The Golden Age of Compilers

in an era of Hardware/Software co-design

International Conference on
Architectural Support for Programming Languages and
Operating Systems (ASPLOS 2021)

Chris Lattner
SiFive Inc

April 19, 2021

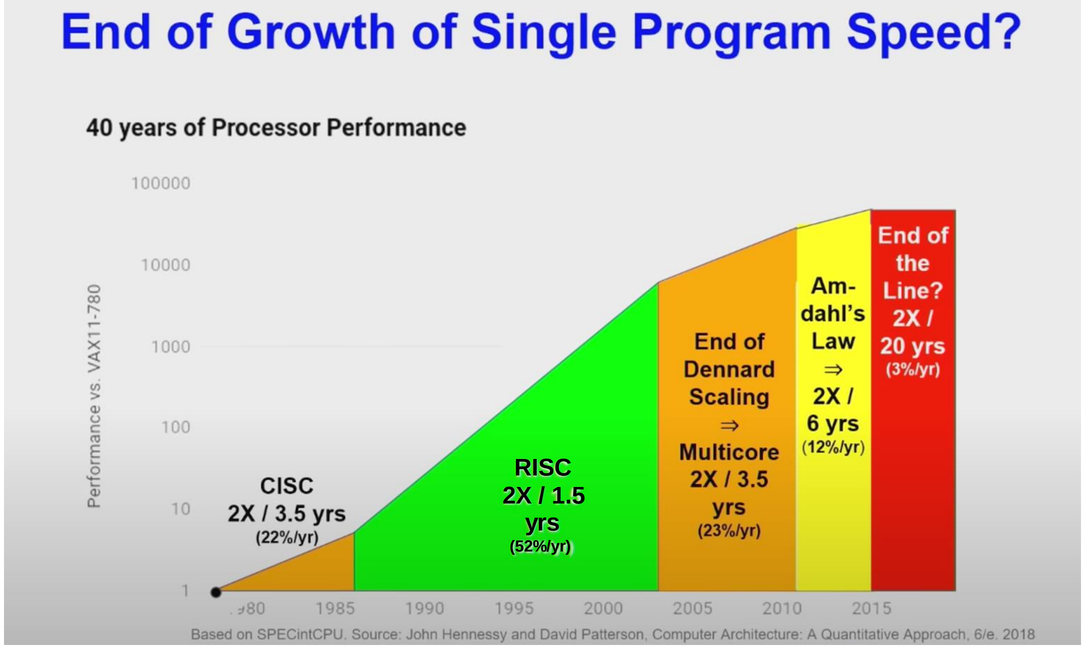
YouTube Video Recording

Let’s talk compilers + accelerators

- Classical Compiler Design
- Modular Compiler Infrastructure
- Domain Specific Architectures
- Accelerator Compilers
- Silicon Compilers
- A Golden Age of Compilers

A New Golden Age for Computer Architecture

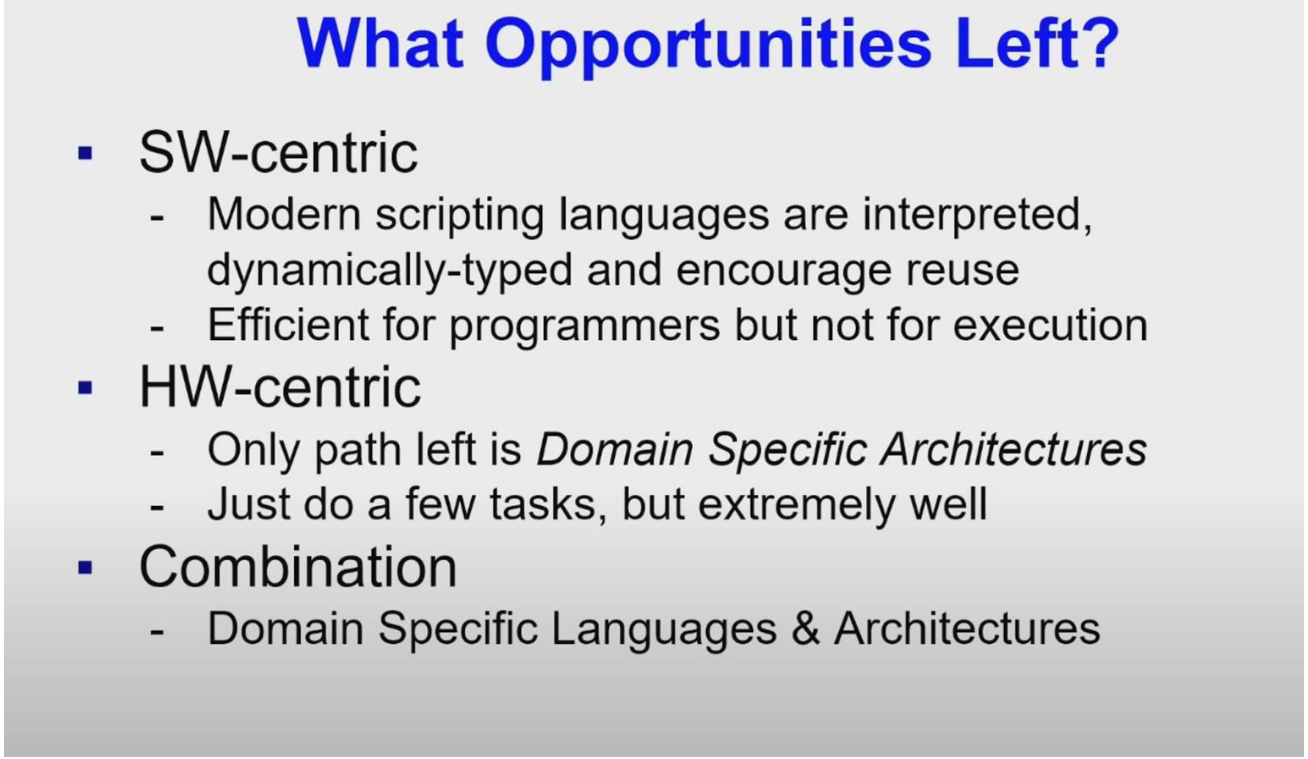
John L. Hennessy, David A. Patterson; June 2018



End of Growth of Single Program Speed?

Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

[cite] Turing Lecture, Hennessy, Patterson; June 2018 / CACM Feb 2019

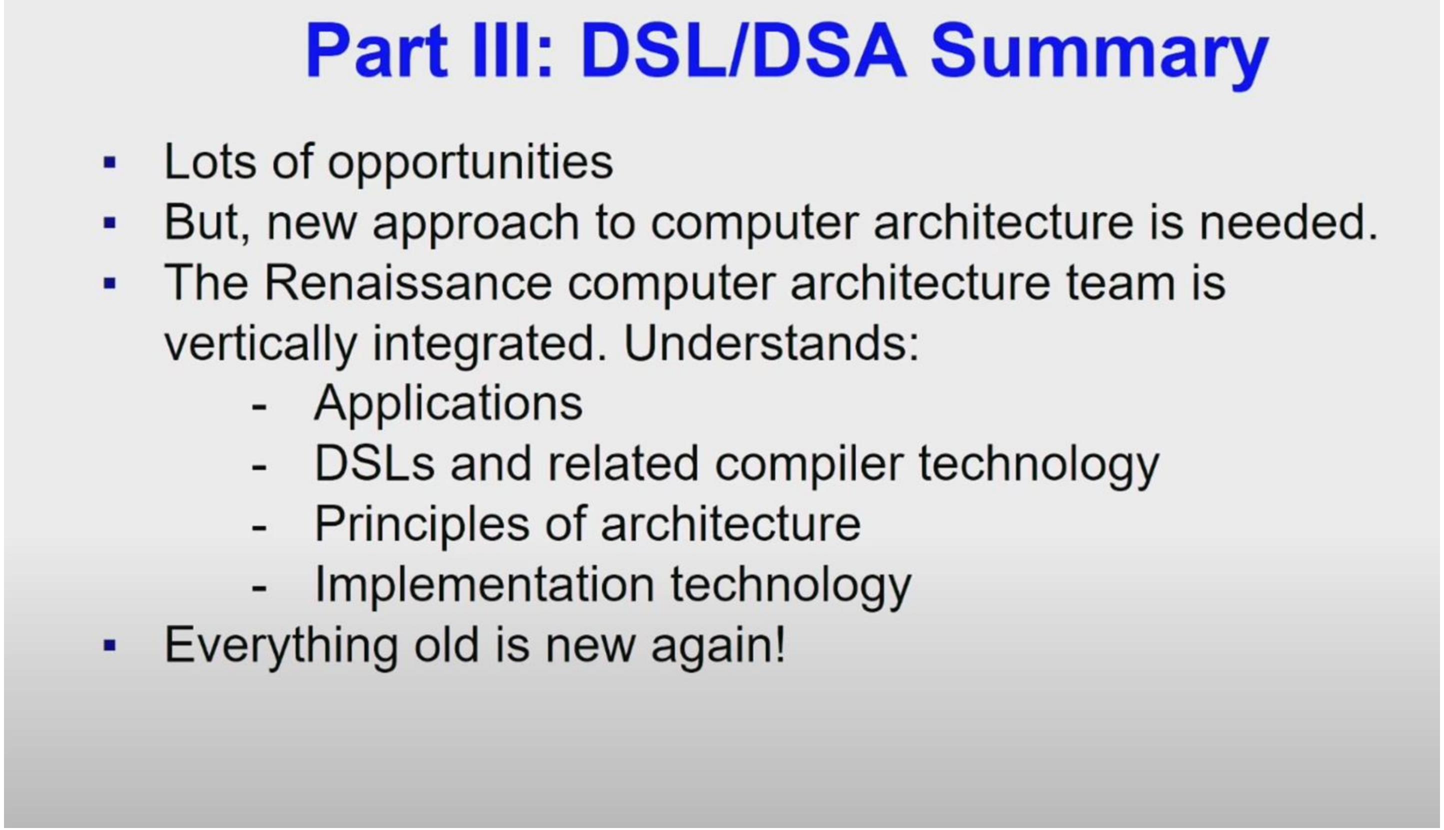


What Opportunities Left?

- SW-centric
  - Modern scripting languages are interpreted, dynamically-typed and encourage reuse
  - Efficient for programmers but not for execution
- HW-centric
  - Only path left is *Domain Specific Architectures*
  - Just do a few tasks, but extremely well
- Combination
  - Domain Specific Languages & Architectures

HW / SW co-design is the best way to expose parallelism of silicon… and utilize it

[cite] Hennessy, Patterson; June 2018 / CACM Feb 2019



Part III: DSL/DSA Summary

- Lots of opportunities
- But, new approach to computer architecture is needed.
- The Renaissance computer architecture team is vertically integrated. Understands:
  - Applications
  - DSLs and related compiler technology
  - Principles of architecture
  - Implementation technology
- Everything old is new again!

🧐 How do we program these things?

[cite] Hennessy, Patterson; June 2018 / CACM Feb 2019

Hardware is getting **hard**er

Modern compute acceleration platforms are multi-level and explicit:

- Scalar, SIMD/Vector, Multi-core, Multi-package, Multi-rack
- Non-coherent memory subsystems increase efficiency

Heterogeneous compute incorporating domain-specific accelerators

- Standard in high-end SoCs, domain-specific hard blocks in FPGAs

Many accelerator IPs are configurable:

- Optional extensions, tile / core count, memory hierarchy, etc

🤯 How can “normal people” write Software for this in the first place?
… and how can you *afford* to build generation-specific SW?

Next-Gen compilers and 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

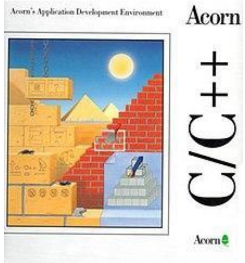
This opportunity is beckoning a golden age in compiler and PL technology!

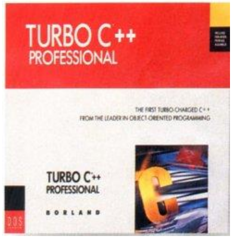
Let's learn from the past, then project into the future 🚀

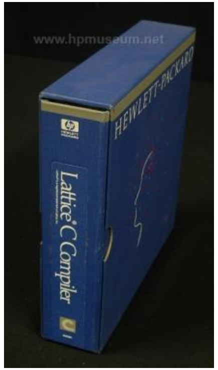
Classical Compiler Design

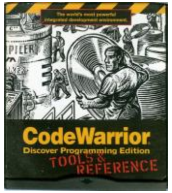
C Compilers leading into the early 90s

IBM









About Microsoft Visual C++

Microsoft Visual C++ 1.00

Copyright © 1993 Microsoft Corporation
Portions developed for Microsoft by Nat Systèmes

This copy of Microsoft Visual C++ is licensed to:

Horkin Mcphlanders

Available Memory: 491567 KB Free
Math Co-processor: Present
Disk Space: 666848 KB Free

⇒ Expensive, not very compatible, inconsistencies abound

- … and didn’t share any code

Also: Came in boxes, with printed manuals, often on floppy disks!

Three Phase Compiler Design

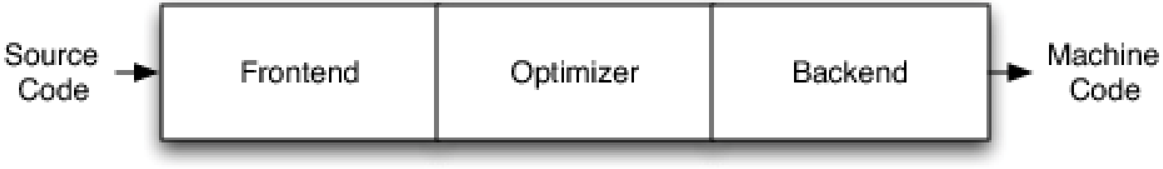


Figure 11.1: Three Major Components of a Three-Phase Compiler

[cite] LLVM @ The Architecture of Open Source Applications

FOSS Enables Collaboration & Reuse

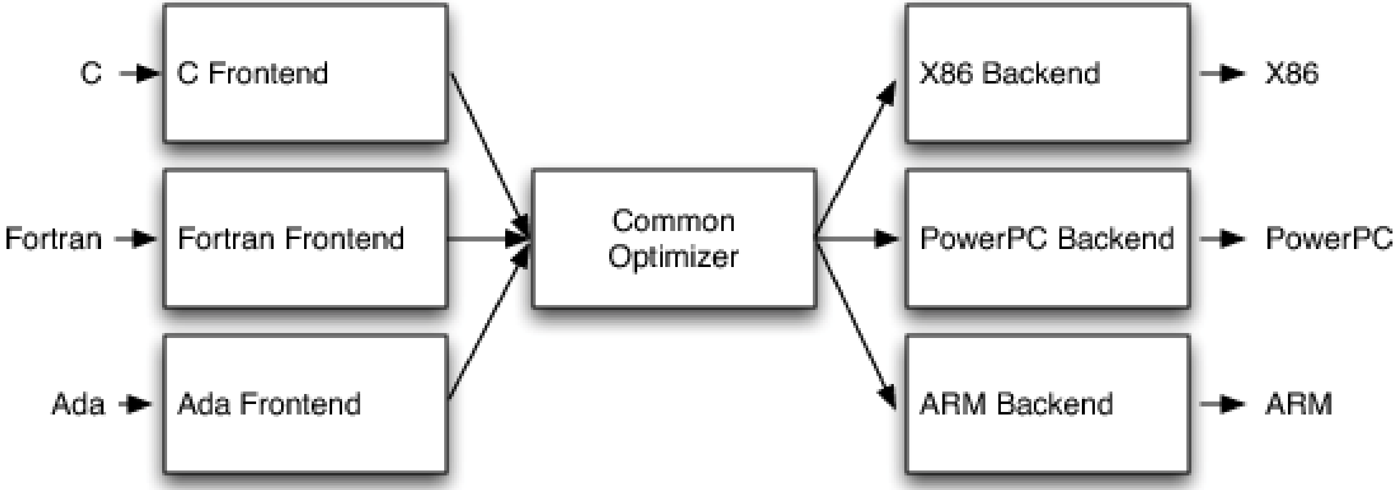


Figure 11.2: Retargetablity

One frontend for many backends, one backend for many frontends

Lessons Learned

Achieved “O(frontend+backends)” scalability of compiler ecosystem

Larger center of gravity concentrated scarce compiler engineering effort

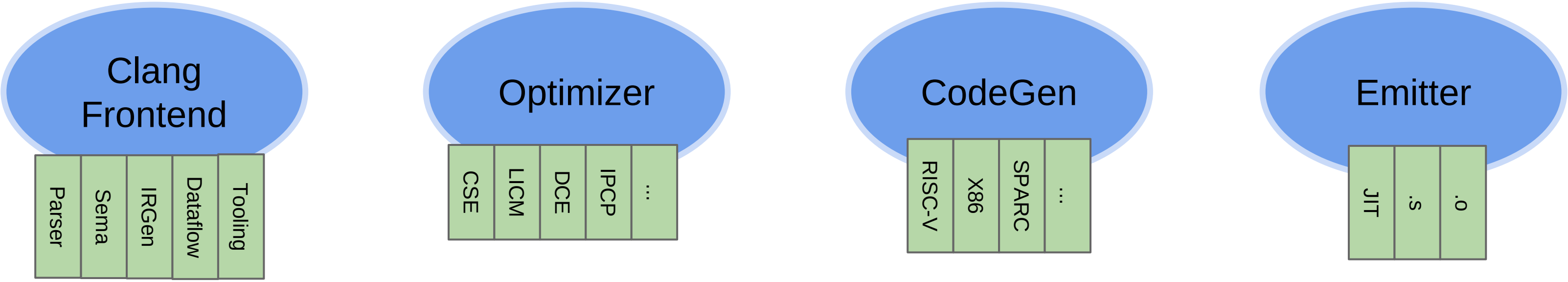
- Enables innovations in languages, frontends and backends

Reduced 💢**fragmentation**, standardized “C in practice”

- Enabled new business models 💸
- Untied the CPU ISA war from inconsequential impl details

Modular Compiler Infrastructure

Library Based Design



Key insight: Compilers as libraries, not an app!

- Enable embedding in other applications
- Mix and match components
- No hard coded lowering pipeline

LLVM

Components and interfaces!

Better than monolithic approaches for large scale designs:

- Easier to understand and document components
- Easier to test
- Easier to iterate and replace
- Easier to subset
- Easier to scale the community

LLVM

Lessons Learned

Larger center of gravity concentrated scarce compiler engineering effort

- Enables innovations in languages, frontends and backends

Reduced 💢**fragmentation** of JIT compilers, standardized CPU codegen

- Enabled new business models
- Databases, graphics shader compilers, GPGPU, EDA HLS tools, ...

Scalable community architecture:

- Design methodology / developer policies
- Community policies: inclusion, licensing, extensions etc

LLVM

Limitations of LLVM

20 years in perspective on LLVM:

- “One size fits all” quickly turns into “one size fits none”
- LLVM is: 👍 CPUs, “just ok” 👈 for SIMT, but 👎 for many accelerators
- … is not great for parallel programming models 💩

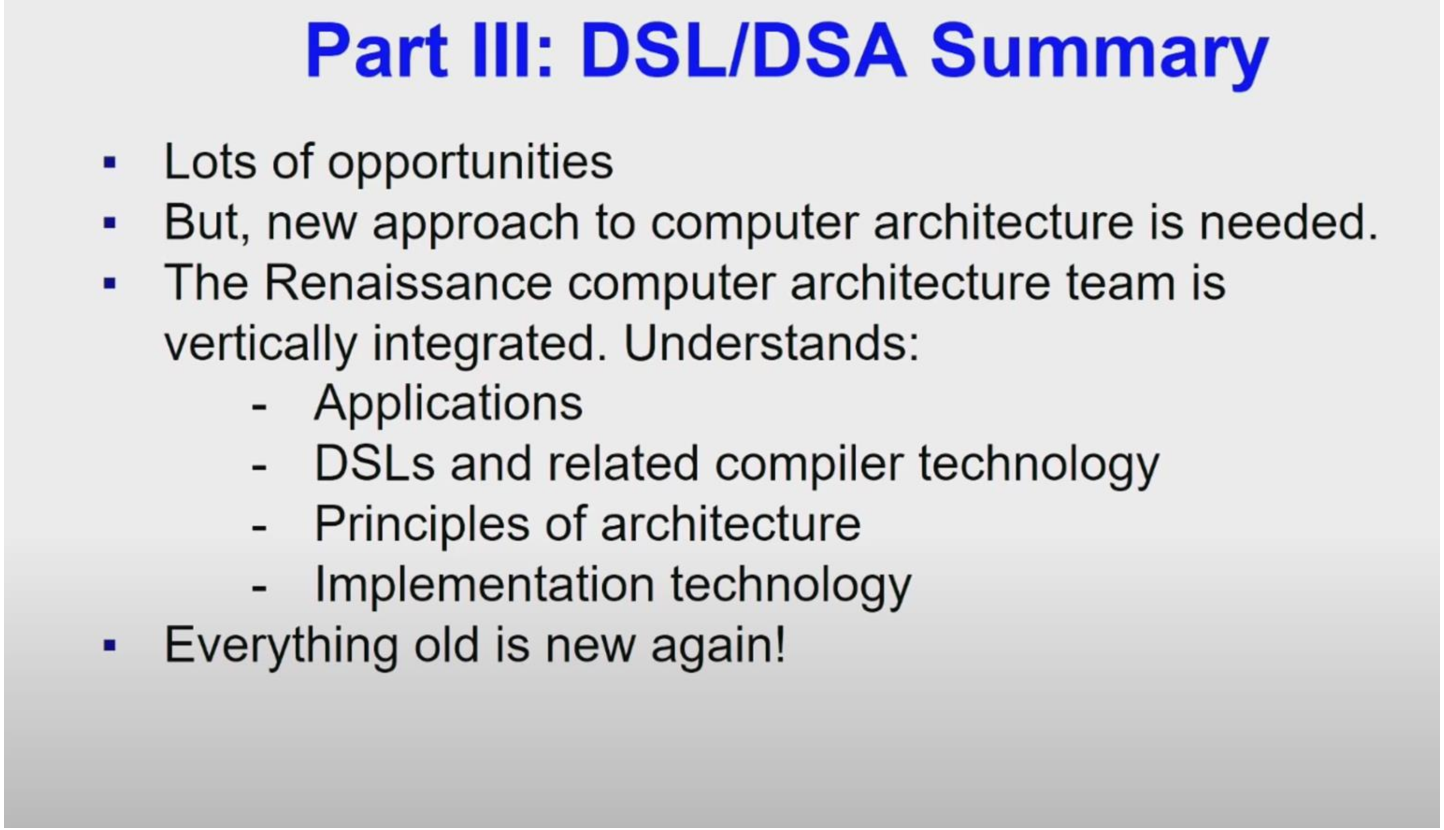
Engineering is “pretty good” but could be better:

- Lots of redundancy/reimplementation @ different levels of abstraction
- Deeper discussion @ CGO 2020 talk

Going beyond basic CPUs means going beyond LLVM IR!

LLVM

Domain Specific Architectures

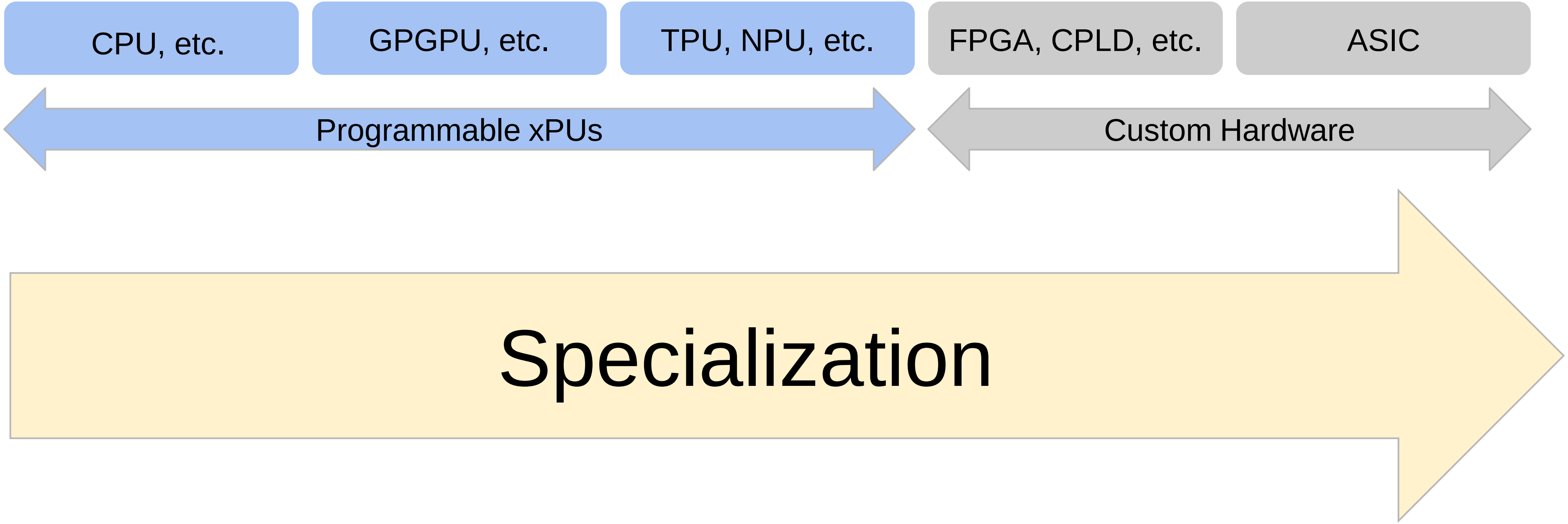


Part III: DSL/DSA Summary

- Lots of opportunities
- But, new approach to computer architecture is needed.
- The Renaissance computer architecture team is vertically integrated. Understands:
    - Applications
    - DSLs and related compiler technology
    - Principles of architecture
    - Implementation technology
- Everything old is new again!

[cite] Hennessy, Patterson; June 2018 / CACM Feb 2019

It’s happening!



[cite] Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications, **Mike Urbach**, MLSys Chips And Compilers Symposium 2021

Lots of players! (an incomplete list!)

CPU, etc.

- SiFive
- AMD
- arm
- intel

GPGPU, etc.

- NVIDIA
- AMD
- intel

TPU, NPU, etc.

- Google
- intel
- arm
- XILINX
- cerebras

FPGA, CPLD, etc.

- XILINX
- MICROCHIP
- AMD
- intel

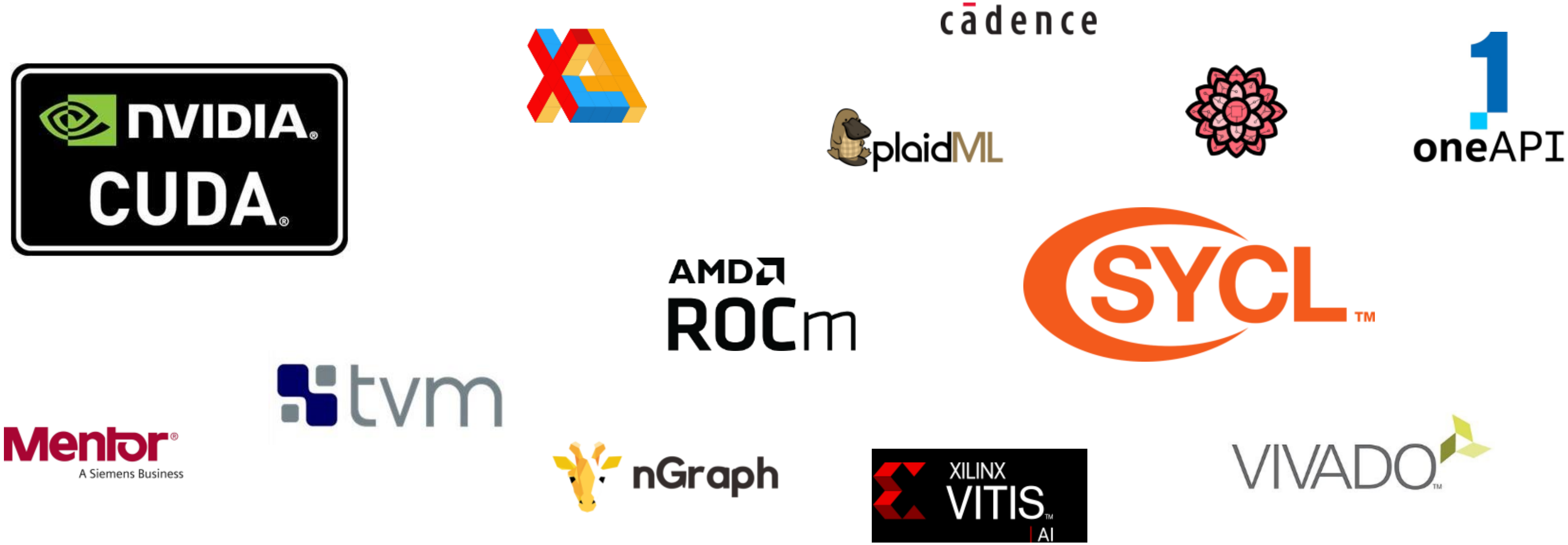
ASIC

- openfive
- tsmc
- SAMSUNG
- intel

Programmable xPUs

Custom Hardware

How do we compile for this?

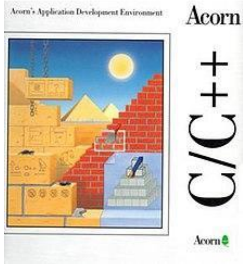


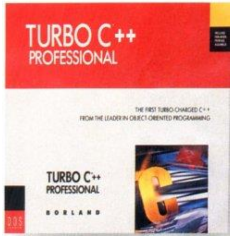
⇒ Not very compatible, inconsistent quality and scope

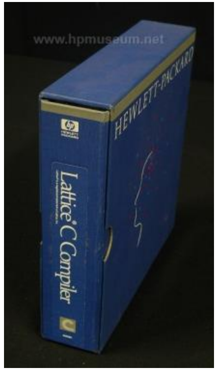
- … and don’t share much code

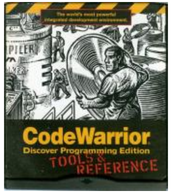
We’ve seen this before!

IBM




TURBO C++
PROFESSIONAL
THE FIRST TURBO-CHARGED C++
FROM THE LEADER IN OBJECT-ORIENTED PROGRAMMING
TURBO C++
PROFESSIONAL
BORLAND





About Microsoft Visual C++

Microsoft Visual C++ 1.00

Copyright © 1993 Microsoft Corporation
Portions developed for Microsoft by Nat Systèmes

This copy of Microsoft Visual C++ is licensed to:

Horkin Mcphlanders

Available Memory: 491567 KB Free
Math Co-processor: Present
Disk Space: 666848 KB Free

OK

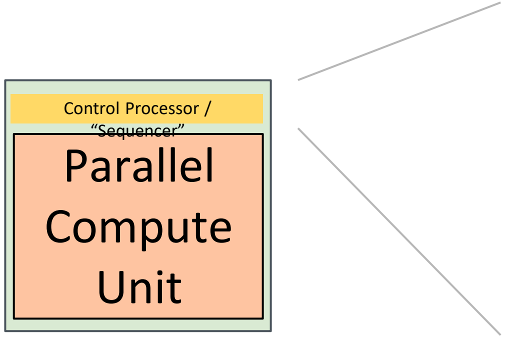
We need some unifying theories!

We need:

- “O(frontend+backends)” scalability of compiler ecosystem
- Larger center of gravity concentrated scarce compiler engineering effort
- Reduced 💢**fragmentation**:
    - Ability to innovate in the programming model
    - ... without reinventing the whole stack

Accelerator Compilers

How do accelerators work?



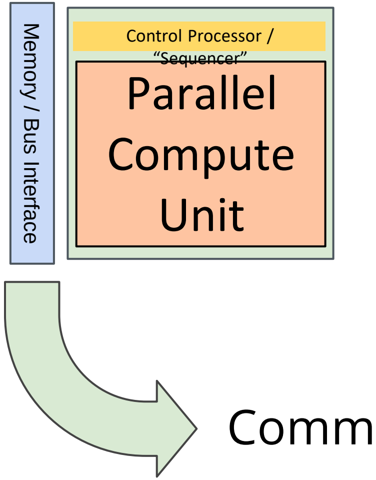
Control Processor / Sequencer

- Executes commands by the host driver app
- Handles booting and other housekeeping
- Diagnostics, security, debug, other functions

Some accelerators may do significantly more!

Ratio of control to parallel compute vary, as do the internal arch's of both

Add a system interface

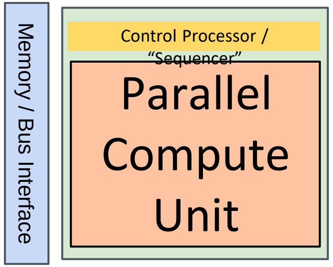


Communicate w other parts of the SoC, or to off-chip resources

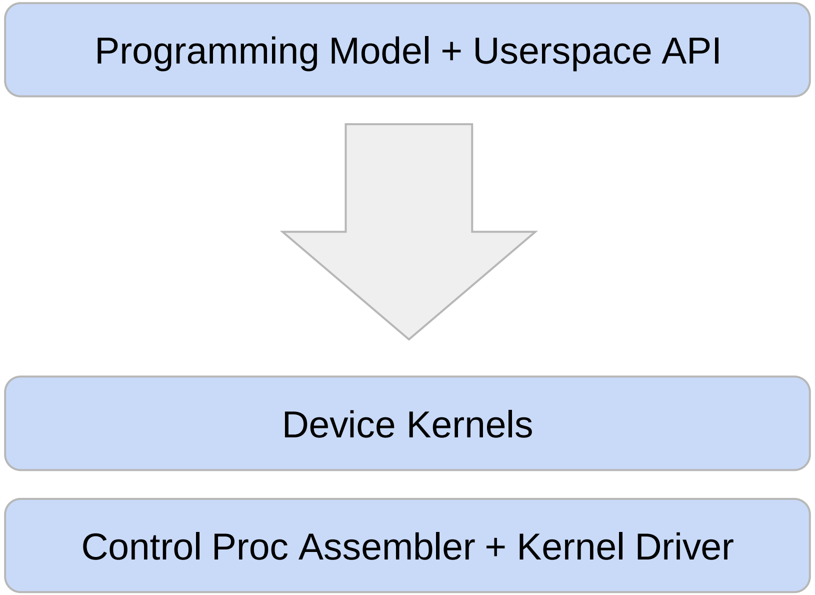
Including DDR, HBM, … AMBA, PCI, CXL, etc depending on integration level

“Oops we need some software”

Hardware



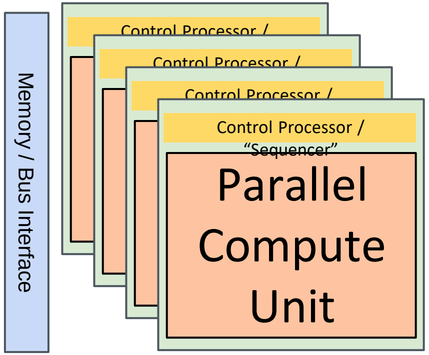
Software


Programming Model + Userspace API
Device Kernels
Control Proc Assembler + Kernel Driver

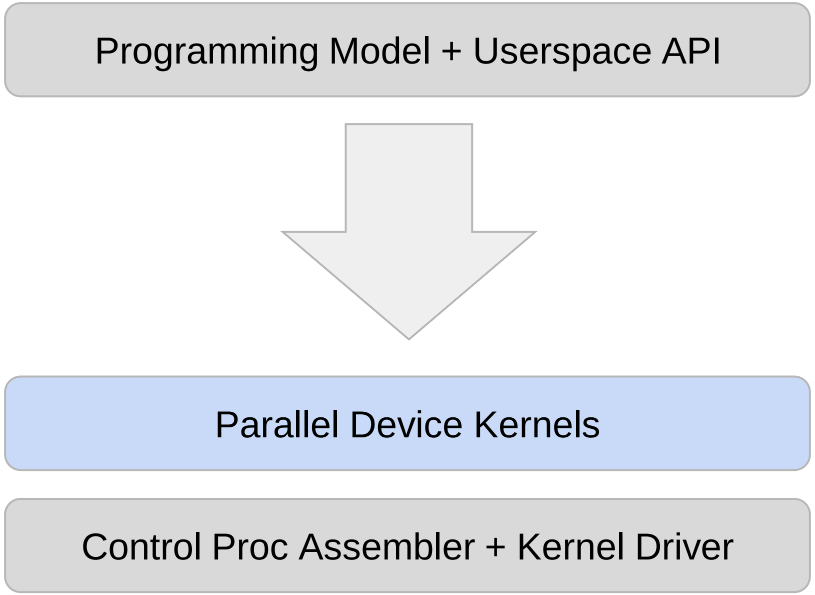
The SW people are called in after the accelerator is defined to “make it work”

Larger accelerators go multicore/SIMT…

Hardware



Software



Programming Model + Userspace API

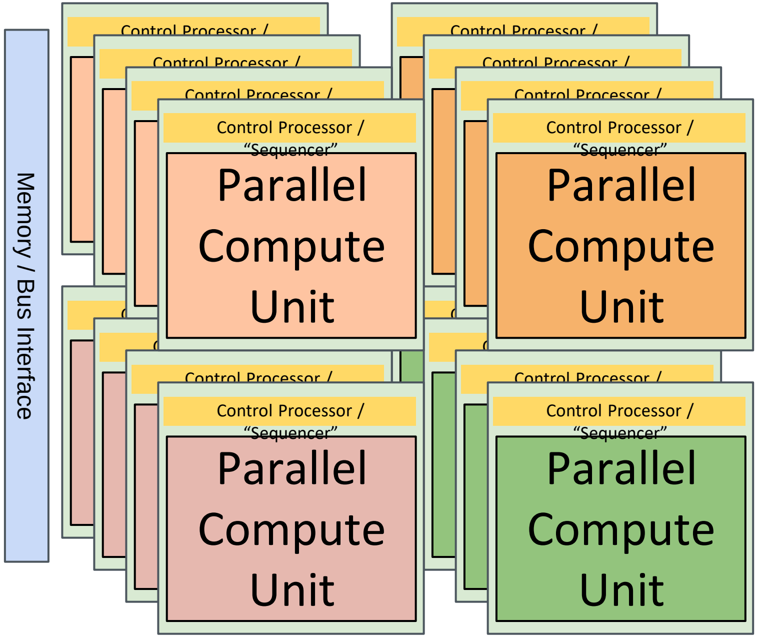
Parallel Device Kernels

Control Proc Assembler + Kernel Driver

Use of more HW area is desired, requiring parallel control logic

Tiling and heterogeneity for generality

Hardware



Software

Programming Model + Userspace API

Multistream Mgmt / Interop Parallelism
Memory + Communication Optimization
Heterogenous Device + Host fallback

Parallel Device Kernels

Control Proc Assembler + Kernel Driver

⇒ Also, hierarchical compute at the board, rack, and datacenter level

Pro & Cons of hand written kernels

Benefits:

- Easy to get started, ability to get peak performance, hackability

Pro & Cons of hand written kernels

Benefits:

- Easy to get started, ability to get peak performance, hackability

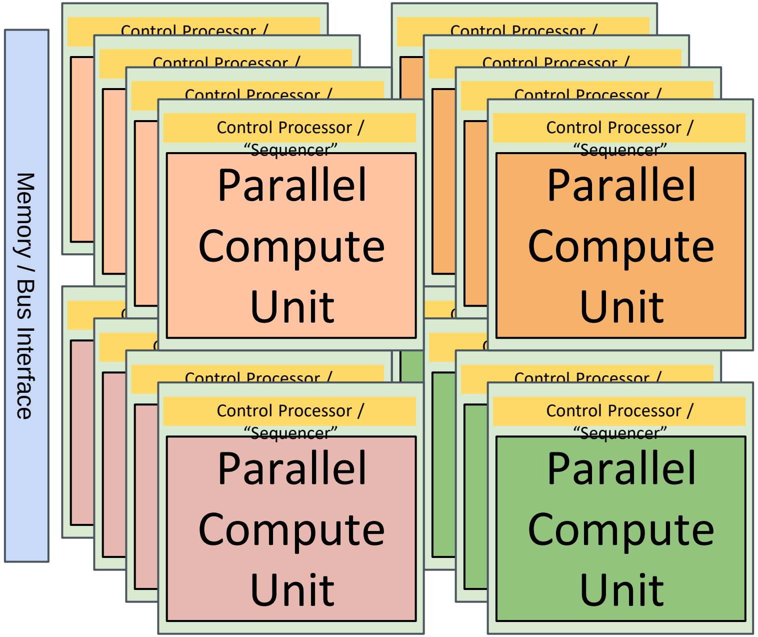
Problem: **hand written kernels don’t scale**

- Expensive to maintain a library of 100’s to 1000’s of kernels
- Don’t scale to configurable IPs, not even memory hierarchy dimensions
- Don’t scale to device families, or evolving µarch’s over time
- Eventually end up limiting HW design space exploration / evolution

Often addressed with metaprogramming (aka “mini compilers”)

“DSA Compilers” to the rescue

Hardware


Memory / Bus Interface
Control Processor / “Sequencer”
Parallel Compute Unit
Control Processor / “Sequencer”
Parallel Compute Unit
Control Processor / “Sequencer”
Parallel Compute Unit
Control Processor / “Sequencer”
Parallel Compute Unit

Software

Programming Model + Userspace API

Accelerator Kernel Compiler

Multistream Mgmt / Interop Parallelism
Memory + Communication Optimization
Heterogenous Device + Host fallback
Kernel Code Generation

Control Proc Assembler + Kernel Driver

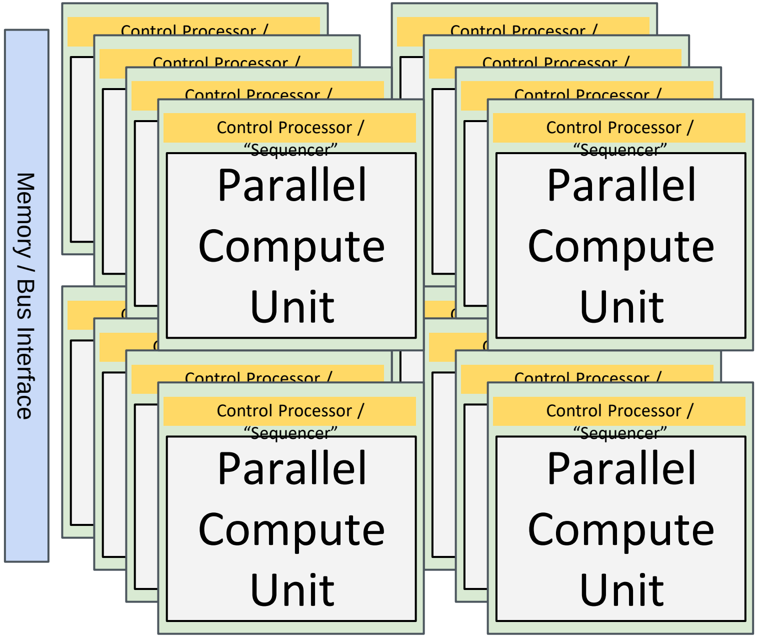
This is hard!

… and we keep reinventing it over and over again

… at the expense of usability and quality

Mostly needless **reinvention**, not **co-design**!

Hardware



Software

Programming Model + Userspace API

Accelerator Kernel Compiler

Multistream Mgmt / Interop Parallelism
Memory + Communication Optimization
Heterogenous Device + Host fallback
Kernel Code Generation

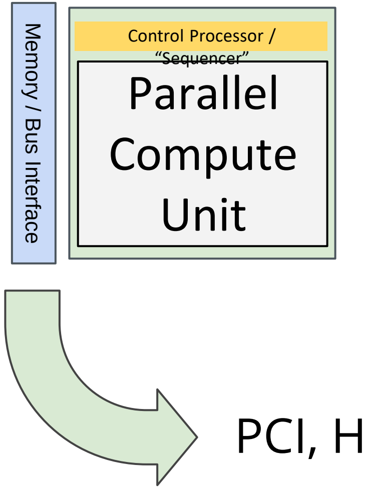
Control Proc Assembler + Kernel Driver

Most complexity is in non-differentiated (table stakes) components

Innovate where it matters

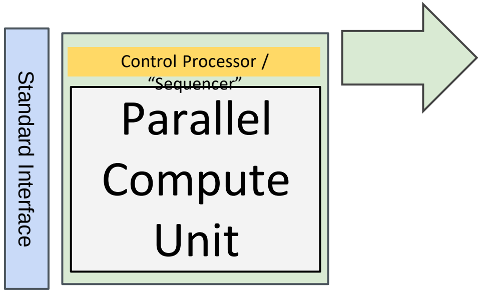
… use open standards to accelerate the rest

Industry already standardized the buses



PCI, HBM, DDR, CXL, AMBA, etc are all standardized

Standardize the Control Processor?



SW is a bigger problem than HW for accelerators:

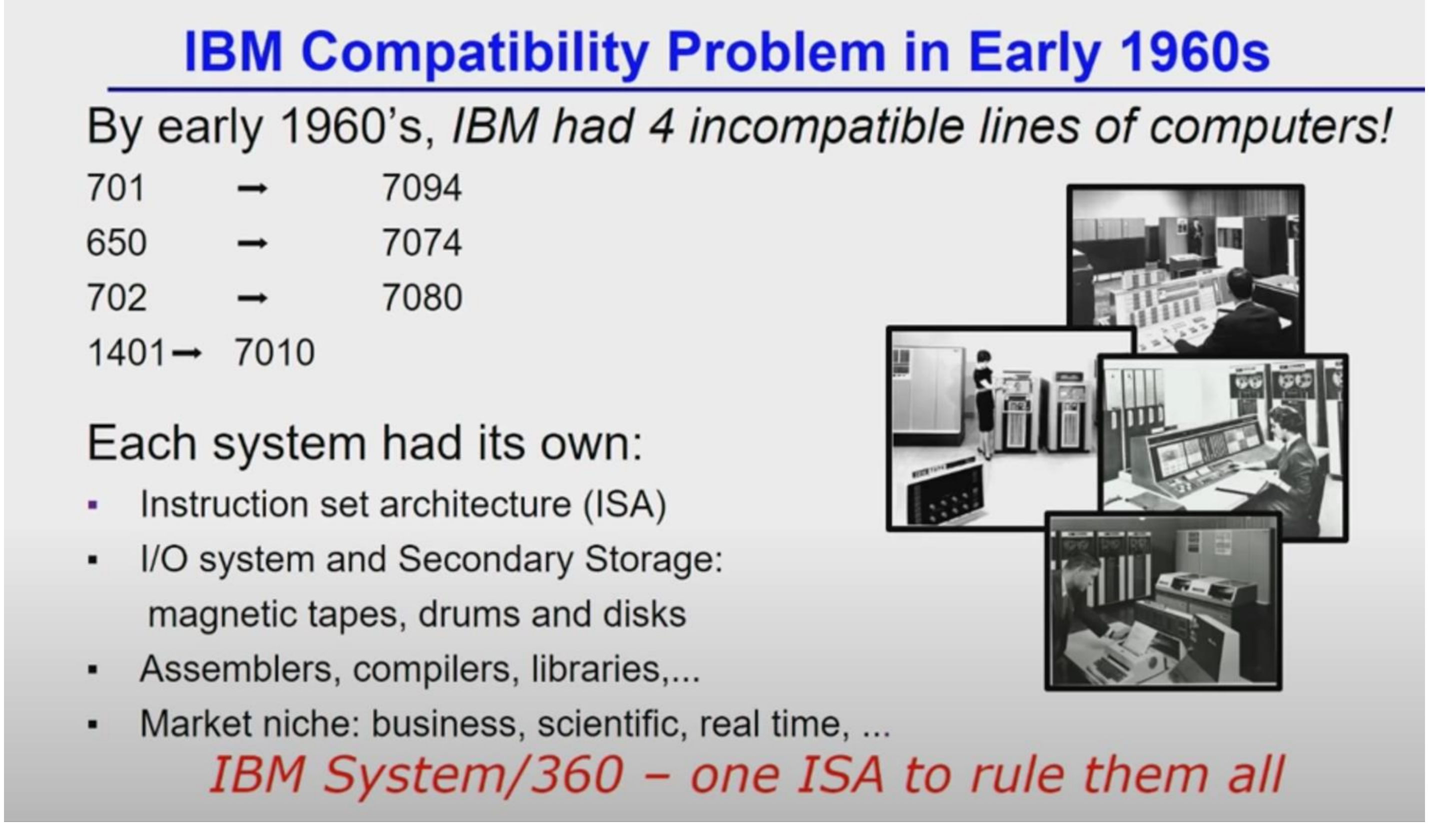
- Control processor is bottom of the SW stack

We fool ourselves into building trivial CPUs:

- it can seem fun to design a new solution here
- … except reset, debug, power management, security, etc (the hard parts!)

"Saving a few gates" slows down what matters

- … and hobbles the critical path: **software**



IBM Compatibility Problem in Early 1960s

By early 1960's, *IBM had 4 incompatible lines of computers!*

701 → 7094

650 → 7074

702 → 7080

1401 → 7010

Each system had its own:

- Instruction set architecture (ISA)
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche: business, scientific, real time, ...

IBM System/360 – one ISA to rule them all

[cite] Hennessy, Patterson; June 2018 / CACM Feb 2019

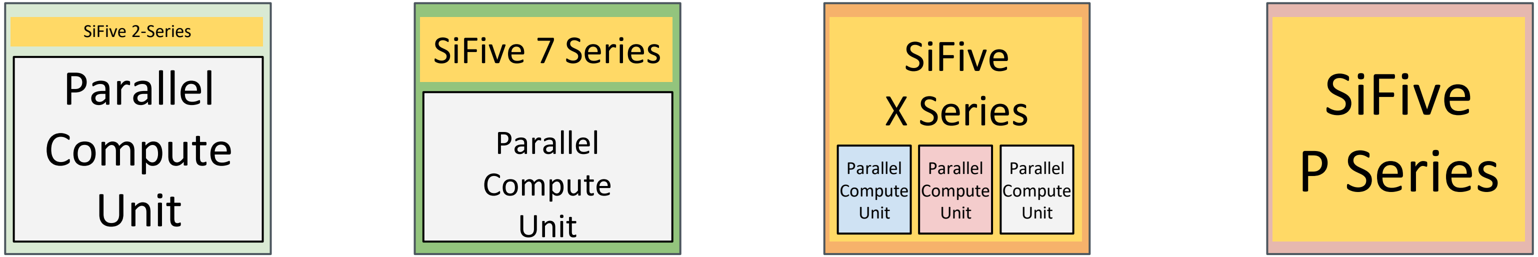
Open Industry Standard

- Many implementations available

Modular and subset-able ISA design:

- Extensibility allows easy addition of heterogeneous units

Scalability allows full spectrum of design points!


SiFive 2-Series
Parallel Compute Unit

Hard Coded Accelerator

SiFive 7 Series
Parallel Compute Unit

Programmable Accelerator

SiFive X Series
Parallel Compute Unit
Parallel Compute Unit
Parallel Compute Unit

Heterogeneous Workload Accelerator

SiFive P Series

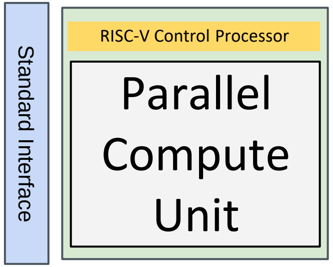
General Purpose CPU

“15,000 gates? I can’t count that low!”

Cliff Young, TPU architect, Google Brain

MLSys 2021 Chips and Compilers Symposium Panel

Standardize your base Software



Write your kernels in C or LLVM IR!

- Use *existing* code generators
- Use *existing* simulators
- Step through them in a **debugger**

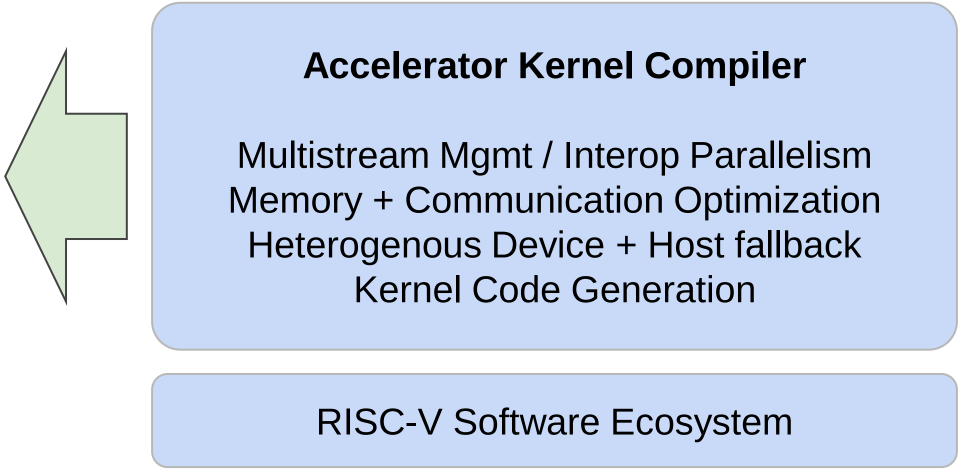
RISC-V **Compiler** + Kernel Drivers

The next frontier: DSA Compilers?

“No one size fits all” compiler!

Shape of the problem is the same…
… but the accel details always vary

How do we get reuse?



Accelerator Kernel Compiler

Multistream Mgmt / Interop Parallelism
Memory + Communication Optimization
Heterogenous Device + Host fallback
Kernel Code Generation

RISC-V Software Ecosystem

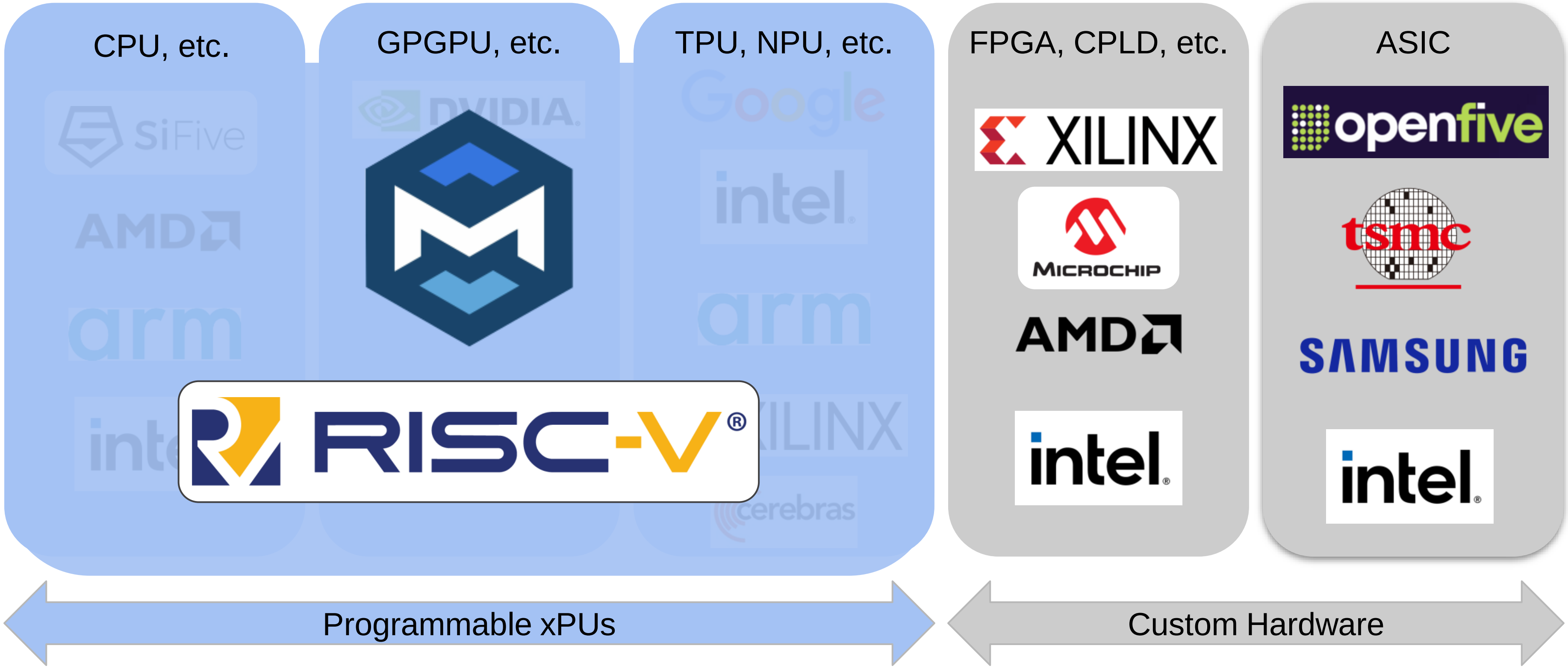
MLIR: Compiler Infra at the End of Moore’s Law

- **M**ulti-**L**evel **I**ntermediate **R**epresentation
- 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

See more (e.g.):
2020 CGO Keynote Talk Slides
2021 CGO Paper

RISC-V+MLIR: Uniting an Industry



CPU, etc.

GPGPU, etc.

TPU, NPU, etc.

FPGA, CPLD, etc.

ASIC

Programmable xPUs

Custom Hardware

What is the benefit of this?

Larger center of gravity concentrated scarce compiler engineering effort

- Enables innovations in programming models and hardware

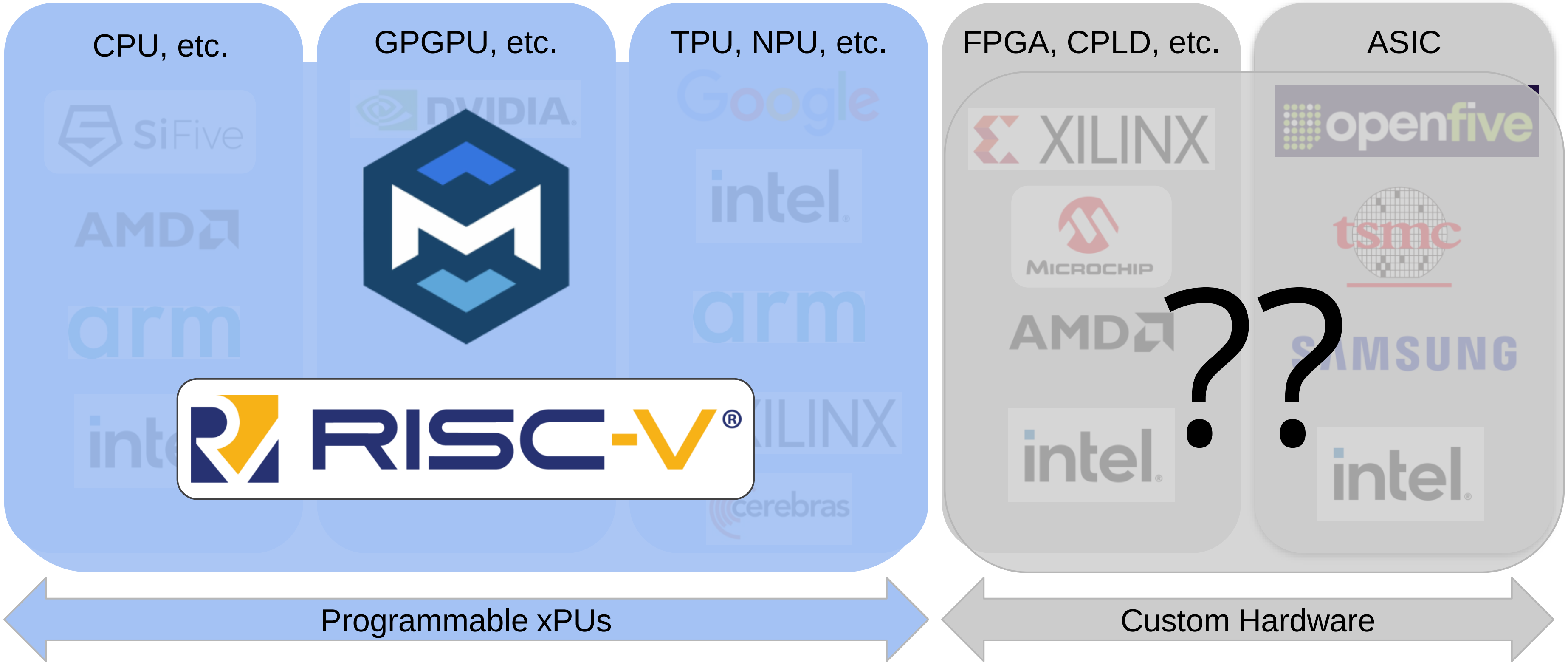
Achieved “O(frontend+backends)” scalability of compiler ecosystem

Reduced 💢**fragmentation**, improved 🧩 ***modularity***

- Focus on the differentiated parts of the stack

But… what about hardware?

HW design is fragmented too



CPU, etc.

GPGPU, etc.

TPU, NPU, etc.

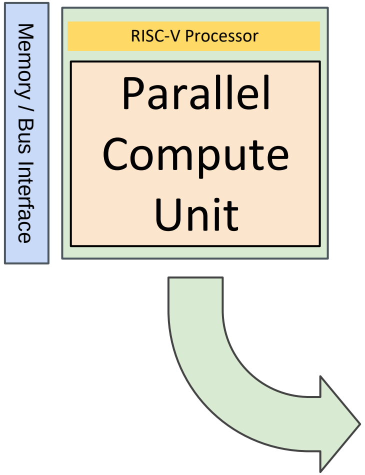
FPGA, CPLD, etc.

ASIC

Programmable xPUs

Custom Hardware

Building Parallel Compute Units?



Notice how I conveniently omitted how to build the “interesting” part!

Silicon Compilers

Hardware Design is ripe with opportunity

SystemVerilog is industry standard, but:

- Huge, complicated, incompletely implemented
- Is it an IR? or programming language for humans? neither? both?

EDA tools are mature, but not always:

- ... innovating rapidly, now that process technology has slowed
- ... designed for usability
- ... using best practices in SW architecture
- ... cost efficient

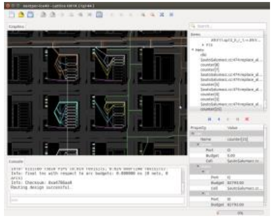
Open Source tools to the rescue?

Wonderful ecosystem of Open Source tools, but:

- Generally aspiring to be “as good” as proprietary tools
- Fragmented communities, not sharing much code
- Monolithic designs connected by unfortunate standards

SymbiFlow

OpenROAD



nextpnr

Yosys

ICARUS

VERILATOR

Innovation Explosion Underway!

Research is producing new HW design models and abstraction approaches

CHISEL

LLHD

Magma

bluespec

Dahlia

XLS

SpinalHDL

Aetherling

migen

A great opportunity to pull PL + type system + compiler tech from SW world…

… held back by poor interop standards and ecosystem 💢fragmentation

See also: ASPLOS LATTE’21 Workshop

CIRCT: **C**ircuit **IR** for **C**ompilers and **T**ools

Compiler infrastructure for design and verification

- LLVM incubator project built on **MLIR** & **LLVM**
- Composable toolchain for different aspects of hardware design / EDA processes
- 🧩 **Modularity**, library based design, ecosystem
- High quality, usability, performance

Goals:

- Unite HW design tools community
- “Accelerate” design of the accelerators!

https://circt.llvm.org

CIRCT Ambition / Path Ahead

Support **multiple** different “hardware design models” in one framework:

- Generators, HLS, atomic transactions, ...

Increase **abstraction level** in the hardware design IR:

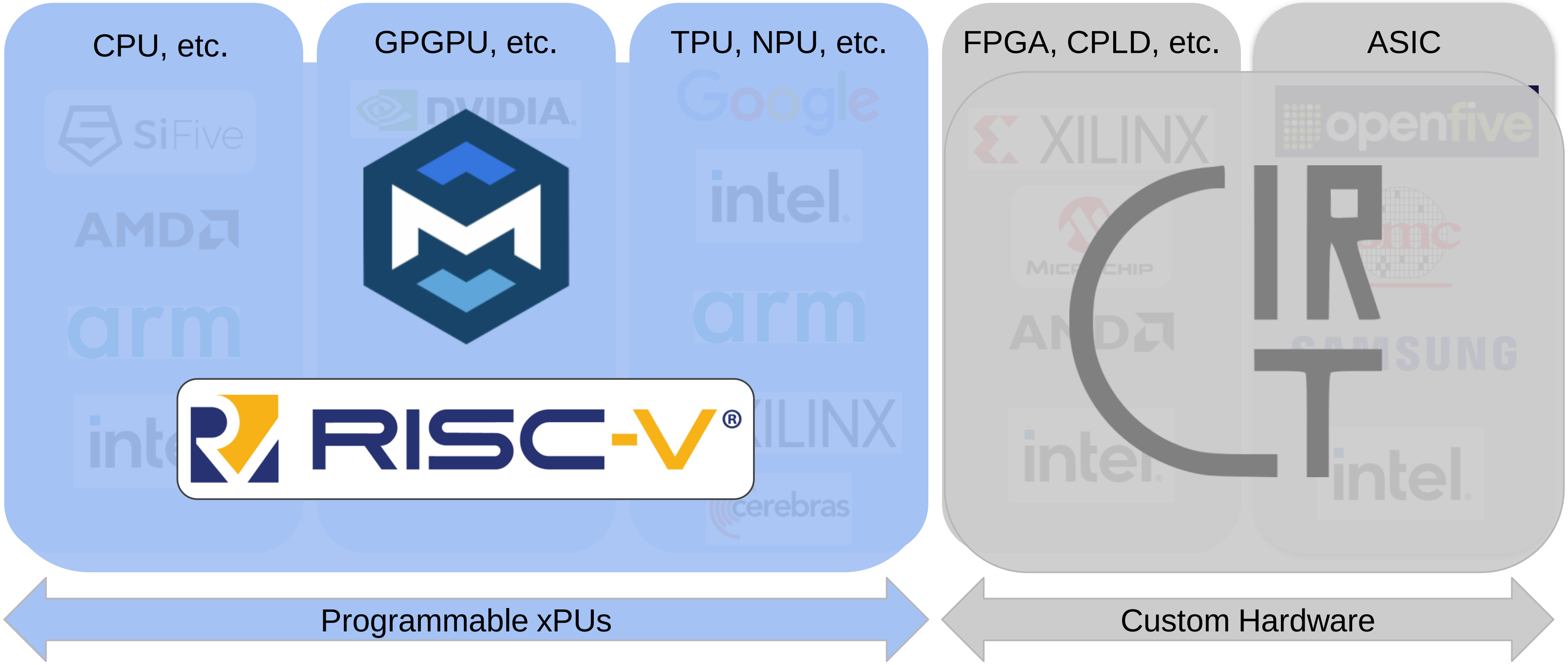
- Integrate modern type system features from the SW world
- Capture more design intent, higher level verification and tools
- Better integrate formal methods into the design flow

Increase **quality** of the tools themselves**:**

- Compile time: shrink development cycle time
- Usability: robust location tracking for good error messages

“10x” design and **verification**, change economics of hardware design

Co-design of HW and SW design



CPU, etc.

GPGPU, etc.

TPU, NPU, etc.

FPGA, CPLD, etc.

ASIC

Programmable xPUs

Custom Hardware

A Golden Age of Compilers

in an era of Hardware/Software co-design

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:

- … and is validating well known but sparsely adopted techniques

We need compiler and PL experts to step up!

Get involved!

https://mlir.llvm.org/

https://circt.llvm.org/

We’re hiring!

SiFive