IXPUG Annual Meeting 2020

Characterizing Machine Learning Workloads with the Intel® VTune™ Profiler and TensorFlow*

Christopher Lishka, Intel Corporation, October 2020

Motivation

- TensorFlow* (TF) + TensorBoard* (TB): popular method for ML/DL profiling
  - The TensorFlow (TF) framework is implemented as a Python* package
  - TensorFlow includes a compiler which compiles the model's computational graph to TF-Ops
  - TF-Ops run on the hardware using kernels from libraries
  - TB generally profiles at the model and TF-Ops level, with some hardware profiling for GPU
- Intel® VTune™ Profiler: popular and deep profiler for Intel CPU hardware
  - How can we enhance TensorBoard 2.x profiling with VTune for hardware insights?
- VTune and TensorBoard have learning curves
  - How can we quickly get going with both, preferably in a single combined run?

Installing and Running

1. Install VTune 2021.1-beta09

Download and run OneAPI Beta09 installer from software.intel.com (see below)

2. Install TensorFlow in Conda Env

Install miniconda or anaconda

```
$ conda create -n py37-tf22-mkl python=3.7
$ conda activate py37-tf22-mkl
$ conda install tensorflow=2.2  # Look for mkl
$ pip install -U tensorboard_plugin_profile
```

3. Copy Scripts from Backup Slides

Python model scripts with profiling additions:

mnist_dense_train_PROFILE.py,
mnist_infer_PROFILE.py

Shell scripts to run VTune (edit paths):

run_vtune_train.sh, run_vtune_infer.sh

4. Run training (without VTune)

```
$ python mnist_dense_train_PROFILE.py
```

Creates **saved_model** directory

5. Run inference with VTune

```
$ ./run_vtune_infer.sh
```

TF profiling is saved to **logs-infer** directory

VTune saves to a **r000ue** directory

6. Open VTune results in GUI

On VTune's "Welcome" tab, use "Open Result..." button to open **r000ue/r000ue.vtune**

7. Run TensorBoard

```
$ tensorboard --logdir logs-infer
```

In Chrome browser, open **http://localhost:6006**

Simple Script to Run VTune on TF Workload

run_vtune_infer.sh

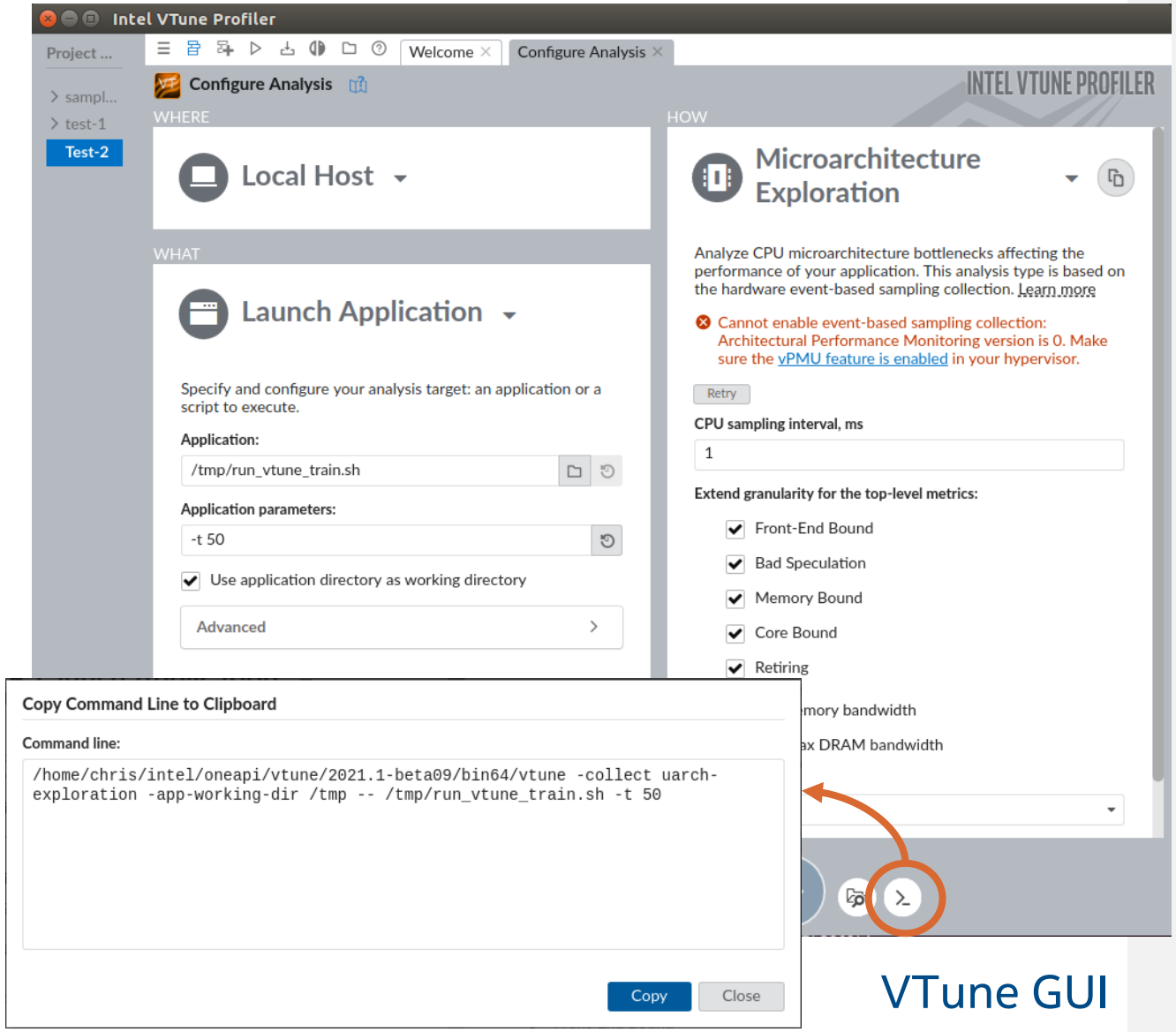
```
#! /bin/bash

RUN_DIR="${HOME}/local-projects/IXPUG-models-CURRENT"
RUN_CMD='python mnist_infer_PROFILE.py'

# Put any vtune setup commands here, like OneAPI or module loading
source /opt/intel/oneapi/setvars.sh
echo -n '===== VTune Being Used =====: ' ; which vtune ; vtune --version

# Load conda and activate environment
source "${HOME}/miniconda3/etc/profile.d/conda.sh"
conda activate py37-tf22-mkl  # Python 3.7, TF 2.2 built with MKL
echo '===== Python Being Used =====' ; which python ; python --version

# Environment variable settings - search for "Maximize TensorFlow Performance on CPU" on internet
export KMP_AFFINITY=granularity=fine,compact,1,0
export OMP_NUM_THREADS=4

cd $RUN_DIR
vtune -collect uarch-exploration -app-working-dir "${RUN_DIR}" -- ${RUN_CMD}
echo "Results can be found in ${PWD}"
```

Intel VTune Profiler

Running VTune

- GUI
  - Create project, configure analyses
  - Open previously collected *.vtune files for display

    Welcome (Tab) -> Open Result
- Command-Line
  - Easy to include in scripts
  - Produces a directory “r000ue” with “r000ue/r000ue.vtune” file (and other files)
  - Can run collection on a remote server, then display results (r000ue.vtune file) in GUI on laptop
  - Note: make sure that the VTune GUI’s build version (in Help -> About menu) is at least as high as command-line VTune’s (vtune --version)



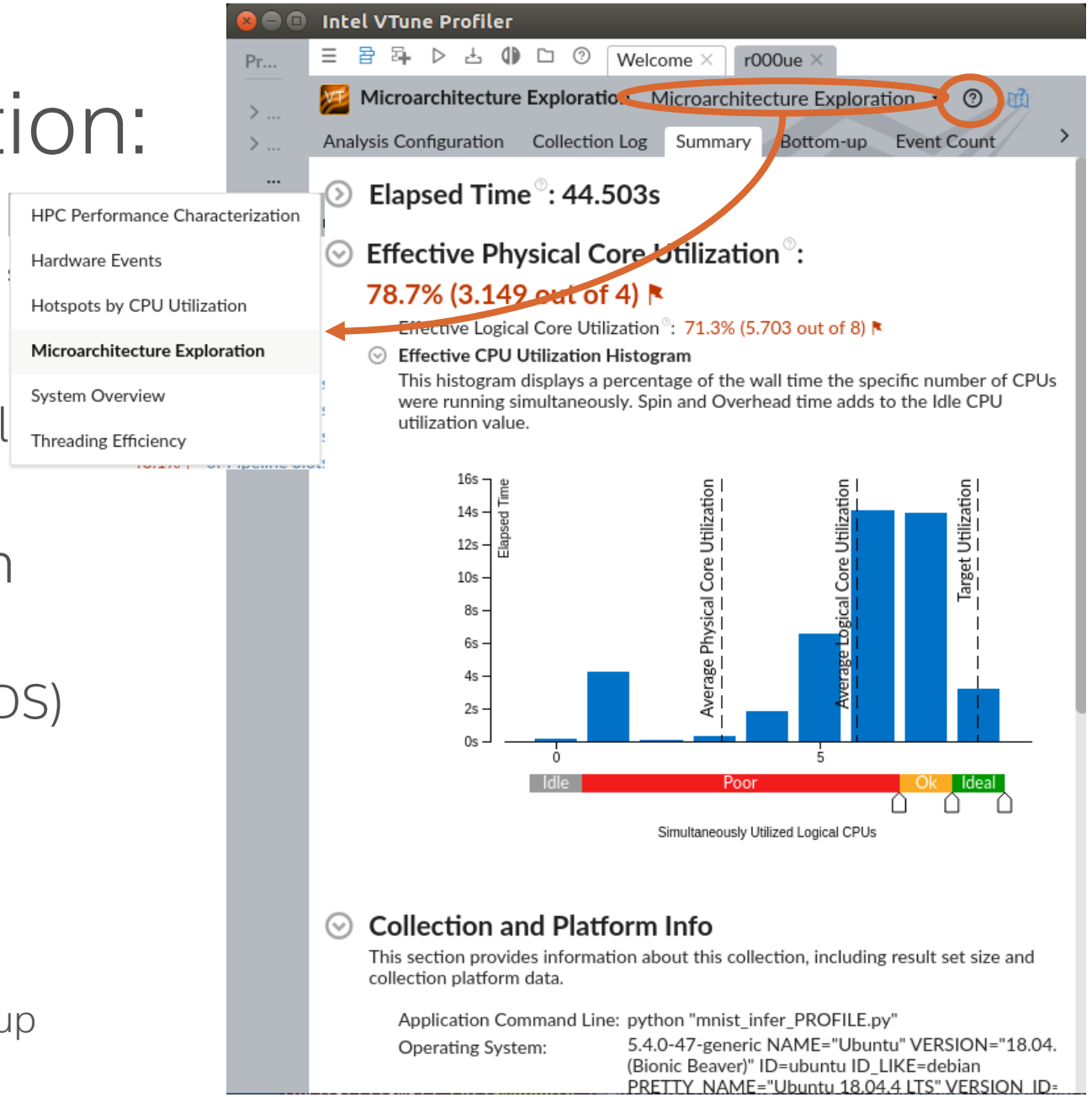
VTune GUI

VTune Command Line

```
$ source /opt/intel/oneapi/setvars.sh
$ vtune -collect uarch-exploration -app-working-dir /tmp -- python mnist_infer_PROFILE.py
```

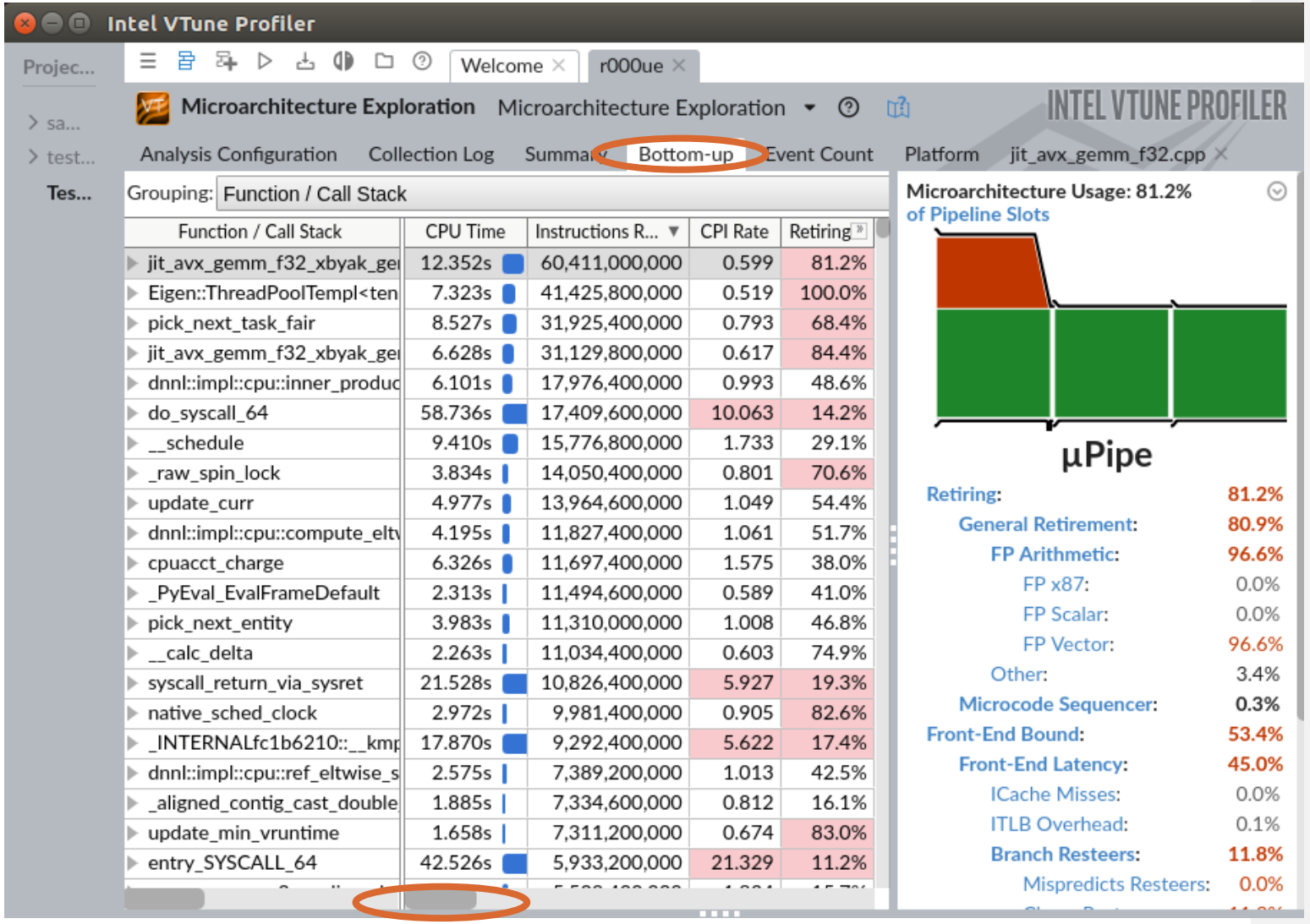
VTune µArch Exploration: Summary View

- Elapsed time shows wall-clock
  - Expanding this section shows overall µArch usage (example on next slide)
- Effective physical core utilization
  - I directed MKL to use all 4 physical cores on server (OMP_NUM_THREADS)
    - Setting sliders will apply this information to displayed data
  - 3.149 out of 4 cores used (78.7%) is decent
    - Overall, stalls/idle time likely due to TF startup and batch loading (more on this in a bit)

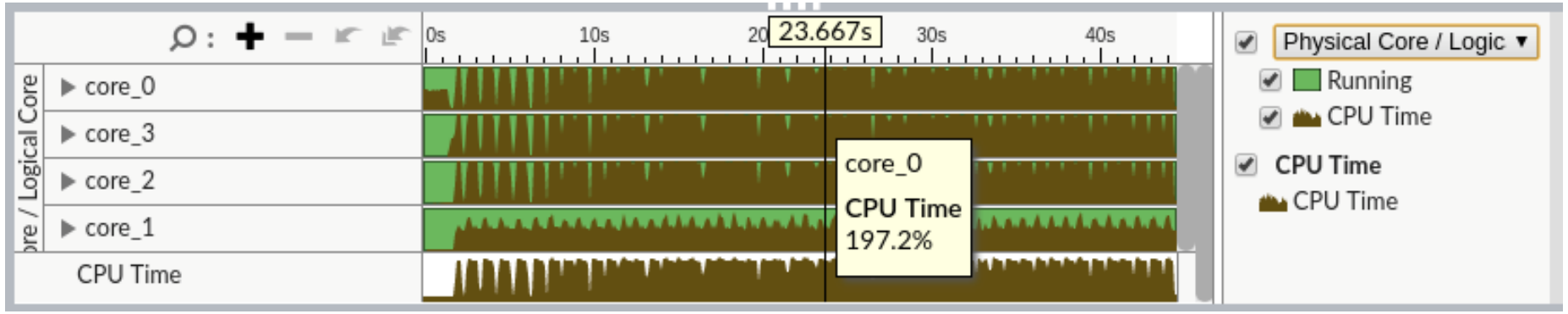


VTune µArch: Bottom-Up View

- Sorted by Instructions Retired column
  - Top represents functions which ran most instructions during VTune run
- µArch Usage view is for highlighted "jit_avx_gemm"
  - Double-clicking on name shows this is an MKL-DNN function
  - 2nd jit_avx_gemm in 4th place
  - µPipe shows areas in pipeline that VTune recommends are running well (green) and areas where optimization might help (red)
- Can hover over many fields and µPipe for explanations and recommendations from VTune



VTune µArch Exploration: Bottom-Up → CPU Core Utilization

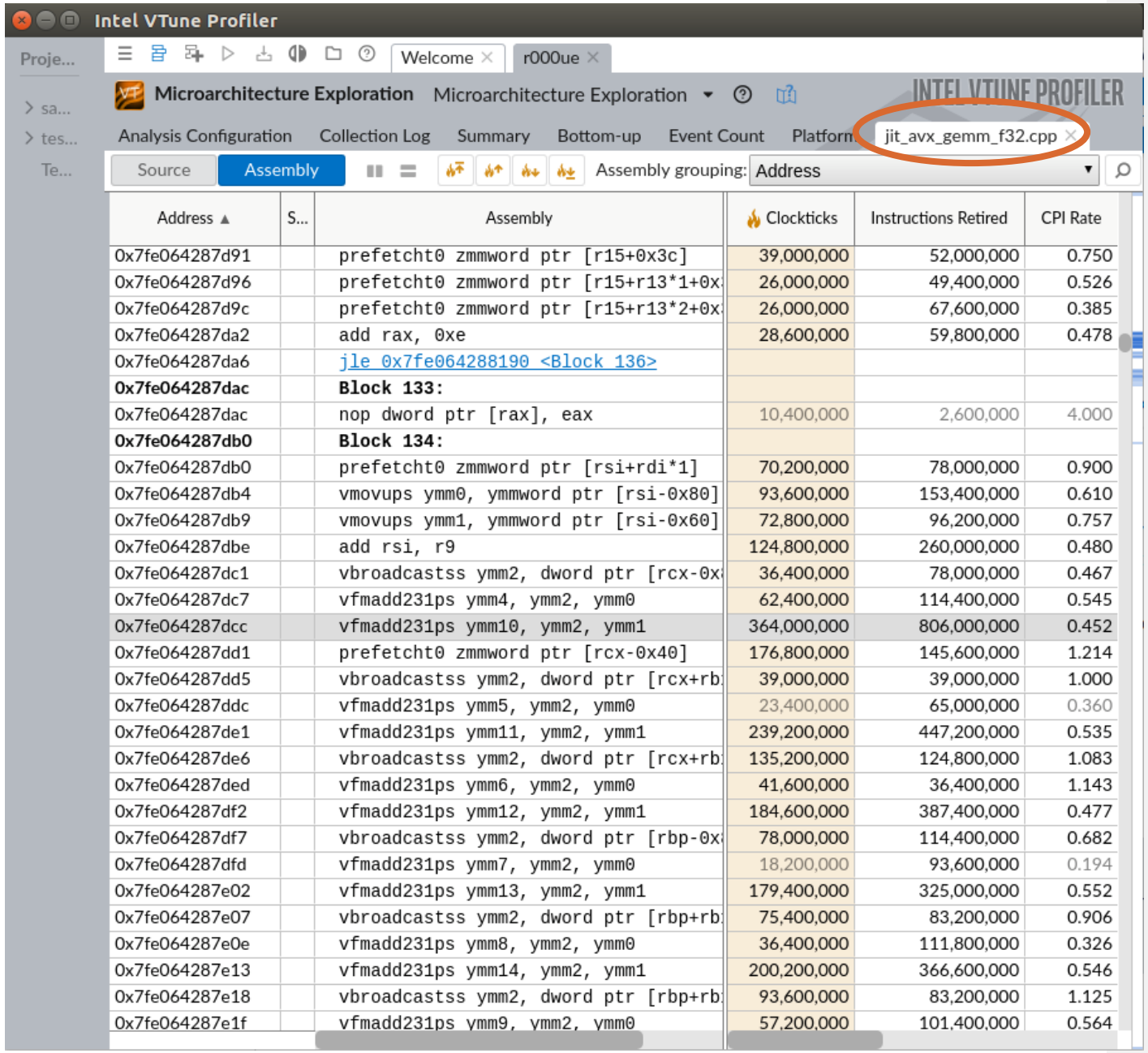


- Displayed on Bottom-Up Tab
- Can display by threads, cores, and processes
- **+** and **–** allow for zooming in/out
- Platform tab provides larger view

- Inference runs are characterized by:
  1. TF startup
  2. Loading each batch
  3. Predicting images in batch
- This delayed see-saw pattern can be seen in the CPU utilization

VTune µArch: Assembly View

- Double-clicking a function will open a Source/Assembly View
- For individual instructions, can see:
  - Clockticks run
  - Instructions retired
  - Cycles Per Instruction (CPI)
  - Further columns (not shown here) for front-end latency, bad speculation, and other scheduling information



Address ▲	S...	Assembly	Clockticks	Instructions Retired	CPI Rate
0x7fe064287d91		prefetcht0 zmmword ptr [r15+0x3c]	39,000,000	52,000,000	0.750
0x7fe064287d96		prefetcht0 zmmword ptr [r15+r13*1+0x	26,000,000	49,400,000	0.526
0x7fe064287d9c		prefetcht0 zmmword ptr [r15+r13*2+0x	26,000,000	67,600,000	0.385
0x7fe064287da2		add rax, 0xe	28,600,000	59,800,000	0.478
0x7fe064287da6		jle 0x7fe064288190 <Block 136>			
0x7fe064287dac		**Block 133:**			
0x7fe064287dac		nop dword ptr [rax], eax	10,400,000	2,600,000	4.000
0x7fe064287db0		**Block 134:**			
0x7fe064287db0		prefetcht0 zmmword ptr [rsi+rdi*1]	70,200,000	78,000,000	0.900
0x7fe064287db4		vmovups ymm0, ymmword ptr [rsi-0x80]	93,600,000	153,400,000	0.610
0x7fe064287db9		vmovups ymm1, ymmword ptr [rsi-0x60]	72,800,000	96,200,000	0.757
0x7fe064287dbe		add rsi, r9	124,800,000	260,000,000	0.480
0x7fe064287dc1		vbroadcastss ymm2, dword ptr [rcx-0x	36,400,000	78,000,000	0.467
0x7fe064287dc7		vfmadd231ps ymm4, ymm2, ymm0	62,400,000	114,400,000	0.545
0x7fe064287dcc		vfmadd231ps ymm10, ymm2, ymm1	364,000,000	806,000,000	0.452
0x7fe064287dd1		prefetcht0 zmmword ptr [rcx-0x40]	176,800,000	145,600,000	1.214
0x7fe064287dd5		vbroadcastss ymm2, dword ptr [rcx+rb	39,000,000	39,000,000	1.000
0x7fe064287ddc		vfmadd231ps ymm5, ymm2, ymm0	23,400,000	65,000,000	0.360
0x7fe064287de1		vfmadd231ps ymm11, ymm2, ymm1	239,200,000	447,200,000	0.535
0x7fe064287de6		vbroadcastss ymm2, dword ptr [rcx+rb	135,200,000	124,800,000	1.083
0x7fe064287ded		vfmadd231ps ymm6, ymm2, ymm0	41,600,000	36,400,000	1.143
0x7fe064287df2		vfmadd231ps ymm12, ymm2, ymm1	184,600,000	387,400,000	0.477
0x7fe064287df7		vbroadcastss ymm2, dword ptr [rbp-0x	78,000,000	114,400,000	0.682
0x7fe064287dfd		vfmadd231ps ymm7, ymm2, ymm0	18,200,000	93,600,000	0.194
0x7fe064287e02		vfmadd231ps ymm13, ymm2, ymm1	179,400,000	325,000,000	0.552
0x7fe064287e07		vbroadcastss ymm2, dword ptr [rbp+rb	75,400,000	83,200,000	0.906
0x7fe064287e0e		vfmadd231ps ymm8, ymm2, ymm0	36,400,000	111,800,000	0.326
0x7fe064287e13		vfmadd231ps ymm14, ymm2, ymm1	200,200,000	366,600,000	0.546
0x7fe064287e18		vbroadcastss ymm2, dword ptr [rbp+rb	93,600,000	83,200,000	1.125
0x7fe064287e1f		vfmadd231ps ymm9, ymm2, ymm0	57,200,000	101,400,000	0.564

Additional VTune Capabilities

- Additional collection types:
  - Performance Snapshot
  - Hotspots
  - Memory Access
  - HPC Performance Characterization
- Can attach to running process:

```
$ vtune -collect hotspots -target-pid=17704
```

- Event Count View
  - See CPU event counters for functions
- Reporting:

```
$ vtune -report summary -result-dir=r000ue
```

  - Text, HTML, XML, and CSV formats
- Can defer finalization using

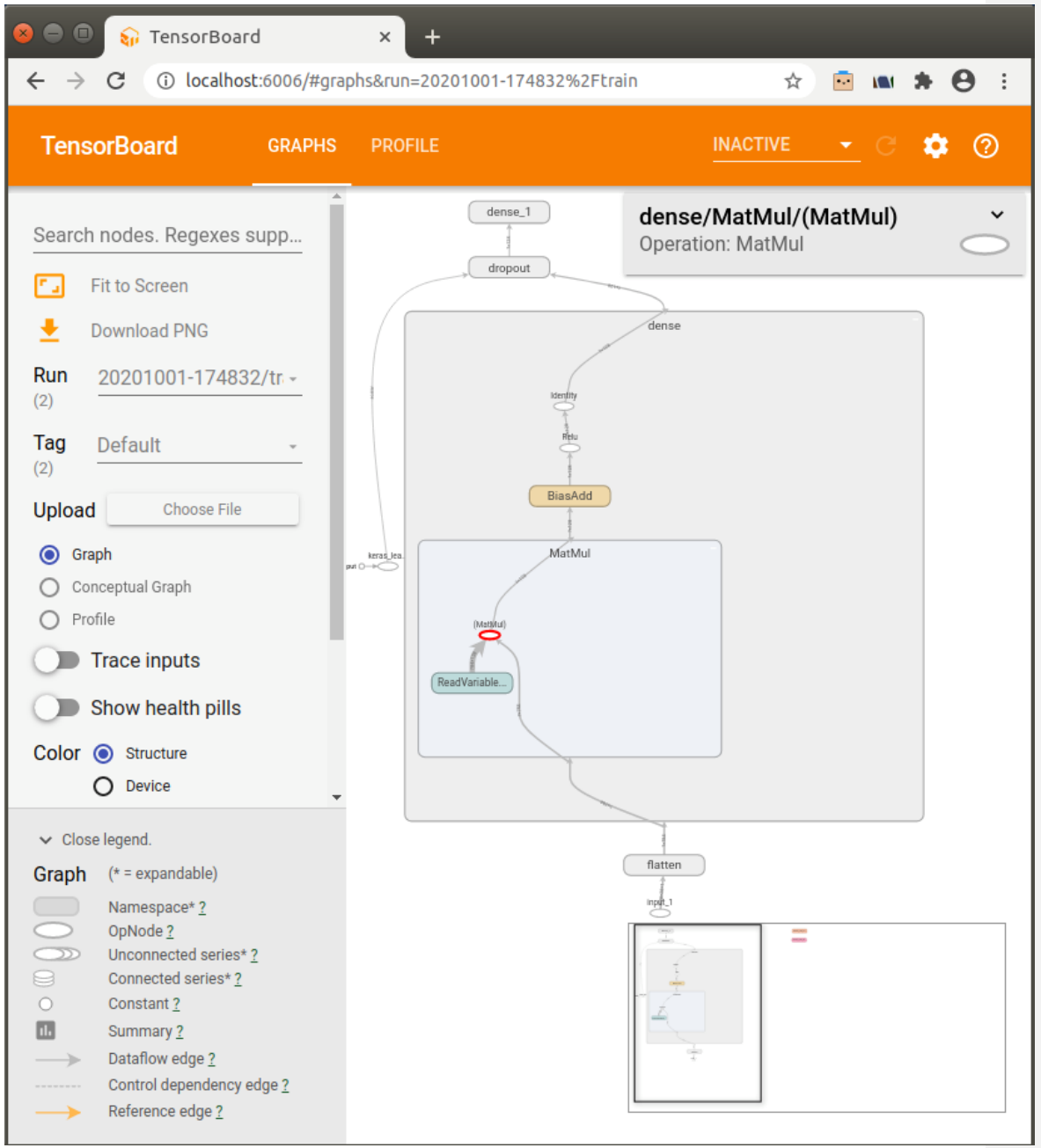
```
-finalization-mode=deferred
```

  - Finalization happens when .vtune file is first opened in GUI

Correlating with TensorBoard

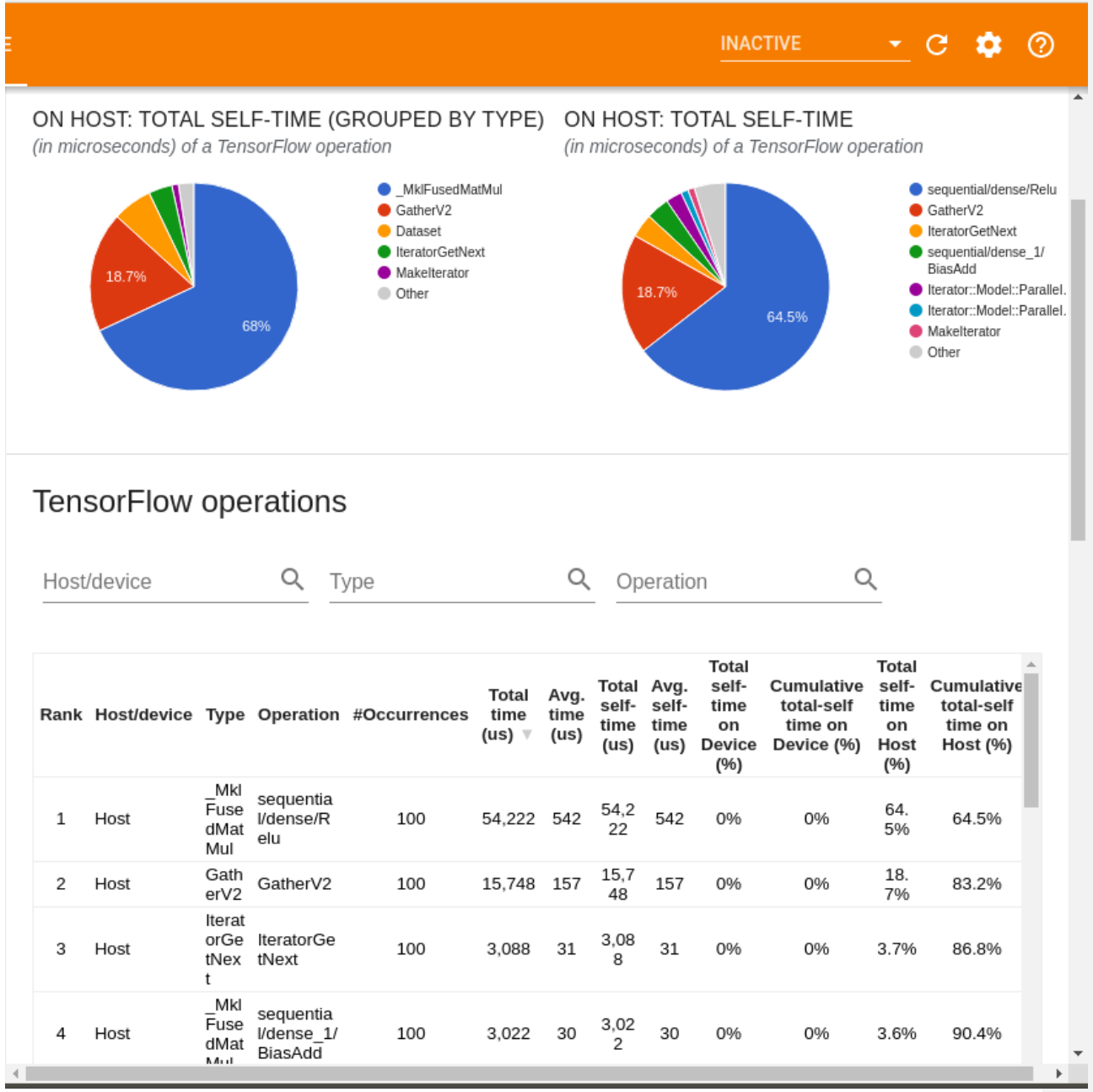
TensorBoard: Graph View

- Conceptual nested graph of model structure
- Round-rect boxes are logical operations which can be expanded by double-clicking
- Ovals are TF-Ops
- Various heat-map views color graph for:
  - Compute time
  - Memory usage



TensorBoard: Profile Stats View

- TB 2.x Profile tab has a new, very nice TF Stats view
- Pie charts quickly show what TF-Ops are using most time
  - MKLFusedMatMuls are using most
- Table view provides more information on TF-Ops usage
- Note that this is at the TF-Op level
  - VTune gives additional insight at the hardware level



Rank	Host/device	Type	Operation	#Occurrences	Total time (us)	Avg. time (us)	Total self-time (us)	Avg. self-time (us)	Total self-time on Device (%)	Cumulative total-self time on Device (%)	Total self-time on Host (%)	Cumulative total-self time on Host (%)
1	Host	_MklFusedMatMul	sequential/dense/Relu	100	54,222	542	54,222	542	0%	0%	64.5%	64.5%
2	Host	GatherV2	GatherV2	100	15,748	157	15,748	157	0%	0%	18.7%	83.2%
3	Host	IteratorGetNext	IteratorGetNext	100	3,088	31	3,088	31	0%	0%	3.7%	86.8%
4	Host	_MklFusedMatMul	sequential/dense_1/BiasAdd	100	3,022	30	3,022	30	0%	0%	3.6%	90.4%

Going Further

Online Resources

- Intel VTune Profiler

  https://software.intel.com/content/www/us/en/develop/tools/oneapi.html – “Get the Base Kit” link

  https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/vtune-profiler.html

- TensorFlow

  https://www.tensorflow.org/tutorials – General page for many good TensorFlow tutorials

  https://www.tensorflow.org/tutorials/keras/classification – Source of models used

- TensorBoard

  https://www.tensorflow.org/tensorboard – Links to tutorials, videos

  https://github.com/tensorflow/tensorboard/issues/1961 – Details Keras eager-mode limitations with TB

Interesting Articles

- Maximizing TensorFlow performance in Intel CPUs

  https://software.intel.com/content/www/us/en/develop/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference.html

- VTune with MPI for distributed runs

  https://software.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/configuration-recipes/profiling-mpi-applications.html

  https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/code-profiling-scenarios/mpi-code-analysis.html

- Intel optimization for TensorFlow

  https://software.intel.com/content/www/us/en/develop/articles/intel-optimization-for-tensorflow-installation-guide.html

- Building TensorFlow from source to optimize for your server's CPU

  https://www.tensorflow.org/install/source

  -march=...: builds for specific CPU; **--config=mkl**: builds TensorFlow with MKL kernels

Summary

- TensorBoard is the standard tool to gain insight to how TensorFlow models and TF-Ops are running
- VTune can be added to see how the CPU hardware is running the kernels which implement the TF-Ops and TensorFlow models
- Both profiling types can be collected together in a single run using the methods documented here
- VTune has many more rich capabilities – please try them out!

Backup

TensorFlow Mini-Workload with Profiling Added

Keras: Training

- Simple dense network with standard MNIST dataset
- Produces **saved_model** directory used by inference

Training Core (green text is added for profiling)

```
# TensorBoard tracing and profiling (for steps 500-504)
# Profiling requires: "pip install -U tensorboard_plugin_profile"
log_dir = "logs-train/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                    histogram_freq=1, profile_batch='500,504',
                    embeddings_freq=10, write_graph=True)

# Train -- adjust model parameters (weights) to minimize the loss
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test),
        callbacks=[tensorboard_callback])
```

Keras: Inference

- To overcome small MNIST dataset size, repeat multiple runs

Inference Core (green text is added for profiling)

```
def  run_inference(model, name, input_images, input_labels, callbacks=None):
   predictions = model.predict(input_images, batch_size=100, callbacks=callbacks)
   [...]

log_dir = "logs-infer/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                    histogram_freq=1, profile_batch='500,504',
                    embeddings_freq=10, write_graph=True)

with tf.profiler.experimental.Profile(log_dir):  # Profile one test run
  run_inference(model, 'test', x_test, y_test, tensorboard_callback)
# Artificially run more inference, to get reasonable sample for timing
for i in range(1, 50):   # Run for over a minute on targeted NUC hardware
  run_inference(model, 'test', x_test, y_test)
  run_inference(model, 'train', x_train, y_train)
```

TensorFlow Keras: Training Model with Profiling

- Code added for profiling is in green
- Can run on the command-line with:

```
$ python mnist_dense_train_PROFILE.py
```

- Produces `saved_model` directory
- Script to run VTune is below

run_vtune_train.sh

```
#! /bin/bash

RUN_DIR="${HOME}/local-projects/IXPUG-models-CURRENT"
RUN_CMD='python mnist_dense_train_PROFILE.py'

# Put any vtune setup commands here, like OneAPI or module loading
source /opt/intel/oneapi/setvars.sh
echo -n '===== VTune Being Used =====: ' ; which vtune ; vtune --version

# Load conda and activate environment
source "${HOME}/miniconda3/etc/profile.d/conda.sh"
conda activate py37-tf22-mkl  # Python 3.7, TF 2.2 built with MKL
echo '===== Python Being Used =====' ; which python ; python --version

# Environment variable settings - see "Maximize TensorFlow Performance on CPU"
export KMP_AFFINITY=granularity=fine,compact,1,0
export OMP_NUM_THREADS=4

cd $RUN_DIR
vtune -collect uarch-exploration -app-working-dir "${RUN_DIR}" -- ${RUN_CMD}
echo "Results can be found in ${PWD}"
```

mnist_dense_train_PROFILE.py

```
#!  /usr/bin/env python

# Adapted from TF tutorials at:
#   https://www.tensorflow.org/tutorials/keras/classification

import datetime
import tensorflow as tf

# Load MNIST dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Put into NHWC form, by adding channel=1 to end
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([  # Sequential densenet model
  tf.keras.layers.Flatten(input_shape=input_shape),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
model.summary()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])

# TensorBoard tracing and profiling (for steps 500-504)
# Profiling requires: "pip install -U tensorboard_plugin_profile"
log_dir = "logs-train/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                      histogram_freq=1, profile_batch='500,504',
                      embeddings_freq=10, write_graph=True)

# Train -- adjust model parameters (weights) to minimize the loss
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])

test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('Test accuracy: ', test_acc)  # Evaluate model against test set

model.save('./saved-model')  # Save model for inference runs
```

TensorFlow Keras: Inference Model with Profiling

- Code added for profiling is in green
- Can run on the command-line with:

```
$ python mnist_infer_PROFILE.py
```

- Must run training model first, to produce `saved_model` directory
- Script to run VTune is below

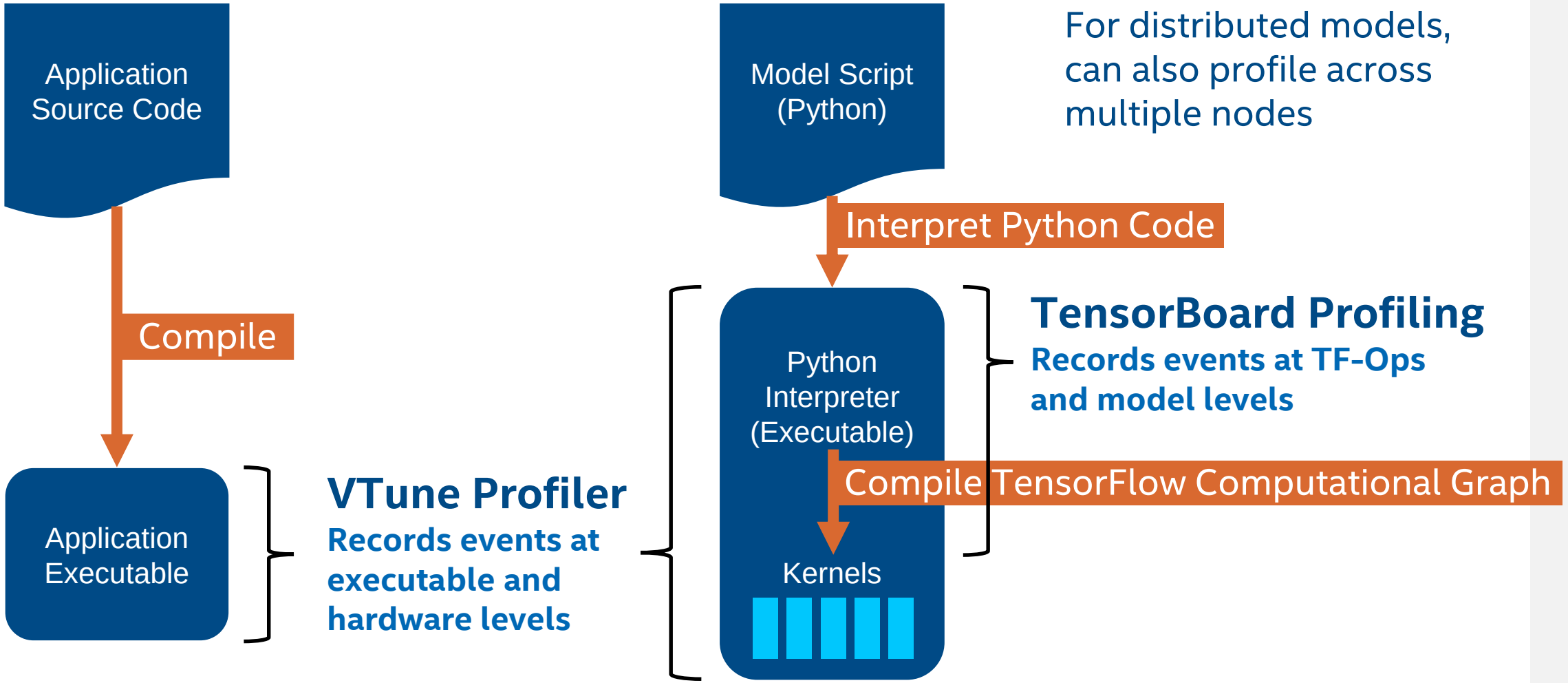
run_vtune_infer.sh

```
#! /bin/bash

RUN_DIR="${HOME}/local-projects/IXPUG-models-CURRENT"
RUN_CMD='python mnist_infer_PROFILE.py'

# Put any vtune setup commands here, like OneAPI or module loading
source /opt/intel/oneapi/setvars.sh
echo -n '===== VTune Being Used =====: ' ; which vtune ; vtune --version

# Load conda and activate environment
source "${HOME}/miniconda3/etc/profile.d/conda.sh"
conda activate py37-tf22-mkl  # Python 3.7, TF 2.2 built with MKL
echo '===== Python Being Used =====' ; which python ; python --version

# Environment variable settings - see "Maximize TensorFlow Performance on CPU"
export KMP_AFFINITY=granularity=fine,compact,1,0
export OMP_NUM_THREADS=4

cd $RUN_DIR
vtune -collect uarch-exploration -app-working-dir "${RUN_DIR}" -- ${RUN_CMD}
echo "Results can be found in ${PWD}"
```

mnist_infer_PROFILE.py

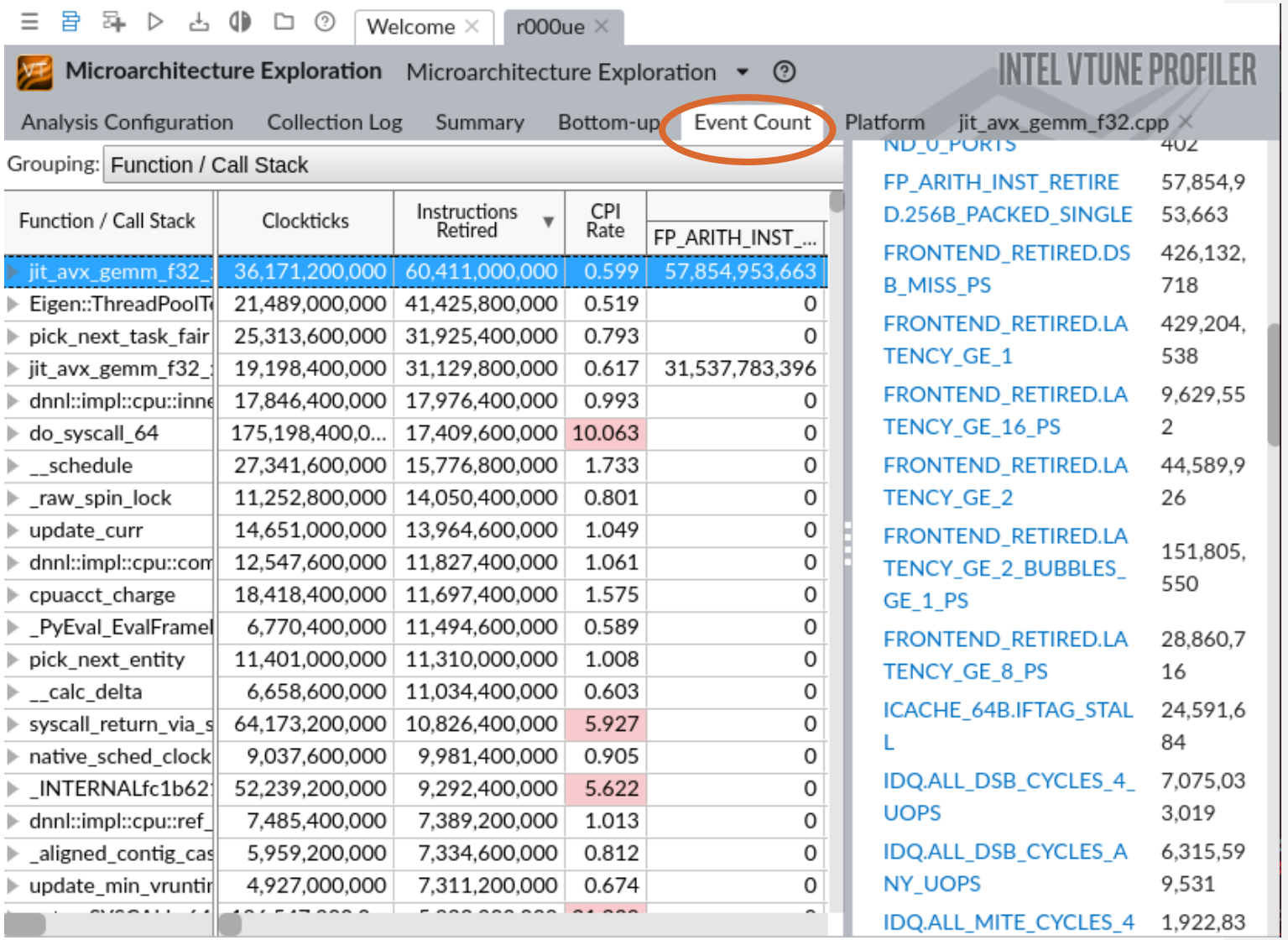
```
#!  /usr/bin/env python

# Adapted from TF tutorials at:
#   https://www.tensorflow.org/tutorials/keras/classification

import datetime
import tensorflow as tf
import numpy

def  run_inference(model, name, input_images, input_labels, callbacks=None):
  predictions = model.predict(input_images, batch_size=100, callbacks=callbacks)
  matches = 0
  for idx in range(0, len(input_images)):
    predicted = numpy.argmax(predictions[idx])
    if int(predicted) == int(input_labels[idx]):  matches += 1
  accuracy = float(matches) / float(len(input_images))
  print('Accuracy on %d %s images: %f' % (len(input_images), name, accuracy))
# End def run_inference(...)

mnist = tf.keras.datasets.mnist  # Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Put into NHWC form, by adding channel=1 to end, input_shape = (28, 28, 1)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.load_model('./saved-model') # Load previously saved model
model.summary() ; model.compile()

log_dir = "logs-infer/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                    histogram_freq=1, profile_batch='500,504',
                    embeddings_freq=10, write_graph=True)

with tf.profiler.experimental.Profile(log_dir):  # Profile one test run
  run_inference(model, 'test', x_test, y_test, tensorboard_callback)
# Artificially run more inference, to get reasonable sample for timing
for i in range(1, 50):   # Run for over a minute on targeted NUC hardware
  run_inference(model, 'test', x_test, y_test)
  run_inference(model, 'train', x_train, y_train)
```

Levels of Profiling in Deep Learning



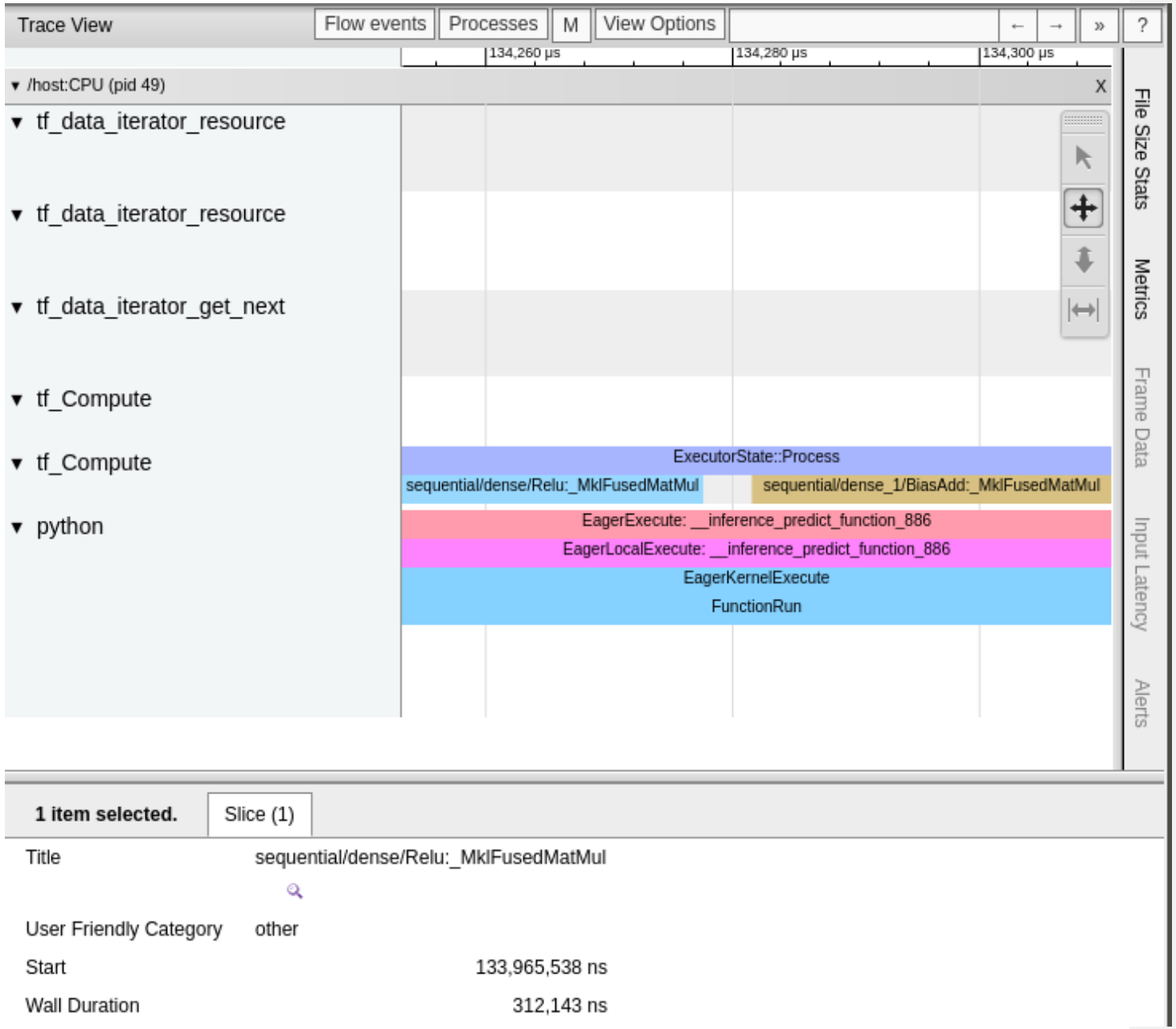
VTune µArch: Event-Count View

- Event-Count tab shows CPU hardware counters
- Again, sorted by Instructions Retired
- For selected "jit_avx_gemm..." function, FP_ARITH... counter is also very high
  - Large fraction of total instructions retired
  - Right column provides easily scrollable list of all counters collected for function



TensorBoard: Profile Trace View

- TB 2.x now includes a timeline
- TF-Ops are tracked to the nano-second resolution
- Can zoom in/out to see overall shape or details
- Here we see a couple MKLFusedMatMuls running at “tf_Compute” level
  - These are related to the jit_avx_gemm functions we saw in VTune’s µArch collection



Environment Configuration Details

Configuration from October 1, 2020:

- Ubuntu 18.04 Linux
- Python 3.7, TensorFlow 2.2, and TensorBoard 2.2 from Anaconda:

```
$ conda create -n py37-tf22-mkl python=3.7
$ conda activate py37-tf22-mkl
$ conda install tensorflow=2.2  # Look for mkl
$ pip install -U tensorboard_plugin_profile
```

- Intel VTune Profiler 2021.1-beta09 from OneAPI Beta09
  - Downloaded from software.intel.com
- Intel NUC with Core i7-6770HQ CPU
  - Training and inference runs performed with OMP* and KMP* settings:

```
export KMP_AFFINITY=granularity=fine,compact,1,0
export OMP_NUM_THREADS=4
```

Notices & Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

Intel technologies may require enabled hardware, software or service activation.


intel