Re-targetable C/C++ LLVM Compiler for RISC-V

Zdenek Prikryl, Codasip

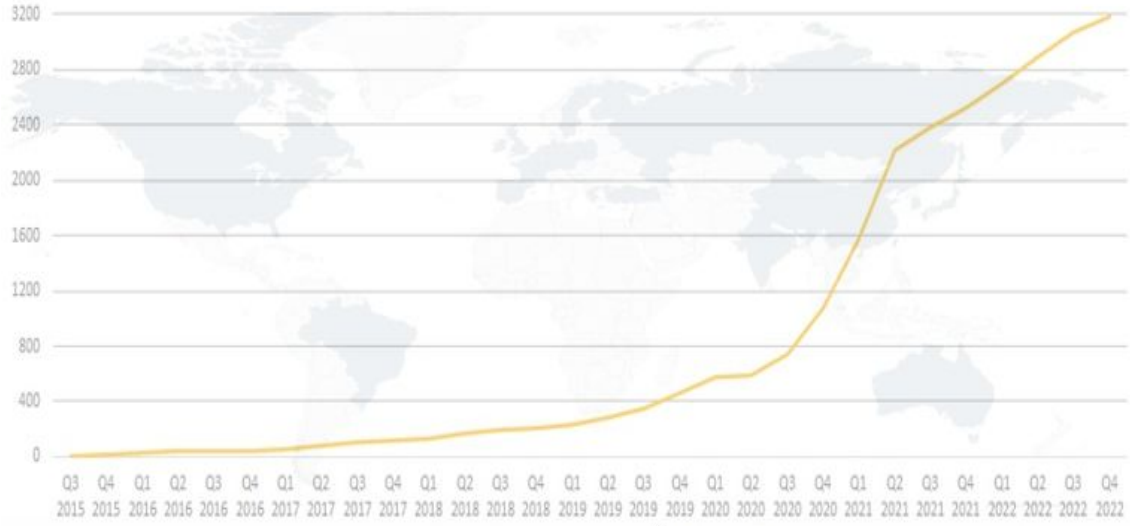
Agenda

- RISC-V
- Codasip
- Re-targetable C/C++ LLVM
- Results
- Conclusion

What is RISC-V®?

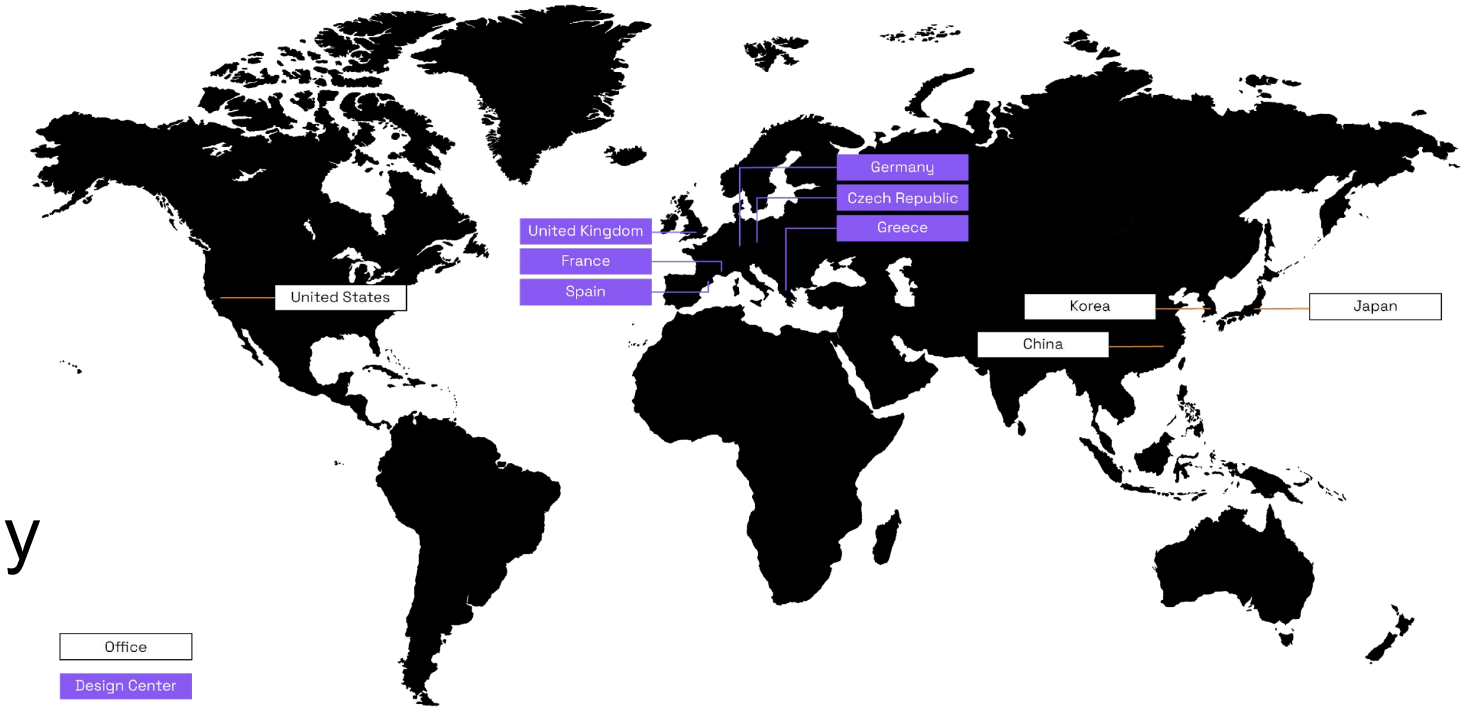
- Open standard
- Modular Instruction Set Architecture
- Tailor an architecture or microarchitecture to a workload
- Allows different implementations
  - Embedded vs. application
  - Open source vs. proprietary, etc.

- Whole ecosystem
- 3,180+ RISC-V members in 70 countries



Codasip Overview

- Founded in 2014
- HQ in Munich
- ~200 employees
- Design teams in Europe
- Processor solution company
- Co-founder
- Enabling custom compute



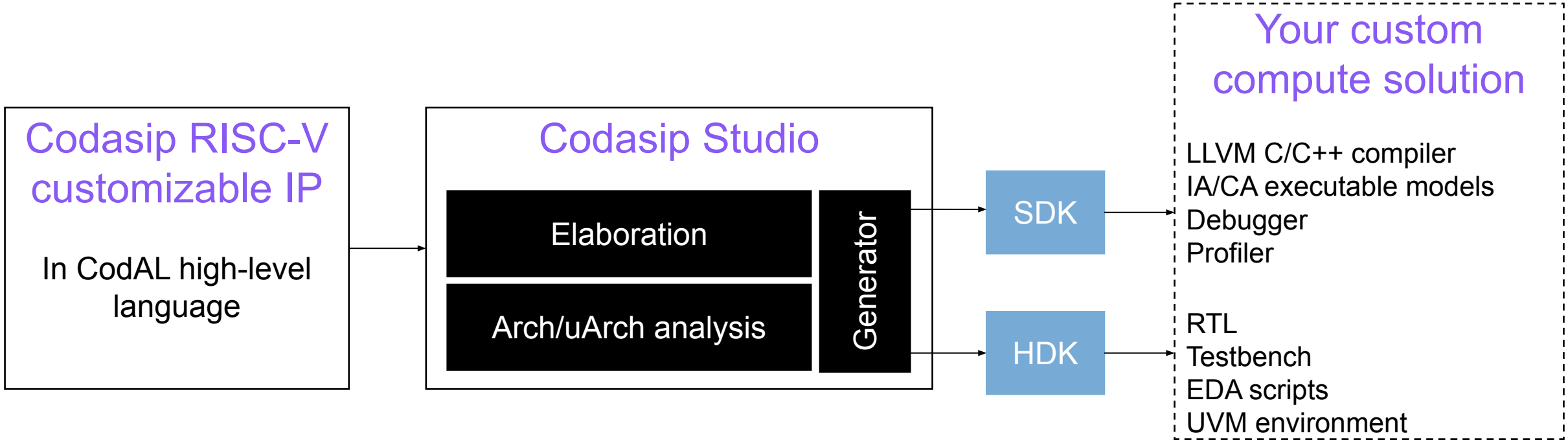
Codasip Studio

- Design automation tool
- Fast architectural iteration loop
  - Profiling & optimization
  - Co-simulation
  - Results analysis
- Using CodAL high-level language
- Generating custom RTL and SDK

Codasip RISC-V Processors

- Production-ready processors
- Written in CodAL
- Designed for customization
- Fully RISC-V compliant
- Best-in-class verification
- Also available as ready-to-use RTL

Codasip Studio and RISC-V Processors

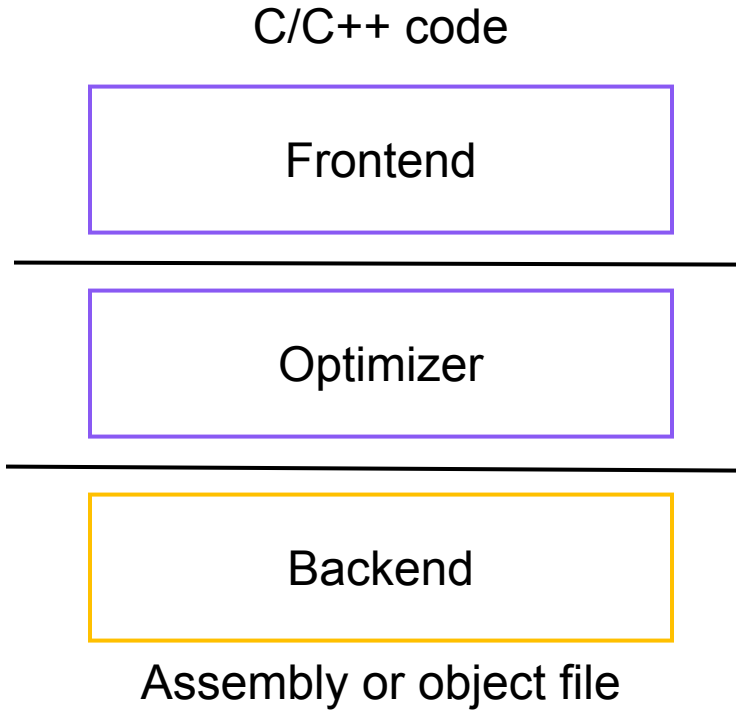


- Start with a standard core
  - Embedded and application cores
  - High quality, production-ready
  - Fully RISC-V compliant
- Differentiate with Codasip Studio
  - Configure / Modify
  - Using CodAL architecture description language

Re-targetable C/C++ LLVM

- Codasip uses LLVM as a base line
- LLVM is the re-targeted based on the CodAL processor description
- Beside the C/C++ compiler, Codasip generates
  - LLVM assembler, disassembler
  - LLVM linker, binutils
  - LLVM debugger (LLDB)
  - …
- Complete SDK/toolchain is generated

- Structure



Benefits and Features

- CodAL processor description serves as an input to the generator
  - Generator extracts:
    - Behavior of every single instruction in a form of a graph
    - Architectural and microarchitectural features for a scheduler or register allocation
    - Application binary interface
    - Peephole and other optimizations
  - Informative report is generated
    - Designer may see which instructions are recognized and how they will be used
- C/C++ compiler uses the instructions automatically
  - No need to change the C/C++ code
- If an instruction is too complex, then:
  - Intrinsic is automatically generated
  - User may use the instruction via the intrinsic or inline assembly

Not Just Vanilla LLVM Compiler

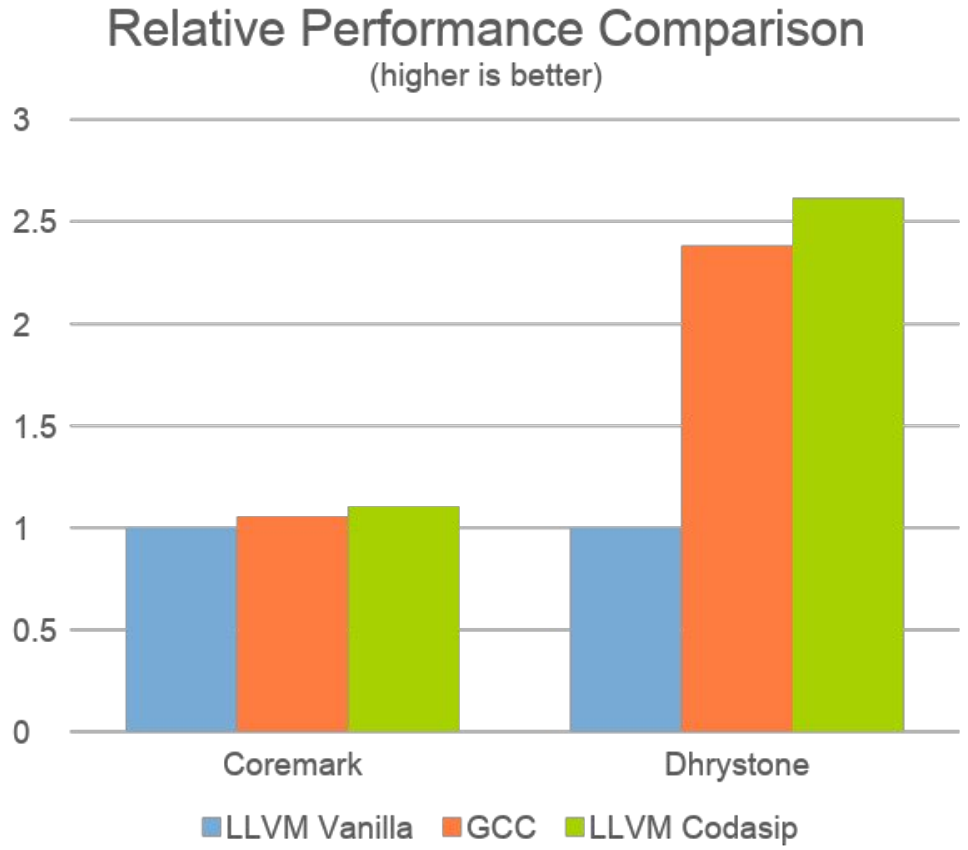
- Codasip LLVM Compiler is enhanced compiler
- LLVM is enhanced to provide
  - Better performance
  - Smaller code
  - DSP features that are missing in vanilla LLVM (e.g., zero overhead loops)
  - Etc.
- LLVM may be tuned using custom LLVM files
  - See the next slide
- Performance enhancing features
  - Improved jump threading
  - Superblock scheduling
  - Loop collapsing/flattening
- Code size lowering features
  - -msave-restore
  - Function merging by sequence alignment
  - Machine outliner
- General features
  - Zero overhead loops
  - Dual-stack architecture support
  - And many more

LLVM Backend is Open

- Studio generates LLVM compiler backend
  - Frontend and optimizer are parametrized
- Generated files are available to designers
- Most of the time, there is no reason to touch it, but ...
- … in some cases, designers want to have a fine control and integrate what they have already done
  - These aspects might be important to the C/C++ compiler
  - Designers may change the generated backend is several ways
    - Add specific optimization passes
    - Change the generated files
    - Override generated C++ methods

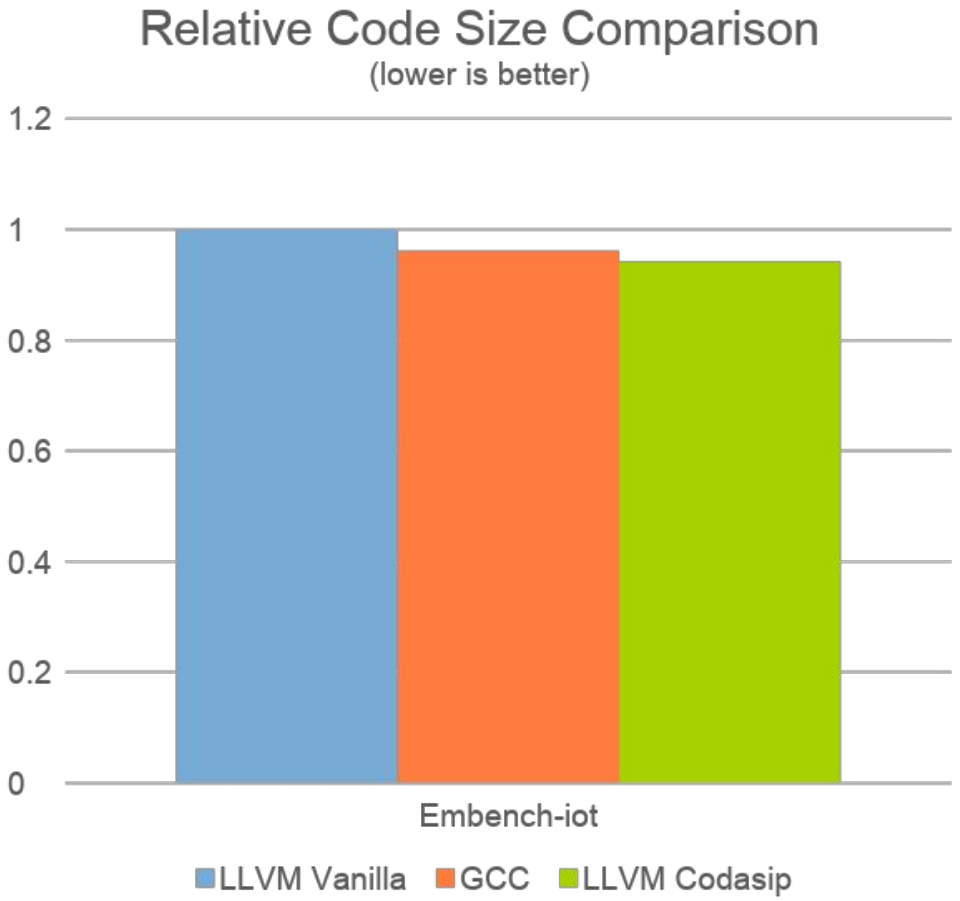
Benchmarking – Performance

- LLVM Vanilla as a base line
  - GCC as well as Codasip LLVM is then compared relatively to it
- Coremark compiled for performance
- Dhrystone is compiled using legal arguments only
- Codasip outperforms both, GCC as well as LLVM Vanilla
  - Can be improved further using custom compute with Codasip Studio



Benchmarking – Code Size

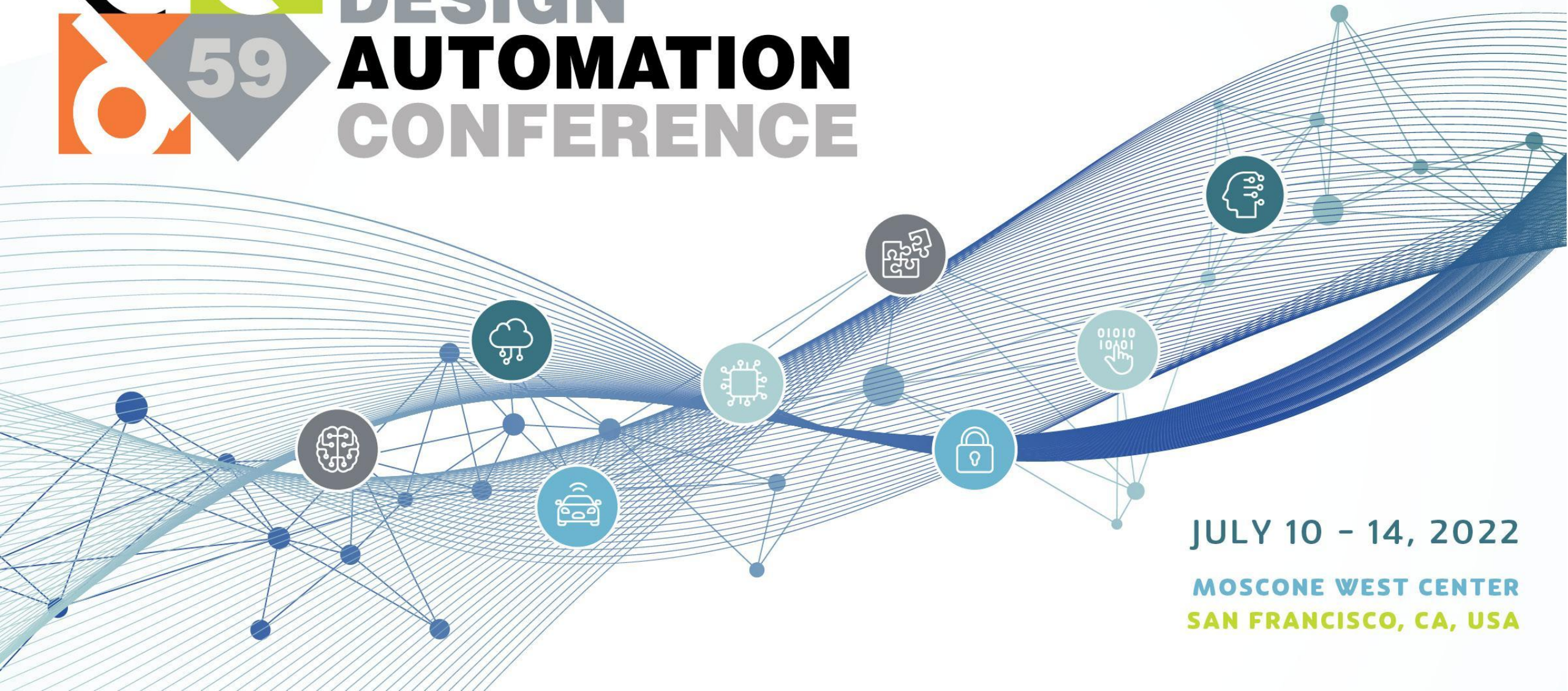
- LLVM Vanilla as a base line
  - GCC as well as Codasip LLVM is then compared relatively to it
- Embench-iot used as a benchmark
- The same optimization options are used for the compilation across compilers
- Codasip produces smaller applications
  - Can be improved further using custom compute with Codasip Studio



Conclusion

- RISC-V open standard enables innovations and differentiation through customization
- Codasip Studio automates this process
  - Processor is described in CodAL, then the whole processor IP is generated
    - SDK (LLVM toolchain, executable models, profilers, debuggers, …)
    - HDK (RTL, testbenches, EDA scripts, …)
- Re-targetable LLVM compiler is aware of differentiation
  - It can use any instruction in the design automatically
  - Users may add their own optimizations too (either through CodAL or LLVM)
  - Other parts of LLVM, such as LLDB, are aware of differentiation too

59 DESIGN AUTOMATION CONFERENCE



JULY 10 - 14, 2022

MOSCONE WEST CENTER

SAN FRANCISCO, CA, USA