Leveraging TVM to optimize AI models for custom HW Accelerators and RISC-V extended instructions

Alexander Belke*, Gilles Miet**
Robert Bosch Mobility Electronics, Engineering Integrated Circuits *GmbH (Germany), **SAS (France)

Introduction

HW manufacturers development flow targets product that are:

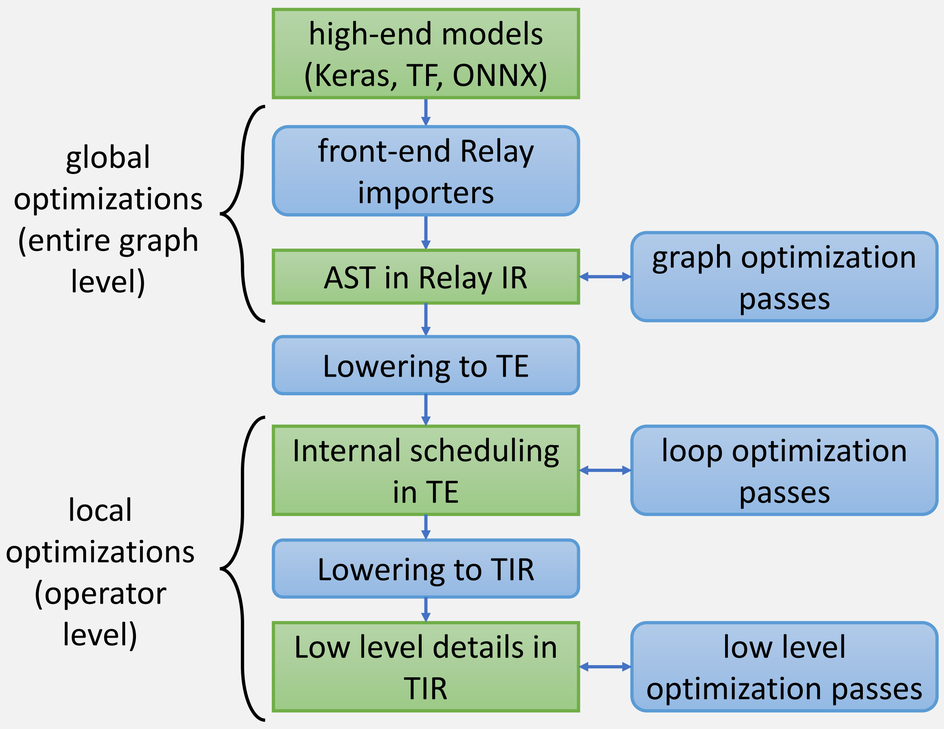
- Efficient in term of time to market
- Flexible for easy upgrade
- Low cost with a silicon area as small as possible
- Low power via optimal choice and usage of processing resources

We choose and customized Apache TVM as HW/SW co-design framework enabling to generate and test the software regardless of the HW availability.

TVM overview

TVM enables to import, optimize, compile, and run a high-level defined Deep Learning network model; it is mainly composed of:

- A Deep Learning model Relay importer
- A graph optimizer in Relay language (Abstract Syntax Tree)
- An operator internal scheduler in TE language
- An operator low-level optimizer in TIR language



TVM customization

Processing simulation model

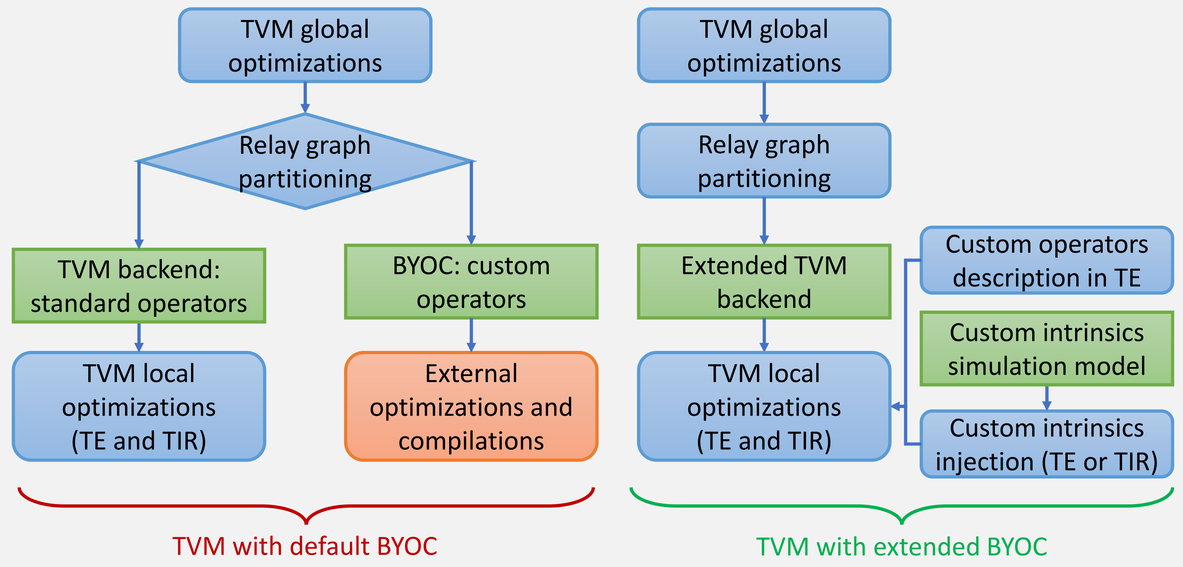
Creation of C++ components to simulate the HWA and the RISC-V extended instructions. Compiled and run in the context of TVM or outside TVM in standalone environments.

Hybrid methodology for int8 Quantization

We are using QNN operators for the Relay backend part by adding Relay passes that transform standard float32 NN Relay graph into a quantized version. QNN relay expressions are inserted, and operators are replaced with their quantized variants.

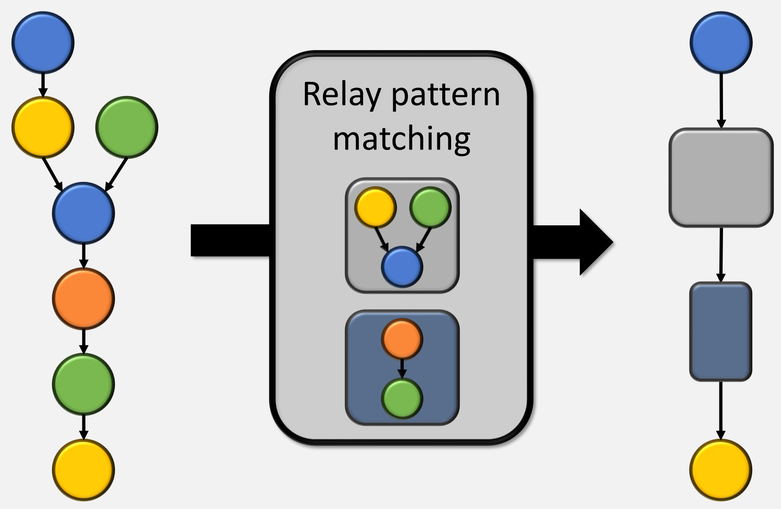
BYOC concept extended

With Bring Your Own Codegen, custom HW components programming is left to proprietary toolchain and TVM optimization of the related code stops at Relay level.
We modified BYOC to handle offloaded operations as a TVM TE and TIR language representation enabling TVM to be used for schedule definition and optimization for custom hardware (via Autotune to optimize tiling).



RISCV customized instruction integration into TVM

After relay graph partitioning, the TE schedule of the offloaded operators to custom RISCV compute lib is created.




```
%22 = strided_slice(%21, begin=[0], /*…*/)
%23 = reshape(%22, newshape=[4, 4])
%24 = cast(%23, dtype="int32")
%25 = subtract(%24, 0.5f)
%26 = cast(%25, dtype="float32")
%27 = multiply(%26, 0.0625f)
nn.softmax(%27)
```

```
%22 = strided_slice(%21, begin=[0], /*…*/)
%23 = reshape(%22, newshape=[4, 4])
@tvmgen_default_tristan_compute_lib_main_0(%2)
```

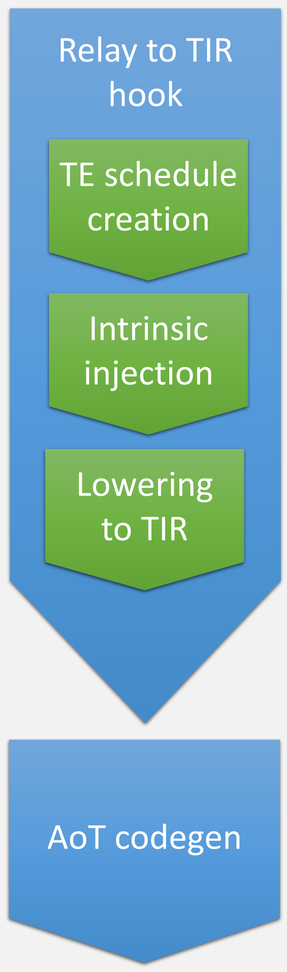
```
def _create_schedule(self, cfg=None):

    # In case of mode profile, inject intrinsic as external
    if self.config['tvm']['execution_mode']=='profile' or sel
        logger.debug('>>>>>AOT Fixed intrinsic injection<<<<<
        res_outInt8 = te.compute(self.ifm.shape, lambda *inde
        res_out = te.compute(res_outInt8.shape, lambda *index
        s = te.create_schedule([res_out.op])

        def intrin_softmax(l):
            a = te.placeholder((l,), "int8", name="a")
            c = te.compute((l,), lambda i: tvm.tir.Cast("uint
            Ab = tvm.tir.decl_buffer(a.shape, a.dtype, name="
            Cb = tvm.tir.decl_buffer(c.shape, "uint8", name="

            def intrin_func(ins, outs):
                ib = tvm.tir.ir_builder.create()
                aa = ins[0]
                cc = outs[0]
                ib.emit(
                    tvm.tir.call_extern(
                        "uint8",
                        "riscv_softmax",
                        aa.access_prt("r"),
                        cc.access_prt("w"),
                        l,
                    )
                )
                return ib.get()

            return te.decl_tensor_intrin(c.op, intrin_func,
```




```
tvmgen_default_fused_strided_slice_reshape(/*…*/);
tvmgen_default_tristan_compute_lib_main_0(/*…*/);

// with:

tvmgen_default_tristan_compute_lib_main_0(/*…*/) {
    // …
    riscv_softmax(/*…*/);
    // …
}
```

Conclusion

We leverage TVM to integrate a DL NN model in a heterogeneous system (RISC-V CPU with custom extended instructions and HWA). There we can validate, prepare, analyze, and optimize FW w/o the need for HW availability.

TRISTAN Project has received funding from the Chips Joint Undertaking (Chips-JU) under the grant agreement nr. 101095947. Chips-JU receives support from the European Union's Horizon Europe's research and innovation programme and Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia and Turkey.