Tracing and profiling dataflow applications

Pierre Zins, Michel Dagenais

May, 2018

Polytechnique Montréal

Laboratoire **DORSAL**

Agenda

- Introduction
- TensorFlow with GPU
- Proposed tool
- Results
- Conclusion and future work

Introduction

- Traditional CPUs get support from co-processing units to speedup specific tasks
- Highly parallel processing units
- Applications : graphic, network, signal processing, …
- Dataflow model

- TensorFlow with GPU : an easier case to start
  - Dataflow model
  - Computation graph

TensorFlow™

Research goals :

- Propose a profiling tool for TensorFlow
- Better understand the execution of the graph and its performance
- Insure that the GPU is used efficiently
- Detect problems or suggest optimizations

TensorFlow with GPU

- Official TensorFlow : CUDA
  - TensorFlow 1.8
  - *https://github.com/tensorflow/tensorflow*

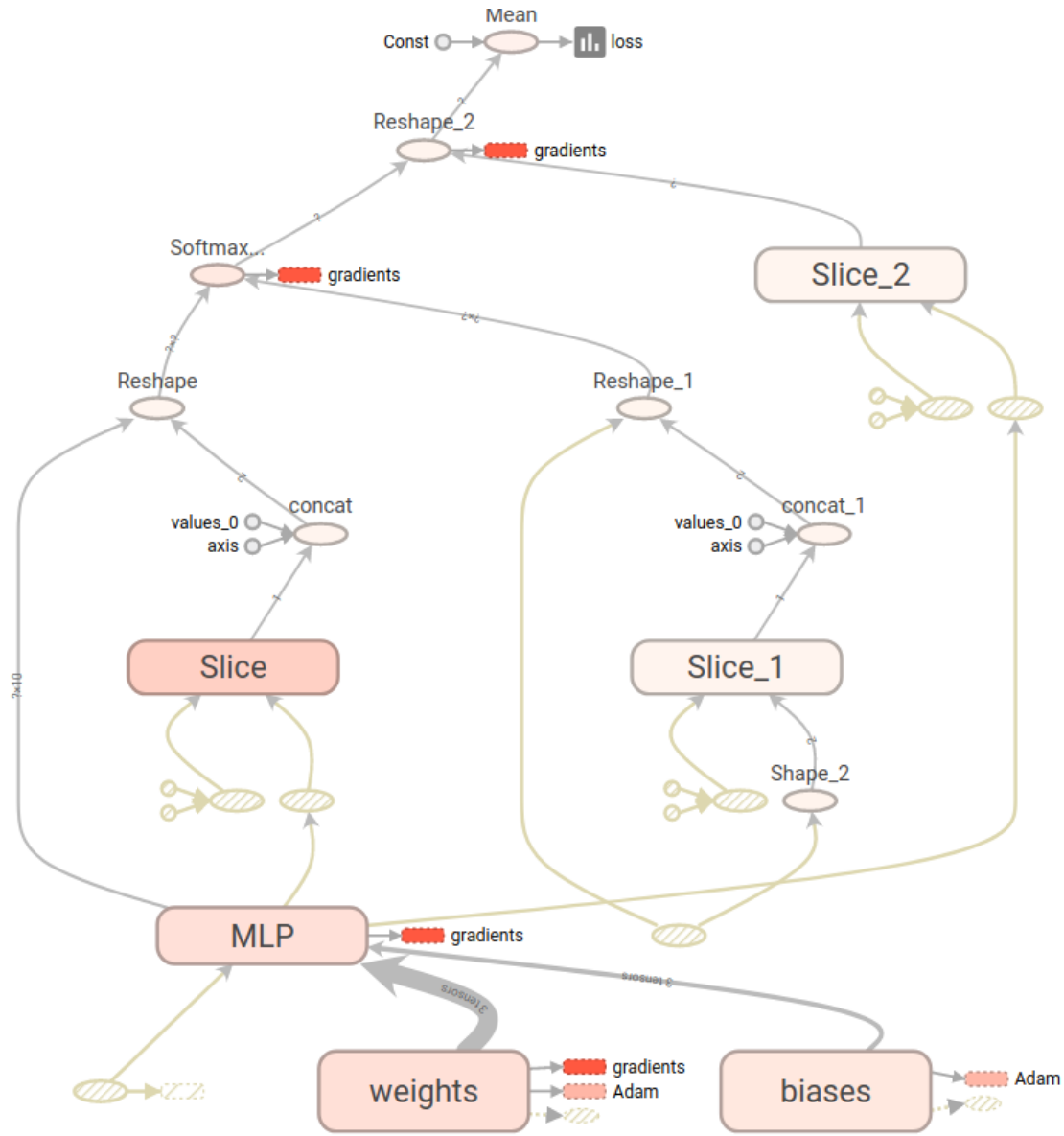
- AMD : TensorFlow : ROCm platform + HIP
  - HipTensorFlow 1.0.1 : ROCm 1.6 (deprecated)
  - *https://github.com/parallelo/hiptensorflow-1*
  - TensorFlow 1.3 : ROCm 1.7
  - *https://github.com/ROCmSoftwarePlatform/tensorflow*

- TensorFlow : SYCL
  - TensorFlow 1.6
  - *https://github.com/lukeiwanski/tensorflow*

TensorFlow instrumentation

Main elements :

- Sessions
- Operations :
  - Synchronous
  - Asynchronous
  - GPU
  - CPU
- Scheduling
- Rendez-vous
- GRPC
- Memory :
  - Allocations / Deallocations
  - Transfers



NVIDIA GPUs

Device Tracer inside TensorFlow

CUPTI library, Activity API

Two callbacks

- API Callback invoked synchronously on the thread making the API Call
  - Mapping : Kernel Name ↔ API Call ↔ TensorFlow operation
  - Correlation ID

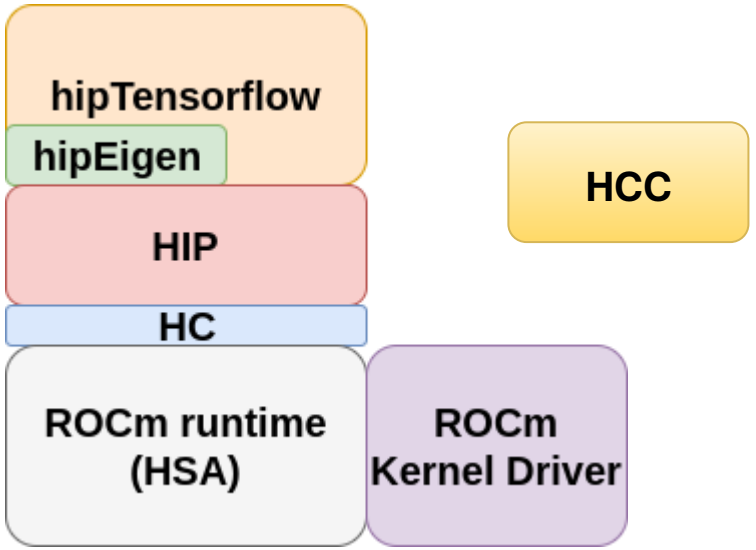
- Activity callback for asynchronous events
  - GPU kernels
  - Asynchronous memory copies
  - CUDA Runtime API
  - CUDA Driver API

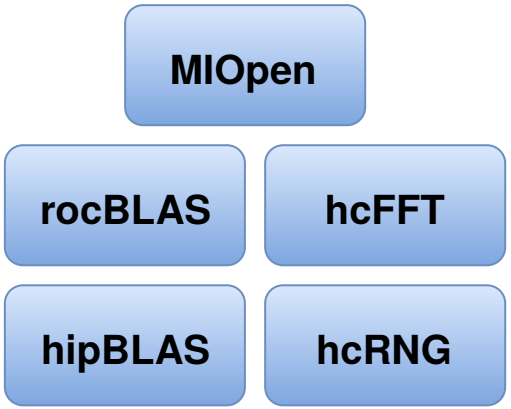
AMD GPU - ROCm

Port of TensorFlow on the ROCm platform

- HSA API
  - ROCm runtime instrumentation
  - Interception libraries based on Paul Margheritta's work

- HIP API
  - Replacing AMD Markers with LTTng tracepoints



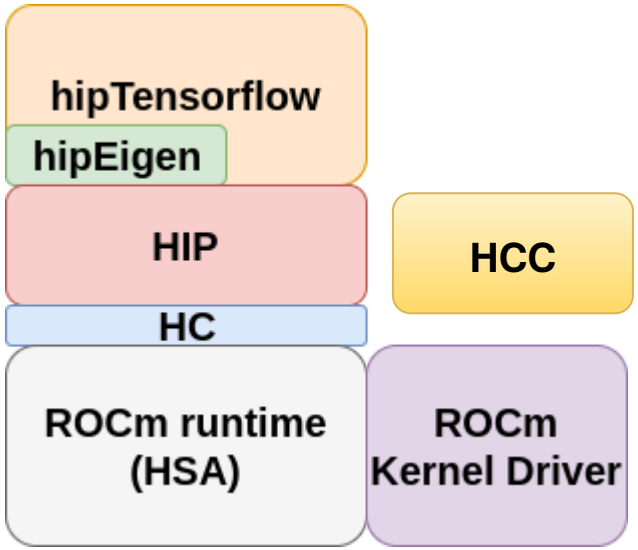


AMD GPU - ROCm

Asynchronous events :

- HC instrumentation
  - *hsa_amd_profiling_get_dispatch_time, hsa_amd_profiling_get_async_copy_time*
  - Use signals
  - Add tracepoints into asynchronous events destructors
  - Collected during the execution

- AMD profiling functions + interception
  - *hsa_ext_tools_queue_create_profiled*
  - *hsa_ext_tools_get_kernel_times*
  - Only for kernels
  - Collected at the exit of the application



AMD GPU - SYCL

SYCL API

- Highly templated
- Library : closed source
- Profiling functions : limited

OpenCL API

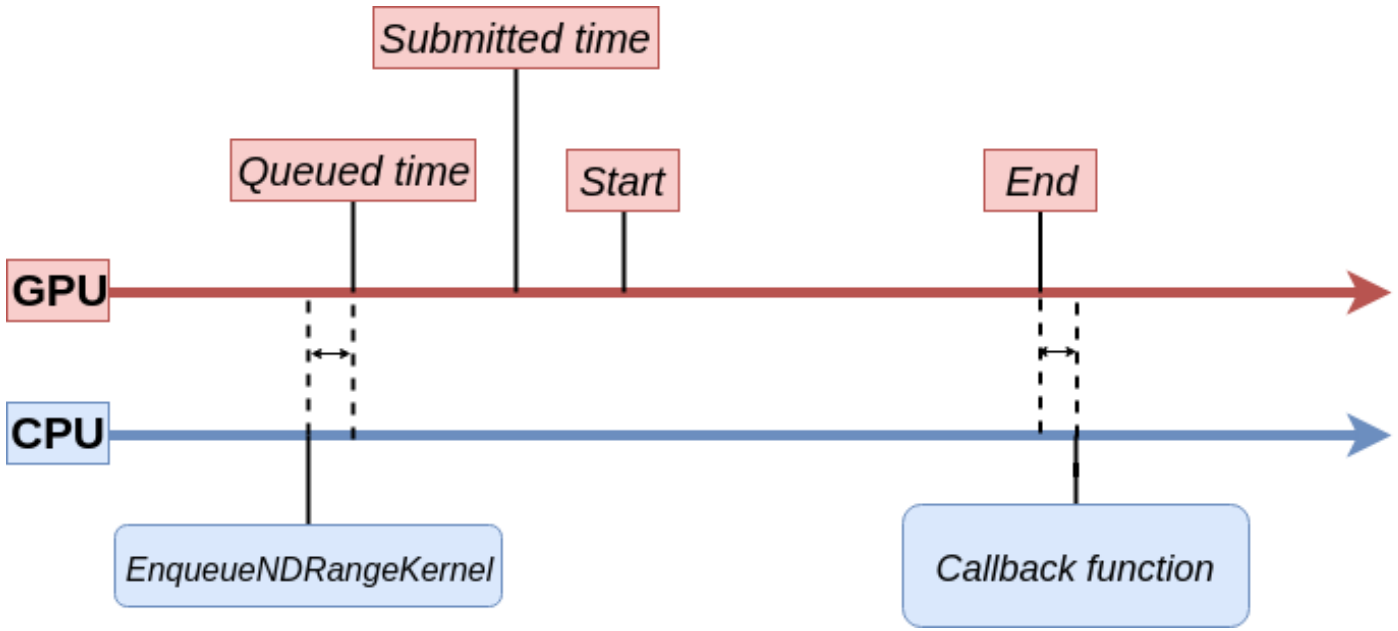
- Based on CLUST by David Couturier
- Interception mechanisms
  - ClCreateCommandQueue : CL_QUEUE_PROFILING_
  - clGetKernelInfo
  - clSetEventCallback
  - clGetEventProfilingInfo

Post processing

Kernels, barriers, memory copies are collected asynchronously

All GPU-related events should use the same clock as the CPU :

- AMD profiling functions and CUPTI functions : Monotonic clock
- OpenCL : device clock has no relation with the CPU clock
  - Convex Hull synchronization algorithm by Poirer et al. (2010)
  - More efficient improvement proposed by Jabbarifar (2013)

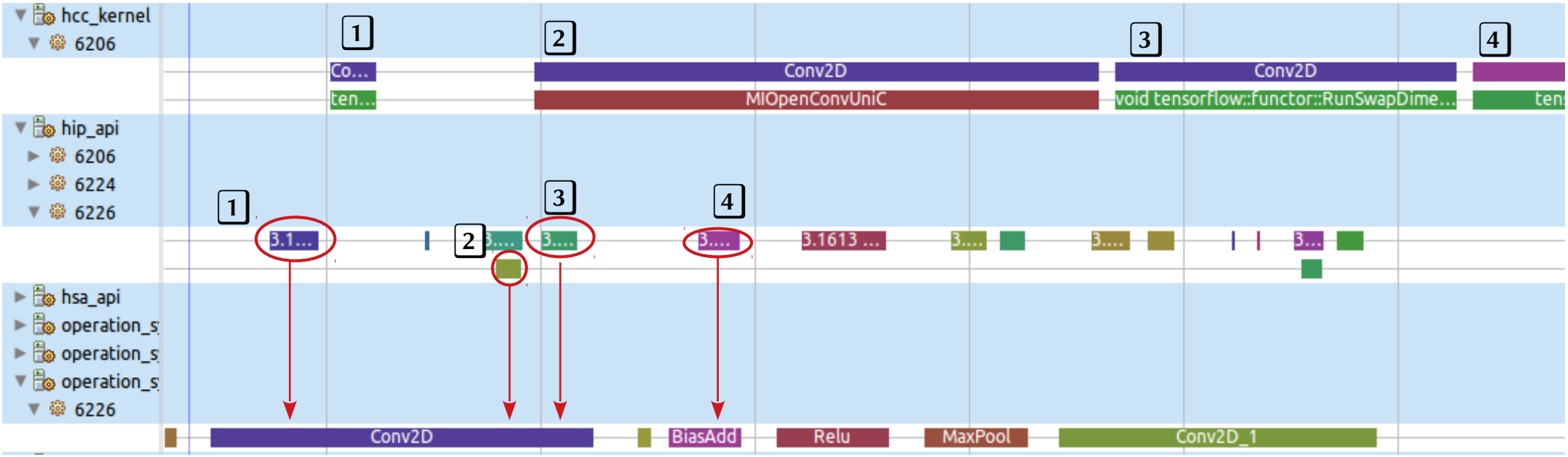


Post processing

Link GPU kernels with TensorFlow operations

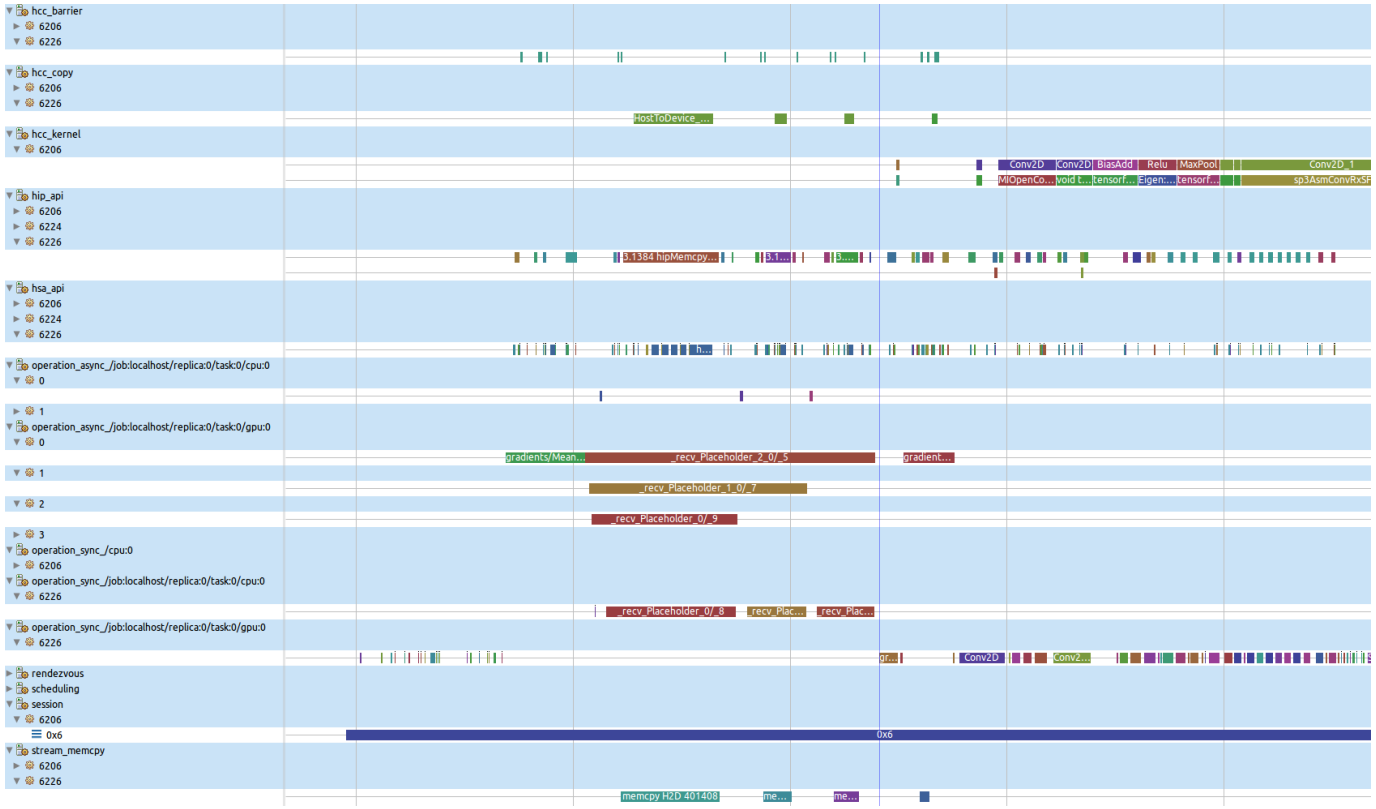
- NVIDIA : correlation ID
- HIP or SYCL : no correlation ID linking the API function to the GPU kernel

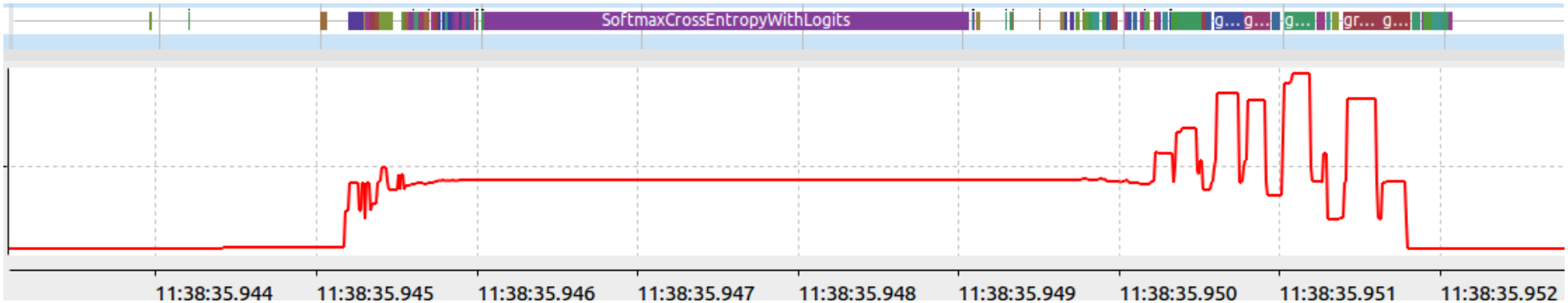
- TensorFlow only use 1 Compute Stream/Queue
- Within a Stream/Queue, kernels are executed in-order



HipLaunchKernel

XML Analysis

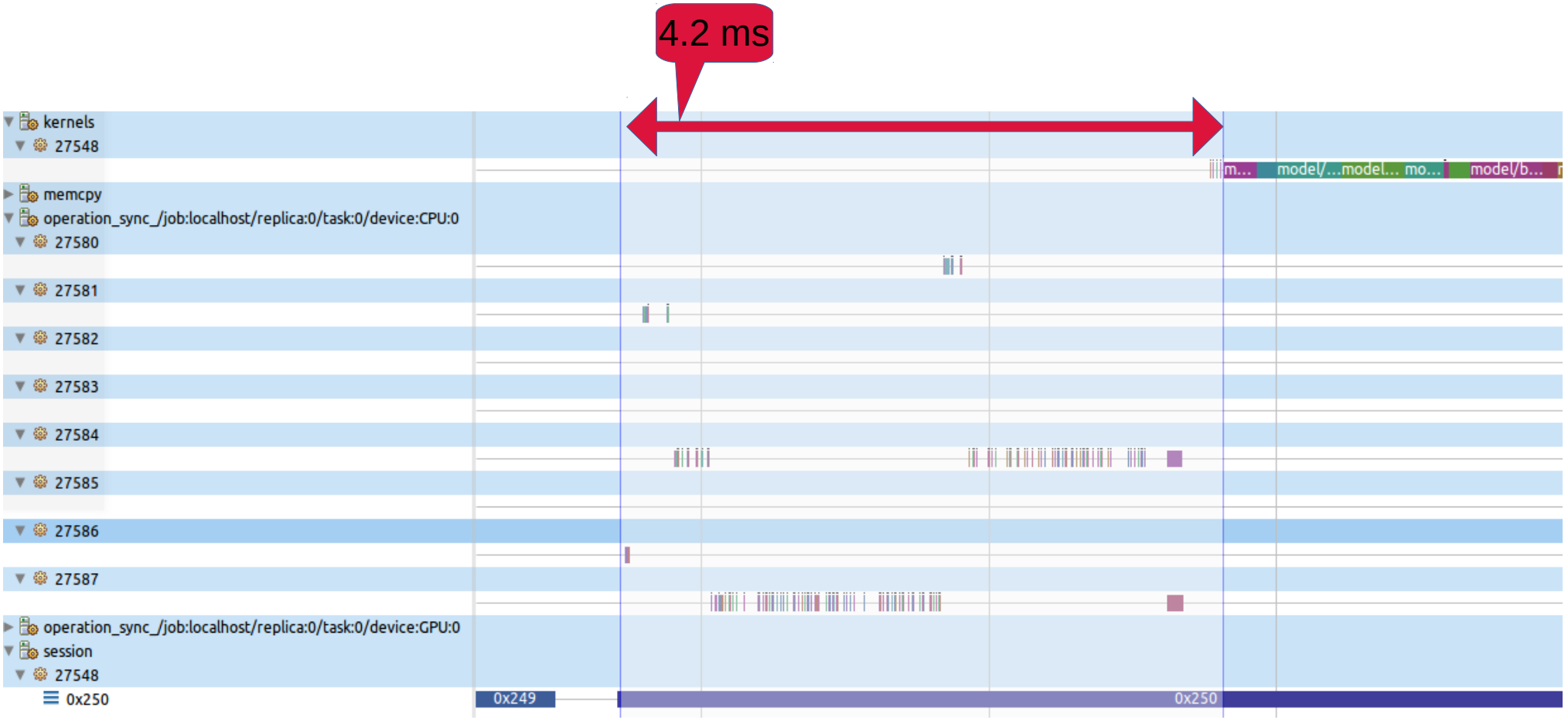




Example – Triplet loss

Triplet loss used in face recognition

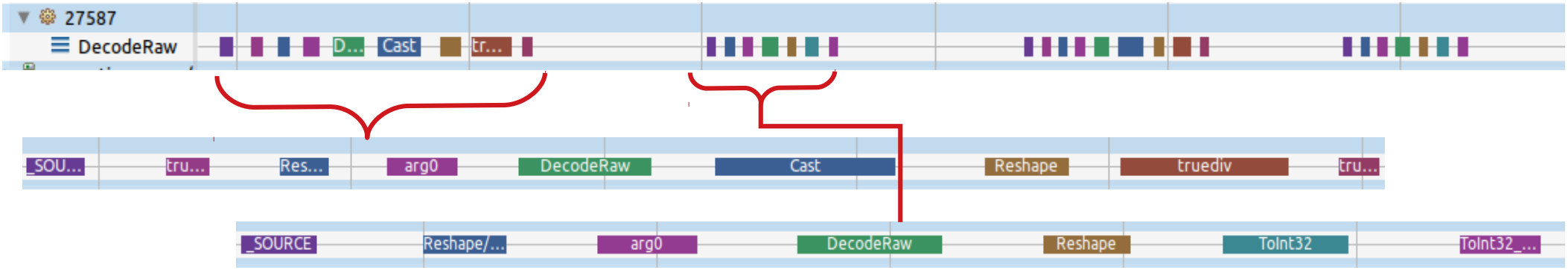
Olivier Moindrot :
https://github.com/omoindrot/tensorflow-triplet-loss



Example – Triplet loss

Triplet loss used in face recognition

Olivier Moindrot :
https://github.com/omoindrot/tensorflow-triplet-loss



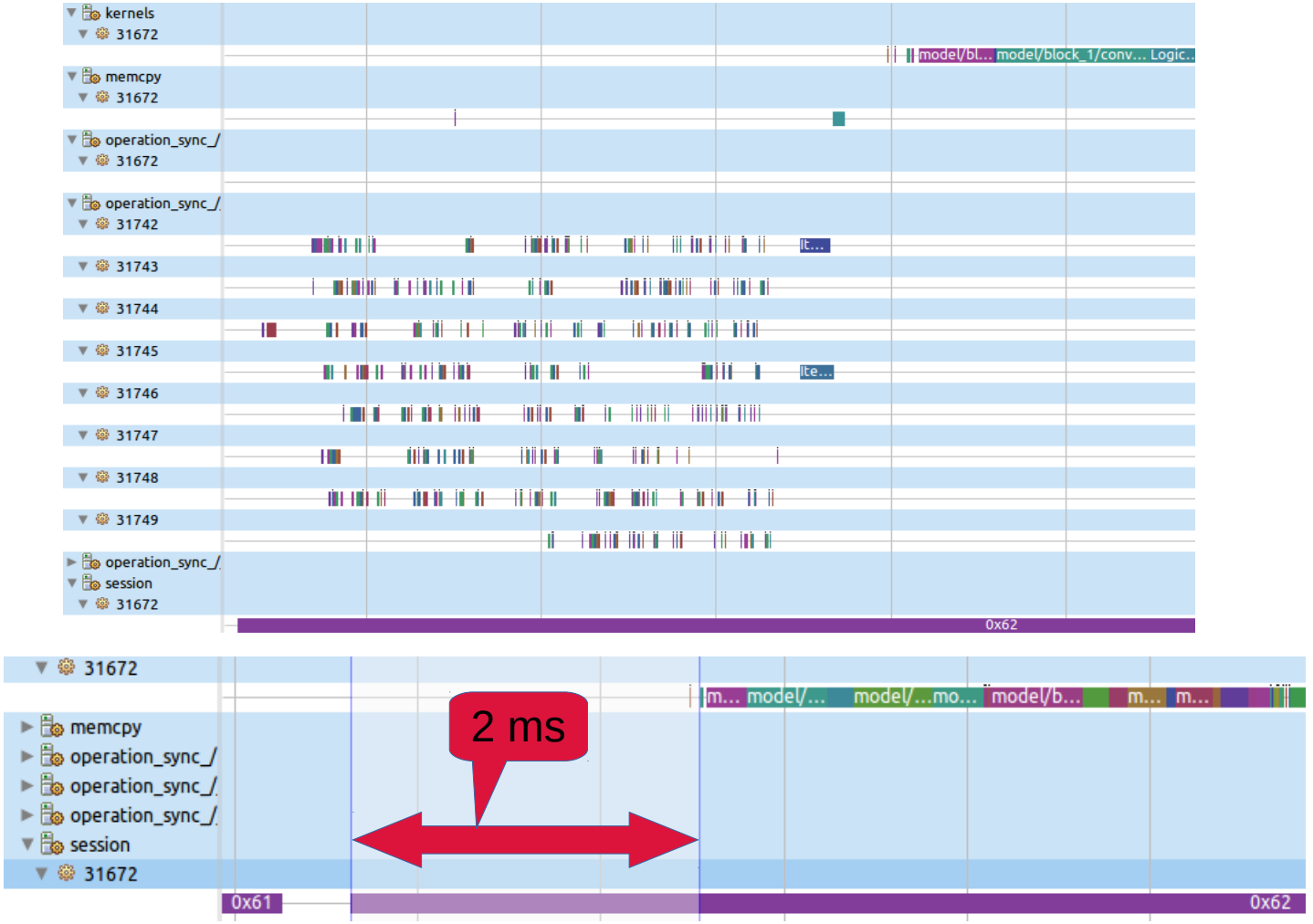

```python
def decode_image(image):
    # Normalize from [0, 255] to [0.0, 1.0]
    image = tf.decode_raw(image, tf.uint8)
    image = tf.cast(image, tf.float32)
    image = tf.reshape(image, [784])
    return image / 255.0

def decode_label(label):
    label = tf.decode_raw(label, tf.uint8)  # tf.string -> [tf.uint8]
    label = tf.reshape(label, [])  # label is a scalar
    return tf.to_int32(label)

images = tf.data.FixedLengthRecordDataset(images_file, 28 * 28, header_bytes=16)
images = images.map(decode_image)
labels = tf.data.FixedLengthRecordDataset(labels_file, 1, header_bytes=8)
labels = labels.map(decode_label)

return tf.data.Dataset.zip((images, labels))
```

Example – Triplet loss

Triplet loss used in face recognition

Olivier Moindrot : https://github.com/omoindrot/tensorflow-triplet-loss

```
images = tf.data.FixedLengthRecordDataset(images_file, 28 * 28, header_bytes=16)
images = images.map(decode_image, num_parallel_calls=16)
labels = tf.data.FixedLengthRecordDataset(labels_file, 1, header_bytes=8)
labels = labels.map(decode_label, num_parallel_calls=16)
```



Example – Triplet loss

Triplet loss used in face recognition

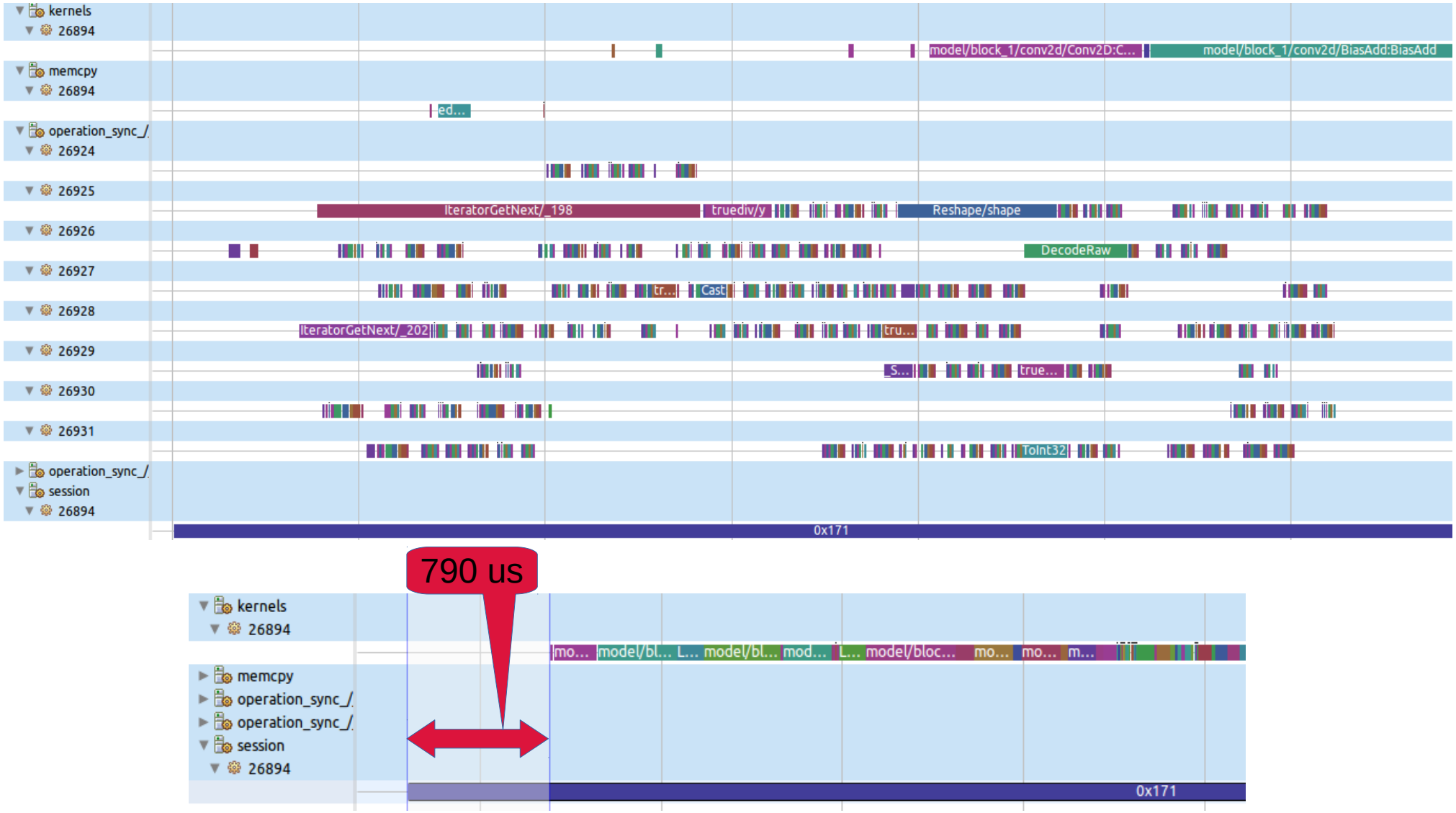
Olivier Moindrot :
https://github.com/omoindrot/tensorflow-triplet-loss

```python
def train_input_fn(data_dir, params):
    """Train input function for the MNIST dataset.

    Args:
        data_dir: (string) path to the data directory
        params: (Params) contains hyperparameters of the model (ex: `params.num_epochs`)
    """
    dataset = mnist_dataset.train(data_dir)
    dataset = dataset.shuffle(params.train_size)  # whole dataset into the buffer
    dataset = dataset.repeat(params.num_epochs)  # repeat for multiple epochs
    dataset = dataset.batch(params.batch_size)

    dataset = dataset.prefetch(1)  # make sure you always have one batch ready to serve

    return dataset
```

Example – Triplet loss

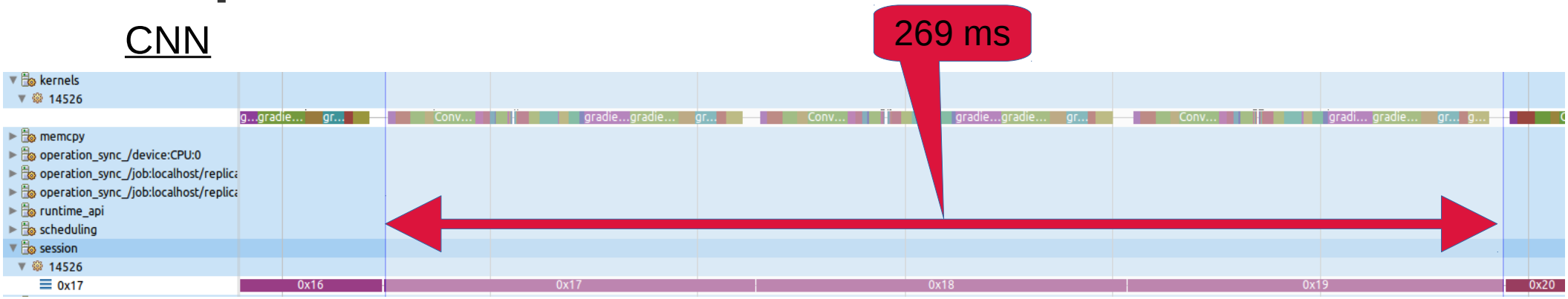
Triplet loss used in face recognition

Olivier Moindrot :
https://github.com/omoindrot/tensorflow-triplet-loss



Example – Kernel fusion

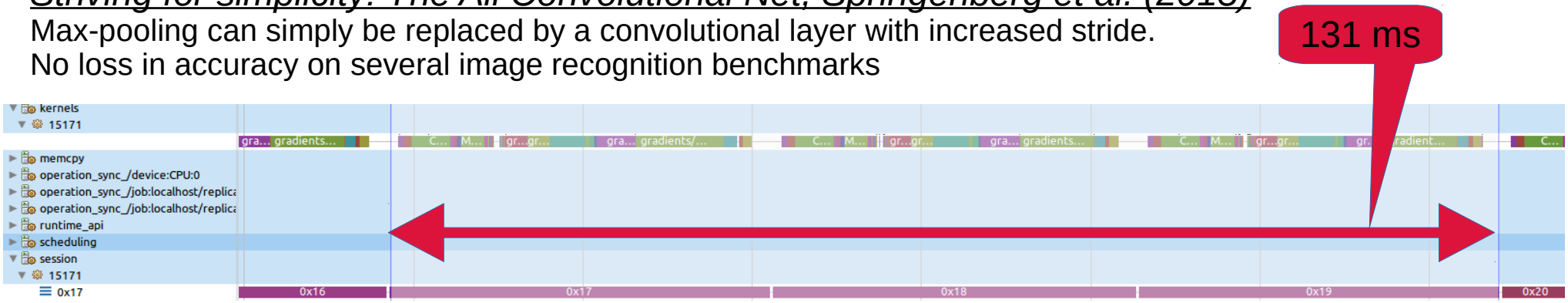
CNN



1.5256344541203E+018	1.52563445413805E+018	Conv2D_1:Conv2D	17757000	14
1.52563445314211E+018	1.52563445315802E+018	gradients/Conv2D_1_grad/Conv2DBackpropInput:Conv2DBackpropInput	15917000	3
1.52563445392312E+018	1.52563445393832E+018	Conv2D_1:Conv2D	15204000	12
...				
1.52563445411306E+018	1.52563445411748E+018	Relu:Relu	4421000	14
1.5256344547475E+018	1.52563445475182E+018	BiasAdd:BiasAdd	4327000	21
1.52563445334996E+018	1.52563445335402E+018	gradients/MaxPool_grad/MaxPoolGrad:MaxPoolGrad	4058000	5
1.5256344532605E+018	1.52563445326456E+018	gradients/MaxPool_grad/MaxPoolGrad:MaxPoolGrad	4057000	4
...				

Striving for simplicity: The All Convolutional Net, Springenberg et al. (2015)

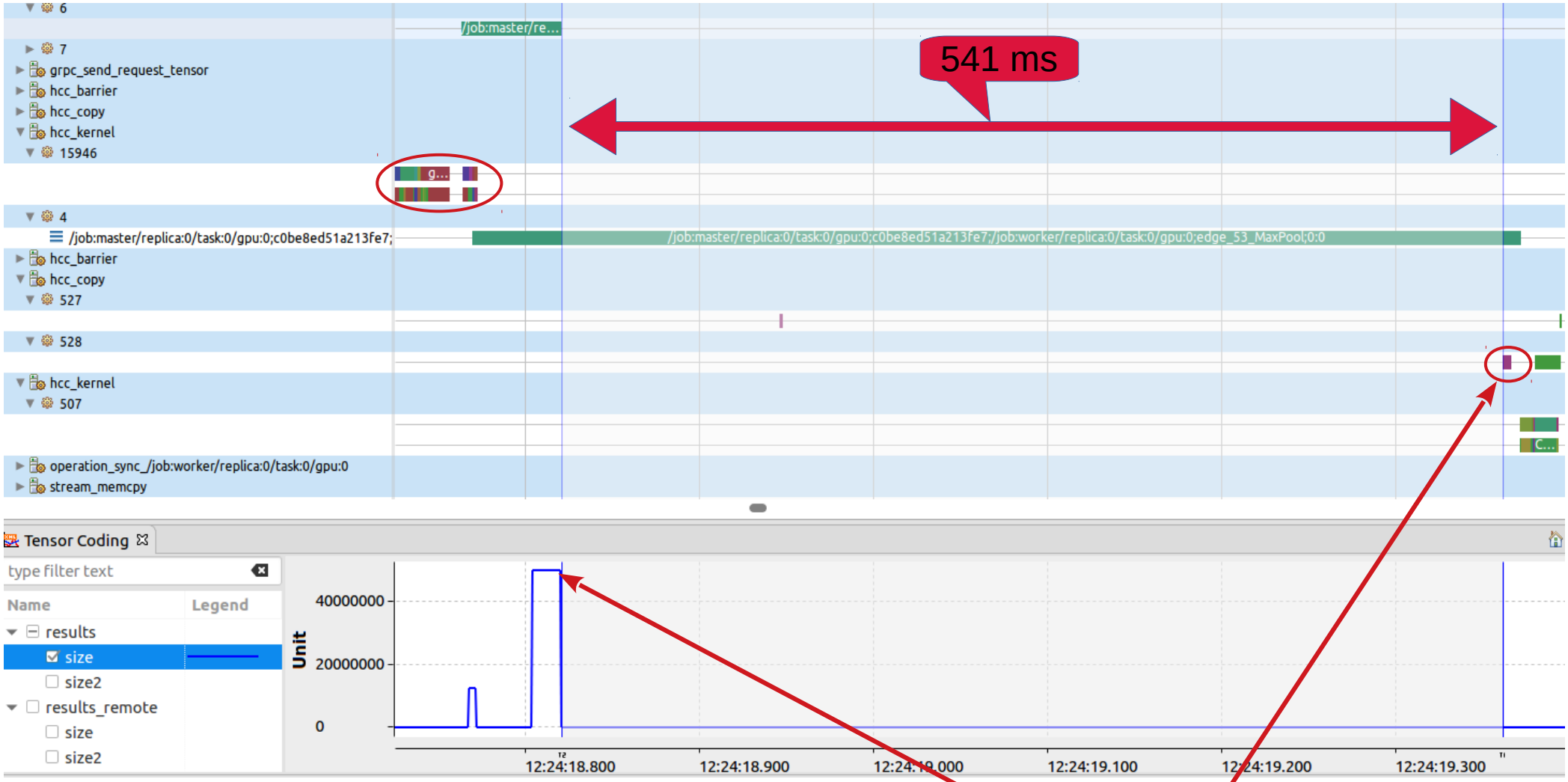
Max-pooling can simply be replaced by a convolutional layer with increased stride.
No loss in accuracy on several image recognition benchmarks



Example – Distributed CNN

```
def conv_net(x, weights, biases, dropout):
    # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
    # Reshape to match picture format [Height x Width x Channel]
    # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
    x = tf.reshape(x, shape=[-1, 28, 28, 1])
    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)
    # Convolution Layer
    with tf.device("/job:worker/task:0"):
        conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
        # Max Pooling (down-sampling)
        conv2 = maxpool2d(conv2, k=2)
```

Example – Distributed CNN



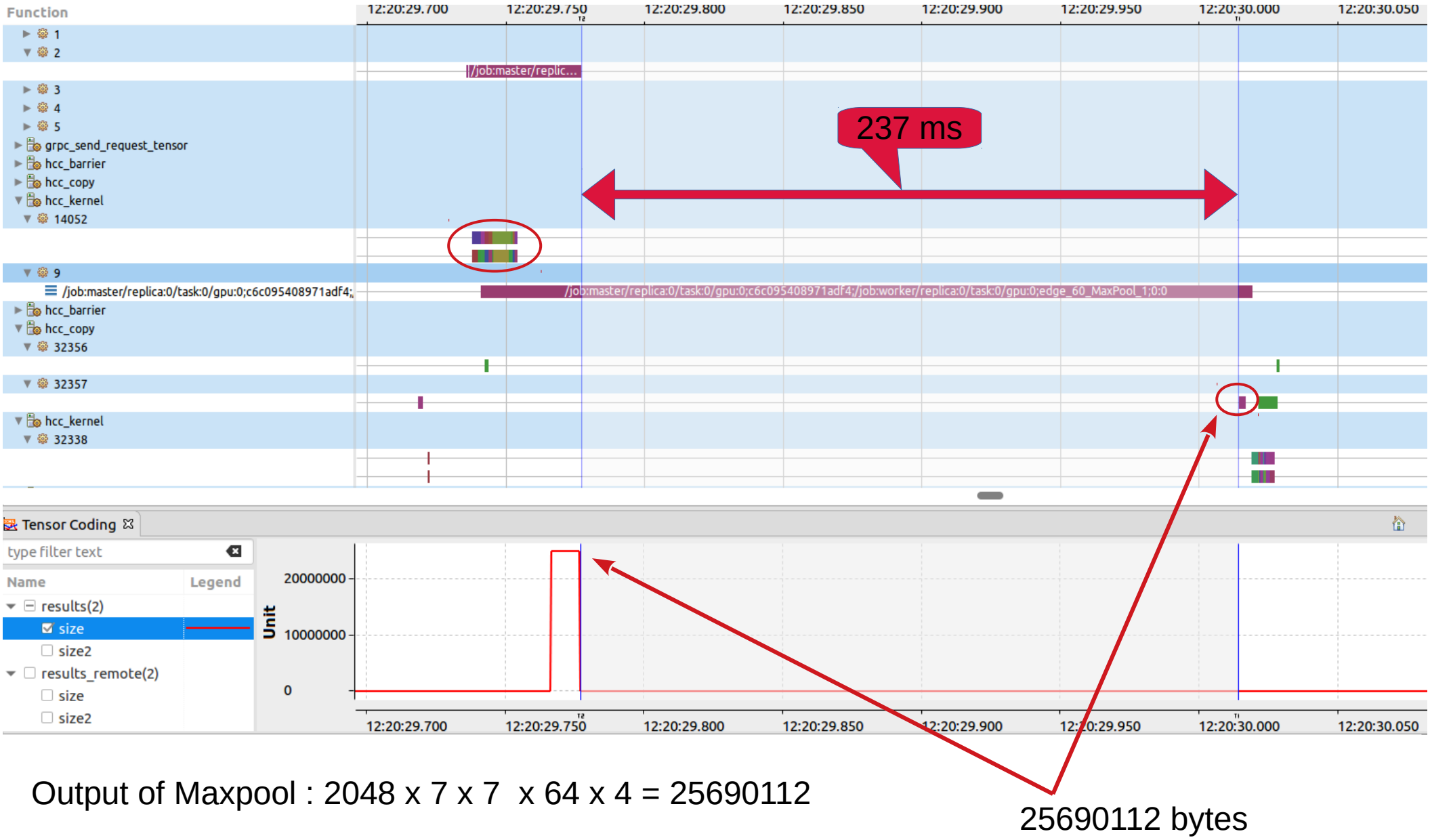
Output of Maxpool : 2048 x 14 x 14 x 32 x 4 = 51380224

51380224 bytes

Example – Distributed CNN

```
# Create model
def conv_net(x, weights, biases, dropout):
    # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
    # Reshape to match picture format [Height x Width x Channel]
    # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    with tf.device("/job:worker/task:0"):
        fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
        fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
```

Example – Distributed CNN



Output of Maxpool : 2048 x 7 x 7 x 64 x 4 = 25690112

Conclusion

- Proposed a profiling tool / environment for TensorFlow
- Compatible with different TensorFlow versions depending on the GPU vendor
- Can help TensorFlow users to profile their application and improve it
- Can help TensorFlow developers (scheduling, performance counters, kernel traces, ...)

Future work

- Distributed TensorFlow : between model parallelism and “parameter server” analysis
- High level API support : Keras or Horovod
- Extend TensorFlow instrumentation
- How to generalize this workflow and analysis to other dataflow applications using co-processing units ?

Thank you!

Questions?

https://github.com/pzins