The Golden Age of Compilers

in an era of Hardware/Software co-design

张献伟

xianweiz.github.io

DCS290, 06/24/2021
Compilation Phases

- **Lexical**: source code → tokens
  - RE, NFA, DFA, ...

- **Syntax**: tokens → AST or parse tree
  - CFG, LL(1), LALR(1), ...

- **Semantic**: AST → AST + symbol table
  - SDD, SDT, typing, scoping, ...

- **Int. Code Generation**: AST → TAC
  - IR, offset, CodeGen, ...

- **Optimization**: TAC → (optimized) TAC
  - BB, CFG, DAG, ...

- **Code generation**: TAC → Instructions
  - Instruction, register, stack, ...
Modern Compilers [现代编译器]

• Compilation flow [编译流程]
  – First, translate the source program to some form of intermediate representation (IR, 中间表示)
  – Then convert from there into machine code

• IR provides advantages [IR的优势]
  – Increased abstraction, cleaner separation, and retargeting, etc
Opportunities for Compiler?

Compiler???
Outline

• The Trends [需求趋势]
  – Application, software, hardware

• The Issues [传统编译的问题所在]
  – Limitations of classical compilers

• The Solutions [潜在编译技术方案]
  – TVM, MLIR

• Summary
ML Applications [机器学习应用]

• 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

[1] IR Design for Heterogeneity: Challenges and Opportunities
ML Software Explosion

• Many frameworks

• Many different graph implementations

• Each framework is trying to gain a usability and performance edge over each other
High Performance Computing

• Larger scale applications
  – Climate change, new drug discovery
  – Data analytics, modeling and simulation

• Various parallel programming models
  – MPI, OpenMP, OpenACC, SYCL/DPC++
More Hardware... More Complexity...

• Lots of players! (an incomplete list!)

Programmable xPUs

Custom Hardware

Specialization
Architecture Trends

Architecture 2030 Workshop @ ISCA 2016

John L. Hennessy, David A. Patterson

• Current challenges
  – End of Moore's Law and Dennard Scaling
  – Overlooked security

• Future opportunities in computer architecture
  – Domain-specific architectures
  – Domain-specific languages
  – Open architectures
  – Agile hardware development

Domain Specific Arch. [领域专用架构]

• Achieve higher efficiency by tailoring the architecture to characteristics of the domain

  – Not one application, but a domain of applications
    □ Different from strict ASIC
  – Requires more domain-specific knowledge than general purpose processors need

• Examples:
  – Neural network processors for machine learning
  – GPUs for graphics, virtual reality
  – Programmable network switches and interfaces
Domain Specific Arch. (cont.)

- More effective parallelism for a specific domain
  - SIMD vs. MIMD
  - VLIW vs. Speculative, out-of-order
- More effective use of memory bandwidth
  - User controlled versus caches
- Eliminate unneeded accuracy
  - IEEE replaced by lower precision FP
  - 32-64 bit bit integers to 8-16 bit integers
- Domain specific programming language
  - DSAs require targeting of high level operations to the architecture
  - Need matrix, vector, or sparse matrix operations
Domain Specific Languages

• Invent a new language for the specialized HW
  – Better exploit application knowledge: directly connecting users to HW, bypassing the ISA

• OpenCL is such an example
  – HW vendors can regularly change GPU ISAs without affecting user code
    □ OpenCL is the new contract

• It can be a bitter experience for programmers
  – Rewrite code for new HW
  – Not all new languages will survive in the long term
Outline

• The Trends [需求趋势]
  – Application, software, domain specific

• The Issues [问题所在]
  – Limitations of classical compilers

• The Solutions [潜在编译技术方案]
  – TVM, MLIR

• Summary
The History

• For more than 50 years, we have enjoyed exponentially increasing compute power.

• The growth is based on a fundamental contract between HW and SW.
  – HW may change radically “under the hood”
    □ Old SW can still on new HW (even faster)
  – HW looks the same to SW, always speaking the same language
    □ The ISA, allows the decoupling of SW development from HW dev

• Three-phase compiler design (e.g., LLVM)
  – One frontend for many backends, one backend for many frontends
The Issue [当前面临的问题]

• The contract is breaking
  – The end of Moore’s Law forces new design approaches
    □ Develop specialized HW to gain massive performance
    □ Program and use the specialized and heterogeneous HWS

• Limitations of LLVM
  – “One size fits all” quickly turns into “one size fits none”
    □ “fits all”: a single abstraction level to interface with the system
  – LLVM is: 👍 CPUs, “just ok” 👈 for SIMT, but 👎 for many accelerators
  – … is not great for parallel programming models 💩

• Many problems are better modeled at a higher- or lower-level abstraction
  – e.g. source-level analysis of C++ code is very difficult on LLVM IR
Issue: Modern Languages

• Modern languages pervasively invest in high level IRs
  − To solve domain-specific problems, like language/library-specific optimizations, flow-sensitive type checking (e.g., for linear types)
  − To improve the implementation of the lowering process

• Each compiler frontend is creating one or more high level IR in addition to their AST representations
Issue: ML Frameworks

- Compiler tech is widely deployed in others fields, including machine learning frameworks
- ML systems typically use “ML graphs” as a domain-specific abstraction
- TensorFlow is basically a huge compiler ecosystem
  - These boxes are all different domain-specific compiler systems:
    - Different limitations, challenges, owners, etc
    - No unifying theory and infrastructure to support this
Next-Gen Compilers & PL are Needed

• We need:
  – Hardware abstraction spanning diverse accelerators
  – Support for heterogeneous compute platforms
  – Domain specific languages and programming models
  – Quality, reliability, and scalability of infrastructure

• We see:
  – “No one size fits all” compiler
  – Shape of the problem is the same, but the accel details always vary

• This opportunity is beckoning a golden age in compiler and PL technology!
Outline

• The Trends [需求趋势]
  – Application, software, domain specific

• The Issues [问题所在]
  – Limitations of classical compilers

• The Solutions [潜在编译技术方案]
  – TVM, MLIR

• Summary
• Bring ML to a wide diversity of hardware devices
  – Current frameworks rely on vendor-specific operator libraries and optimize for a narrow range of server-class GPUs
  – Deploying workloads to new platforms – such as mobile phones, embedded devices, and accelerators (e.g., FPGAs, ASICs) – requires significant manual effort

• TVM: an end to end ML compiler framework for CPUs, GPUs and accelerators
  – Aims to enable machine learning engineers to optimize and run computations efficiently on any hardware backend

[1] TVM: An Automated End-to-End Optimizing Compiler for Deep Learning, OSDI’2018
[2] Apache TVM,
TVM (cont.)

- Execution steps in TVM
  - ① First takes as input a model from an existing framework and transforms it into a computational graph representation
  - ② Then performs high-level dataflow rewriting to generate an optimized graph
  - ③ The operator-level optimization module must generate efficient code for each fused operator in this graph
TVM (cont.)

• Execution steps in TVM
  - ④ TVM identifies a collection of possible code optimizations for a given hardware target’s operators
    - Possible optimizations form a large space, so we use an ML-based cost model to find optimized operators
  - ⑤ Finally, the system packs the generated code into a deployable module
MLIR: Multi-Level Intermediate Representation

• MLIR: Compiler Infra at the End of Moore’s Law
  – Joined LLVM, follows open library-based philosophy
  – *Modular*, extensible, general to many domains
    □ Being used for CPU, GPU, TPU, FPGA, HW, quantum, ....
  – Easy to learn, great for research
  – MLIR + LLVM IR + RISC-V CodeGen = 💘💖

https://mlir.llvm.org
MLIR (cont.)

• MLIR is a novel approach to building reusable and extensible compiler infrastructure
  − Addresses software fragmentation, compilation for heterogeneous hardware
  − Significantly reducing the cost of building domain specific compilers, and connecting existing compilers together

• MLIR is intended to be a hybrid IR which can support multiple different requirements in a unified infrastructure
  − The ability to represent dataflow graphs (such as in TensorFlow)
  − Ability to host HPC-style loop optimizations across kernels, and to transform memory layouts of data
  − Ability to represent target-specific operations, e.g. accelerator-specific high-level operations.
  − Quantization and other graph transformations done on a Deep-Learning graph.

[1] MLIR: Scaling Compiler Infrastructure for Domain Specific Computation, CGO’2021
Outline

• The Trends [需求趋势]
  – Application, software, domain specific

• The Issues [问题所在]
  – Limitations of classical compilers

• The Solutions [潜在编译技术方案]
  – TVM, MLIR

• Summary
Summary

• Compiler/PL tech more important than ever!
  – The world is evolving fast at the “End of Moore’s Law”
    ● Changing assumptions, expanding possibilities

• HW changes require new programming models and approaches:
  – Various models and frameworks
  – More high-level semantics

• We need compiler and PL experts to step up!
References


The Golden Age of Compilers
in an era of Hardware/Software co-design

张献伟
xianweiz.github.io
DCS290, 06/24/2021
RISC-V+MLIR: Uniting an Industry

• The free and open RISC instruction set architecture
  – Free and open ISA enabling a new era of processor innovation through open standard collaboration [彻底开放]
  – RISC-V ISA delivers a new level of open, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation
Co-design of HW and SW design

• CIRCT: Circuit IR for Compilers and Tools
  – LLVM incubator project built on MLIR & LLVM
  – Composable toolchain for different aspects of hardware design / EDA processes
  – To unite HW design tools community and “accelerate” design of the accelerators