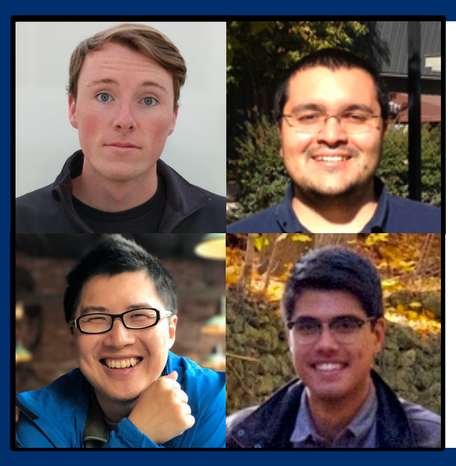


Bring Your Own Datatypes:

Enabling Datatype Exploration in Deep Learning with tvm

Gus Smith, Luis Vega, Tianqi Chen, Thierry Moreau, Luis Ceze

CRISP-T3: Scaling Applications and Making the Programmer's Life Easy

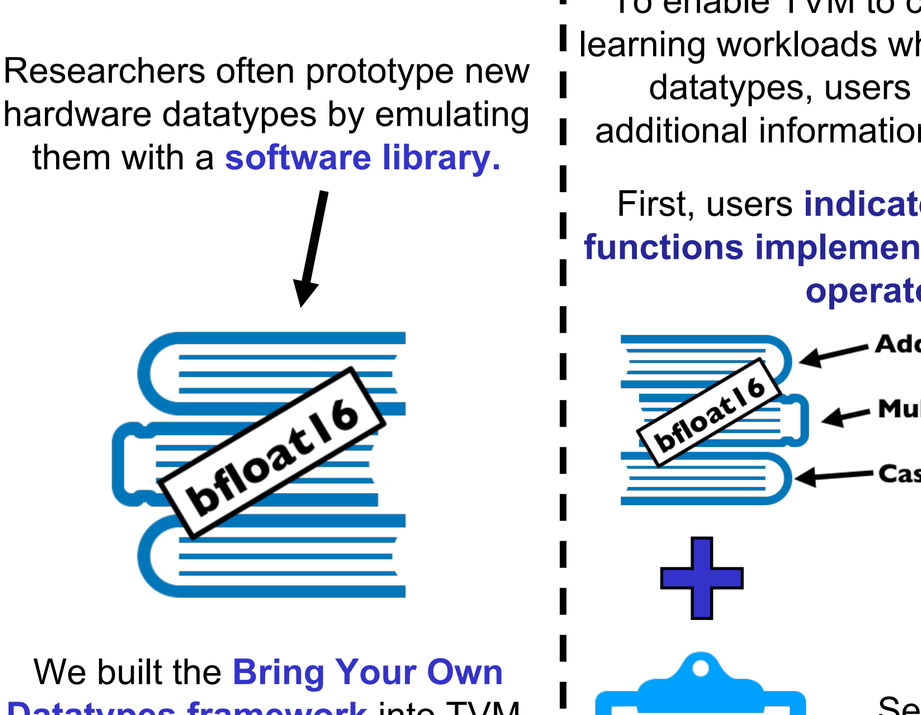
CRISP
Center for Research on Intelligent Storage and Processing in Memory

2780.014 – Programming Framework

Background

What is tvm?

An **open-source compiler for deep learning**, easily hackable for research and exploration!



Relay is TVM's high-level, differentiable intermediate representation, where we will allow users to insert custom datatypes.

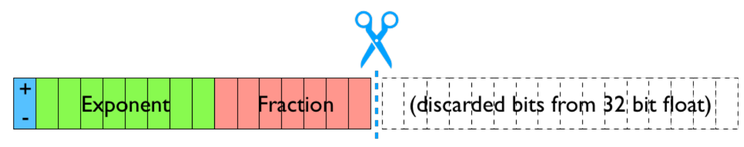
Learn more at tvm.ai!

Background

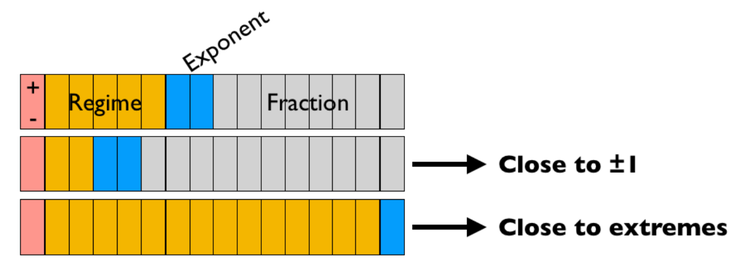
New Datatypes

To build highly optimized deep learning accelerators, architects are exploring **alternatives to IEEE 754 floating point** for representing numbers in hardware.

One example is the **bfloat16**, which saves space by dropping 16 bits from a float:



Another example is the **posit**, which uses a variable-size fraction to represent more useful numbers using fewer bits:



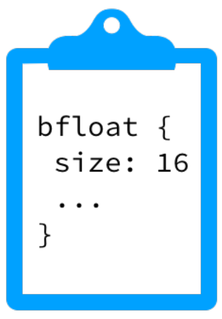
What We Did

Researchers often prototype new hardware datatypes by emulating them with a **software library.**

We built the **Bring Your Own Datatypes framework** into TVM, which allows users to compile real deep learning workloads which use these emulated datatypes.

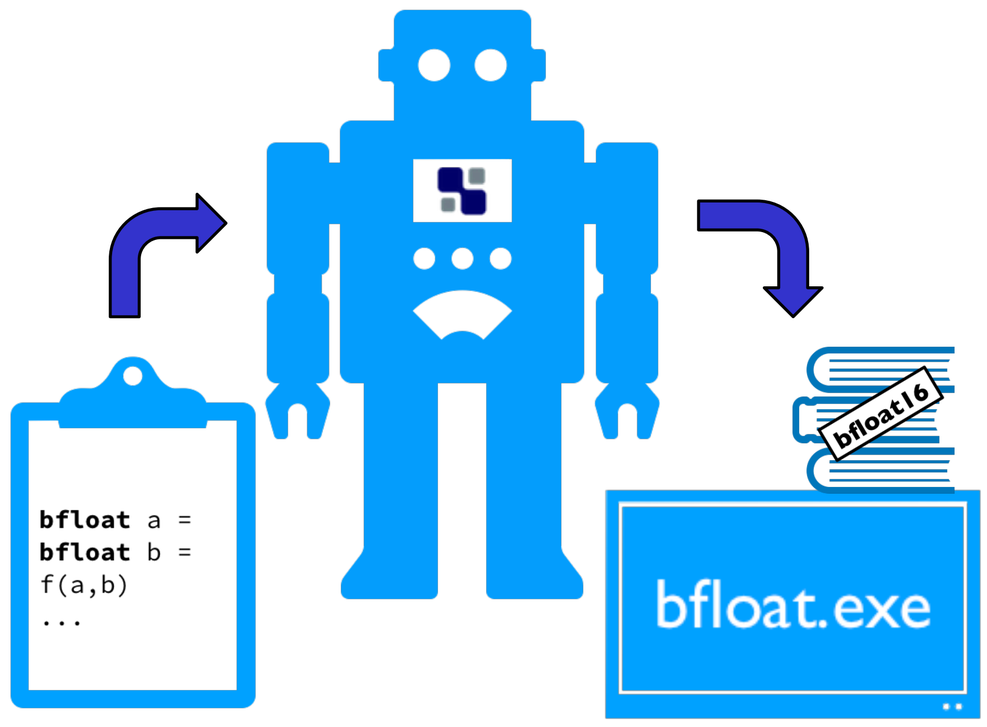
To enable TVM to compile machine learning workloads which utilize custom datatypes, users provide some additional information to the compiler:

First, users **indicate which library functions implement which datatype operators**:



Second, the user **specifies some additional information about the datatype**, such as its size.

With this minimal information, TVM can compile deep learning workloads which rely on custom datatypes!



Example

```python
import tvm
from tvm import relay
from ctypes import RTLD_GLOBAL, CDLL

# Load the bfloat library into the address space
CDLL("libbfloat16.so", RTLD_GLOBAL)

# Register the datatype with type code 131
tvm.datatype.register("bfloat", 131)

# Register the "Add" operation for bfloats.
# The second, third, and fourth arguments specify which operation we are
# registering: in this case, an Add of two bfloats when compiling for LLVM.
# The first argument tells TVM how to actually lower the bfloat adds:
# specifically, it creates a function which turns the bfloat adds into calls to
# the BFloat16Add function in libbfloat16.
tvm.datatype.register_op(tvm.datatype.create_lower_func("BFloat16Add"), "Add",
                         "llvm", "bfloat")

# Create a simple Relay program: Add two bfloats!
shape = (3, )
dtype = 'custom[bfloat]16'
a = relay.var('a', shape=shape, dtype=dtype)
b = relay.var('b', shape=shape, dtype=dtype)
expr = a + b
expr = relay.Function([a, b], expr)

a_data = ...
b_data = ...
executor = relay.create_executor()

# TVM will run the program, calling out to libbfloat16 when it encounters the
# bfloat Add!
result = executor.evaluate(expr)(a_data, b_data)
```

Future Work

What's Next?

Make it **faster**

Automate exploration of datatypes in deep learning workloads

Support actual **hardware implementations** of datatypes

For more info…

For more information, please see the links to the Bring Your Own Datatype slides, presentation, and Python notebook here:
sampl.cs.washington.edu/tvmfcrc/#program