Towards Automated LLVM Support and Autovectorization for RISC-V ISA Extensions

Philipp van Kempen, Mathis Salmen, Daniel Mueller-Gritschneider, Ulf Schlichtmann

Problem Statement

Motivation

- Exploring new RISC-V instruction for emerging workloads (automatically)
- Extending RISC-V with custom instructions needs many steps and can be time-consuming and error prone
- Eliminate manual efforts in Compiler Retargeting to enable ISA DSE

Compiler Retargeting

- The process of supporting new computing platforms in software development tools
- Support Levels:
  1. Assembler: Minimal support for custom instructions (ASM only)
  2. Intrinsics/Builtins: Manual insertion of custom instruction in high-level languages
  3. CodeGen: Pattern based instruction selection based on DAG
- Autovectorization: Automatic SIMD support (Loop-level & Basic-block)

Contributions

- Fully-automated model-based code generation for bulltins and assembly-level
- Semi-automated pattern generator for scalar and SIMD ALU-type instructions
- Autovecstoration support for “narrow” (sub-word) SIMD instructions GPRs

Methodology

Inputs

A. CoreDSL Code

- High-level Language to describe RISC-V Cores & their ISA
- Encoding, Assembly Syntax & Semantics
B. YAML Settings

- Configure Seal5 tools, passes, filters, logging,...
C. Test Sources

- Hand-written Assembly/Codegen tests

Evaluation

- Core-V
  - Core-V Extension (implemented on CV32E40P [3])
  - 300+ MAC/Mem/ALU/Bitmanip/SIMD instructions
- Configurations
  - Baseline LLVM: Upstream LLVM 17 ([U2139])
  - Core-V Reference LLVM: Developed by OpenHWGroup community
    ([U2139,XCVU2a,XCVU2a,XCVU2a])
  - Seal5 generated LLVM ([U2139,XCVU2a,XCVU2a,XCVU2a]([XCVU2a])
- Benchmarks: 100+ programs (MLPerfTiny, Embench, TACLeBench, TACLeBench, Coremark)
- Results: Without SIMD, Seal5 performs similar to Reference LLVM. With SIMD-support Seal5 outperforms Reference LLVM drastically.

Roadmap

Recent Additions

- Migration from ISelDAG to GlobalISel
- Support compressed instructions, 64-bit targets and custom registers

Work in Progress

- Generation of test cases
- Support register-pairs (→ LLVM Support for RISC-V Packed Extension)

Planned Features

- Support more datatypes: float32, float16, int4,...
- uArch-aware scheduling, ...

Methodology

Flow

- Initial Patch Generation
  - Generates Tablegen and C++ artifacts for:
    - Assembly-level support
    - Builtins/Intrinsics
    - Legalization rules
    - Heuristics and cost functions

Extraction of Code-Generation Patterns

- How to avoid manual definition of ISel patterns in LLVM?
  1. Convert CoreDSL behavior to LLVM-IR functions
  2. Perform lowering in a similar way to target SW
  3. Add hook to emit final DAG right before instruction Selection would take place
  4. Transform DAG nodes to TableGen code for patterns

Autovecstoration support

- While CoreDSL has no notion of vectors, LLVM can detect SIMD instructions automatically to generate patterns using vector-types
- Challenge: Adjusting compiler heuristics and cost functions of existing autovectorizers to generate efficient code for “narrow” SIMD

Getting Started

1. Installation pip install seal5
2. Running Examples python3 examples/demo.py
3. Read Documentation https://seal5.readthedocs.io

Usage

Python API

```
small_flow = small_flow("init-project")
small_flow.initialize...
small_flow.set...(variable=...)  
small_flow.execute(...)  
small_flow.execute_instruction(...)
```

Command-Line Interface

```
small init init-project
small set small
small load small ...
small dump small ...
small run small ...
small clean small ...
```

References


Contact:
philipp.van-kempen.de
mathis.salmen@tum.de
daniel.mueller-gritschneider@tum.de

Open Source:
https://github.com/tum-ai-edu/seal5

Technical University of Munich
School of Computation, Information and Technology
Chair of Electronic Design Automation
Department of Computer Engineering