Programming Languages & Emerging Hardware

Peter Braam
peter@braam.io
Physics Dept, Oxford
Oct 2019

me

math physics @oxford (gauge theory, CFT)

distributed systems cs @cmu

@6 startups & jobs @3 acquirers - Lustre

SKA @cambridge

@oxford

taught OS course for “Computing Laboratory”

work with 100’s of largest compute centers and many major system & CPU/GPU vendors

New Hardware

Revolutionary Hardware

- AI/ML accelerators
- Image processing accelerators

Evolutionary Hardware

- Faster RAM
- Persistent RAM
- Open Instruction sets - RISC-V
- New number formats
- Very many cores

Anti-development: increasing network & memory bottlenecks

Software support for new Hardware

Domain Specific PL support

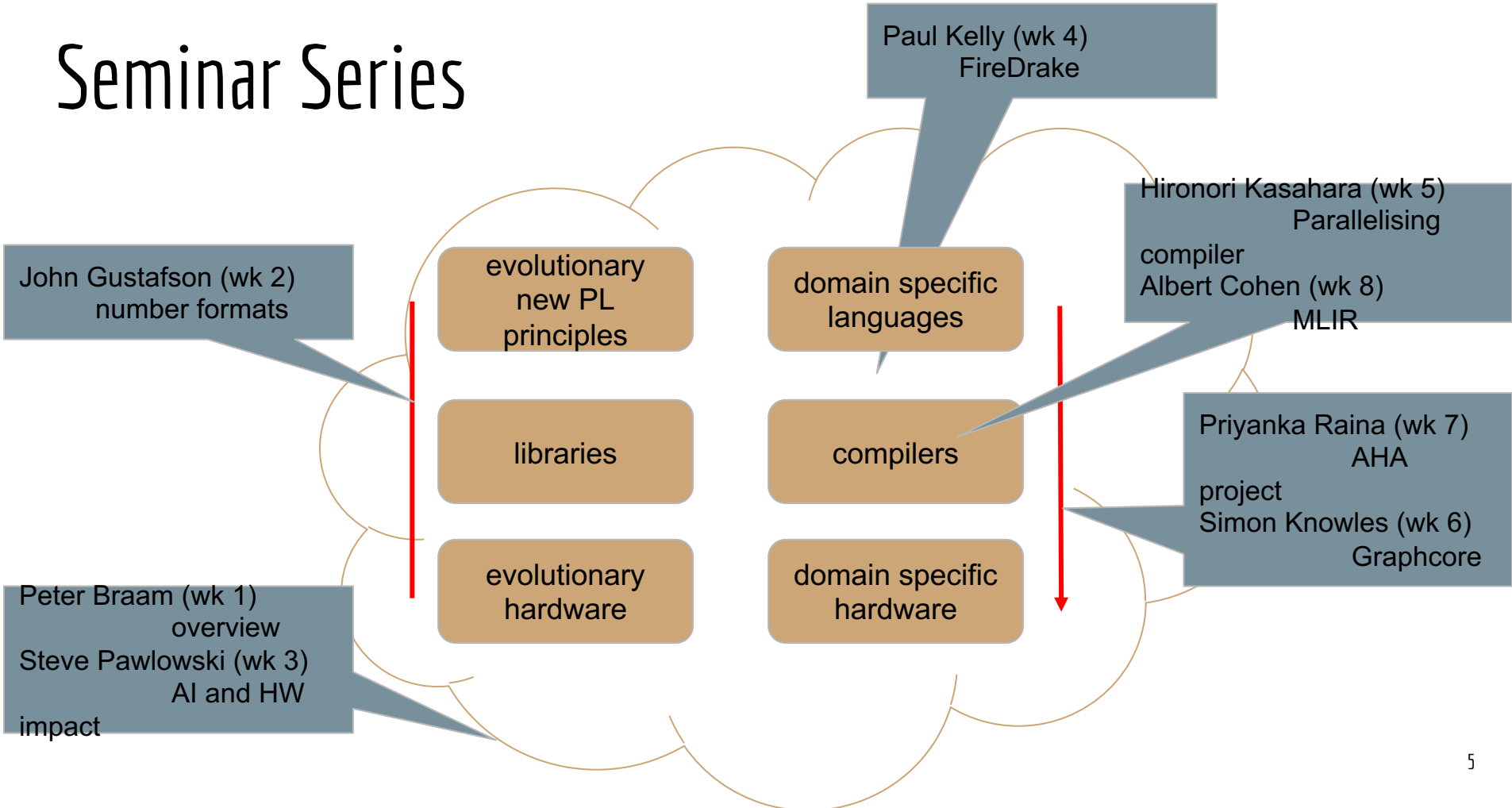
- ML as overpowering successor to HPC
  - user level
  - compiler level
- Mathematics as program specifications for ML and HPC

Evolutionary Software

- Dealing with persistent RAM
- Cache coherency many cores
- Programming for hybrid memory types
- Great diversity of HW

- Evolution away from von Neumann architecture
- Industry developing widely adopted practical solutions
- Great opportunities for research

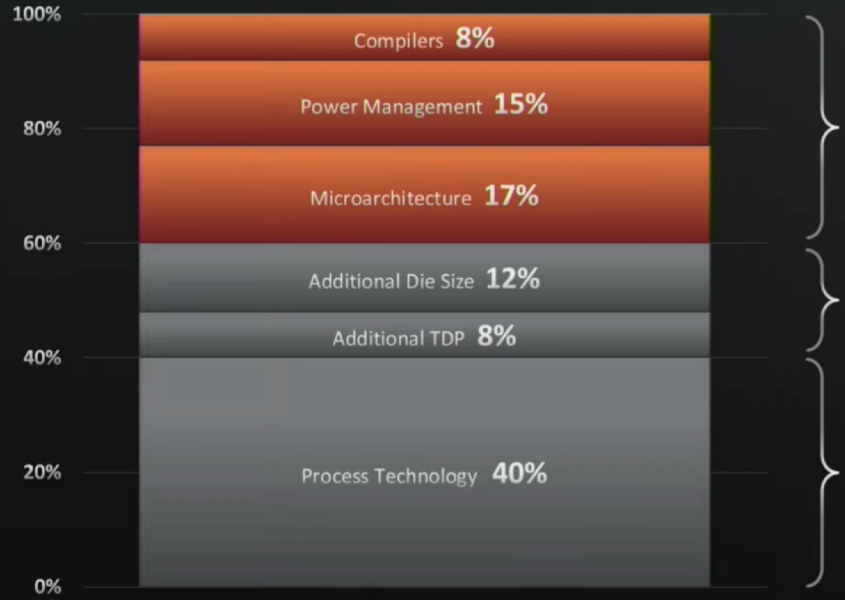
Seminar Series



From HotChips 2019
AMD Keynote

PERFORMANCE GAINS OVER THE PAST DECADE

Lisa Su, CEO



- Integration of System Components
- Micro-Architectural Efficiency
- Power Management
- Software

- More Silicon Power
- Bigger Die

HIGHER PERFORMANCE, DENSER, LOWER POWER TRANSISTORS

ELEMENTS OF 2x IN 2.5 YEAR PERFORMANCE GAIN OVER THE PAST DECADE

Hardware

Hardware this talk

In this talk I will only focus on memory & accelerators.

Memory Technologies

Now:

- Faster on-package memory: HBM
- Persistent memory: Optane / xPoint

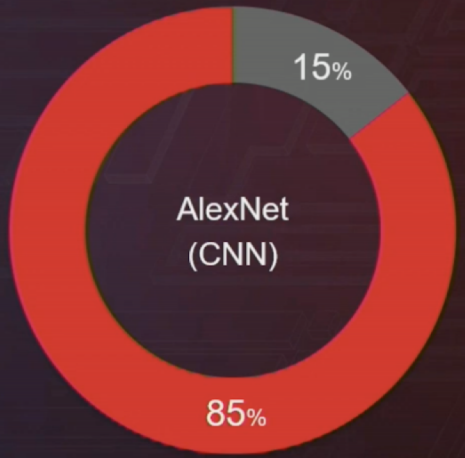
Coming:

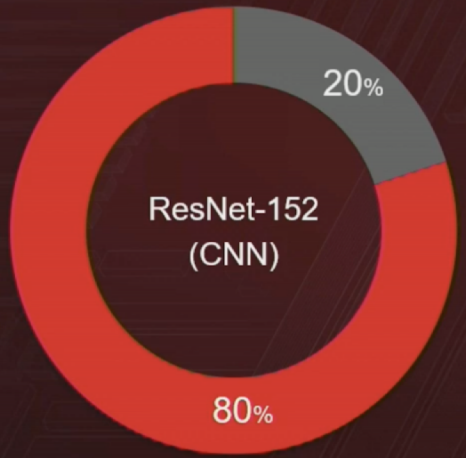
- At least a few dozen memory technologies are being explored
- 3D memory / logic integrated chips

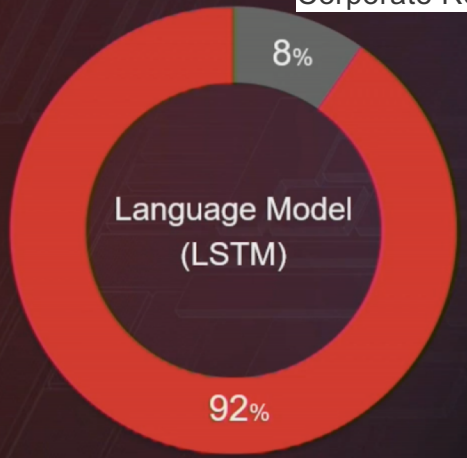
DATA MOVEMENT HITS THE MEMORY WALL

ABUNDANT-DATA APPLICATIONS: ENERGY MEASUREMENTS

Hotchips 2019,
Dr. Philip Wong, VP
Corporate Research,
TSMC







Memory Compute

Deep Learning Accelerators

Source: S. Mitra (Stanford)

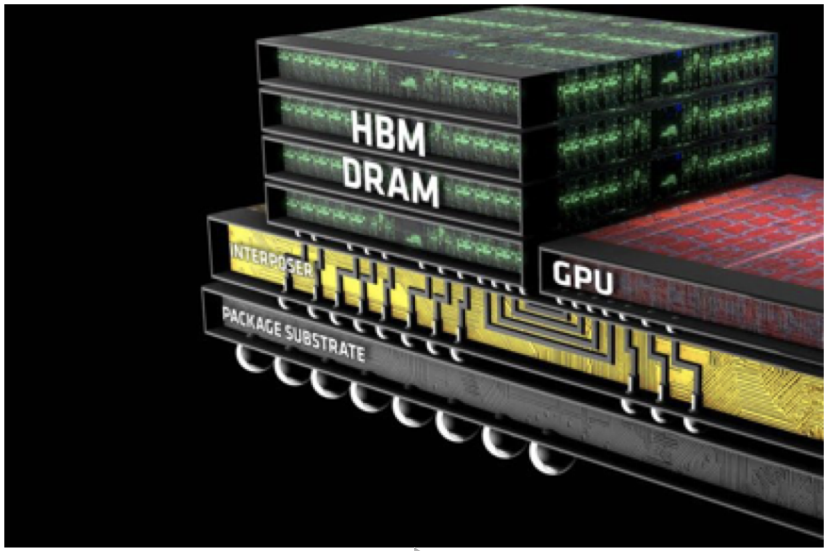
Intel performance counter monitors 2 CPUs, 8-cores/ CPU + 128GB DRAM

HBM-high bandwidth memory

Key features:
- HBM2 16GB capacity
- 250-400 GB/s BW
- ~5-10x > DRAM BW

Hardware control:
- HBM is a cache
- HBM is part of address space

Technology:
- 2.5D - using many vertical TSV's (through silicon via)
- on-chip **interposer with** 1024 (instead of 32) signalling pins (explains 10x BW increase)



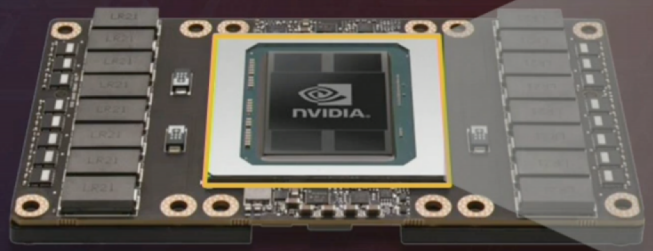
packages with integrated memory and compute are becoming standard

SUPER AI ACCELERATOR
ENABLED BY CoWoS®

Many chips and accelerators are moving towards this architecture

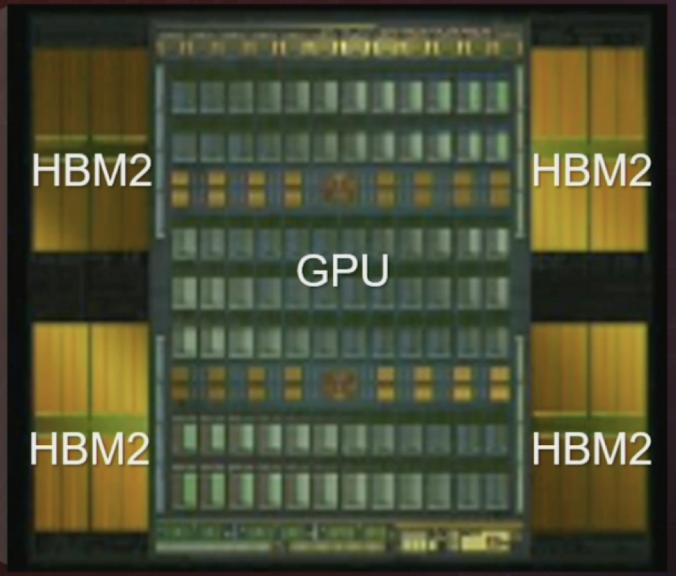
chip on wafer on substrate

CoWoS Module



[redacted]

Heterogeneous Integration:
GPU + High Bandwidth Memory (HBM2)



>300 B transistors

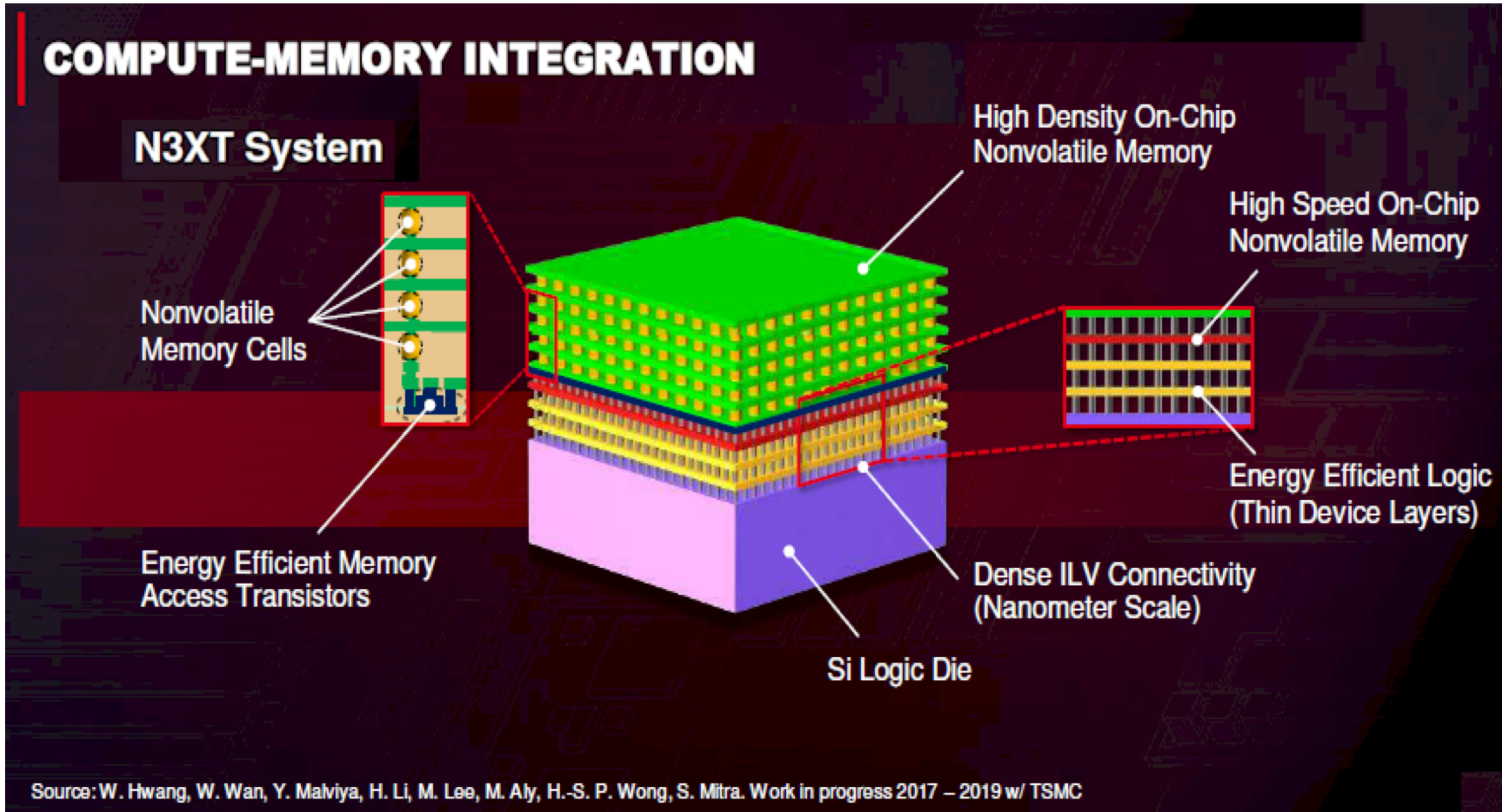
Source: "Inside Volta" , Nvidia GPU Tech. Conf. , May 10, 2017.

Memory - Logic Integration Will Likely Continue

N3XT project models and samples. Target perhaps 2030's technology

1000x Energy gain
10x lower latency
100G-1TB on package

Key innovation: logic (for compute) and memory created with one process (PIM - blue)



Key issues with memory

- Apps always want more in one node
  - distributed execution is seriously hampered by network bandwidth
- Most applications are seriously memory bound
- Latency remains high for apps moving small data
- Energy of data movement is major consideration:
  - HBM - 50pJ / byte
  - Large scientific experiments are looking for 100 PB/sec - i.e. 5 MW in data movement

Persistent Memory - Optane

Source: Intel, available now

Many new memory technologies promise "persistent RAM".

- byte addressable
- non-volatile

Intel Optane (xPoint) is first commodity product.

Cheaper than RAM, costlier than Flash.

Spin Transfer Torque (STT) RAM may move the numbers by at least a further order of magnitude

watch orders of magnitude

Technology	Latency		Bandwidth
	Rd	Wr	
DRAM	50ns	50ns	13GB/sec
NAND flash	50us	15us	3GB/sec
Disk	10ms	10ms	100MB/sec

Large memory configurations

Typically hardware has provided cache-coherency among pieces of memory: L1,2,3 and RAM.

Increasing core counts

- consume a lot of chip real estate
- consume energy
- cause performance overhead

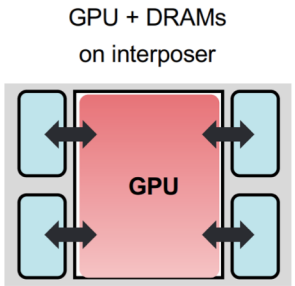
Virtually no instructions exist to control memory caches.

Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture, Daniel Molka etal, ICPP 2015

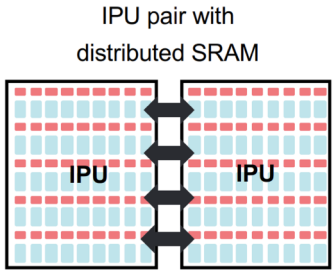
Multicore Cache Coherence Control by a Parallelizing Compiler, Hironori Kasahara, Keiji Kimura, etal, COMPSAC 2017

Compiler driven cache control can be more efficient. An extreme example is the Graphcore IPU accelerator & SW stack:

- 2000 cores, each with “local memory”



16GB @ 900GB/s



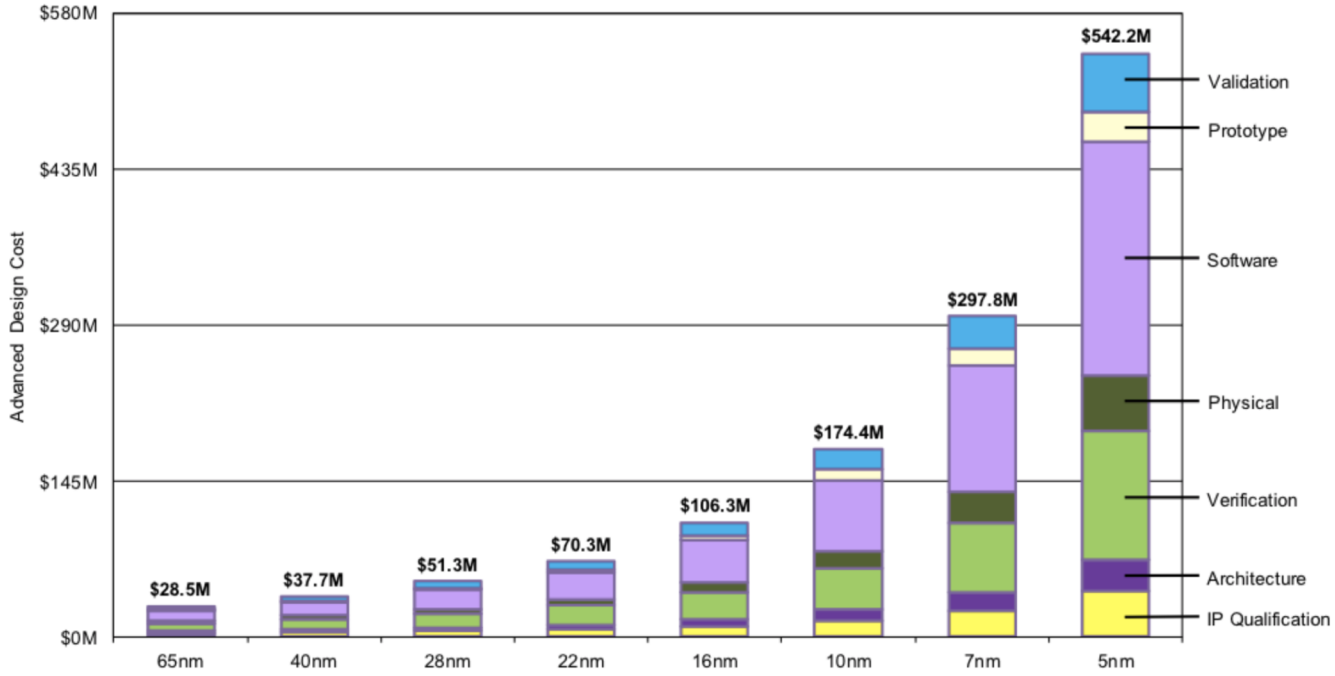
600MB @ 90TB/s, zero latency

Accelerators

Chip Development is Hugely Expensive. AI is providing a market where it is economical.

- Google TPU
- Google's Pixel Visual Core
- Perhaps around 100 accelerator companies, big & small exist
- Investors tell me all startups fight huge software battles for compilation!
- Architectures vary widely
- Reason: power, cost, performance

Chip Manufacturing is expensive



High performance chips need newest processes (low nm's).

Profitability requires shipping high number of units, like 1M.

ML and phones among few areas where this seems possible

Focus is shifting to "domain specific hardware" - c.f. Hennessey & Patterson Turing award.

Source: IBS, 2018

Accelerators

Commercial successes for high performance accelerators have been modest, with notable exceptions:

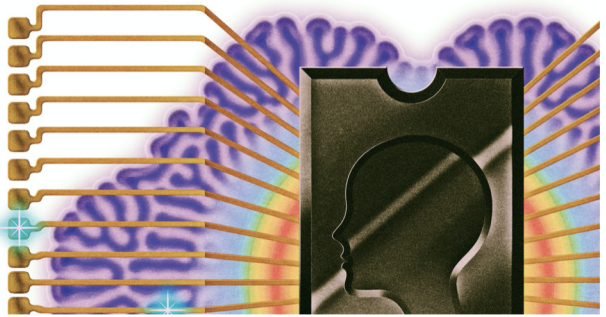
- reconfigurable computing (FPGA's)
- GPU's
- ML accelerators
- Google: TPU's and Pixel Visual Core

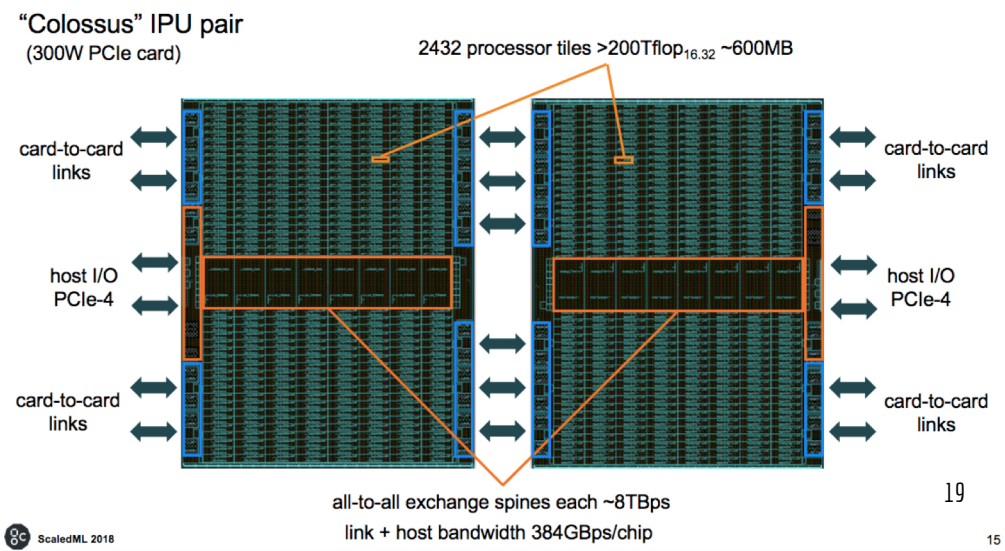
There are **many** startups and main-stream efforts for reconfigurable computing and ML - NYT mentioned 45 in Jan 2018.

Wk 6, Nov 20, Simon Knowles, Graphcore

The New York Times

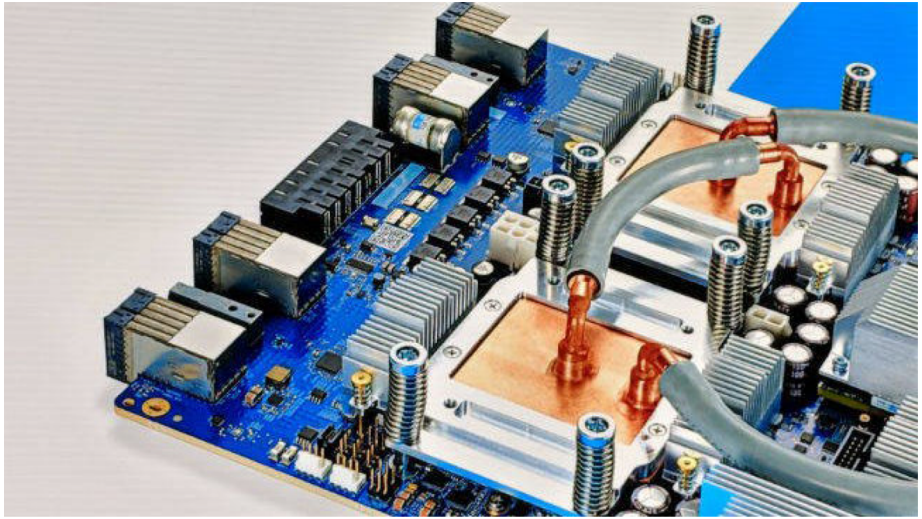
Big Bets on A.I. Open a New Frontier for Chip Start-Ups, Too





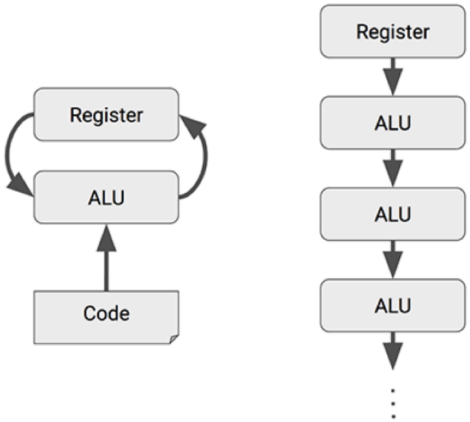
TPU - PCI ML accelerators: cards & clusters

TPU v3 PCI accelerator card

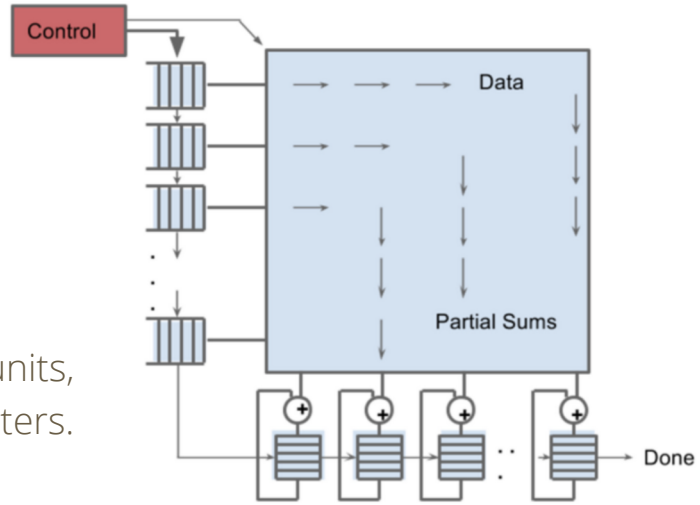


TPU v3 POD (“cluster”)

Example of ML hardware: TPU chips



TPU chips have 4 **systolic arrays** MXU (matrix multiply unit) reducing memory accesses by ~100x:



Pass data between ~100K ALU's. Small processing units, using a global clock, no registers.

Only for TensorFlow ops.
~100T Ops/cycle (limited precision)

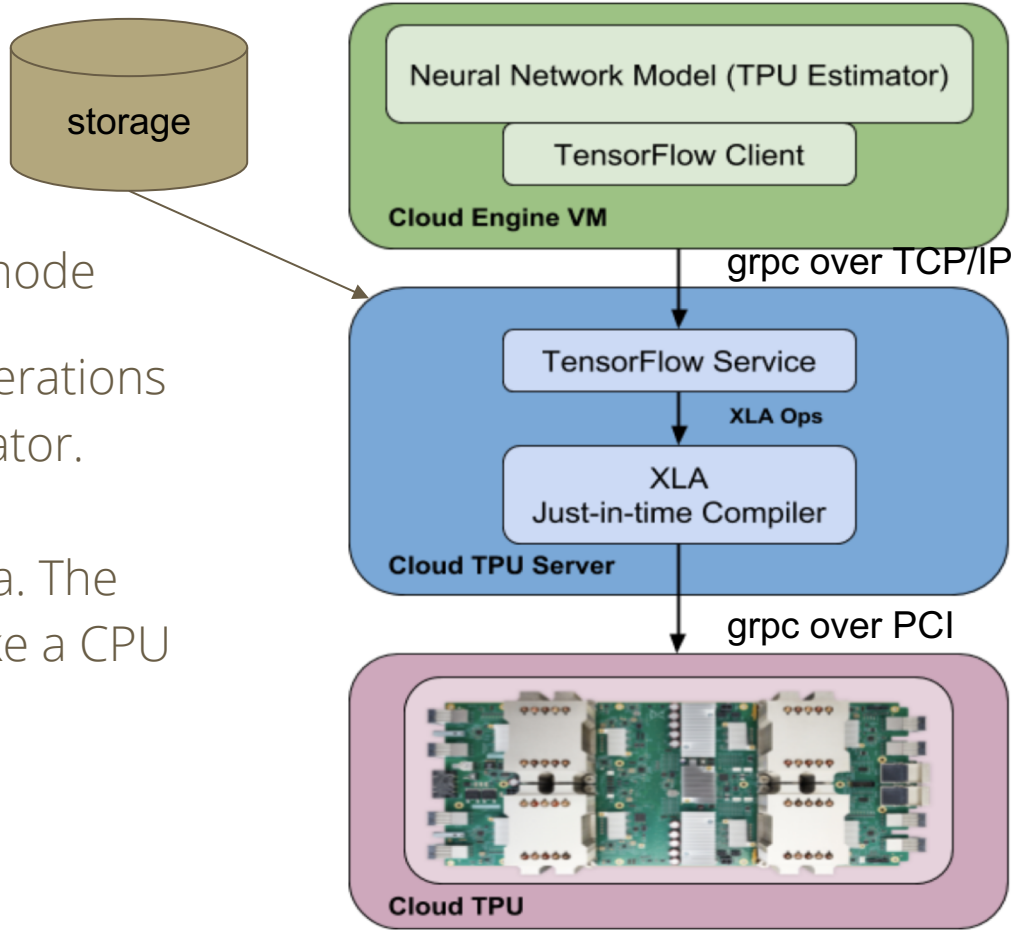
Matrix Multiplier Unit (MXU) of TPU

System Organization

Send TF graph as a whole to a TF node

Send individual XLA generated operations with their data to the TPU accelerator.

This includes instructions and data. The TPU does not fetch instructions like a CPU



TPU v3.0 specs

	TPU 3.0 card	TPU 3.0 node	TPU pod
#TPU's	1 card, 4 chips, 16 MXU	4 cards	1024 cards, 256 nodes
mem BW	5 TB/sec	20 TB/sec	5 PB/sec
flops / sec (*)	100 TF/sec	400 TF/sec	100 PF/sec

Operations per clock cycle

CPU 10's
(cores)
CPU vectorized 1000 (core x
vector length)
GPU 10K 's
TPU 128K
(TPU v1)

Instruction model is TensorFlow op RPC to chip

Read_Host_Memory
Write_Host_Memory
Read_Weights
MatrixMultiply/Convolve
Activate (ReLU, Sigmoid, Maxpool, LRN, ...)

This should raise eyebrows …

256 nodes for 5PB/sec of BW and 100PF ???

pretty much a top 5 machine in top500.org with 100x fewer systems (or 25x fewer allowing for 16 vs 64 bit precision).

It would work very well for moderate granularity computations, like SKA (and AI for which it was made). Wouldn't help with AMR likely, but surrogates may do that.

GPU's around 2003 evolved to GPGPU's through HPC.

Possible further enablers:

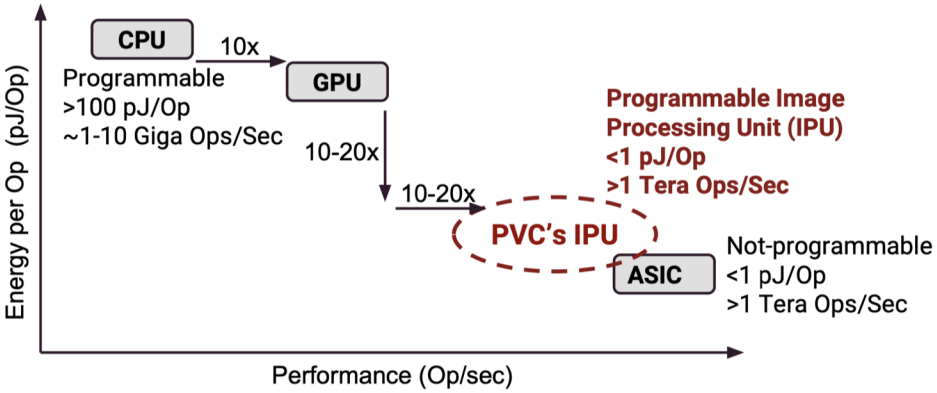
1. Are TensorFlow operations sufficient for HPC?
2. does a more general systolic network interconnect offer more opportunities?
3. Is mixed floating point precision required? (cf. posithub.org)

PVC - pixel visual core (Google)

Objective: save power for image processing (and AI) in cameras.

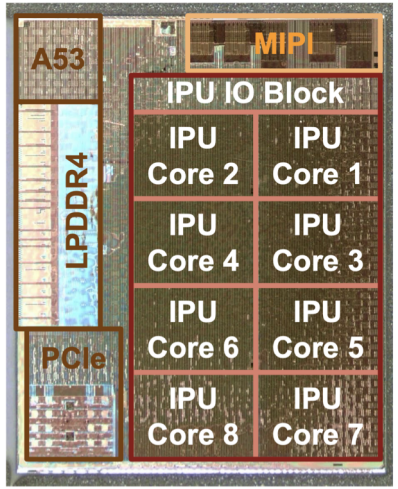
Keep it programmable (Halide subset).

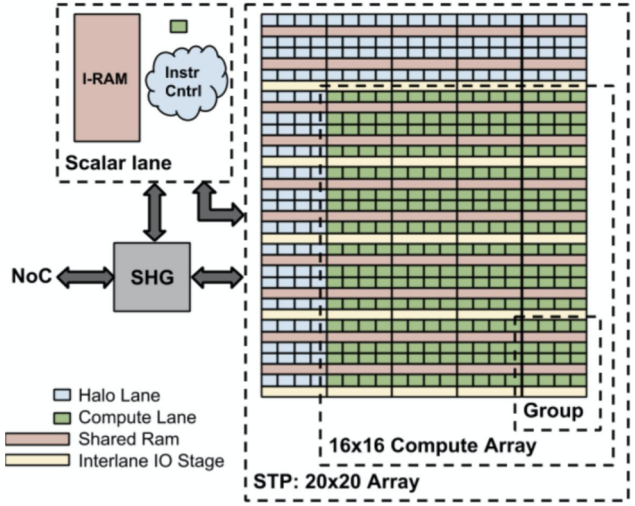
Internal architecture is complex: line buffers connect stencil processing cores



PVC has IPU's
Each IPU has

- memory
- 8 STP and 8 LBP
- comms





Each STP has:

- 256 cores
- ALU focus
- neighbor memory access

Languages and Compilers

Persistent Memory

Great opportunities but with software challenges

In practice, used as RAM or storage device :(

Persistent Memory

Much memory will become persistent, but not SRAM used for caches, registers etc. Hence PM requires specific cache flush machine instructions.

PM reads currently on-par with RAM, but writes are 10x slower. Hence PM will be used in conjunction with faster memory for caching.

Restartable programs

But transactions require lock ordering and do not compose. Software Transactional Memory overcomes this, can be used with PM[1].

Key challenge 1: automatically transform code for volatile memory data structures into code that is meaningful with persistent memory. Even for a linked list, this is not so simple[2].

Key challenge 2: If all or most memory in computers is non-volatile - programs should restart after battery exhaustion (and maybe after failures)

1. Composable Memory Transactions, Tim Harris Simon Marlow Simon Peyton Jones Maurice Herlihy
2. Relayed to me by Intel's lead of PM software, Andy Rudoff

Memory Tiers

Hardware Mechanisms Rule

Probably extremely good software opportunities

Tiers of memory / memory layout

Compiler is targeting systems with complex data in a complex memory layout

- multi dimensional arrays with tiles, slices, alignment, etc.
- storage class memory, RAM, HBM, level 1-3 caches
- cache coherency sometimes for dozens of cores

It is unlikely that hardware memory management solutions do a great job.

Memory Tier Opportunities

Array calculations and HBM

- Consider ultra-popular package for array calculations: NumPy
- NumPy uses Python memory management with optimized kernels
- **When should NumPy arrays be cached (e.g. placed in HBM)?**
- Suggestion (Simon Peyton-Jones): the garbage collector is best place to make this decision (in this case Python's GC)

John Gustafson, week 2 may say more

Compressed data movement

- In scientific problems, precision is lost only during computation
- Data movement, storage is very costly
  - e.g. new radio telescope needs 200 PB/sec memory bandwidth
  - cost is 10 MW in power
- **Major improvement: store and transport compressed arrays of adequate, lower precision. Decompress only during computation.**
- Managing near-core data out of "the address space" not well supported

Compiler Challenges

LLVM innovation & challenges

MLIR

LLVM

Around 2000 LLVM was invented

LLVM is a software virtual machine
separate HW specific aspects from remainder of the compiler

Apple based its C compiler on this CLANG, numerous other languages followed.

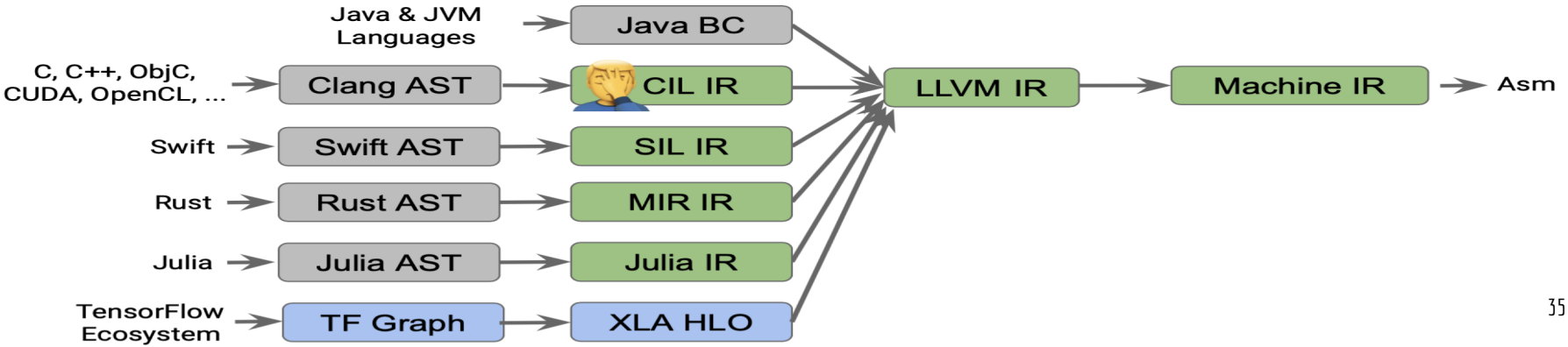
in 2010's abstractions for vector instructions, GPU's and FPGA's were embedded in LLVM. Even CUDA compiler leveraged LLVM.

Interview: Chris Lattner: Compilers, LLVM, Swift, TPU, and ML Accelerators | Artificial Intelligence Podcast, https://bit.ly/31iFY7o

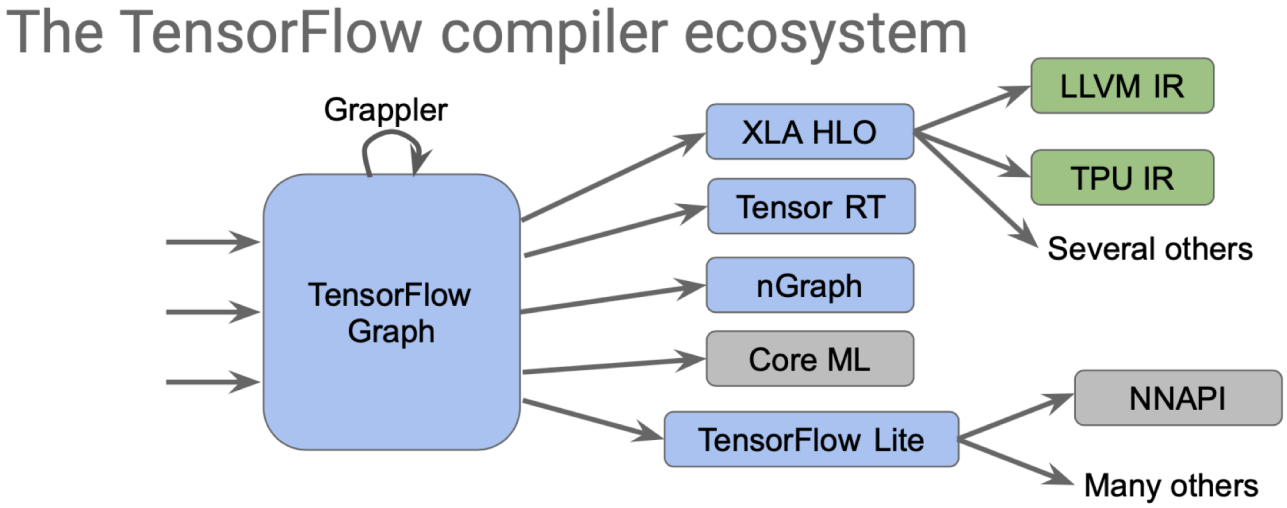
However … language specialization

- Optimizations often remained part of the compiler above LLVM
- New languages required new intermediate layers which LLVM didn't support
- Much duplication in optimization

- The ML accelerators came with completely different instruction sets
- Usability of the languages was affected by this.
- During Apple's Swift development and Google's TensorFlow this came to a head.....



Tensorflow intermediate formats



TensorFlow targets:

- CPU
- GPU
- distributed
- TPU / edge TPU
- training and inference mode
- edge devices

TF Optimizations:

- Fusion
- Graph pruning
- Graph splitting
- Constant handling
- Array tiling

MLIR - Google project

Multi-Level Intermediate Representation Compiler Infrastructure

- similar to LLVM in spirit, addressing recent issues
- dialects with custom instructions (e.g. LLVM dialect, TF dialect)
- flexible definition of SSA intermediate formats
- generically usable polyhedral optimization
- traceability

Hope: eliminate all custom IR's in the world. Ambitious, but great!

Github: https://github.com/tensorflow/mlir
Wk 8, Dec 4: Albert Cohen, MLIR.

Programming in Mathematics Domains

LLVM innovation & challenges

TensorFlow Complexity

MLIR

Machine Learning / Scientific Computing

Important/Shameless Advertisement: Thursdays 14:30 I and Miha Zgubic run “ML & Physics” seminar in Wilkinson Building.

What is special about Machine Learning and its eco-system?

Answer: extended types! Tensors in the language, tensor-friendly hardware.

- Scientific computing takes this further: tensors + differential / integral operators. Applied mathematics uses dozens of types.
- Pure mathematics uses hundreds of types (Sheaves, Number fields...)
- Programming languages only mildly helpful.

Scientific Computing Problems

Scientific problems often have an extremely short full specification along the following lines:

- solve a differential equation F(D, v) = 0 for v, given the operator F(D, -)
- let v obey initial and boundary conditions
- select a mesh, discretization and solver to approximate the solution
- here we enter “known programming terrain”

If the programming language is enhanced with types and a calculus for these mathematical objects, programming becomes much simpler.

Nov 6, Wk 4 - Prof. Paul Kelly, Firedrake .

Domain Specific Languages

Very many domain specific languages have been developed:

- for Machine Learning
- for High Performance Computing
- Image Processing

A few are hugely successful: NumPy, TensorFlow, Halide

However:

- most DSLs rely on complex library functions which, e.g. cannot be fused
- many chosen domains are too narrow. E.g. Halide is truly a fantastic image processing DSL, yet not for noisy, sparse astronomy data
- there are too many DSL's

Halide DSL example

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

Halide image processing

State of the art in separating computation from mechanism.

Is the future perhaps something like the following?

Schedule: a function between linear/dependent types representing:

- data in the computation
- hardware in the computer

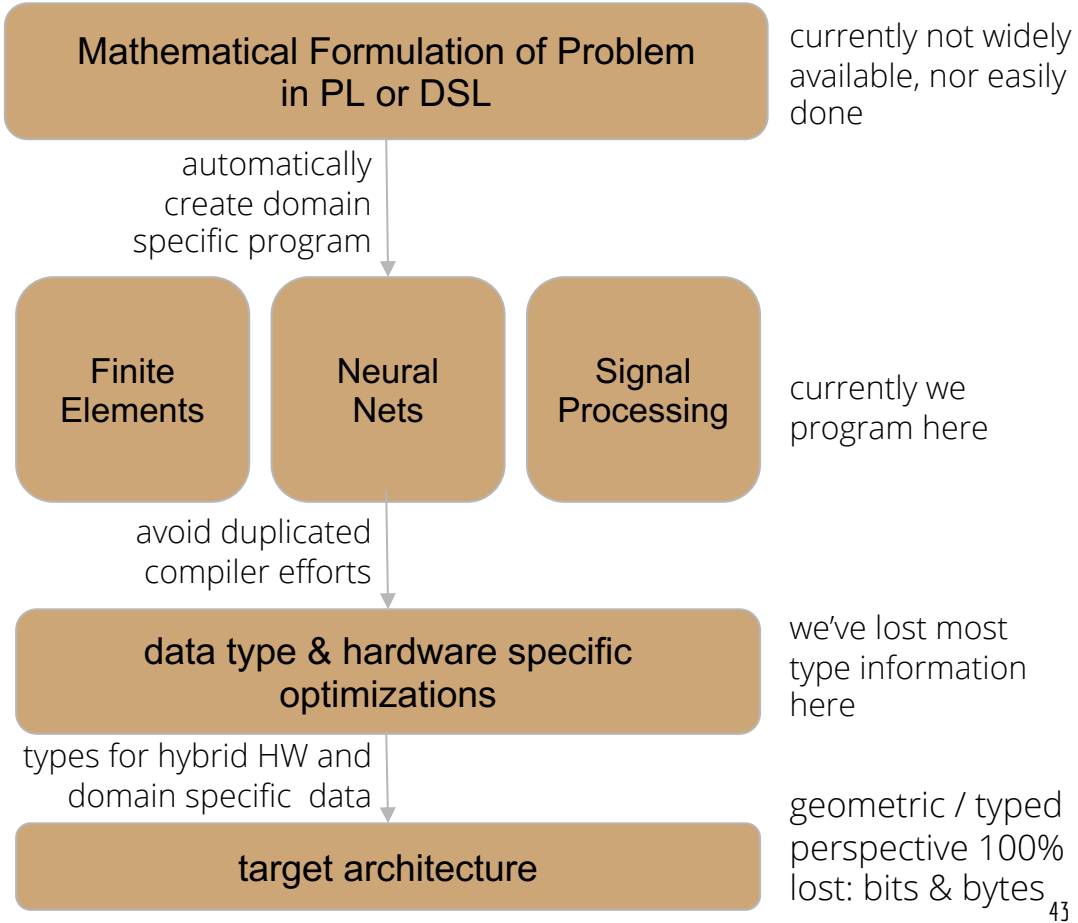
Note: various Haskell libraries, notably Accelerate, have similar expressivity.

Suggested approach

Conal Elliott, Paul Kelly, Oleg Kiselov, and many others.

Challenge 1: CS to design good systems

Challenge 2: drag domain specialists along



Conclusions
Thank you.
Questions

Big changes are under way

Industry driven

Deep CS opportunities