IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

APPLIED ACCELERATED ARTIFICIAL INTELLIGENCE

Accelerated TensorFlow – XLA Approach

Dr. Satyajit Das
Assistant Professor
Data Science
Computer Science and Engineering
IIT Palakkad

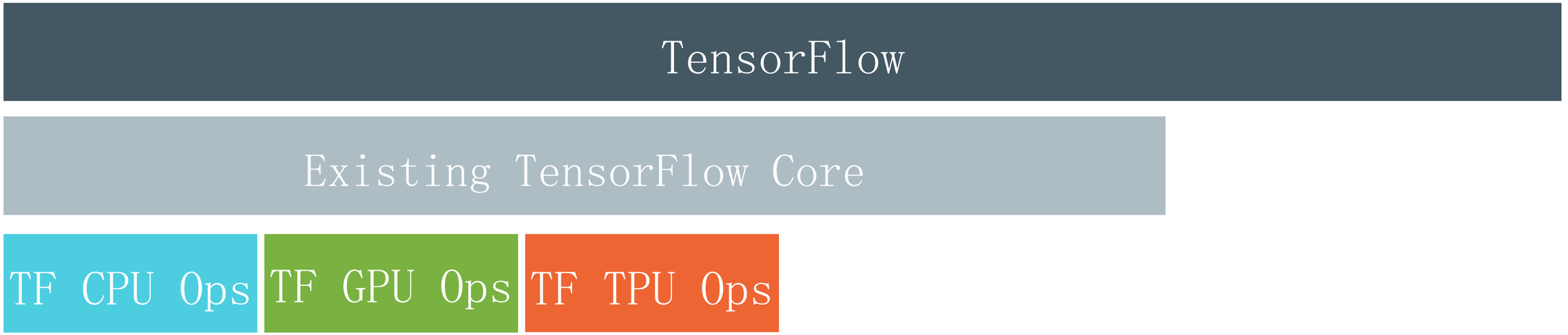
National Supercomputing Mission

Centre for Development of Advanced Computing

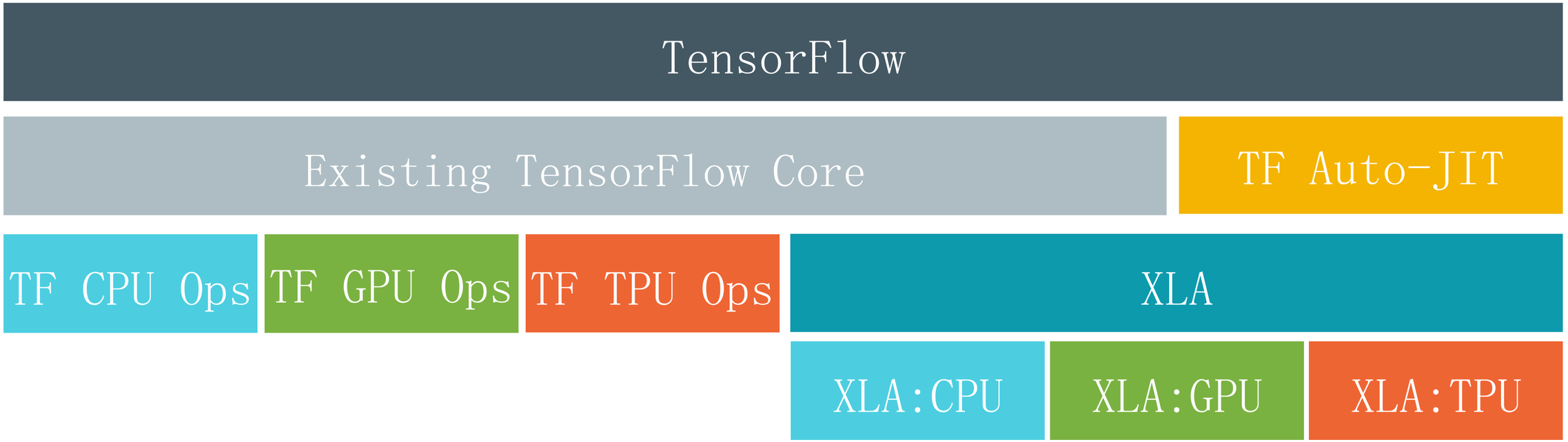
Computing with TensorFlow

- Write simple Python Code
- Run on different accelerators
- Get optimized performance

TF block diagram



TF block diagram



Running Example: Mean of Vector Sum

```
import tensorflow as tf


def running_example(x, y):
    return tf.reduce_mean(tf.multiply(x ** 2, 3) + y)


x = tf.random.uniform((16384,16384))
y = tf.random.uniform((16384,16384))
print(running_example(x, y))
```

Credit: George Karpenkov

TF Eager GPU Performance

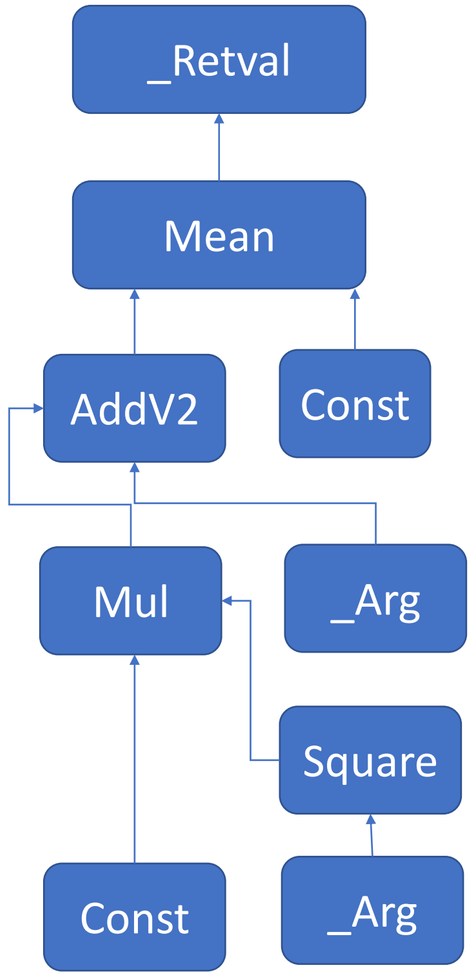
- Use Tensorboard for profiling
- Each op: separate CUDA kernel

Credit: George Karpenkov

Graph Performance: @tf.function

- Graph rewrites: pattern matching optimizations
- Generates serializable representation
  - Runnable without Python

```
@tf.function
def running_example(x, y):
  return tf.reduce_mean(tf.multiply(x ** 2, 3) + y)
```



Credit: George Karpenkov

@tf.function Performance Limitations

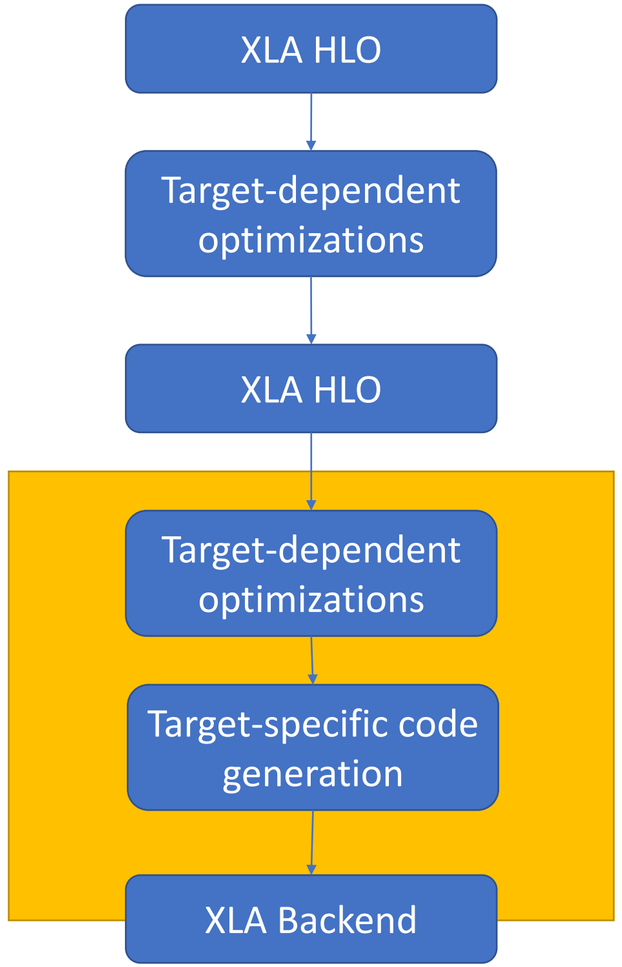
- Does not generate new code
- Limited to a fixed set of predefined kernels
- What if we could generate new kernels on the fly?

Credit: George Karpenkov

XLA Compiler tensorflow.org/xla

XLA: Linear Algebra Compiler

- Fast Optimizing Compiler
- Operates on HLO IR
- Many backends: GPU/CPU/TPU/…

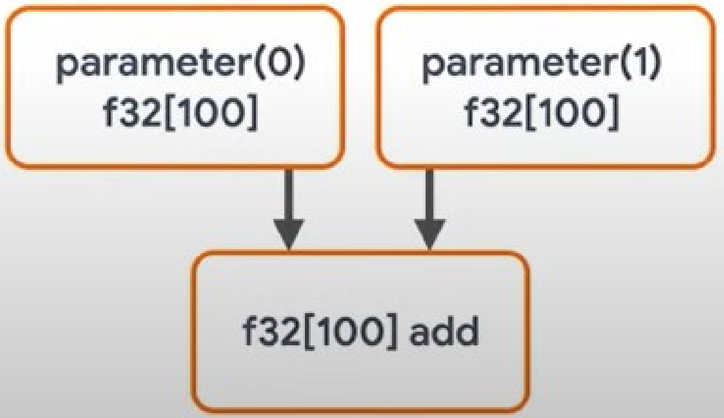


Credit: George Karpenkov

HLO IR

- Mostly functional (excluding collectives e.g. all-reduce)
- ~50 available operations (in contrast with ~1500 in TF)
- Type: datatype + shape + [layout]

```
Add (x: f32[100,100], : f32[100,100]) -> f32[100,100] {
  x = f32[100,100] parameter(0)
  y = f32[100,100] parameter(1)
  ROOT add = f32[100,100] add(x, y)
}
```



Credit: George Karpenkov

Fusion

- One of the most impactful optimizations on GPU
- Combines different computations together
- Drastically reduces the memory bandwidth

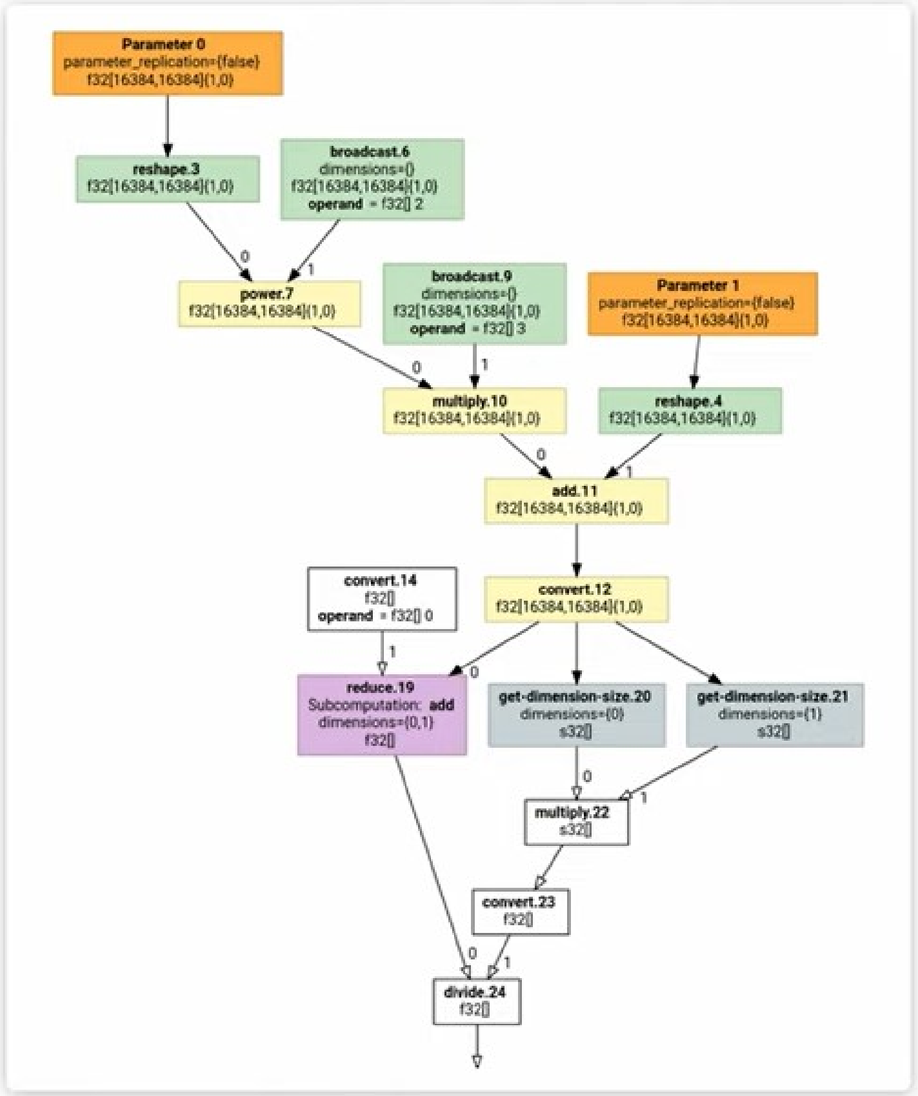
```
__global__ void Multiply(float a, float* x) {
  int idx = threadIdx.x;
  x[idx] = a * x[idx];
}
__global__ void Add(float* x, float* y) {
  int idx = threadIdx.x;
  x[idx] = x[idx] + y[idx];
}
```

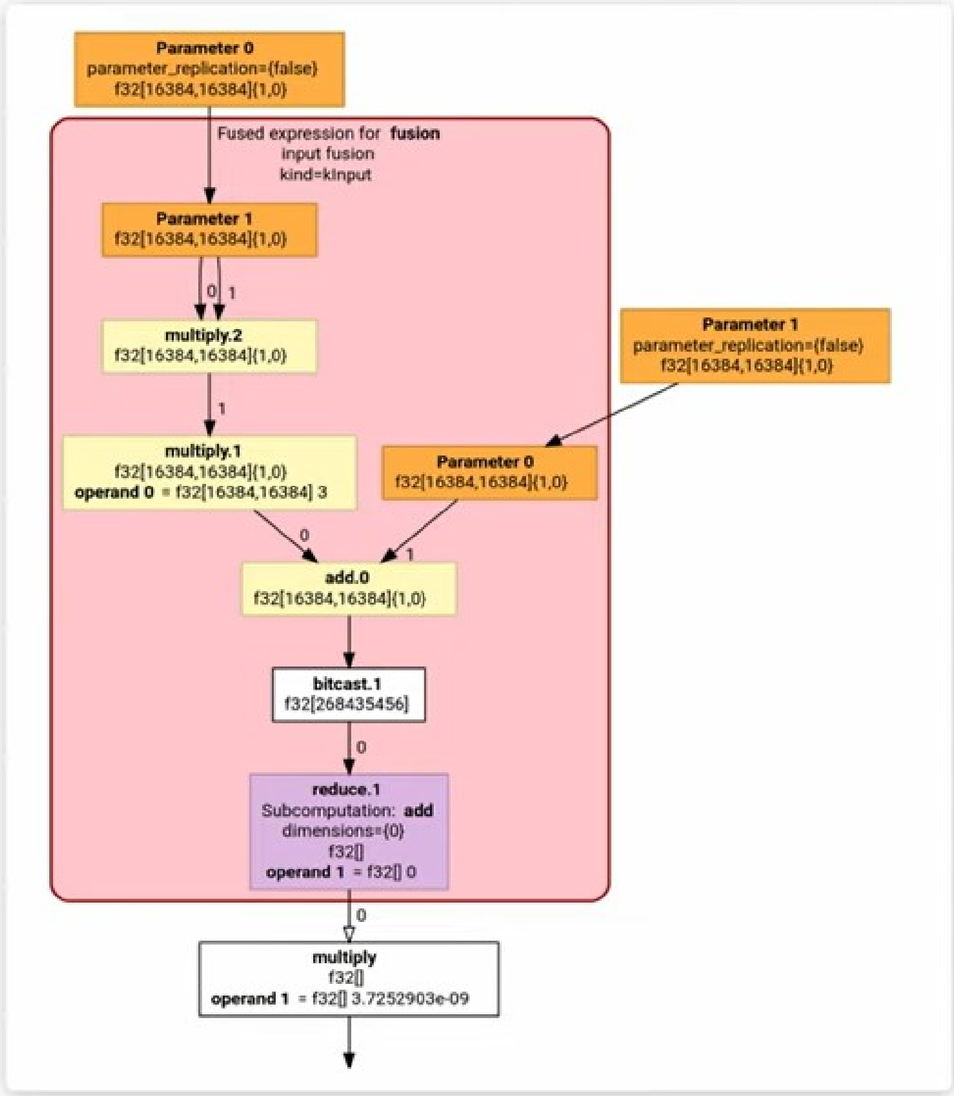
Fusion

```
__global__ void FusedMulAdd(float a,
float* x, float* y) {
  int idx = threadIdx.x;
  x[idx] = a * x[idx] + y[idx];
}
```

Credit: George Karpenkov

HLO Before/After Optimizations





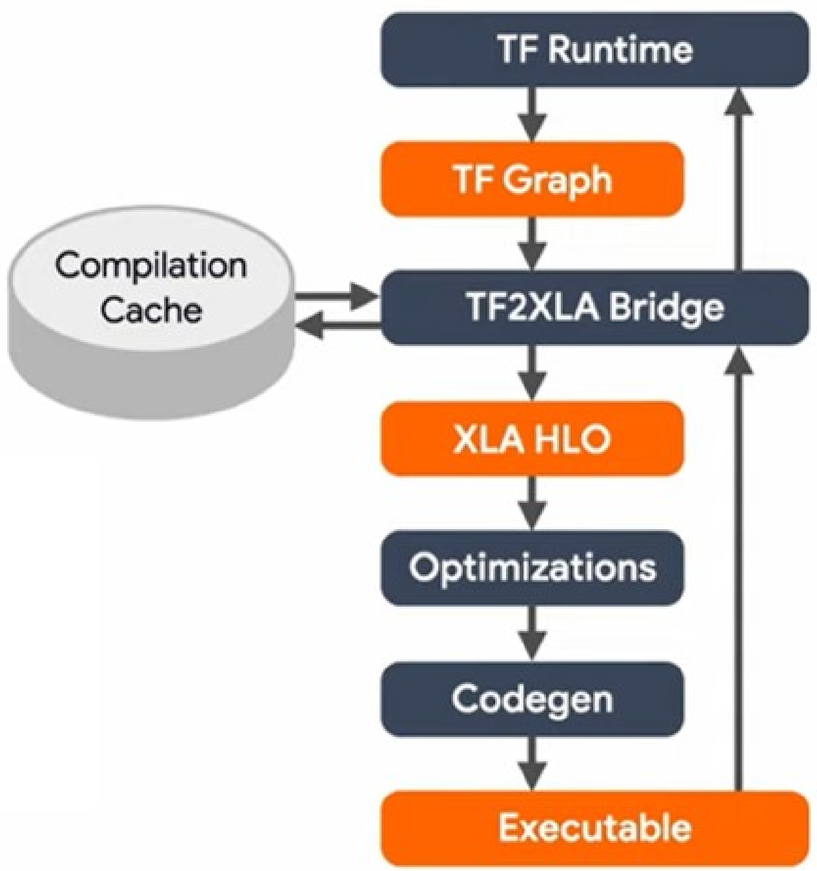
Credit: George Karpenkov

Using XLA from TensorFlow

Details and Limitations

Using XLA from TF

- @tf.function(jit_compile = True)
- Compile the graph from @tf.function
- Performs just-in-time Compilation

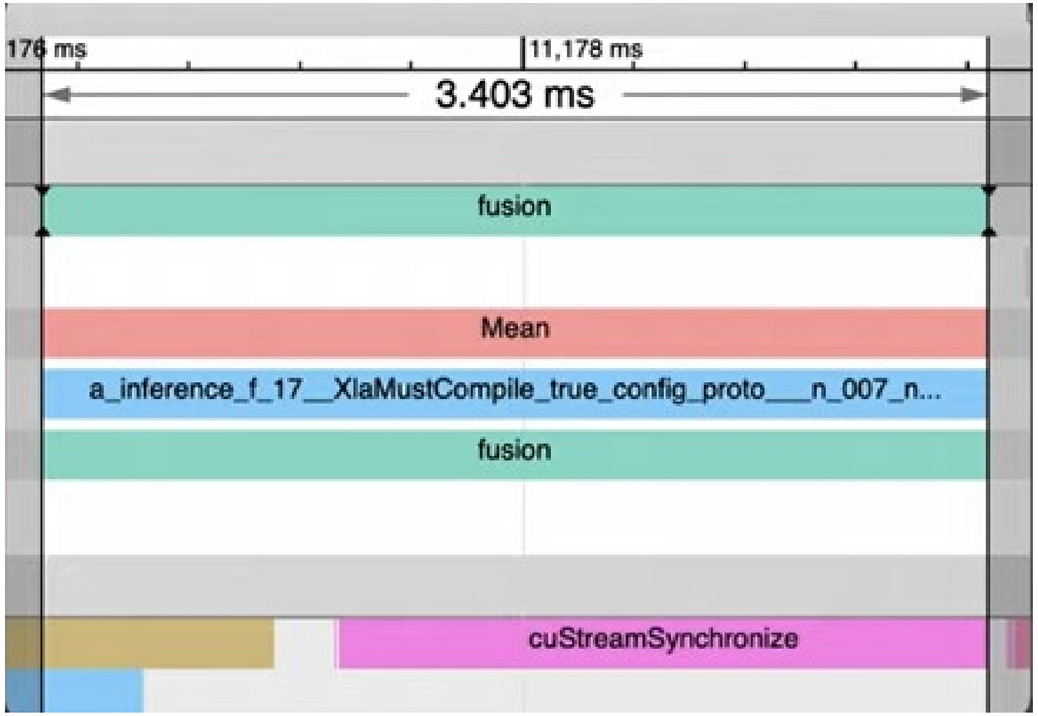



```
@tf.function(jit_compile=True)
def running_example(x, y):
  return tf.reduce_mean(tf.multiply(x ** 2, 3) + y)
```

Credit: George Karpenkov

Compiled Example with XLA: fused kernel, ~4x Speed-up

```
@tf.function(jit_compile=True)
def running_example(x, y):
  return tf.reduce_mean(tf.multiply(x ** 2, 3) + y)
```



Credit: George Karpenkov

XLA Usage Examples in TF

- Expensive Computation
- Entire Training Loop
- Distributed Training Loop using TF distributed strategy
- Distributed Training Loop with uncompilable collectives

Credit: George Karpenkov

Expensive Computation

```
@tf.function(jit_compile=True)
def entry(a, b, c, d):
    return very_expensive_computation(a, b, c, d)
```

Credit: George Karpenkov

MNIST: Entire Training Loop compiled

```
@tf.function(jit_compile=True)
def train_mnist(images, labels):
    images, labels = cast(images, labels)

    with tf.GradientTape() as tape:
      predicted_labels = layer(images)
      loss = tf.reduce_mean(my_layer(
          logits=predicted_labels, labels=labels
      ))
    layer_variables = layer.trainable_variables
    grads = tape.gradient(loss, layer_variables)
    optimizer.apply_gradients(zip(grads, layer_variables))
```

Credit: George Karpenkov

Collective Training Step

```
# Compilation is only supported for MirroredStrategy
strategy = tf.distribute.MirroredStrategy()

@tf.function(jit_compile=True)
def train_step(inputs):
  loss = ...
  return loss

per_replica_loss = strategy.run(train_step, args=...)
```

Credit: George Karpenkov

Uncompiled Collectives (e.g. using Horovod)

```
@tf.function(jit_compile=True)
def compiled_step(images, labels):
  with tf.GradientTape() as tape:
    return ...

def train_step(images, labels):
  loss = compiled_step(images, labels)
  optimizer.apply_gradients(...)

@tf.function
def train_step_dist(image, labels):
  strategy.run(train_step, args=(image, labels))
```

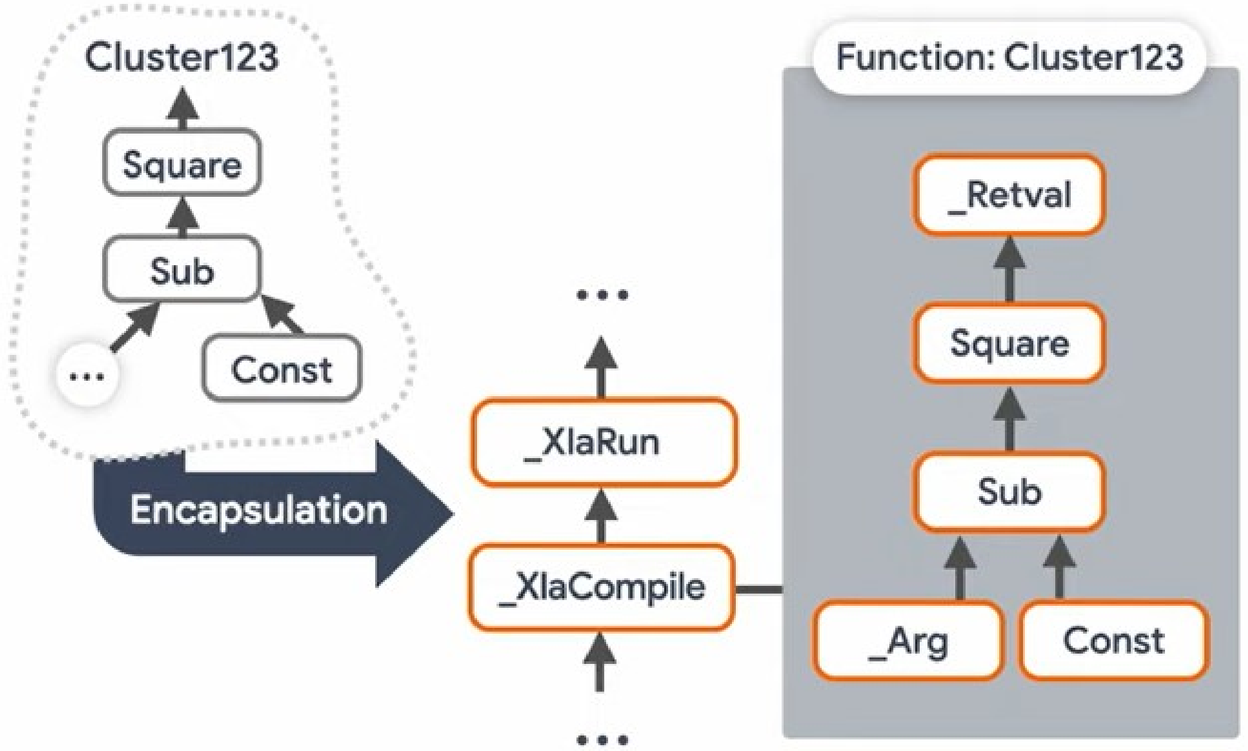
Credit: George Karpenkov

Identifying Blocks to Compile

- Can only contain compilable operations
- Rely on profiler
- Identify many smaller kernels
- Usually: larger compiled block => better performance

Autoclustering: Automatic alternative

- TF has “autoclustering” mode
  - Inside tf.function: find and compile largest clusters
  - No model changes required
- Environment Variable
  - TF_XLA_FLAGS=-tf_xla_auto_jit=2
- Or with:
  - Tf.config.optimizer.set_jit(‘autoclustering’)



Credit: George Karpenkov

Autoclustering vs. jit_compile = True

- Autoclustering:
  - +Fully automatic
  - -Hard-to-predict behavior
  - -Weird performance cliffs
- jit_compile = True
  - +Predictable behavior
  - -Might require refactoring the model code

Credit: George Karpenkov

TF/XLA Limitations

- Unsupported Operations
- Recompilation on shape changes
- Recompilation for (some) constant changes

Credit: George Karpenkov

Unsupported Operations

- TF has ~1500 operations
- ~500 of them are compilable
  - Example inherently uncompilable: image decoding ops
- Uncompilable ops inside jit_compile = True
  - Runtime Exception
  - Stack trace for op creation in the python code

Credit: George Karpenkov

Unsupported Operations Exceptions

```
@tf.function(jit_compile=True)
def load(image_file):
  image = tf.io.read_file(image_file)
  return tf.image.decode_jpg(image)
load('/tmp/blah.jpg')

> InvalidArgumentError: Function is not compilable: ...
> File "load.py", line 7, in <module>
>   print(load('/tmp/blah.jpg'))
> File "load.py", line 5, in f
>   return tf.io.read_file(image_file) [Op:__inference_f_9]
```

Credit: George Karpenkov

Static Shapes

- XLA compiler requires shapes to be fixed
- Recompiles on different shapes
- Can add unexpected latency

```
@tf.function(jit_compile=True)
def cumsum(x, y): ...

f(tf.ones([10, 10], tf.ones([10, 10]))
f(tf.zeros([10, 10], tf.zeros([10, 10]) // Cache hit
f(tf.ones([100, 100], tf.ones([100, 100])) // Recompilation
```

Credit: George Karpenkov

Must-be-constant Arguments

- XLA: certain values need to be constant at compilation
  - E.g. “axis” of argmax
- Requires recompilation when these change

```
@tf.function(jit_compile=True)
def f(a, axis):
  return tf.math.argmax(a, axis=axis)

f(tf.random.normal([10, 10]), 0)
f(tf.random.normal([10, 10]), 0) // Cache hit
f(tf.random.normal([10, 10]), 1) // Requires recompilation
```

Credit: George Karpenkov

Inspecting Compiled HLO

- Inspect generated HLO/PTX/LLVM IR
  - E.g. verify that fusions have happened
  - Or that aliasing took place
  - Or to file bugs…
- Environment variable
  - XLA_FLAGS=--xla_dump_to=/my/dump/dir
- API: f.experimental_get_compiler_ir

Credit: George Karpenkov

Inspecting the Genertaed HLO using experimental_get_compiler_ir API

```
Inspecting the Generated HLO using experimental_get_compiler_ir API

@tf.function(jit_compile=True)
def f(x, y): ...

# Generated HLO:
print(f.experimental_get_compiler_ir(x, y)(stage='hlo'))

# To see the optimizations:
f.experimental_get_compiler_ir(x, y)(stage='optimized_hlo')

# To plot (use Graphviz on output):
f.experimental_get_compiler_ir(x, y)(
    stage='optimized_hlo_dot')
```

Credit: George Karpenkov

Conclusion

- Compilation can greatly improve the performance
- Annotate with tf.function(jit_compile = True)
- Ideally, compile the largest cluster
- Performance Boost

Credit: George Karpenkov

Thank You