Google

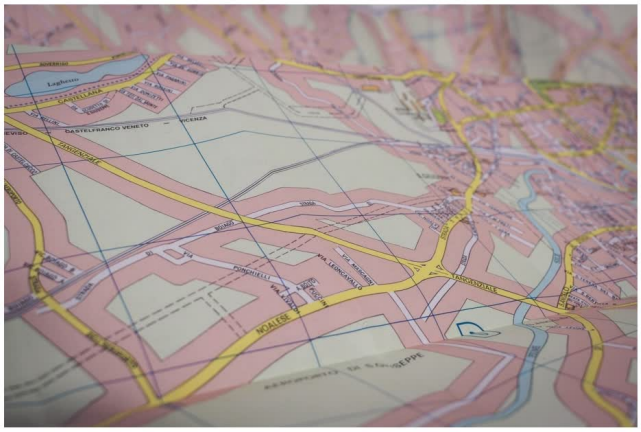
MLIR: ML infra at Google

MLIR Community Meetup China 2022

Jacques Pienaar
(presenting the work of many)
jpienaar@google.com

Overview

MLIR



- Intros (me & MLIR & community)
- Roadmap

Google

A collection of modular and reusable software components that enables the progressive lowering of high level operations, to efficiently target hardware in a common way

What is MLIR?

A toolkit for representing and transforming “code” ⟲⇄⇓

Represent Multiple Levels of

- tree-based IRs (ASTs),
- graph-based IRs (TF Graph, HLO),
- machine instructions (LLVM IR)

IR at the same time

Batteries included

Common compiler infrastructure

- location tracking
- richer type system
- offline reproducers
- test case reducers
- common set of passes (analysis/optimization)
- codegen components and “ready-to-use” abstractions

And much more!

Google

Open, under framework neutral governance (without CLA impediments)

Origin

Infrastructure for building reusable and extensible domain-specific compilers

ML frameworks contain multiple internal representations and perform numerous compiler-style transformations for optimizations and representation conversions

Authoring is slow, painful	Developers reimplement continuously Lack of interoperability only exacerbates this
ML systems need to adapt to fast-paced industry	ML models are getting bigger and more complex, needs are dramatically changing
Need to build standardized infrastructure	Standardizes of basic concepts to enable reuse, composability. Built to supports full customizability and extensibility

Google

Origin

Infrastructure

ML fram
compiler

umerous
nversions

Authoring is slo

ML systems nee
fast-paced indus

eeds are

Need to build sta
infrastructure

mposability. Built

Google

Much more info!

Whole tutorial (mlir.llvm.org)

Talk at CGO 2021 about design/concepts, TF dev summits, open design meetings (recorded videos)

MLIR Community

Google

Community in nutshell

- Open community of collaborators across the world working together to improve the state of art domain specific compilation
- Academia
  - Multiple research collaborations with universities across the world
  - National laboratories in US & Europe
  - +83 citations in 2021
- Industry
  - Usage at multiple companies including AMD, Apple, AMD, Cerebras, Cruise, Deepmind, Google, IBM, Intel, Microsoft, NVIDIA, Samba Nova, Xilinx, ...
  - Contributions from >50 domains last year
- Open designs, open meetings, contributions under LLVM community guidelines

Google

MLIR in scientific community

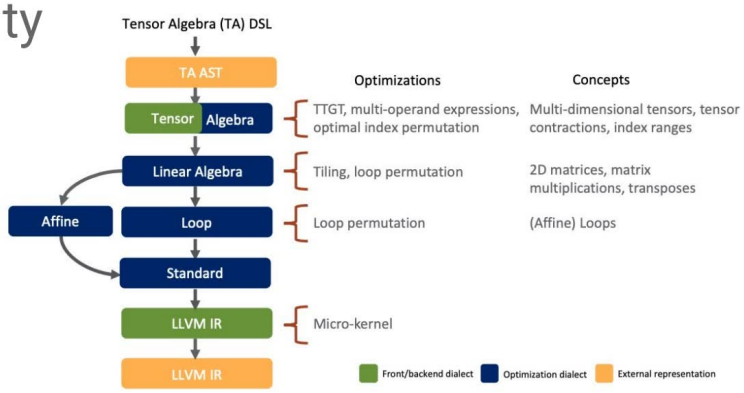


Fig. 1: COMET execution flow and compilation pipeline

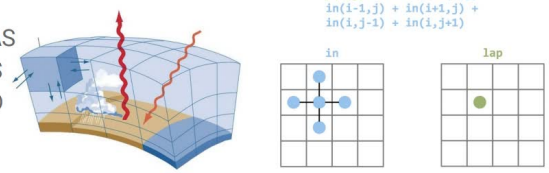
Accelerating climate modelling

- solve PDE
- finite differences
- structured grid

- element-wise comput
- fixed neighborhood

```
lap(i,j) = -4.0 * in(i,j) +
  in(i-1,j) + in(i+1,j) +
  in(i,j-1) + in(i,j+1)
```

in lap



JEAN-MICHEL GORIUS, TOBIAS WICKY, TOBIAS GROSSER, AND TOBIAS GYSI

Google

High-Performance Computational Chemistry

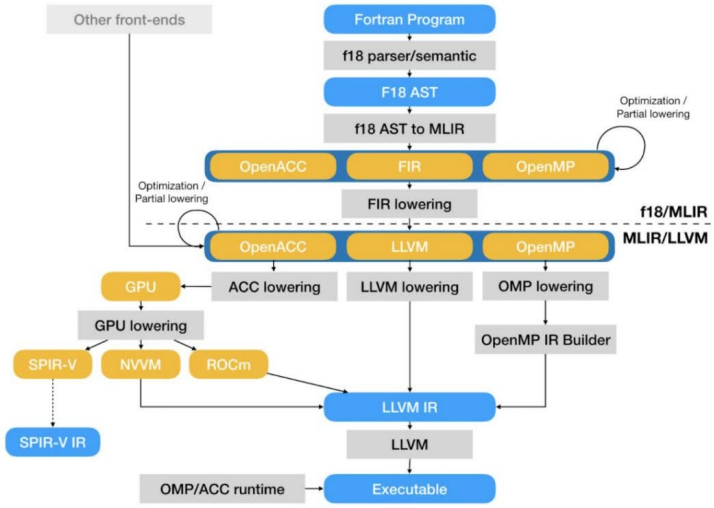
LCPC 2020

Keynote: Preparing for Extreme Heterogeneity in High Performance Computing
Jeffrey S Vetter, Group Leader - Future Technologies Group ORNL

Leveraging LLVM Ecosystem to Meet a Critical ECP (community) need : FORTRAN

ECP Projects: Flang, SOLLVE, PROTEAS-TUNE
Many other contributors: NNSA, NVIDIA, ARM, Google, …

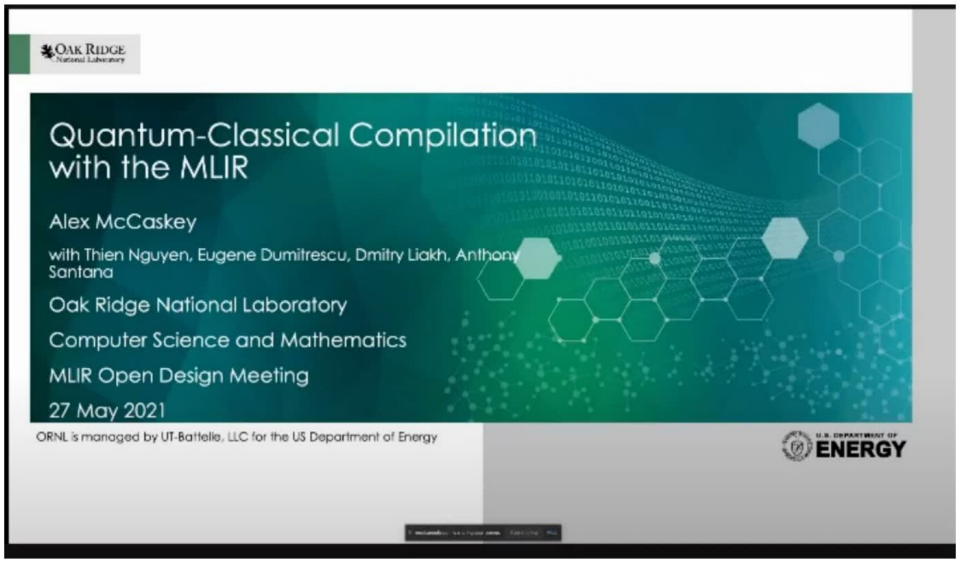
- Fortran support continues to be an ongoing requirement
- Flang project started in NNSA funding NVIDIA/PGI to open source compiler front-end into LLVM ecosystem
- SOLLVE is improving OpenMP dialect, implementation, and core optimizations
- PROTEAS-TUNE is creating OpenACC dialect and improving MLIR
- ECP projects are contributing many changes upstream to LLVM core, MLIR, etc
- Many others are contributing: backends for processors, optimizations in toolchain, …
  - Google contributed MLIR



Go EXASCALE COMPUTING PROJECT

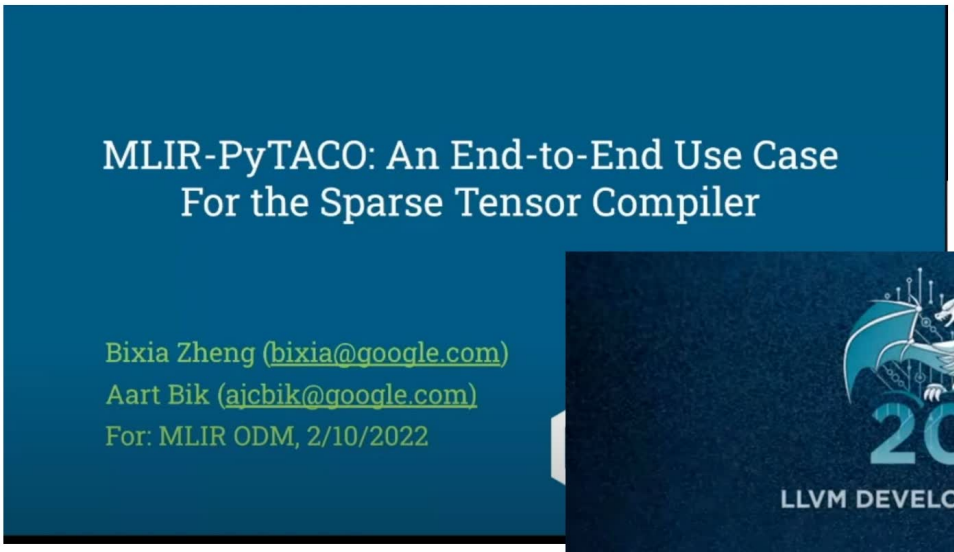
2

Quantum compilation



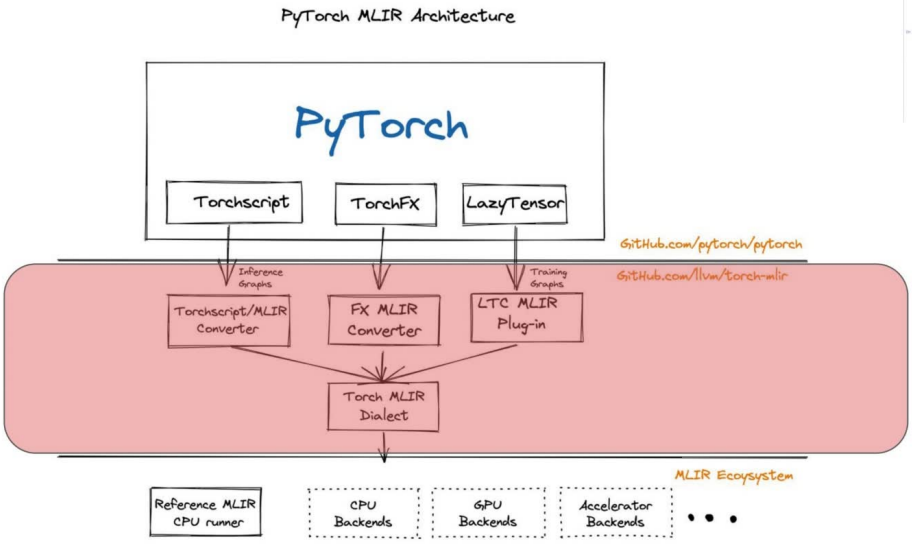
Google

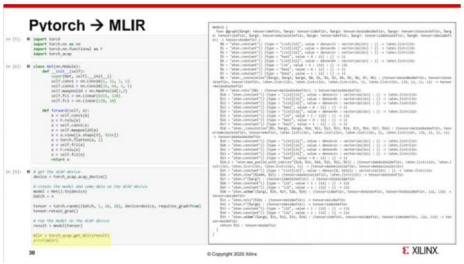
Sparse codegen



Google

PyTorch MLIR integration





Samuel Bayliss, Xilinx: "Compiling for Xilinx AI Engine using MLIR", C4ML 2020

Google

Most viewed post on PyTorch forum excluding bug reports (CUDA) and debugging support in last **year**

Example: CIRCT Project

Apply MLIR and the LLVM development methodology to the domain of hardware design tools

https://github.com/llvm/circt

Circuit IR Compilers and Tools: MLIR for Hardware Design

Amalee Wilson, Stanford University, amalee@cs.stanford.edu
Stephen Neuendorffer, Xilinx, stephenn@xilinx.com
Chris Lattner, SiFive, clattner@sifive.com
Other Collaborators
Microsoft, PNNL, ETH, EPFL
U. Edinburgh, Cornell, Illinois

Introduction

- **Designing and programming complex, heterogeneous systems-on-chip mixing general purpose and specialized components is difficult.** The EDA industry has well-known and widely used proprietary and open source tools. However, these tools are often inconsistent, have usability concerns, and were not designed together into a common platform.
- **The CIRCT project is a new effort to apply MLIR and the LLVM development methodology to the domain of hardware design tools.** A coherently designed set of abstractions leveraging best practices in compiler infrastructure and compiler design techniques will result in a new generation of tools for both designing and programming complex, heterogeneous systems-on-chip mixing general purpose and specialized components.

Aim

Create a ***comprehensive open-source infrastructure*** capable of representing ***multiple levels of abstraction*** to improve both hardware and software.

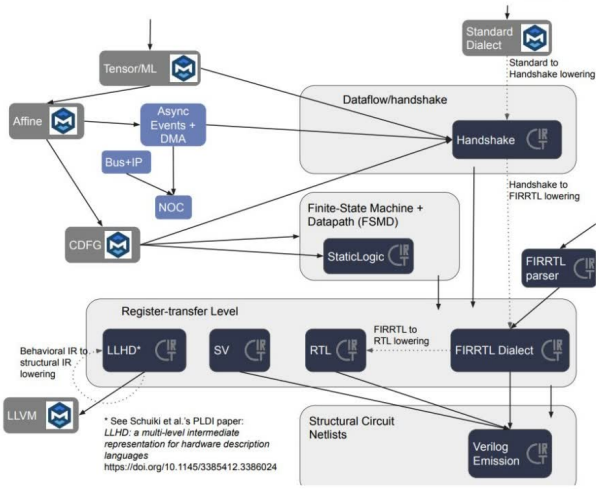
Hardware
- Accelerator Design
- Circuit Synthesis
- Memory Hierarchy
- Interconnect Design
- System Simulation

Software
- Accelerator Programming
- Code Generation
- Memory Allocation
- Host Code Partitioning
- JIT Execution

Next generation open source synthesis infrastructure
- LLVM incubator project

Focus on RTL level and above
- Interfaces with SystemVerilog, Chisel, C++
- FPGA+ASIC targets
- Integrated simulation through LLVM backends

Leverage unique MLIR Capabilities
- **Parallel Compilation => Reduced design time**
- **Multiple Abstractions/Dialect => Improved predictability**
- **Unified Framework => Better integration** between high level and low level tools
- **Cyclic SSA graphs** (contributed by CIRCT developers) **=> hardware-oriented semantic**



Google

Poster from LLVM Dev Meeting'20 https://llvm.org/devmtg/2020-09/

Tensor Operator Set Architecture (TOSA) Dialect

- Community contributed (ARM) ML Abstraction for deployment
- TOSA provides a standardized set of tensor level primitives
  - TOSA guarantees consistent operation when deploying networks
  - Stable layer of composable operators for software and hardware design
  - Specification defines the functional and numerical precision of operators
  - Profiles provide capability sets from microcontrollers to large systems
  - Reference implementation and tests to verify compatibility.
- TOSA in MLIR – a mid level dialect
  - The specification is not part of LLVM/MLIR, but there is an implementation of the spec.
  - TOSA dialect is published in the MLIR repository
  - TensorFlow and TensorFlow Lite legalizations to TOSA dialect live in the TensorFlow repository, and PyTorch legalizations in torch-mlir

Google

MLIR Compiler Ecosystem @ Google

More independent abstractions enables reuse

Google

MLIR usage in ML stacks

- Multiple layers, can roughly divide into
  - frameworks/frontends
  - deployments/backends
  - infrastructure
- MLIR used at these levels by various projects
  - I've been member of/contributor to 7 of these teams, but can't speak for all :)
  - Sharing at multiple levels

Google

* onnx-mlir purely community project

Frontends

- JAX produces MLIR (MHLO) by default
  - Since Jan 27, 2022 JAX's default output is MHLO
  - Able to produce non-MHLO MLIR too (used in active experimentation)
- TensorFlow
  - All TensorFlow execution currently running through TensorFlow Graph dialect
  - Integration into TF up to Python level for constructing & transforming functions (target Q2)
  - Offline deployment tool for model optimization
- TensorFlow to XLA bridge
  - Default format for TF/XLA is MHLO for targeting XLA TPU already
  - CPU/GPU bridge migration & unification during 2022
    - Already used in KernelGen & Autofusion in TF (e.g., every TF add already uses it)
- PyTorch
  - torch-mlir targeting complete coverage of core ops in 2022
    - Lowering to TOSA and LinAlg for execution/codegen [active, open community]
  - PyTorch-XLA is product and partner team with different focus and will adopt/integrate where & when it makes sense

Google

Deployments

- TFLite
  - Default converter & optimization format since TF 2.2 (May 2020) uses MLIR & TFL dialect
    - Reused parts of TFLite converter and TF/XLA bridge for partial JAX support
  - Dynamic range quantization using MLIR part of upcoming TF 2.9 release
- XLA CPU/GPU
  - XLA CPU rebased on top of MLIR/existing Autofusion work completely during 2022
  - XLA GPU backend migrates to MLIR + TFRT in 2022
- IREE
  - Very active, open community project (good to invite for roadmap talk!)
  - See blog posts by Nod.ai post on using & tuning IREE "SHARK: The fastest PyTorch runtime – 3x over Torchscript, 1.6x over TF/XLA, 23% faster than ONNXRuntime"

Google

Core MLIR

Google

Declarative pattern rewrites

- Significantly reduces boilerplate
- Enables compiling to efficient fused automata
  - Underlying PDL strata: new class of dynamic & distributable rewrites
- Notation follows industry standard S-expr:

```
# For any float comparison operation, "cmp", if you have "a == a && a cmp b
# then the "a == a" is redundant because it's equivalent to "a is not NaN"
# and
for o
  op
  ]
```

```
(plus (mult (SIN @0) (SIN @0))
      (mult (C
```

```
(flag_unsa
 { build_one_c
```

```
// Prefer RORX which is non-destructive and doesn't update EFLAGS.
let AddedComplexity = 10 in {
  def : Pat<(rotr GR32:
            (RORX32ri
  def : Pat<(rotr GR64:
            (RORX64ri
```

```
// addi(subi(x, c0), c1) -> addi(x, c1 - c0)
def AddISubConstantRHS :
    Pat<(Arith_AddIOp:$res
          (Arith_SubIOp $x, (Arith_ConstantOp APIntAttr:$c0)),
          (Arith_ConstantOp APIntAttr:$c1)),
        (Arith_AddIOp $x, (Arith_ConstantOp (SubIntAttrs $res, $c
```

- But that's not that user friendly

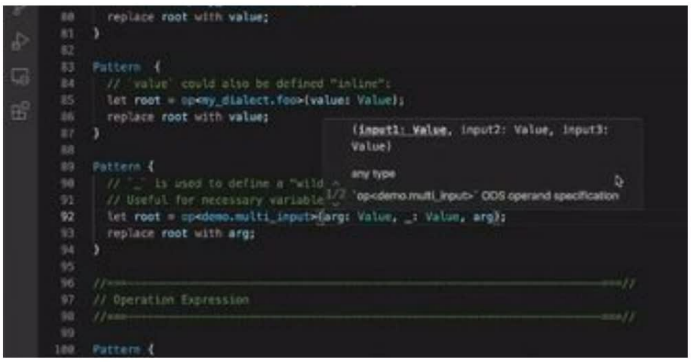
Google

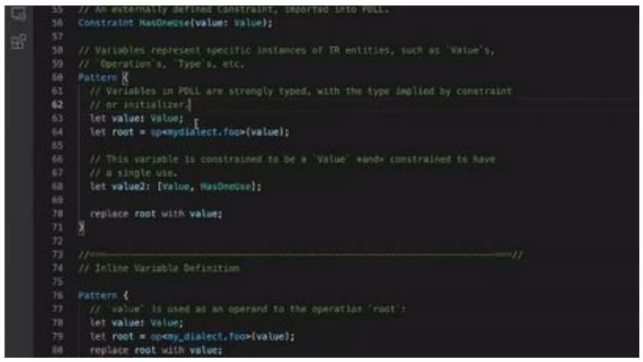
Declarative rewrite pattern language

- Developed a new, dedicated rewrite language

  [YouTube video from open design meeting]

  - Integrates with op definitions
  - LSP support including autocomplete, error reporting [integration with vscode not rolled out]
- General rollout Q3





Google

Debugging tooling UX

- Reproducers
  - General reproducers automatically generated
- Reducer
  - Automatic failure case reduction tool
  - Current interface too developer focussed
  - Difficult for end-to-end testing due to lack of composable components
- Debug actions employed
  - Bisection tooling
  - Interactive stepping

(demo ware)

Google

Supported Language Bindings

- Enable development and research across multiple domains
  - Need driven and contributions welcome
- Python
  - Enabler to JAX emitting MLIR by default
  - Kernel codegen exploration/tie in to ML autotuners
- Haskell
  - Primary target is Dex (dependently typed programming language, used as custom kernel language for ML) able to utilize higher level/reusable constructs
- Support community in building others
  - C API as common integration
- Wishlist/pure community: Swift & Rust

Google

Conclusion

Google

Conclusion

- MLIR is OSS project with many internal and external collaborators
  - Some of biggest challenges is connecting everyone
  - Discovering connections next door sometimes as difficult as internationally!
- Very excited to see venue for adjacent collaborators
  - Help cross time zone divide (support, learning, collaboration)
  - "Sun never setting on development"

Google

Get involved!

Visit us at mlir.dev & tensorflow.org

- Code, documentation, tutorial
- Developer forum/mailing list
  LLVM Discourse server
  mlir@tensorflow.org
- Open design meetings
- Contributions welcome!

Google