ALCF Webinar

Profiling Deep Learning Performance with Intel® VTune™

Christopher Lishka, Software Applications Engineer, Intel Corporation

Nalini Kumar, Software Applications Engineer, Intel Corporation

December 16, 2020

Deep Learning Profiling Domains

Domain	Conceptual	Operations	Tools
Distributed	Many Servers or Device Instances	• MPI Operations • Horovod* Gradient Ops	• Horovod* Timeline • MPI Analyzers
Model	• Model Graph • Single Server	• TensorFlow* Ops	• TensorBoard*
Hardware	Within CPU or Device	• Assembly Code • Hardware Counters • Micro-Ops	• Intel® VTune™ Profiler

* Other names and brands may be claimed as the property of others.

Deep-Learning Frameworks:
OneMKL and OneDNN Integration

TensorFlow built with MKL-DNN

- Available from Anaconda Cloud
- MKL is a build option, if you build TensorFlow from source
- See Environment Configuration Details slide (at end) for how to download from Anaconda Cloud
- Further information: https://software.intel.com/content/www/us/en/develop/articles/intel-optimization-for-tensorflow-installation-guide.html

PyTorch optimized with MKL-DNN

- Default build is MKL-DNN-enabled
- Further information: https://software.intel.com/content/www/us/en/develop/articles/getting-started-with-intel-optimization-of-pytorch.html

Deep Learning Mini-Workflows

https://github.com/crlishka/dl-mini-workloads

Mini-Workload Directories

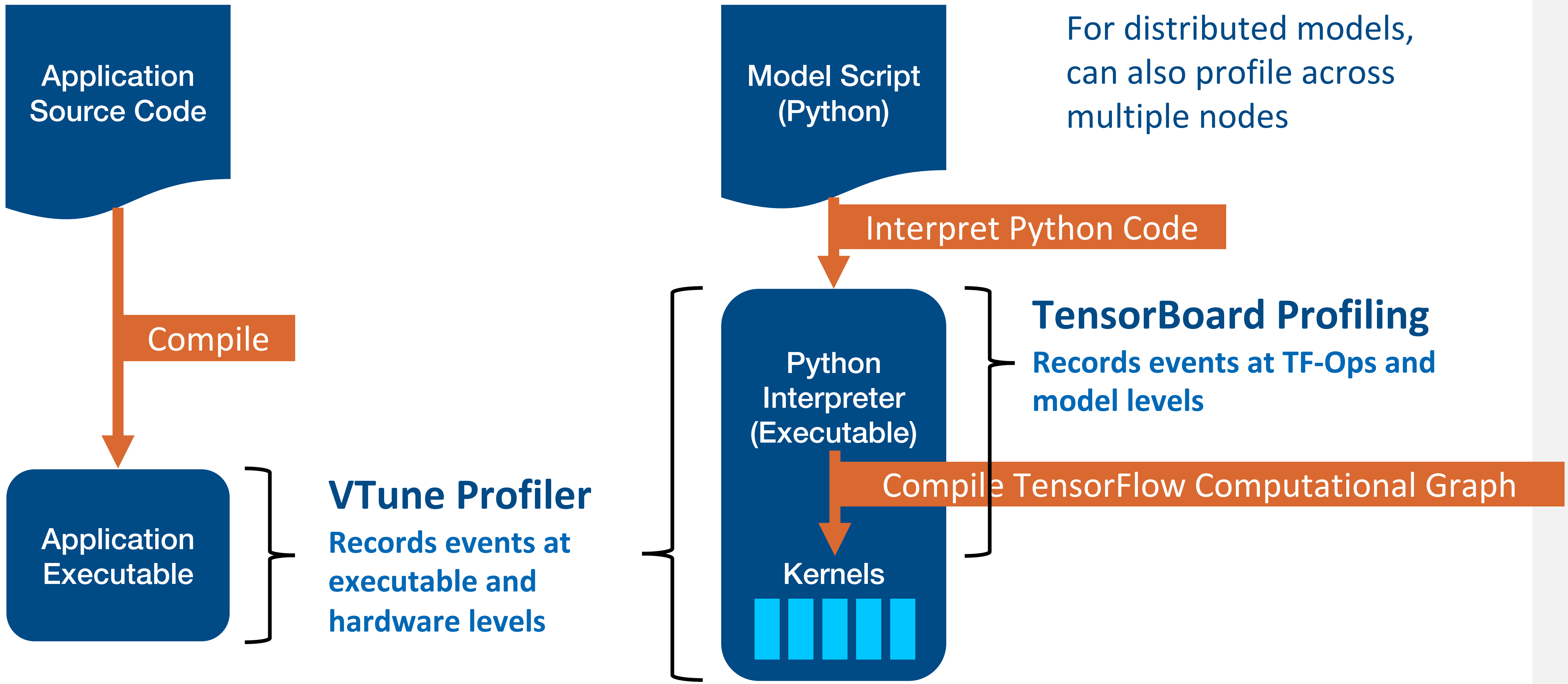
- cnn-cifar10-tf2
- cnn-cifar10-pytorch
- simple-mnist-tf1
- simple-mnist-tf2

Based on standard TensorFlow examples

Each Directory Provides

- Simple training model script
- Simple inference model script
- Script with model-level profiling
- Scripts to run VTune

What VTune Collects



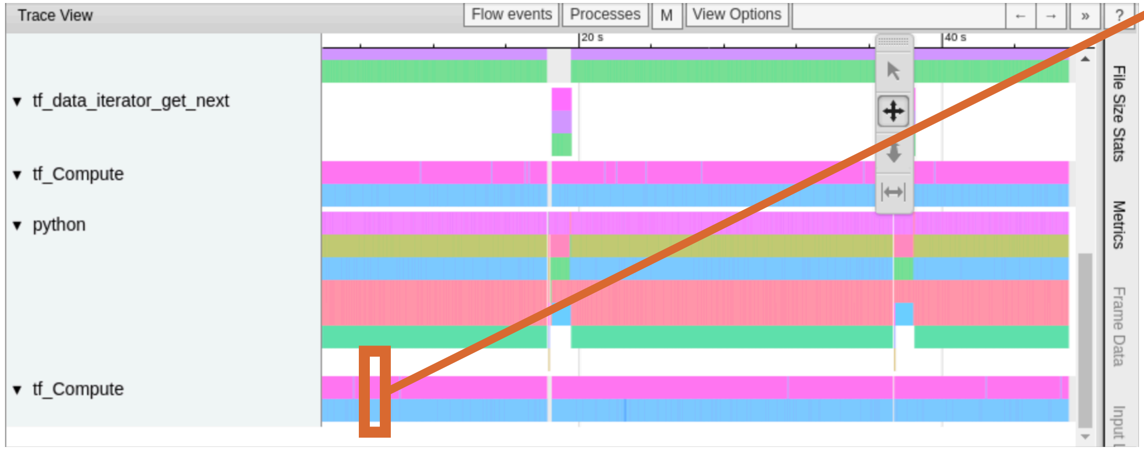
For distributed models, can also profile across multiple nodes

Model Domain Profiling: TensorBoard

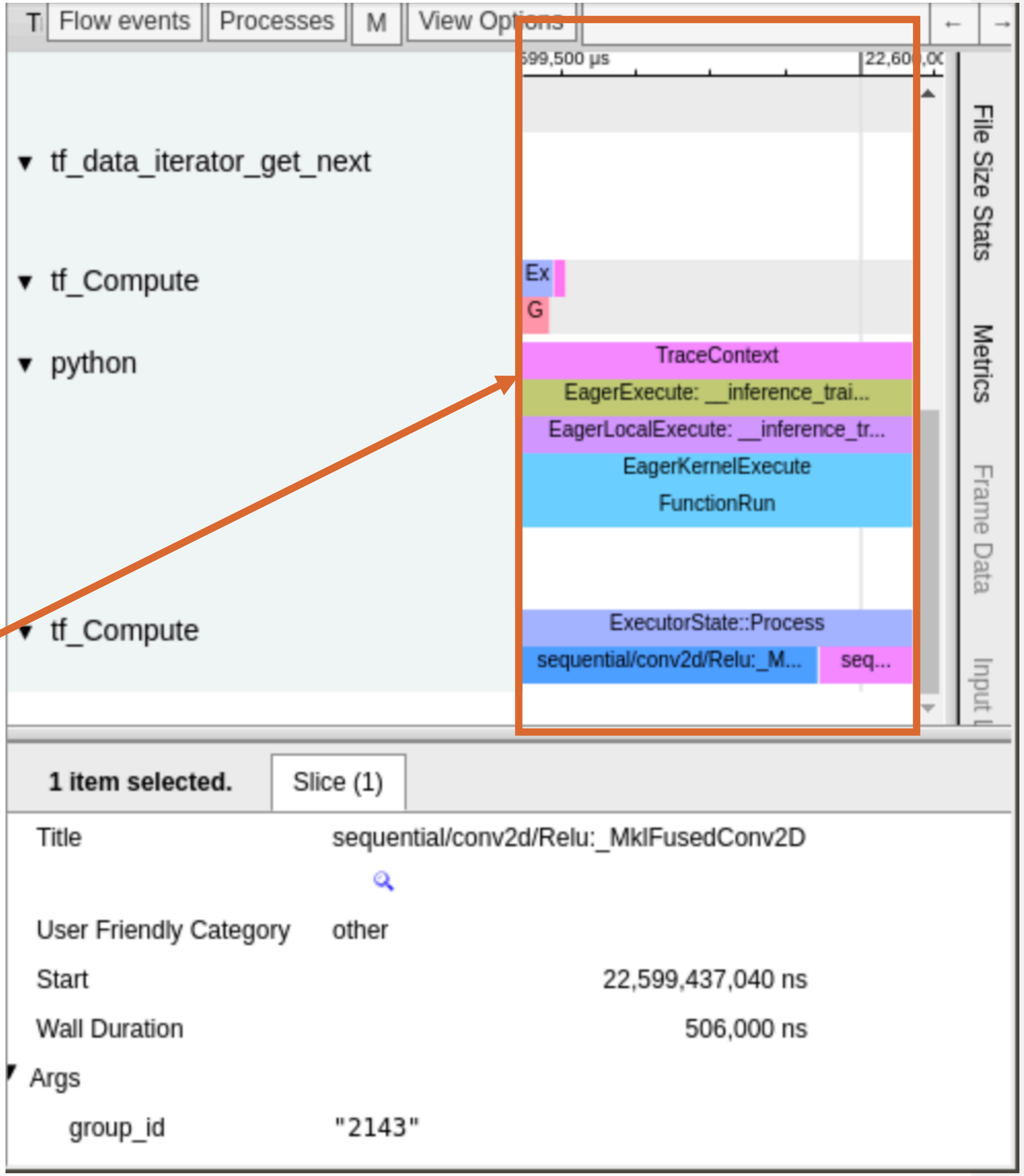
TensorFlow with TensorBoard

Timeline View:

- View of TF-Ops over time
- Can zoom in/out to see overall shape or details



Further information: https://www.tensorflow.org/tensorboard

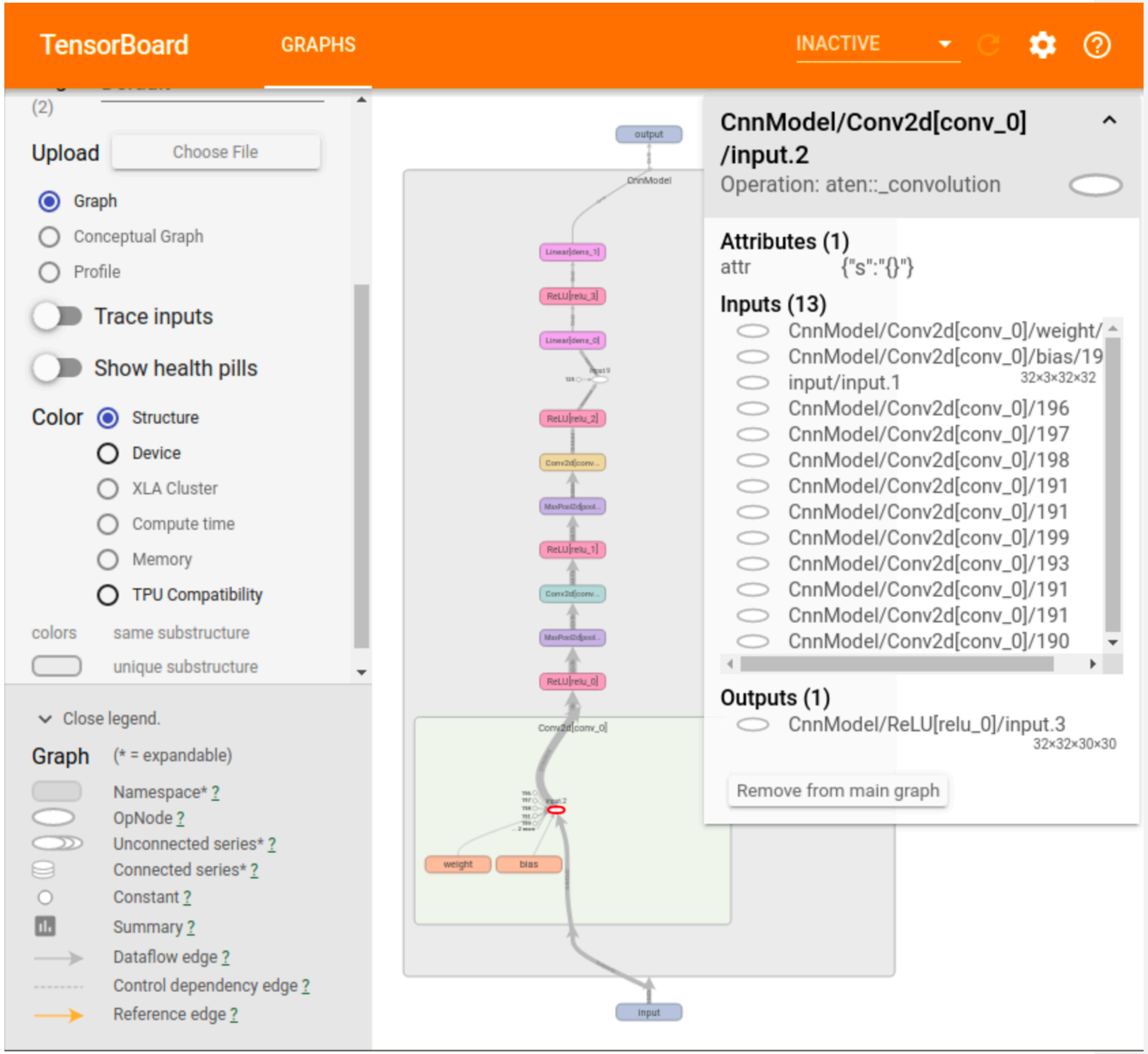


Captured with dl-mini-workloads/cnn-cifar10-tf2 model

PyTorch with TensorBoard

Graphs 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



Captured with dl-mini-workloads/cnn-cifar10-pytorch model

Further information:
https://pytorch.org/docs/stable/tensorboard.html

Hardware Domain CPU Profiling:
Intel VTune Profiler

Getting VTune: OneAPI BaseKit

VTune is freely available in the OneAPI BaseKit at:

https://software.intel.com/content/www/us/en/develop/tools/oneapi.html

"Get It Now" link

To set up shell environment to use OneAPI, run (for bash):

```
$  source ${HOME}/intel/oneapi/setvars.sh
```

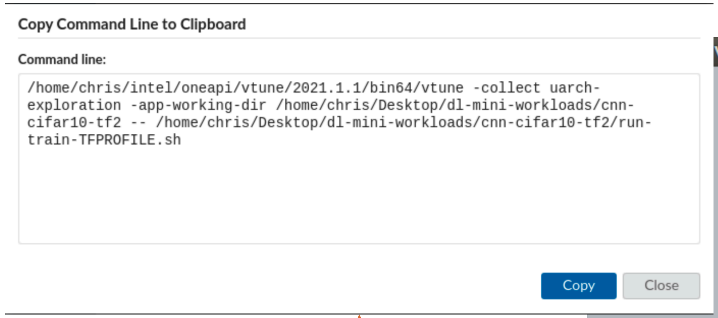
or

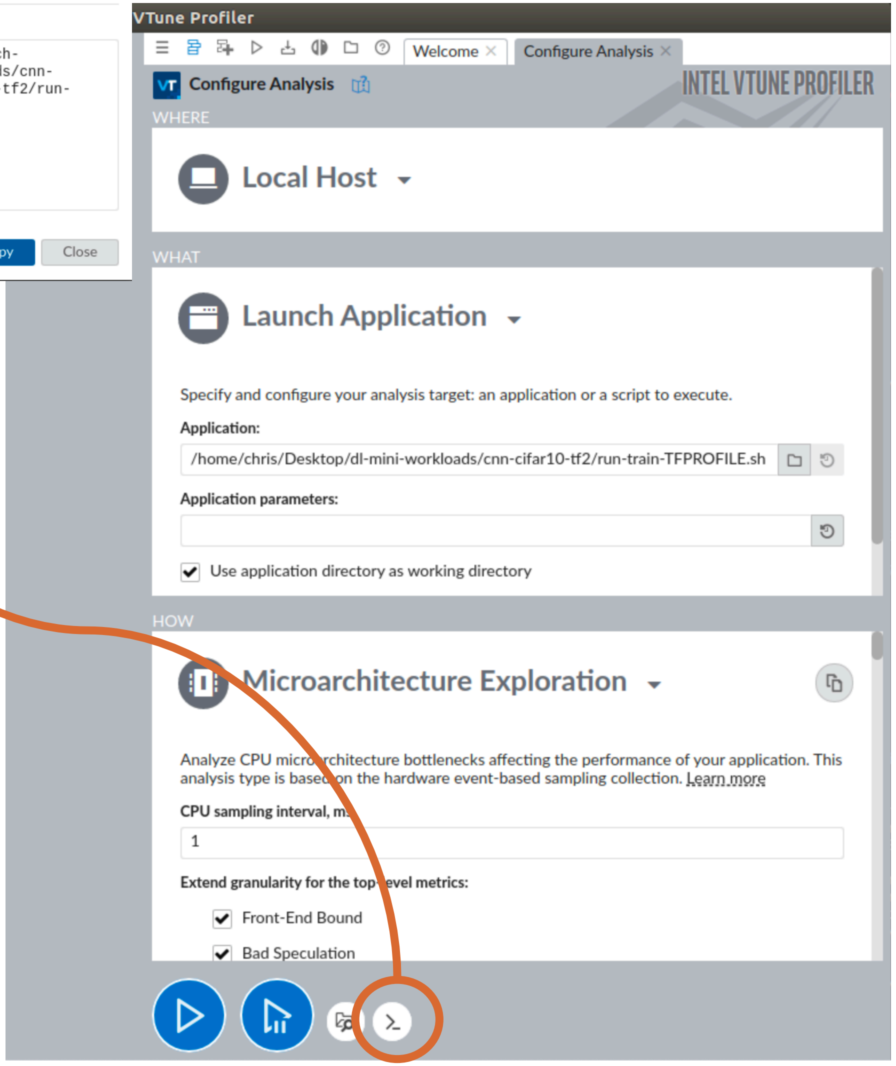
```
$  source /opt/intel/oneapi/setvars.sh
```

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

Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/command-line-interface.html

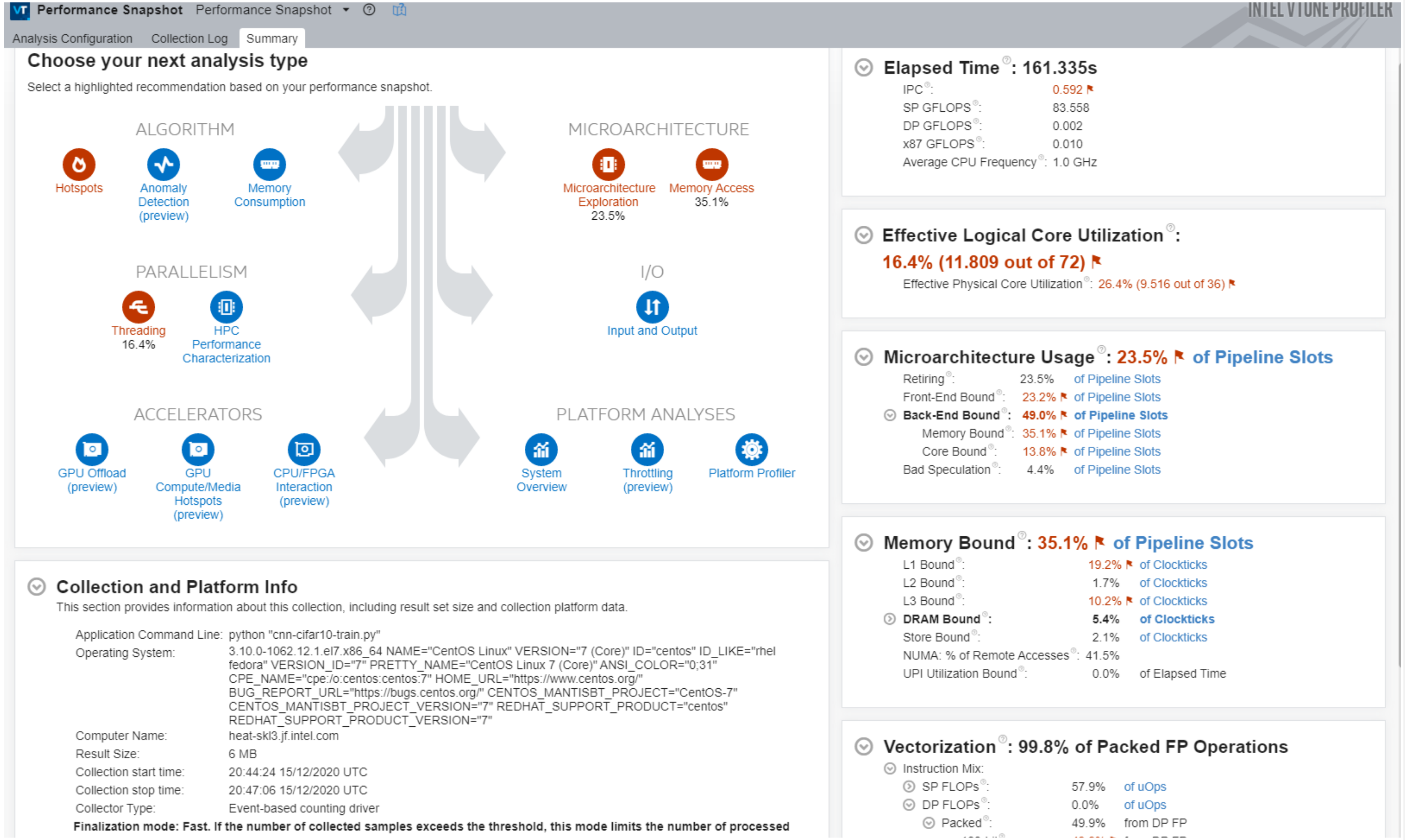
Example Script for Running VTune

run-vtune-training.sh

```
#! /bin/bash

RUN_DIR="${HOME}/dl-mini-workloads/cnn-cifar10-tf2"
RUN_CMD='python cnn-cifar10-train-TFPROFILE.py'

source “${HOME}/intel/oneapi/setvars.sh” # Put vtune setup commands here, like OneAPI or module loading
echo -n '===== VTune Being Used =====: '; vtune --version

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

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

VTUNE_OPTS=‘-finalization-mode=deferred’                            # Common VTune command-line options

cd $RUN_DIR
vtune -collect uarch-exploration $VTUNE_OPTS -- ${RUN_CMD}                                # Run VTune
vtune -collect hotspots -knob sampling-mode=hw $VTUNE_OPTS -- ${RUN_CMD}
echo "Results can be found in ${PWD}”
```

VTune: Performance Snapshot

- Some VTune analysis types take a long time to collect.
- Performance snapshot provides good first insight
- Provides suggestions on other analysis types which may be helpful in further analysis



VTune: Performance Snapshot (Skylake Server)

With OMP_NUM_THREADS= cores x 2

Elapsed Time	**170.986s**	
IPC	0.481	
SP GFLOPS	79.231	
DP GFLOPS	0.001	
x87 GFLOPS	0.011	
Average CPU Frequency	1.0 GHz	
Effective Logical Core Utilization	**21.2% (15.254 out of 72)**	
Effective Physical Core Utilization	33.6% (12.100 out of 36)	
Microarchitecture Usage	**20.0%**	**of Pipeline Slots**
Retiring	20.0%	of Pipeline Slots
Front-End Bound	24.5%	of Pipeline Slots
Back-End Bound	**51.5%**	**of Pipeline Slots**
Memory Bound	36.4%	of Pipeline Slots
Core Bound	15.1%	of Pipeline Slots
Bad Speculation	4.0%	of Pipeline Slots
Memory Bound	**36.4%**	**of Pipeline Slots**
L1 Bound	21.7%	of Clockticks
L2 Bound	1.6%	of Clockticks
L3 Bound	9.4%	of Clockticks
DRAM Bound	**5.3%**	**of Clockticks**
Store Bound	1.9%	of Clockticks
NUMA: % of Remote Accesses	32.8%	
UPI Utilization Bound	0.0%	of Elapsed Time
Vectorization	**99.8% of Packed FP Operations**	

With OMP_NUM_THREADS=cores

Elapsed Time	**161.335s**	
IPC	0.592	
SP GFLOPS	83.558	
DP GFLOPS	0.002	
x87 GFLOPS	0.010	
Average CPU Frequency	1.0 GHz	
Effective Logical Core Utilization	**16.4% (11.809 out of 72)**	
Effective Physical Core Utilization	26.4% (9.516 out of 36)	
Microarchitecture Usage	**23.5%**	**of Pipeline Slots**
Retiring	23.5%	of Pipeline Slots
Front-End Bound	23.2%	of Pipeline Slots
Back-End Bound	**49.0%**	**of Pipeline Slots**
Memory Bound	35.1%	of Pipeline Slots
Core Bound	13.8%	of Pipeline Slots
Bad Speculation	4.4%	of Pipeline Slots
Memory Bound	**35.1%**	**of Pipeline Slots**
L1 Bound	19.2%	of Clockticks
L2 Bound	1.7%	of Clockticks
L3 Bound	10.2%	of Clockticks
DRAM Bound	**5.4%**	**of Clockticks**
Store Bound	2.1%	of Clockticks
NUMA: % of Remote Accesses	41.5%	
UPI Utilization Bound	0.0%	of Elapsed Time
Vectorization	**99.8% of Packed FP Operations**	

With OMP_NUM_THREADS=cores & numactl --cpucorebind=0 --membind=0

Elapsed Time	**155.347s**	
IPC	0.611	
SP GFLOPS	86.507	
DP GFLOPS	0.001	
x87 GFLOPS	0.008	
Average CPU Frequency	1.0 GHz	
Effective Logical Core Utilization	**14.8% (10.680 out of 72)**	
Effective Physical Core Utilization	17.7% (6.388 out of 36)	
Microarchitecture Usage	**32.2%**	**of Pipeline Slots**
Retiring	32.2%	of Pipeline Slots
Front-End Bound	31.1%	of Pipeline Slots
Back-End Bound	**31.8%**	**of Pipeline Slots**
Memory Bound	17.6%	of Pipeline Slots
Core Bound	14.3%	of Pipeline Slots
Bad Speculation	4.8%	of Pipeline Slots
Memory Bound	**17.6%**	**of Pipeline Slots**
L1 Bound	16.7%	of Clockticks
L2 Bound	2.7%	of Clockticks
L3 Bound	6.7%	of Clockticks
DRAM Bound	**1.3%**	**of Clockticks**
Store Bound	2.2%	of Clockticks
NUMA: % of Remote Accesses	4.6%	
UPI Utilization Bound	0.0%	of Elapsed Time
Vectorization	**99.8% of Packed FP Operations**	

Using DNNL_VERBOSE to Augment VTune

- Not a replacement for Vtune!
  - Much faster than uArch analysis
- Runtime control via environment variables

$export DNNL_VERBOSE=1

$export DNNL_TIMESTAMP=1

Environment variable	Value	Description
DNNL_VERBOSE	0	**no verbose output (default)**
	1	primitive information at execution
	2	primitive information at creation and execution
DNNL_VERBOSE_TIMESTAMP	0	**display timestamps disabled (default)**
	1	display timestamps enabled

```
(/home/nalinik/miniconda3/py36_envs/ipw) [nalinik@heat-skl3 cnn-cifar10-tf2]$ head -10 cifar10-nhwc.log
dnnl_verbose,info,oneDNN v1.4.0 (commit N/A)
dnnl_verbose,info,cpu,runtime:OpenMP
dnnl_verbose,info,cpu,isa:Intel AVX-512 with AVX512BW, AVX512VL, and AVX512DQ extensions
dnnl_verbose,info,gpu,runtime:none
dnnl_verbose,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:acdb:f0 dst_f32::blocked:abcd:f0,,,32x3x32x32,2.20996
dnnl_verbose,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:cdba:f0 dst_f32::blocked:Acdb16a:f0,,,32x3x3x3,0.0009765
62
dnnl_verbose,exec,cpu,convolution,jit:avx512_common,forward_training,src_f32::blocked:abcd:f0 wei_f32::blocked:Acdb16
a:f0 bia_f32::blocked:a:f0 dst_f32::blocked:aBcd16b:f0,post_ops:'eltwise_relu;';,alg:convolution_direct,mb32_ic3oc32_
ih32oh30kh3sh1dh0ph0_iw32ow30kw3sw1dw0pw0,0.773926
dnnl_verbose,exec,cpu,pooling,jit:avx512_common,forward_training,src_f32::blocked:aBcd16b:f0 dst_f32::blocked:aBcd16b
:f0 ws_u8::blocked:aBcd16b:f0,,alg:pooling_max,mb32ic32_ih30oh15kh2sh2ph0_iw30ow15kw2sw2pw0,0.631104
dnnl_verbose,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:cdba:f0 dst_f32::blocked:ABcd16b16a:f0,,,64x32x3x3,3.141
11
dnnl_verbose,exec,cpu,convolution,jit:avx512_common,forward_training,src_f32::blocked:aBcd16b:f0 wei_f32::blocked:ABc
d16b16a:f0 bia_f32::blocked:a:f0 dst_f32::blocked:aBcd16b:f0,post_ops:'eltwise_relu;';,alg:convolution_direct,mb32_ic
32oc64_ih15oh13kh3sh1dh0ph0_iw15ow13kw3sw1dw0pw0,0.585205
```

info

execution time in ms

engine, primitive name

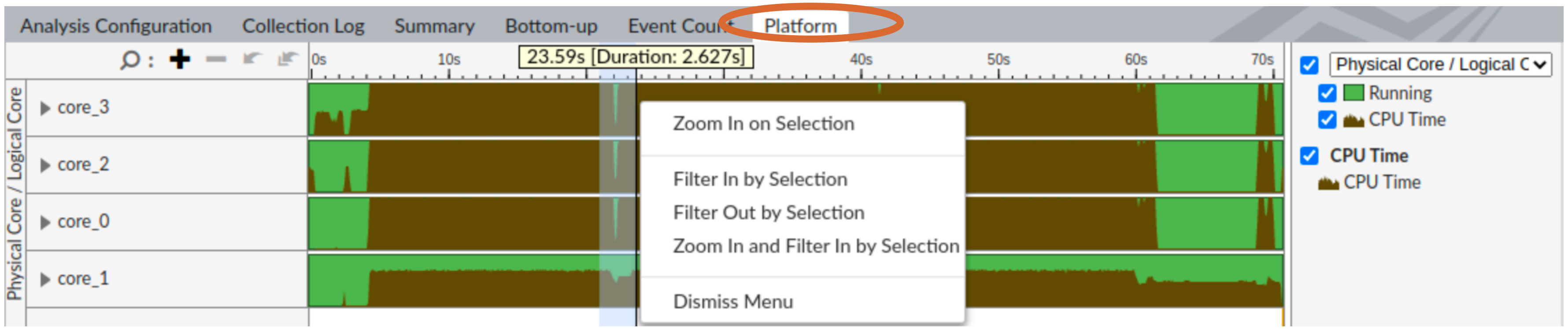
fused operations

algorithm used

Further information:

https://oneapi-src.github.io/oneDNN/dev_guide_verbose.html
https://oneapi-src.github.io/oneDNN/performance_profiling_cpp.html

VTune µArchitecture Exploration: CPU Timeline



Captured with dl-mini-workloads/cnn-cifar10-tf2 model

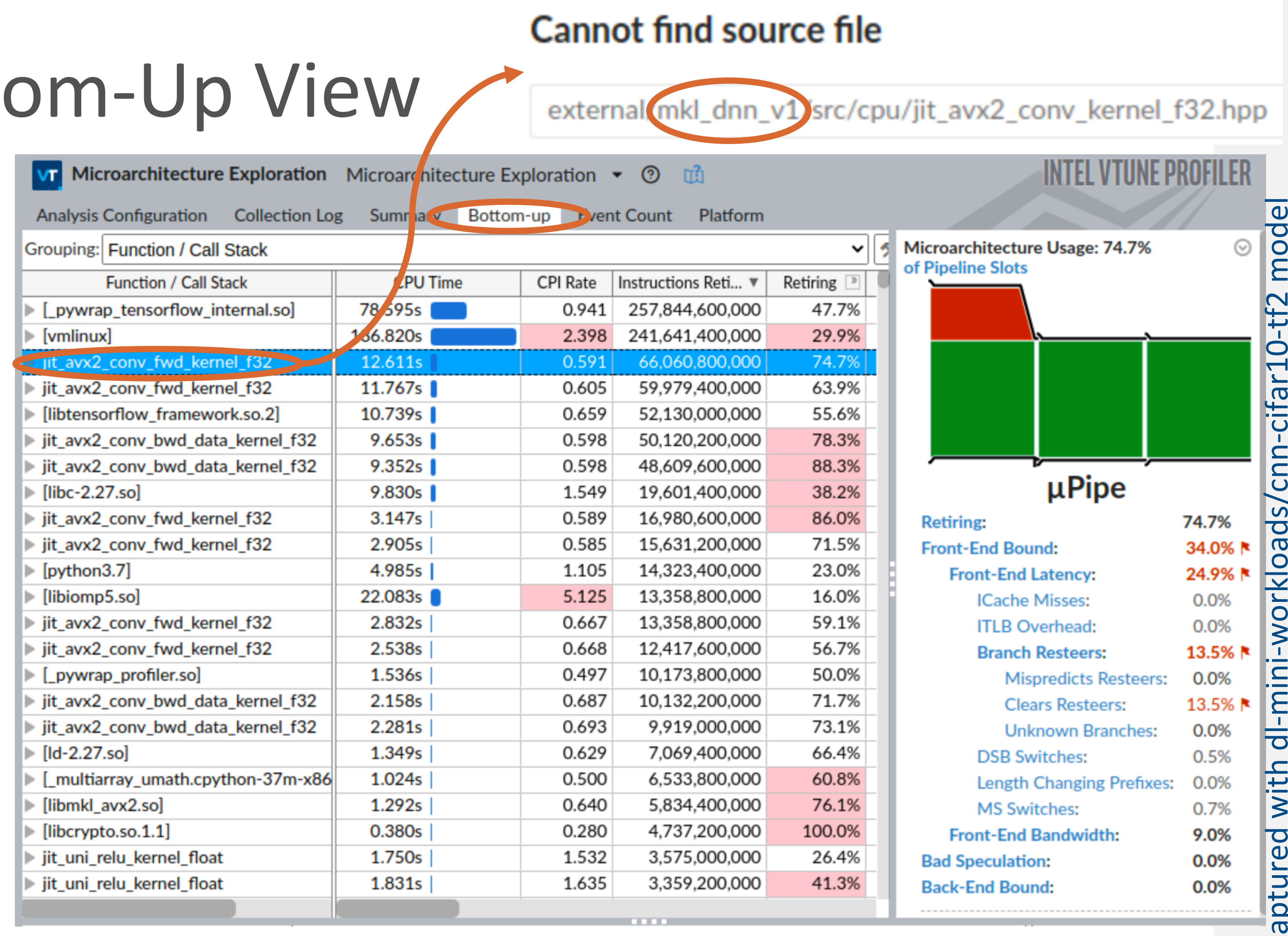
- Displayed on Bottom-Up and Platform tabs
- Can display by threads, processes, packages (sockets), and cores
- Hovering mouse over timeline shows details (e.g. CPU time)

- Zooming
  - **+** and **–** buttons
  - Select region, right-click and choose "Zoom In on Selection"

Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/user-interface-reference/pane-timeline.html

VTune µArch: Bottom-Up View

- Sorted by Instructions Retired column
  - Top represents functions which ran most instructions during VTune run
- Selected function is “jit_avx2_conv_fwd_kernel_f32”
  - Double-clicking on name shows this is an MKL-DNN function (in Source view)
  - Many instances of this function due to MKL’s JIT compilation. Using “Source Function” grouping will collapse these into one entry.
  - µPipe display 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

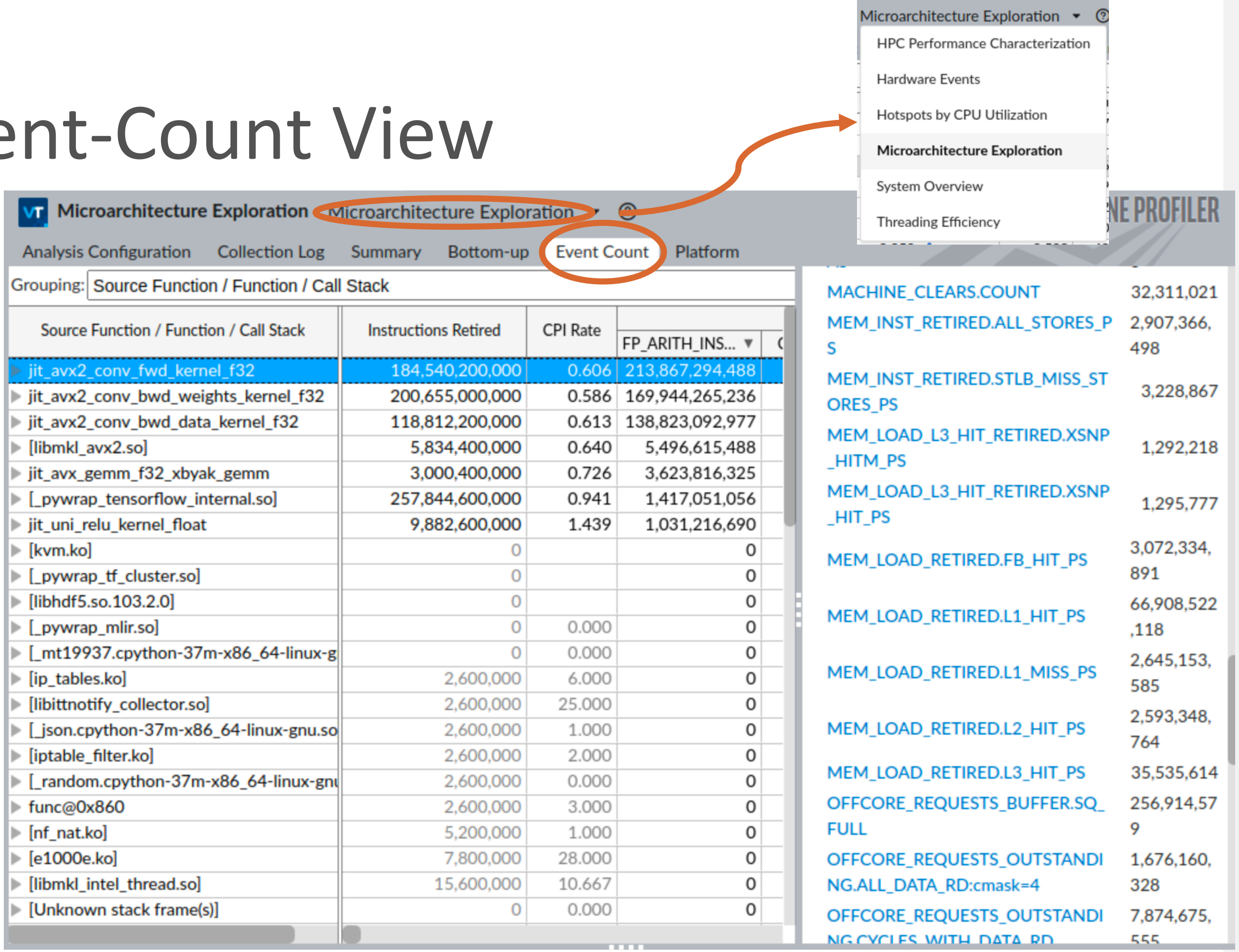


Captured with dl-mini-workloads/cnn-cifar10-tf2 model

Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/user-interface-reference/window-bottom-up.html

VTune µArch: Event-Count View

- 2nd "Microarchitecture Exploration" is a view selector
- Event-Count tab shows CPU hardware counters
- Grouped by "Source Function"
  - Collapsing JITted functions into one expandable list
- Now sorted by FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE
  - Only a few source functions are using vector single-precision floating-point math

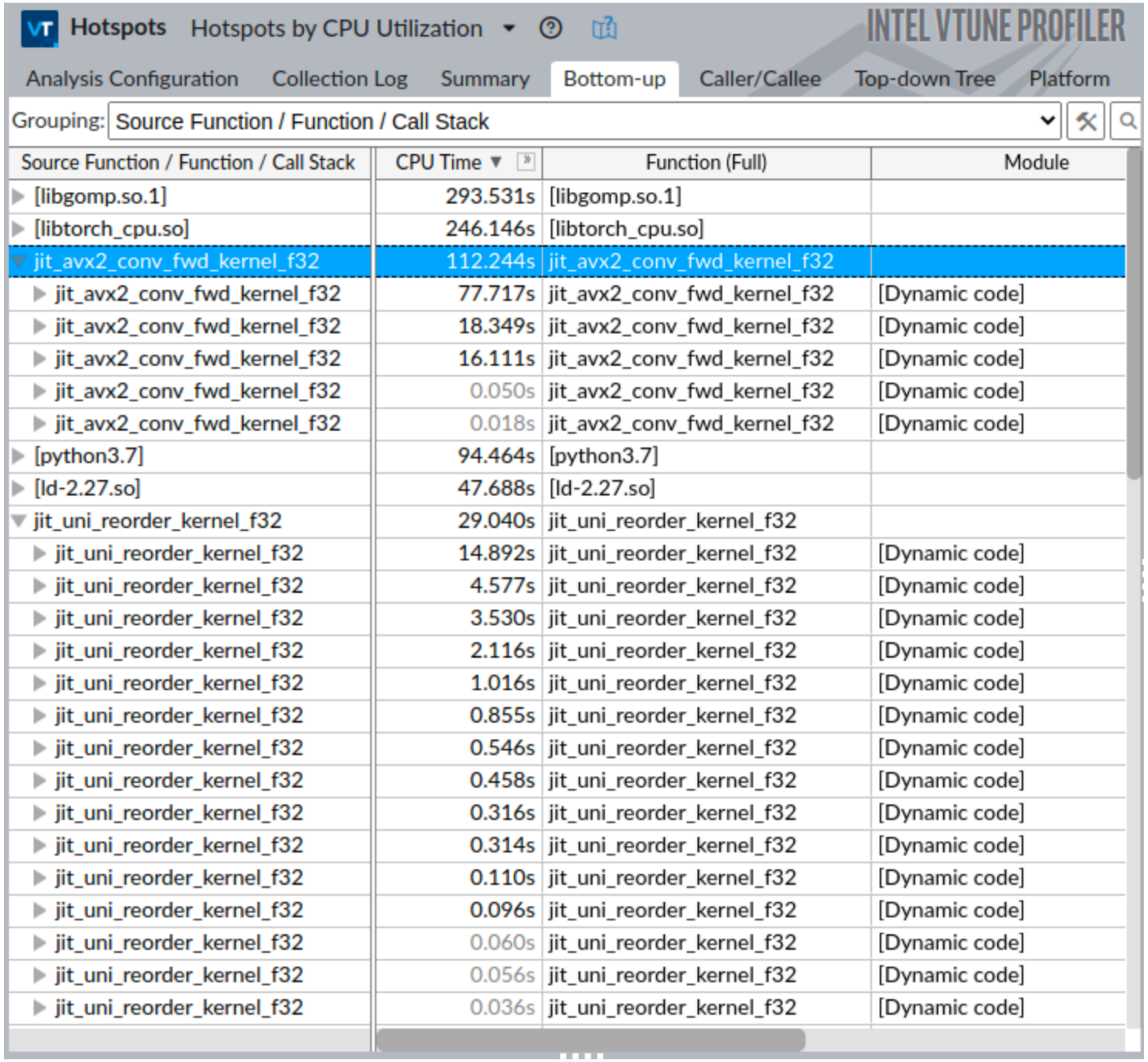


Captured with dl-mini-workloads/cnn-cifar10-tf2 model

Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/intel-processor-events-reference.html

VTune with PyTorch

- PyTorch with MKL shows up the same way as TensorFlow with MKL
  - Note the “libtorch_cpu.so”
- Here I have chosen a ”Hotspots” collection
  - Does not have the event counters
- Grouped by Source Function, with a couple JIT entries expanded



Source Function / Function / Call Stack	CPU Time	Function (Full)	Module
[libgomp.so.1]	293.531s	[libgomp.so.1]	
[libtorch_cpu.so]	246.146s	[libtorch_cpu.so]	
jit_avx2_conv_fwd_kernel_f32	112.244s	jit_avx2_conv_fwd_kernel_f32	
jit_avx2_conv_fwd_kernel_f32	77.717s	jit_avx2_conv_fwd_kernel_f32	[Dynamic code]
jit_avx2_conv_fwd_kernel_f32	18.349s	jit_avx2_conv_fwd_kernel_f32	[Dynamic code]
jit_avx2_conv_fwd_kernel_f32	16.111s	jit_avx2_conv_fwd_kernel_f32	[Dynamic code]
jit_avx2_conv_fwd_kernel_f32	0.050s	jit_avx2_conv_fwd_kernel_f32	[Dynamic code]
jit_avx2_conv_fwd_kernel_f32	0.018s	jit_avx2_conv_fwd_kernel_f32	[Dynamic code]
[python3.7]	94.464s	[python3.7]	
[ld-2.27.so]	47.688s	[ld-2.27.so]	
jit_uni_reorder_kernel_f32	29.040s	jit_uni_reorder_kernel_f32	
jit_uni_reorder_kernel_f32	14.892s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	4.577s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	3.530s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	2.116s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	1.016s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.855s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.546s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.458s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.316s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.314s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.110s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.096s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.060s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.056s	jit_uni_reorder_kernel_f32	[Dynamic code]
jit_uni_reorder_kernel_f32	0.036s	jit_uni_reorder_kernel_f32	[Dynamic code]

Captured with dl-mini-workloads/cnn-cifar10-pytorch model

Further information:

https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/algorithm-group/basic-hotspots-analysis.html

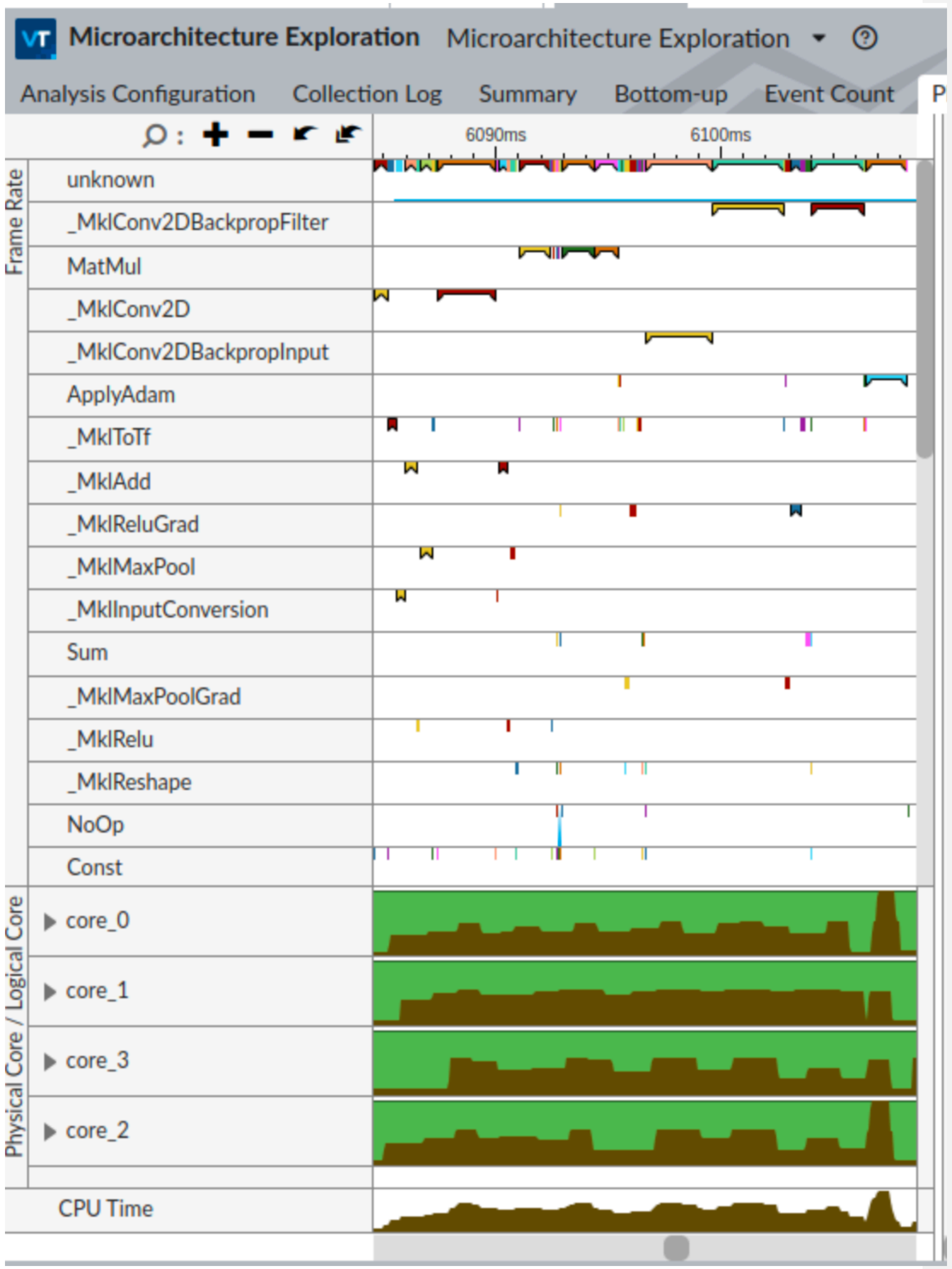
Hybrid: Including Model-Level Profiling in VTune Timeline

VTune has a powerful feature for including external timeline data, called a *custom collector*

- External data needs to be converted to a simple CSV format
- Times need to be in absolute system time
- VTune integrates this data during finalization
- Integrated and displayed into VTune's timeline

Using a couple simple scripts, TensorFlow 1.1x timelines can be included

- TF-Ops visible in the VTune timeline
- TF 2.x times have changed from absolute to relative, not yet possible to include



Further information:

https://software.intel.com/content/www/us/en/develop/articles/profiling-tensorflow-workloads-with-intel-vtune-amplifier.html

Distributed Domain Profiling: Horovod and MPI

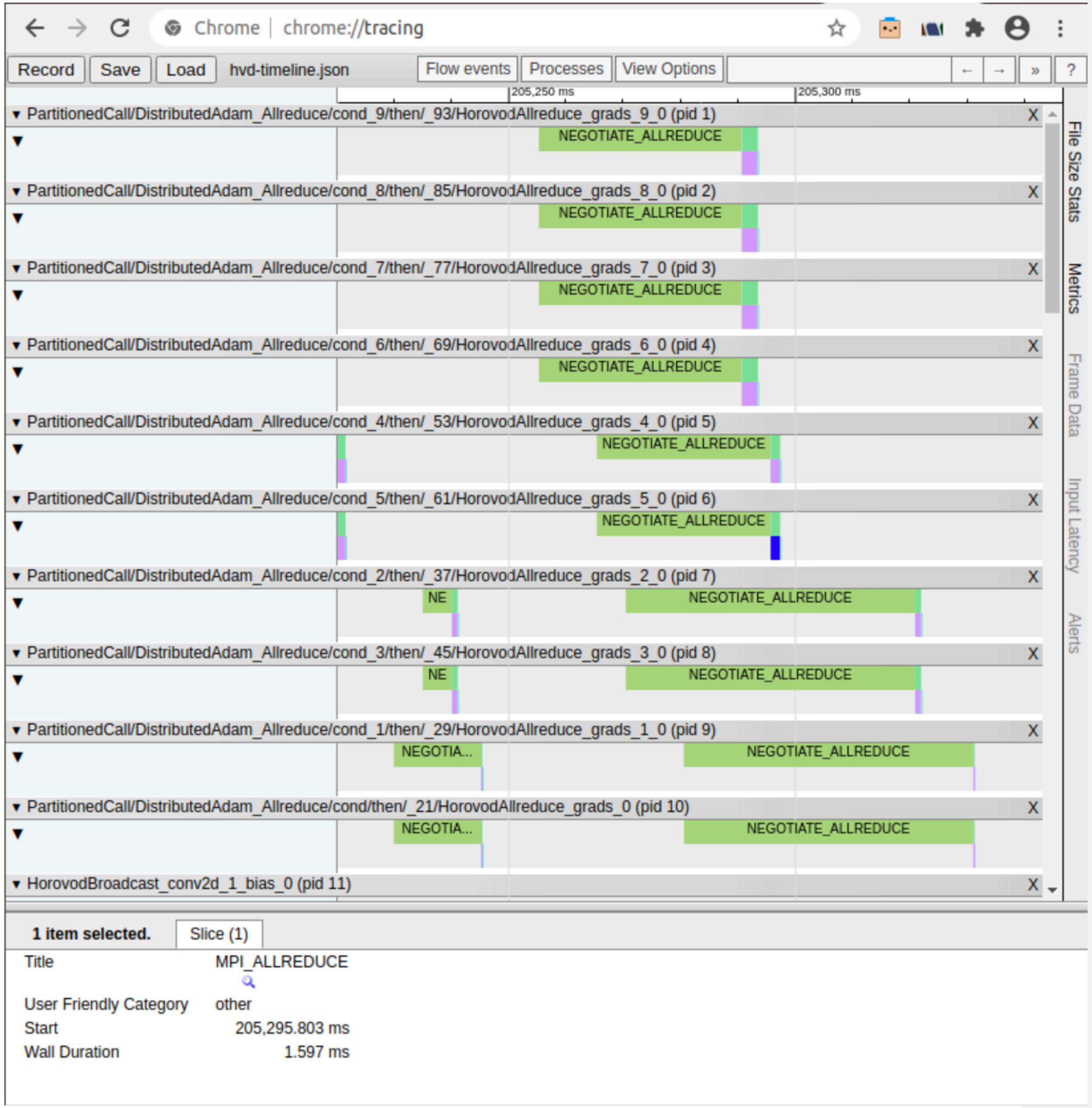
Horovod and Horovod Timeline

Horovod is an easy-to-incorporate data-parallel framework

- Built on MPI
- Available for TensorFlow and PyTorch

Can generate a timeline of Horovod operations with HOROVOD_TIMELINE

- Data format is chrome://tracing compatible JSON file

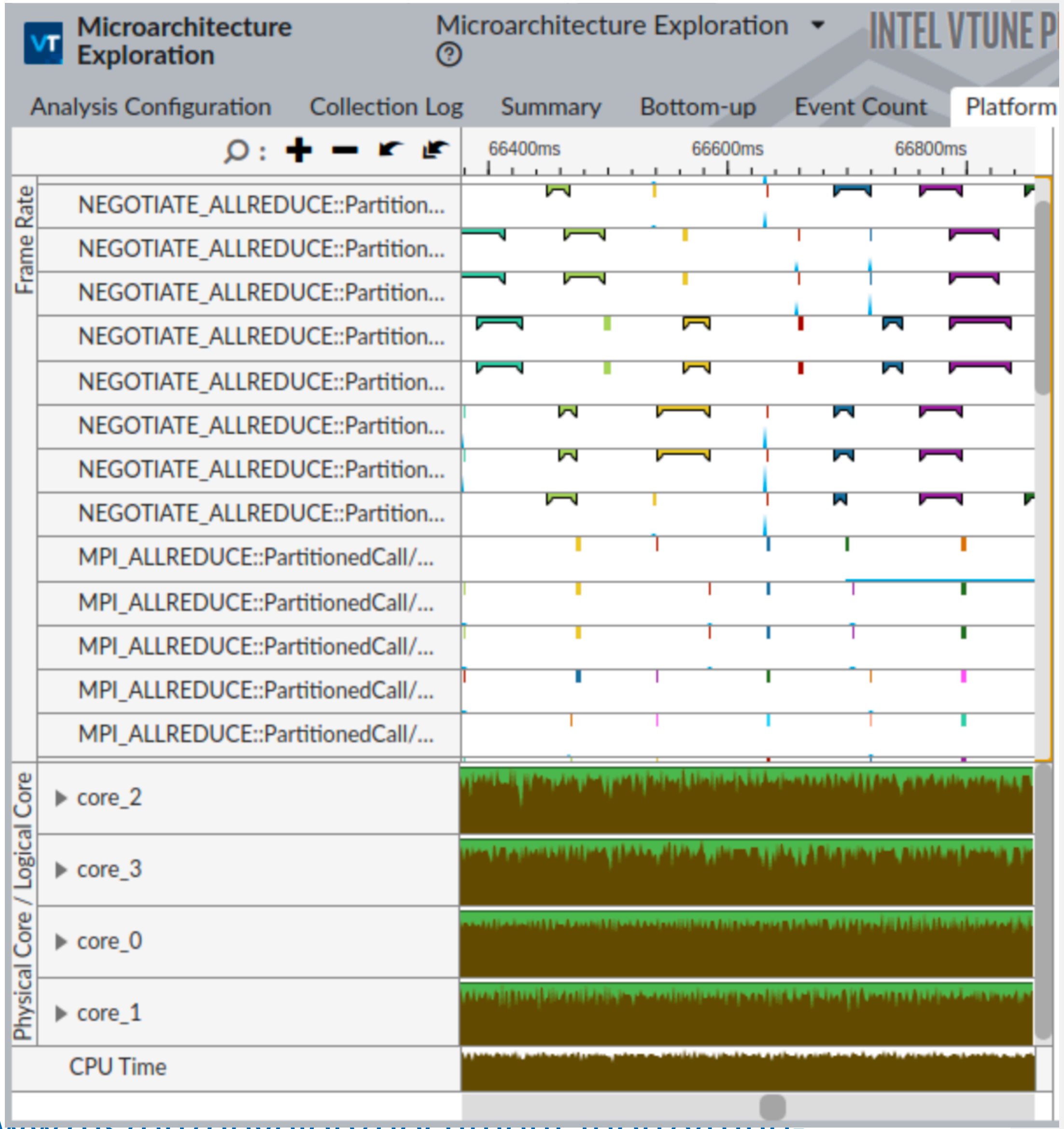


Further information:
https://horovod.readthedocs.io/en/stable/timeline_include.html

Hybrid: Horovod Timeline in VTune Timeline

Using a VTune *custom collector*, Horovod timeline data can be included in VTune's timeline display

- NEGOTIATE_ALLREDUCE and MPI_ALLREDUCE, as seen in previous slide
- Names displayed in "Frame Rate" table are configurable via the conversion script that you write



Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/control-data-collection/external-data-import.html

Running VTune with Horovod/MPI

Attaching VTune to a running process

- Example -- ssh to server running an MPI rank and run the following, replacing “17704” with the PID of the MPI rank:

```
$ vtune -collect uarch-exploration -target-pid=17704
```

- **Further information:** https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/command-line-interface/command-line-interface-reference/target-pid.html

Running VTune across all MPI ranks

- Example (replace “hostname*” with your servers):

```
$  mpirun -n 2 -hosts hostname1:1,hostname2:1  vtune -collect hotspots -trace-mpi -r ./vtune-results -- python cnn-cifar10-horovod-train.py
```

- **Further information:** https://software.intel.com/content/www/us/en/develop/articles/using-intel-advisor-and-vtune-amplifier-with-mpi.html

Intel Trace Anlayzer

- **Further information:** https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/trace-analyzer.html

Going Further

Additional VTune Capabilities

- Many collection types, including:
  - Performance Snapshot
  - Hotspots
  - Memory Access
  - HPC Performance Characterization

- Reporting:

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

  - Text, HTML, XML, and CSV formats

- Deferring VTune finalization:
  - VTune has two phases:
    1. Collection: data collected on running processes
    2. Finalization: computations based on collected data
  - Can defer finalization until VTune's GUI opens the .vtune file:

```
-finalization-mode=none
```

  - May need to do this if VTune command-line tool on server is a later version than VTune-GUI on your laptop

Further information:

https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top.html

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

- Effectively Train and Execute Machine Learning and Deep Learning Projects on CPUs

  https://techdecoded.intel.io/resources/effectively-train-and-execute-machine-learning-and-deep-learning-projects-on-cpus

- 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

- Popular tools can be used to profile at the various Deep Learning domain levels:
  - *Hardware Domain (CPU):* **Intel oneAPI VTune Profiler**
  - *Model Domain:* **TensorBoard** and other model-level profiling tools
  - *Distributed Domain:* **Horovod-Timeline** and MPI profiling tools
- To correlate CPU profiling with model-level and distributed-level profiling data, VTune's *custom collector* can be used to integrate into VTune's timeline
- Profiling data for different domains can be collected together in a single run
  - See examples in https://github.com/crlishka/dl-mini-workloads
- This presentation just scratches the surface of VTune's rich capabilities – please try them out!

Environment Configuration Details

Configuration from December 12, 2020:

Ubuntu 18.04		
How to create TensorFlow 2.2 environment: **`$ conda create -n tf22-mkl python=3.7`** **`$ conda activate tf22-mkl`** **`$ conda install tensorflow=2.2  # mkl`** **`$ pip install -U tensorboard_plugin_profile`**	*How to create TensorFlow 1.14 environment:* **`$ conda create -n tf114-mkl python=3.7`** **`$ conda activate tf114-mkl`** **`$ conda install tensorflow=1.14  # mkl`**	*How to create PyTorch 1.7 environment:* **`$ conda create -n pytorch-mkl python=3.7`** **`$ conda activate pytorch-mkl`** **`$ conda install pytorch cpuonly -c pytorch`** **`$ conda install torchvision -c pytorch`** **`$ conda install tensorboard`**
Intel® oneAPI VTune™ Profiler 2021.1.1 Gold, downloaded from software.intel.com		
Intel NUC with Core i7-6770HQ (AVX2) CPU Skylake Server with Intel Xeon Gold 6140 (AVX512) CPU		

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.

Questions?

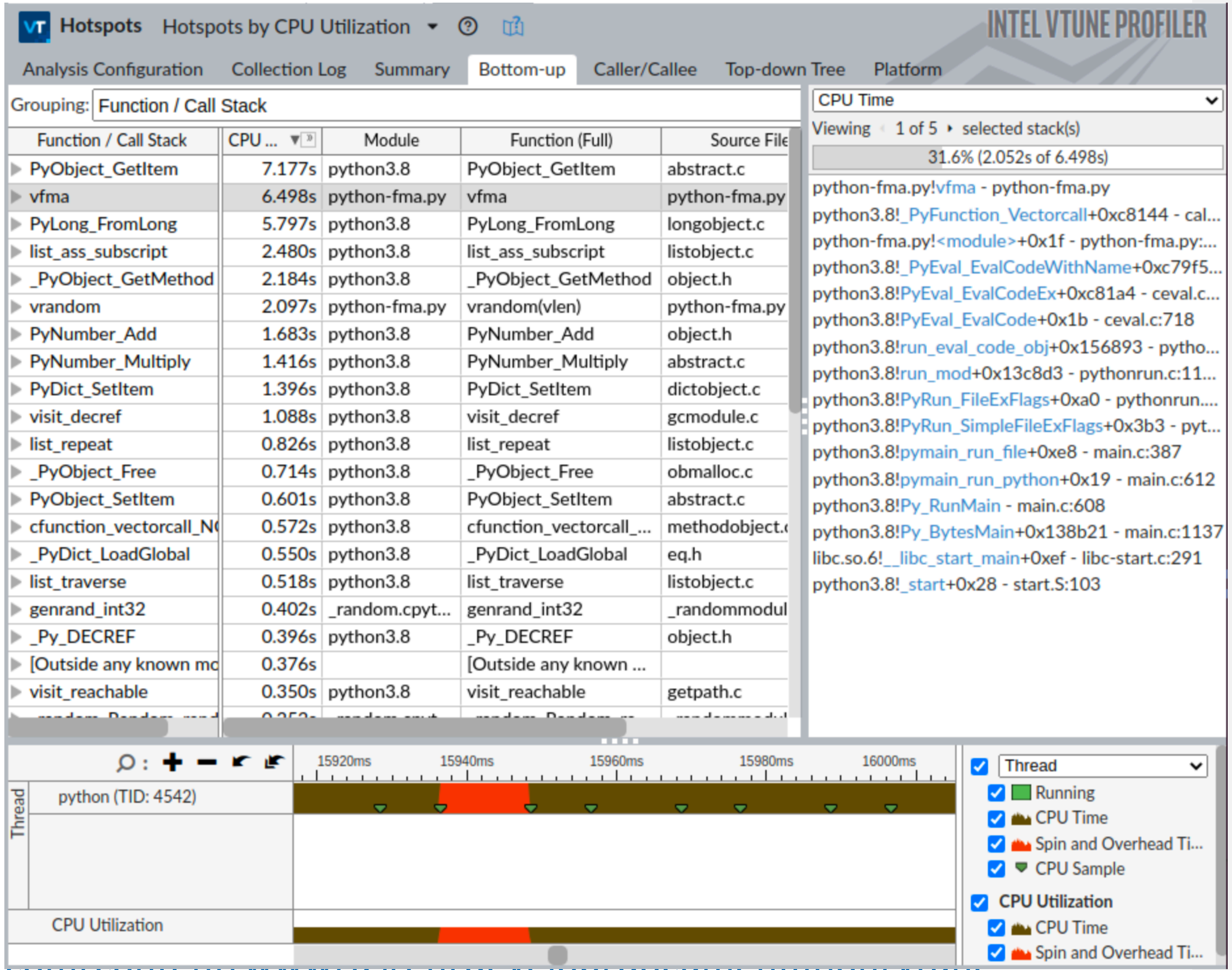
Extra

Explicit Python Support in VTune

Collections supported:

- User-Mode Hotspots
- Memory Consumption
- Threading

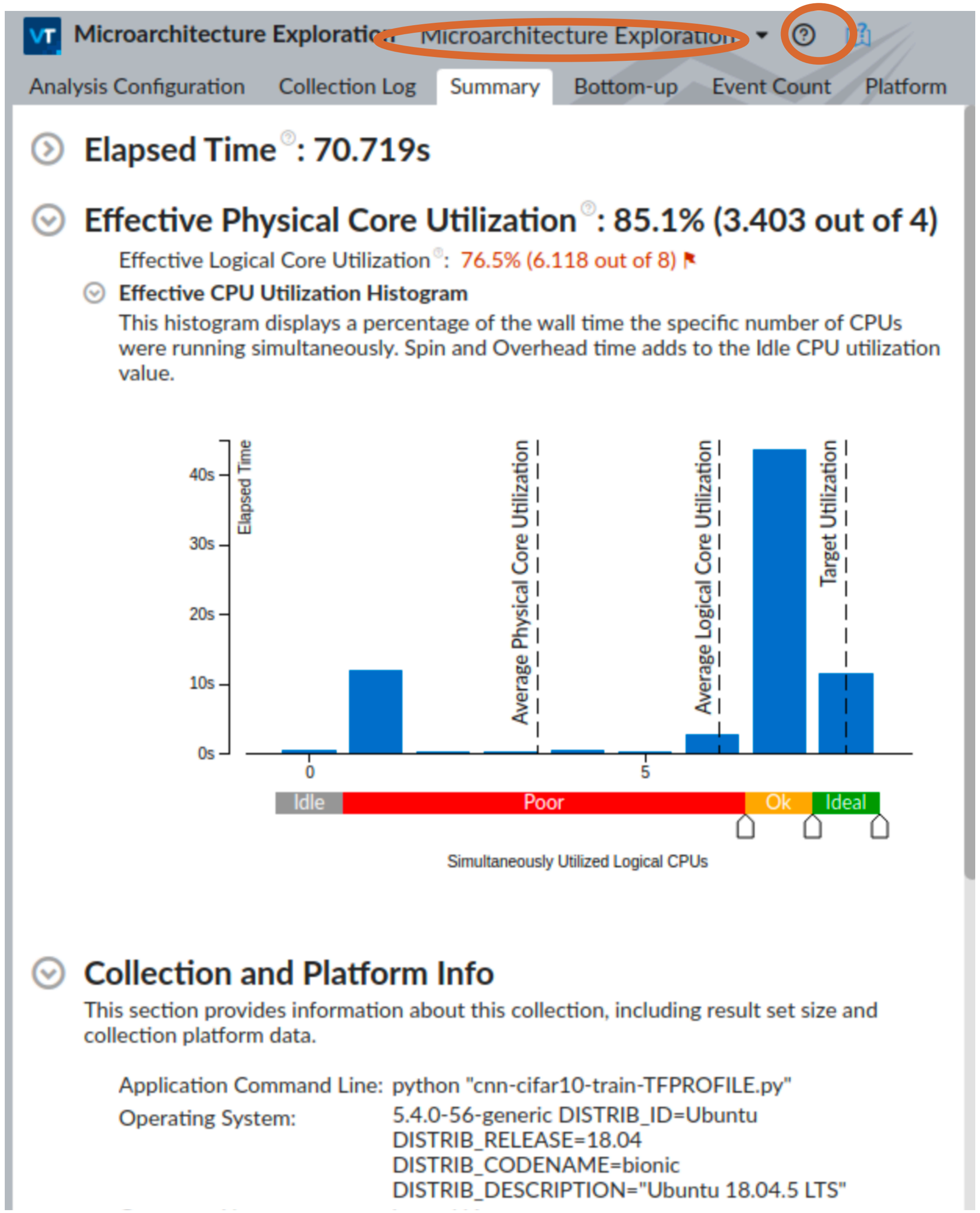
Python source is viewable in the Source View



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

VTune µArch Exploration: Summary View

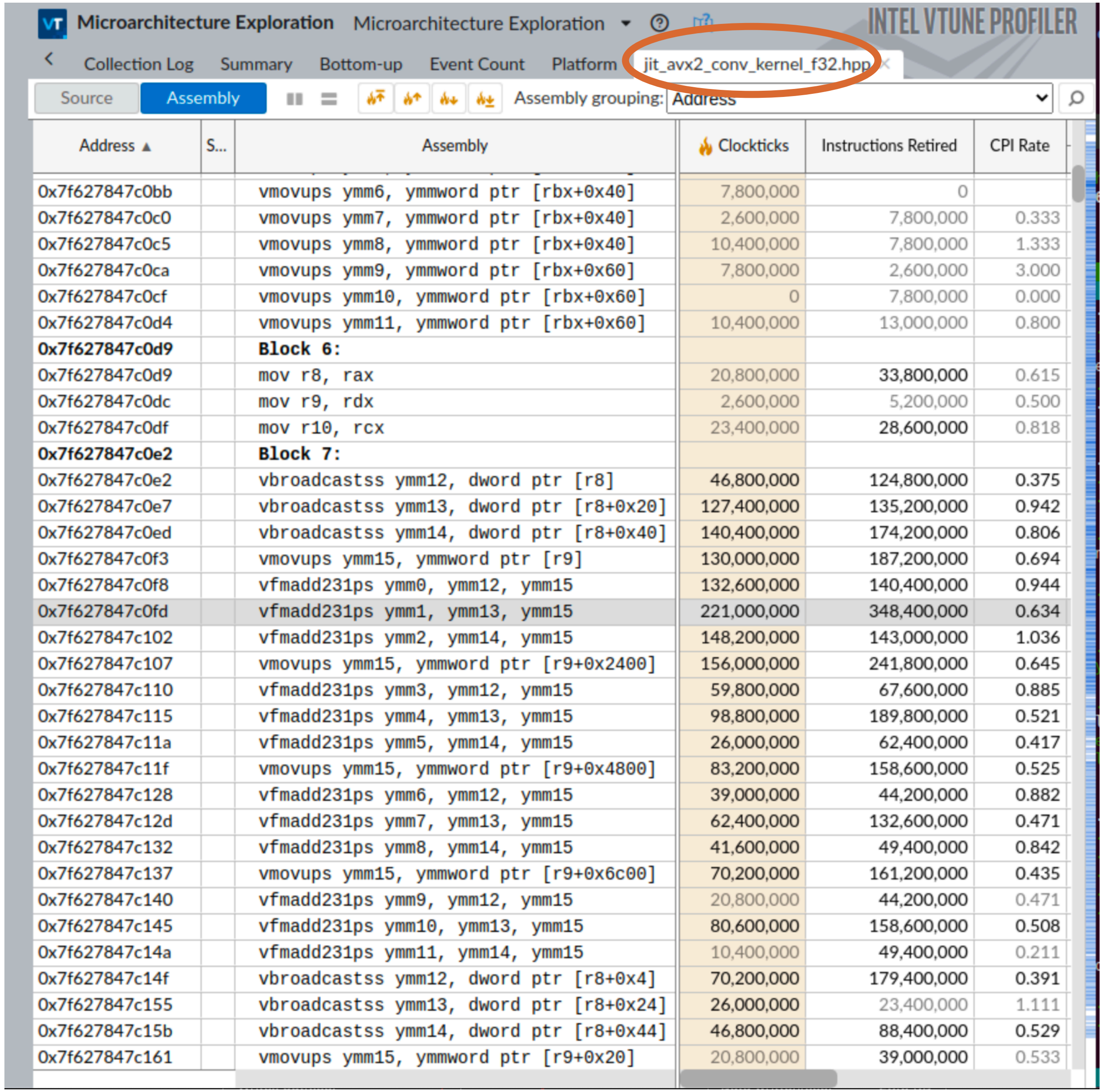
- Elapsed time shows wall-clock
  - Expanding this section shows overall µArch usage (example on next slide)
- Effective physical core utilization
  - I directed MKL to use all 4 physical cores on server (OMP_NUM_THREADS)
    - Setting sliders will apply this information to displayed data
  - 3.403 out of 4 cores used (85.1%) is decent
    - Overall, stalls/idle time likely due to TF startup and batch loading



Captured with dl-mini-workloads/cnn-cifar10-tf2 model

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
0x7f627847c0bb		vmovups ymm6, ymmword ptr [rbx+0x40]	7,800,000	0	
0x7f627847c0c0		vmovups ymm7, ymmword ptr [rbx+0x40]	2,600,000	7,800,000	0.333
0x7f627847c0c5		vmovups ymm8, ymmword ptr [rbx+0x40]	10,400,000	7,800,000	1.333
0x7f627847c0ca		vmovups ymm9, ymmword ptr [rbx+0x60]	7,800,000	2,600,000	3.000
0x7f627847c0cf		vmovups ymm10, ymmword ptr [rbx+0x60]	0	7,800,000	0.000
0x7f627847c0d4		vmovups ymm11, ymmword ptr [rbx+0x60]	10,400,000	13,000,000	0.800
0x7f627847c0d9		**Block 6:**			
0x7f627847c0d9		mov r8, rax	20,800,000	33,800,000	0.615
0x7f627847c0dc		mov r9, rdx	2,600,000	5,200,000	0.500
0x7f627847c0df		mov r10, rcx	23,400,000	28,600,000	0.818
0x7f627847c0e2		**Block 7:**			
0x7f627847c0e2		vbroadcastss ymm12, dword ptr [r8]	46,800,000	124,800,000	0.375
0x7f627847c0e7		vbroadcastss ymm13, dword ptr [r8+0x20]	127,400,000	135,200,000	0.942
0x7f627847c0ed		vbroadcastss ymm14, dword ptr [r8+0x40]	140,400,000	174,200,000	0.806
0x7f627847c0f3		vmovups ymm15, ymmword ptr [r9]	130,000,000	187,200,000	0.694
0x7f627847c0f8		vfmadd231ps ymm0, ymm12, ymm15	132,600,000	140,400,000	0.944
0x7f627847c0fd		vfmadd231ps ymm1, ymm13, ymm15	221,000,000	348,400,000	0.634
0x7f627847c102		vfmadd231ps ymm2, ymm14, ymm15	148,200,000	143,000,000	1.036
0x7f627847c107		vmovups ymm15, ymmword ptr [r9+0x2400]	156,000,000	241,800,000	0.645
0x7f627847c110		vfmadd231ps ymm3, ymm12, ymm15	59,800,000	67,600,000	0.885
0x7f627847c115		vfmadd231ps ymm4, ymm13, ymm15	98,800,000	189,800,000	0.521
0x7f627847c11a		vfmadd231ps ymm5, ymm14, ymm15	26,000,000	62,400,000	0.417
0x7f627847c11f		vmovups ymm15, ymmword ptr [r9+0x4800]	83,200,000	158,600,000	0.525
0x7f627847c128		vfmadd231ps ymm6, ymm12, ymm15	39,000,000	44,200,000	0.882
0x7f627847c12d		vfmadd231ps ymm7, ymm13, ymm15	62,400,000	132,600,000	0.471
0x7f627847c132		vfmadd231ps ymm8, ymm14, ymm15	41,600,000	49,400,000	0.842
0x7f627847c137		vmovups ymm15, ymmword ptr [r9+0x6c00]	70,200,000	161,200,000	0.435
0x7f627847c140		vfmadd231ps ymm9, ymm12, ymm15	20,800,000	44,200,000	0.471
0x7f627847c145		vfmadd231ps ymm10, ymm13, ymm15	80,600,000	158,600,000	0.508
0x7f627847c14a		vfmadd231ps ymm11, ymm14, ymm15	10,400,000	49,400,000	0.211
0x7f627847c14f		vbroadcastss ymm12, dword ptr [r8+0x4]	70,200,000	179,400,000	0.391
0x7f627847c155		vbroadcastss ymm13, dword ptr [r8+0x24]	26,000,000	23,400,000	1.111
0x7f627847c15b		vbroadcastss ymm14, dword ptr [r8+0x44]	46,800,000	88,400,000	0.529
0x7f627847c161		vmovups ymm15, ymmword ptr [r9+0x20]	20,800,000	39,000,000	0.533

Captured with dl-mini-workloads/cnn-cifar10-tf2 model

Further information: https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/viewing-source.html intel®