TensorFlow

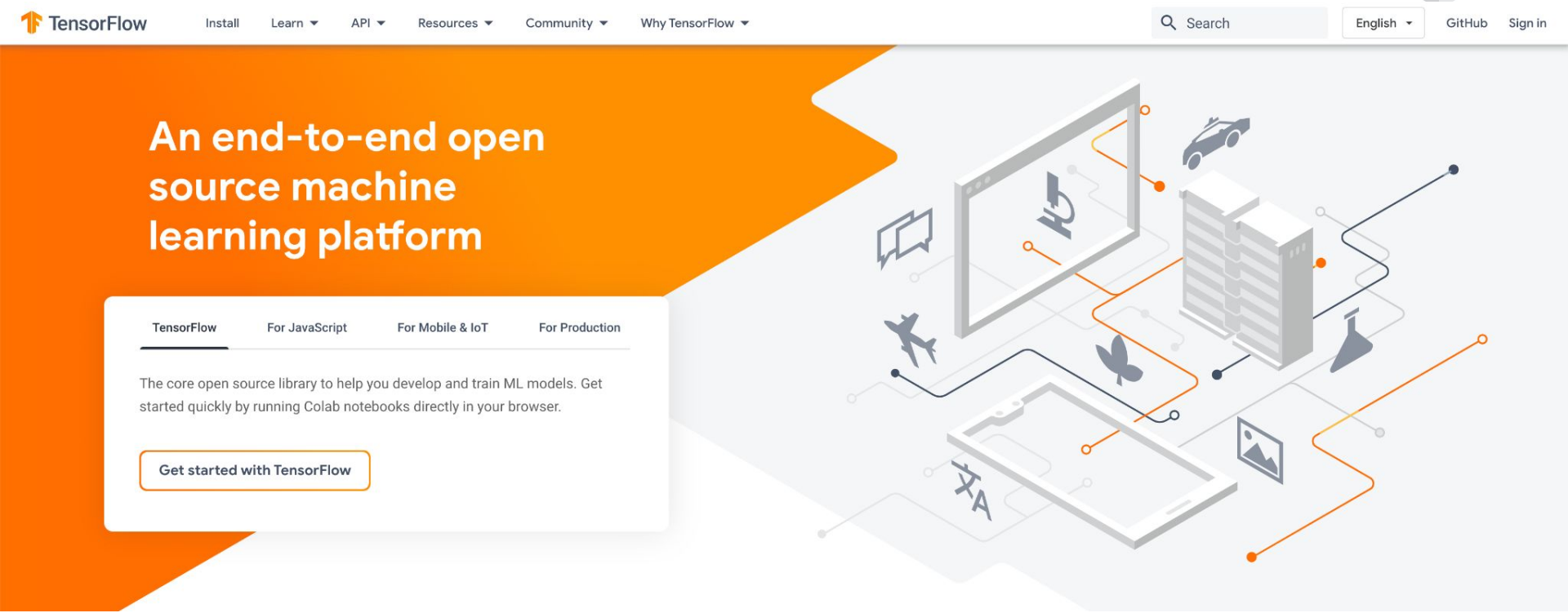
TensorFlow and oneDNN in Partnership

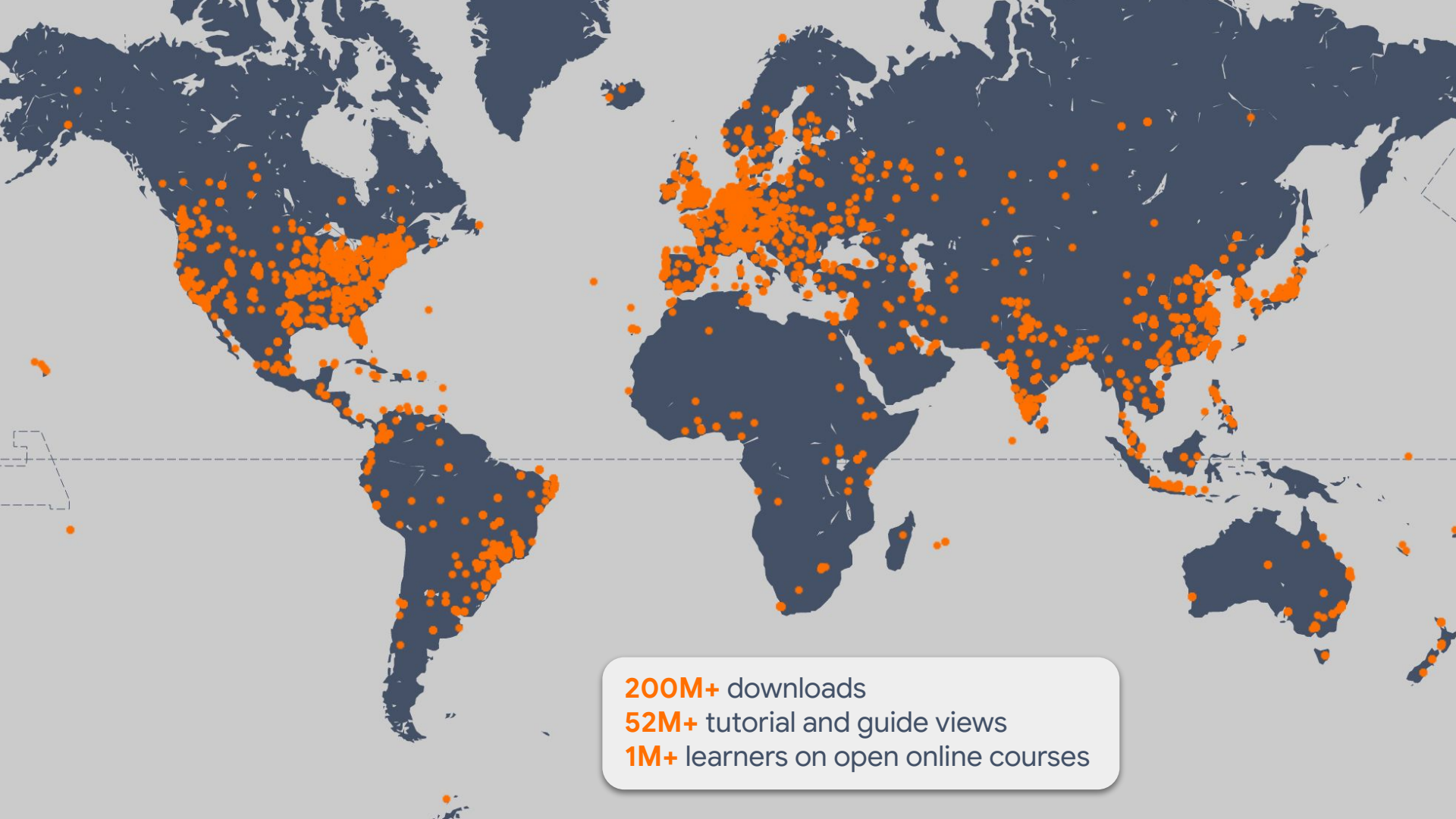
Penporn Koanantakool
Google

oneAPI Developer Summit at ISC, June 22, 2021

Presenting the work of many people from Google and Intel

What is TensorFlow?





200M+ downloads
52M+ tutorial and guide views
1M+ learners on open online courses

Agenda

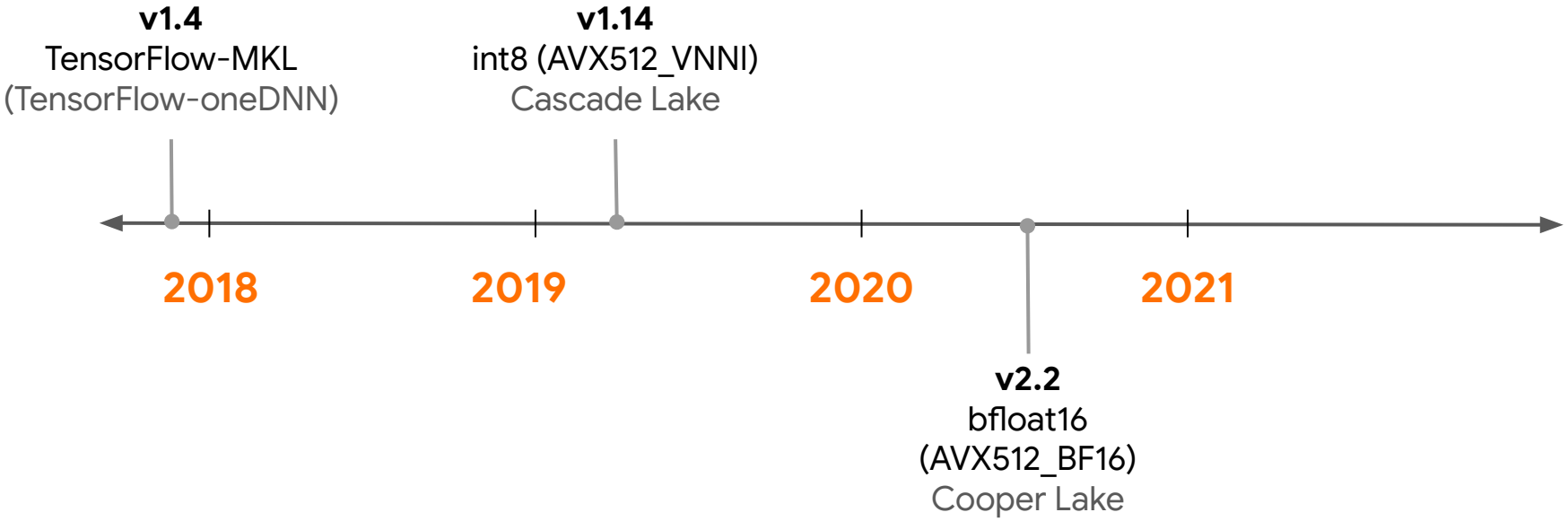
- Intel-optimized TensorFlow (TensorFlow-oneDNN)
  - What it does
  - Where it is used
  - Recent optimizations: bfloat16 and int8 vectorizations
  - Arm aarch64 support
- Vanilla TensorFlow
  - oneDNN experimental flag in default TF packages
- Pluggable Device

Intel-optimized TensorFlow (TensorFlow-oneDNN)

`--config=mkl`

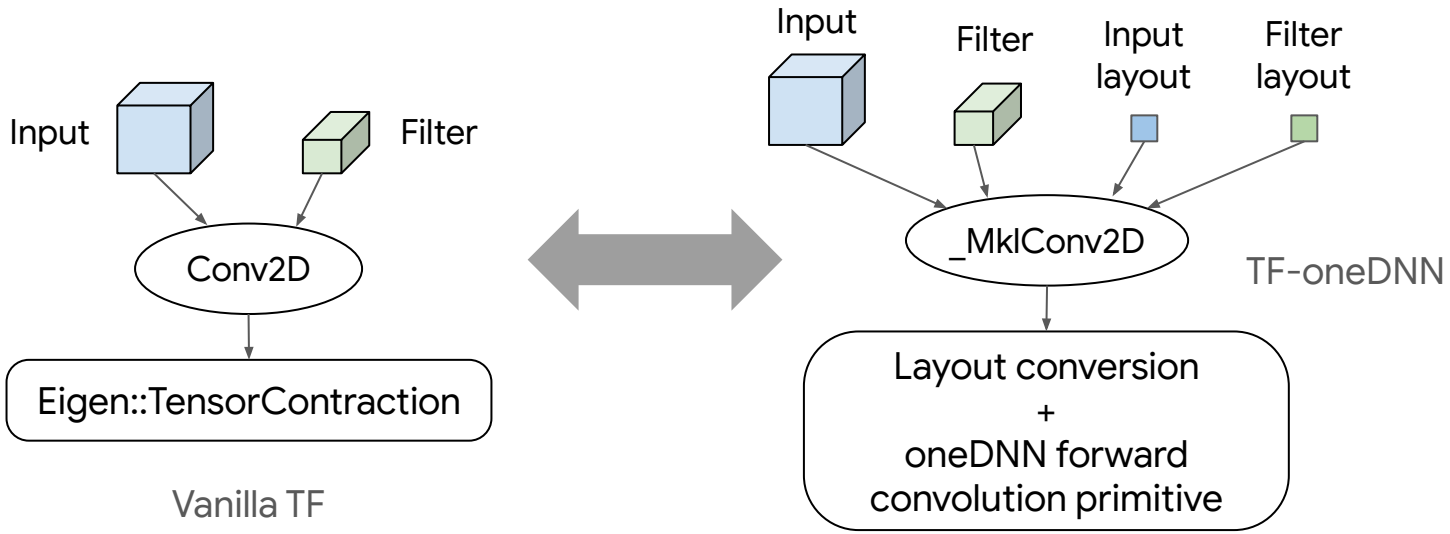
>3 years of collaboration

Some highlights



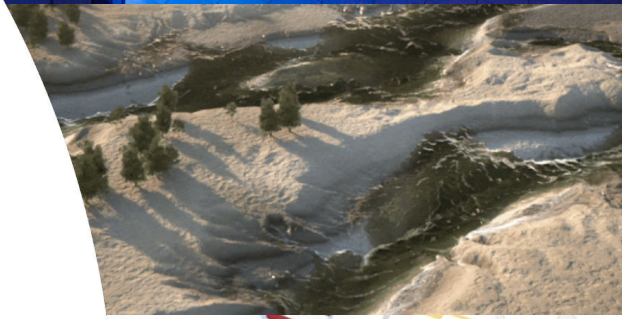
TensorFlow-oneDNN

- Replaces some of the most compute-intensive vanilla TF ops with custom oneDNN ops.
- Has optimizations such as op fusions and primitive caching.
- x86 and Arm backends.



Users

- Google Cloud Deep Learning VMs, containers, notebooks, etc.
  - Also on AWS and Azure.
- Supercomputers:
  - Cori / Perlmutter @NERSC,
  - Fugaku @RIKEN (Arm backend), etc.
- Google Health's DeepVariant
  - Open-source tool for analyzing DNA sequence.
- Tencent's 3D digital face reconstruction
  - For games, education, e-commerce, etc.
- Laika's stop motion animation



int8 vectorization

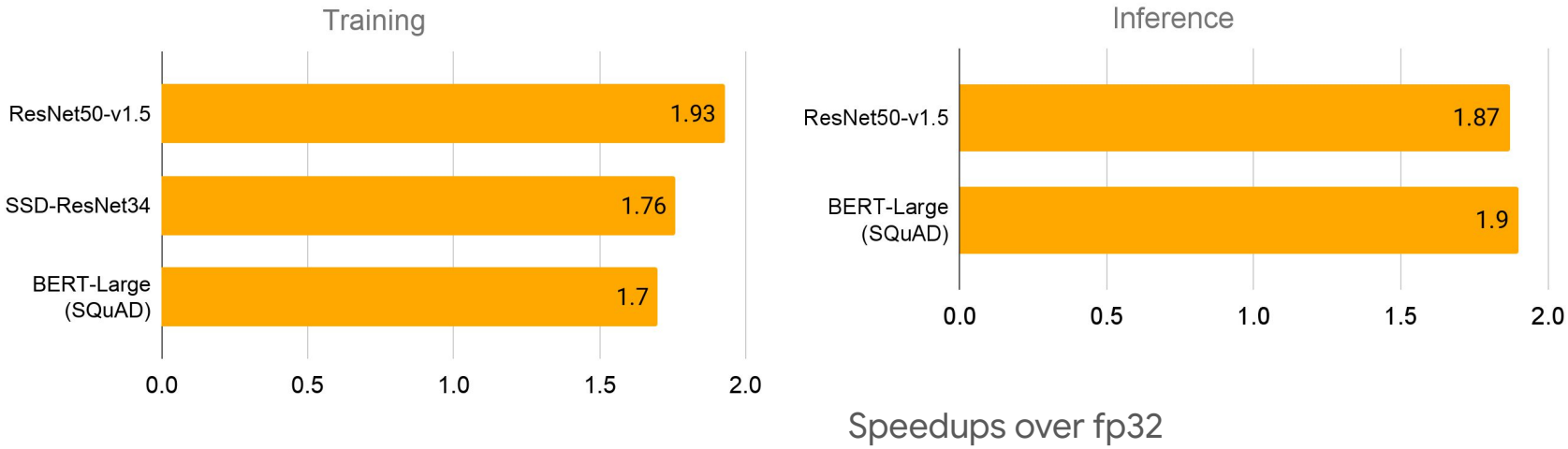
- Utilizes AVX512_VNNI, first available in Cascade Lake.
  - Up to 4× speedups over fp32.
- Quantize models using Intel® Low Precision Optimization Tool (LPOT)

Model	Top-1 Accuracy		Throughput Speedup
	FP32 (Skylake)	INT8 (Cascade Lake)	
ResNet-50	74.30	73.75	3.9×
ResNet-101	76.40	75.66	4.0×
InceptionV3	76.75	76.51	3.1×

Accelerating TensorFlow Inference with Intel® Deep Learning Boost on 2nd Gen Intel® Xeon® Scalable Processors

bfloat16 vectorization

- Utilizes AVX512_BF16, first available in Cooper Lake.
  - ~2x speedups over fp32 for both mixed-precision training and inference.
- Can be used through Keras mixed-precision API.

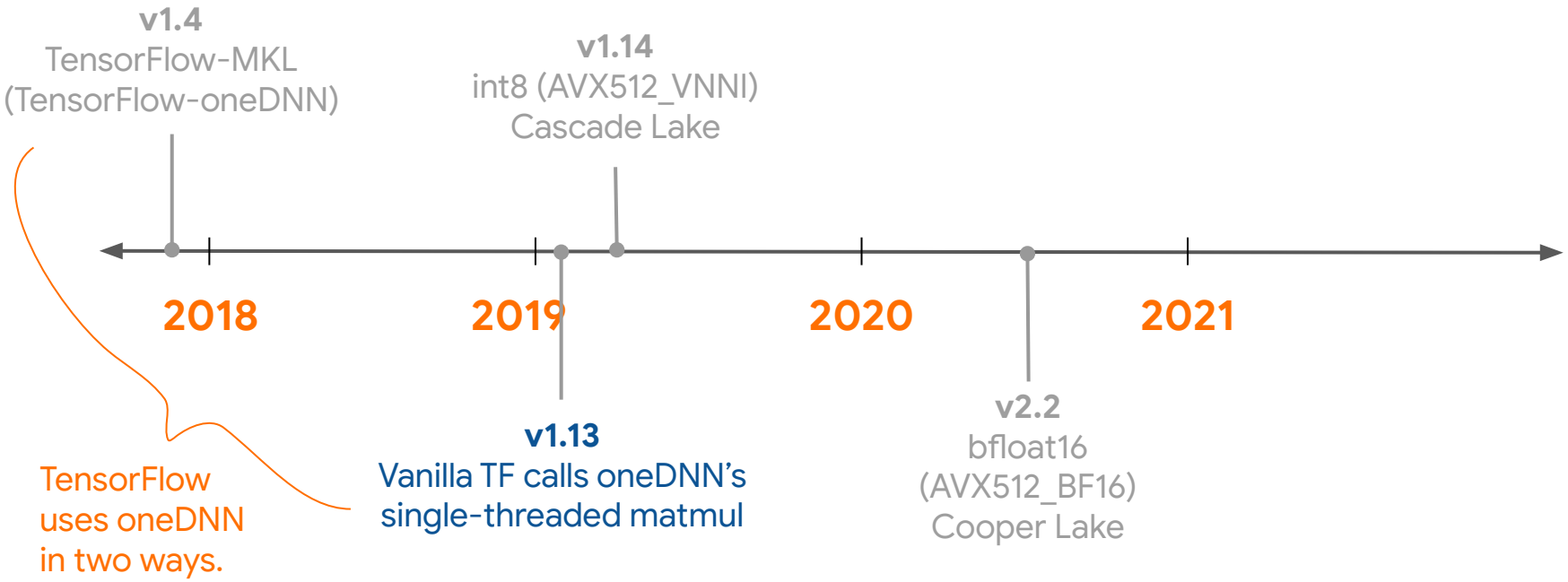


Accelerating AI performance on 3rd Gen Intel® Xeon® Scalable processors with TensorFlow and Bfloat16

Vanilla TensorFlow

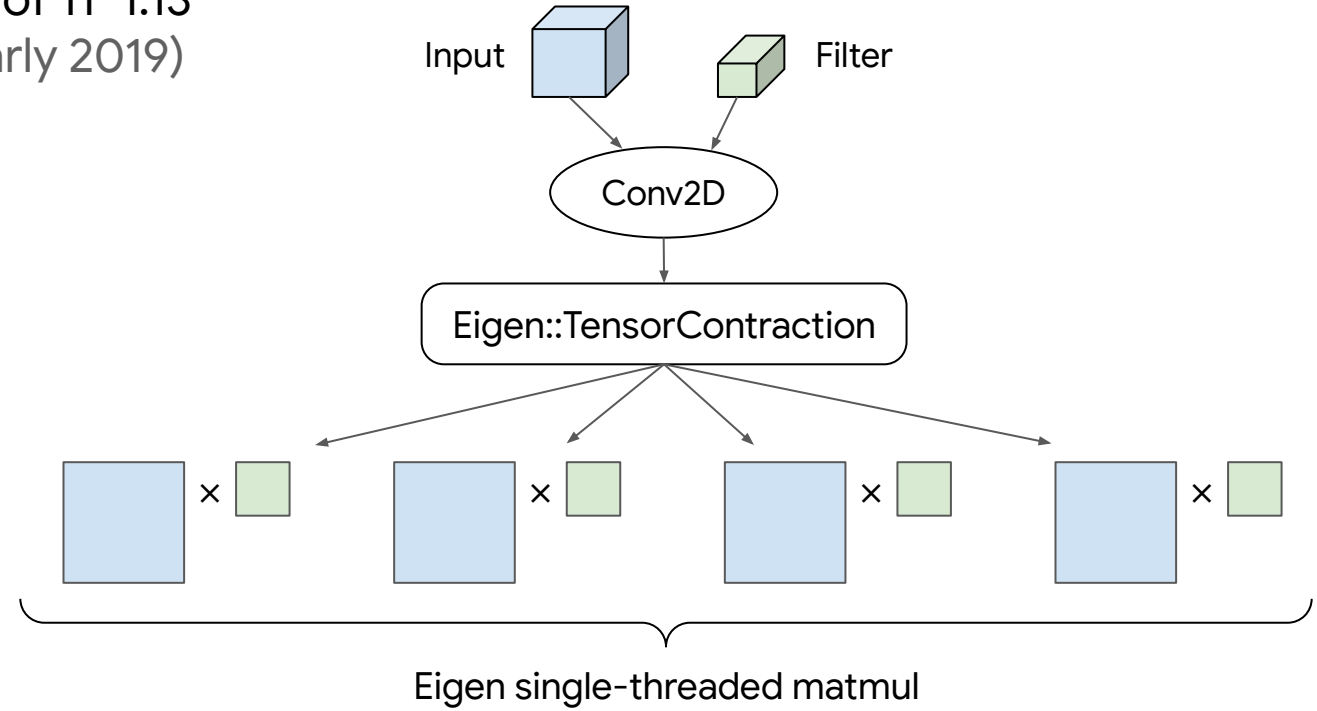
Another way to call oneDNN

oneDNN uses OpenMP, which cannot coordinate with TF's thread pool.



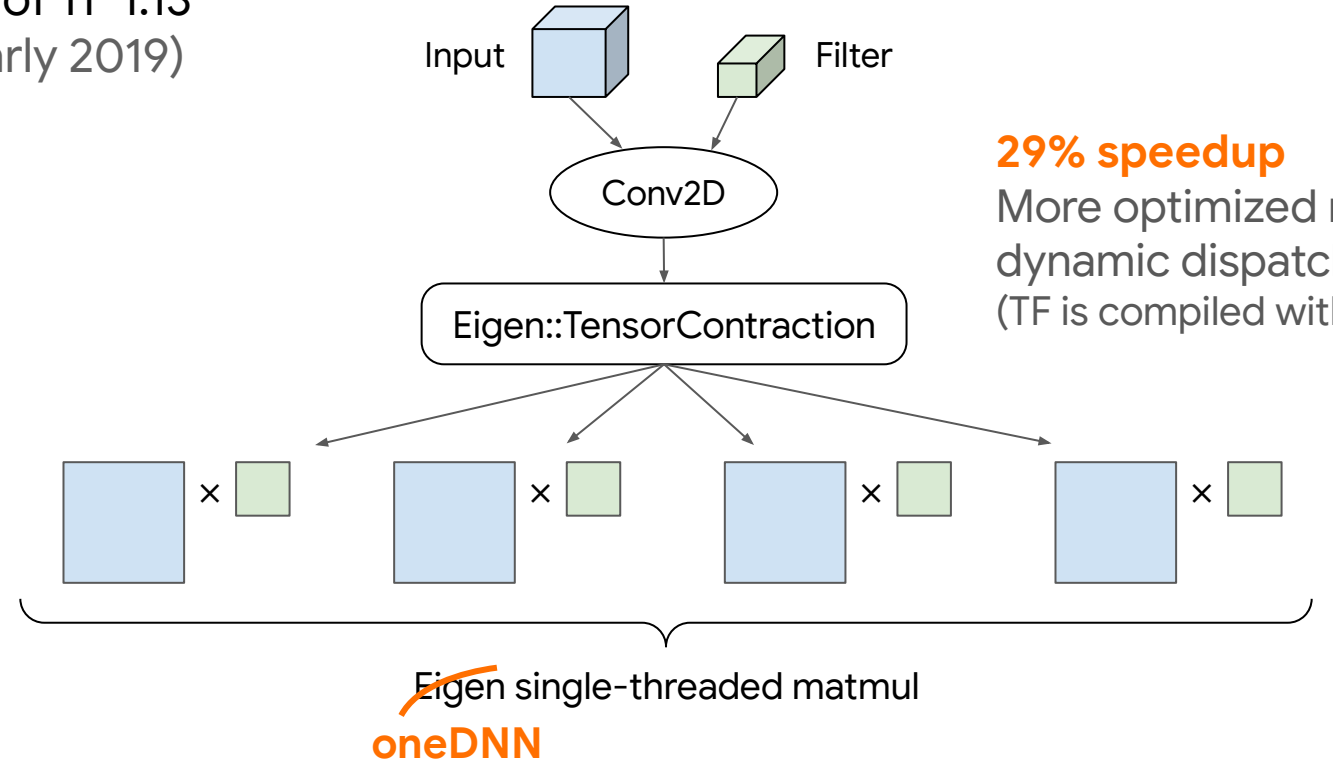
oneDNN in Vanilla TensorFlow

As of TF 1.13
(early 2019)



oneDNN in Vanilla TensorFlow

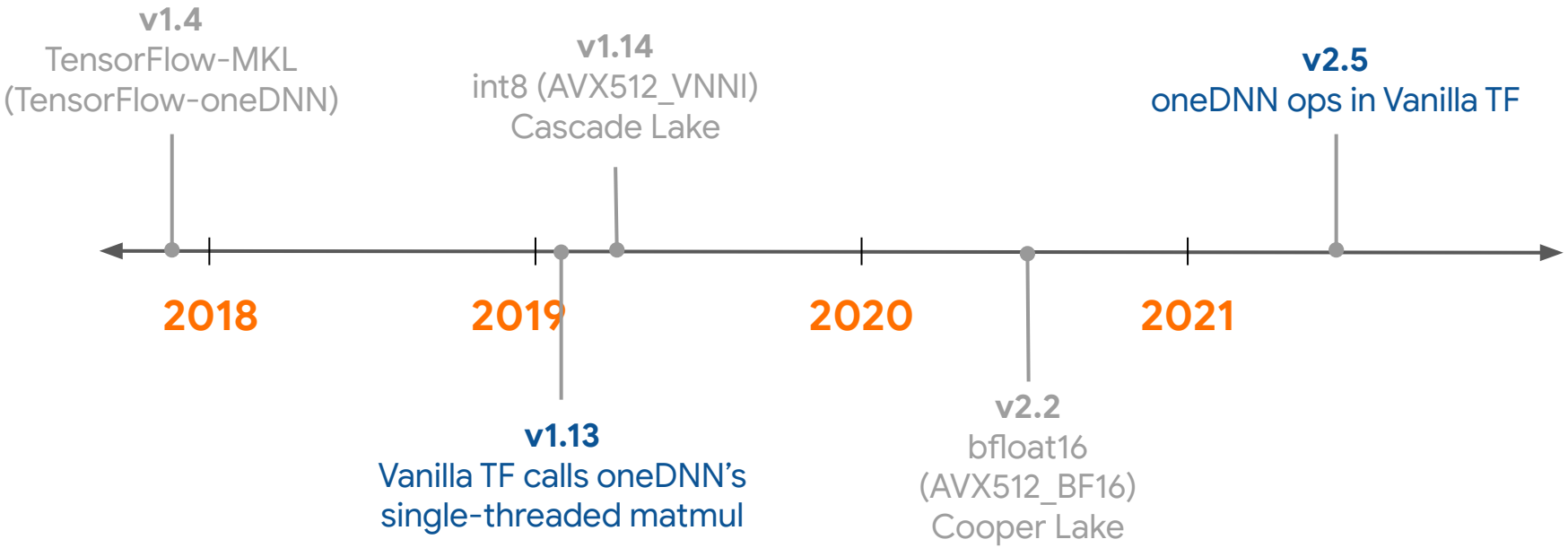
As of TF 1.13
(early 2019)



29% speedup
More optimized matmul, dynamic dispatch.
(TF is compiled with only AVX.)

Fast-forwarding to now...

oneDNN ops available in vanilla TF under a runtime flag



How to Get oneDNN Ops in Vanilla TF

- oneDNN v1.4 (April 2020) can take a custom thread pool.
  - oneDNN v2.x can also support TF native format (NHWC).
    - Convenient for Eager mode.
- Starting in TF 2.5 (May 2021), oneDNN ops are included in vanilla TF.
  - Disabled by default.
  - Enable by setting the environment variable TF_ENABLE_ONEDNN_OPTS=1.
- Throughput improvements up to
  - 3× in inference
  - 2.4× in training
- Supports bfloat16 mixed-precision computation.

TensorFlow Device Support

Pluggable Device

- Before TF 2.5, device integration requires changes to core TF.
  - Complex build dependencies and compiler toolchains.
  - Slow development time (needs PR reviews).
  - Combinatorial #build configurations to test for (multiplicative).
  - Easy to break.
- PluggableDevice
  - Scalable device integration.
  - Builds upon Modular TensorFlow.
  - Device support as plug-ins. (No device-specific code added to TF.)
  - Designed, developed, and driven by the TensorFlow community.
    - Largely by Intel.
    - Apple Metal plug-in available now.

PluggableDevice: Device Plugins for TensorFlow

PluggableDevice: Features

References:

- StreamExecutor C API RFC
- PluggableDevice RFC
- Graph optimization C API RFC
- Kernel C API Extension for Variable Ops RFC
- Pluggable Profiler RFC
- Tutorial under development

More background:

- Modular TensorFlow RFC
- Kernel and op registration C API RFC

Device plug-in

Functions for
- PluggableDevice creation
- Stream management
- Memory management
- Timer

Custom Ops

Custom Kernels

Custom Graph Optimization Pass

StreamExecutor C API

Kernel and Op Registration C API

Graph Optimization C API

TensorFlow

PluggableDevice factory

Op Registry

Kernel Registry

Custom Graph Optimizer Registry

PluggableDevice: Demo

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
```

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

PluggableDevice: Demo

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
```

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

PluggableDevice: Demo

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:APU:0', device_type='APU')]  # TensorFlow can see APU
```

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

PluggableDevice: Demo

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:APU:0', device_type='APU')]  # TensorFlow can see APU

>>> a = tf.random.normal(shape=[5], dtype=tf.float32)  # Runs on CPU
>>> b = tf.nn.relu(a)  # Runs on APU because PluggableDevice has higher device priority.
```

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

PluggableDevice: Demo

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:APU:0', device_type='APU')]  # TensorFlow can see APU

>>> a = tf.random.normal(shape=[5], dtype=tf.float32)  # Runs on CPU
>>> b = tf.nn.relu(a)  # Runs on APU because PluggableDevice has higher device priority.

>>> with tf.device("/APU:0"):  # Users can also use 'with tf.device' syntax
...  c = tf.nn.relu(a)  # Runs on APU
```

PluggableDevice: Demo

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:APU:0', device_type='APU')]  # TensorFlow can see APU

>>> a = tf.random.normal(shape=[5], dtype=tf.float32)  # Runs on CPU
>>> b = tf.nn.relu(a)  # Runs on APU because PluggableDevice has higher device priority.

>>> with tf.device("/APU:0"):  # Users can also use 'with tf.device' syntax
...  c = tf.nn.relu(a)  # Runs on APU

>>> with tf.device("/CPU:0"):  # Users can put ReLU on other devices like normal.
...  d = tf.nn.relu(a)  # Runs on CPU
```

PluggableDevice: Demo

APU: Awesome Processing Unit
APU plug-in has one op: ReLU

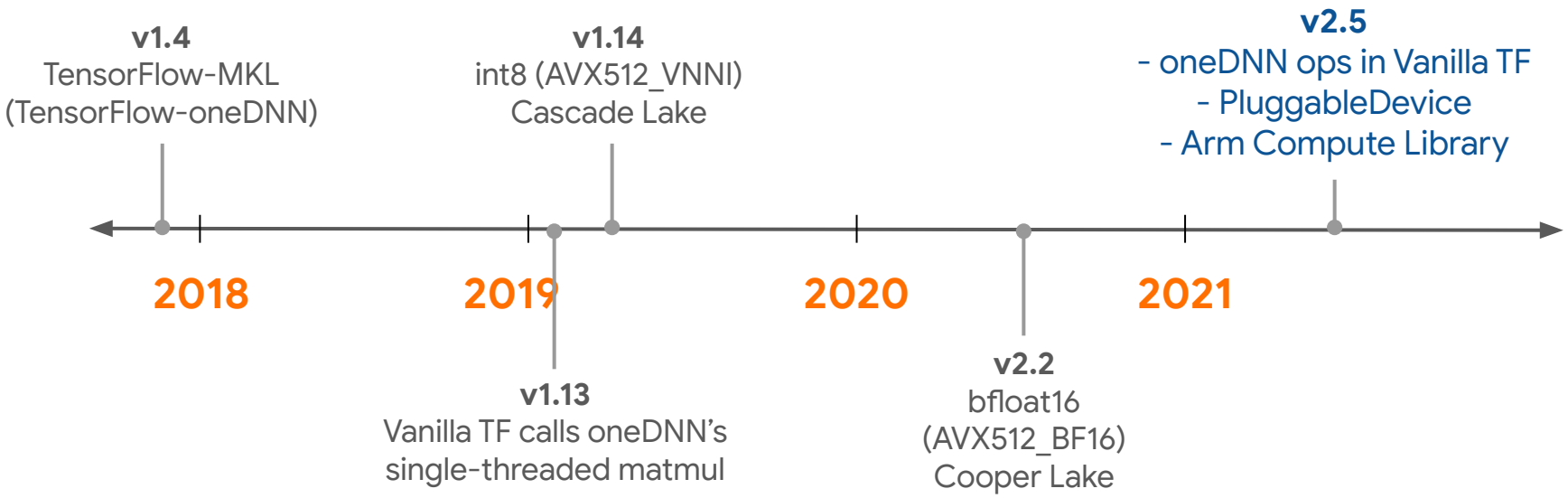
```
$ pip install tensorflow-apu-0.0.1-cp36-cp36m-linux_x86_64.whl
Successfully installed tensorflow-apu-0.0.1
$ python
>>> import tensorflow as tf  # TensorFlow registers PluggableDevices here
>>> tf.config.list_physical_devices()
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:APU:0', device_type='APU')]  # TensorFlow can see APU

>>> a = tf.random.normal(shape=[5], dtype=tf.float32)  # Runs on CPU
>>> b = tf.nn.relu(a)  # Runs on APU because PluggableDevice has higher device priority.

>>> with tf.device("/APU:0"):  # Users can also use 'with tf.device' syntax
...  c = tf.nn.relu(a)  # Runs on APU

>>> with tf.device("/CPU:0"):  # Users can put ReLU on other devices like normal.
...  d = tf.nn.relu(a)  # Runs on CPU

>>> @tf.function  # Defining a tf.function
... def run():
...    e = tf.random.uniform(shape=[100], dtype=tf.float32)  # Runs on CPU
...    f = tf.nn.relu(e)  # Runs on APU
>>> run()  # PluggableDevices also work with tf.function and graph mode.
```

Conclusions

Plenty of exciting work.

More to come: AMX / Sapphire Rapids support, Intel XPU, TPP in MLIR!

Follow us on: TensorFlow GitHub, TensorFlow Forum, MLIR Forum



Thank you!

Questions?

TensorFlow