Using TPUs to Design TPUs
Cliff Young, Google AI
AIDArc Keynote
3 June 2018
Why We’re at AIDArc: Can AI help Computer Architecture?

Speech
Vision
Translation
Ranking
Go
Robotics
Self-Driving Cars
Medical Diagnosis
Astronomy
...

?
Outline: Three Codesign Journeys

Two case studies of successful special-purpose computers:

1. Anton machines for Molecular Dynamics (MD): “radical” codesign.
2. TPUs for deep learning: codesign for a moving target.

And a journey we’re just starting:

3. What ML does well, and how to use it in codesign.
Journey 1: Radical Codesign for Molecular Dynamics

Anton, a Special-Purpose Computer for Molecular Dynamics Simulation (ISCA 2007)

1000X speedup machine, delivered 2008.

Hardware: 512 ASICs, 3D torus, 182ns reg-reg latency, 300Gbit/s/node, multicast. Each ASIC had both a “high-throughput” and a “flexible” subsystem. Lots of compute and network, relatively little memory.

Numerics: mostly 32-bit fixed-point.

Algorithms: NT Method, Gaussian-Split Ewald, Constrained Integration, FFT.
Numerical and Algorithmic Codesign

Numerics: float32 ⇒ int32

Neutral Territory Method, $O(n^3) \Rightarrow O(n^{1.5})$ bandwidth

Gaussian Split Ewald, splines ⇒ spherically symmetric interactions

Constrained Integration: enforce water molecule triangles

Fast Fourier Transforms: 4K points on 512 nodes in 4 microseconds
What does Radical Codesign Mean?

“Take the time to understand the space of solutions.”
And the application!
Iteration of Amdahl’s Law: 60%, 30%, 6%, 1%, 0.1% tasks.
Some HW units can make time effectively 0.
[Much easier to do without time-to-market pressures.]

Not just hardware/software, but hardware/software/algorithm/application.
Re-examined approaches at and across all levels we could imagine.
Find the limit implementation first. Then compromise for real system.
Anticipate the future; buy insurance policies for what might happen.
Solved our users’ (chemists’) problem, rather than modify existing methods.

No revolutionary algorithmic change in MD in the last 15 years.

Google
Journey 2: TPUs for Deep Learning

TPUv1 (ISCA 2017)

Charter: Avert a Neural Network inference “success disaster”. Schedule focus: **15 months** from kickoff to deployment.


Numerics: 8-bit fixed-point, with slower 16-bit through software.

Key limitation: memory bandwidth from DRAM. Peak compute only at reuse>1000. Easier problem: Inference **scales out**.
TPUv2

Charter: Do training, so always bigger problems. More general, more flexible. Supports backprop, activation storage.

Hardware: Still systolic arrays for matrix operations. Much better “everything else”: vector, scalar units. Multi-chip parallelism and interconnect.

Numerics: bfloat16 (same exponent size as float32).

High-Bandwidth Memory (HBM) unlocks peak compute.

Compute, Memory, Network, System ⇒ Supercomputer (for ML).
TPUv3 (just announced in May)

Liquid cooling: more heat.

Bigger pods: more scale.

Rapid iteration!

TPUs are real computers now. Built mostly using normal architectural techniques.
Moving-Target Codesign for TPUs

What’s working for us:
- Systolic array matrix multiplication.
- Reduced-precision arithmetic.
- A small number of primitives cover the design space.

Avoiding overfit by buying insurance policies.
- Last-minute support for LSTMs, a sequence-oriented recurrent model.
- Inception (GoogLeNet) broke many of the TPUv1 design assumptions.
- AlphaGo wasn’t imagined; on-device transpose saved 30% latency.
TPUs Haven’t (Yet) Done Radical Codesign

Can we go beyond application-as-given?

Numerical codesign: int8, binary, float16, and bfloat16, what **lower bound**?

Algorithmic codesign:
- parallelization strategy: async v. synchronous SGD; 1-bit parameter updates
- batch size, learning rate, and batch=1
- many kinds of “sparsity”: ReLU, pruning, CSR, block, embeddings, attention,...

Where are the **fundamental limits**?
- tantalizing hints: distillation, intrinsic dimension, feedback alignment

Not nearly enough “methods” research. **Where’s the next 10x?**
Journey 3: Can ML Help Computer Architecture?

- Speech
- Vision
- Translation
- Ranking
- Go
- Robotics
- Self-Driving Cars
- Medical Diagnosis
- Astronomy
- …
The Unreasonable Effectiveness of Deep Learning

2012 AlexNet
2012 Speech Acoustic Modeling
2014 Inception
2016 AlphaGo
2016 Translate
2016 WaveNets
2016 Diabetic Retinopathy
2017 AlphaZero

This feels like a Scientific Revolution, in Kuhn’s sense:

A paradigm shift, not just “normal science”.

Google
Aside: Getting Out of My Comfort Zone

I think that Anton and TPUs work because my colleagues and I are curious.

Application Codesign requires learning lots about the application.
- Get in the heads of the chemists / neural network researchers!
- Teach them Amdahl’s Law; learn what matters in their fields.
- Embed: the random conversations transfer domain knowledge.

Learning to Architect, is an even bigger, scarier step: be a neural network researcher.
- Find great collaborators.
- But still use the new paradigm on our old problems.
Deep Learning: What’s Working Well?

Supervised Learning!

- Needs a **huge** dataset of labeled examples.
  - Image ⇒ cat, sound ⇔ phoneme, English ⇔ French.
- Used with Stochastic Gradient Descent, which requires **differentiable** models.
- Evidence of **syntax** from Translate.

Promising, but not yet widespread: Reinforcement Learning, Evolutionary Strategies.

- Needs a reward/value function.
- How different from existing combinatorial optimization approaches?
- Evidence of **strategy** from game-playing.

How do we use these things?
Some Brief Research Suggestions

1. Apply supervised learning and/or RL on-line, on-device.
2. Use RL and evolutionary algorithms for design-space exploration.
3. Replace heuristics with machine learning systems.
4. Rebuild tools to enable and expose more design-space exploration.
5. Close the timescale gap between microarchitecture and ML.
6. Remove barriers between CPUs and TPUs.
Takeaway: Three Things to Remember

1. Radical codesign is possible, and can give transformative improvements.

2. We’re just starting to do codesign for TPUs. We haven’t yet gotten radical.

3. Deep learning is already a paradigm shift.
   Can we use it to replace the normal science of our field?
   Might it be the general-purpose technique we’ve been looking for?
Links on One Page

Anton: https://dl.acm.org/citation.cfm?id=1250664
TPUv1: https://dl.acm.org/citation.cfm?id=3080246
ML Milestones:
  https://ieeexplore.ieee.org/document/6296526/
  https://www.nature.com/articles/nature16961
  https://arxiv.org/abs/1409.4842
  https://deepmind.com/blog/wavenet-generative-model-raw-audio/
  https://arxiv.org/abs/1712.01815
Caution about RL: https://www.alexirpan.com/2018/02/14/rl-hard.html
Promising ML for Systems and ML for Architecture results
  https://arxiv.org/abs/1706.04972
  https://openreview.net/forum?id=Hkc-TeZ0W
  https://arxiv.org/abs/1712.01208
  https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/
Backup Figures
1. Apply Supervised Learning and RL On-Line, On-Device

At design-time, it’s hard to get lots of labeled examples.

At **run-time**, we have huge streams of correct examples.
   branch direction, next address; anything in a performance counter
   Kraska’s B-tree results as example.

Gather all the hardware signals you can, run an ML prediction system, see if you can improve prediction accuracy, ignoring cost and time to implement.

If there’s an accuracy signal, then look for efficient implementations.
   Distillation helps with efficient implementations.

Aside: it’d be great if the accuracy signal alone were publishable:
   decouple opportunity from implementation.
2. Use Reinforcement Learning and Evolutionary Algorithms for Design-Space Exploration

Caution: **Deep Reinforcement Learning Doesn’t Work Yet.**

Promising results: GPU placement, *hierarchical* GPU placement

AlphaGo, Datacenter Power

Architects have lots of reward functions (speed, power, area, cost, etc.).

Most RL successes resemble a game, with actions, observations, rewards.

Can we phrase design problems as games? Maybe adversarial games?

Looks a lot like design space exploration, but with new tools.
3. Replace **heuristics** with machine learning systems

“Your Heuristic is My Opportunity”

Heuristics are hints, maybe improve performance; don’t break correctness.

Our systems are full of heuristics:

   “Least Recently Used”, “Earliest Deadline First”, ...

We already supplement BTFNT with (Perceptron!) branch predictors.

Hardware predictors are heuristics. ML is good at prediction.

Selectors are just another layer of predictor.

Also, correctness looks hard to learn.

Most random programs / designs are uninteresting or wrong.

Lots of traditional computing won’t tolerate “mostly correct”.

Negative reward doesn’t point toward correct. What’s the gradient?
4. Rebuild tools to enable design-space exploration

Many of our tools find one correct solution, rather than enumerating many or all possible solutions.

Maybe it’s more work. Or an exponential space to enumerate. (don’t be afraid of exponential spaces – Go is a wide fan-out tree.)

But if you had the enumeration, ML techniques could search the space for you.

Compiler example: consider the differences among -O0, -O4, -fdo, and -superoptimize
5. Close the Timescale Gap between μarch and ML

nanosecond/cycle: branch prediction, network routing decision
10s of ns/cycles: cache prefetch, OoO scheduling
...

microsecond: fastest NN inference
millisecond: web-facing NN inference, OS scheduling quantum
seconds: slowest NN inference
minutes: AlphaGo think time (inference + MCTS)
minutes: hero training time
days: typical training time

Build faster and cheaper inference and training, and we can use them in all chips.
6. Remove Barriers between CPUs and TPUs

TPUs are already moving toward more generality. CPUs already have machine-learning resources on them.

Today, they’re separated by PCIe for many good reasons.

Assuming Deep Learning is a paradigm shift, should they stay separate?

Might future CPUs have a generalized learning unit?
- What interfaces would it have? How much state would it hold?
- What latency and throughput would it respond with?
- How widely might it be used? What fraction of area would go to it?