Heterogeneous Machines Challenges for the Compiler Community

Norm Rubin
NVIDIA
Northeastern University

How are these three conferences different?

Differences in titles 2012,2013, bigger font means greater delta, Angle determined by the ratio

What is a heterogeneous machine? some history

- ***Moore's law is the observation that over the history of computing hardware the number of transistors on integrated circuits doubles approximately every two years - Gordon Moore, 1965 "Electronic News" article***
- **Ice-age to 2005 single core world**
  - **Use extra transistors for more performance**
  - **70, 80s one big feature at a time (floating point on die, oo exec, pipelines)**
  - **90-2000's multiple smaller features at time**
  - **Cores get more and more complex**
  - **Developers are happy – just recompile and everything goes faster**
  - **Complier writers are happy- understand the machine**

2005- multiple identical cores

- Multi-cores
  - All the cores are the same,
  - Performance through replication
  - One unified memory
  - Software responds through libraries and a few parallel applications (Parallel make)
  - For a small number of complex cores, there is a simple interconnect, and we can build 2 digits of cores
  - Questionable if this can scale – within a power budget

2009- heterogeneous cores

- Power becomes an issue
- Within desktop and mobile form factors, power and heat, and height are limited

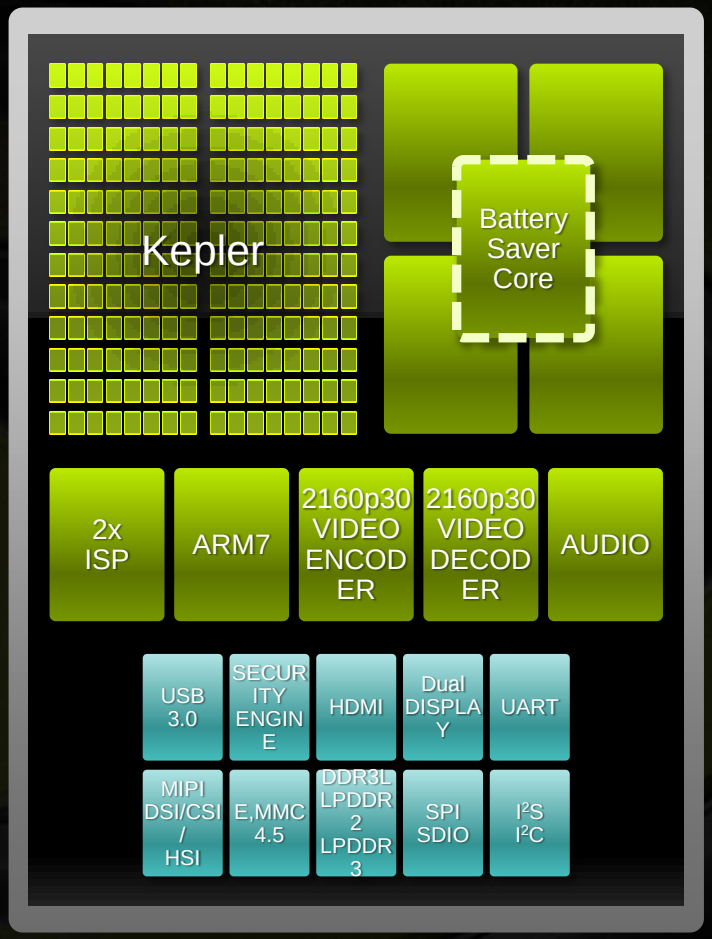
- Either big/little cores – same isa but still different,
- Or different designs - latency and throughput cores
- Specialized cores for specific tasks
- This is a permanent change –
  - Specialized cores are power efficient
- Software does not know how to split work
  - Up to the user, new languages DirectX, CUDA, OpenCL

Heterogeneous cores and memory

- Memory might be unified
  - APU/ TEGRA
- Or memory might be split into memory per core type
  - GPU discrete cards
- Software – let the user code all data motion, figure out what computations on what core

- As parallelism goes up, the memory interconnect gets more complex so layout matters, but it is a up to the developer

Tegra K1

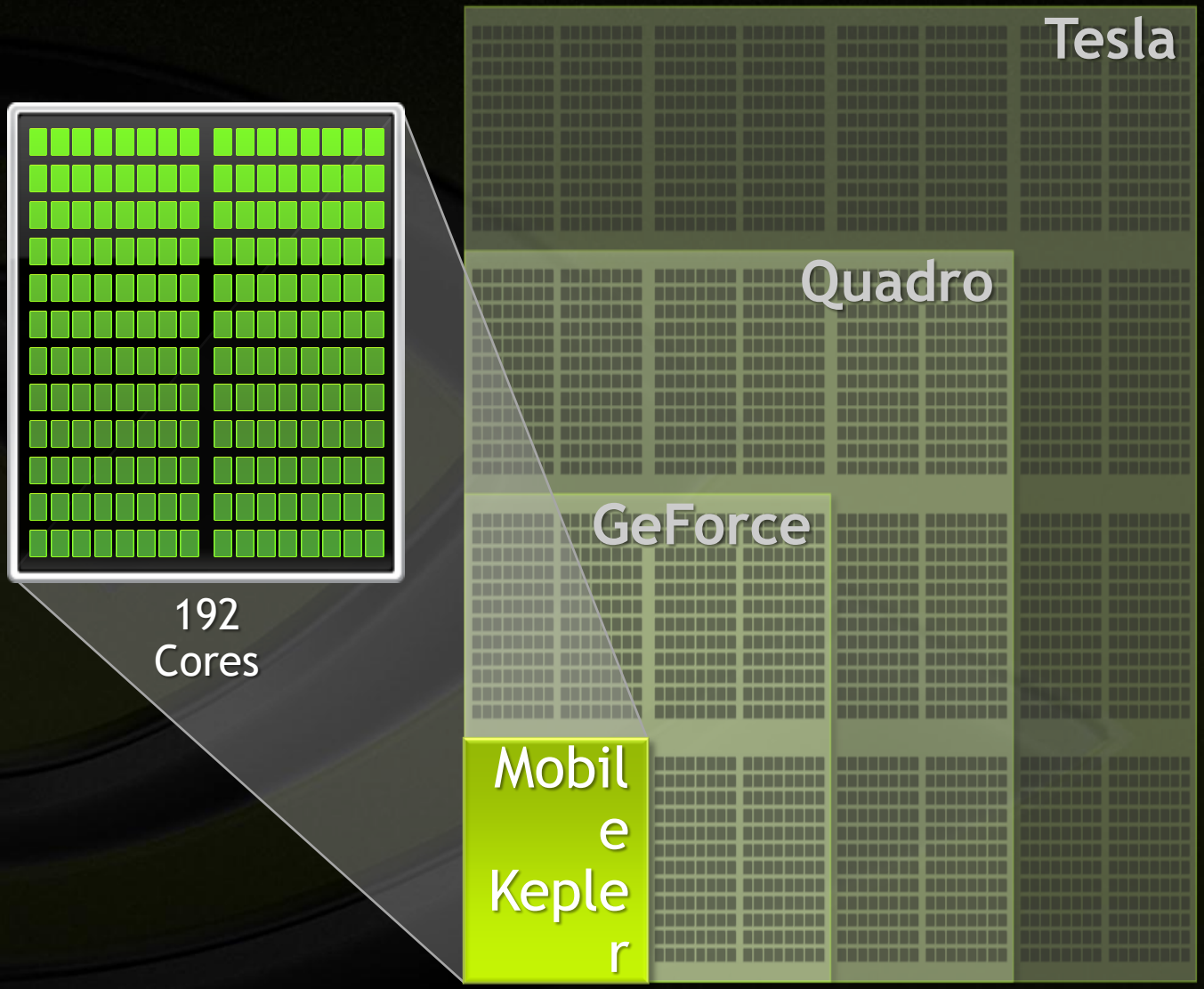


GPU	**Kepler GPU (192 CUDA Cores)** *Open GL 4.4, OpenGL ES3.0, DX11, CUDA 6*
CPU	**Quad Core Cortex A15 "r3"** *With 5th Battery-Saver Core; 2MB L2 cache*
CAMERA	**Dual High Performance ISP** *1.2 Gigapixel throughput, 100MP sensor*
POWER	**Lower Power** *28HPM, Battery Saver Core*
DISPLAY	**4K panel, 4K HDMI** *DSI, eDP, LVDS, High Speed HDMI 1.4a*

On a single Chip there are limits

- Dark silicon - more transistors per chip, but you cannot turn them all on at once within stock form factors.
- From a compiler point of view – no simple cost model
- Power sloshing – There is enough power for one class of cores to run at speed but all the others have to slow down

Can we generate cores that scales?



What do these machine designs mean?

- The question became, what can we do for the poor programmer now? You see programming and debugging were the largest parts of the computer budget, and it was easy to see that, with machines like the 704 getting faster and cheaper, the situation was going to get far worse.
  - John Backus
  - Source: www-03.ibm.com/ibm/history/exhibits/builders/builders_backus3.html

2010 - elastic cloud cores

- The cost model changes
  - Hardware as a service
  - Lots of cores but they come and go
  - Memory is distributed
  - Much more parallelism
  - Software – map reduce/ hadoop, compilers do not understand what is happening
    - Can we minimize cost for a solution, rather than wall clock time?

Challenge

- Remove hand coded data motion
- Remove hand coded core assignments
- Compiler cost models
- How do existing mainstream languages change?

isa

- There is a strong pressure to provide an alternative to ISA
  - Ship in Byte-code, finalize to the native instruction set
- Once byte-code starts being common, there is performance pressure to stop using ISA at all
- As Bytes code becomes common there is commercial pressure to go to one byte-code

observations

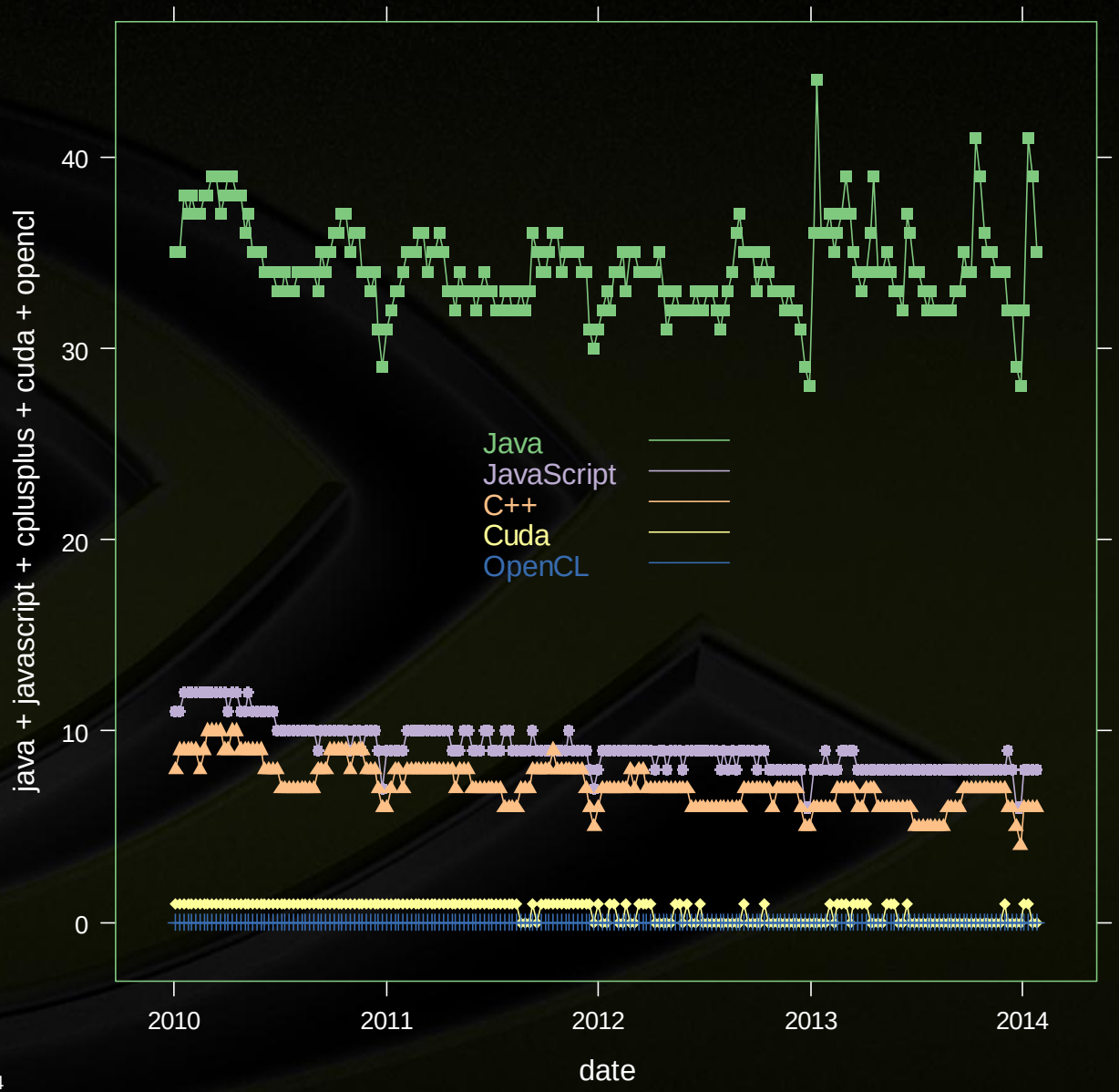
- Common wisdom: ByteCode is slow
- But really its faster than native code
  - Because it can correct hardware errors, and speeds up hardware release dates
- Challenge: one ir to rule them all?
  - Ptx good for NVIDIA
  - HSAIL – one step toward a general ir
  - Can you design a single IR that works over a wide range of parallel devices?
  - Can you design an IR that lets the first part, pass analysis information efficiently?

Programming language changes

- Managed languages – bytecode – are a better fit for multi-isa machines.
- All mainstream programming languages need to change to address parallel programming

- Adding parallelism is a big language change-
- Compare this to object oriented programming
  - C managed to stay mainstream without objects

Google trends



Does performance matter

- Loading a web page –
  - Download, compile, run
  - Performance includes compile time, and source size

Shopzilla: page load 6 sec to 1.2 sec, -> 12% revenue
Amazon: every 100ms improvement -> 1% revenue
Yahoo: every 400ms improvement -> 9% more traffic
Mozilla: removed 2 sec from load -> 60million downloads

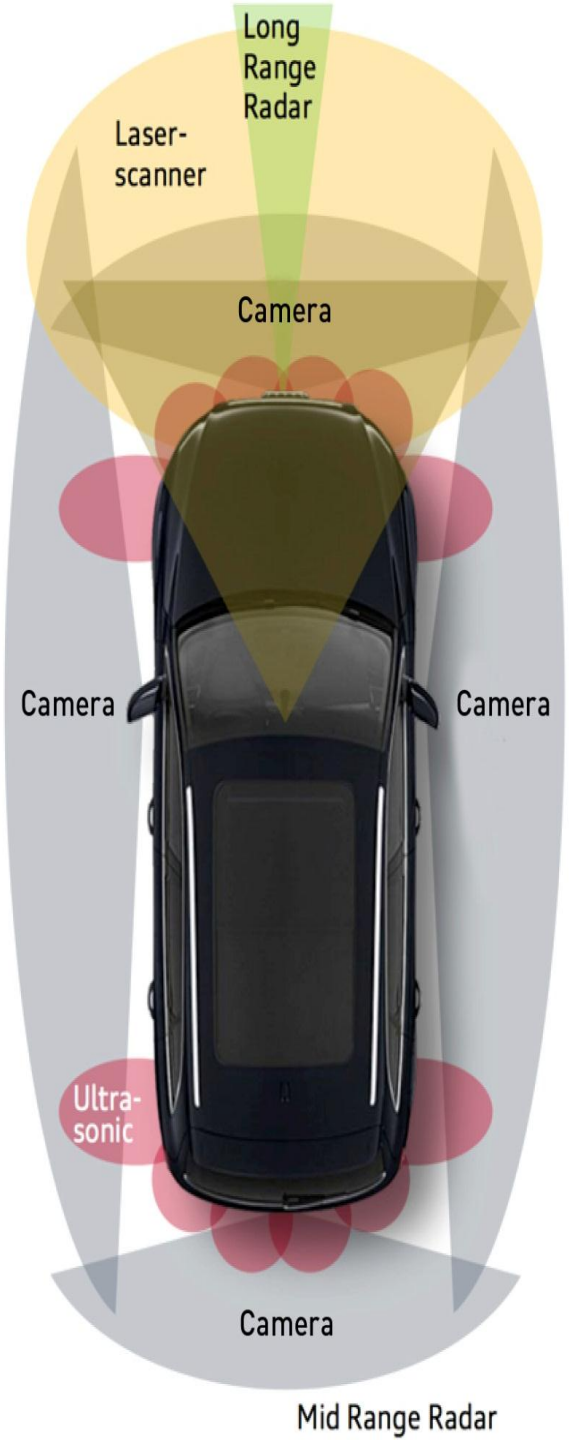
Source: www.strangeloopnetworks.com/assts/images/infographic2.jpg

Meaning of Performance

JavaScript (fast compiler -> ubiquity)

- **Web - Netscape, ie, firefox, safari, chrome**
- **V8 compiler generates fast code**
- **Unity - game engine in JS**
- **Node.js - server side JS**
- **node copter – hardware JS**
- **Phone Gap - mobile JS**

- **Compilers for js have improved performance 30 times in four years – actual success!**

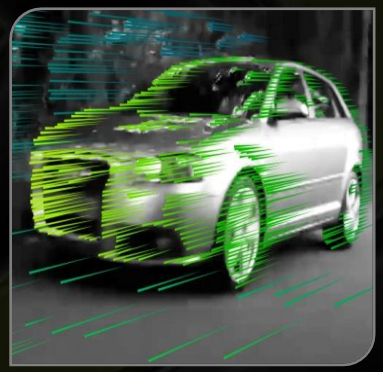


NVIDIA

TEGRA K1 FOR ADAS

Pedestrian Detection
Blind Spot Monitoring
Lane Departure Warning
Park assist

Collision Avoidance
Traffic Sign Recognition
Adaptive Cruise Control
Driver Monitoring

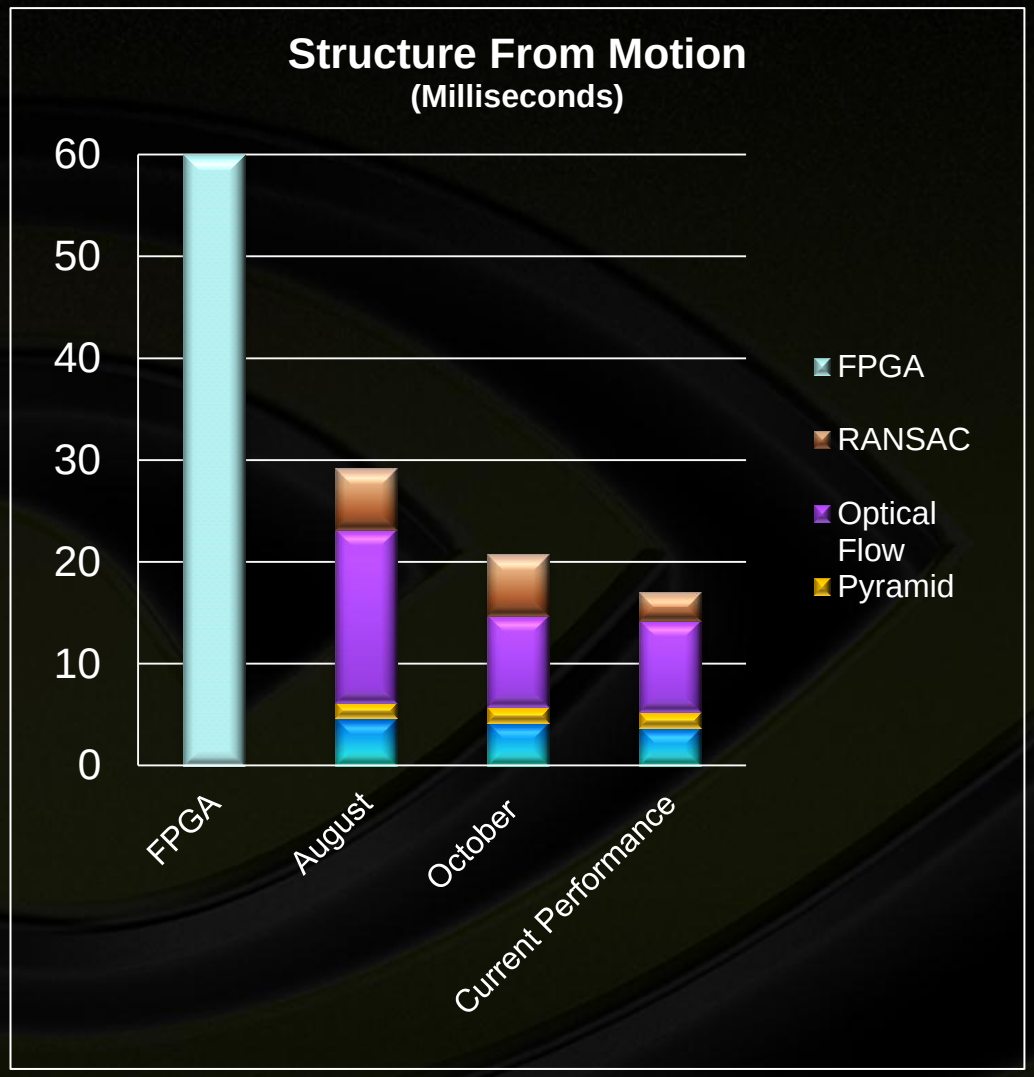


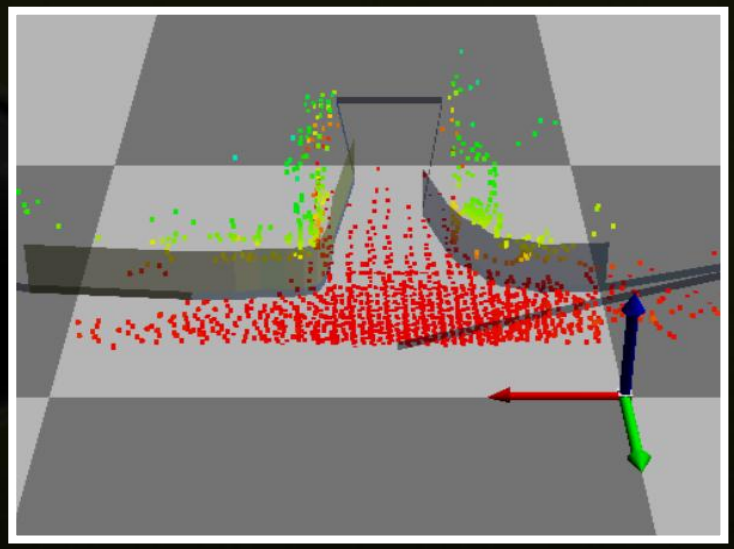
Optical Flow

Histogram

Feature Detection

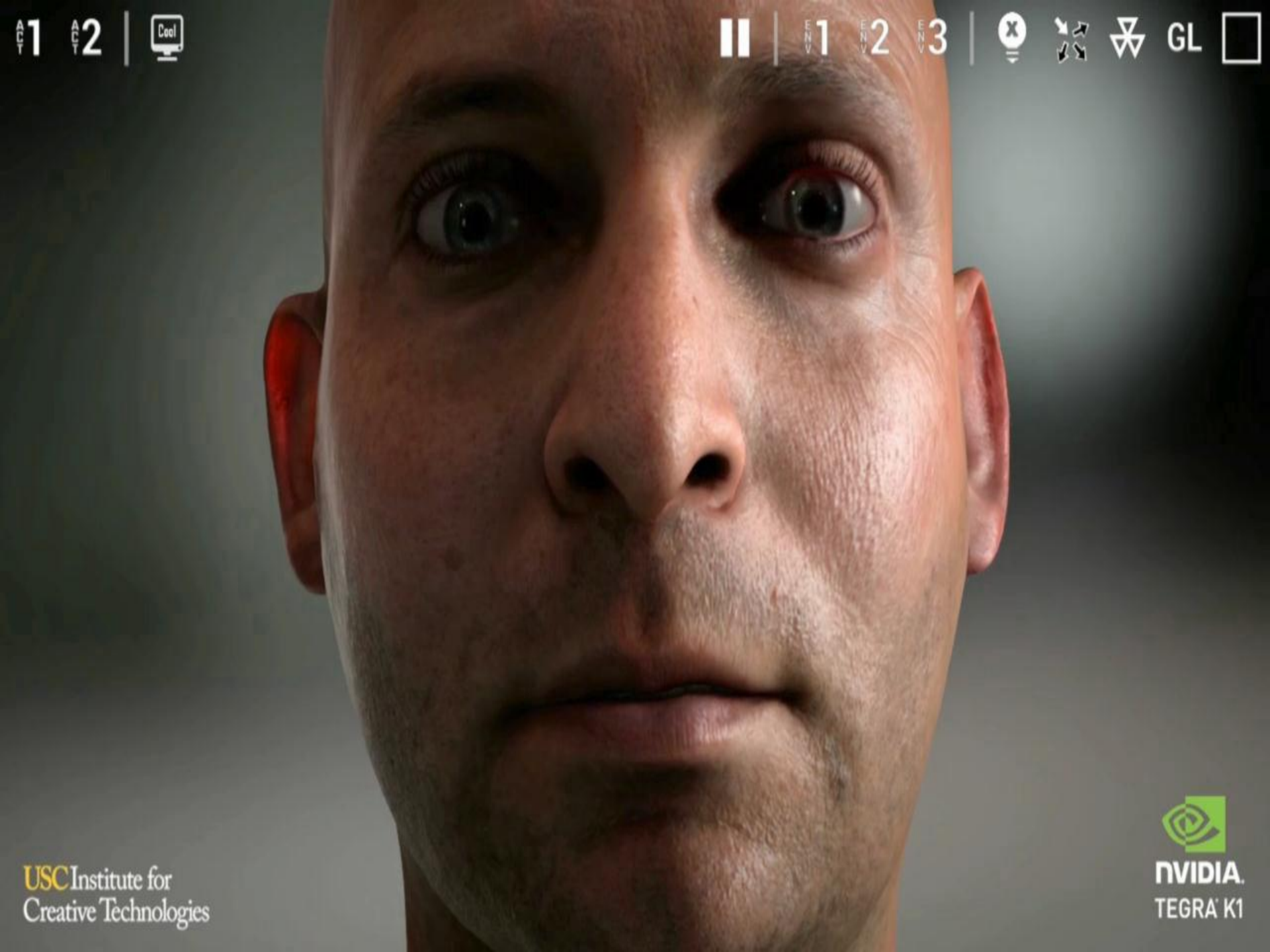
Performance Improvements (SfM)





Initial CUDA port : 25 ms / frame
Current Perf. : 17 ms / frame
NVIDIA Projection : 10–12 ms/ frame


Highscore: 60000
Score: 0


ACT 1
ACT 2
Cool
ENV 1
ENV 2
ENV 3
GL
USC Institute for Creative Technologies
NVIDIA
TEGRA K1

What we would like

- One source program
- Runs on tablet
- Runs faster when connected to a network, uses remote cores (of many types)
- Runs bigger data on the cloud
- Does not require developers to decide what computation runs where
- Does not require massive restructuring every year

What we have

- **One program for a machine with a gpu and a cpu**
- **Another program for cloud, and one for cpu**

One of the things that is interesting about platforms today vs. the traditional desktop is that these cloud services are becoming increasingly central to the core platform experience. This presents a special challenge to an open-source platform, which can't really provide such cloud services as part of the standard platform implementation. In Android our solution to this is to design the platform so that cloud services can plug-in and integrated with it in various ways.

Source: http://arstechnica.com/information-technology/2014/02/neither-microsoft-nokia-nor-anyone-else-should-fork-android-its-unforkable/?comments=1&start=80

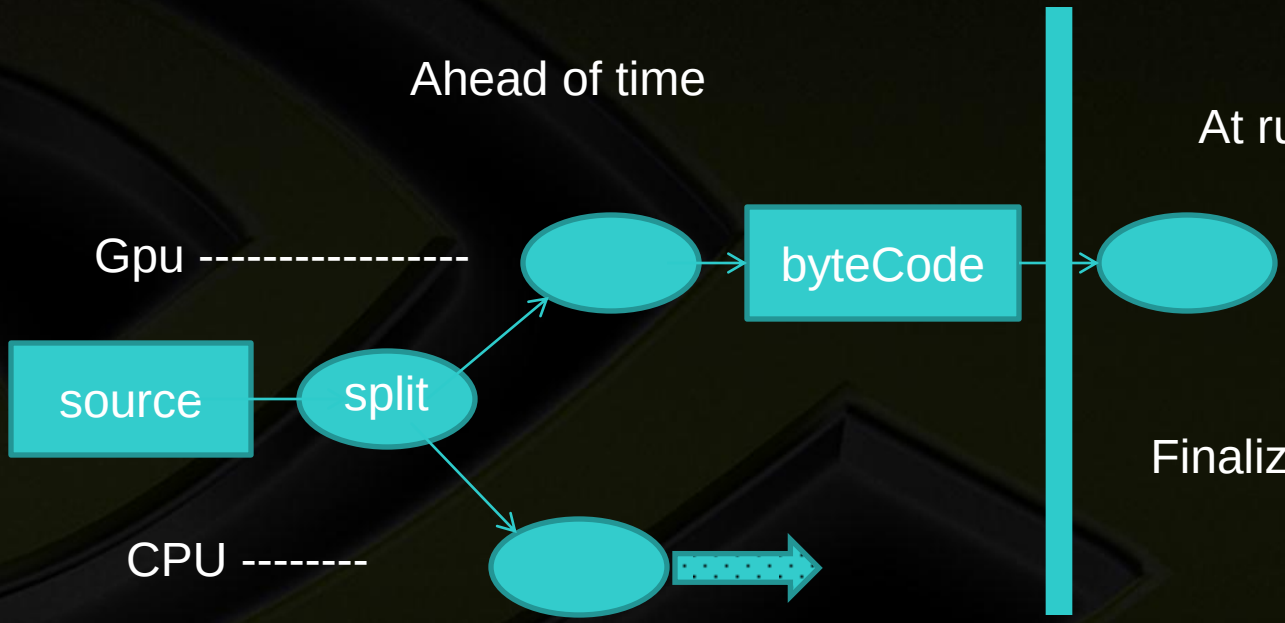
Changes in the compiler landscape

- Compilers used to be standalone tools
- Compile and then ship
- But today compilers are moving into run-times
- So ship then compile

- Hotspot, v8, gpu compilers
- Byte codes and performance

- Compile time counts
- Finalizers do not have flags!
- Challenge: no user flags or knobs

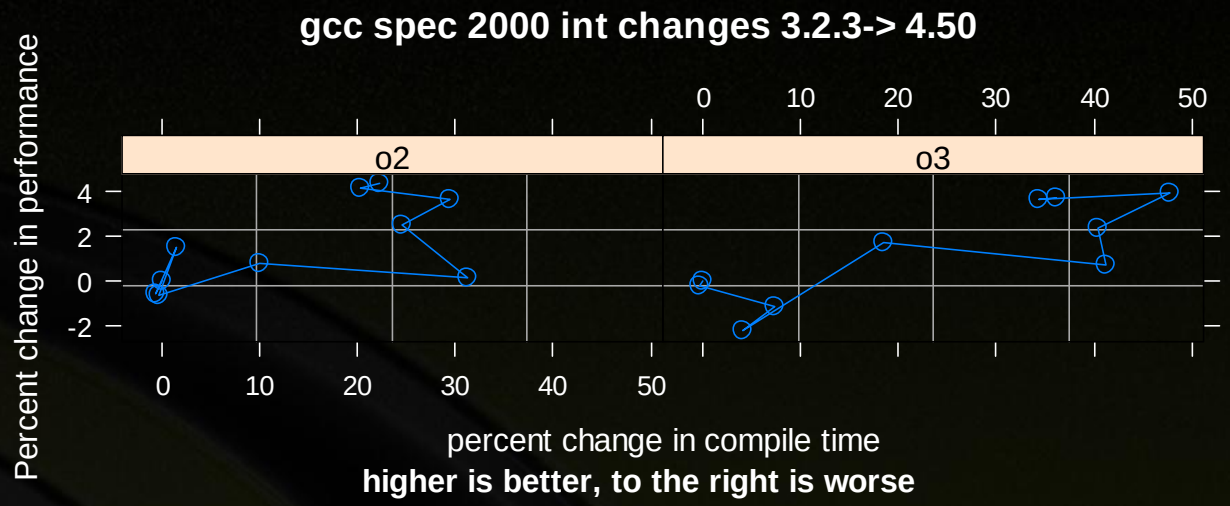
Current GPU compiler model

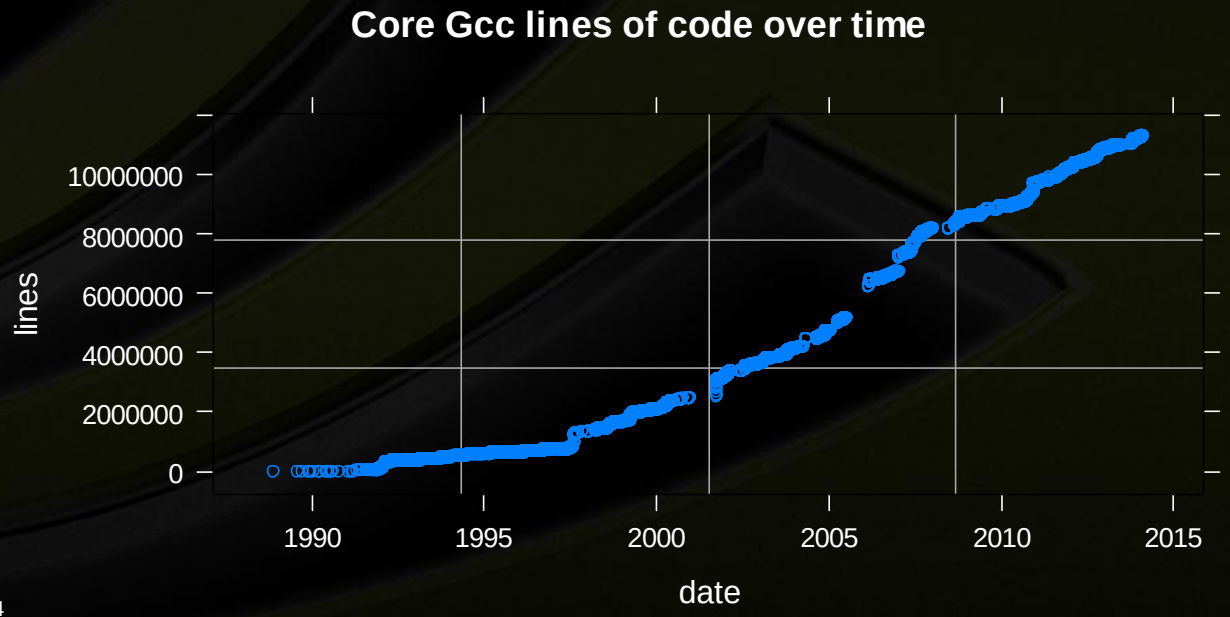


Lots of time/ do not know the device

No time, know the device

Compilers over time





Compilers over time

- Gcc looks good
  - 5% speedup (spec 2000 might be mined out)
  - 50% longer compile time (ignoring adaptive ops)
  - 300% more code (in the core)
- More and more optimizations
  - Some amount of skipping unproductive optimizations
- Challenge – speed improvements do not keep pace with compile time
- Challenge – Can we build reliable software at 1Meg-13Meg lines?

Finalizer as hardware

- For standalone compilers, developers can pick and chose
- For compilers in the chipset/driver/browser/os users have only one option
- Update the finalizer – does not happen
- Move program to a new machine – new finalizer

- Challenge: Can we write reliable compilers?
  - Can we do that and keep optimization

Multiple compilers

- Do multiple compilers cooperate or counteract?
- Compiler unrolls a loop, finalization rolls it back up.
- GPU finalization uses pattern matchers to undo a lot of front compiler work

- If we move code from one compiler to the other at runtime- will the program get the same answer?
  - What about ignoring floating point and rounding?
  - What if the program is not correct.

Incorrect programs

- char* buf; unsigned int len;
- If (buf + len < buf) { // wrong way to check overflow
- report error;
- }
- ----------
- Itm = p->field;
- If (!p) { // wrong way to check for null ptr
  - Report error
  - }

Gpu example

- Spec says threads cannot communicate without synchronizing result is “undefined”
- but real hardware has a magic number – the warp size (might be 32). Threads in a warp can communicate without synchronizing
- Synch is expensive (maybe) so developers like to leave it out

- A[tid] = a[tid +k] // a shared variable
  - Only way that this avoids communication is for k==0

What does undefined mean

- Compilers writers – undefined means anything can happen
- Developers- behavior is a secret

- Rule of three – try the program on three compilers, if it gets the same answer, the code is legal
- Rule of one – try the program on one compiler, if it gets the answer you want, the code is legal

When does undefined matter (defective wrapping check)

compiler	Ptr	Integer	unsigned
	If (p + 100 < p)		
Gcc-2.95.3	-	-	01
Gcc-3.4.6		02	01
Gcc-4.2.1	o0	-	02
Gcc-4.8.1	o2	02	02
Icc-14.0.0	-	02	01
Msvc-11.0	-	01	-
Armcc-5.03	-	-	o2

Source Xi Wang et.al. SOSP 13, 2013

Correctness

- Some developers think invalid programs form a hierarchy-
  - Benign errors, catastrophic errors, warnings?
  - = p->x
  - If (p == NULL) { report error }
  - C spec says dereference of null ptr is undefined
  - Since there is a dereference, the compiler can assume p is not null and remove the if.

Early versions of GCC did not notice, but one day GCC did notice

Cert vulnerability note vu 162289

- **Avoid affected compiler implementations**
  Application developers and vendors of large codebases that cannot be audited for use of the defective wrapping checks are urged to avoid using compiler implementations that perform the offending optimization. Vendors and developers should carefully evaluate the conditions under which their compiler may perform the offending optimization. In some cases, downgrading the version of the compiler in use or sticking with versions of the compiler that do not perform the offending optimization may mitigate resulting vulnerabilities in applications.

How compiler writers view this

On two occasions I have been asked 'Pray, Mr. Babbage, if you put into the machine the wrong figures, will the right answers come out?' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question

- **Charles Babbage**

- Source: www.brainyquote.com/quotes/quthors/c/charles_babbage.html

The gcc response

- Added a new flag
  - Which turns off the optimization
    - Sometimes (sadly there was a bug in the patch)
    - Non-standard dialect of C
- New flag only works for next version of the compiler
- Will not retrofit the change into the past
  - Add flag to old versions of the compiler
  - Get all fielded copies to change

Even compiler writers make this mistake

- Firefox
  - Js - Allocate an array, set the length to something big
  - Do a reduction using a user supplied function on the array from the right
  - Var a, b,c,d = -6.828527034422786e-229;
  - // string of x86 noops

  - Length is unsigned, reduceRight walks the array using signed numbers, did the wrong check for out of bounds,
  - makes a call to a random point, sometimes it lands in the constant.

Challenge of incorrect programs

- Can we detect these without false positives
- Can we define languages where “undefined” is limited –
  - Java tried this (no race condition can turn into a security hole) but that effort ran into trouble

Two compilers in the path Open up a new way to look at optimization

- Split compilation
  - The first compiler has lots of time, but no ideas about the target machine
  - The second compiler has no time but lots of ideas about the target machine
  - Split the optimization over both compilers
- First compiler does an optimization, the second pattern matches the ir and undoes the optimization and then does it itself

Split compilation

- SSE – lots of flavors – several prototype compilers that recognize vector opportunities in the first compiler and then pick the specific vectors ops in the second
- Why is hadoop written in Java?
  - Compile to java bytecode
  - Let hotspot compile to isa
  - Hotspot looks across libraries, static compilers do not
- How to annotate ir?

Using an ir to transfer info

- Register allocation
- what can a compiler do for register allocation if it has lots of time but no knowledge of the number of registers?, and instructions might be re-ordered

- Computing spill costs is expensive

“likely to spill” in ir

- Some compilers will not generate it, some finalizers will ignore it.
- Time in a register allocator is related to the number of simultaneously live ranges.

- Goal – finalizer has to be fast, so reduce the number of live ranges but not so much that it needs to spill a lot
- If it does spill make that fast

Statistical views

Observation: a live range that spills given N registers are statistically likely to spill given N+1 registers

So the compiler could assign a spill probability to each live range – Diouf, et.al. "Split Register Allocation", HiPEAC'10

Some evidence that "likely to spill" is still true if the code is rescheduled

The first compiler could identify "likely to spill" live ranges, the second compiler could use that info or not

HSAIL and spilling

- How does the ir represent likely to spill, and allow for multiple finializers

- HSAIL:
  - Small fixed number of registers and infinite set of spill slots, split slots are a mix of memory and register
  - Spill slots are a special memory space so they never interact with normal loads and stores
  - The spill slots are either less important registers or they are memory. Different finalizers can pick and chose

What about all the other optimizations

- Challenge: How do we split info so that time consuming parts are done ahead of time?

The challenges:

- new notions of performance
- mainstream languages change
- cost models
- performance portability
- single ir to rule them all
- compiler reliability
- incorrect programs
- No user input or flags to the compiler
- split compilation
- interaction of optimizations
- speed improvements do not keep pace with compile time
- Remove hand coded ...

Questions?