machine learning and mobile

brettkoonce.com/talks

november 15th, 2019

overview

- goal: explore image recognition on mobile/edge devices
- review current state of the art
- where things are going
- demo, recap

assumptions

- edge: phone, car, sensor, satellite
- compute: edge << cloud
- bandwidth: limited/unreliable
- power: have to be efficient
- decision time: bounded/interactive

why

- resnet: 224x224 pixels
- gpipe: 499x499 pixels
- iphone xr: 4k @ 60 fps

coreml

- coremltools: xgboost, scikit-learn, keras
- turicreate: above + numpy + metal
- cons: iOS only :: pros: fast
- good starting point if new to field
- demo: mnist + keras + coreml (2017)

tensorflow-lite

- ios/android/edge devices
- build tensorflow model, graphdef
- convert to tensorflow lite operations
- demo: ios + video feed + mobilenet v1

pytorch

- 1.3 release: october 2019
- pytorch model —> jit trace —> .pt model
- .pt model + ios/android libraries
- demo: ios + video feed + mobilenet v2

embedded linux

- bring own hardware + linux in some form
- opencv all the things
- dlib, matlab, c++ libraries + shims
- good prototyping platform

custom hardware

- arbitrary integer, floating point depth
- asics/macs
- probabilistic processors
- build it and they will come
- new limit: software

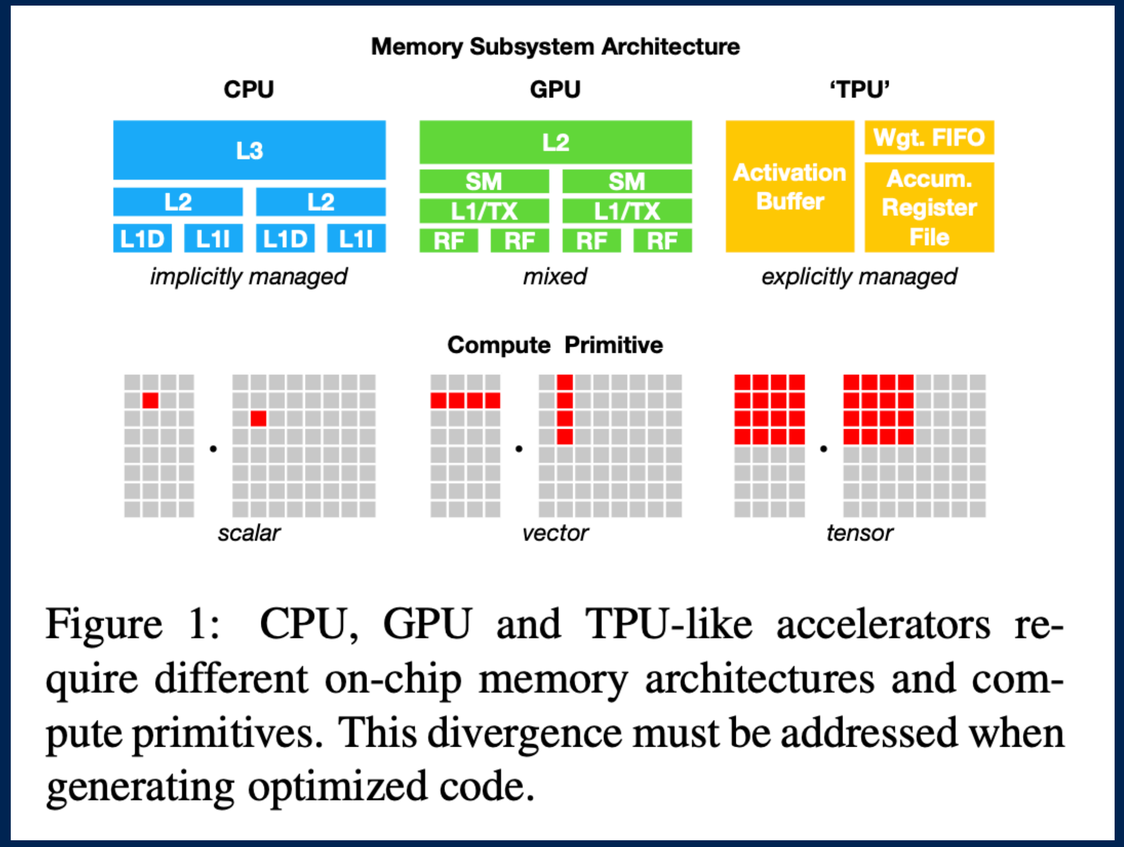
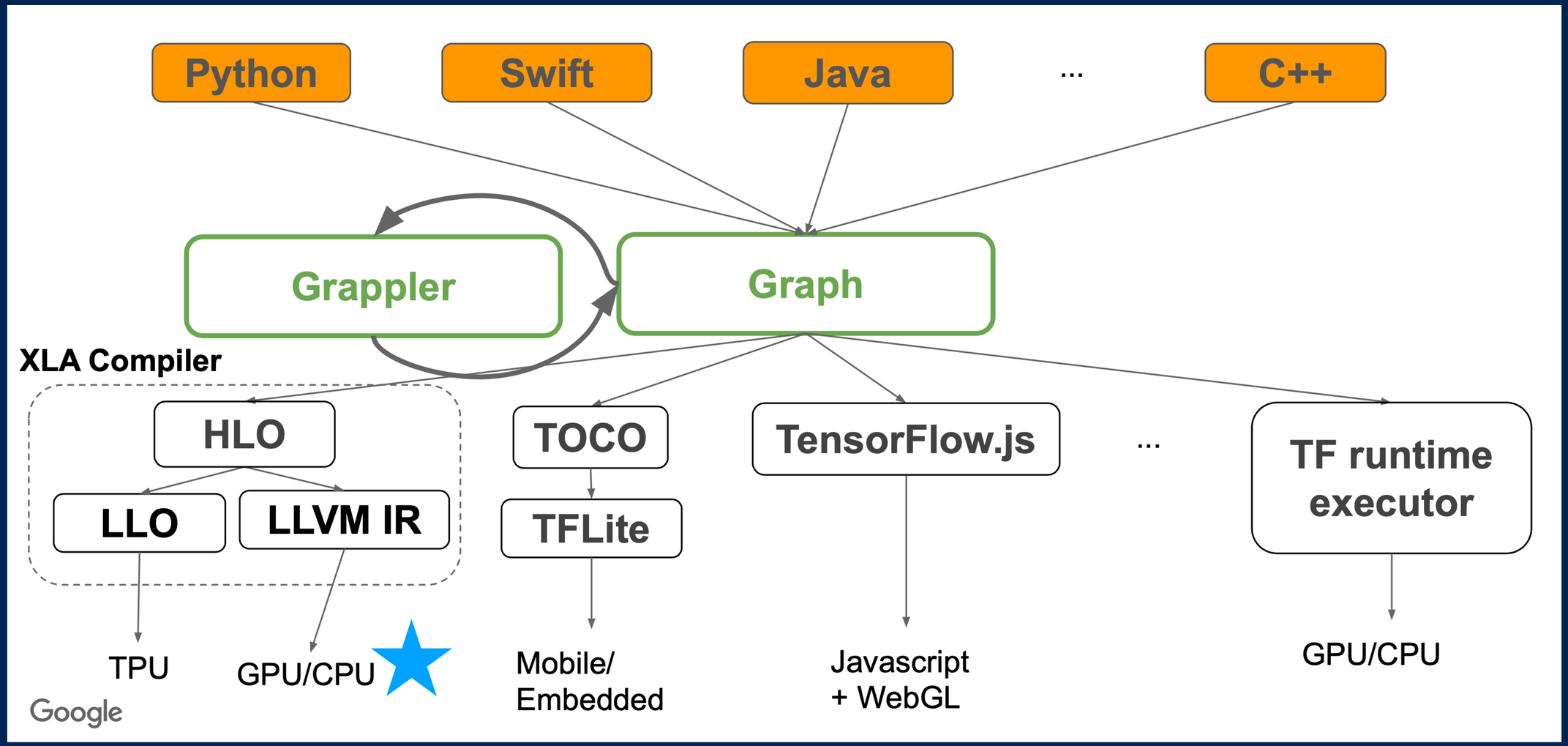


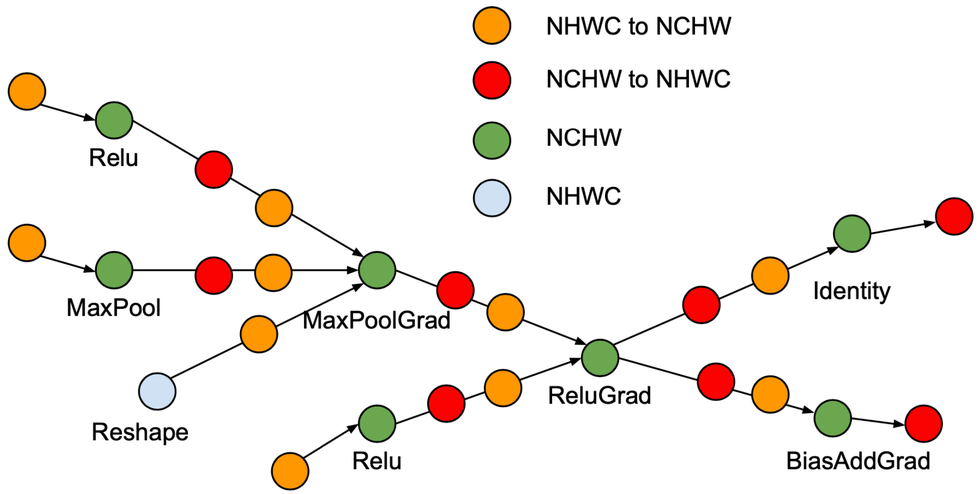
Figure 1: CPU, GPU and TPU-like accelerators require different on-chip memory architectures and compute primitives. This divergence must be addressed when generating optimized code.

tensorflow today

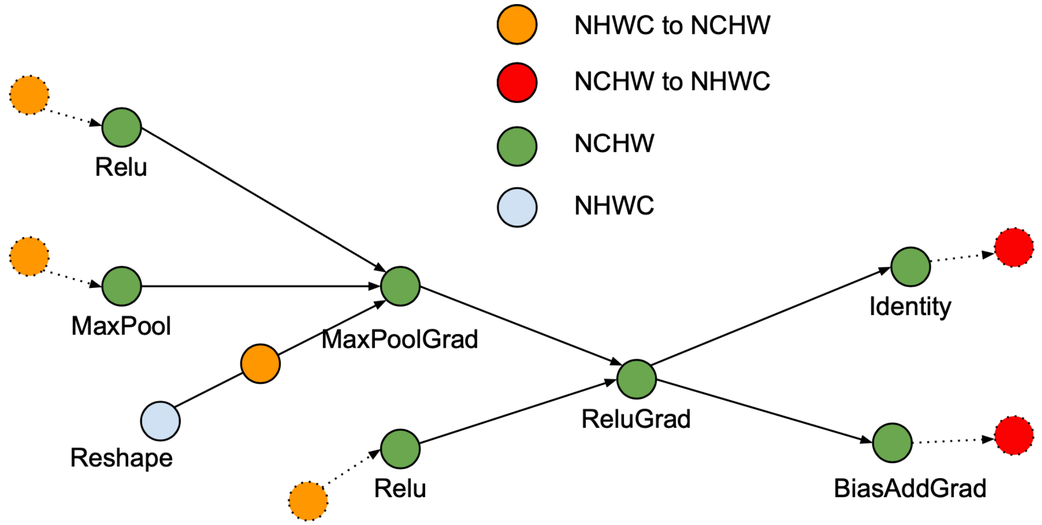


operator fusion

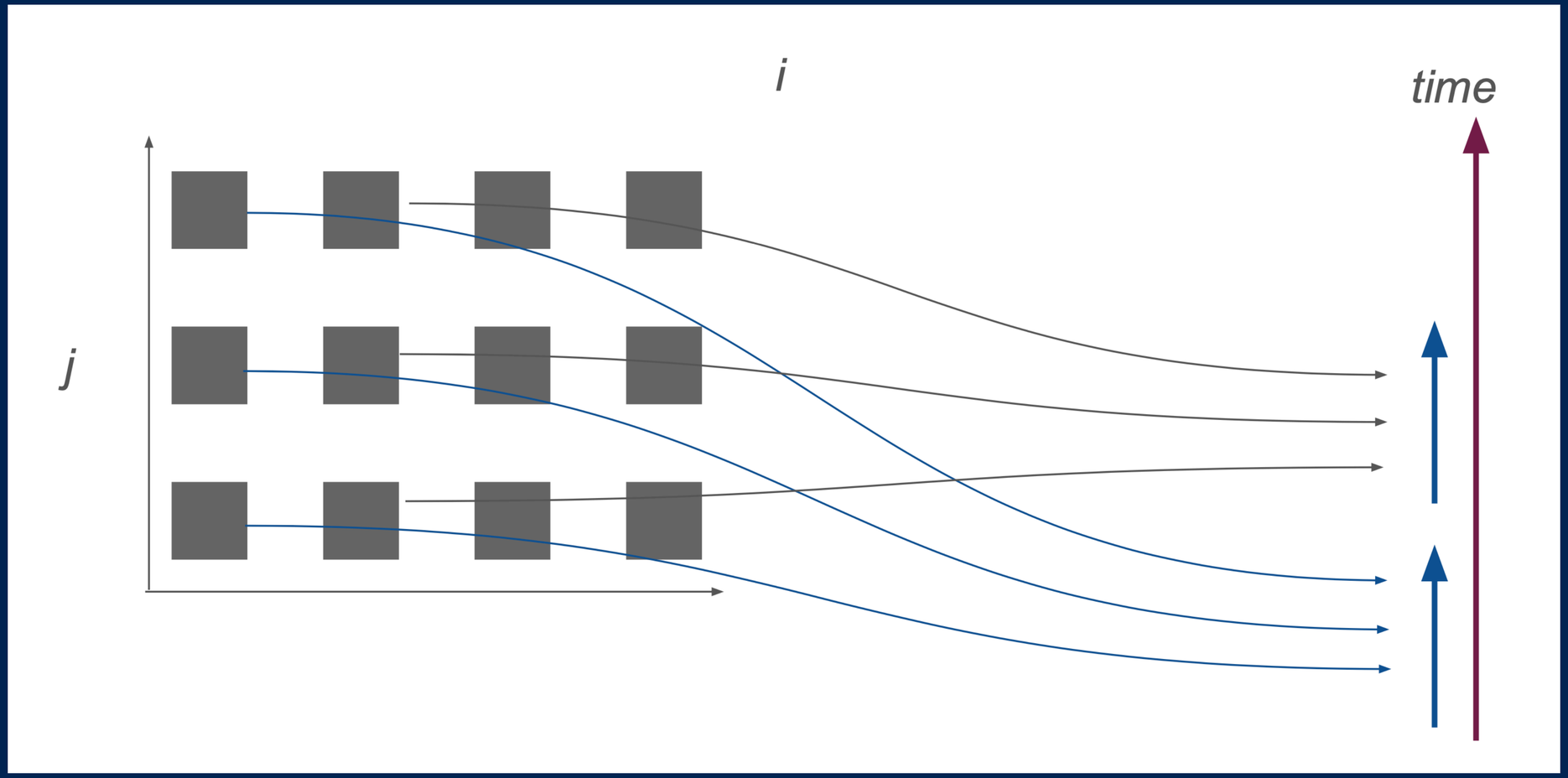
Phase 1: Expand by inserting conversion pairs



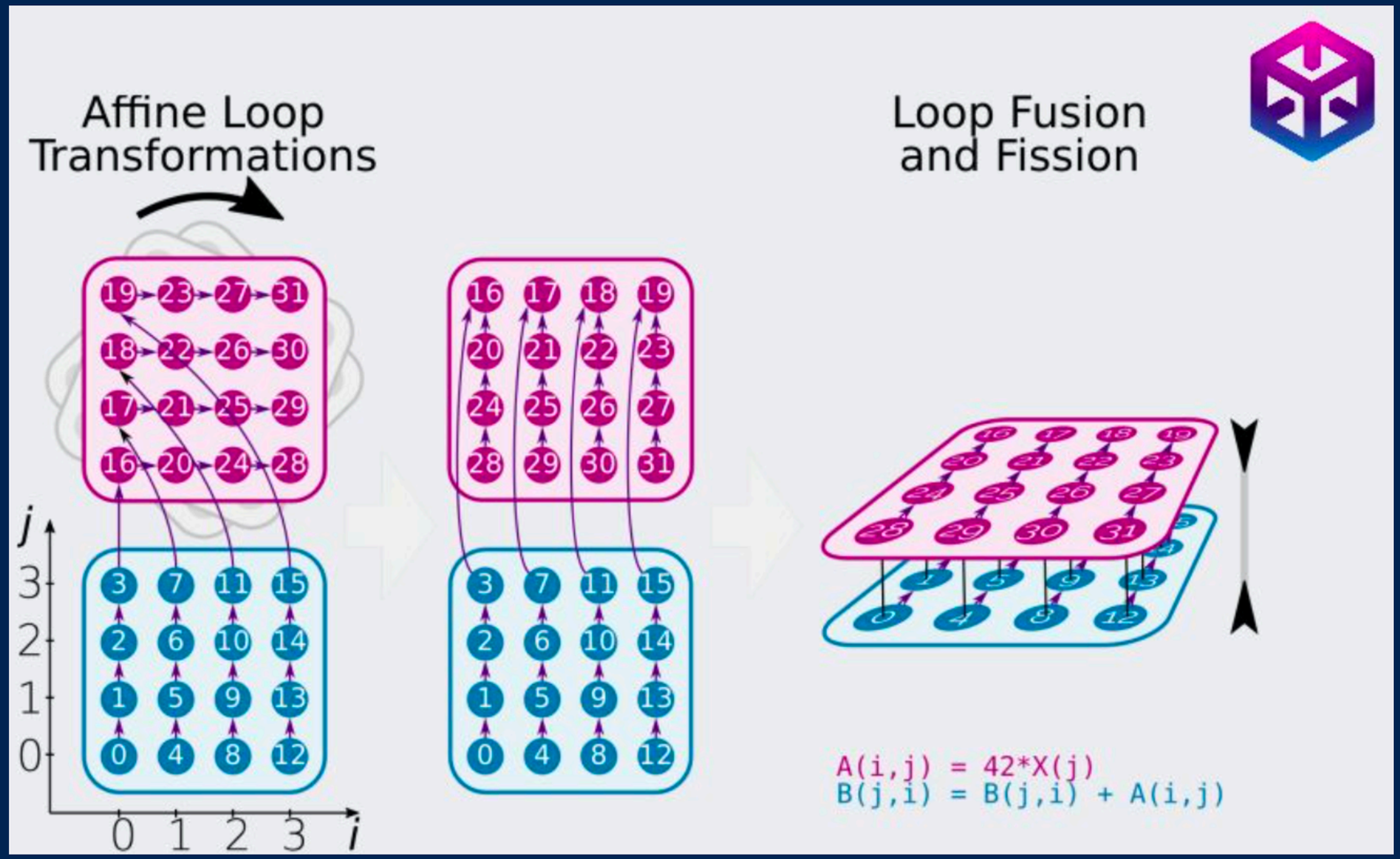
Phase 2: Collapse adjacent conversion pairs



pipelining



polyhedral approach



glow

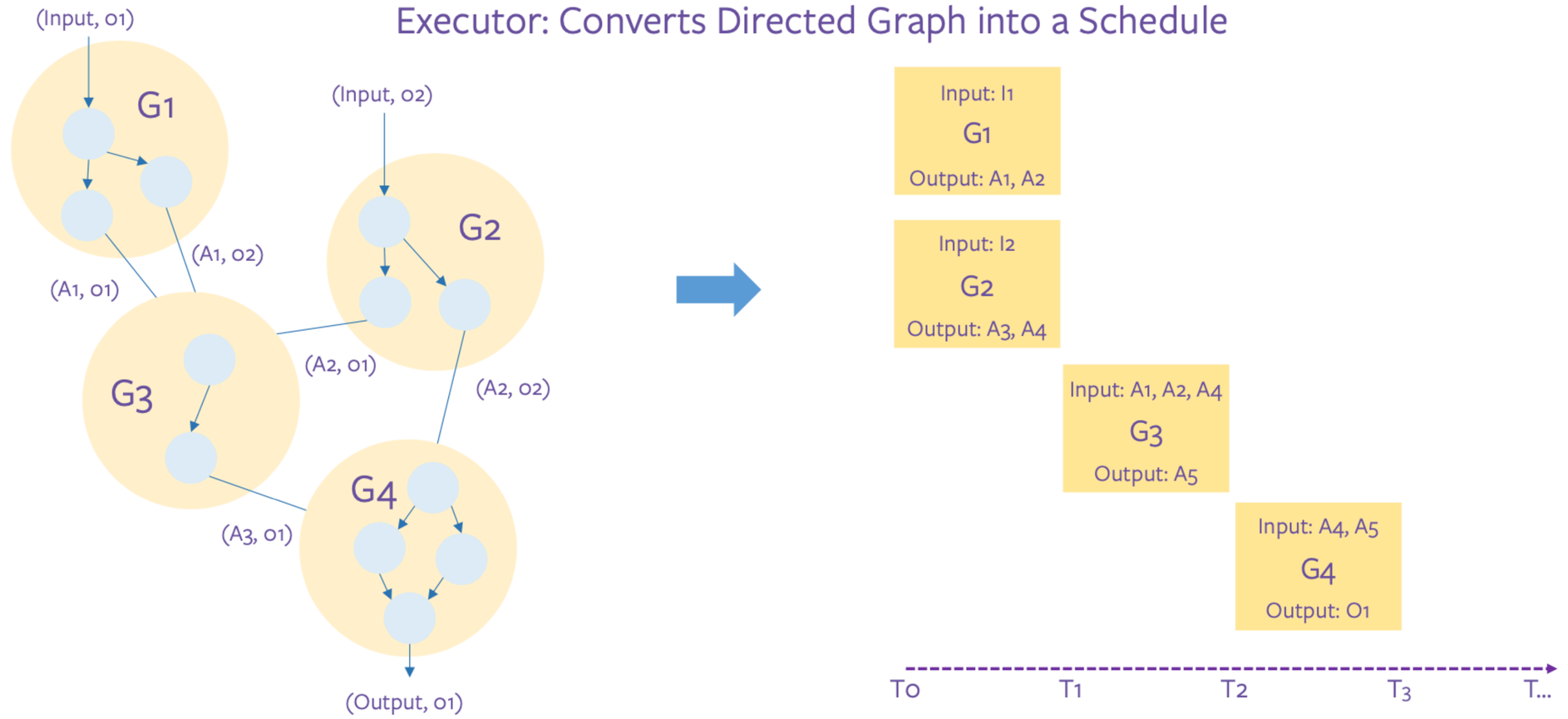


Figure 9: A simple example showing a graph partitioned into multiple sub-graphs, themselves making up a directed graph, and then converted into a schedule by the executor.

mesh-tf

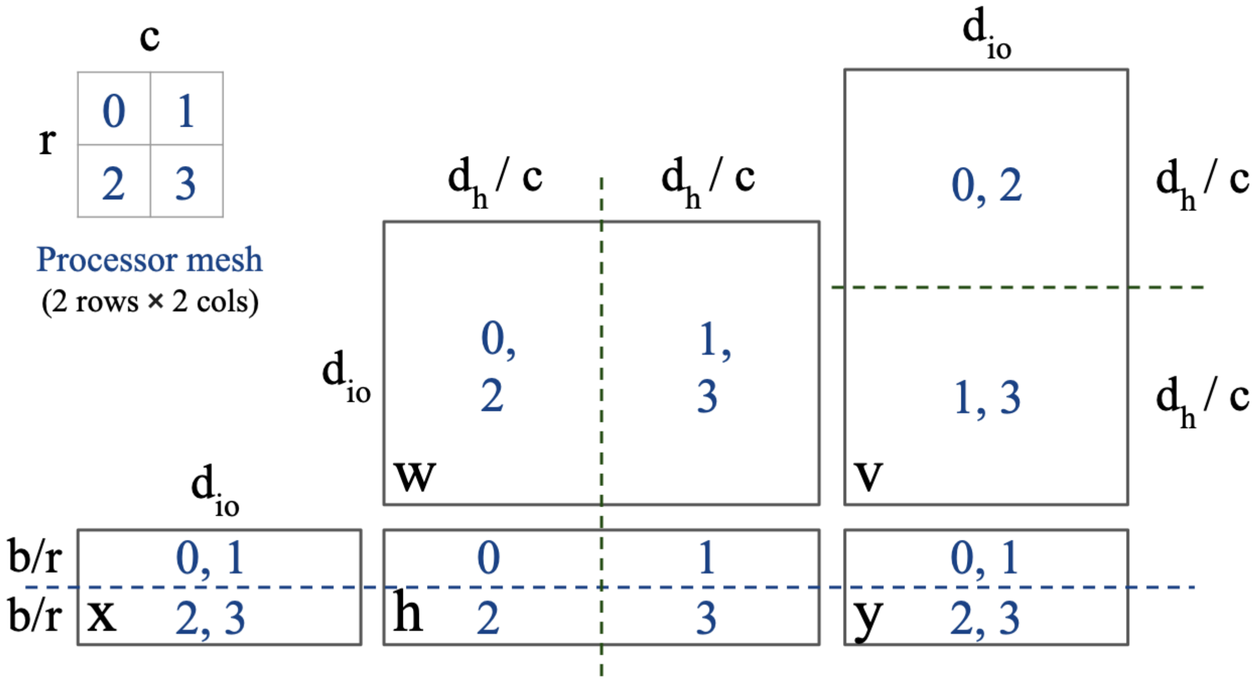
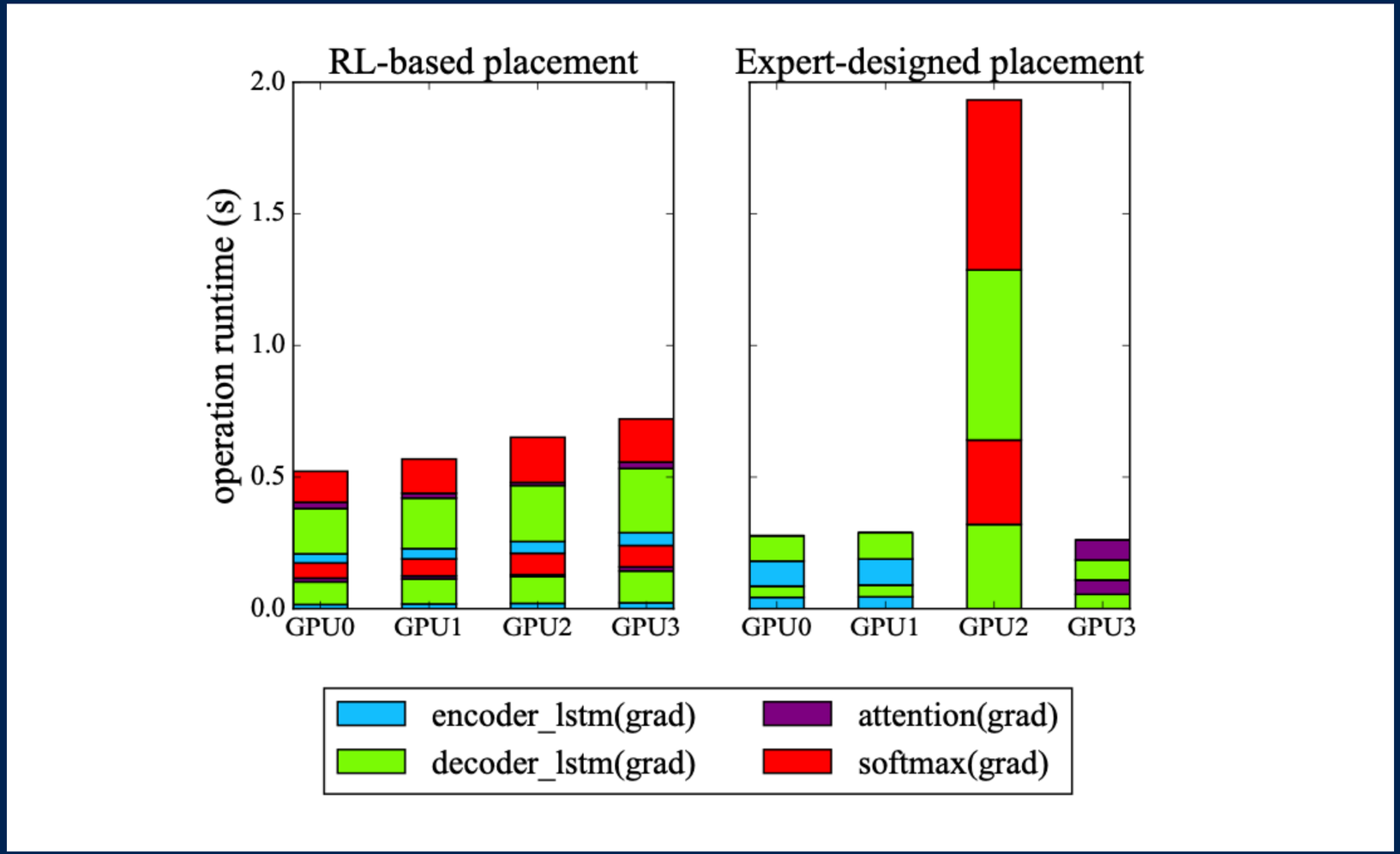


Figure 4: The mixed data-and-model-parallel layout for the Two Fully-Connected Layers example. There are 4 processors, arranged into a 2-by-2 mesh. Each processor is assigned a serialized rank which is used to label matrix slices that it owns.

evolutionary approaches



compiler exploration

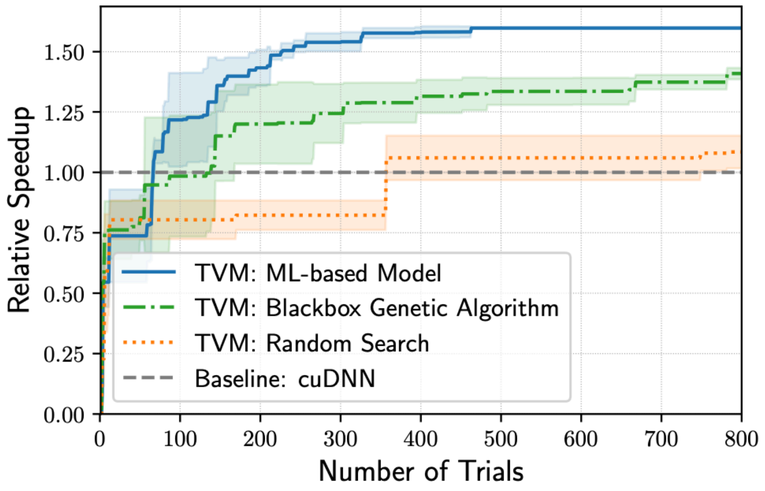
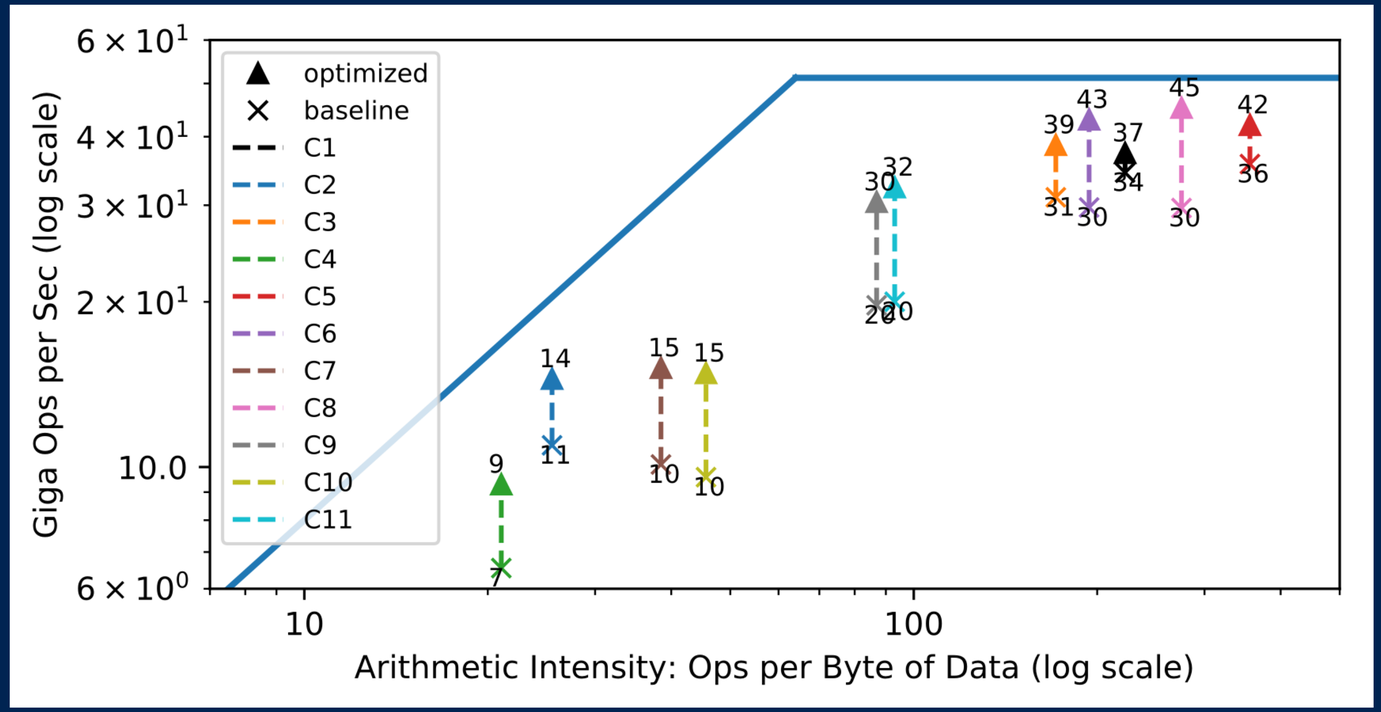
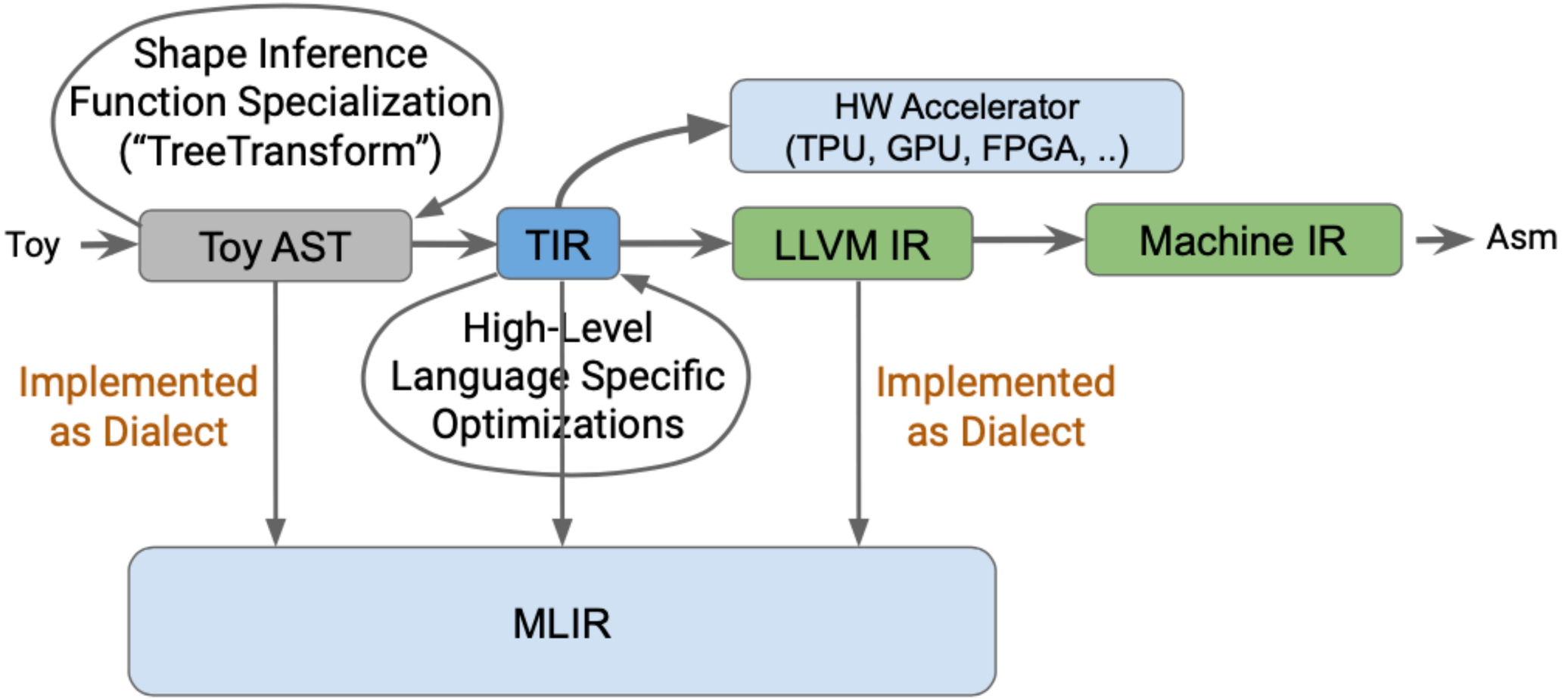
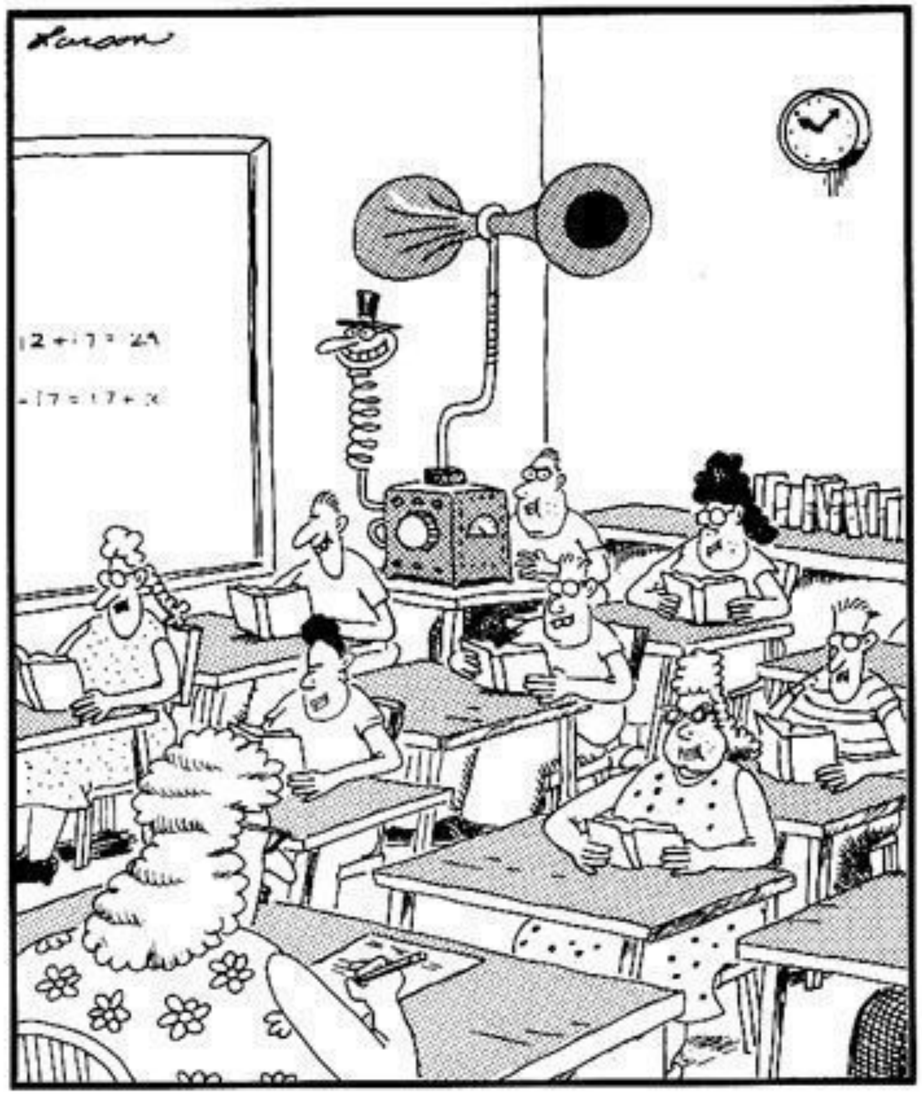


Figure 12: Comparison of different automation methods for a conv2d operator in ResNet-18 on TITAN X. The ML-based model starts with no training data and uses the collected data to improve itself. The Y-axis is the speedup relative to cuDNN. We observe a similar trend for other workloads.



MLIR allows every level to be represented as a Dialect





The class was quietly doing its lesson when Russell, suffering from problems at home, prepared to employ an attention-getting device.

swift/🐍/mlir/tf-lite

- swift —> cifar —> train —> model
- s4tf —> numpy —> keras —> h5
- freeze —> .pb —> mlir —> tf-lite —> device
- demo: swift —> keras, h5 —> .pb, .pb —> mlir —> tf-lite, tf-lite + ios

```
struct CIFARModel: Layer {
    typealias Input = Tensor<Float>
    typealias Output = Tensor<Float>

    var conv1a = Conv2D<Float>(filterShape: (3, 3, 3, 32), padding: .same, activation: relu)
    var conv1b = Conv2D<Float>(filterShape: (3, 3, 32, 32), activation: relu)
    var pool1 = MaxPool2D<Float>(poolSize: (2, 2), strides: (2, 2))

    var conv2a = Conv2D<Float>(filterShape: (3, 3, 32, 64), padding: .same, activation: relu)
    var conv2b = Conv2D<Float>(filterShape: (3, 3, 64, 64), activation: relu)
    var pool2 = MaxPool2D<Float>(poolSize: (2, 2), strides: (2, 2))

    var flatten = Flatten<Float>()
    var dense1 = Dense<Float>(inputSize: 6 * 6 * 64, outputSize: 512, activation: relu)
    var dense2 = Dense<Float>(inputSize: 512, outputSize: 512, activation: relu)
    var output = Dense<Float>(inputSize: 512, outputSize: 10, activation: identity)

    @differentiable
    func callAsFunction(_ input: Input) -> Output {
        let conv1 = input.sequenced(through: conv1a, conv1b, pool1)
        let conv2 = conv1.sequenced(through: conv2a, conv2b, pool2)
        return conv2.sequenced(through: flatten, dense1, dense2, output)
    }
```

```
func save() {
    let keras = Python.import("keras")
    let model = keras.Sequential()

    model.add(keras.layers.Conv2D(filters: 32, kernel_size: [3, 3],
              padding: "same", activation: "relu", input_shape: [32, 32, 3]))
    model.add(keras.layers.Conv2D(filters: 32, kernel_size: [3, 3], activation: "relu"))
    model.add(keras.layers.MaxPooling2D(pool_size: [2, 2], strides: [2, 2]))

    model.add(keras.layers.Conv2D(filters: 64, kernel_size: [3, 3],
              padding: "same", activation: "relu"))
    model.add(keras.layers.Conv2D(filters: 64, kernel_size: [3, 3], activation: "relu"))
    model.add(keras.layers.MaxPooling2D(pool_size: [2, 2], strides: [2, 2]))

    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(512, activation: "relu", input_shape: [2304]))
    model.add(keras.layers.Dense(512, activation: "relu"))
    model.add(keras.layers.Dense(10, activation: "softmax"))

    print (model.summary())

    model.layers[0].set_weights([conv1a.filter.makeNumpyArray(), conv1a.bias.makeNumpyArray()])
    model.layers[1].set_weights([conv1b.filter.makeNumpyArray(), conv1b.bias.makeNumpyArray()])

    model.layers[3].set_weights([conv2a.filter.makeNumpyArray(), conv2a.bias.makeNumpyArray()])
    model.layers[4].set_weights([conv2b.filter.makeNumpyArray(), conv2b.bias.makeNumpyArray()])

    model.layers[7].set_weights([dense1.weight.makeNumpyArray(), dense1.bias.makeNumpyArray()])
    model.layers[8].set_weights([dense2.weight.makeNumpyArray(), dense2.bias.makeNumpyArray()])
    model.layers[9].set_weights([output.weight.makeNumpyArray(), output.bias.makeNumpyArray()])

    model.save("cifar.h5")
  }
```

```python
import numpy as np
import tensorflow as tf

def freeze_session(session, keep_var_names=None, output_names=None, clear_devices=True):
    """
    Freezes the state of a session into a pruned computation graph.

    Creates a new computation graph where variable nodes are replaced by
    constants taking their current value in the session. The new graph will be
    pruned so subgraphs that are not necessary to compute the requested
    outputs are removed.
    @param session The TensorFlow session to be frozen.
    @param keep_var_names A list of variable names that should not be frozen,
                          or None to freeze all the variables in the graph.
    @param output_names Names of the relevant graph outputs.
    @param clear_devices Remove the device directives from the graph for better portability.
    @return The frozen graph definition.
    """
    graph = session.graph
    with graph.as_default():
        freeze_var_names = list(set(v.op.name for v in tf.global_variables()).difference(keep_var_names or []))
        output_names = output_names or []
        output_names += [v.op.name for v in tf.global_variables()]
        input_graph_def = graph.as_graph_def()
        if clear_devices:
            for node in input_graph_def.node:
                node.device = ''
        frozen_graph = tf.graph_util.convert_variables_to_constants(
            session, input_graph_def, output_names, freeze_var_names)
        return frozen_graph


from tensorflow import keras
from tensorflow.keras import layers

model = keras.models.load_model('/home/skoonce/swift/cifar-mlir-demo/swift-models/cifar.h5')

frozen_graph = freeze_session(tf.keras.backend.get_session(), output_names=[out.op.name for out in model.outputs])

tf.io.write_graph(frozen_graph, './', 'cifar.pbtxt', as_text=True)
tf.io.write_graph(frozen_graph, './', 'cifar.pb', as_text=False)
```

```
bazel build --config opt tensorflow/compiler/mlir/...

./tensorflow/bazel-bin/tensorflow/compiler/mlir/lite/tf_tfl_translate \
-tf-input-shapes=1,32,32,3 \
-tf-input-data-types=DT_FLOAT \
-tf-output-arrays=dense_3/Softmax \
/home/skoonce/swift/cifar-mlir-demo/cifar.pb \
--tf-input-arrays=conv2d_1_input \
-o cifar-mlir-demo/cifar-mlir.tflite
```

recap

- goal: image recognition on edge devices
- current high level approaches
- software/hardware fusion
- swift —> mlir —> tf-lite on device

kudos

- mlir team
- s4tf team
- fast.ai team
- meir rosendorff blog
- koan-sin tan mlir/tf slides

papers

- mlir/llvm presentations
- albert cohen / “Polyhedral Compilation as a Design Pattern for Compiler Construction”
- tvm (1802.04799), glow (1805.00907), mesh-tf (1811.02084)
- Device Placement Optimization with Reinforcement Learning (1706.04972)

thanks for coming!