Compiler Construction for Hardware Acceleration: Challenges and Opportunities

ISC 2020 ML Hardware Workshop — June 25, 2020

Albert Cohen, Google, Paris

A New Golden Age for Computer Architecture

John Hennessy and David Patterson's
ISCA 2018 Turing Award Lecture

"We believe the deceleration of performance gains for standard microprocessors, the opportunities in high-level, domain-specific languages and security, the freeing of architects from the chains of proprietary ISAs, and (ironically) the ending of Dennard scaling and Moore's law will lead to another Golden Age for architecture"

A New Golden Age for *Optimizing Compilers*

"We live in a heterogeneous world of domain-specific languages and accelerators, freeing programming language designers and computer architects from the chains of general-purpose, one-size-fits-all designs."

→ A call to action for **compiler construction**

A New Golden Age for *Optimizing Compilers*

“We live in a heterogeneous world of domain-specific languages and accelerators, freeing programming language and computer architects from the chains of general-purpose, one-size-fits-all designs.”

What to expect in the next 25mn

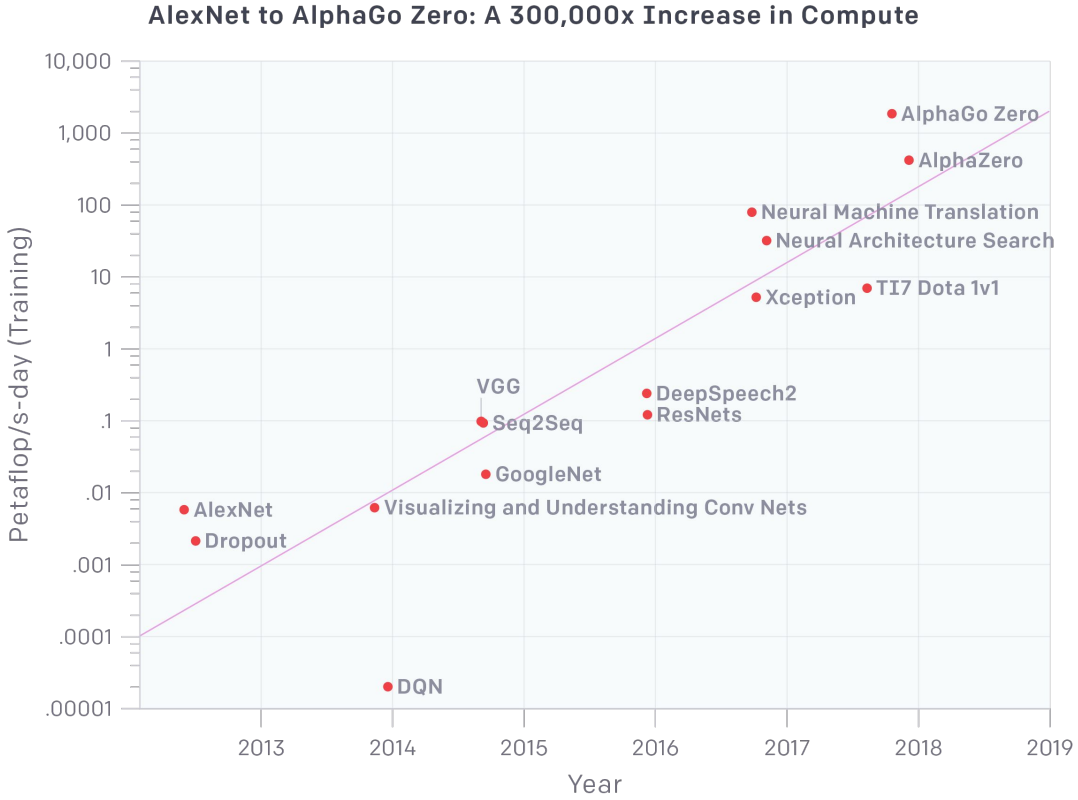
1. Some ML and HPC context
   opportunities for compilers
2. Compiler construction directions and research
   industry perspective, academic perspective

A Detour Through ML Applications

Google AI

Models are growing and getting more complex

- **Model Size:** larger models require more multiply accumulate operations.
- **Model Complexity:** as model complexity increases it becomes harder to fully utilize hardware.
- **Much faster than Moore's law**



Source: OpenAI - AI & Compute

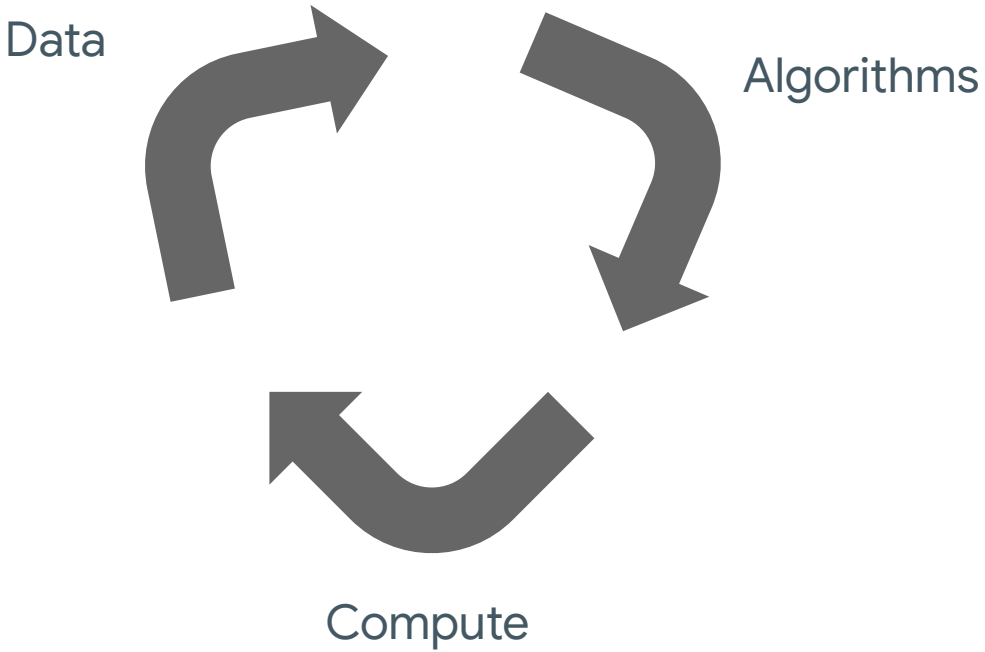
A Detour Through ML Applications

ML is: **data + algorithms + compute**

~ Data drives the continuous improvement cycle for ML models

~ Researchers provide new algorithmic innovations unlocking new techniques and models

~ Compute allows it all to scale as datasets get larger and algorithms need to scale on that accordingly



Cloud and HPC Accelerators

Chip Manufacturers:

Volta, Vega, Ampere

Nervana

Habana

Cerebras Systems

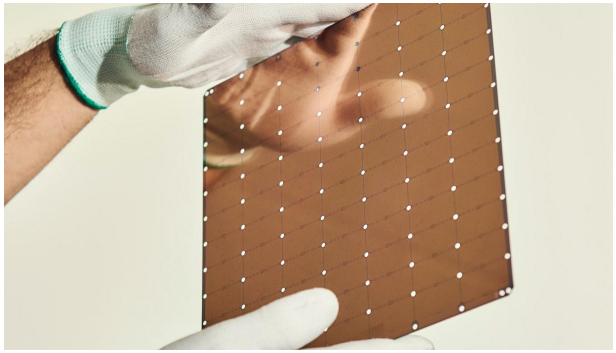
Graphcore

SambaNova

... and many more

Habana

Intel



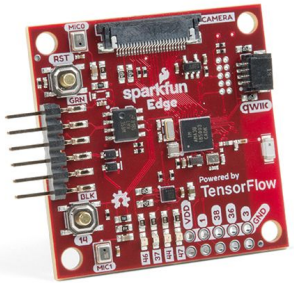
Cerebras Systems

Graphcore

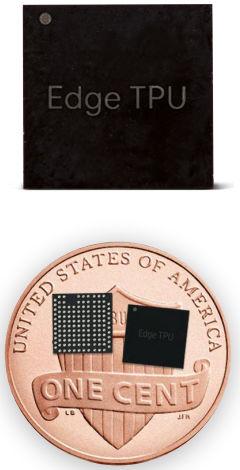
Embedded, Mobile, Edge Hardware

Google AI

~5.5B Mobile Phones



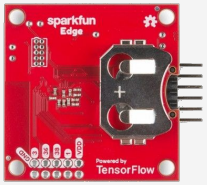
250B+ Microcontrollers



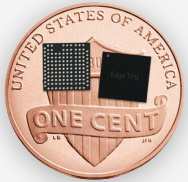
Edge TPUs

CPU GPU DSP NPU

Heterogeneous Compute



Microcontrollers



Edge TPUs

With increasingly complexity

~ Heterogeneous hardware is now the norm

~ Scaling from phones down to microcontrollers

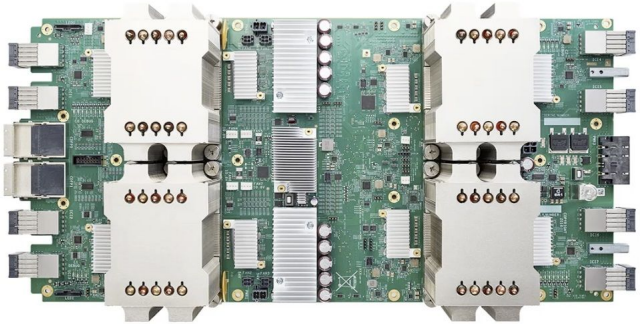
~ Memory, energy, performance and latency constraints become paramount

More Hardware... More Complexity...

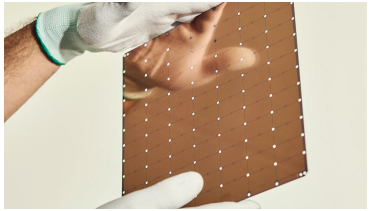
~ Many different hardware accelerators focused on ML

~ Many different types and architectures: 4-bit, 16-bit, 32-bit...

~ Inability to quickly scale up and down hardware consistently and varying levels of abstractions

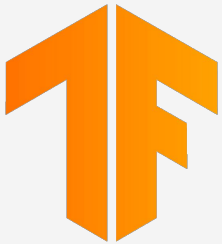


TPU's



Cerebras Systems

Graphcore



PyTorch

CNTK

mxnet

ONNX

HW is not just to blame here

ML Software Explosion too...

~ Many frameworks

~ Many different graph implementations

~ Each framework is trying to gain a usability and performance edge over each other

None of this is scaling

Because

~ Systems don't interoperate

~ Cannot handle all these operators and types consistently on all hardware

~ Poor developer usability and debuggability across hardware

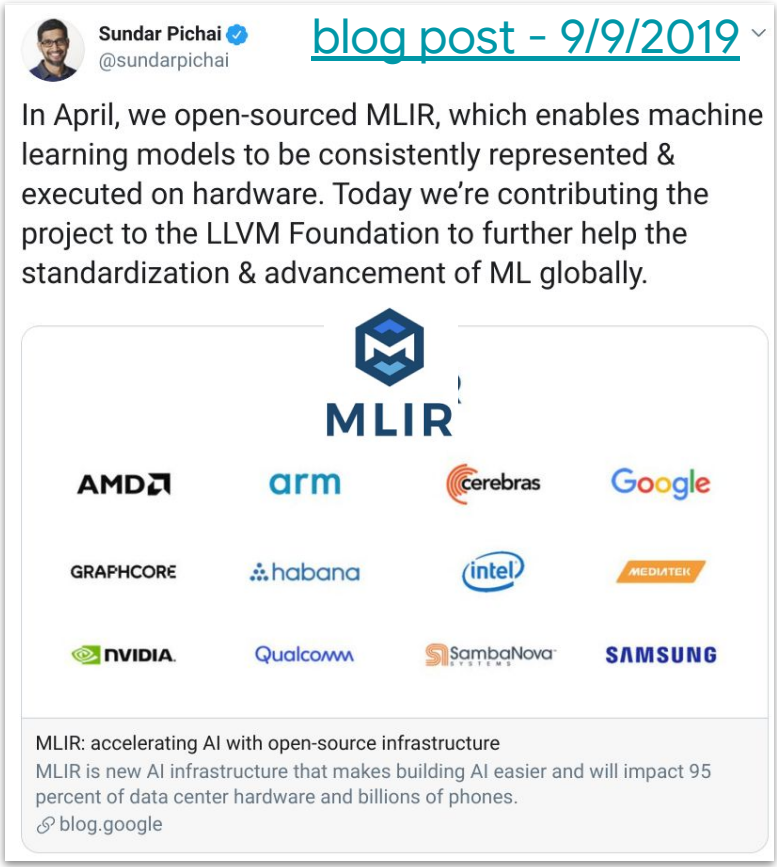
~ No generalizable standard for ensuring software and hardware scales together

Any relief from programming languages?

Investment in a new software infrastructure?

MLIR

MLIR — Multi-Level Intermediate Representation



Rationalizing the **TensorFlow ecosystem**

- **from cloud to on-device AI**
- graph representations
- execution environments
- compilers

And now much more and **growing**

- support **domain-specific frameworks**
- beyond TensorFlow and ML
- contributed to LLVM foundation
- https://mlir.llvm.org

Industry Adoption

95% of the world's data-center accelerator hardware

Deployment on 4 billion mobile phones and countless IoT devices

Governance moved to LLVM
https://mlir.llvm.org

What is MLIR?

~ An extensible representation for types and operations, control & compute

~ Driven by ML training and inference, scaling from **mobile** to **cloud**

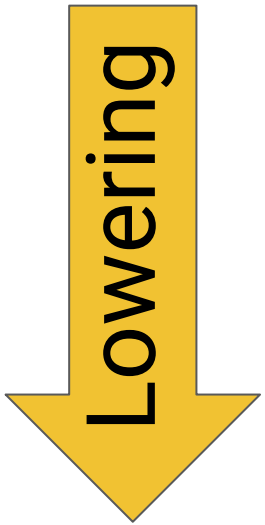
~ Best in class programming models and compiler technology

~ Independent of the execution environment

~ Modular, reusable components

~ Enabling the progressive lowering of higher level abstractions

MLIR — Compute Graphs to Instructions in One Slide



TensorFlow

```
%x = "tf.Conv2d"(%input, %filter)
          {strides: [1,1,2,1], padding: "SAME", dilations: [2,1,1,1]}
     : (tensor<*xf32>, tensor<*xf32>) -> tensor<*xf32>
```

XLA HLO

```
%m = “xla.AllToAll"(%z)
          {split_dimension: 1, concat_dimension: 0, split_count: 2}
     : (memref<300x200x32xf32>) -> memref<600x100x32xf32>
```

LLVM IR

```
%f = llvm.add %a, %b
     : !llvm.float
```

And many more abstractions and levels: TF-Lite, structured linear algebra operations, nested control flow, affine loops, quantized operations, GPU, etc.

Mix and Match in one IR

MLIR — Modeling TensorFlow Control & Concurrency

Control flow and dynamic features of TensorFlow 1, TensorFlow 2

- Conversion from control to data flow
- Both lazy and eager evaluation modes

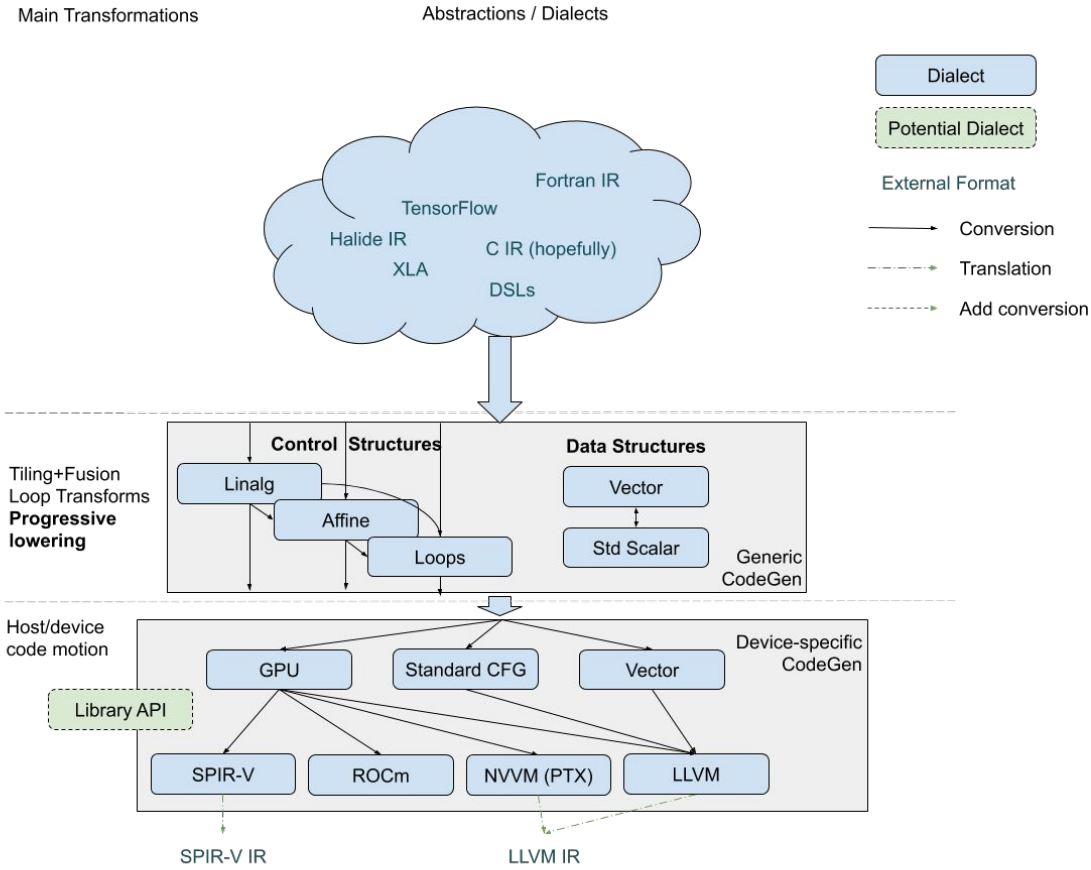
Concurrency

- Sequential execution in blocks
- Distribution
- Offloading
- Concurrency in tf.graph regions
  Implicit **futures** to capture asynchronous task parallelism within SSA and CFG graph representations
- TFRT: https://blog.tensorflow.org/2020/04/tfrt-new-tensorflow-runtime.html

MLIR — GPU Acceleration

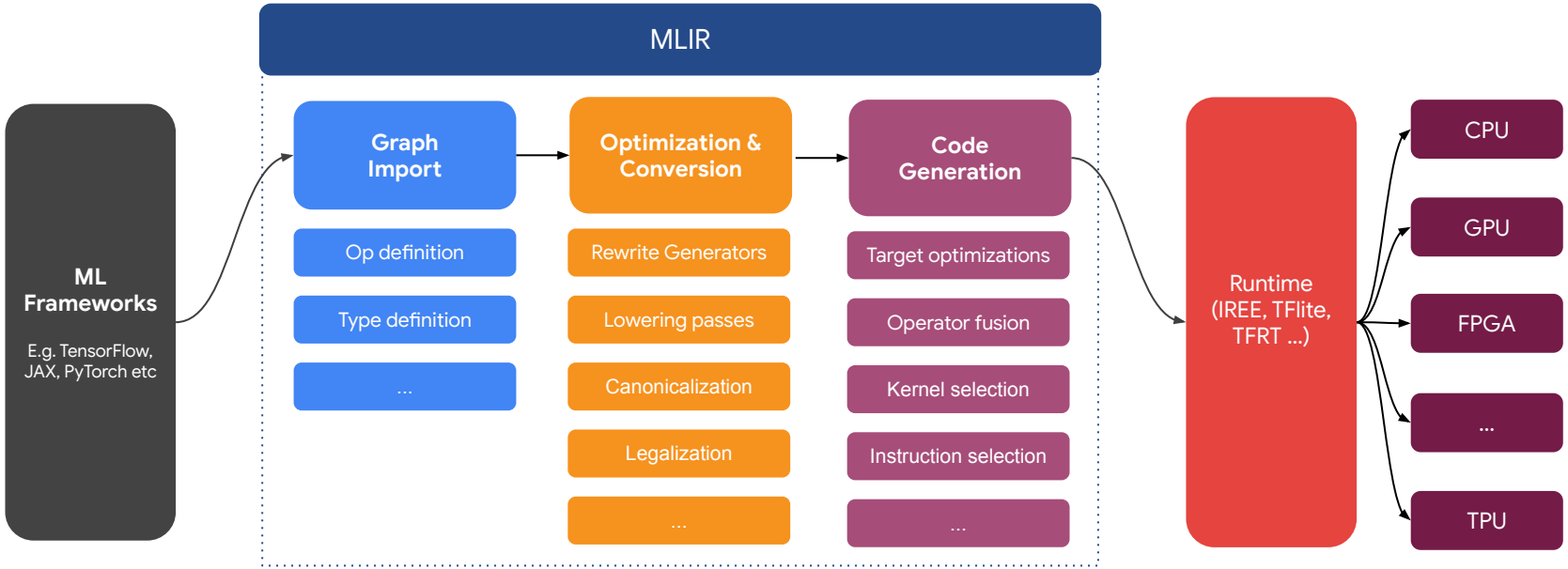
MLIR Open Design Meeting
December 12, 2019

And many more dialects, projects



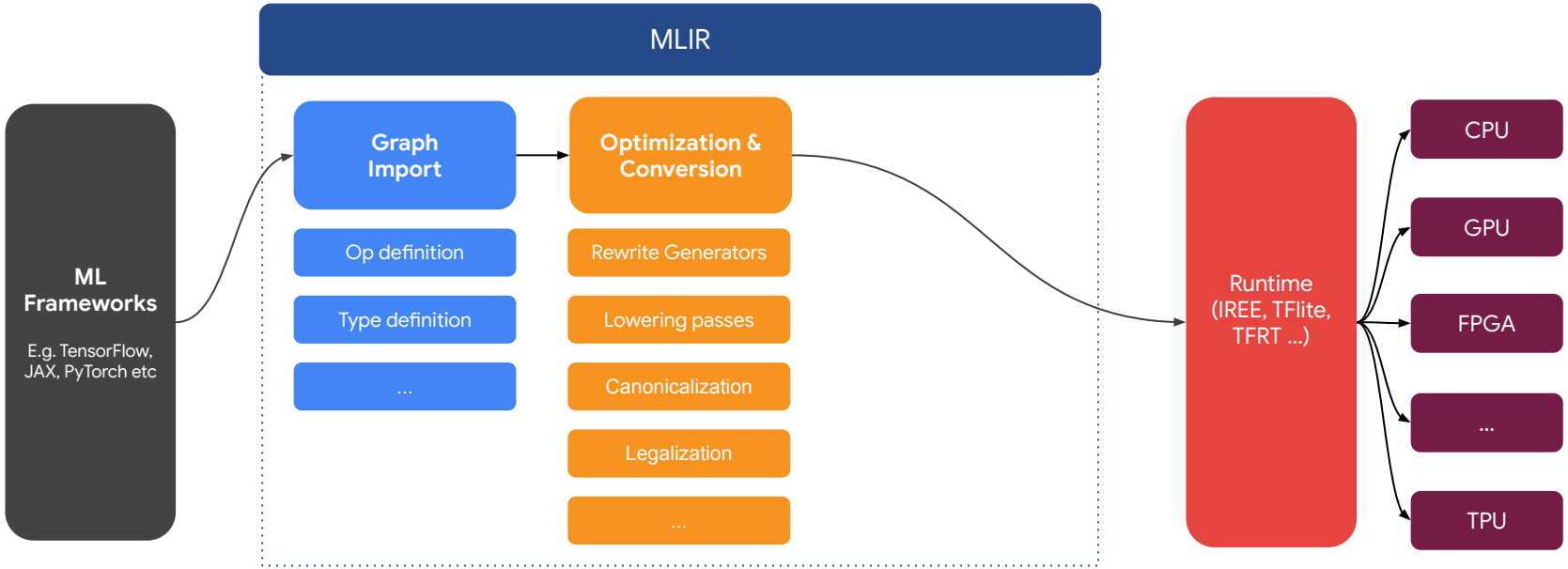
MLIR Compiler Infrastructure

A common graph representation and legalization framework,
a common set of optimization and conversion passes and code generation pipeline



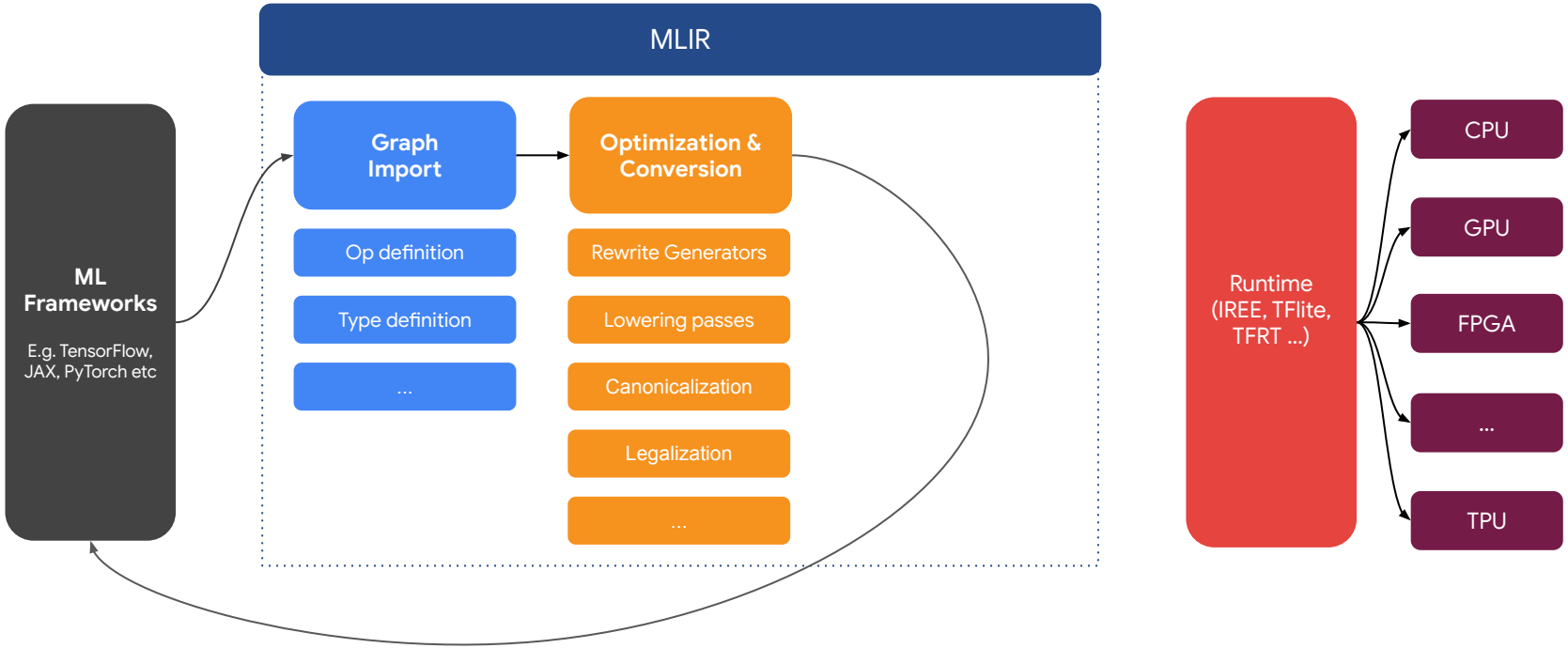
It's Unopinionated

Leverage different components of the system as needed



One Size Fits None

MLIR can also be modularized as a graph rewriting tool, e.g. for TensorFlow Lite



Zooming in:
MLIR-based research

Compile to Learn

High-performance ML layers, generated automatically
Compilation algorithms tailored for tensor computing

Learn to Compile

Automatic synthesis of code optimization
Heuristics, performance auto-tuning

Problem Statement: Synthesizing Fast ML Operations

Context: “superoptimizing” loop nests in numerical kernels, finding best implementation/optimization decisions

- Optimizations do not compose well, they may enable or disable others
- Cannot infer precise performance estimates from intermediate compilation steps

Optimizing compilation never seems to catch up

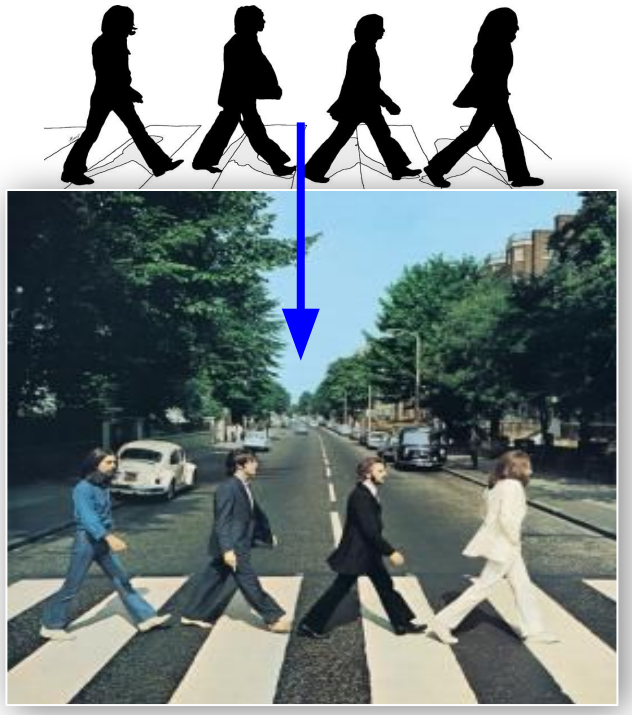
... new hardware, optimization tricks

... witnessing a widening gap in performance portability

Synthesizing Fast ML Operations

Synthesizing Fast ML Operations

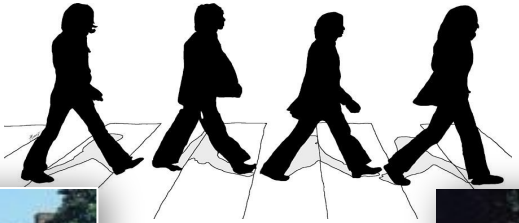
algorithmic specification



hardware-accelerated implementation

Synthesizing Fast ML Operations

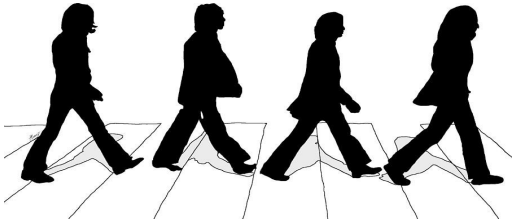
algorithmic specification



many possible implementations

Synthesizing Fast ML Operations

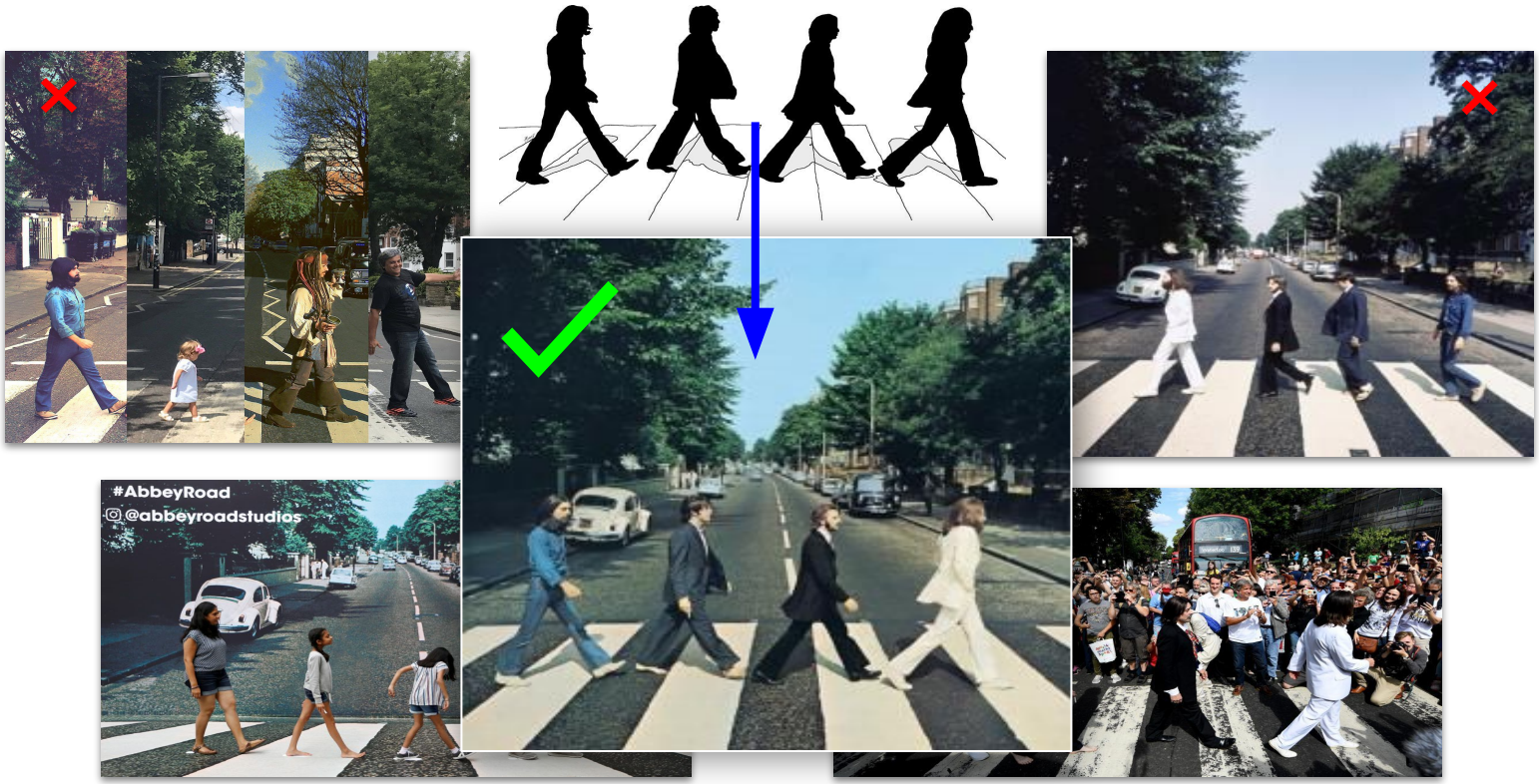
algorithmic specification



way too many **possible implementations**

Synthesizing Fast ML Operations

→ Compiler + Constraint Solver + Reinforcement Learning



Candidates

Partially instantiated vector of decisions

- Every choice is a decision variable
- Taking a decision = restricting a domain
- Fully specified implementation ⇔ All decision variables assigned a single value

Candidates and Constraints

Kernel

```
%x = add [0, 16) {
%y = add [0, 10){
for %d0 in [0, 4) {
    %z = add %y, %d0
}
```

Decisions

```
order(%x, %d0) ∈ { Before, Inner } <- decision
order(%x, %y)  ∈ { Before }
order(%y, %d0) ∈ { Before, Inner } <- constraint propagation
...
```

Enforce decision coherence with constraints

```
order(x, d0) = Inner && order(x, y) = Before => order(y, d0) ∈ { Inner, After }
```

Enabling Better Search Algorithms

Well behaved set of actions

- Commute
- All decisions known upfront
- Constraint propagation almost never backtracks in practice

Flat, fixed sized, ideal environment for Reinforcement Learning (RL)

- Extract features from the decision vector
- Global heuristics, aware of all potential optimizations
- Infer all possible decisions (actions) and/or estimate performance

Constraint Satisfaction Problem (CSP)

Find an assignment for functions

```
kind: Dimension -> { Loop, Unrolled, Vector, Thread, Block }

order: Statements x Statements -> { Before, After, Inner, Outer, Fused }
```

Satisfying a system of constraints

```
∀ a, b ∈ Dimension. order(a, b) = Fused => kind(a) = kind(b)
```

(a.k.a. typed fusion)

Synthesizing GPU Optimizations

Generic loop nest and array optimizations + GPU-specific optimizations

- Strip mining factor
- Loop interchange
- Loop fusion
- Software pipelining
- Statement Scheduling
- Rematerialization
- Memory layout
- Copy to local memories
- Double buffering
- Vectorization

Branch and Bound + Monte Carlo Tree Search (MCTS)

Performance model of a lower bound on the execution time

$$\forall x \in S.\ \mathrm{Model}(S) \leq \mathrm{Time}(x)$$

- Enable Branch & Bound, with feedback from real executions
  - Reduce the search space by several orders of magnitude
  - Prune early in the search tree (75% in the first two levels for matmul on GPU)
- Possible because it is aware of choices that are yet to come
- GPU model of block- and thread-level micro-architecture
  - Roofline model of the interaction between bottlenecks

Match our outperform state of the art code generators

Halide, TVM, Lift, Triton, etc.

Search Issues (Ongoing Research)

- **High variance of the search time (stuck in suboptimal areas)**
- **Lots of dead-ends**
  - Mostly due to performance model
  - ~20x more dead-ends than implementations
- **Non-stationary distribution due to cuts**
  - Somewhat intrinsic to MCTS
  - Branch & bound strategy makes it trickier

MLIR

Call to Action: *Extensibility* & *Hackability* & *Research*

Heterogeneity ⇒ need for a **super-extensible = super-reusable** system

foster next-generation accelerator adoption and research

- domain-specific languages as first-class constructs
- domain-specific hardware interface as first-class operations
- lowering and mixing language and hardware abstractions
- type systems: novel numerics, sparse tensors, logic properties, dependent types
- concurrency, parallel constructs, memory modeling
- model and carry debug information, traceability, security properties
- model structured search spaces of program transformations

Compiler Construction Design for Diversity

We are hiring!
mlir-hiring@google.com