Deep Learning Graph Compilers

linkedin.com/in/likarajo | likarajo.github.io

Motivation

Considering how Deep Learning (DL) can be useful in Artificial Intelligence, e-commerce, smart city, drug discovery etc., plenty of DL frameworks are available for use. Having huge amount of data available for gaining knowledge, DL can be used to extract the value from the data.

The DL models use features (data) of huge sizes. In order to perform DL efficiently, the process of training needs to be made faster through parallelization. CPUs and GPUs should not be the first choice for such parallelisation. In their place, specially designed accelerator chips, with custom programming language supported by libraries can be used.

However, for each accelerator/hardware back-end, the software stack to interact with the DL frameworks needs to be rebuilt separately for each hardware-framework pair. This is not a desirable technique. As a result, it is important to enable DL models generation using DL frameworks on diverse DL hardware. This can be fulfilled using DL compilers.

DL compilers take the DL models described in different DL frameworks as input, and then generate optimized codes for diverse DL hardware as output. Several DL compilers have been proposed from both industry and academia such as TensorFlow XLA and TVM.

Here, we describe how Graph compilers optimizes the Deep Neural Network graphs and then generates an optimized code for a target hardware/backend, thus accelerating the training and deployment of DL models. We specifically focus on TVM Apache.

Overview

Background

- Neural Networks as Graph
- Deep Learning
- Deep Learning Frameworks and Architecture
- CPUs and GPUs
- Accelerators
- Optimized Deep Learning Architecture Chips
- Drawback of hardware specific libraries

Deep Learning Compilers

- Deep Learning Software and Hardware Requirements
- Deep Learning Optimization Stack
- Computational Graph Optimization
- Tensor Expression Language
- Schedule Optimizations
- Tensorization Challenge
- Problem Statement: Why TVM?

Tensor Virtual Machine (TVM)

- Apache TVM
- End to End Optimization Stack
- Key Features
- Usage
- Automated End to End Optimizing Compiler

Background

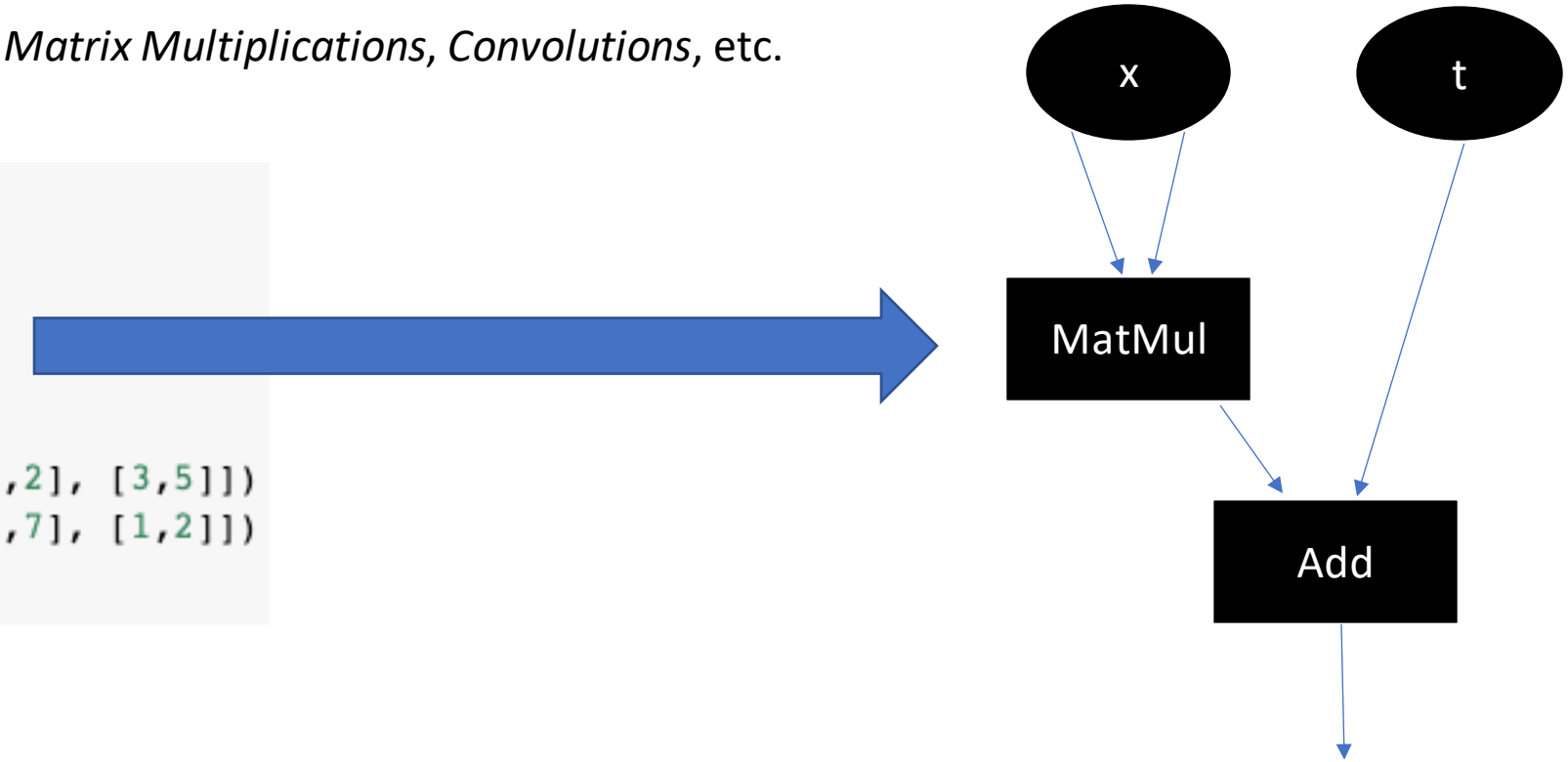
Neural Networks as Graphs

Neural Networks are ***graphs*** where each ***node*** in the graph is some mathematical operation that works on ***weights***, which are matrices.

These operations involve *Matrix Multiplications*, *Convolutions*, etc.

```
1 import torch
2
3 def foo(x, t):
4   y = x.mm(x)
5   return y + t
6
7 x = torch.Tensor([[1,2], [3,5]])
8 t = torch.Tensor([[3,7], [1,2]])
9 foo(x, t)
```

```
tensor([[10., 19.],
        [19., 33.]])
```



Deep Learning

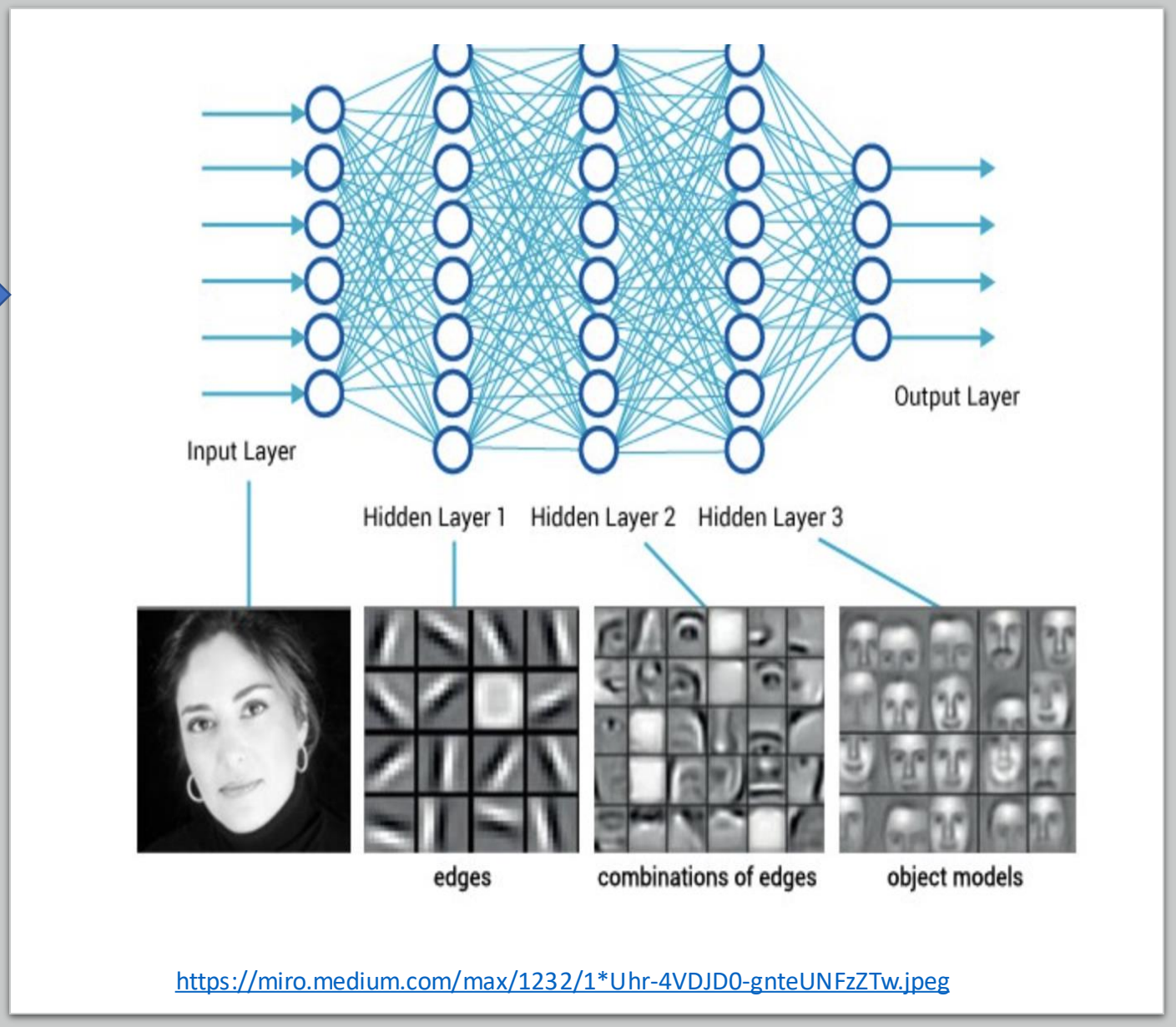
Uses of deep learning:

- Artificial intelligence
  - Natural Language Processing
  - Computer Vision
- e-commerce
- smart city
- Drug Discovery

Versatile deep learning models:

- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
- Long short-term memory (LSTM)
- Generative adversarial network (GAN)

Cannot afford to be inefficient!



https://miro.medium.com/max/1232/1*Uhr-4VDJD0-gnteUNFzZTw.jpeg

Deep Learning Frameworks and Architectures

Frameworks

- TensorFlow
- PyTorch
- MXNet
- CNTK
- ONNX

Architectures

- ResNet
- GoogleNet

Model	MACC	COMP	ADD	DIV	Activations	Params	SIZE(MB)
SimpleNet	1.9G	1.82M	1.5M	1.5M	6.38M	6.4M	24.4
SqueezeNet	861.34M	9.67M	226K	1.51M	12.58M	1.25M	4.7
Inception v4*	12.27G	21.87M	53.42M	15.09M	72.56M	42.71M	163
Inception v3*	5.72G	16.53M	25.94M	8.97M	41.33M	23.83M	91
Incep-Resv2*	13.18G	31.57M	38.81M	25.06M	117.8M	55.97M	214
ResNet-152	11.3G	22.33M	35.27M	22.03M	100.11M	60.19M	230
ResNet-50	3.87G	10.89M	16.21M	10.59M	46.72M	25.56M	97.70
AlexNet	7.27G	17.69M	4.78M	9.55M	20.81M	60.97M	217.00
GoogleNet	16.04G	161.07M	8.83M	16.64M	102.19M	7M	40
NIN	11.06G	28.93M	380K	20K	38.79M	7.6M	29
VGG16	154.7G	196.85M	10K	10K	288.03M	138.36M	512.2

Stats: https://www.researchgate.net/figure/Flops-and-Parameter-Comparison-of-Models-trained-on-ImageNet_tbl4_306376794

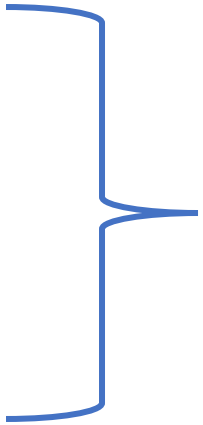
Deep Learning require a lot of compute power as they often work on large weight matrices. For example, ***ResNet50*** operates on ~100 MB of weights.

Can be made faster by **parallelizing!**

CPUs and GPUs

Tasks of CPUs and GPUs:

- Exception handling
- Dynamix dispatch using vtables
- Load port forwarding optimizations
- Branch predictions
- Out of order engines
- Caches



All these are not required to extract parallelism.

There is no need to waste power/area on useless features

Matrix Multiplications are regular and expose lots of parallelism.

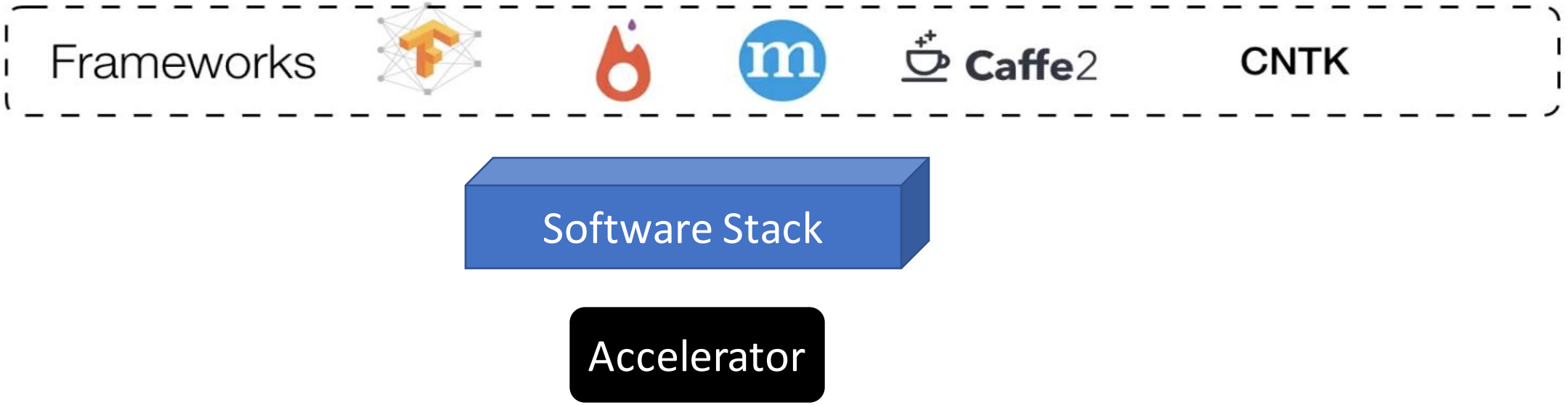
Specially designed **Accelerator Chips** can be used.

Use of CPUs and GPUs is an **overkill** for the purpose of just extracting parallelism.

Accelerators

Accelerators are more than Hardware:

- Have many arithmetic execution units
- Use dedicated local memories – Software Managed SRAMs, not caches
- Reduce the arithmetic bit-widths – work on fp16 or int8, not higher floating-point numbers



Software stack needed on top of the accelerator for it to work with different DL frameworks

Optimized Deep Learning Architecture Chips

Uses a specialized programming model – custom or domain specific language, not general-purpose programming languages.

Chip architectures

- Google TPU
- Hisilicon NPU
- Apple Bonic
- NVIDIA Turing
- Intel NNP
- Amazon Inferentia
- Alibaba Hanguang
- Cambricon
- Graphcore

Support Libraries

- Basic Linear Algebra Subprograms (BLAS): for efficient computation
  - MKL
  - MKL-DNN
  - CuBLAS
  - cuDNN
- TensorRT: supports graph optimization (e.g., layer fusion) and low-bit quantization with large collection of highly optimized GPU kernels.

Specially optimized libraries support the chips to make use of the accelerated DL models.

Drawback of hardware specific libraries

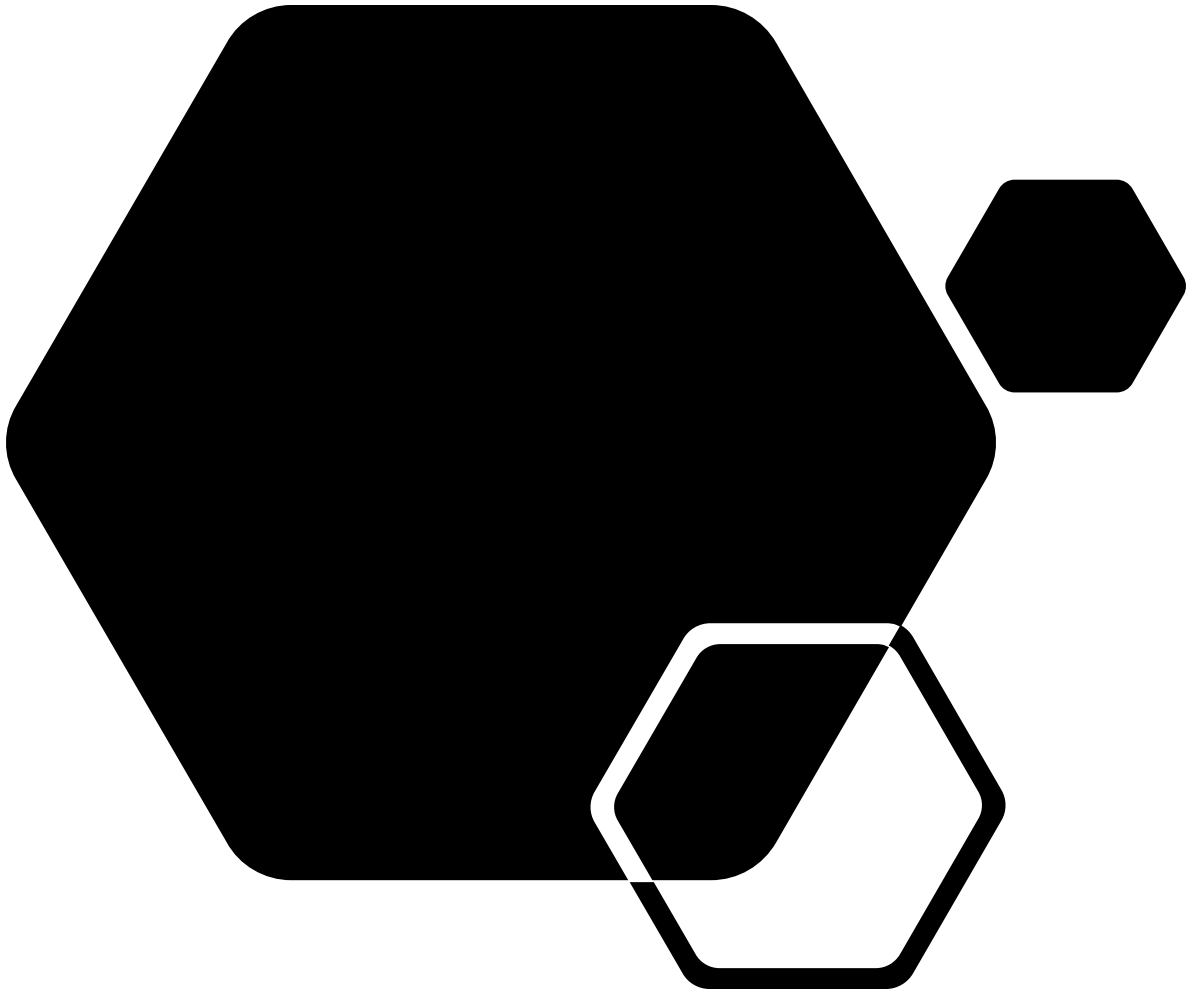
- Need to optimize the DL models on each DL accelerator manually.
- Need to rebuild entire software stack to work with the DL frameworks.
- Fall behind the rapid development of Deep Learning models.
- Fail to utilize the Deep Learning chips efficiently.

Deep Learning Compilers

- TVM
- Tensor Comprehension
- Glow
- NGraph
- XLA

Domain Specific **Compilers** are used to address the drawback of DL libraries and tools

Deep Learning Compilers



Deep Learning Software Requirements

- DAG of High-level Operators
- Early-Binding to compute
- Mixed Memory and Compute requirements

Deep Learning Hardware Requirements

- Heterogeneous hardware
  - Optimize workload for different h/w
- Layered Memory hierarchy
  - Complex scheduling
- Parallel computation
  - Single Instruction Multiple Data
  - Single Instruction Multiple Threads

Deep Learning Optimization Stack

Compilers are a bridge to lower the programs supporting the DL frameworks *auto matically or semi-automatically* to the accelerators.

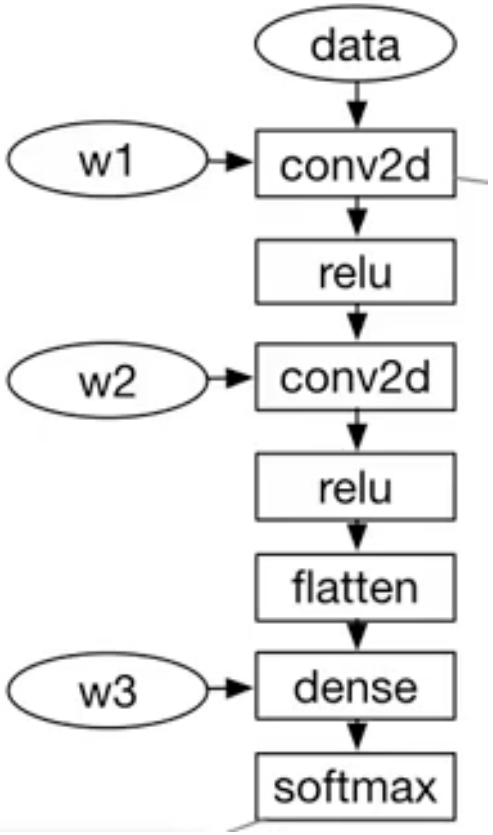
Computational Graph Optimizations

Tensor Expression Language

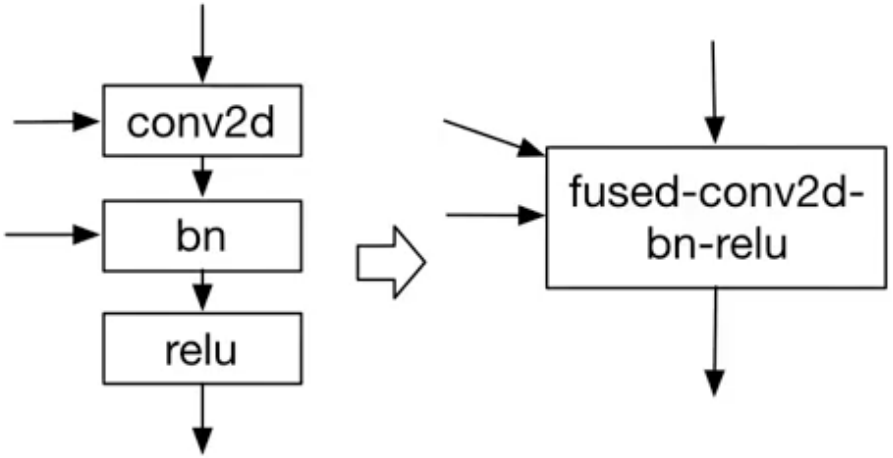
Schedule Optimizations

Computational Graph Optimization

High level representation of computation using primitive tensor operators



Equivalent transformation to early-binding compute for graph optimization



Need to build and optimize memory plan, operator fusion, data layout variations, precision, threading patterns, etc.

The **implementation** of tensor operations is **opaque**!

Tensor Expression Language

- Lower-level representation that can be used to express the computations.
- Support automatic code generation.
- Each compute operation is described in an index formula expression.
- Each compute operation specifies both the shape of the output tensor and an expression describing how to compute each element of it.
- Support common arithmetic and math operations.
- Covers common DL operator patterns.
- Provides flexibility for adding hardware-aware optimizations for various backends

```
C = tvm.compute((m, n),
    lambda i, j: tvm.sum(A[i, k] * B[j, k], axis=k))
```

Result Shape

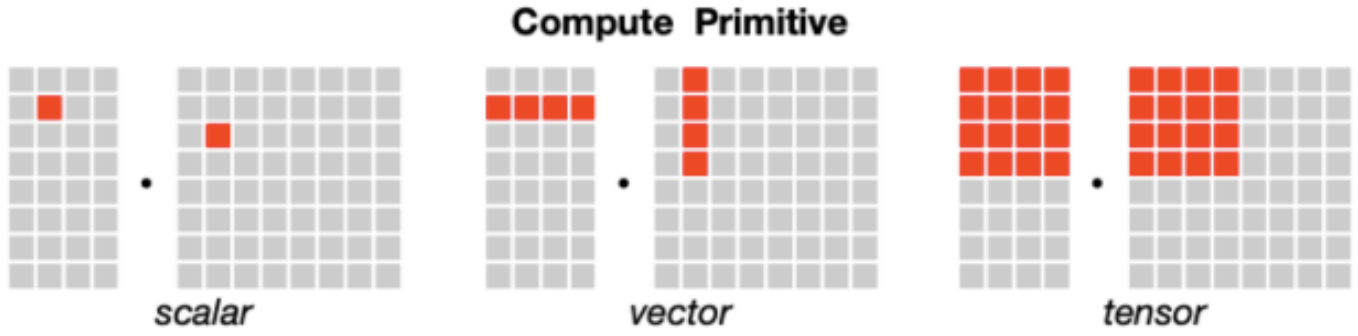
Computing rule

Hardware-aware search space.

Need to be **scheduled** to different hardware back-ends

Schedule Optimizations

- Bridge gap between Tensor Expression Language to hardware.
- Achieve high performance on many back-ends.
- Support schedule primitives to cover a diverse set of optimizations on different hardware back-ends.
- Transform high level programs to specific accelerator back-ends.



Tensorization Challenge

Tensorization Challenge

Different accelerators have different tensor utilization choices.

- Declare ***tensor instruction interface***.
- ***Tensorize***: Transform Program to use tensor instructions

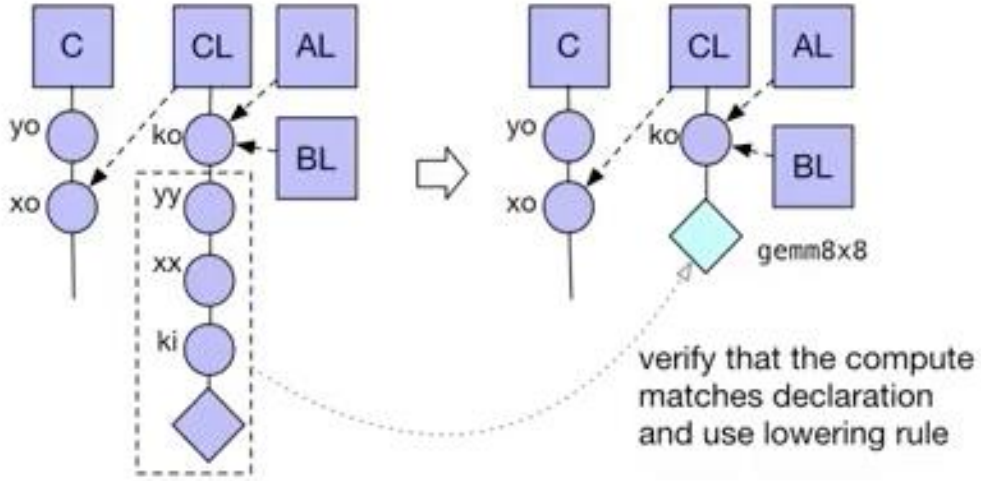
```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
                t.sum(w[i, k] * x[j, k], axis=k))

def gemm_intrin_lower(inputs, outputs):
    ww_ptr = inputs[0].access_ptr("r")
    xx_ptr = inputs[1].access_ptr("r")
    zz_ptr = outputs[0].access_ptr("w")
    compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
    reset = t.hardware_intrin("fill_zero", zz_ptr)
    update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
    return compute, reset, update

gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

declare behavior

lowering rule to generate hardware intrinsics to carry out the computation



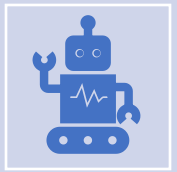
Challenge: To Support emerging tensor instructions

Problem Statement: Why TVM?

Express an operator in a simple language to abstract away the hardware complexity.

Automatically optimize an operator for different hardware.

TVM



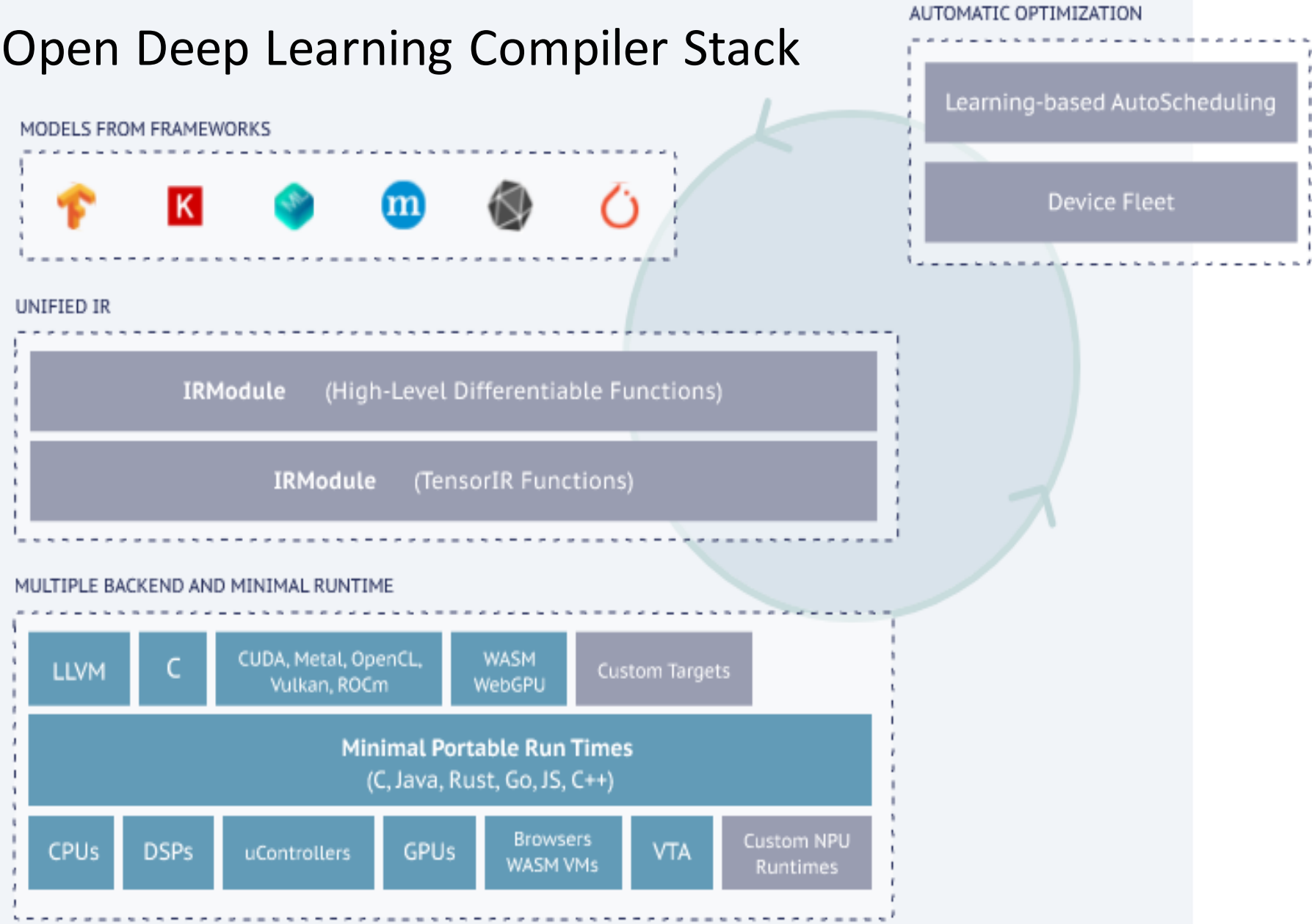
Find an efficient implementation of a new operator without thinking about hardware.

Tensor Virtual Machine (TVM)

Apache TVM

Open Deep Learning Compiler Stack

- Accessible, extensible, and automated open-source framework.
- Automatically generates and optimizes deep learning models on multiple hardware backend with minimal runtime.
- Compiles deep learning models into minimum deployable modules.



TVM: End to End Optimization Stack

High-level differentiable IR

Computational Graph Optimizations

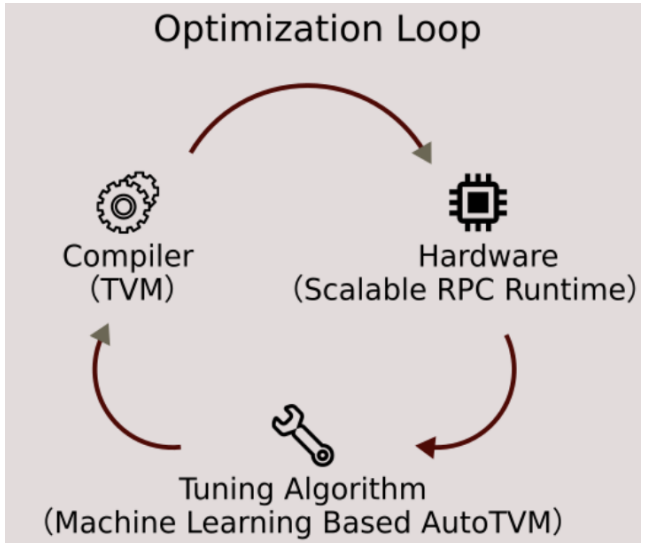
Tensor Expression Language

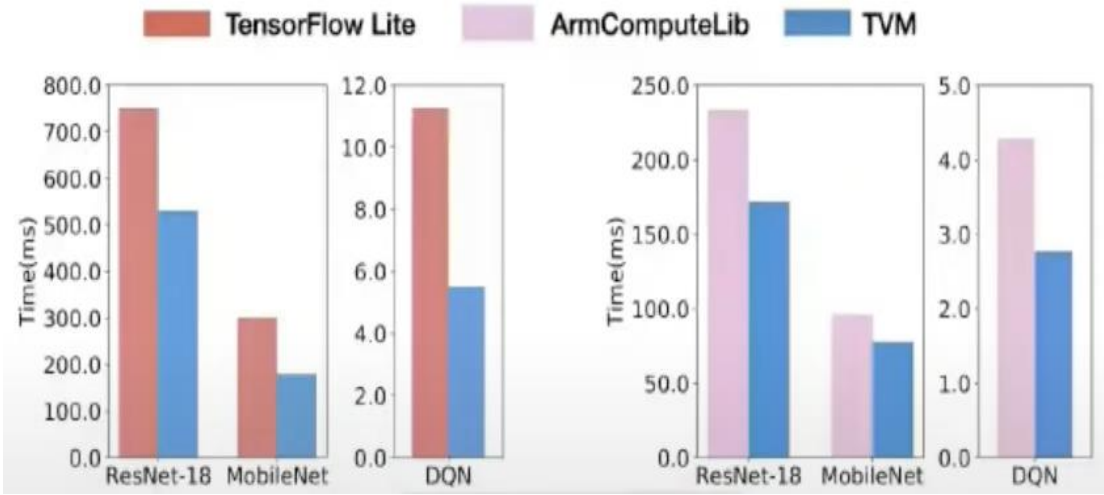
Optimization Search Space

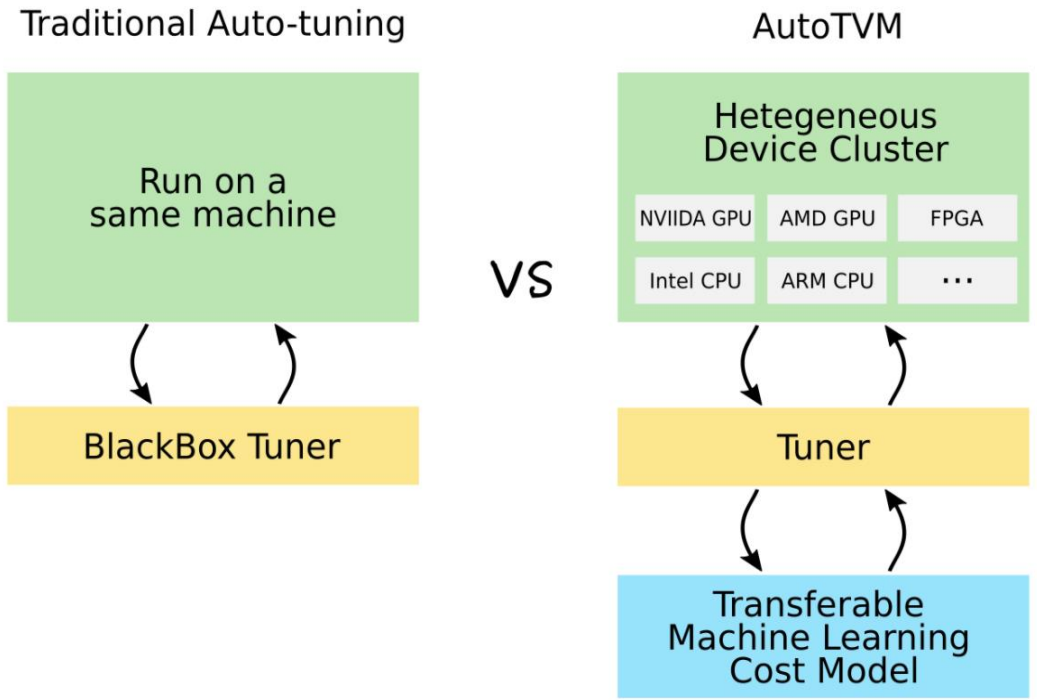
Loop Transformations	Thread Bindings	Cache Locality	Previous Primitives
Thread Cooperation	Tensorization	Latency Hiding	New Primitives for accelerators
...			

TVM: Key Features

- ***Performance***:
  Compilation with minimal runtime
- ***Unified Runtime***: in heterogeneous devices
- ***Unified IR and library packaging***
- ***Compatibility***: Automatically generates and optimizes tensor operators on multiple backends – CPUs, GPUs, Microcontrollers, FPGAs, browsers, etc.







TVM: Key Features cont...

- ***Flexibility***: Enables support for
  - Block Sparsity
  - Quantization
  - Classical ML and Random Forests
  - Memory Planning
  - MISRA-C, Python, C++, Rust, Java
- ***Usability***: Keras, MXNet, PyTorch, TensorFlow, CoreML, DarkNet, more.
- ***Full-stack automation***
  - Open source
  - Open development
  - Open governance

TVM: Usage

Create deployable module from a model

```
import tvm
import nnvm.frontend
import nnvm.compiler

graph, params =
nnvm.frontend.from_mxnet(mx_resnet50)
graph, lib, params =
     nnvm.compiler.build(graph, target)

module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```

Model In

Module Out (Deployable)

Tensor Expression

Describe Computation

```
from tvm import te

n = te.var("n")

A = te.placeholder((n,), name="A")

B = te.placeholder((n,), name="B")

C = te.compute(A.shape, lambda i: A[i] + B[i], name="C")
```

Schedule the Computation

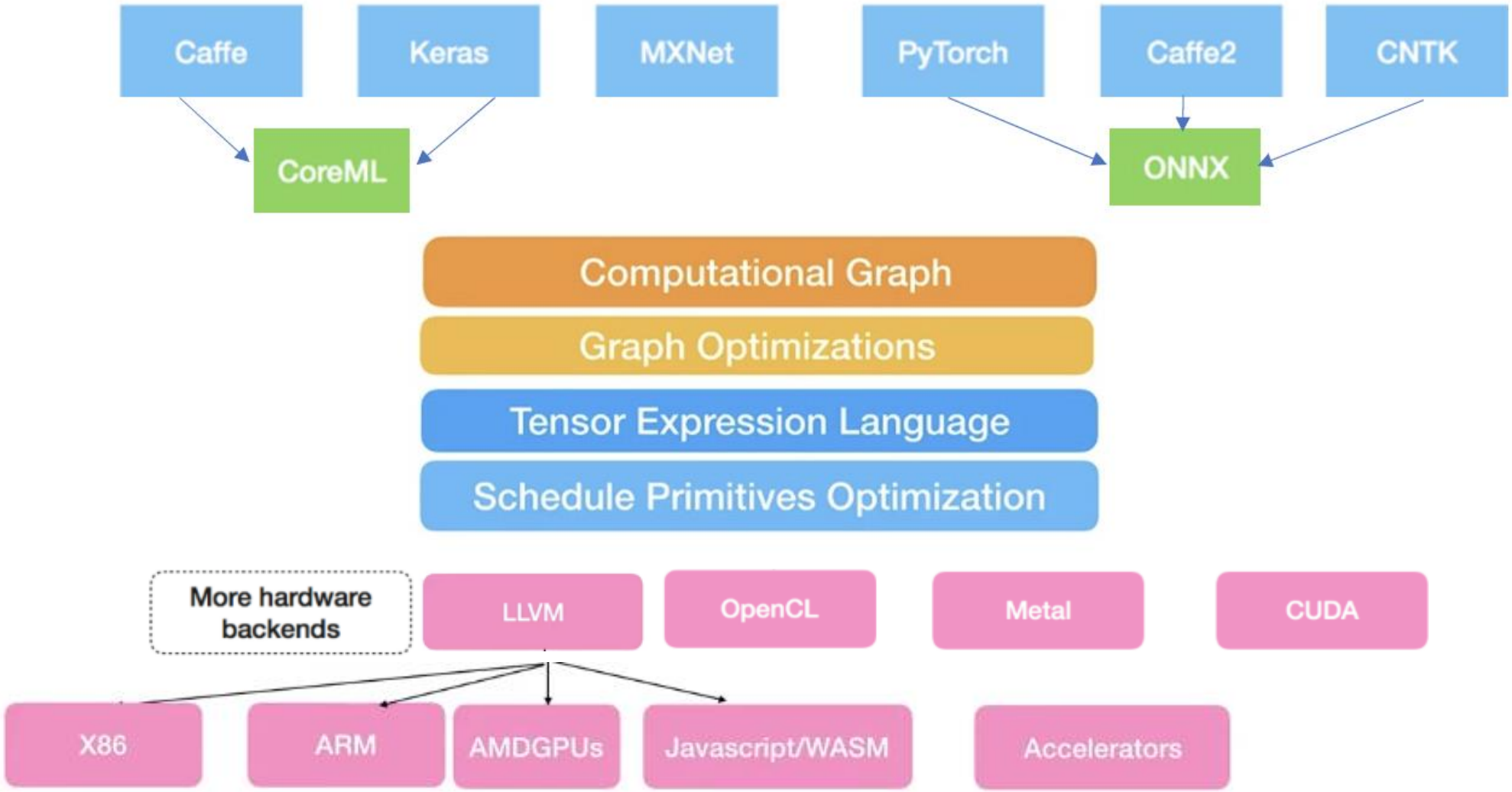
```
for (int i = 0; i < n; ++i) {

  C[i] = A[i] + B[i];

}

s = te.create_schedule(C.op)
```

TVM: Automated End to End Optimizing Compiler



References

- The deep learning compiler: a comprehensive survey: https://arxiv.org/abs/2002.03794
- TVM: an automated end-to-end optimizing compiler for deep learning: https://arxiv.org/abs/1802.04799
- Apache TVM: https://tvm.apache.org/
- Apache TVM documentation: https://tvm.apache.org/docs/
- 2019 TVM and deep learning compilation conference: morning keynote & session1: https://www.youtube.com/watch?v=npqO0hVXZwU
- Computer Architecture: https://www.elsevier.com/books/computer-architecture/hennessy/978-0-12-811905-1
- Torch documentation: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
- AWS DL AMIs: https://aws.amazon.com/machine-learning/amis/
- Paper: https://homes.cs.washington.edu/~arvind/papers/tvm.pdf