

Making code run fast On all the things (with OpenCL)

Beau Johnston

Not last years talk!

- Thanks for having me back, and for being here!
- Last year was all about the kernel (also prototyping parallelism with OpenMP), now we target the host!
- What we are really focusing on today is:

OpenCL, saving parallel programmers ~~pain,~~ today! *Time*

- Ok there will be some overlap (but who was at my last talk)?
- This talk is all about time...
- “*Yesterday is a canceled check; tomorrow is a promissory note; today is the only cash you have - so spend it wisely*” ~Kay Lyons
- How much time must we as developers have to spend to appease our clients?

This morning we'll cover:

- No fence sitting! I'm firmly pro OpenCL
- A quick recap on OpenCL (including its caveats)
- The Host is boring... why is it so?
- The Host is boring... fine I'll do the boring bits for you!
- Has it been a year already and just what's happened to OpenCL since?

- Who has slow code?
- Who has time for slow code?
- Anyone heard of Accelerator devices?

All the Accelerator Devices!

Namely:

- CPUs ~ really fast, few cores, good at branching... expensive
- GPUs ~ not so fast, many cores, bad branching! Cheap?
- DSPs ~ Cheap!
- FPGAs ~ Fast IO's -> great for streaming, pricey initial cost, needs configuring.

We are interested use them for non-standard purposes (to accelerate computation).

Imagine a world...

Where each device has it's own language, APIs, IDE & ecosystem.

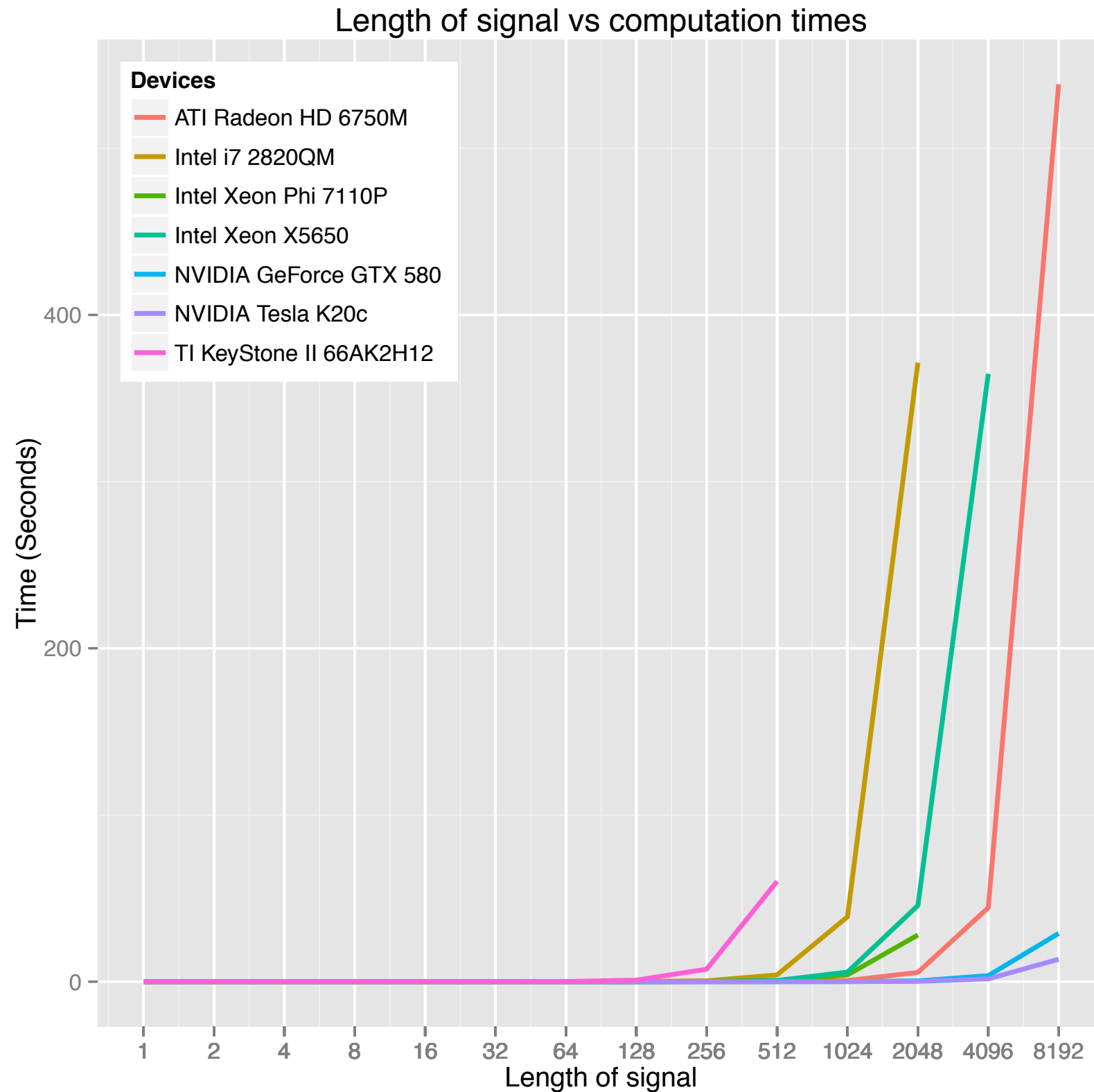
Doesn't it sounds time consuming?

and we're going back there, unless...

Introducing OpenCL

- Develop for a large range of devices in one shot.
- Code reuse (and reinventing the wheel stinks).
- ...And Open (...well mostly)!

OpenCL on all the things!



OpenCL through the ages...

- It's Mature (~5 years old) and evolving:
 - v1.0 in 2008
 - v1.1 in 2010
 - v1.2 2011
 - v2.0 2013.
- Revisions are made to keep the standard current with modern features.

(It's not all)

Fun & Games

OpenCL version	Intel	NVIDIA	AMD	TI	ARM	Apple
1.0	✓	✓	✓	✓	✓	✓
1.1	✓	✓	✓	✓	✓	✓
1.2	✓		✓		✓	✓
2.0	✓		✓			

More bad news...

Vendor specific device extensions reduce portability.

- atomics
- 3d_image
- Vendor specific sharing (OpenGL, Direct3D)
- Odd precision (half & double) primitives
- SPIR (we'll get back to that)
- etc...

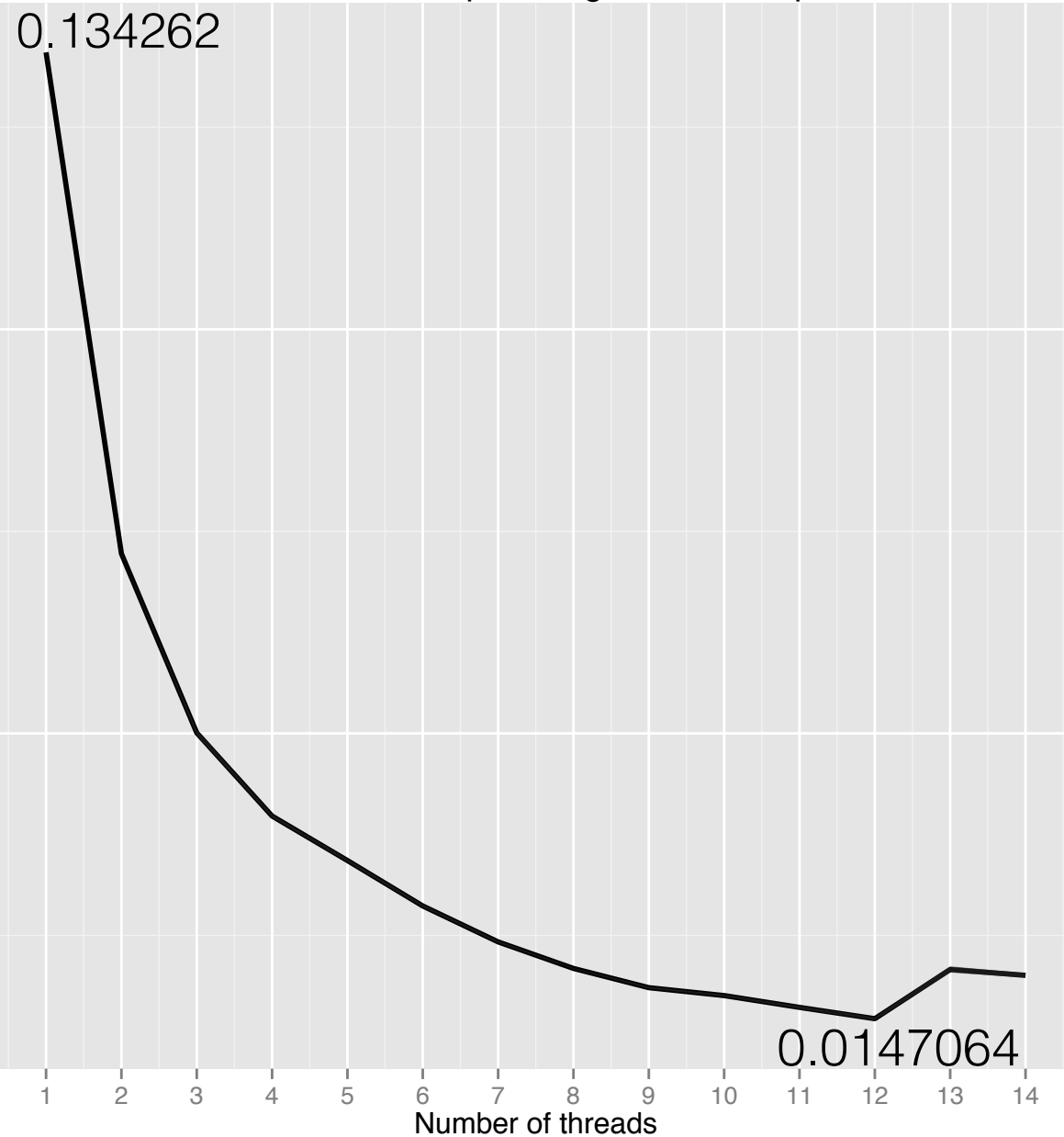
But thankfully...

Well written OpenCL can be:

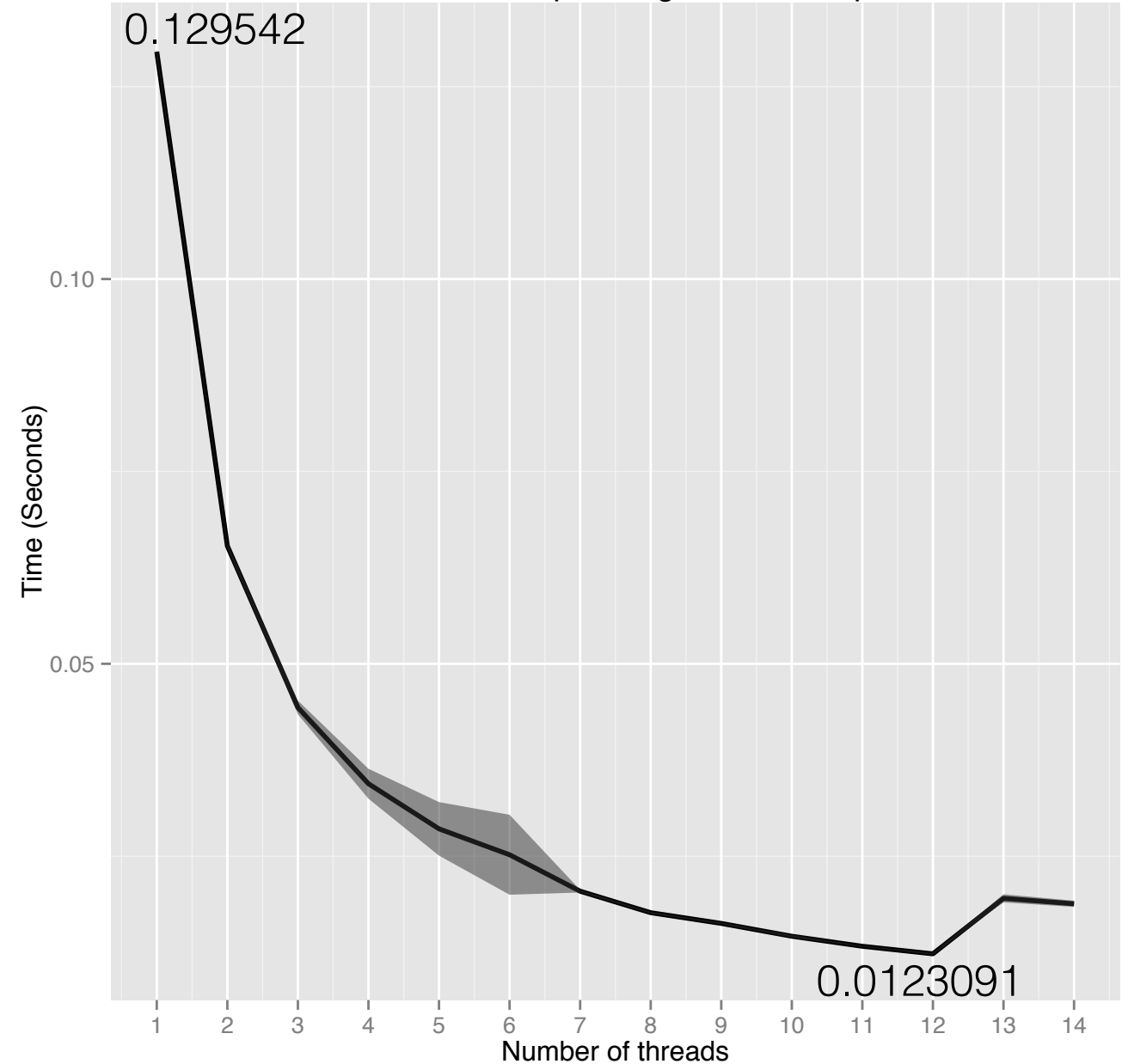
- OpenCL is faster than OpenMP on CPUs

OpenMP vs OpenCL (wall-clock times)

Xeon X5650 @ 2.67GHz OpenMP gcc-O3 Computation Times

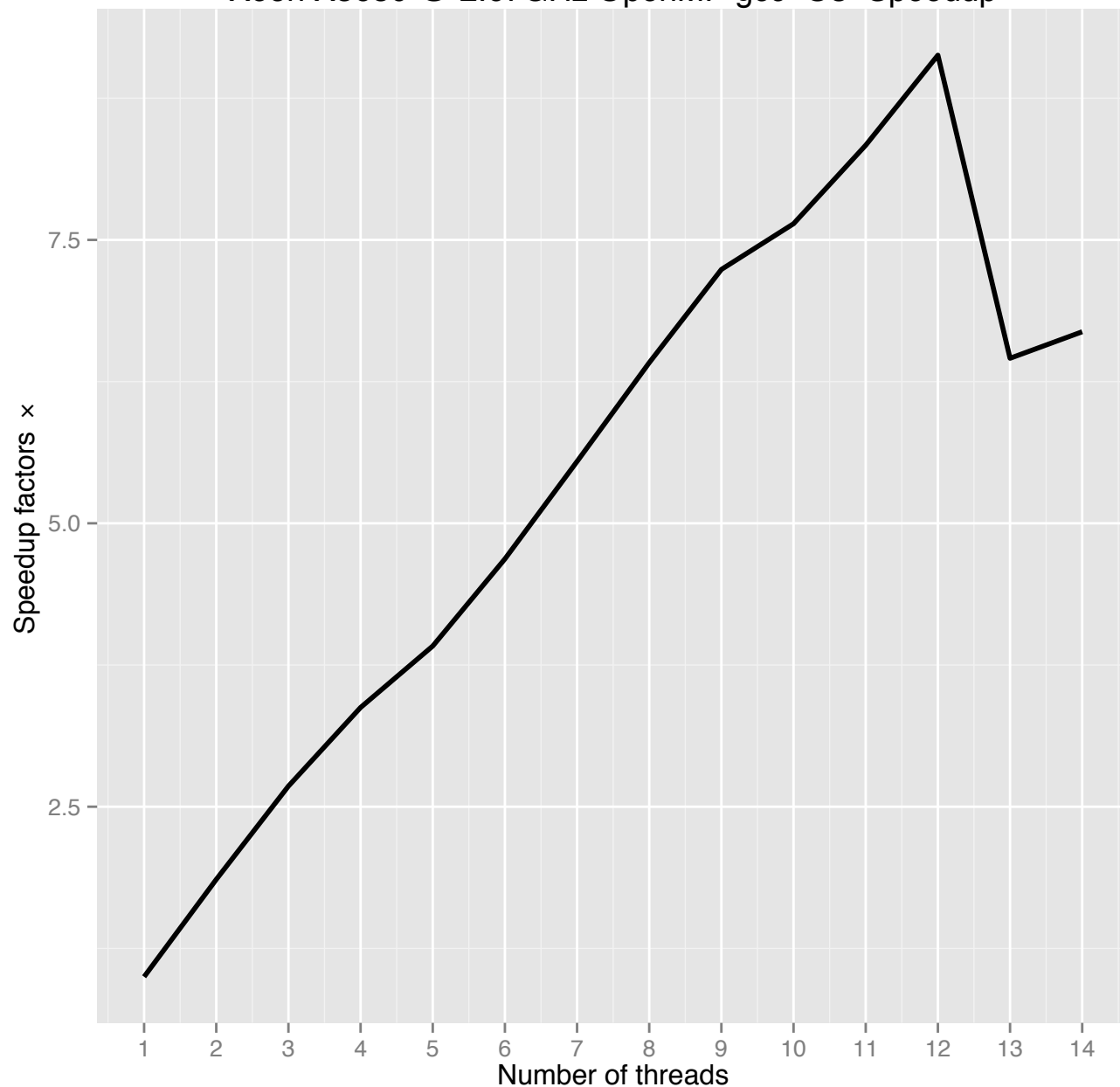


Xeon X5650 @ 2.67GHz OpenCL gcc-O3 Computation Times



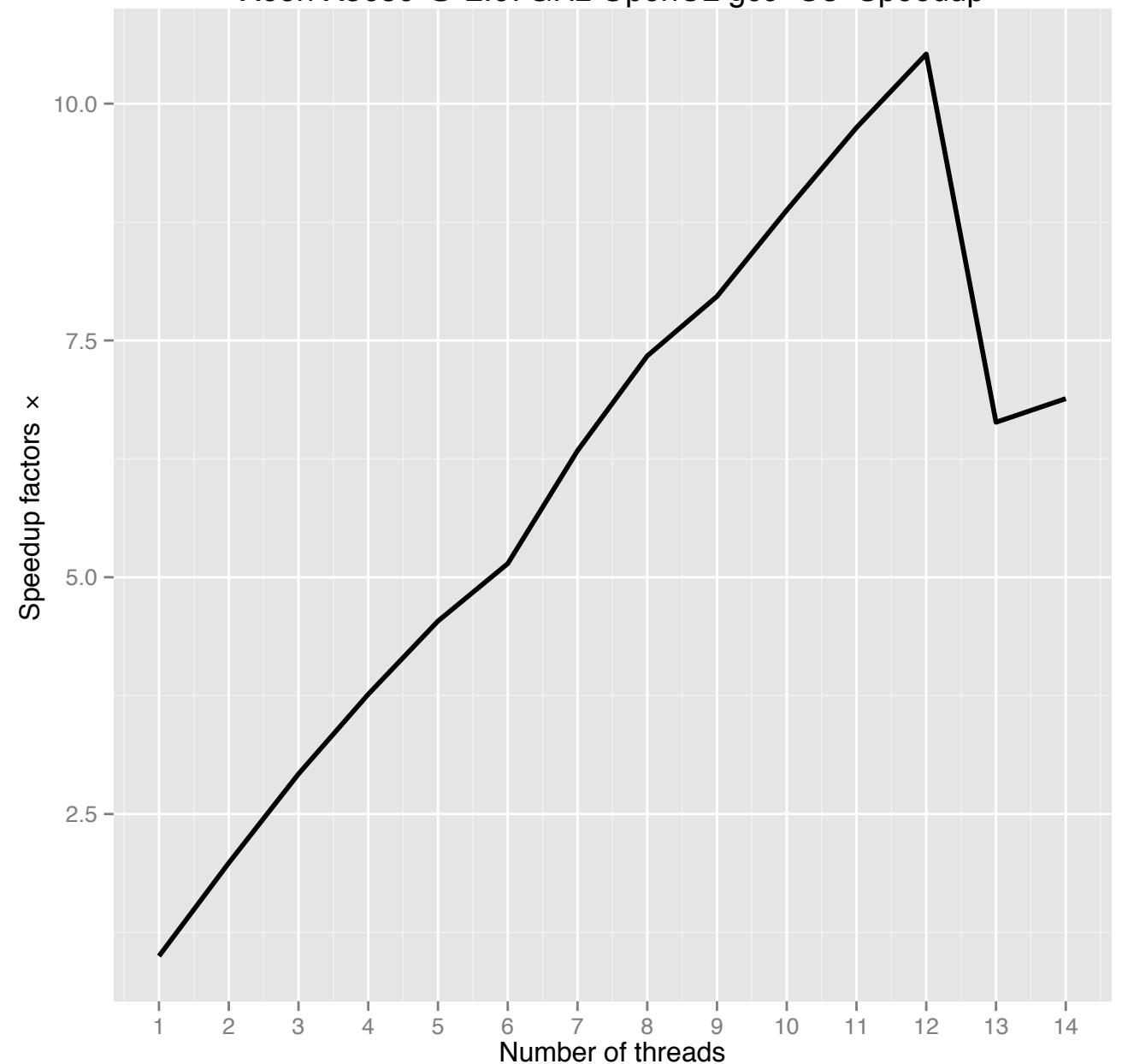
OpenMP vs OpenCL (speedup factor)

Xeon X5650 @ 2.67GHz OpenMP gcc-O3 Speedup



9.1x

Xeon X5650 @ 2.67GHz OpenCL gcc-O3 Speedup

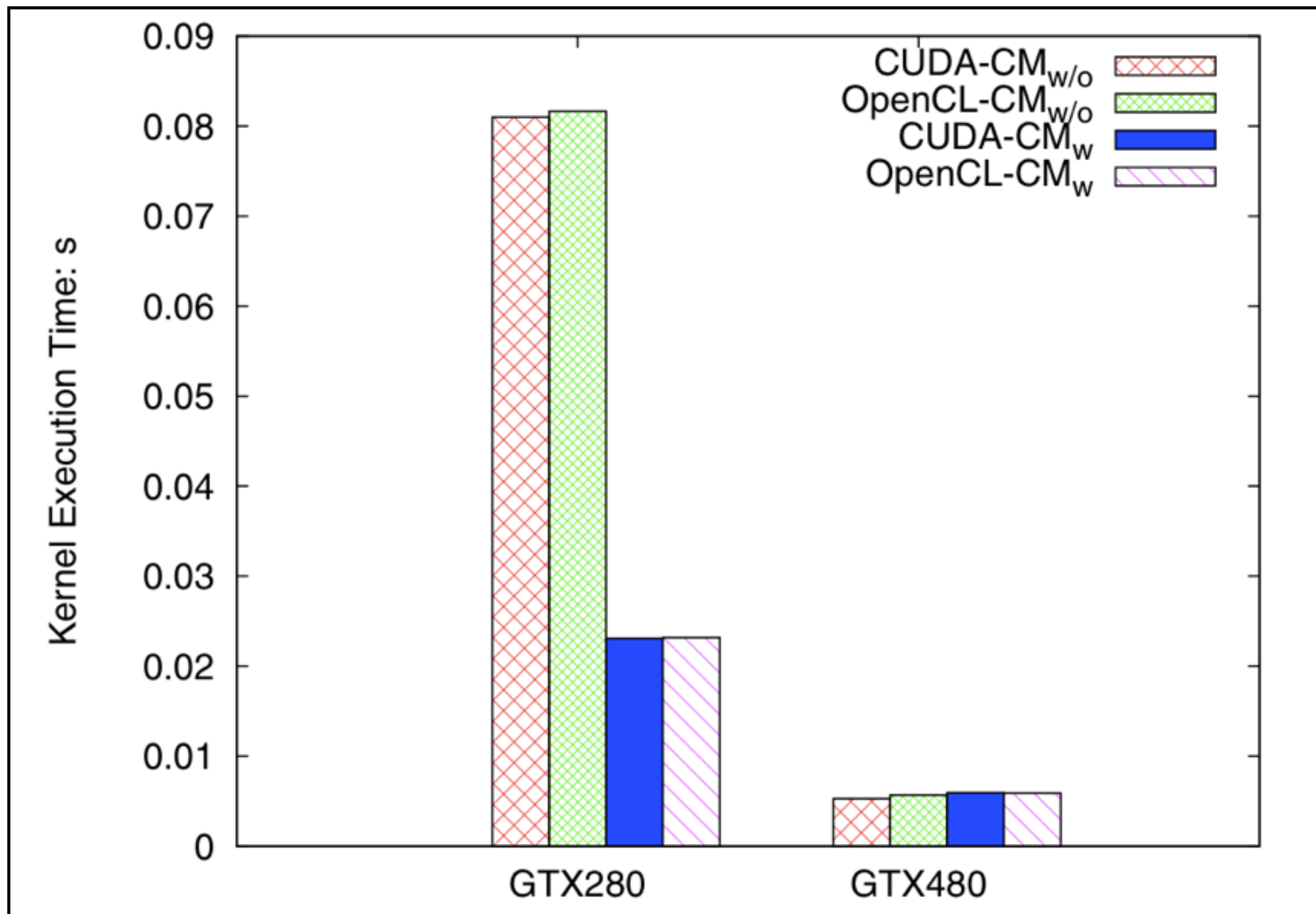


10.5x

But thankfully...

Well written OpenCL can be:

- OpenCL is faster than OpenMP on CPUs
- Comparable to CUDA on GPUs



A performance comparison of a Sobel benchmark with and without constant memory.^[1]

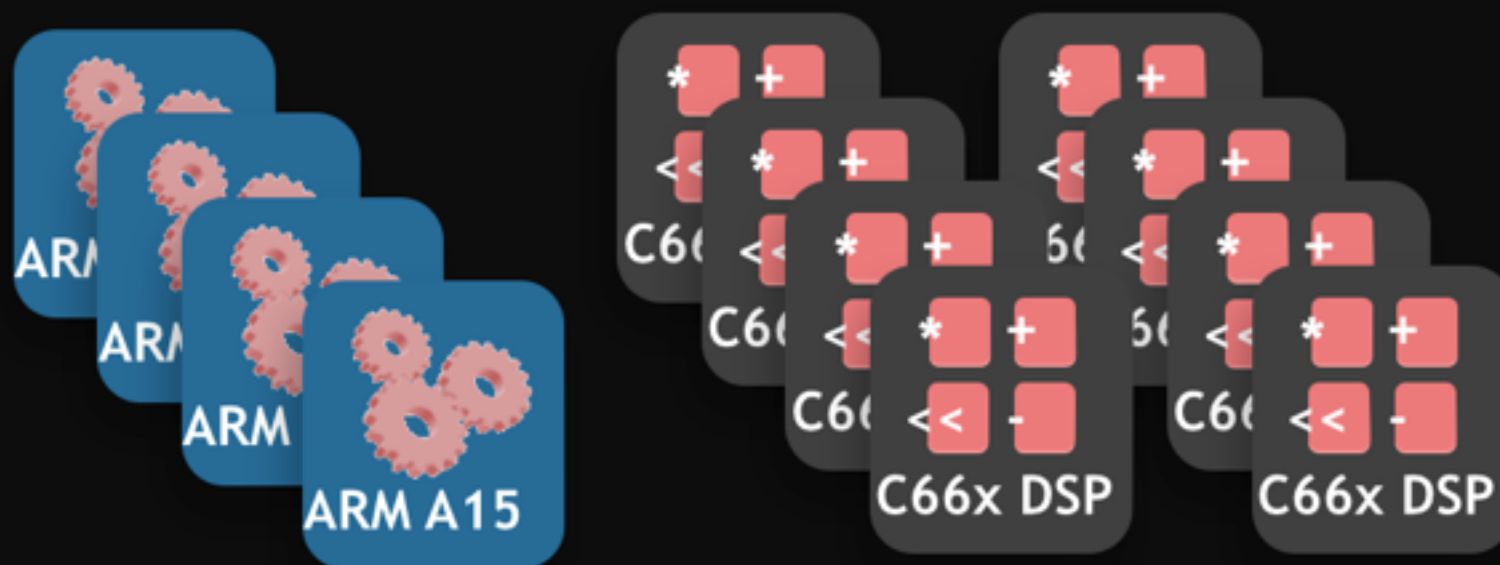
^[1] Karimi, K., Dickson, N. G., & Hamze, F. (2010). A performance comparison of CUDA and OpenCL. *arXiv preprint arXiv:1005.2581*.

But thankfully...

Well written OpenCL can be:

- OpenCL is faster than OpenMP on CPUs
- Comparable to CUDA on GPUs
- The way forward for DSPs

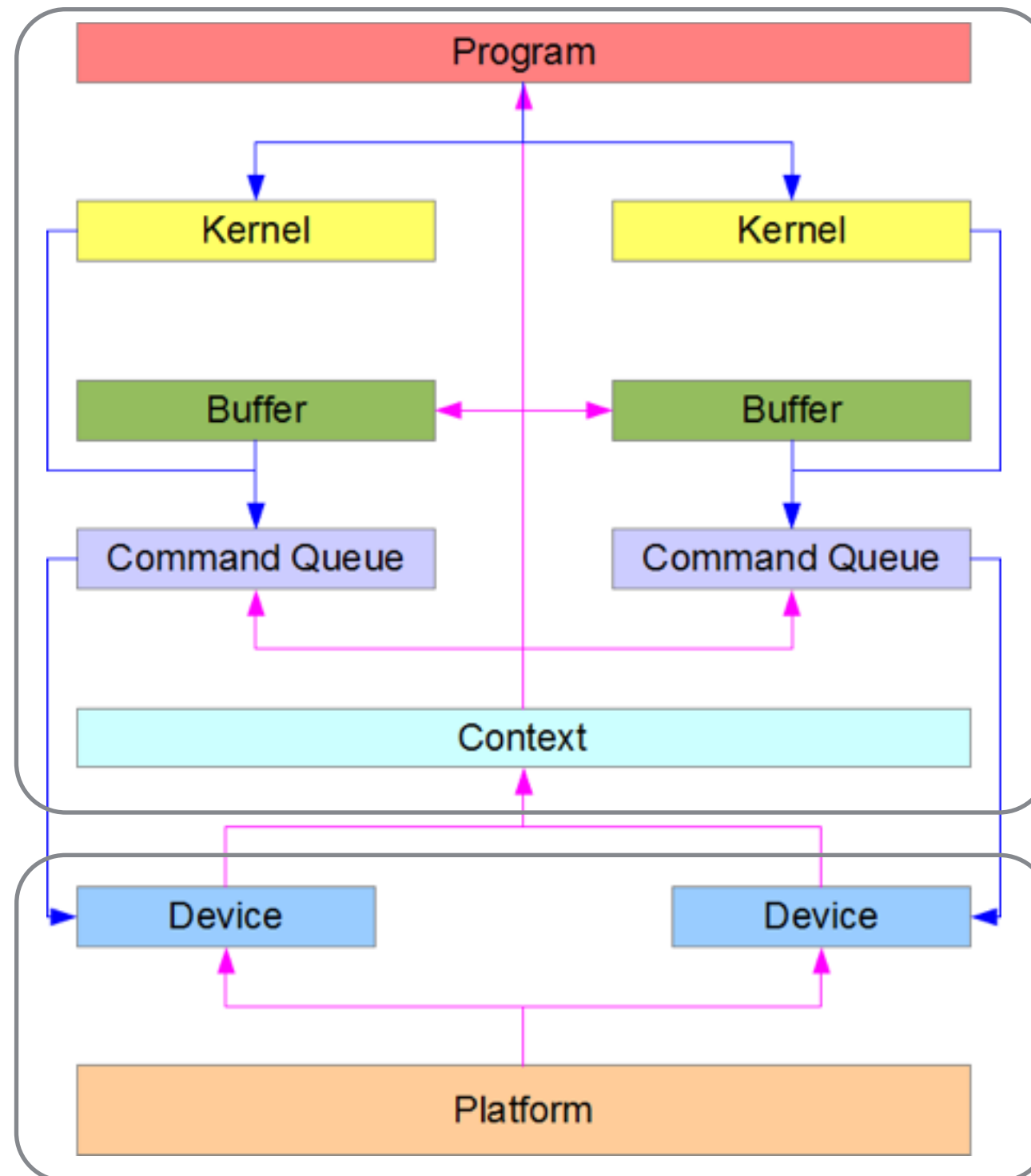
66AK2H12 KeyStone II Multicore DSP + ARM



Texas Instruments KeyStone II is an example of current DSP hardware using OpenCL.

- ARM quad-core CPU
- 8 DSPs
- Shared memory

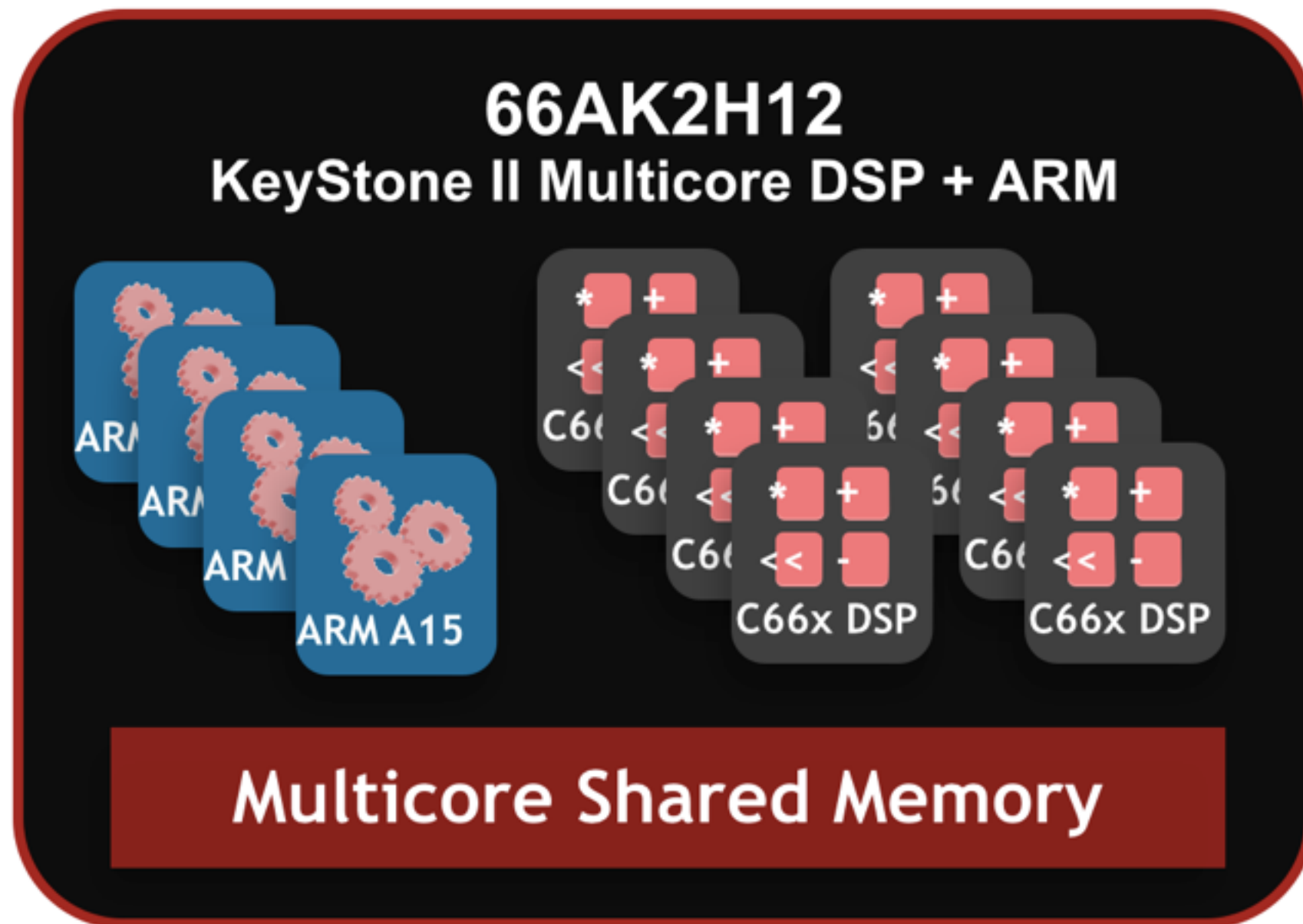
The host is boring... why is it so?



Runtime Layer

Platform Layer

An example of OpenCL in action!





How does the Keystone II use OpenCL?

- ARM quad-core CPU (running host & potentially 3 other compute units)
- 8 DSPs (compute units) cores for kernel computation
- Shared memory (so the host can keep on feeding those circular buffers)

The host is boring...
fine I'll do the boring bits for you!

- I've taken myself off the market! Would you employ a programmer that works this slowly?
- Remember how we're trying to save on developer time?
- I've timed how long it takes me to write the host code, from scratch, in C, C++ and Python (pyopencl module).
- Do high level languages sacrifice execution speed? Is it significant (& does it matter on the host?)
- I'll also look at lines of code.

Time out!

	C		C++		Python	
	Dev Time	Search Time	Dev Time	Search Time	Dev Time	Search Time
Command line parsing	 Platform layer included					
Generate Signals						
Create Context	 Runtime layer					
Create Command Queue						
Load & Compile Kernel						
Generate & Fill Buffer						
Set Arguments						
Execute Kernel						
Wait for Execution						
Get Result and Write to File						
Clean up						
Environment Setup						
Total time spent (mins)						

A walk through the code
(I hope this works!)

Development time

	C		C++		Python	
	Dev Time	Search Time	Dev Time	Search Time	Dev Time	Search Time
Command line parsing	123	4	76	2	33	2
Generate Signals	66	0	27	2	6	1
Create Context	10	1	30	1	0.5	0
Create Command Queue	4	1	1	6	0.5	0
Load & Compile Kernel	41	2	30	4	5	2
Generate & Fill Buffer	25	6	11	6	5	0.5
Set Arguments	12	1	16	2	3	20
Execute Kernel	8	1	5	2	0.5	0
Wait for Execution	0.5	1	1	1	0.5	0
Get Result and Write to File	8	1	10	1	6	4
Clean up	14	1	2	1	-	-
Environment Setup	9	3	8	18	20	25
Total time spent (mins)	320.5	22	217	46	80	54

Lines of Code

	C	C++	Python
Command line parsing	222	113	63
Generate Signals	46	45	5
Create Context	10	3	1
Create Command Queue	7	1	1
Load & Compile Kernel	38	26	8
Generate & Fill Buffer	41	17	12
Set Arguments	70	10	6
Execute Kernel	13	4	5
Wait for Execution	1	1	1
Get Result and Write to File	22	15	6
Clean up	14	4	0
Environment Setup	17	16	10
Total lines of code	501	255	118

Running Times

On an i5-4250U CPU @ 1.30GHz

	C	C++	Python
Wall clock time	1.30	1.34	1.57
(only) kernel execution	1.17	1.13	1.36

Python seems slower (big shock right?) but that's mostly the host code. I've checked with different kernel work loads and that's a one shot, fixed cost.

Concluding thoughts...

I should have learnt python sooner!

You'll never have to start from scratch

- I've also used the CWT example as boiler plate code for a new application (and kernel).
- This should gauge the amount of work required to rapidly develop OpenCL code in your language of choice (provided that's C, C++ or python).
- But due to poor lifestyle choices, it isn't in slide form yet! (I'll push it up on github). The python version only required 8 minutes of work to have a new application up and running!

Fin!

Now the boring part has taken care of
what's your excuse?

Has it been a year already and just what's happened to OpenCL since?

- OpenMP 4.0 & OpenACC accelerator directives still seem the way forward to save us from writing boring hosts.
- Commercial compilers (PGI, CRAY and CAPS) and Open Source (OpenUH, OpenARC, accULL) exist for OpenACC but OpenMP 4.0 is not there yet.
- GNU GCC is also working on adding OpenACC support.
- Their runtimes will probably leverage OpenCL to achieve this.
- So we'll have to keep writing host code until it becomes tenable. This should encourage better involvement with the standard.

What else has happened?

- We're finally getting FFT and BLAS support!
- This was needed as researchers and industry professionals often use proprietary equivalents of OpenCL such as CUDA due to their mature BLAS and FFT libraries.
- clMath is the open-source project for OpenCL based BLAS and FFT libraries:

<https://github.com/clMathLibraries>

- Alpha release of OpenCL BLAS:

<http://openclblas.sourceforge.net>

Also there's: SPIR

- Standard Portable Intermediate Representation
- Subset of LLVM
- Saves developers from shipping source kernel code.
- Saves time!
- But also is an optional extension :(Could pressure from the masses encourage vendors to adopt it?

Even more vendor support!

- Version 2.0 specification was released in November 2013, I thought we'd have to wait longer but AMD & Intel already support it.
- How do we encourage other vendors to keep up?
- I think the more people that use OpenCL the more pressure will be placed on vendors to continue to support it.
- That's why I need you!

With your support...

- We're seeing OpenCL being fragmented with optional extensions.
- If you're involved we may be able to get the important ones made mandatory.
- Without all vendors sharing equal weight (and supporting the same standard) what's the point of OpenCL?
- We all need to get behind OpenCL to avoid fragmenting the software development community further.

Get Involved!

(... also thanks)

For questions & collaborations please contact me at:

beau@inbeta.org

All OpenCL host code available here:

<https://github.com/BeauJoh/opencl demos>

Last years talk on the kernel side of OpenCL can be found here:

<http://mirror.linux.org.au>