Extended Abstract: Automated Generation of a RISC-V LLVM Toolchain for Custom MACs

Philipp van Kempen[1]*, Karsten Emrich[1], Daniel Mueller-Gritschneder[1] and Ulf Schlichtmann[1]

[1]School of Computation, Information and Technology, Technical University of Munich

Abstract

RISC-V is an instruction set architecture (ISA) that, as a core feature, can be extended with special instructions to customize embedded processors to special applications such as from the control and machine learning domain. There exist several instruction set simulators (ISS), that can quickly evaluate the benefit of special instructions for a given application. Next to the core, also the compiler and assembler support for creating a binary from embedded C code is required by designers to exploit performance benefits of special instructions such as Multiply and Accumulate (MAC) operations. We introduce a code generation tool for extending existing LLVM implementations with support for custom RISC-V instructions described in the CoreDSL format.

Introduction

Integrating a new instruction set extension or single instructions in simulators and embedded SW toolchains such as LLVM[1] or GCC currently usually involves a lot of manual work. In the past a description language for ISA Extensions (CoreDSL[2]) has been developed. Using a tool written in the Python language tool called M2-ISA-R we are able to automatically generate a compatible architecture for the ETISS instruction set simulator [3]. M2-ISA-R follows the meta-modelling approach depicted in Figure 1 supporting several frontends/backends for conversions between different domain specific languages and tools.

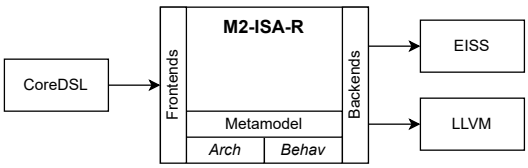

Figure 1: *Metamodel-based approach.*

In the academia as well as the industry numerous ISA extensions for embedded RISC-V targets are being developed. The MAC operation used for evaluations is part of the Xcorev (formerly XpulpNN[4]) extensions.

Methodologies

As a first step the different versions of MAC instructions included in the Core-V ISA extension shown in Table 1 have been implemented with the CoreDSL Syntax. This involves describing the instructions behavior and encoding based on the specification.

Table 1: *Core-V MAC Instructions.*

Instructions	Sign	Width	Shift	Round
MAC/MSU	S	32		
MACUN	U	16 (low)	✓	
MAC[HHUN	U	16 (high)	✓	
MACSN	S	16 (low)	✓	
MACHHSN	S	16 (high)	✓	
MACURN	U	16 (low)	✓	✓
MACHHURN	U	16 (high)	✓	✓
MACSRN	S	16 (low)	✓	✓
MACHHSRN	S	16 (high)	✓	✓

We implemented a backend for the aforementioned tool, which processes the metamodel generated from the previously written CoreDSL code. Since the instructions behavior is stored in a tree-like data structure, transformation passes for mapping certain operators and types to LLVM primitives, can be applied effortlessly. To extend the toolchain with the new instructions, Tablegen[5] code, which integrates well with the LLVM build infrastructure, is emitted automatically using Mako templates. These generated artifacts can be applied to an existing LLVM codebase in a following step. In addition to base level assembler-level (machine code) support, our tool can automatically insert simple dataflow patterns based on the instruction's behavior to allow low level optimizations such as pattern matching. Intrinsic functions are automatically added to LLVMs clang frontend, exposing the new instructions for usage in high-level C++ programs or third-party frameworks such as TVM[6].

Discussion

The implementation was evaluated on the Xcorevmac subset for the custom Core-V instruction set extension

*Corresponding author: philipp.van-kempen@tum.de

This work has been developed in the project MANNHEIM-FlexKI funded by the German Federal Ministry of Education and Research (BMBF) under contract no.01IS22086L.

Table 2: *Utilized MAC Instructions.*

Instruction	Used
`MAC`	✓
`MSU`	✓
`MACUN/MACHHUN`	
`MACSN/MACHHSN`	✓
`MACURN/MACHHURN`	
`MACSRN/MACHHSRN`	(✓)

used by the CV32E40P core. It provides 18 signed/unsigned multiplication instructions with optional accumulation and shift/rounding options for 32 bit (full-word) and 16 bit (half-word) data. However only the 10 actual Multiply and Accumulate instructions listed in Table 1 are discussed in the following. In addition to the LLVM compiler, an instruction-accurate (CPI=1) simulation architecture modelling a RV32GC core with MAC instructions was generated.

An evaluation to analyze the impact of the automatically inserted instructions was performed using the MLPerf Tiny[7] benchmark suite and the TVM machine learning compilation framework[6]. A reduction in the number of executed instructions is expected due to the MAC instruction effectively replacing the very common MUL & ADD pattern with a single instruction.

The actually utilized instructions during the benchmarks are given in Table 2. None of the unsigned MAC instructions are inserted, which is expected for `int8` quantized networks. Since TVM mainly performs 16 bit multiplications, half words fetched from memory, need to be sign extended first, to utilize the default 32 bit MAC instruction. This can be avoided by using the 16 bit MACSN and MACHHSN instructions, operating on lower or upper elements packed into a single 32 bit word, which also allows fetching two elements from memory in a single instruction.

The MACSRN and MACHHSRN instructions are suitable to perform 16 bit fixed point multiplications with rounding which is a common operation in quantized neural networks. However since TVM uses a `q31` fixed-point representation, these instructions can not be used without making changes to TVM's quantization parameters, sacrificing accuracy while achieving only neglectable improvements in performance.

The benchmark results are shown Figure 2 for the 4 quantized MLPerfTiny models. The unoptimzed benchmarks use TVMs default schedules and a channels-last (NHWC) data layout while the optimized variants are using tuned generic schedules and a channels-first (NCHW) layout. For both dense and convolutional neural networks a reduction in the number of executed instructions of about 15 % (unoptimized) to 30 % (optimized) was observed.

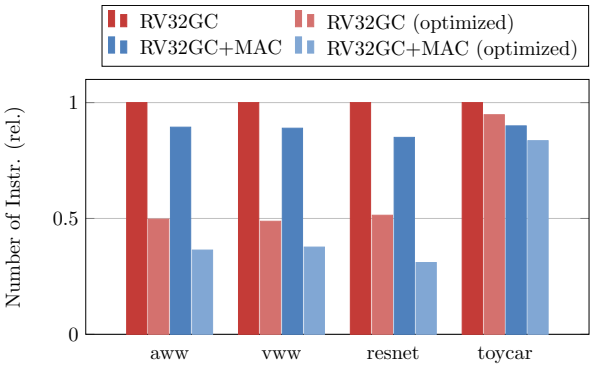

Figure 2: *Benchmark Results for 3 CNNs and 1 DNN*

Outlook

The automatic generation of a LLVM toolchain for RISC-V instruction set extensions was demonstrated using the Multiply-Accumulate operation. Compared to the manually written reference implementation, a patch of roughly the same size and equivalent performance is generated effortlessly. In particular, it can save a lot of time and human efforts which is a crucial aspect for workflows in the modern Electronic Design Automation space.

Work is ongoing to support more types of instructions such data parallel (SIMD) operations which should speed up modern Embedded ML workloads. Likewise, it is essential to also model the micro architecture and memory hierarchy of the evaluated target platform to estimate the cycle-accurate runtime, which will be made possible in the near future.

References

[1] *The LLVM Compiler Infrastructure*. URL: `http://llvm.org/`.

[2] Wolfgang Ecker et al. "The Scale4Edge RISC-V Ecosystem". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 808–813.

[3] Daniel Mueller-Gritschneder et al. "The extendable translating instruction set simulator (ETISS) interlinked with an MDA framework for fast RISC prototyping". In: *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. 2017, pp. 79–84.

[4] Angelo Garofalo et al. "XpulpNN: Accelerating quantized neural networks on RISC-V processors through ISA extensions". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 186–191.

[5] *Tablegen Fundamentals*. URL: `http://llvm.org/docs/TableGenFundamentals.html`.

[6] Tianqi Chen et al. "TVM: An automated end-to-end optimizing compiler for deep learning". In: *arXiv preprint arXiv:1802.04799* (2018).

[7] Colby Banbury et al. "MLPerf Tiny Benchmark". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).