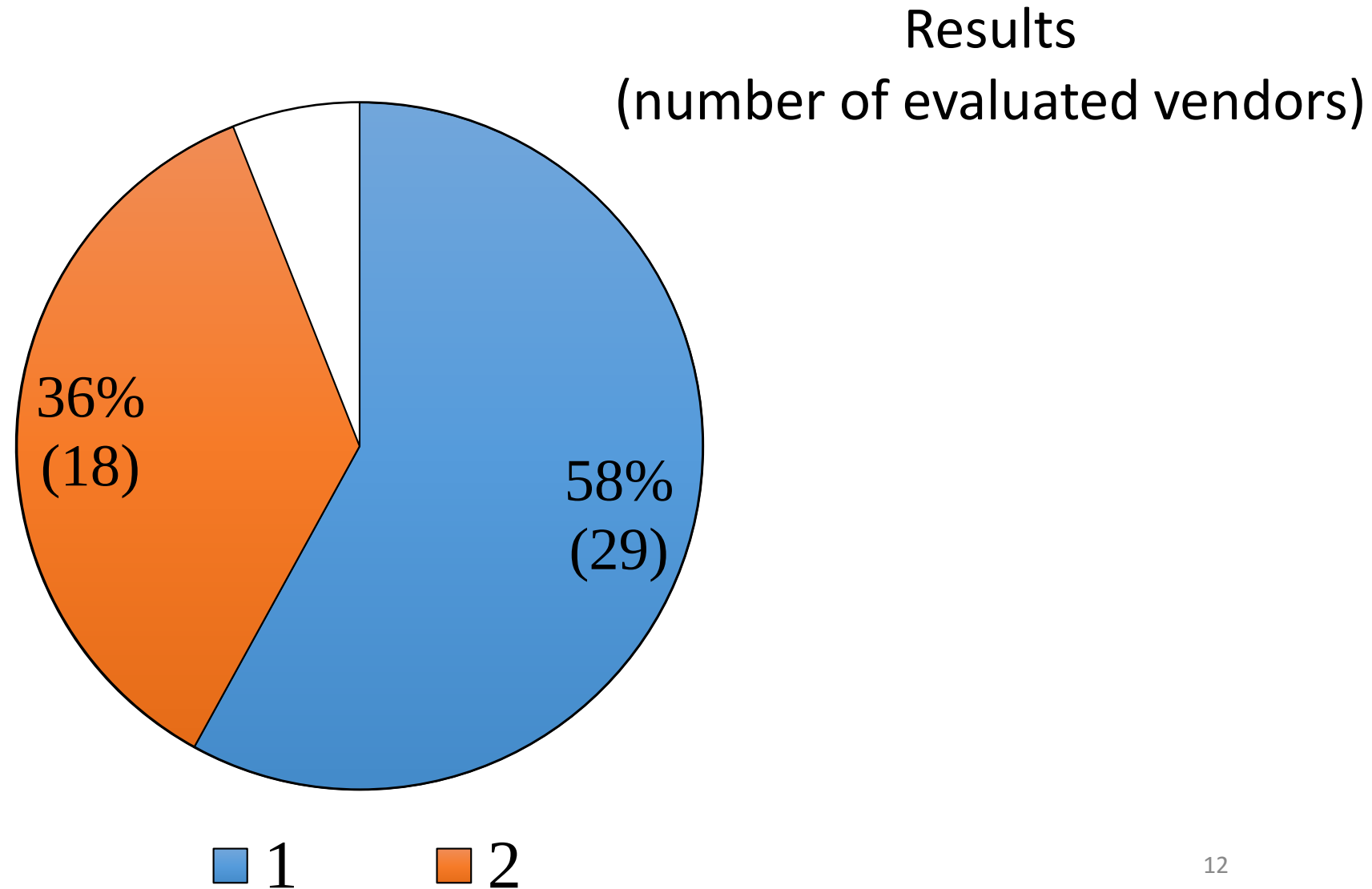
Portability in research literature

- 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?

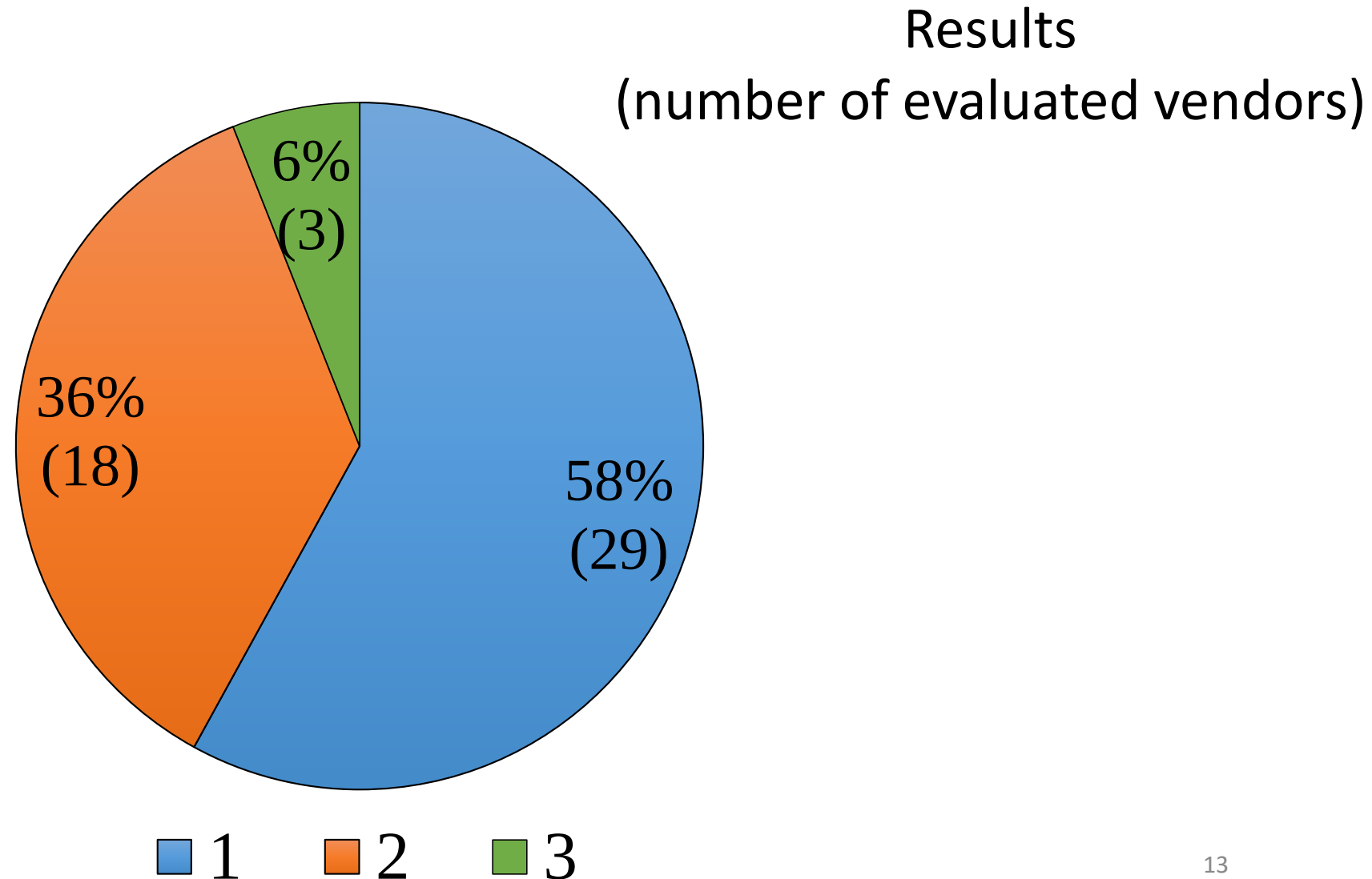
Portability in research literature



Portability in research literature

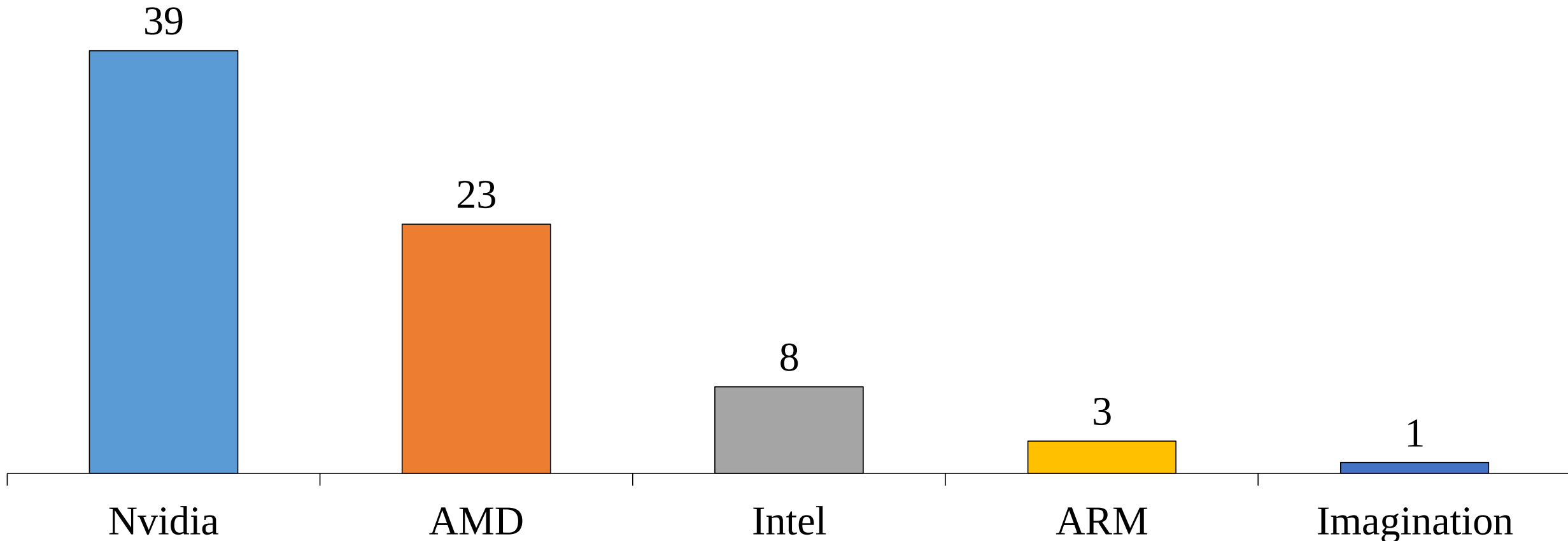


Portability in research literature



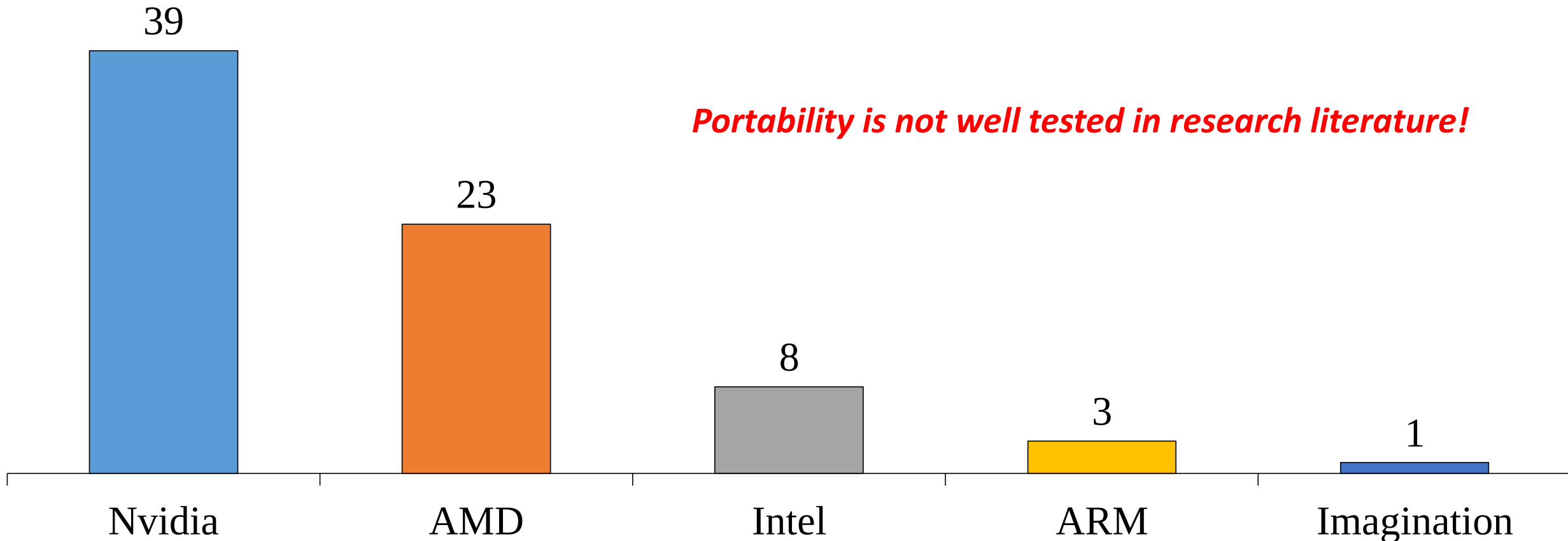
Portability in research literature

Results
(which vendor)



Portability in research literature

Results
(which vendor)



An experience report on OpenCL portability

- How well is portability evaluated?
- Our experience running applications on 8 GPUs spanning 4 vendors
- Recommendations going forward

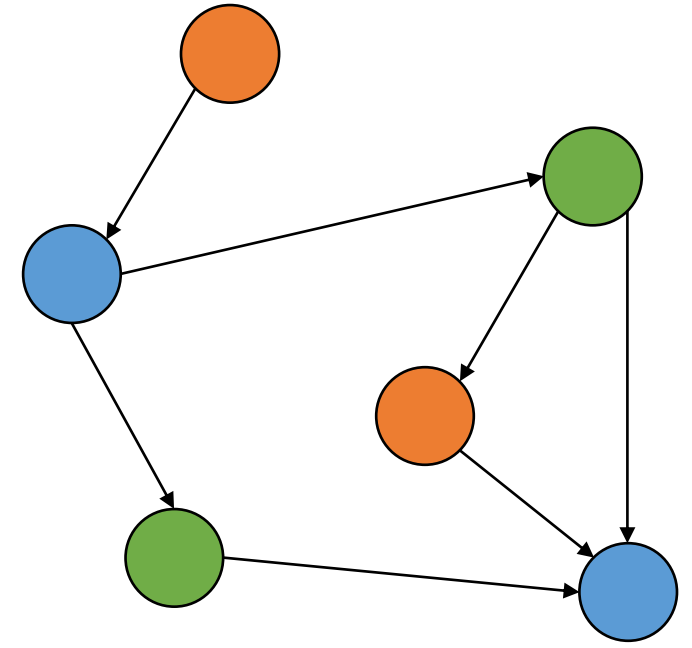
Applications

- Part of a larger study on GPU irregular parallelism

Applications

Pannotia

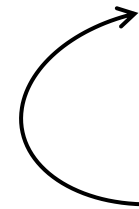
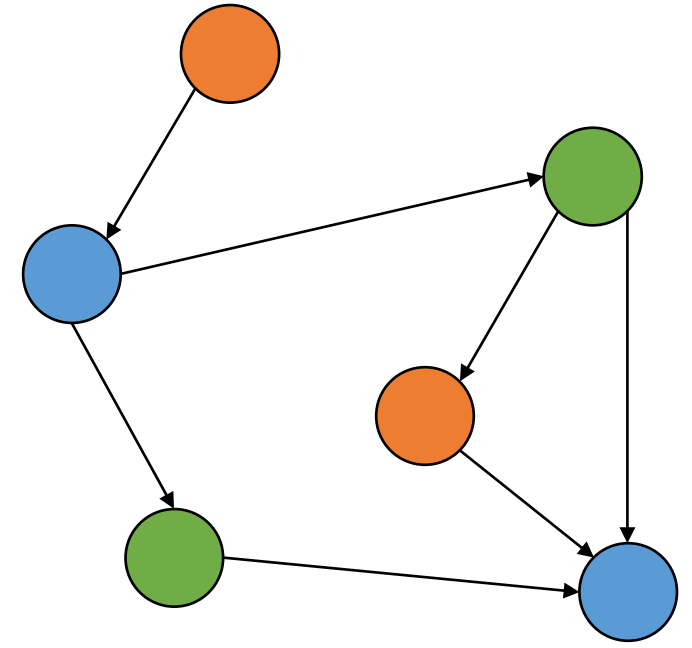
- Target AMD Radeon HD 7000
- Written in OpenCL 1.x
- 4 graph algorithms applications
- Our aim: run these benchmarks on OpenCL platorms from several vendors



Applications

Pannotia

- Target AMD Radeon HD 7000
- Written in OpenCL 1.x
- 4 graph algorithms applications
- Our aim: run these benchmarks on OpenCL platorms from several vendors



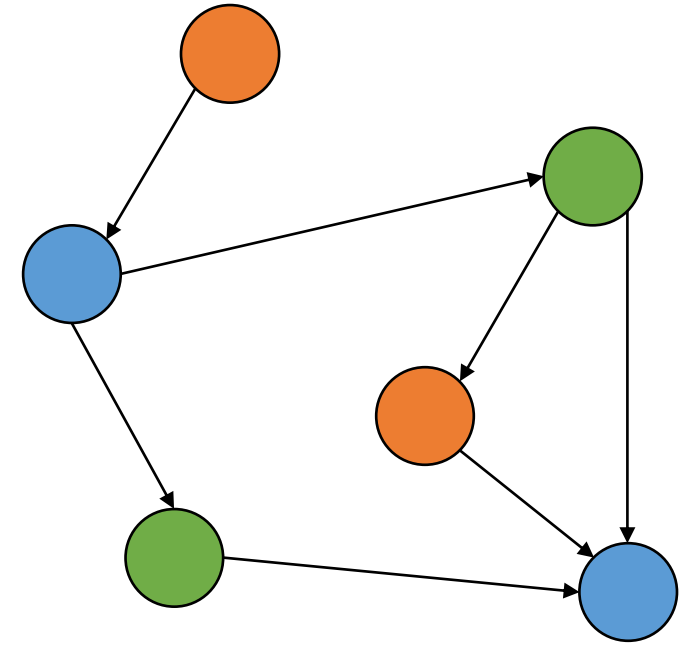
```
GPU_linear_algebra_routine1;  
GPU_linear_algebra_routine2;  
GPU_linear_algebra_routine3;
```

Loop until a fixed point is reached.

Applications

LonestarGPU

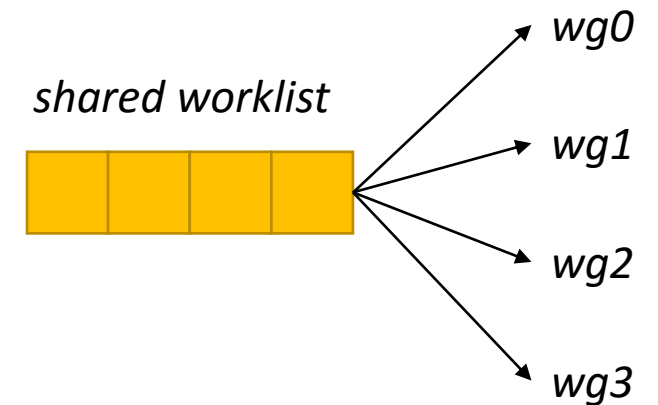
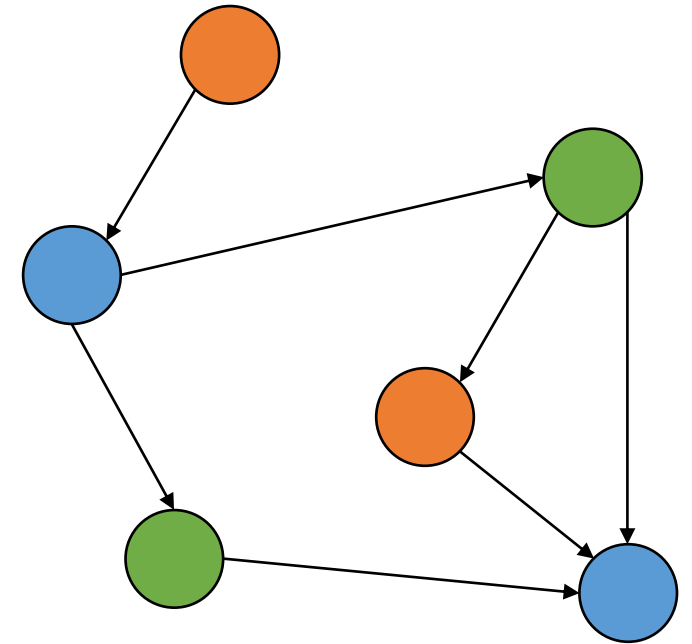
- Target Nvidia Kepler and Fermi
- Written in CUDA
- 4 graph algorithms applications
- Our aim: port these benchmarks to OpenCL to run across a range of platforms



Applications

LonestarGPU

- Target Nvidia Kepler and Fermi
- Written in CUDA
- 4 graph algorithms applications
- Our aim: port these benchmarks to OpenCL to run across a range of platforms



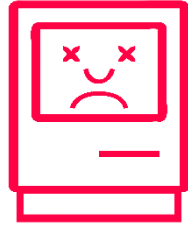
GPUs

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

Portability Issues

12 issues encountered, grouped into categories

- 3 Framework bugs



- 6 Specification limitations



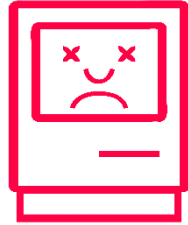
- 3 Programming bugs



Portability Issues

12 issues encountered, grouped into categories

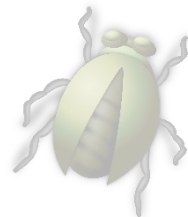
- 3 Framework bugs



- 6 Specification limitations



- 3 Programming bugs



Framework bugs

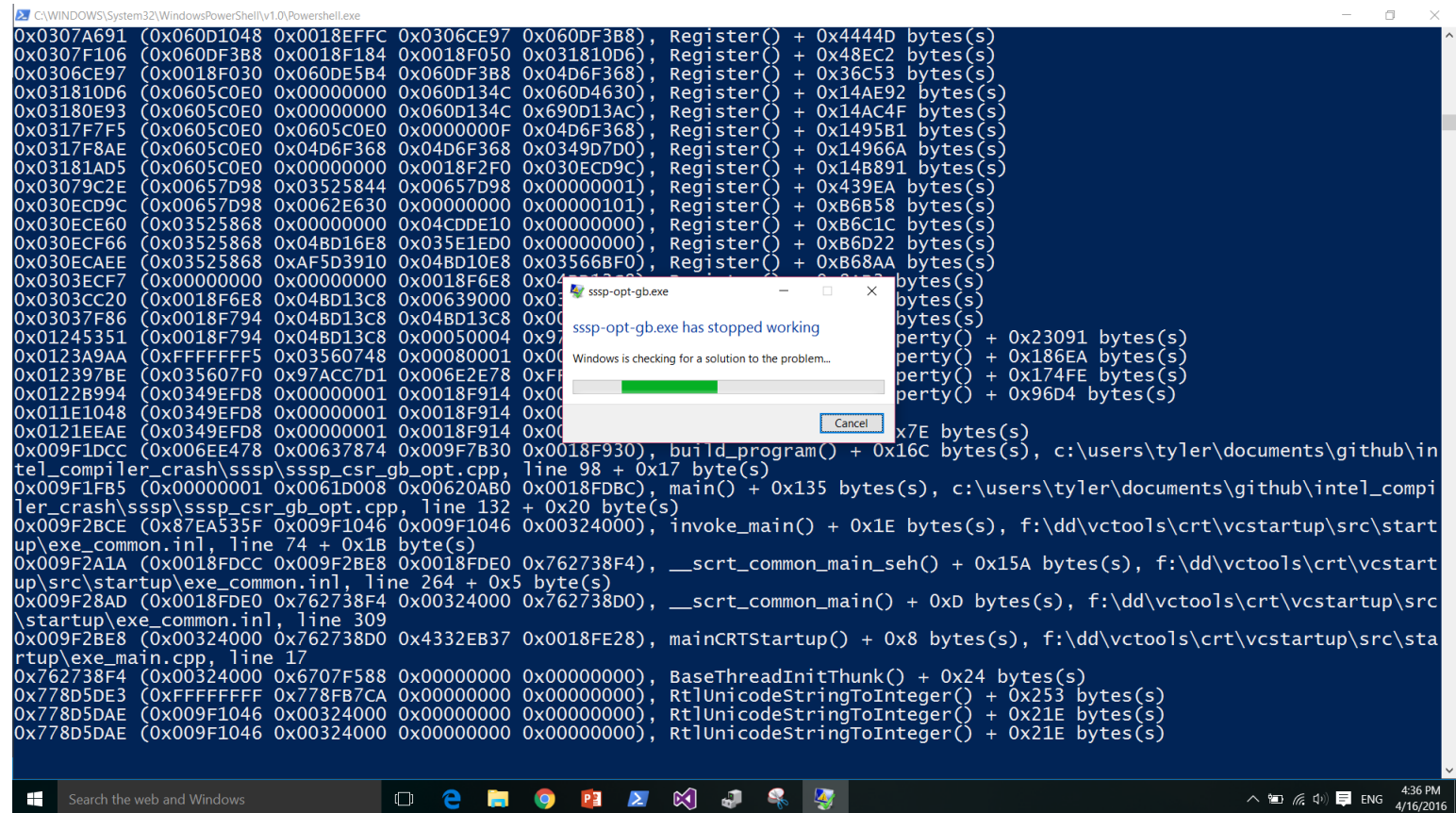
#1 Compiler crash

Platforms: Intel

Framework bugs

#1 Compiler crash

Platforms: Intel



The screenshot shows a Windows PowerShell window with a dark blue background. The title bar reads "C:\WINDOWS\System32\WindowsPowerShell\v1.0\Powershell.exe". The window contains a large amount of text, which appears to be a stack trace or a log of a program's execution. The text is white and includes various hexadecimal addresses and function names, such as "Register()", "build_program()", "mainCRTStartup()", and "BaseThreadInitThunk()". A Windows error dialog box is overlaid on the PowerShell window. The dialog box has a title bar that says "sssp-opt-gb.exe" and a message that says "sssp-opt-gb.exe has stopped working". Below the message, it says "Windows is checking for a solution to the problem...". There is a progress bar and a "Cancel" button at the bottom of the dialog box. The taskbar at the bottom of the screen shows the Windows logo, a search bar, and several application icons. The system tray in the bottom right corner shows the time as 4:36 PM and the date as 4/16/2016.

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

This looping idiom used in kernel code

Platforms: Nvidia and AMD

*Does not terminate on
Nvidia and AMD platforms!!*

```
while(true) {  
    more_work = false;  
  
    .. // Do computation,  
    .. // if more work, set more_work  
  
    if (!more_work)  
        break;  
}
```

Framework bugs

#2 Non-terminating loops

This looping idiom used in kernel code

Platforms: Nvidia and AMD

Change while loop to for loop

End value of `i` is consistent across platforms

```
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;  
}
```

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

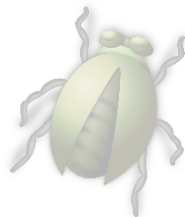
- 3 Framework bugs



- 6 Specification limitations



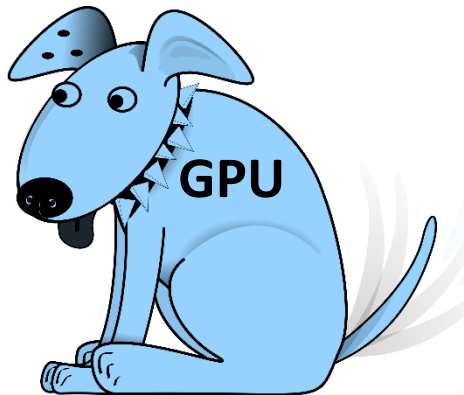
- 3 Programming bugs



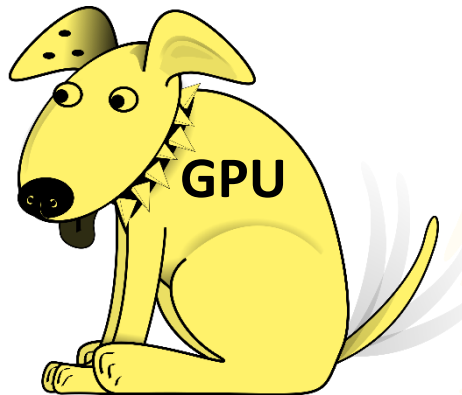
Specification limitations

#1 GPU watchdogs

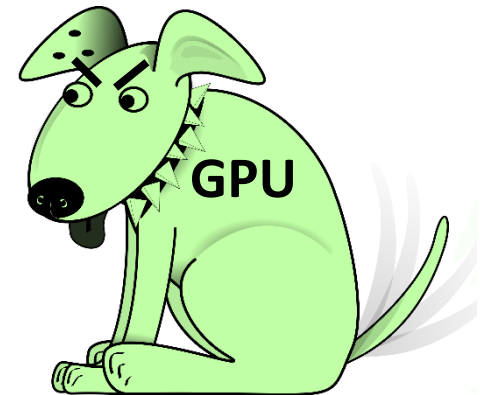
Platforms and operating systems handle watchdogs differently.



Windows



Linux (Ubuntu)



Chrome OS

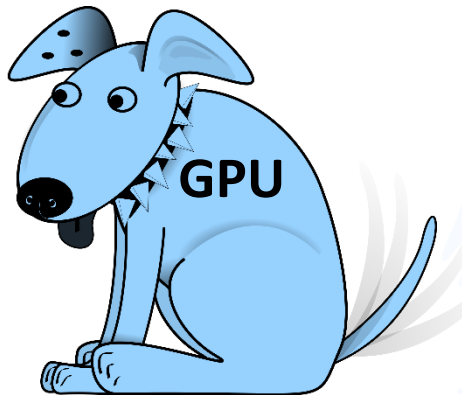
Specification limitations

#1 GPU watchdogs

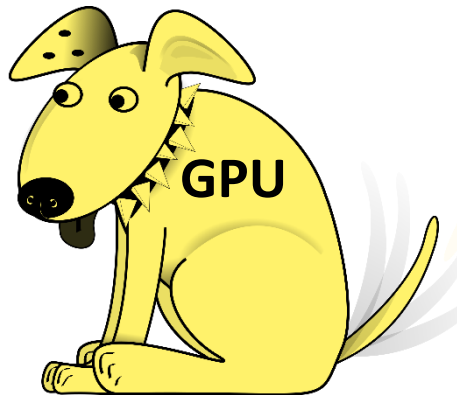
Platforms and operating systems handle watchdogs differently.

Controlled with registry

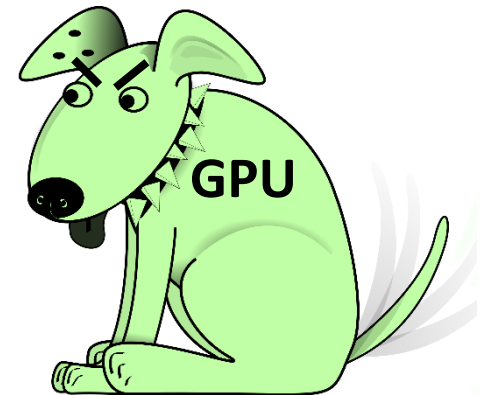
Watchdog kills entire
OpenCL process



Windows



Linux (Ubuntu)



Chrome OS

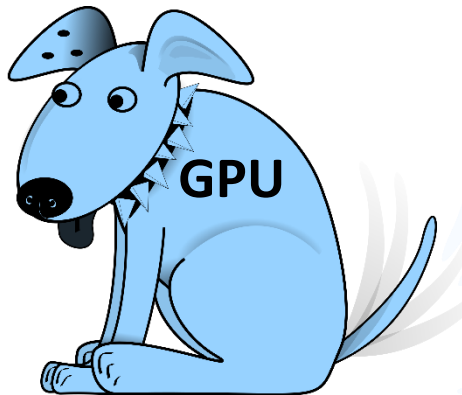
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

Controlled with registry

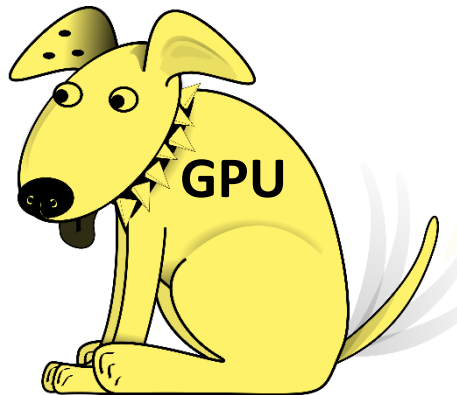
Watchdog kills entire
OpenCL process



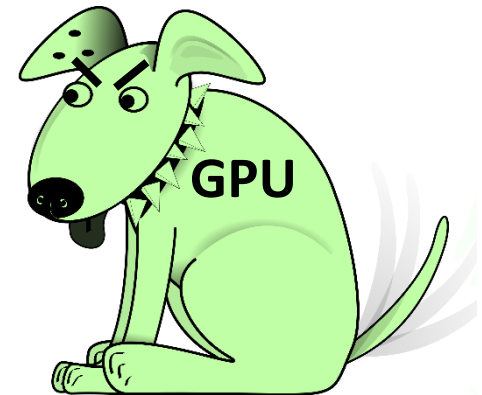
Windows

Controlled in X server settings

Watchdog only kills kernel



Linux (Ubuntu)



Chrome OS

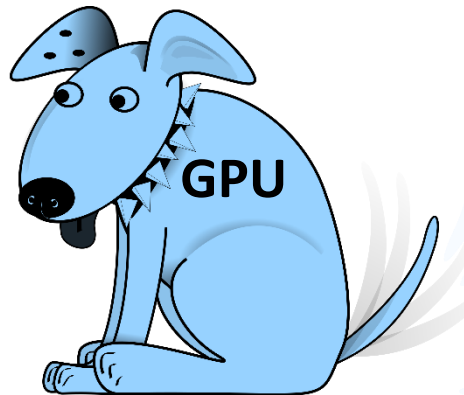
Specification limitations

#1 GPU watchdogs

Platforms and operating systems handle watchdogs differently.

Controlled with registry

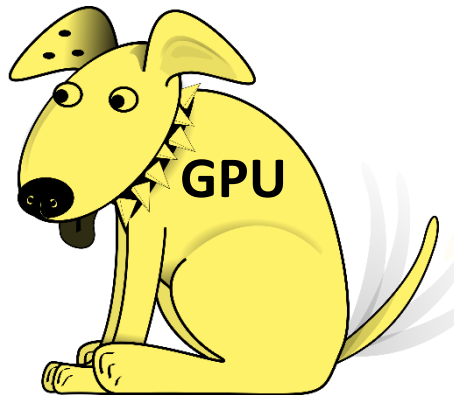
Watchdog kills entire
OpenCL process



Windows

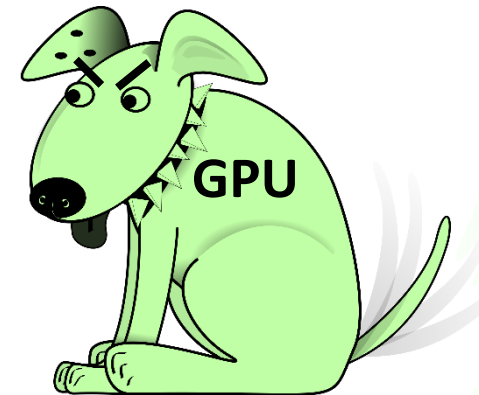
Controlled in X server settings

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): *once scheduled, a workgroup is guaranteed to make progress*

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): *once scheduled, a workgroup is guaranteed to make progress*

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	

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

Compute units are safe
and optimal

Compute units are safe but
not optimal

Specification limitations

#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

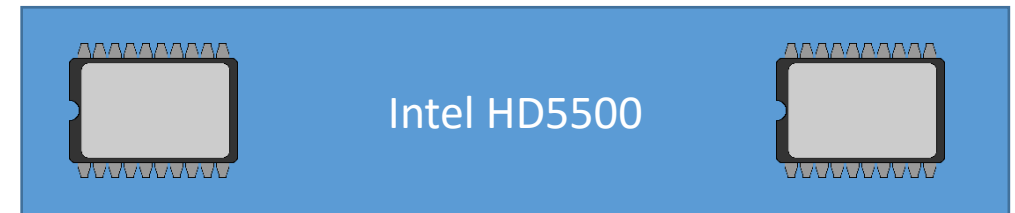
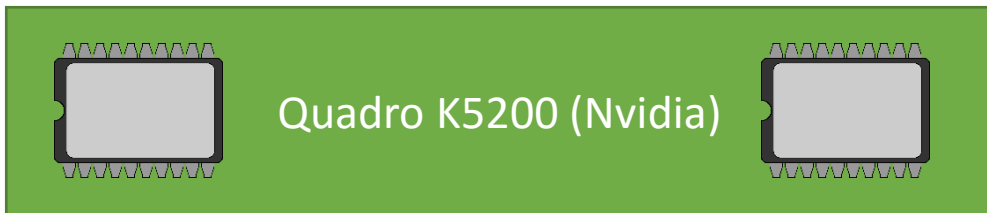
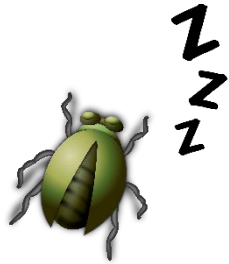


Programming bugs

#1 Data-races

Application: LonestarGPU bfs and sssp

Fix: Add additional synchronisation barriers



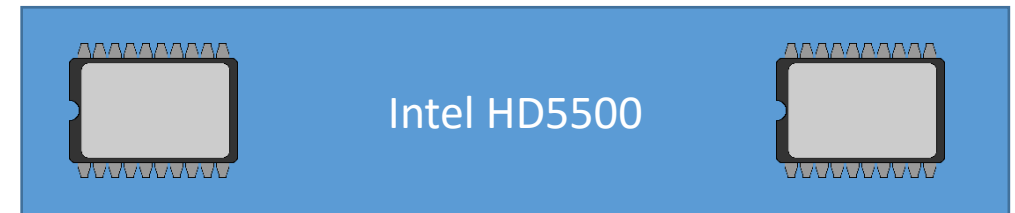
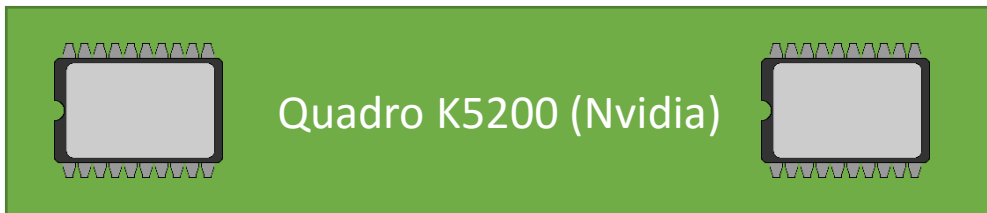
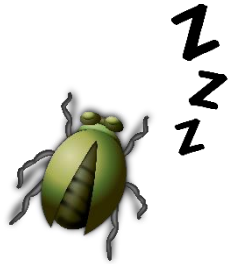
Programming bugs

#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.

Some struct members are global memory pointers

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”, IWOCCL’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!

We are hiring

- Postdoctoral researcher in *Reliable Many-Core Programming*
- Two fully-funded UK/EU PhD studentships on reliability and efficiency of concurrent and parallel software
- Talk to me, or email (afd@imperial.ac.uk) if you are interested
- About our group: <http://multicore.doc.ic.ac.uk>

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

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

Defunct process bug

Specification limitations

#5 OS portability

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

Defunct process bug



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



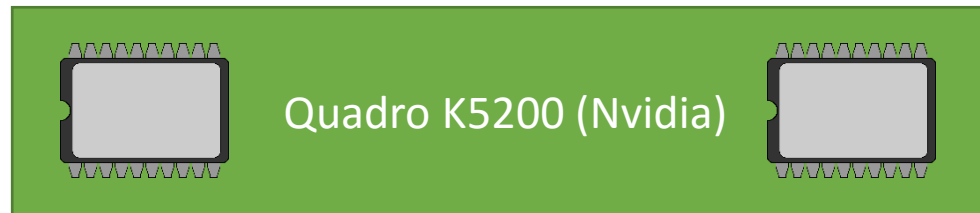
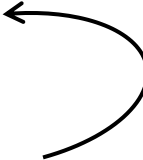
Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

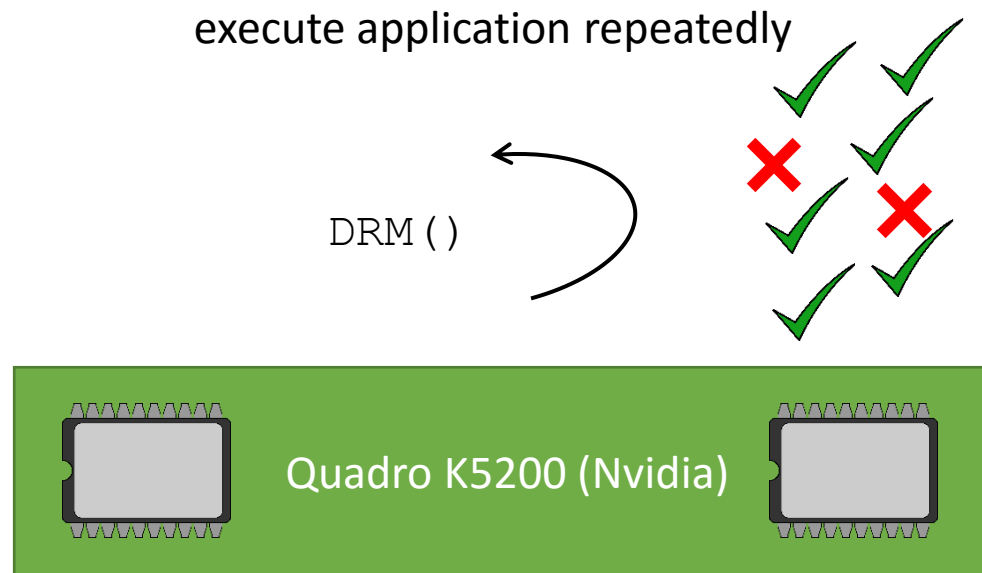
DRM ()



Programming bugs

#2 Stability

Application: LonestarGPU DMR



occasional failures
(known by developer
and deemed acceptable)

Due to floating point precision

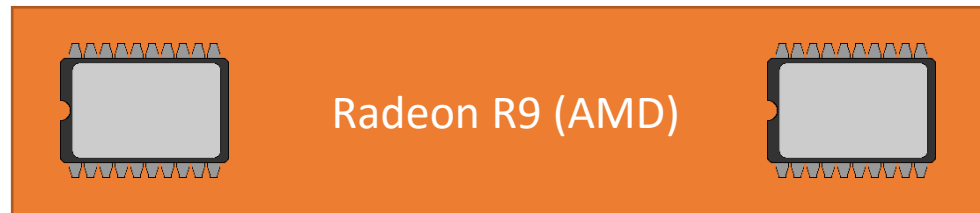
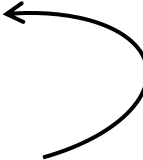
Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

DRM ()



Programming bugs

#2 Stability

Application: LonestarGPU DMR

execute application repeatedly

DRM ()



Fails nearly every iteration
on AMD chips

