

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

The Intel logo, consisting of the word "intel" in a lowercase, sans-serif font, with a registered trademark symbol (®) to its upper right. The logo is positioned in the bottom left corner of the slide.

intel®

Deep Learning Profiling Domains

Domain	Conceptual	Operations	Tools
Distributed	Many Servers or Device Instances	<ul style="list-style-type: none">• MPI Operations• Horovod* Gradient Ops	<ul style="list-style-type: none">• Horovod* Timeline• MPI Analyzers
Model	<ul style="list-style-type: none">• Model Graph• Single Server	<ul style="list-style-type: none">• TensorFlow* Ops	<ul style="list-style-type: none">• TensorBoard*
Hardware	Within CPU or Device	<ul style="list-style-type: none">• Assembly Code• Hardware Counters• Micro-Ops	<ul style="list-style-type: none">• Intel® VTune™ Profiler

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

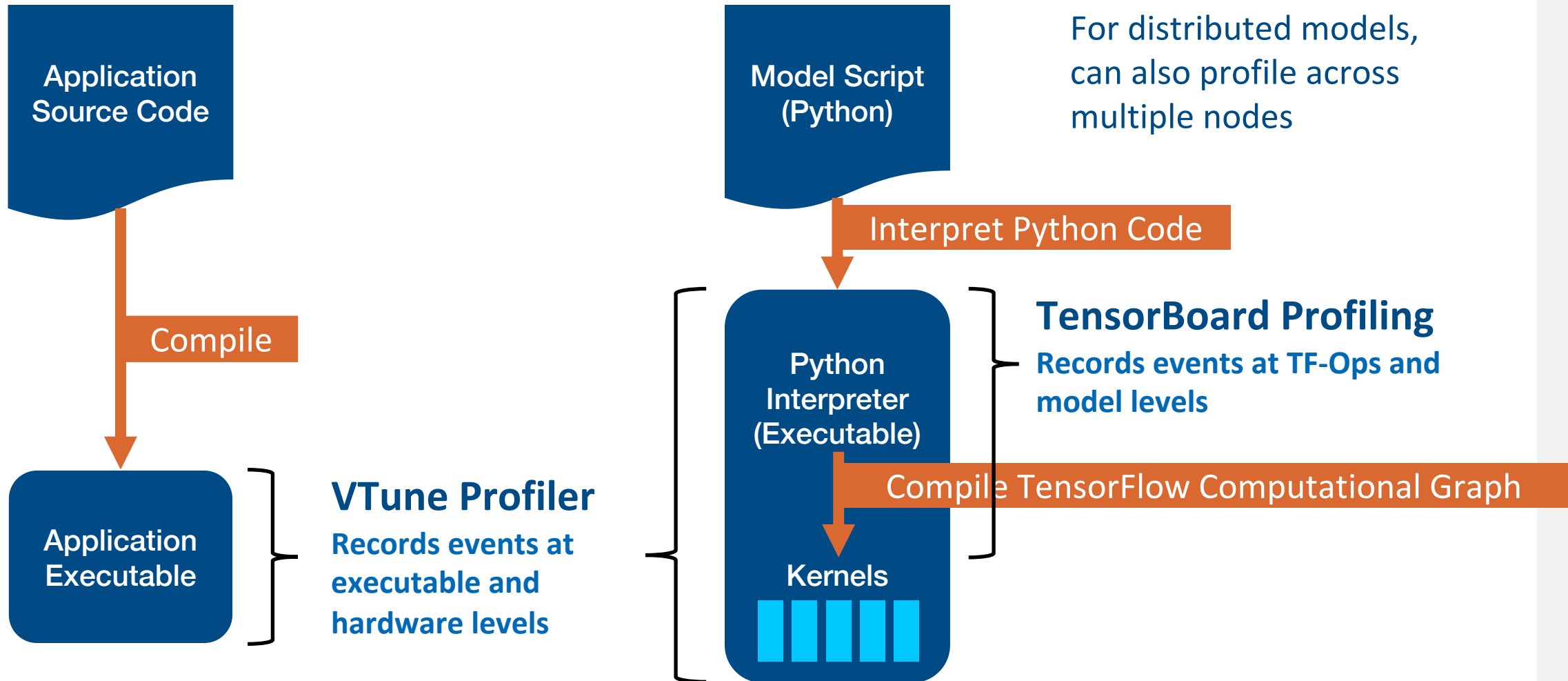
- cnn-cifar10-tf2
- cnn-cifar10-pytorch
- simple-mnist-tf1
- simple-mnist-tf2

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

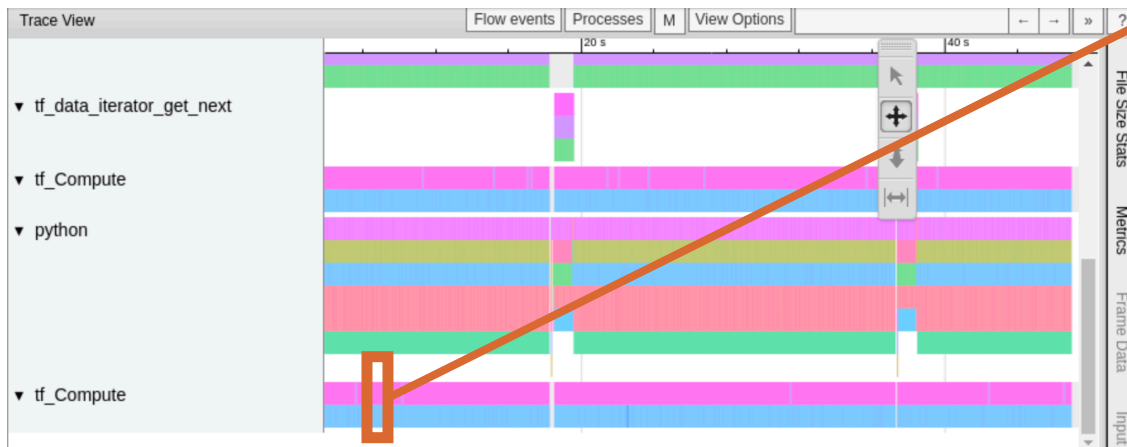


Model Domain Profiling: TensorBoard

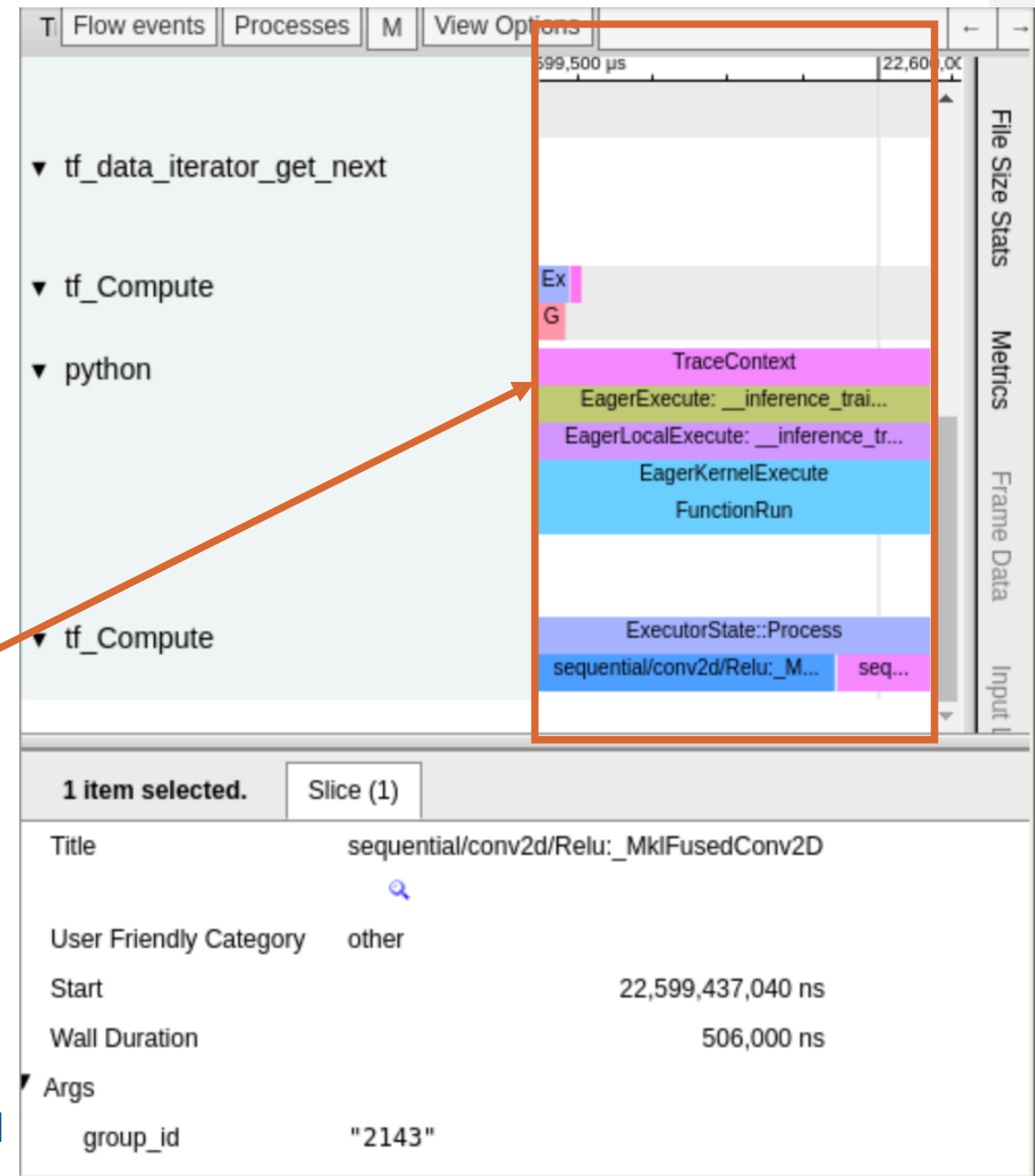
TensorFlow with TensorBoard

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

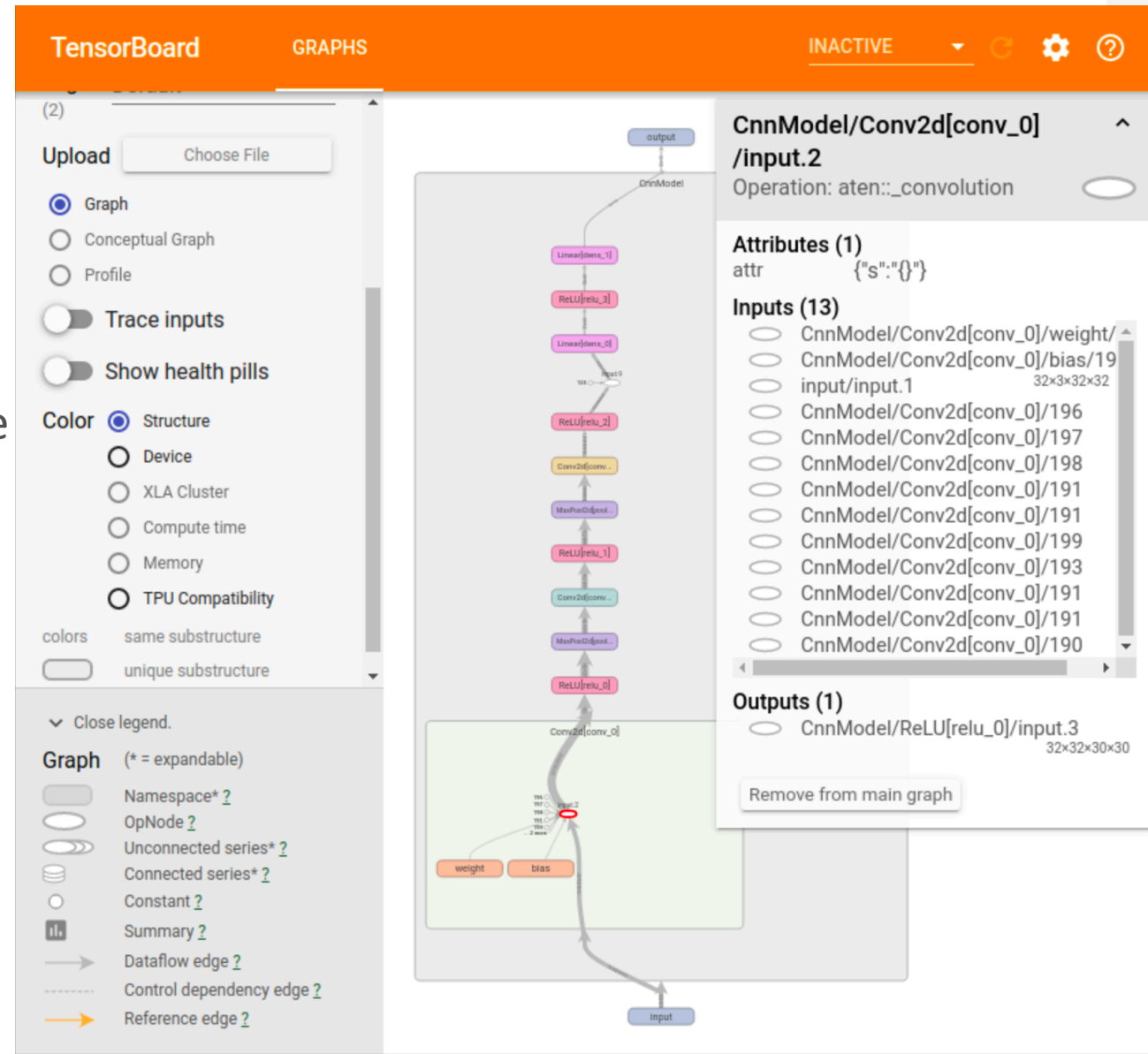
PyTorch with TensorBoard

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

Further information:

<https://pytorch.org/docs/stable/tensorboard.html>



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

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

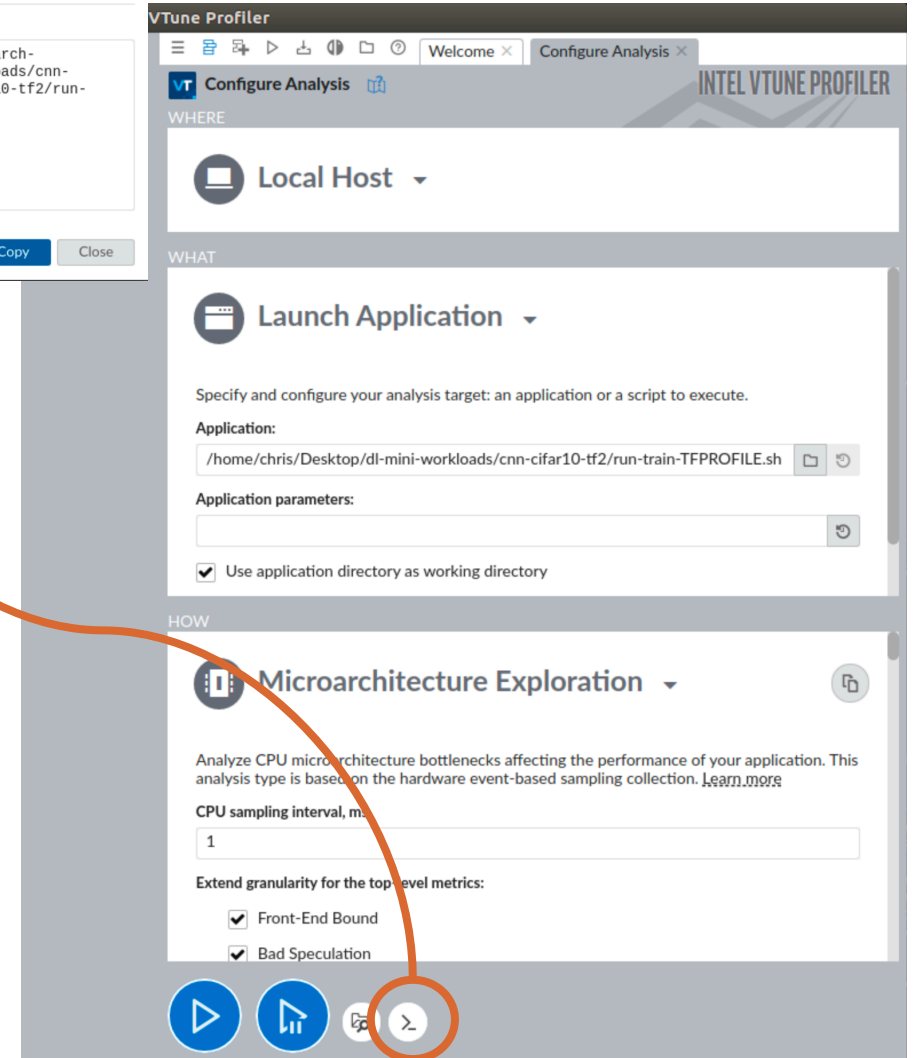
- GUI
 - Create project, configure analyses
 - Open previously collected *.vtune files for display
- Command-Line
 - Easy to include in scripts
 - Produces a directory “r000ue” with “r000ue/r000ue.vtune” file (and other files)
 - Can run collection on a remote server, then display results (r000ue.vtune 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)

```
Copy Command Line to Clipboard
```

Command line:

```
/home/chris/intel/oneapi/vtune/2021.1.1/bin64/vtune -collect uarch-exploration -app-working-dir /home/chris/Desktop/dl-mini-workloads/cnn-cifar10-tf2 -- /home/chris/Desktop/dl-mini-workloads/cnn-cifar10-tf2/run-train-TFPROFILE.sh
```

Copy Close



VTune Command Line

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

```
$ vtune -collect uarch-exploration -app-working-dir /tmp -- python mnist_infer_PROFILE.py
```

VTune GUI

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 Performance Snapshot

Analysis Configuration | Collection Log | Summary

Choose your next analysis type

Select a highlighted recommendation based on your performance snapshot.

ALGORITHM

- Hotspots
- Anomaly Detection (preview)
- Memory Consumption

PARALLELISM

- Threading 16.4%
- HPC Performance Characterization

ACCELERATORS

- GPU Offload (preview)
- GPU Compute/Media Hotspots (preview)
- CPU/FPGA Interaction (preview)

MICROARCHITECTURE

- Microarchitecture Exploration 23.5%
- Memory Access 35.1%

I/O

- Input and Output

PLATFORM ANALYSES

- System Overview
- Throttling (preview)
- Platform Profiler

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: python "cnn-cifar10-train.py"

Operating System: 3.10.0-1062.12.1.el7.x86_64 NAME="CentOS Linux" VERSION="7 (Core)" ID="centos" ID_LIKE="rhel fedora" VERSION_ID="7" PRETTY_NAME="CentOS Linux 7 (Core)" ANSI_COLOR="0;31" CPE_NAME="cpe:/o:centos:centos:7" HOME_URL="https://www.centos.org/" BUG_REPORT_URL="https://bugs.centos.org/" CENTOS_MANTISBT_PROJECT="CentOS-7" CENTOS_MANTISBT_PROJECT_VERSION="7" REDHAT_SUPPORT_PRODUCT="centos" REDHAT_SUPPORT_PRODUCT_VERSION="7"

Computer Name: heat-sk13.jf.intel.com

Result Size: 6 MB

Collection start time: 20:44:24 15/12/2020 UTC

Collection stop time: 20:47:06 15/12/2020 UTC

Collector Type: Event-based counting driver

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed

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

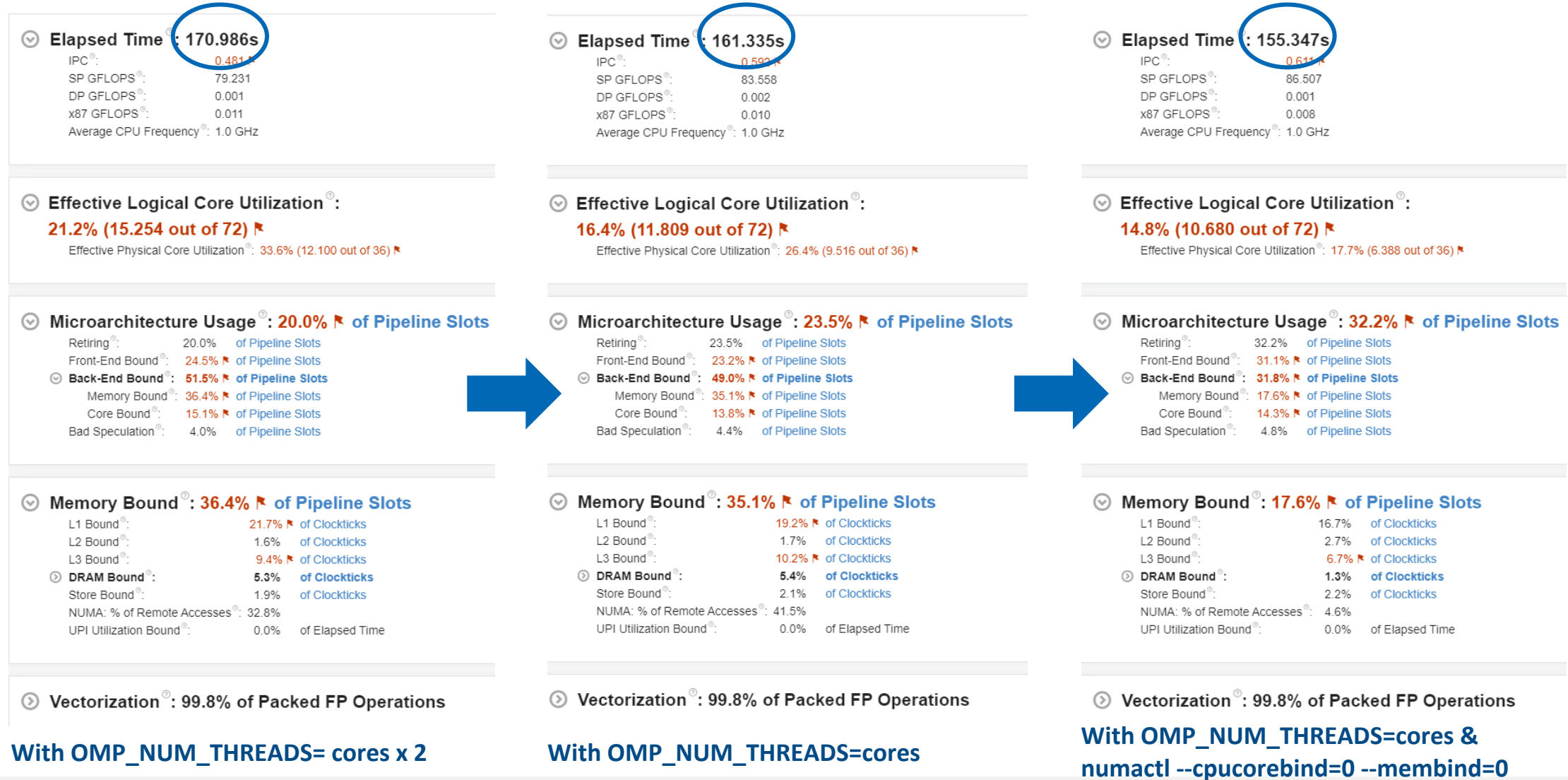
Memory Bound[®]: 35.1% of Pipeline Slots

- 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

- Instruction Mix:
 - SP FLOPs[®]: 57.9% of uOps
 - DP FLOPs[®]: 0.0% of uOps
 - Packed[®]: 49.9% from DP FP**

VTune: Performance Snapshot (Skylake Server)



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

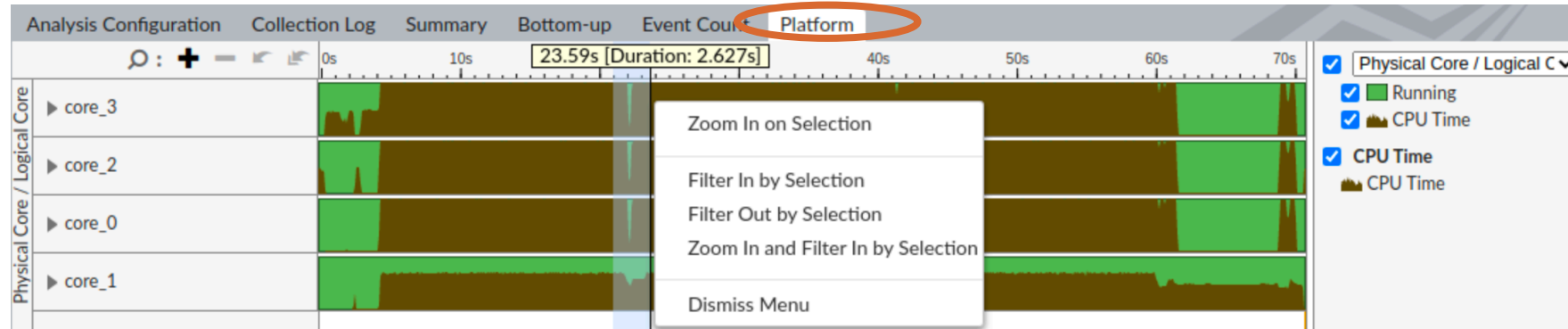
```
(/home/nalinik/miniconda3/py36_envs/ipw) [nalinik@heat-sk13 cnn-cifar10-tf2]$ head -10 cifar10-nhwc.log
dnnl_verbose,info,oneDNN v1.4.0 (commit N/A)
dnnl_verbose,info,cpu,runtime:OpenMP info
dnnl_verbose,info,cpu,isa:Intel AVX-512 with AVX512BW, AVX512VL, and AVX512DQ extensions
dnnl_verbose,info,gpu,runtime:none
dnnl_verbose,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:acdb:f0 dst_f32::blocked:abcd:f0,,32x3x32x32,2.20996 execution time in ms
dnnl_verbose,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:cdba:f0 dst_f32::blocked:Acdb16a:f0,,32x3x3x3,0.0009765
62 engine, primitive name
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 fused operations algorithm used
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
```

Further information:

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

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

VTune μ Architecture 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 μ Arch: 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.
 - μ Pipe 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 μ Pipe for explanations and recommendations from VTune

Cannot find source file

external\mkl_dnn_v1\src\cpu\jit_avx2_conv_kernel_f32.hpp

Function / Call Stack	CPU Time	CPI Rate	Instructions Reti...	Retiring
[_pywrap_tensorflow_internal.so]	78.695s	0.941	257,844,600,000	47.7%
[vmlinux]	186.820s	2.398	241,641,400,000	29.9%
jit_avx2_conv_fwd_kernel_f32	12.611s	0.591	66,060,800,000	74.7%
jit_avx2_conv_fwd_kernel_f32	11.767s	0.605	59,979,400,000	63.9%
[libtensorflow_framework.so.2]	10.739s	0.659	52,130,000,000	55.6%
jit_avx2_conv_bwd_data_kernel_f32	9.653s	0.598	50,120,200,000	78.3%
jit_avx2_conv_bwd_data_kernel_f32	9.352s	0.598	48,609,600,000	88.3%
[libc-2.27.so]	9.830s	1.549	19,601,400,000	38.2%
jit_avx2_conv_fwd_kernel_f32	3.147s	0.589	16,980,600,000	86.0%
jit_avx2_conv_fwd_kernel_f32	2.905s	0.585	15,631,200,000	71.5%
[python3.7]	4.985s	1.105	14,323,400,000	23.0%
[libiomp5.so]	22.083s	5.125	13,358,800,000	16.0%
jit_avx2_conv_fwd_kernel_f32	2.832s	0.667	13,358,800,000	59.1%
jit_avx2_conv_fwd_kernel_f32	2.538s	0.668	12,417,600,000	56.7%
[_pywrap_profiler.so]	1.536s	0.497	10,173,800,000	50.0%
jit_avx2_conv_bwd_data_kernel_f32	2.158s	0.687	10,132,200,000	71.7%
jit_avx2_conv_bwd_data_kernel_f32	2.281s	0.693	9,919,000,000	73.1%
[ld-2.27.so]	1.349s	0.629	7,069,400,000	66.4%
[_multiarray_umath.cpython-37m-x86_64-linux-gnu.so]	1.024s	0.500	6,533,800,000	60.8%
[libmkl_avx2.so]	1.292s	0.640	5,834,400,000	76.1%
[libcrypto.so.1.1]	0.380s	0.280	4,737,200,000	100.0%
jit_uni_relu_kernel_float	1.750s	1.532	3,575,000,000	26.4%
jit_uni_relu_kernel_float	1.831s	1.635	3,359,200,000	41.3%

Microarchitecture Usage: 74.7% of Pipeline Slots

μ Pipe

- Retiring: 74.7%
- Front-End Bound: 34.0%
- Front-End Latency: 24.9%
- ICache Misses: 0.0%
- ITLB Overhead: 0.0%
- Branch Resteers: 13.5%
- Mispredicts Resteers: 0.0%
- Clears Resteers: 13.5%
- Unknown Branches: 0.0%
- DSB Switches: 0.5%
- Length Changing Prefixes: 0.0%
- MS Switches: 0.7%
- Front-End Bandwidth: 9.0%
- Bad Speculation: 0.0%
- Back-End Bound: 0.0%

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

VTune μ Arch: 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

Microarchitecture Exploration

Analysis Configuration Collection Log Summary Bottom-up **Event Count** Platform

Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	Instructions Retired	CPI Rate	FP_ARITH_INST...
			FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE
jit_avx2_conv_fwd_kernel_f32	184,540,200,000	0.606	213,867,294,488
▶ jit_avx2_conv_bwd_weights_kernel_f32	200,655,000,000	0.586	169,944,265,236
▶ jit_avx2_conv_bwd_data_kernel_f32	118,812,200,000	0.613	138,823,092,977
▶ [libmkl_avx2.so]	5,834,400,000	0.640	5,496,615,488
▶ jit_avx_gemm_f32_xbyak_gemm	3,000,400,000	0.726	3,623,816,325
▶ [_pywrap_tensorflow_internal.so]	257,844,600,000	0.941	1,417,051,056
▶ jit_uni_relu_kernel_float	9,882,600,000	1.439	1,031,216,690
▶ [kvm.ko]	0		0
▶ [_pywrap_tf_cluster.so]	0		0
▶ [libhdf5.so.103.2.0]	0		0
▶ [_pywrap_mlir.so]	0	0.000	0
▶ [_mt19937.cpython-37m-x86_64-linux-g	0	0.000	0
▶ [ip_tables.ko]	2,600,000	6.000	0
▶ [libitnotify_collector.so]	2,600,000	25.000	0
▶ [_json.cpython-37m-x86_64-linux-gnu.so	2,600,000	1.000	0
▶ [iptables_filter.ko]	2,600,000	2.000	0
▶ [_random.cpython-37m-x86_64-linux-gn	2,600,000	0.000	0
▶ func@0x860	2,600,000	3.000	0
▶ [nf_nat.ko]	5,200,000	1.000	0
▶ [e1000e.ko]	7,800,000	28.000	0
▶ [libmkl_intel_thread.so]	15,600,000	10.667	0
▶ [Unknown stack frame(s)]	0	0.000	0

MACHINE_CLEARS.COUNT 32,311,021

MEM_INST_RETIRED.ALL_STORES_PS 2,907,366,498

MEM_INST_RETIRED.STLB_MISS_STORES_PS 3,228,867

MEM_LOAD_L3_HIT_RETIRED.XSNP_HITM_PS 1,292,218

MEM_LOAD_L3_HIT_RETIRED.XSNP_HIT_PS 1,295,777

MEM_LOAD_RETIRED.FB_HIT_PS 3,072,334,891

MEM_LOAD_RETIRED.L1_HIT_PS 66,908,522,118

MEM_LOAD_RETIRED.L1_MISS_PS 2,645,153,585

MEM_LOAD_RETIRED.L2_HIT_PS 2,593,348,764

MEM_LOAD_RETIRED.L3_HIT_PS 35,535,614

OFFCORE_REQUESTS_BUFFER_SQUARED_FULL 256,914,579

OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD:cmask=4 1,676,160,328

OFFCORE_REQUESTS_OUTSTANDING_CYCLES_WITH_DATA_RD 7,874,675,555

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

Further information:

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

Intel VTune Profiler - Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Source Function / Function / Call Stack

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]

Hybrid: Including Model-Level Profiling in VTune Timeline

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

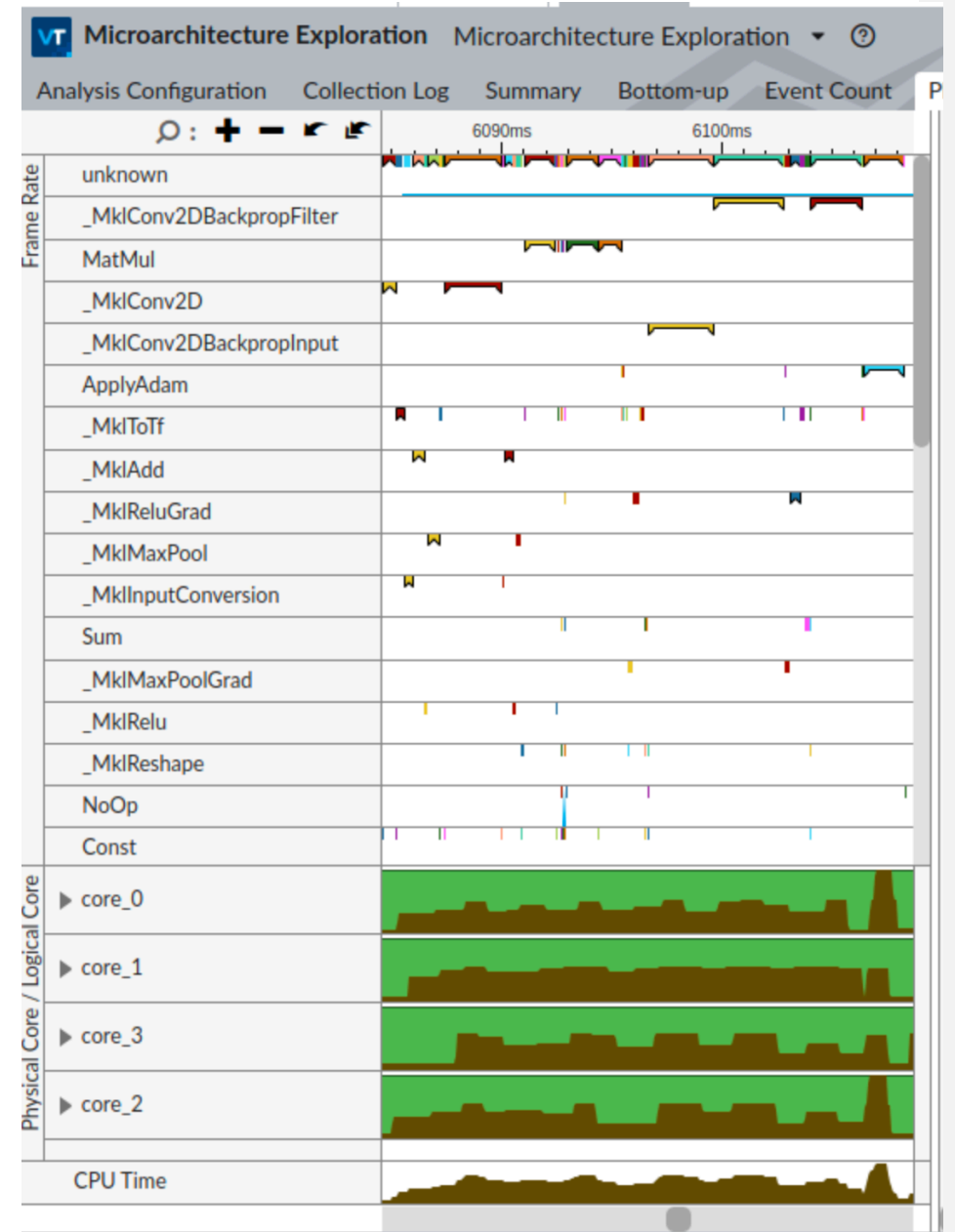
- External data needs to be converted to a simple CSV format
- Times need to be in absolute system time
- VTune integrates this data during finalization
- Integrated and displayed into VTune's timeline

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

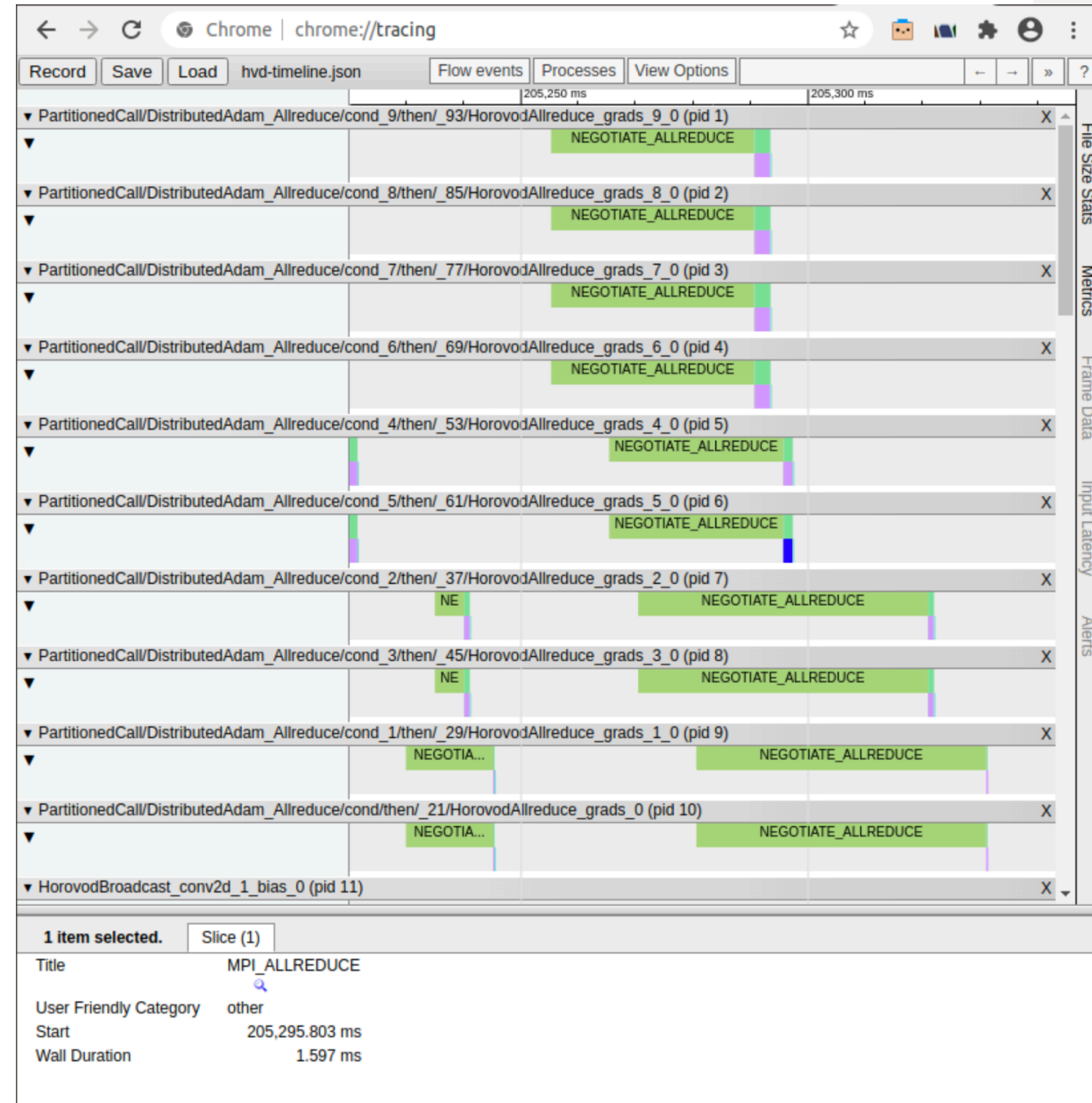
- Built on MPI
- Available for TensorFlow and PyTorch

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

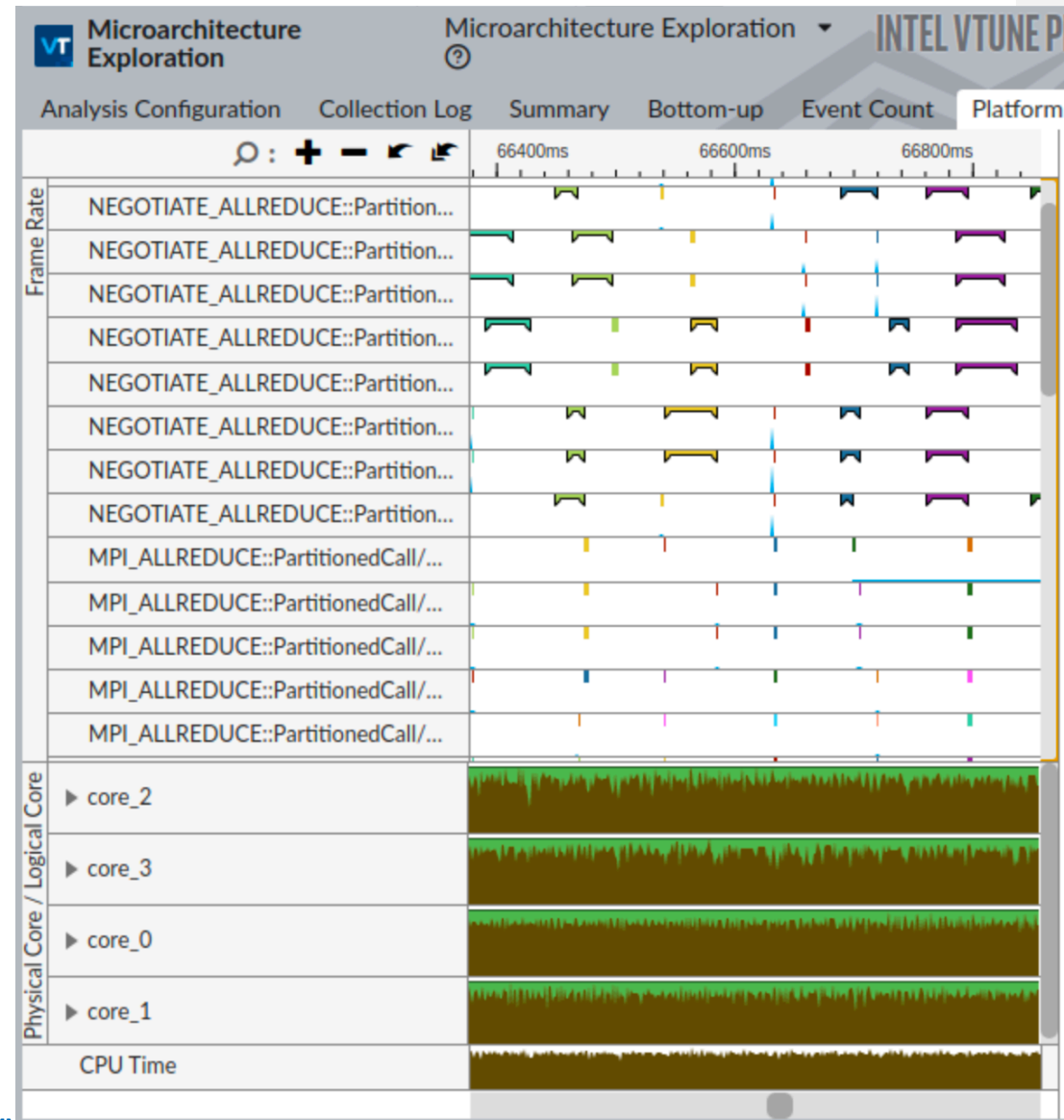


Hybrid: Horovod Timeline in VTune Timeline

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 Analyzer

- **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

Further information:

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

- 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

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.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

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>

The screenshot displays the Intel VTune Profiler interface. The top navigation bar includes 'Hotspots by CPU Utilization', 'Analysis Configuration', 'Collection Log', 'Summary', 'Bottom-up', 'Caller/Callee', 'Top-down Tree', and 'Platform'. The 'Bottom-up' view is active, showing a table of hotspots. The table has columns for 'Function / Call Stack', 'CPU ...', 'Module', 'Function (Full)', and 'Source File'. The 'vfma' function in 'python-fma.py' is the most significant hotspot, taking 6.498s. A detailed view of this function is shown on the right, displaying the source code for 'python-fma.py!vfma'. The bottom of the interface shows a timeline for 'python (TID: 4542)' and 'CPU Utilization'.

Function / Call Stack	CPU ...	Module	Function (Full)	Source File
PyObject_GetItem	7.177s	python3.8	PyObject_GetItem	abstract.c
vfma	6.498s	python-fma.py	vfma	python-fma.py
PyLong_FromLong	5.797s	python3.8	PyLong_FromLong	longobject.c
list_ass_subscript	2.480s	python3.8	list_ass_subscript	listobject.c
_PyObject_GetMethod	2.184s	python3.8	_PyObject_GetMethod	object.h
vrandom	2.097s	python-fma.py	vrandom(vlen)	python-fma.py
PyNumber_Add	1.683s	python3.8	PyNumber_Add	object.h
PyNumber_Multiply	1.416s	python3.8	PyNumber_Multiply	abstract.c
PyDict_SetItem	1.396s	python3.8	PyDict_SetItem	dictobject.c
visit_decref	1.088s	python3.8	visit_decref	gcmodule.c
list_repeat	0.826s	python3.8	list_repeat	listobject.c
_PyObject_Free	0.714s	python3.8	_PyObject_Free	obmalloc.c
PyObject_SetItem	0.601s	python3.8	PyObject_SetItem	abstract.c
cfunction_vectorcall_N	0.572s	python3.8	cfunction_vectorcall_...	methodobject.c
_PyDict_LoadGlobal	0.550s	python3.8	_PyDict_LoadGlobal	eq.h
list_traverse	0.518s	python3.8	list_traverse	listobject.c
genrand_int32	0.402s	_random.cpyt...	genrand_int32	_randommodul
_Py_DECREF	0.396s	python3.8	_Py_DECREF	object.h
[Outside any known mc	0.376s		[Outside any known ...	
visit_reachable	0.350s	python3.8	visit_reachable	getpath.c

VTune μ Arch Exploration: Summary View

- Elapsed time shows wall-clock
 - Expanding this section shows overall μ Arch 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

Microarchitecture Exploration

Analysis Configuration Collection Log Summary Bottom-up Event Count Platform

Elapsed Time[Ⓢ]: 70.719s

Effective Physical Core Utilization[Ⓢ]: 85.1% (3.403 out of 4)
Effective Logical Core Utilization[Ⓢ]: 76.5% (6.118 out of 8)

Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

Simultaneously Utilized Logical CPUs

Collection and Platform Info
This section provides information about this collection, including result set size and collection platform data.

Application Command Line: python "cnn-cifar10-train-TFPROFILE.py"
Operating System: 5.4.0-56-generic DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.5 LTS"

VTune μ Arch: 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

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

intel®