OctoML

µTVM: Running the TVM Stack on Bare Metal
TVM Conf 2020

Andrew Reusch

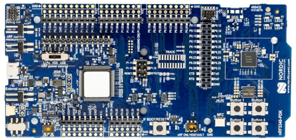
Outline

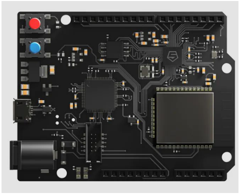
- What is µTVM?
- How µTVM Works
- Demo Walkthrough
- Future Directions
- Q&A

What is µTVM?

using TVM
to run models
on bare-metal devices

libmodel.a





...Bare Metal?

To µTVM, bare metal is not just:

🚀 Raspberry Pi
- These (usually) have operating systems

🚀 Reserved Cloud Instances
- These still have Virtual Memory

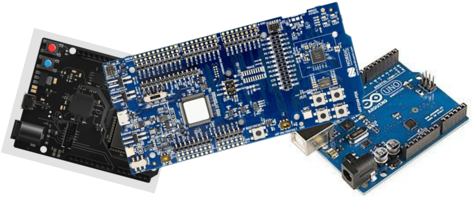
🚀 Running outside a VM
- Traditional TVM can run here

...Bare Metal?

Bare metal is (often) IoT-class devices

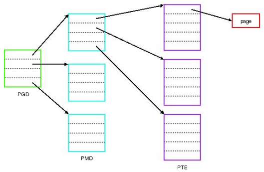
- ⚡ AzureSphere
- ⚡ Arduino
- ⚡ Cortex-M class micro-controllers

µTVM works in places without...



🚫 Operating Systems
- no files, DLLs, .so, memory mapping, kernels

🚫 Virtual Memory
- No malloc, C++ RAII, exceptions, …



🚫 Advanced Programming Languages
- No C++, Rust, Python, …
  (But we like those and you could use them!)

The deployment challenge

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

The deployment challenge

frameworks										
PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

The deployment challenge

targets →

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

The deployment challenge

targets →										
PyTorch	✓	?	?	✓	?	?	?	?	?	?
Caffe2	✓	?	?	✓	?	?	?	?	?	?
TensorFlow	✓	?	?	✓	?	?	?	?	?	?
mxnet	✓	?	?	✓	?	?	?	?	?	?
ONNX	✓	?	?	✓	?	?	?	?	?	?
TensorFlow Lite	✓	?	?	✓	?	?	?	?	?	?

The deployment challenge

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	✓	?	?	?	?	?	?	?	?

The deployment challenge

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	✗	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

Bare Metal Deployment Challenges

- Less abstraction than full OS
    - Less tools to work with
- Resources are tighter
    - Scheduling is harder
- Demands are unique per-chip and per-project
    - Code reuse is tricky

The µTVM Approach

- Batteries Included
  - µTVM can be used with only the standard C library
- Compute-centric
  - µTVM does not configure the SoC--it only runs computations
  - µTVM integrates with RTOS like Zephyr and mBED for SoC configuration
- Transparent
  - µTVM binaries can be compiled directly from source

How µTVM Works

tvm

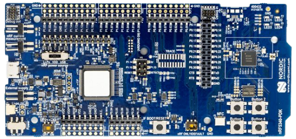
```
int32_t fused_conv2d_right_shift_add() {
  // ...
}
```

How µTVM Works

```
int main() {
  // configure SoC
  TVMInitializeRuntime();
  TVMGraphRuntime_Run();
}
```

+

```
int32_t fused_conv2d_right_shift_add() {
  // ...
}
```



Working with the TVM Compiler

Model import

Relay Module

```
#[version = "0.0.5"]
def @main(%data : Tensor[(1, 3, 64, 64), int8],
          %weight : Tensor[(8, 3, 5, 5), int8]) {
    %1 = nn.conv2d(
         %data,
         %weight,
         padding=[2, 2],
         channels=8,
         kernel_size=[5, 5],
         data_layout="NCHW",
         kernel_layout="OIHW",
         out_dtype="int32");
  %3 = right_shift(%1, 9);
  %4 = cast(%3, dtype="int8");
  %4
}
```

Working with the TVM Compiler

Model import → Optimize Operators (AutoTVM)

Relay Module

```
#[version = "0.0.5"]
def @main(%data : Tensor[(1, 3, 64, 64), int8],
          %weight : Tensor[(8, 3, 5, 5), int8]) {
    %1 = nn.conv2d(
         %data,
         %weight,
         padding=[2, 2],
         channels=8,
         kernel_size=[5, 5],
         data_layout="NCHW",
         kernel_layout="OIHW",
         out_dtype="int32");
  %3 = right_shift(%1, 9);
  %4 = cast(%3, dtype="int8");
  %4
}
```

TensorIR

```
 primfn(placeholder_2: handle,
        placeholder_3: handle,
        T_cast_1: handle) -> ()
  allocate(kernel_vec, int8, [600]) {
    for (bs.c.fused.h.fused: int32, 0, 64)
"parallel" {
      for (w: int32, 0, 64) {
        for (vc: int32, 0, 3) {
          data_vec[(((bs.c.fused.h.fused*192) +
(w*3)) + vc)] =
(uint8*)placeholder_5[(((vc*4096) +
(bs.c.fused.h.fused*64)) + w)]
        }
      }
    }
    // ...
```

Working with the TVM Compiler

Model import → Optimize Operators (AutoTVM) → Generate C/LLVM library

Relay Module

```
#[version = "0.0.5"]
def @main(%data : Tensor[(1, 3, 64, 64), int8],
          %weight : Tensor[(8, 3, 5, 5), int8]) {
    %1 = nn.conv2d(
         %data,
         %weight,
         padding=[2, 2],
         channels=8,
         kernel_size=[5, 5],
         data_layout="NCHW",
         kernel_layout="OIHW",
         out_dtype="int32");
  %3 = right_shift(%1, 9);
  %4 = cast(%3, dtype="int8");
  %4
}
```

TensorIR

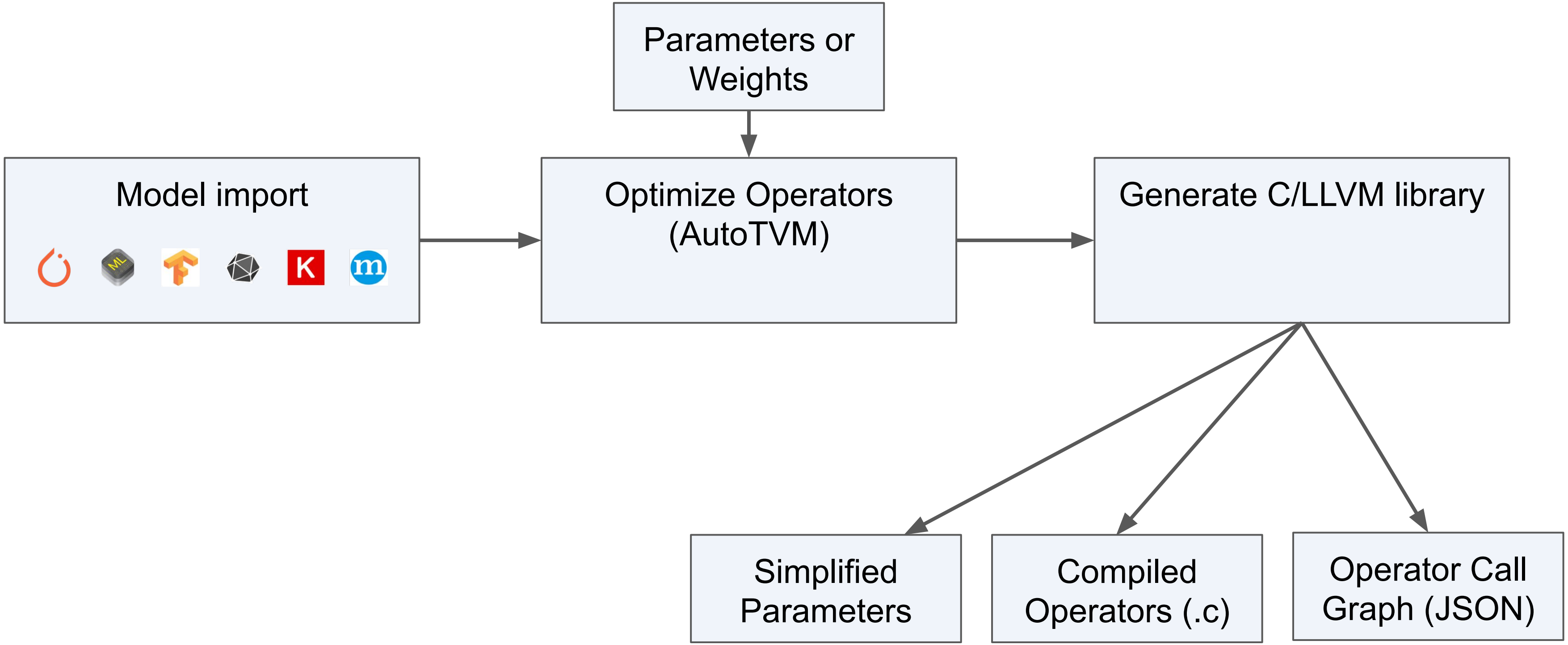
```
 primfn(placeholder_2: handle,
        placeholder_3: handle,
        T_cast_1: handle) -> () {
  allocate(kernel_vec, int8, [600]) {
    for (bs.c.fused.h.fused: int32, 0, 64)
"parallel" {
      for (w: int32, 0, 64) {
        for (vc: int32, 0, 3) {
          data_vec[(((bs.c.fused.h.fused*192) +
(w*3)) + vc)] =
(uint8*)placeholder_5[(((vc*4096) +
(bs.c.fused.h.fused*64)) + w)]
        }
      }
    }
    // ...
```

C Source Code

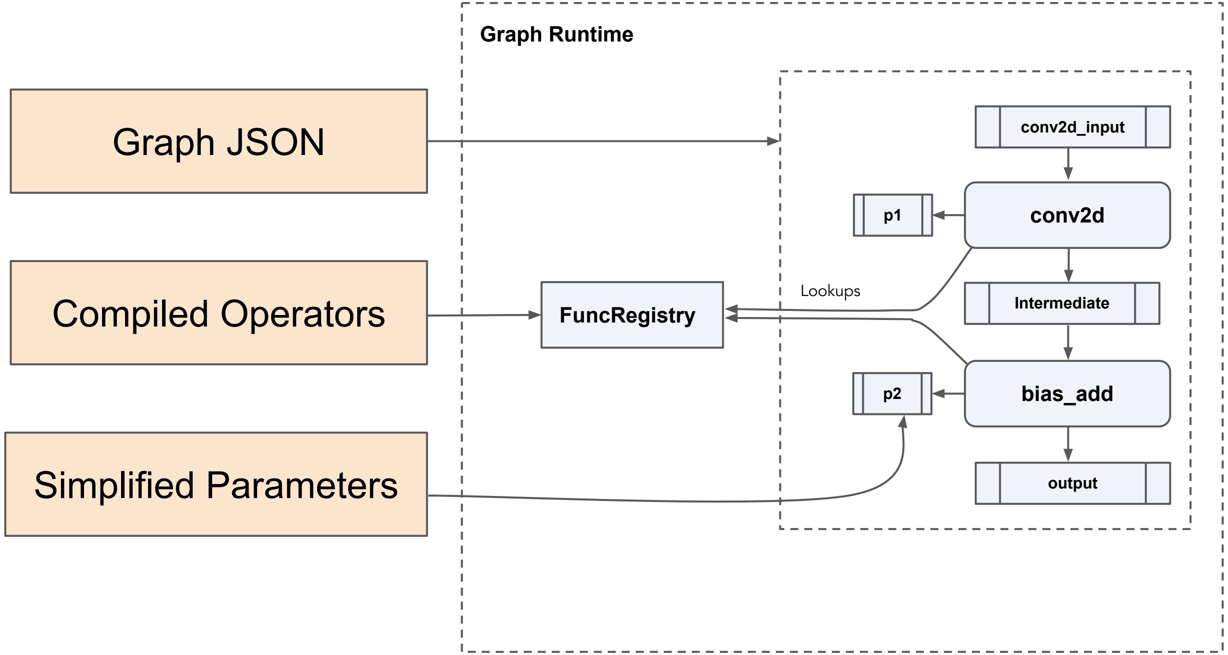
```
int32_t
fused_nn_contrib_conv2d_NCHWc_right_shift_cast(
void* args, void* arg_type_ids,
int32_t num_args, void* out_ret_value,
void* out_ret_tcode, void* resource_handle) {
  void* data_pad = TVMBackendAllocWorkspace(1,
dev_id, (uint64_t)13872, 1, 8);
  for (int32_t i0_i1_fused_i2_fused = 0;
i0_i1_fused_i2_fused < 68;
++i0_i1_fused_i2_fused) {
    for (int32_t i3 = 0; i3 < 68; ++i3) {
      for (int32_t i4 = 0; i4 < 3; ++i4) {

((uint8_t*)data_pad)[((((i0_i1_fused_i2_fused *
204) + (i3 * 3)) + i4))] = (((((2 <=
i0_i1_fused_i2_fused) && (i0_i1_fused_i2_fused
< 66)) && (2 <= i3)) && (i3 < 66)) ?
((uint8_t*)placeholder)[((((((i0_i1_fused_i2_fus
ed * 192) + (i3 * 3)) + i4) - 390))] :
(uint8_t)0);
```

OctoML

Working with the TVM Compiler



Running the model end-to-end



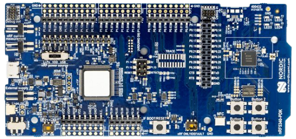
How µTVM Works

tvm

```
int main() {
  // configure SoC
  TVMInitializeRuntime();
  TVMGraphRuntime_Run();
}
```

+

```
int32_t fused_conv2d_right_shift_add() {
  // ...
}
```



TVM Compiler Output

Putting the pieces together

TVM Compiler Outputs

Simplified Parameters

Compiled Operators

Graph JSON

Graph Runtime — Library, from TVM

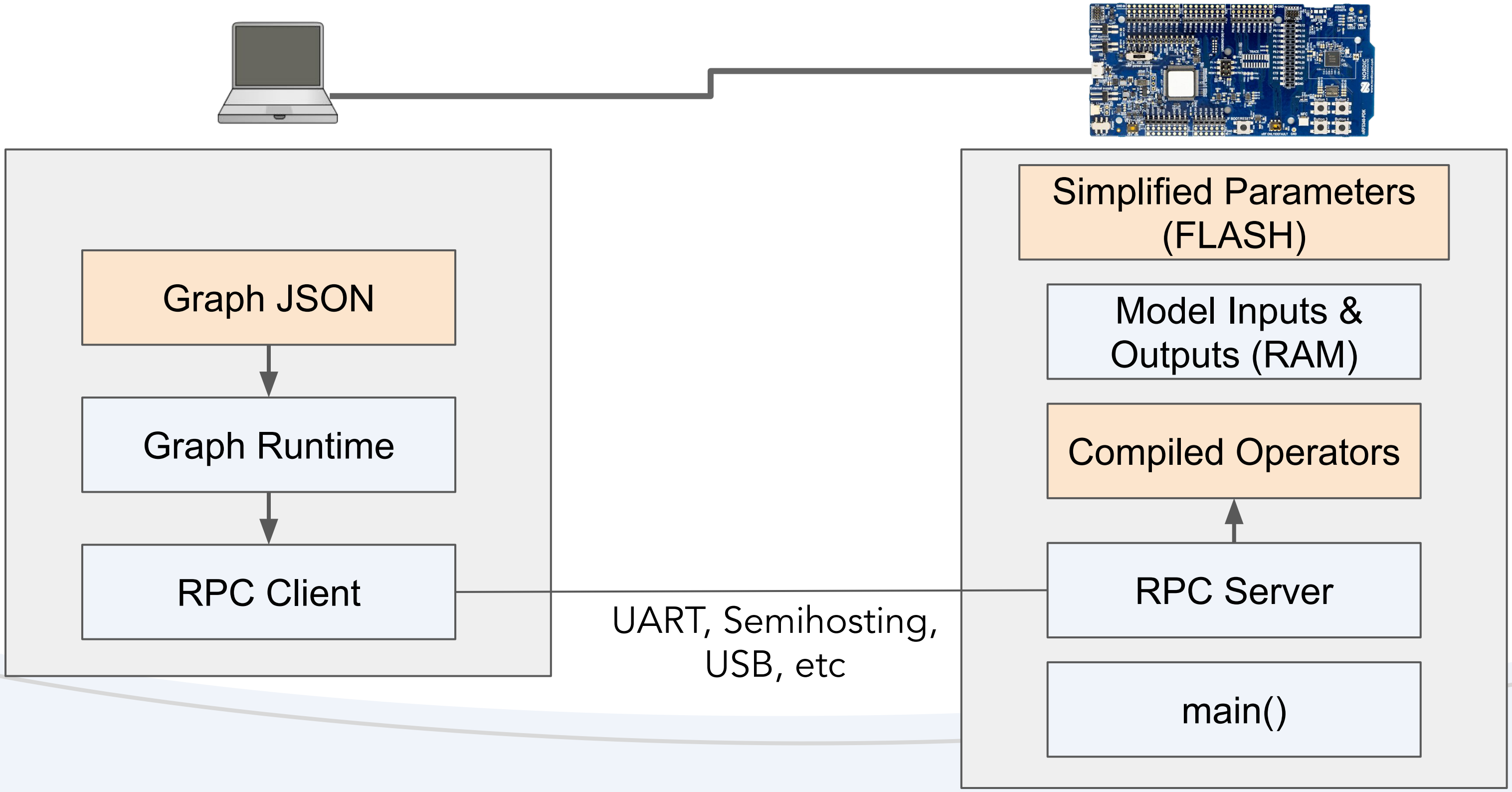
RPC Client/Server — Library, from TVM

Model Inputs & Outputs (RAM) — Data, from user

main() — SoC config & startup libraries, from RTOS/vendor

TVM Compiler Output

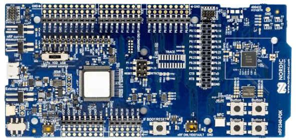
Putting the pieces together - host-driven



OctoML

TVM Compiler Output

Putting the pieces together - standalone



Simplified Parameters (FLASH)

Inputs (RAM)

Compiled Operators

Graph Runtime

Graph JSON

main()

microTVM Reference Virtual Machine

- Lots of moving pieces…
  - Physical hardware
  - TVM compiler
  - GCC, LLVM, etc
  - RTOS (Zephyr, mBED), library code
  - SoC configuration / main()

- How can we collaborate?
  - Use a “Reference VM” to freeze as much of the software as possible
  - Attach hardware to VM with USB passthrough
  - See MicroTVM Reference VM Tutorial for more

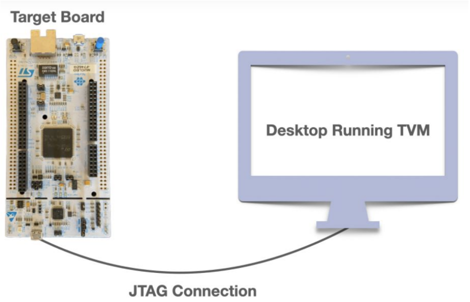
Demo Walkthrough

OctoML

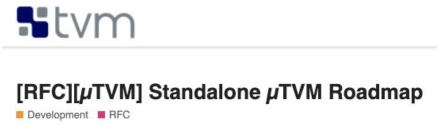
Future Directions

OctoML

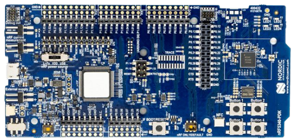
μTVM in 2020



Experimental μTVM



Blog post + roadmap



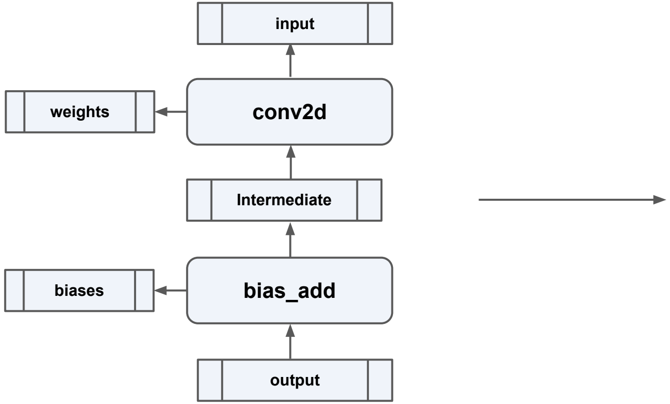
Standalone μTVM

April

June

Dec

Next for µTVM: Ahead-of-Time Compiler

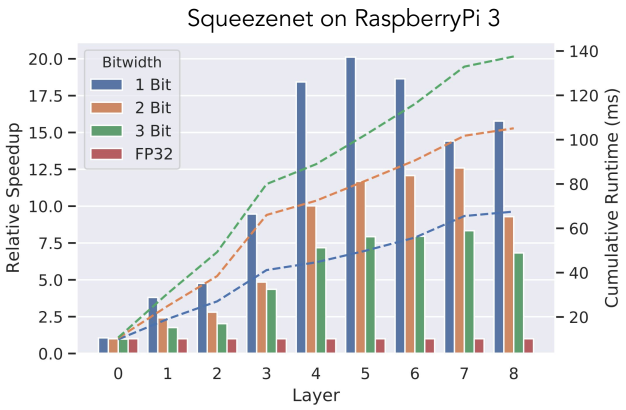



```
const DLTensor weights = {1, 2, ...};
const DLTensor biases = {4, 2, 7, ...};

int32_t classifier(DLTensor* input,
                   DLTensor* output) {
  DLTensor* intermediate =
     TVMBackendAllocWorkspace(512);

  conv2d(input, &weights, intermediate);
  bias_add(intermediate, &biases, output);

  TVMBackendFreeWorkspace(intermediate);
  return rv;
}
```

NOTE: this is a sample from the AOT compiler in development--expect API changes and an RFC.

Next for µTVM: Hardware-aware Quantization

- Data-aware quantization v2
  - Allows quantizing more networks from within TVM
- Ultra-low-bit-width quantization
  - Could reduce the overall model memory footprint
- See [HAGO PR]() and Ziheng Jiang's talk "Hardware-aware Quantization in TVM" on Dec 4



Riptide: Fast End-to-End Binarized Neural Networks. MLSys 2020 (March 3rd)

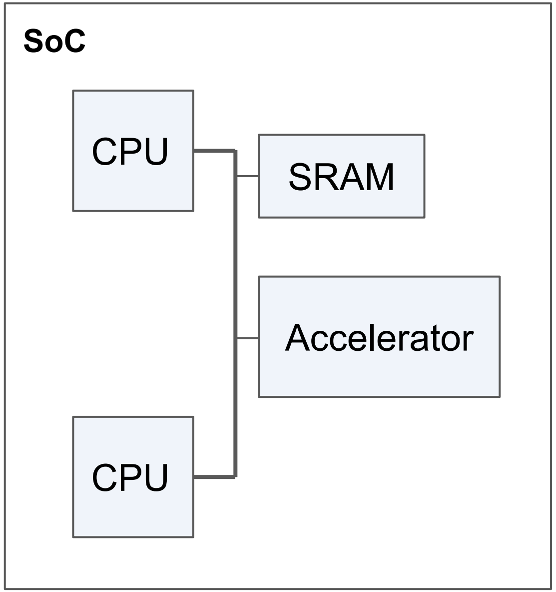
Joshua Fromm · Meghan Cowan · Matthai Philipose · Luis Ceze · Shwetak Patel

Next for µTVM: Heterogeneous Execution

- Hardware acceleration offers:
  - Lower power
  - Better performance
  - More parallelism

- Potential TVM improvements:
  - CRT multi-context execution
  - TIR subgraph offloading

- See ARM's lighting talk:
  "Ethos-U55 : microNPU Support for uTVM"
  by Manupa Karunaratne



OctoML

Next for µTVM: Memory Planning

- Current memory planner has limitations:
  - Unaware of device memory layout
  - Requires a heap-based memory allocator
- New directions:
  - Tensor pinning
  - Accelerator-aware planning
  - Bring-your-own memory planning

Next for µTVM: Increased Coverage

- Much of the work so far has been infrastructure-focused

- Next step: increase µTVM coverage in terms of:
  - Supported ISA
  - Optimized Model Operators

- Auto-Scheduling can help

Next for µTVM: Developer Experience

- Create “getting started” experience
    - Generate e.g. Arduino, Zephyr, etc projects
- Improve TVM C runtime
    - Handle faults and report through RPC server
    - Gather runtime stats to increase visibility on-device
    - Support more complex runtime scenarios -- sensing, multitasking, etc.
- Documentation
    - Targeted to developers from multiple backgrounds -- ML, firmware, etc.
    - Add design documentation and more tutorials

Q&A

- Code at https://github.com/areusch/microtvm-blogpost-eval
- Tutorials at https://tvm.apache.org/docs/tutorials/index.html#micro-tvm