Tensor Processing Unit

Petr Dobiáš

petr.dobias@ensea.fr

Academic Year 2023/2024

References

and sources for this course

- [3] N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Communucitaions of the ACM, 63 (2020), p. 67-78, https://doi.org/10.1145/3360307 + presentation on October 27, 2020
- [1] Fidle (Formation d'Introduction au Deep Learning), https://fidle.cnrs.fr
- [2] Google Cloud TPU System Architecture Documentation, https://cloud.google.com/tpu/docs/system-architecture-tpu-vm#targetText=Tensor%20Processing%20Units%20(TPUs)%20are,and%20leadership%20in%20machine%20learning.

How to Speed Up Execution of Applications?

- In general

$$\text{time} = \frac{\text{distance}}{\text{speed}}$$

- Two possibilities

Software optimisation

- We reduce the distance

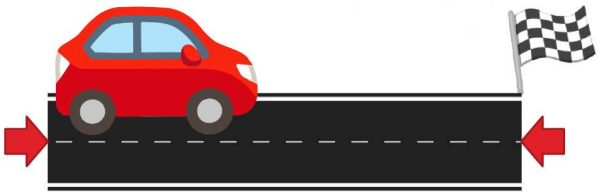


Figure from Slide 14, Lecture 16, Fidle, `https://cloud.univ-grenoble-alpes.fr/index.php/s/wxCztjYBbQ6zwd6?dir=undefined&path=%2FSaison%202022-2023&openfile=638543327`

Hardware optimisation

- We increase the speed

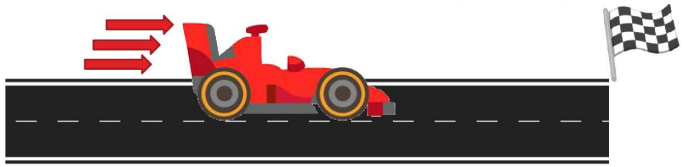
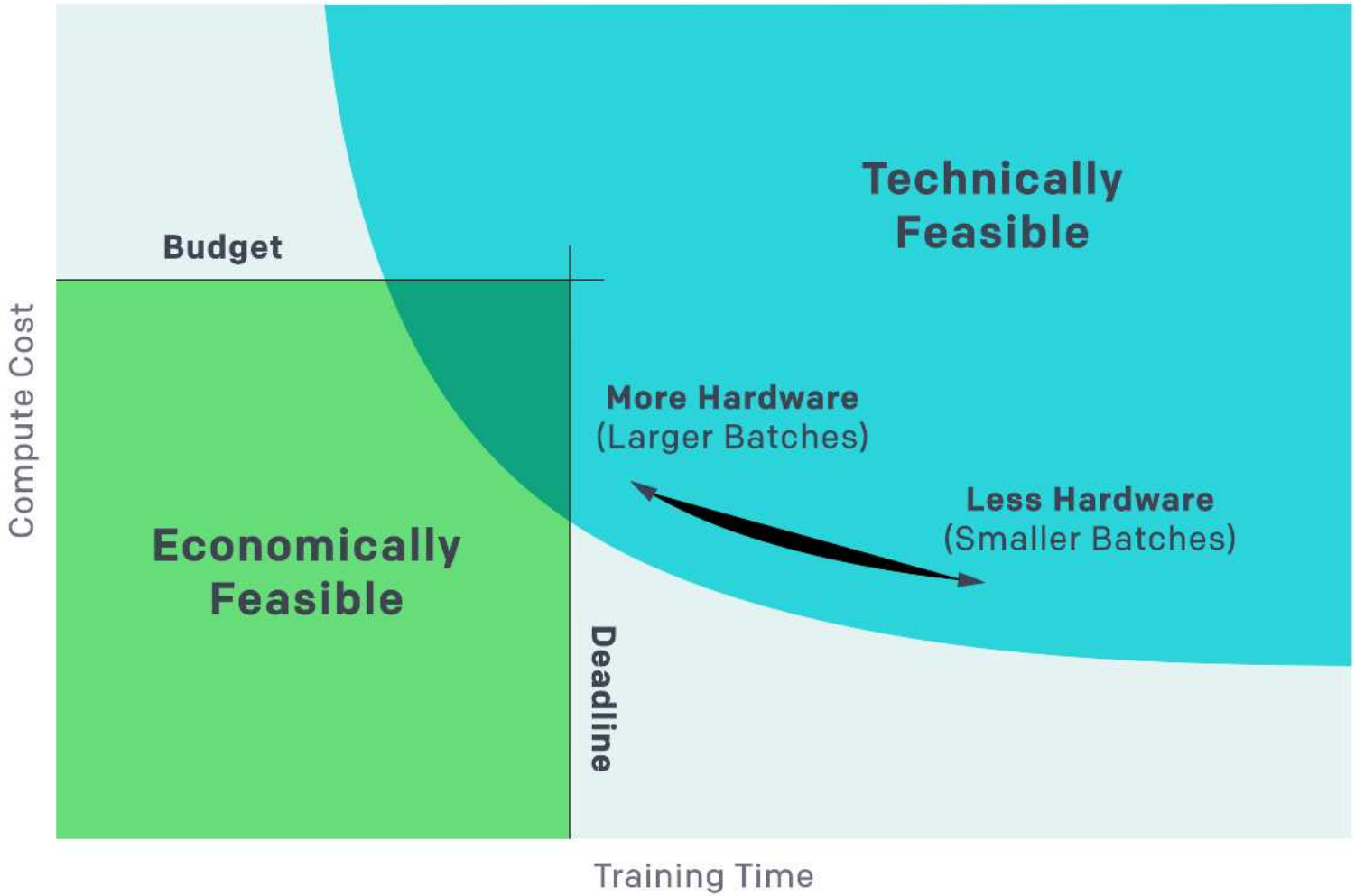


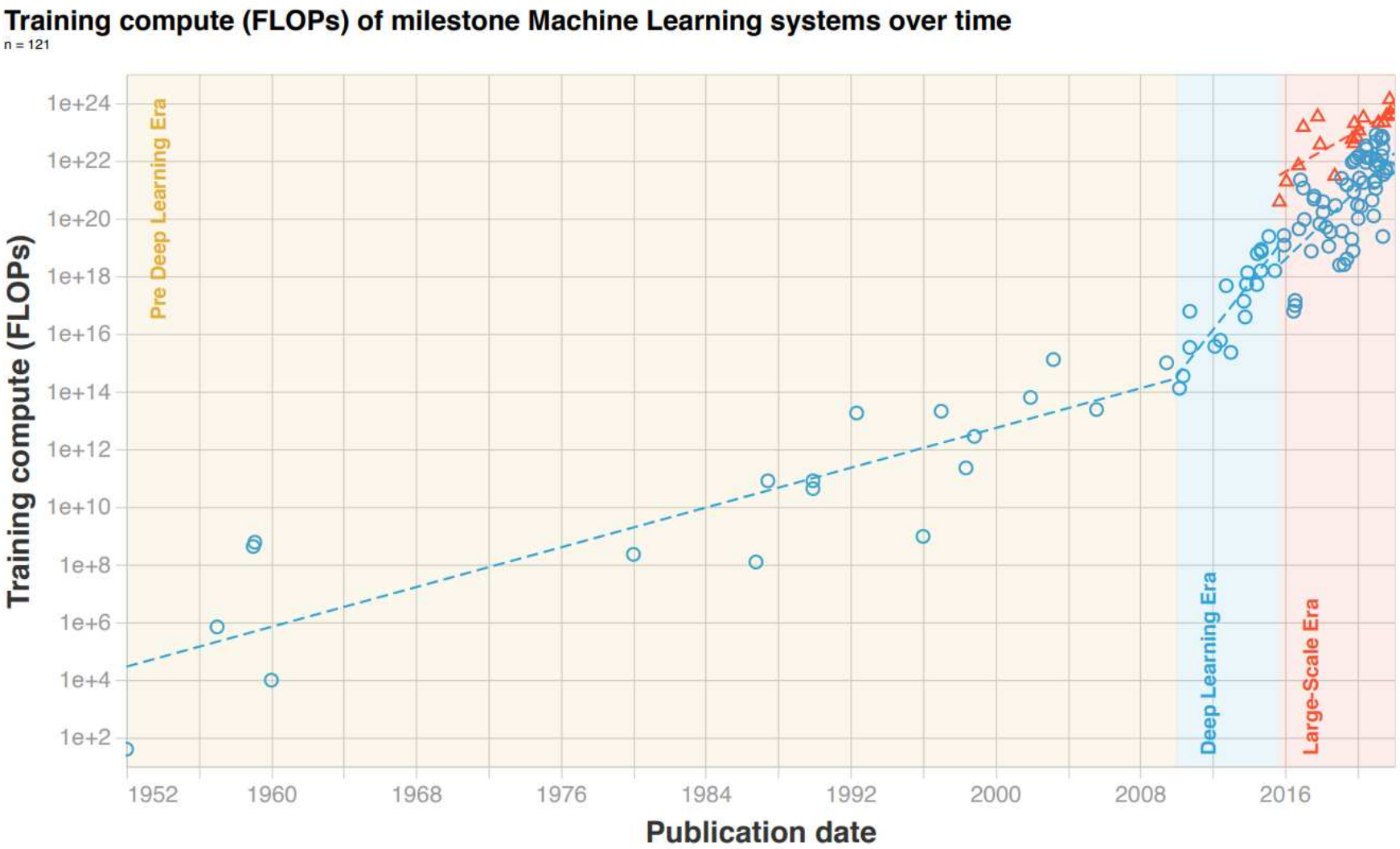
Figure from Slide 14, Lecture 16, Fidle, `https://cloud.univ-grenoble-alpes.fr/index.php/s/wxCztjYBbQ6zwd6?dir=undefined&path=%2FSaison%202022-2023&openfile=638543327`

Trade-off: Time, Compute Cost and Feasibility



https://openai.com/research/how-ai-training-scales

Three Eras



J. Sevilla et al., *Compute Trends Across Three Eras of Machine Learning*, arXiv 2022, `https://arxiv.org/abs/2202.05924`

Tensors (1/3)

- *Tensors*: multidimensional arrays of any dimension
- *Scalar values*: 0-dimensional tensors
- *Vectors*: one-dimensional tensors
- *Matrices*: two-dimensional tensors

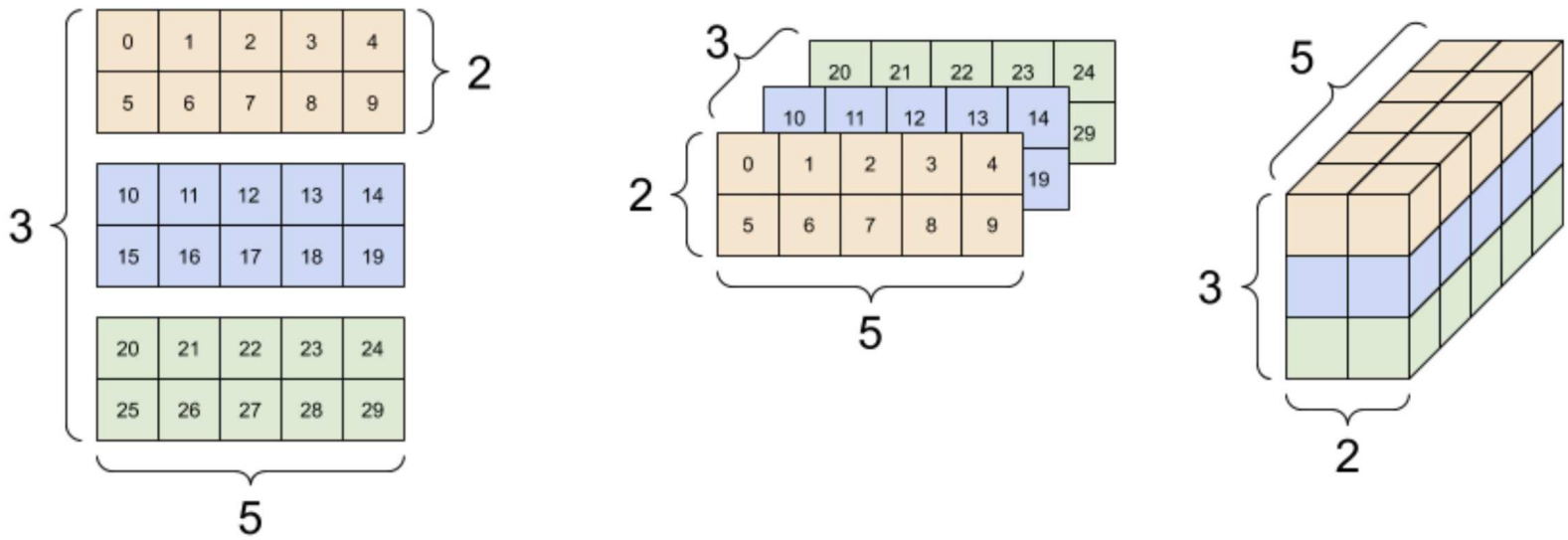
A scalar, shape: [ ]	A vector, shape: [3]	A matrix, shape: [3, 2]
4	2.0 3.0 4.0 3	3 × 2: 1 2 / 3 4 / 5 6

https://www.tensorflow.org/guide/tensor

Tensors (2/3)

- Tensors with more than two axes can be viualised in many ways, e.g.

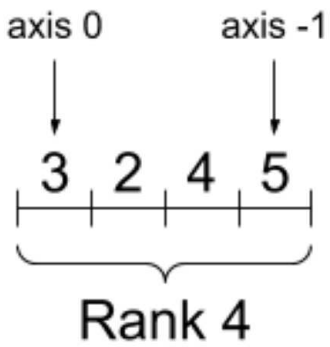
A 3-axis tensor, shape: `[3, 2, 5]`

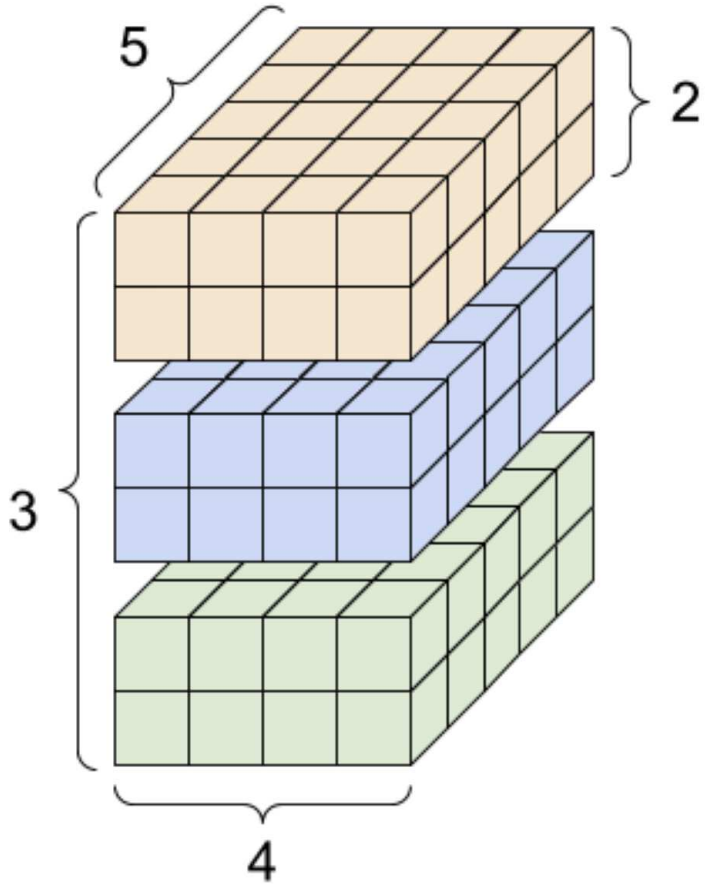


https://www.tensorflow.org/guide/tensor

Tensors (3/3)

A rank-4 tensor, shape: `[3, 2, 4, 5]`





https://www.tensorflow.org/guide/tensor

Layout

TensorCore [2]

- ***One or more matrix-multiply units (MXUs)***
  - It is composed of 128*x*128 multiply-accumulators in a systolic array
  - MXUs provide the bulk of the compute power in a TensorCore: each MXU is capable of performing 16K multiply-accumulate operations per cycle
- ***Vector unit***
  - It is used for general computation such as activations and softmax
- ***Scalar unit***
  - It is used for control flow, calculating memory addresses, and other maintenance operations

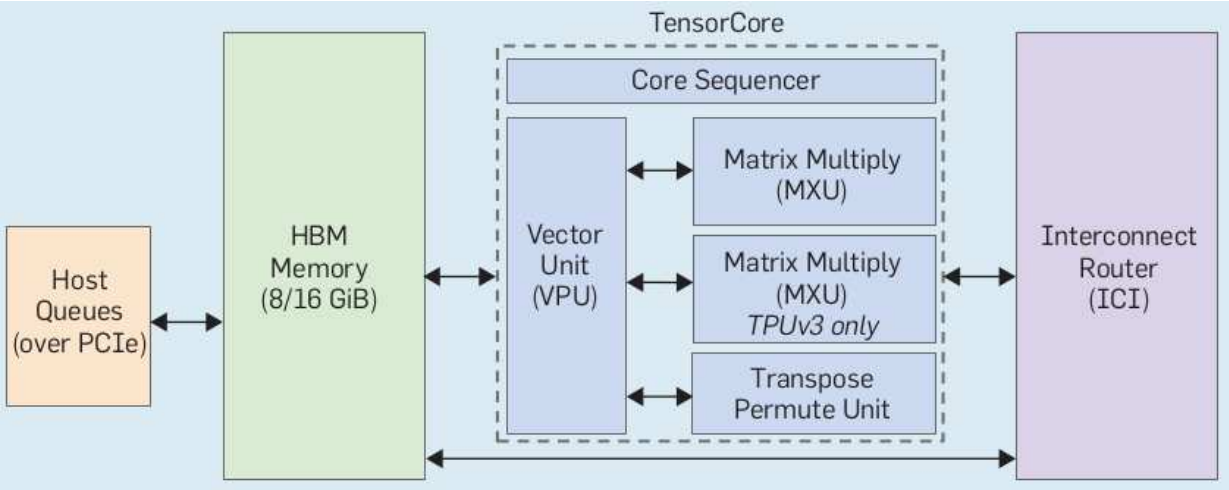


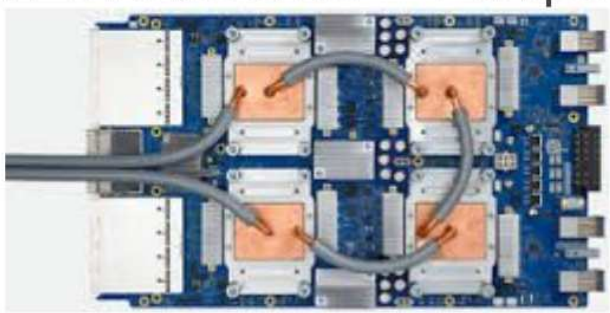
Figure 2, N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

TPU Chip [2]

- It contains one or more TensorCores to run matrix multiplication
- The number of TensorCores depend on the version of the TPU chip
  - v2, v3, and v5e $\rightarrow$ one TensorCore per chip
  - v4 $\rightarrow$ 2 TensorCores per chip

TPUv2 boards = 4 chips

TPUv3 boards = 4 chips

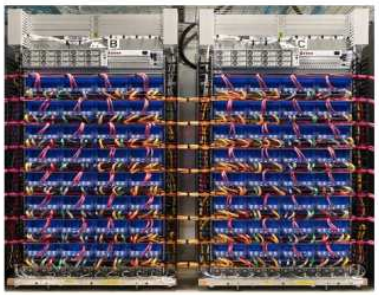


Figures from Slide 54 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

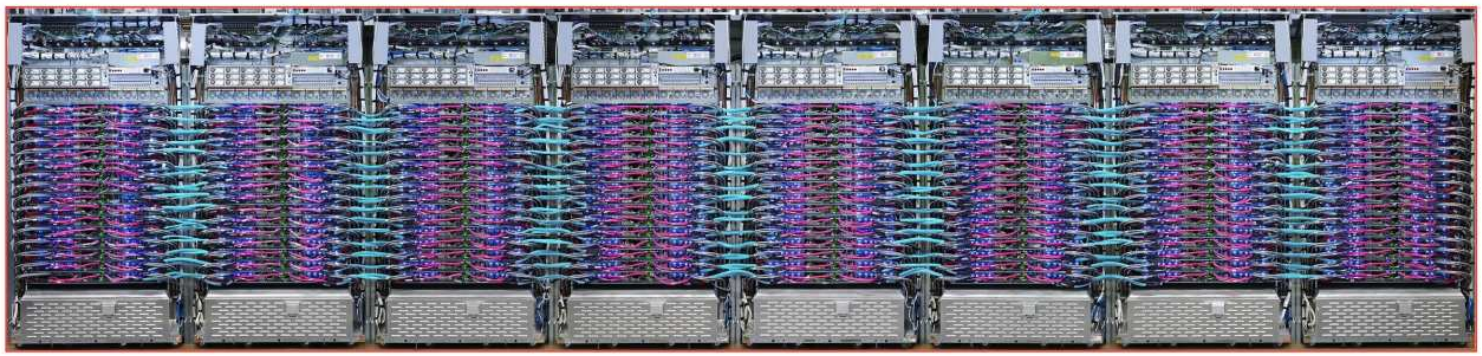
TPU Pod [2]

- It is a contiguous set of TPUs grouped together over a specialized network
- The number of TPU chips in a TPU Pod is dependent on the TPU version

TPUv2 supercomputer (256 chips)



TPUv3 supercomputer (1024 chips)



Figures from Slide 54 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

Layout

TPUv1

- Late 2013
- Example of domain-specific architecture (DSA)
- Single chip system
- It only accelerates inference $\rightarrow$ training capability was the limiting factor
- It performed better than contemporary alternatives

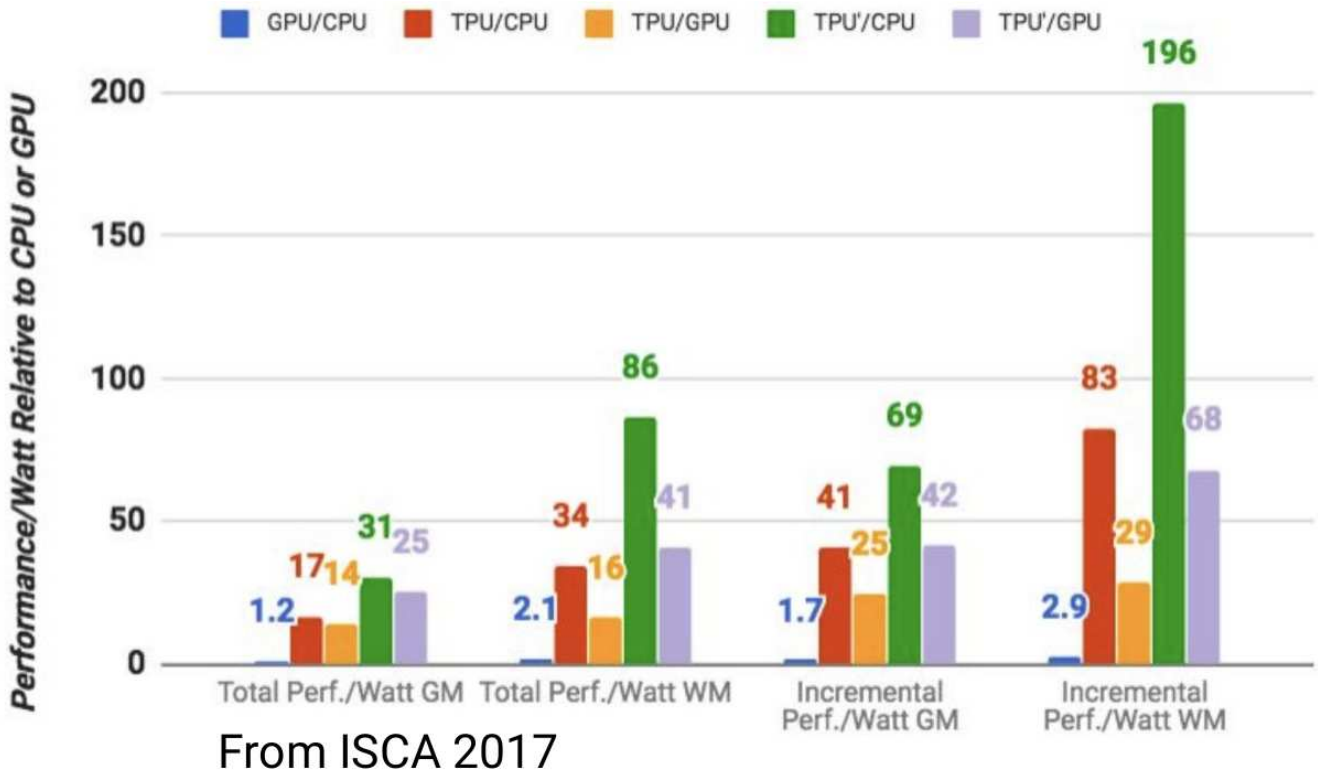


Figure from Slide 4 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

Traning vs. Inference

	Training	**Inference**
Operations per solution	10^{20}	10^{8}
Solution latency	hours or days	several *ms*
Location	Key ML hubs	Several locations worldwide
Data size	petabytes	Modest real-time user input

Question

Do we need different architectures for training and inference?

- Yes
- No

Question

Do we need different architectures for training and inference?

- Yes
- ***No***

- Training and inference share some computational elements, such as matrix multiplications, convolutions, and activation functions $\rightarrow$ they might have similar functional units
- But they have different key architecture aspects: parallisation, computation, memory, programmability, data type

TPUv2

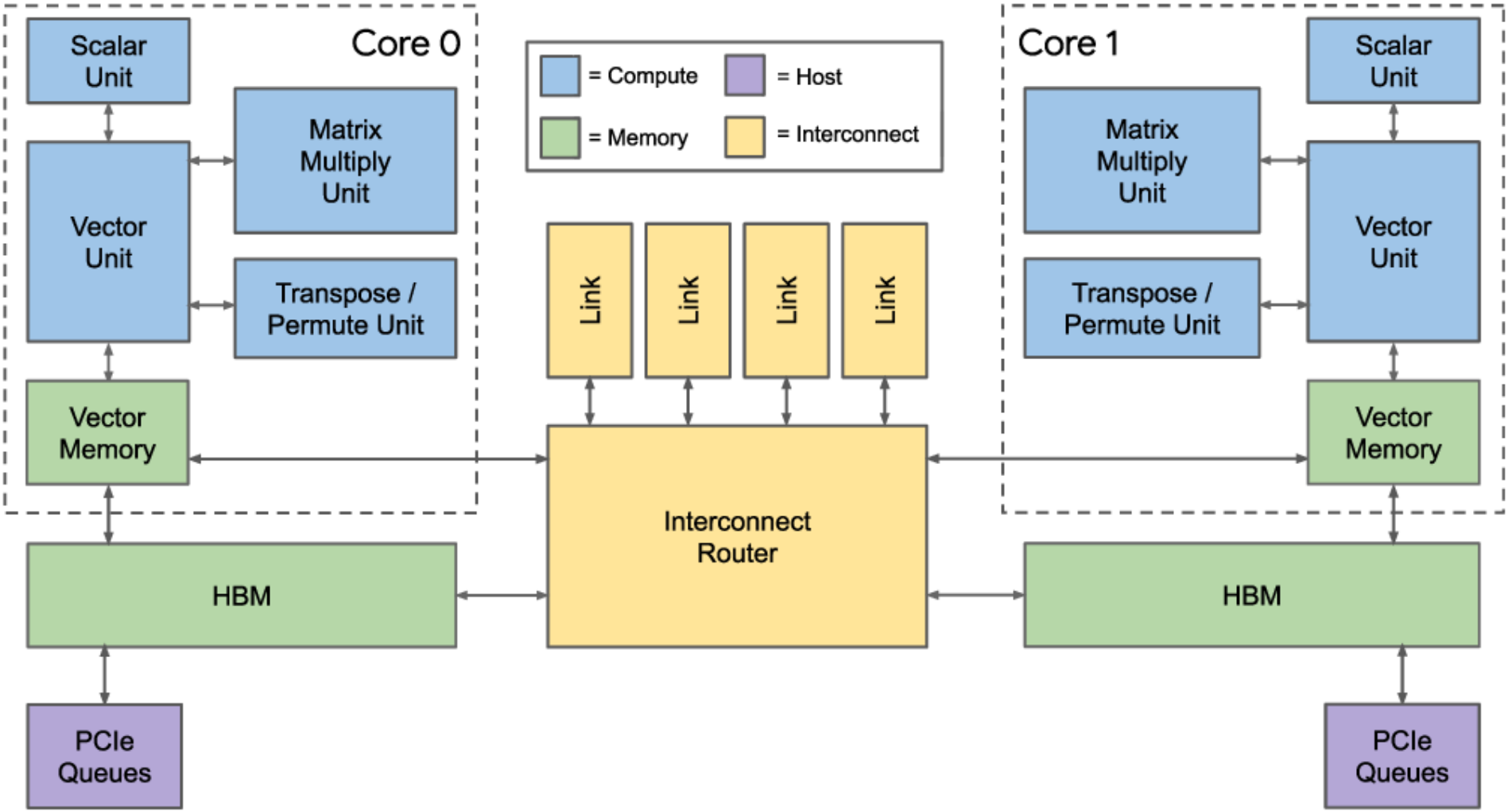


Figure 2, T. Norrie et al., *The Design Process for Google?s Training Chips: TPUv2 and TPUv3*, IEEE Micro, 41 (2021), pp. 56?63.

TPUv2

- VLIW (Very Long Instruction Word) architecture
  - 322-bit VLIW instruction can launch 8 operations: 2 scalar, 2 vector arithmetic logic unit (ALU), vector load and store, and a pair of slots that queue data to and from the matrix multiply and transpose units
- Linear algebra ISA (Instruction Set Architecture)
  - Scalar, vector and matrix
  - Scalar operations using a 4K 32-bit scalar data memory (Smem) and 32 32-bit scalar registers (Sregs)

TPUv2

(a) TPUv2 core scalar unit

(b) Single vector lane. The vector unit contains 128 vector lanes.

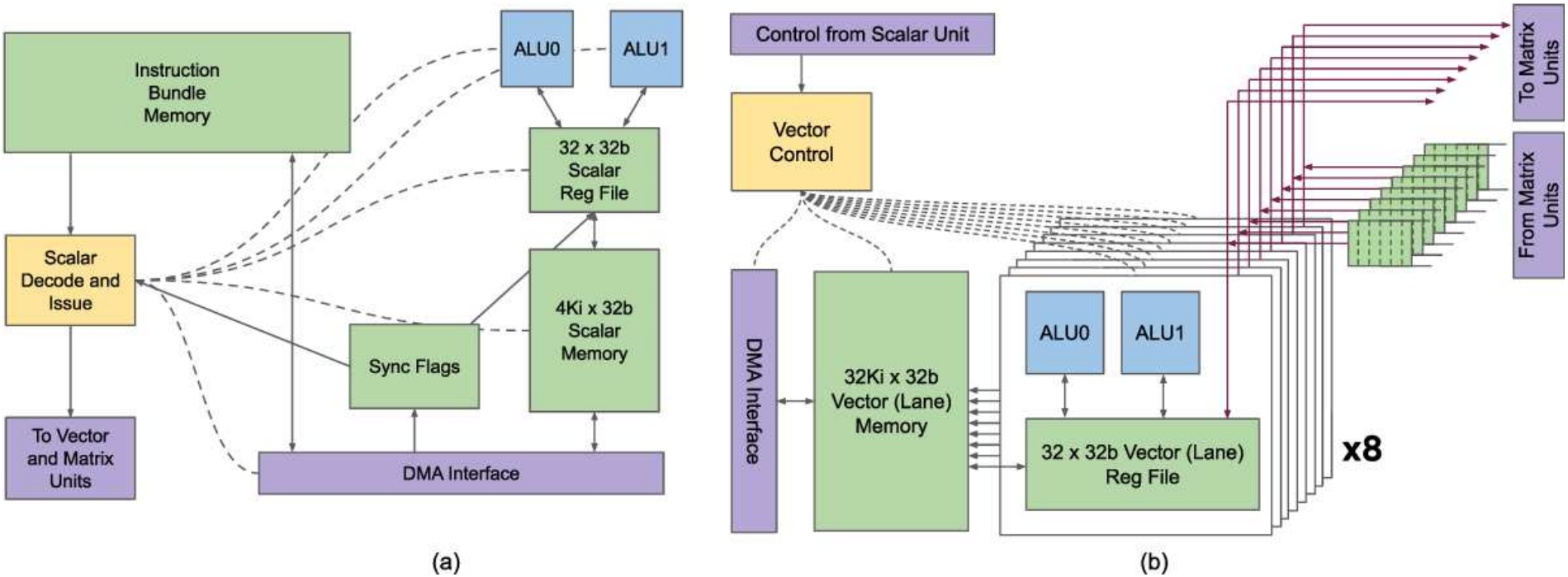


Figure 3, T. Norrie et al., *The Design Process for Google?s Training Chips: TPUv2 and TPUv3*, IEEE Micro, 41 (2021), pp. 56?63.

TPUv2 Floor Plan

- Two TensorCores

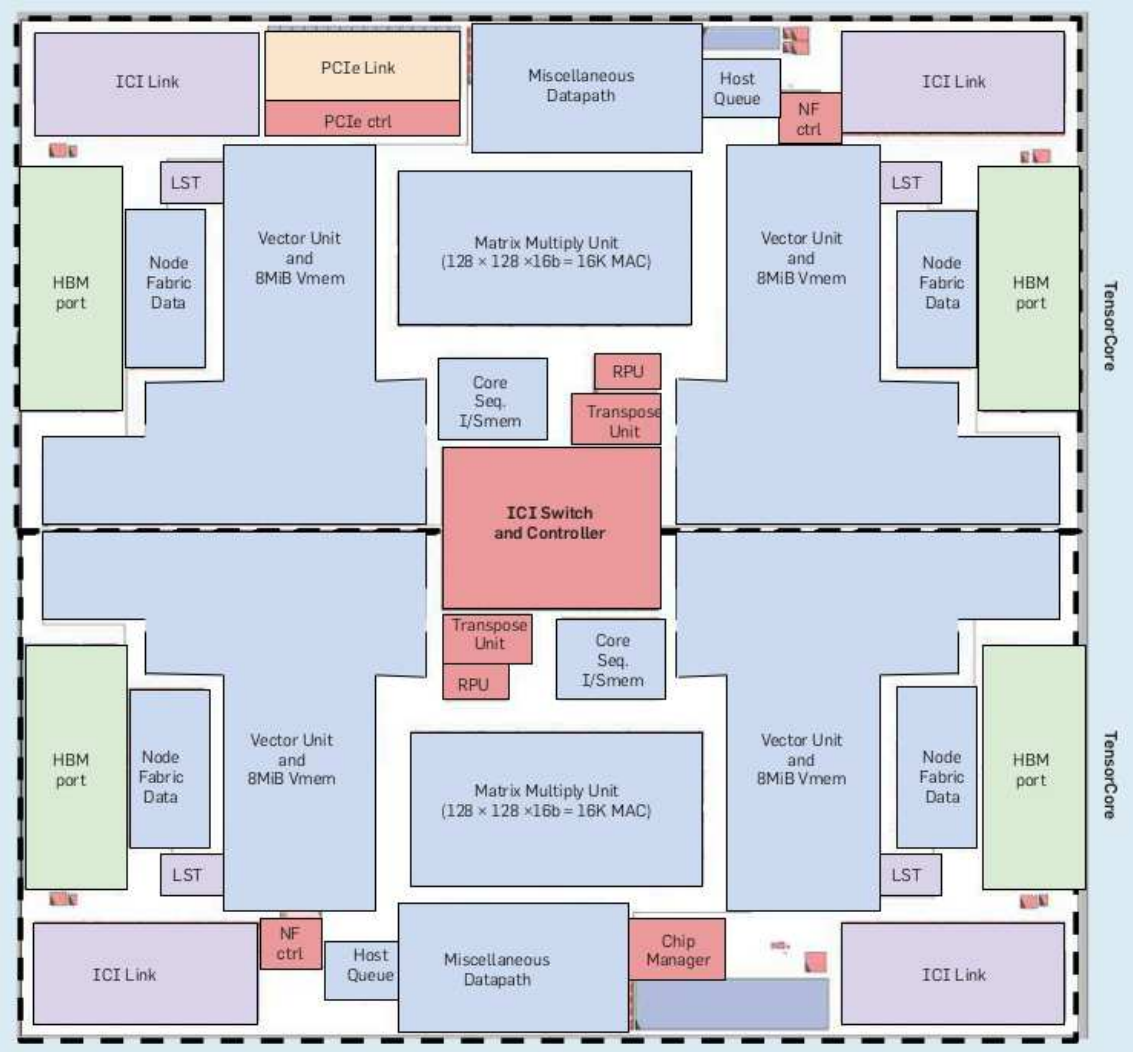


Figure 3, N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

TPUv3

- Based on TPUv2 and following improvements

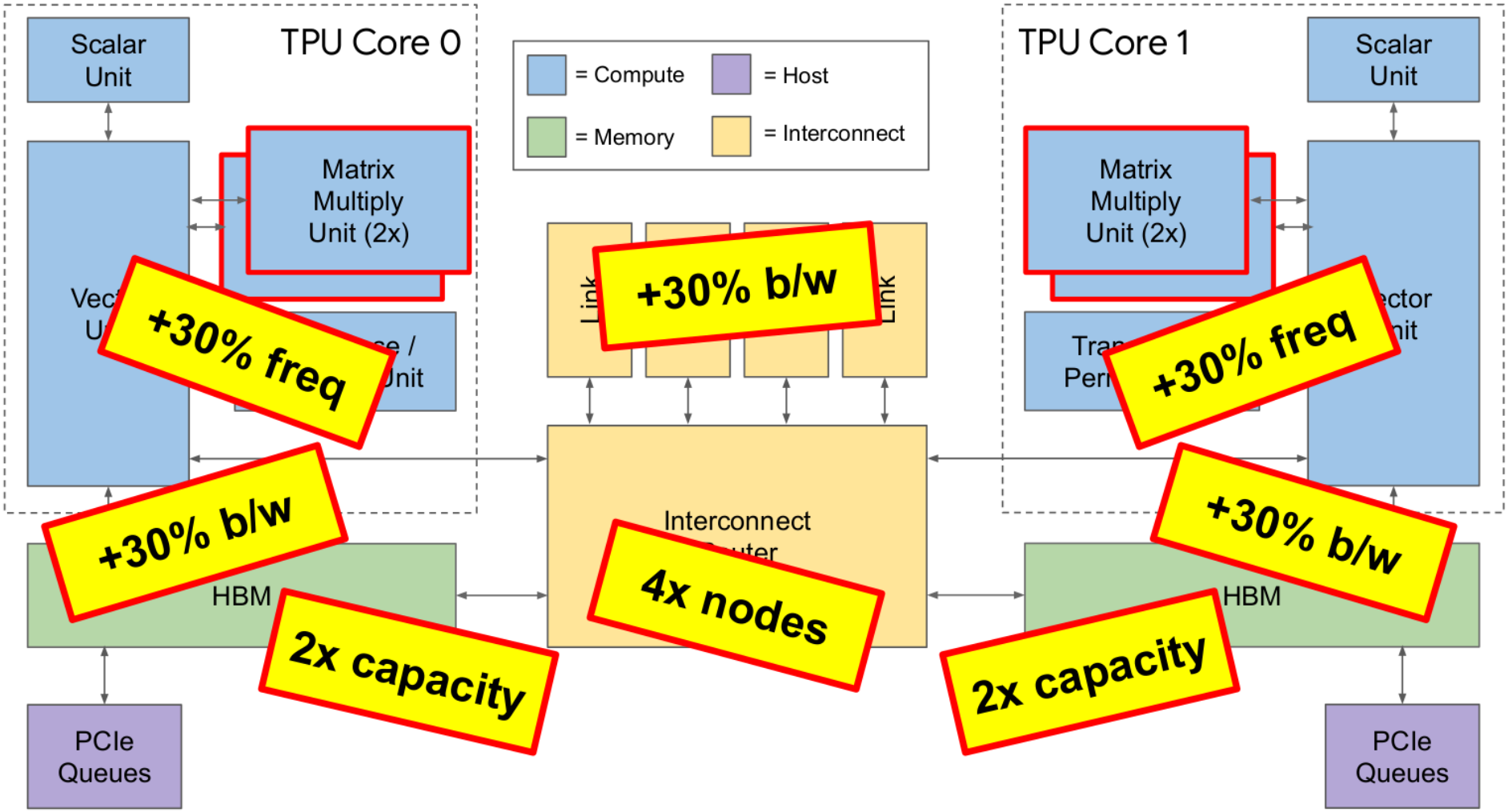
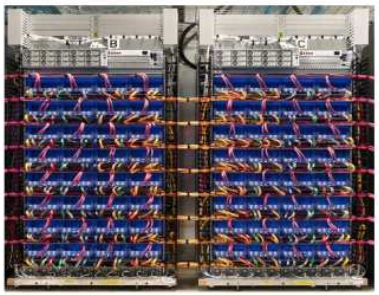


Figure from Slide 49 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

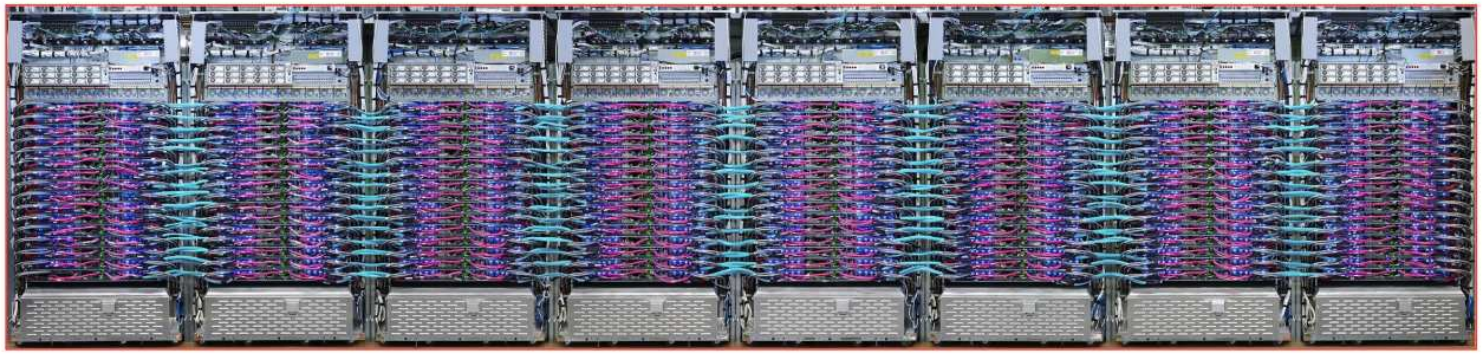
TPUv2 vs. TPUv3

- Supercomputers with Shared-memory Interconnect

TPUv2 supercomputer (256 chips)

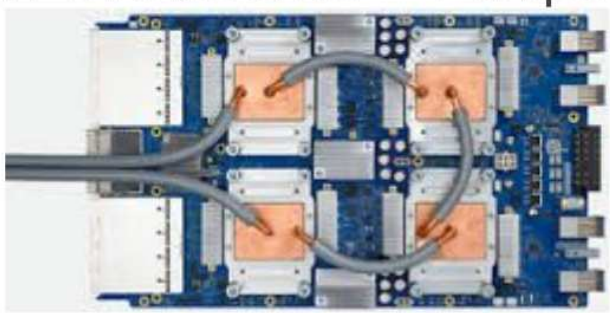


TPUv3 supercomputer (1024 chips)



TPUv2 boards = 4 chips

TPUv3 boards = 4 chips

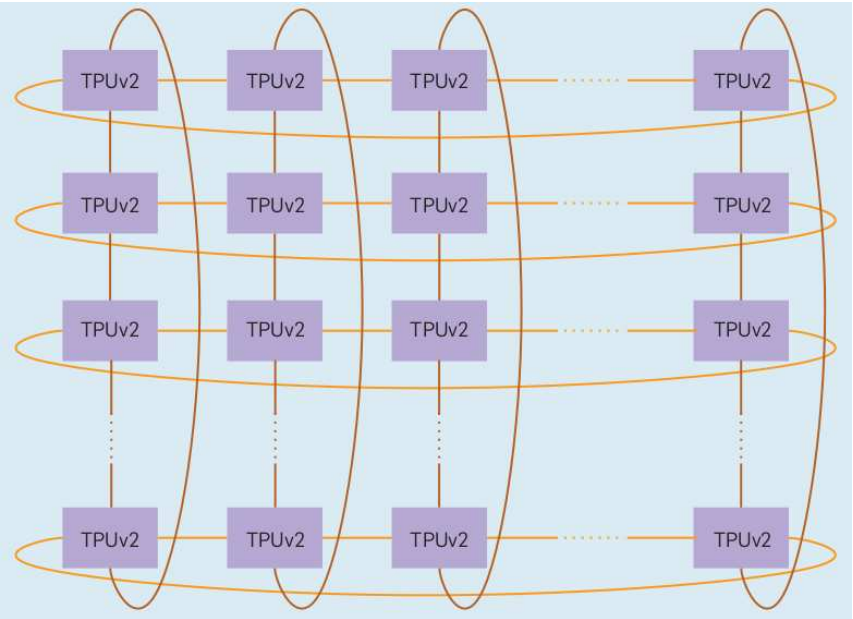


Figures from Slide 54 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

TPUv2 vs. TPUv3

TPUv2 supercomputer

- 256 chips
- 11.5 petaflops
- 4 TB High Bandwidth Memory (HBM)
- Air cooled
- 16*x*16 2-D torus



A 2D-torus topology. Figure 1, N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

TPUv3 supercomputer

- 1024 chips
- More than 100 petaflops
- 32 TB High Bandwidth Memory (HBM)
- Liquid cooled
- New chip + large-scale system

TPUv1 vs. TPUv2 vs. TPUv3

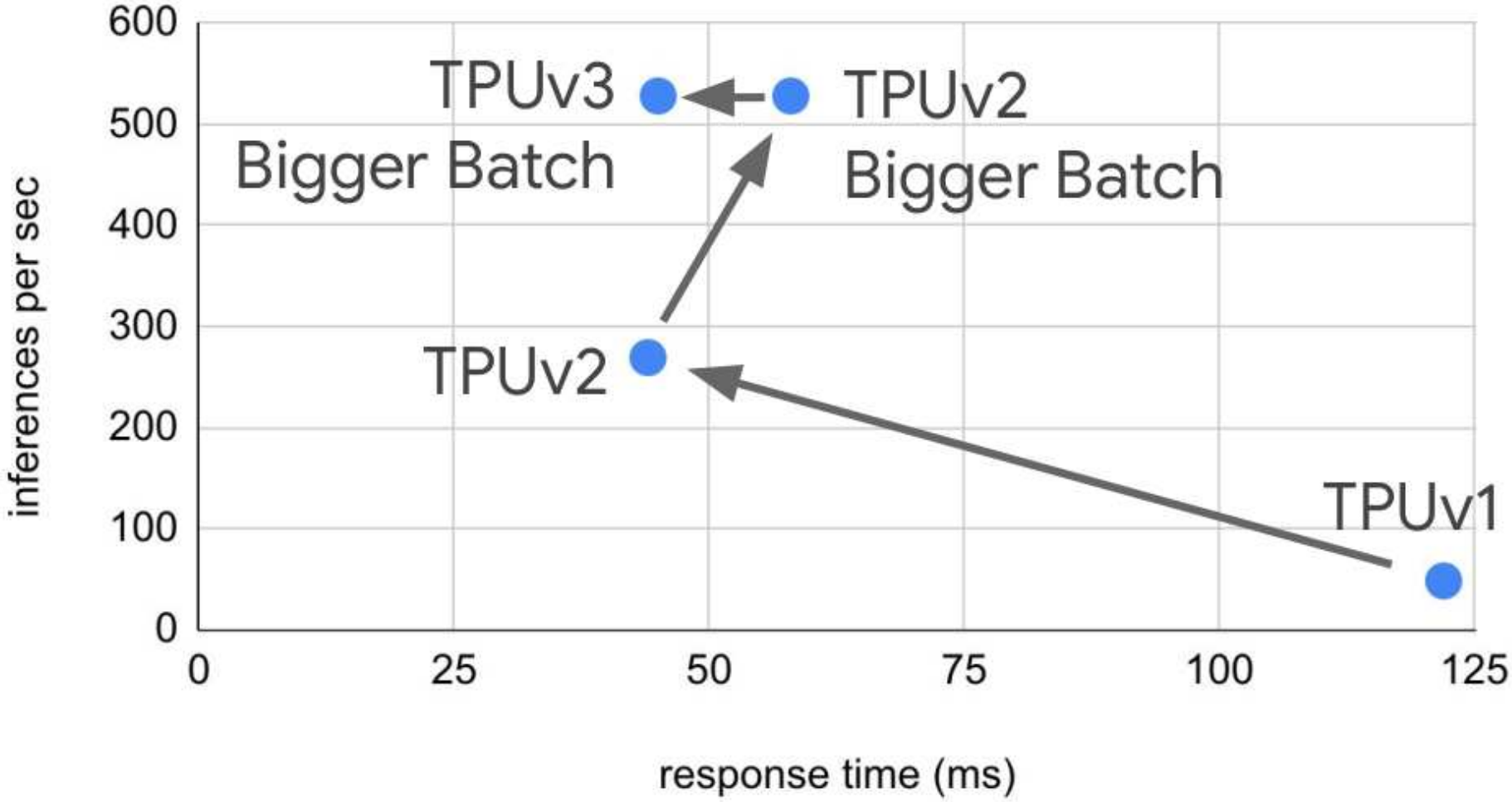
We cannot reveal technology details of our chip partner. Although it is in a larger, older technology, the TPUv2 die size is less than 3/4s of the GPU. TPUv3 is 6% larger in that same technology. TDP stands for Thermal Design Power. The Volta has 80 symmetric multiprocessors.

Feature	TPUv1	TPUv2	TPUv3	Volta
Peak TeraFLOPS/ Chip	92 (8b int)	46 (16b) 3 (32b)	123 (16b) 4 (32b)	125 (16b) 16 (32b)
Network links x Gbits/s/Chip	–	4 x 496	4 x 656	6 x 200
Max chips/supercomputer	–	256	1024	Varies
Peak PetaFLOPS/supercomputer	–	11.8	126	Varies
Bisection Terabits/supercomputer	–	15.9	42.0	Varies
Clock Rate (MHz)	700	700	940	1530
TDP (Watts)/Chip	75	280	450	450
TDP (Kwatts)/supercomputer	–	124	594	Varies
Die Size (mm^2)	<331	<611	<648	815
Chip Technology	28nm	>12nm	>12nm	12nm
Memory size (on-/off-chip)	28MiB/8GiB	32MiB/16GiB	32MiB/32GiB	36MiB/32GiB
Memory GB/s/Chip	34	700	900	900
MXUs/Core, MXU Size	1 256x256	1 128x128	2 128x128	8 4x4
Cores/Chip	1	2	2	80
Chips/CPU Host	4	4	8	8 or 16

Table 3, N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

TPUv1 vs. TPUv2 vs. TPUv3

LSTM0 Inferences per second and response time

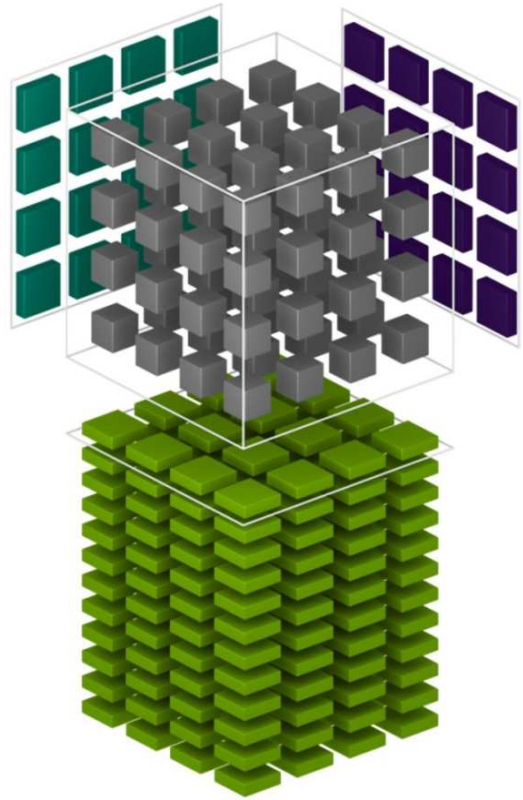


Figures from Slide 61 in presentation of N. P. Jouppi et al., *A domain-specific supercomputer for training deep neural networks*, Commun. ACM, 63 (2020), p. 67-78.

Layout

Tensor Cores

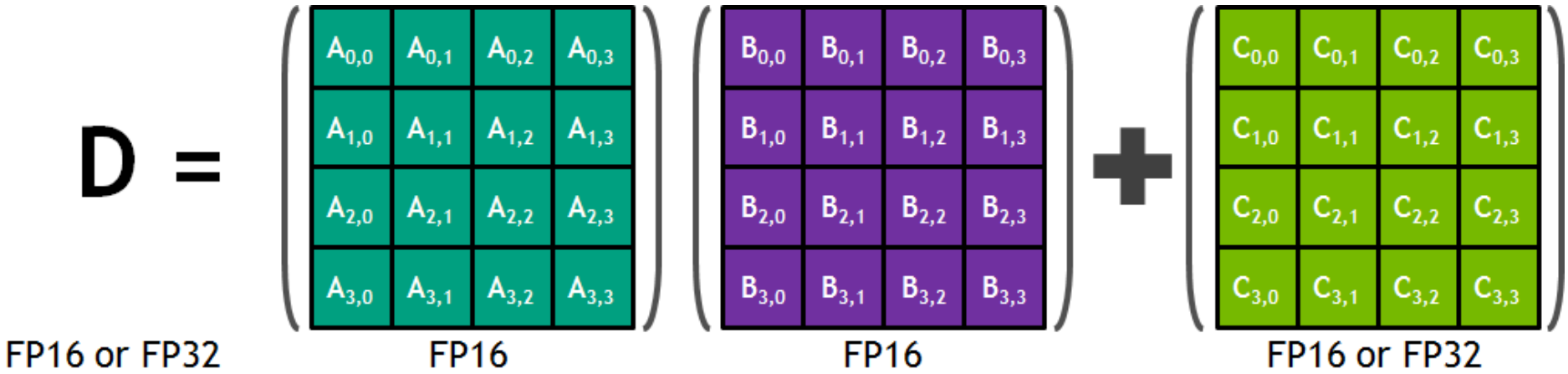
- https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/
- Provide a huge boost to convolutions and matrix operations
- Programmable using NVIDIA libraries and directly in CUDA C++ code
- Use of mixed-precision to achieve higher throughput without sacrificing accuracy
- Used in the new Volta GPU Architecture (Tesla V100 accelerator)
  - Peak throughput is 12 times higher than the 32-bit floating point throughput of the previous-generation Tesla P100
- Already supported for Deep Learning training in many Deep Learning frameworks (including Tensorflow, PyTorch, MXNet, and Caffe2)



https://developer-blogs.nvidia.com/wp-content/uploads/2017/12/tensor_cube_white-624x934.png

What are Tensor Cores?

- Tesla V100's Tensor Cores are programmable matrix-multiply-and-accumulate units that can deliver up to 125 Tensor TFLOPS for training and inference applications
- Tesla V100 GPU contains 640 Tensor Cores
- Each Tensor Core performs 64 floating point FMA mixed-precision operations per clock
- Each Tensor Core provides a $4x4x4$ matrix processing array to perform the operation $D = A * B + C$

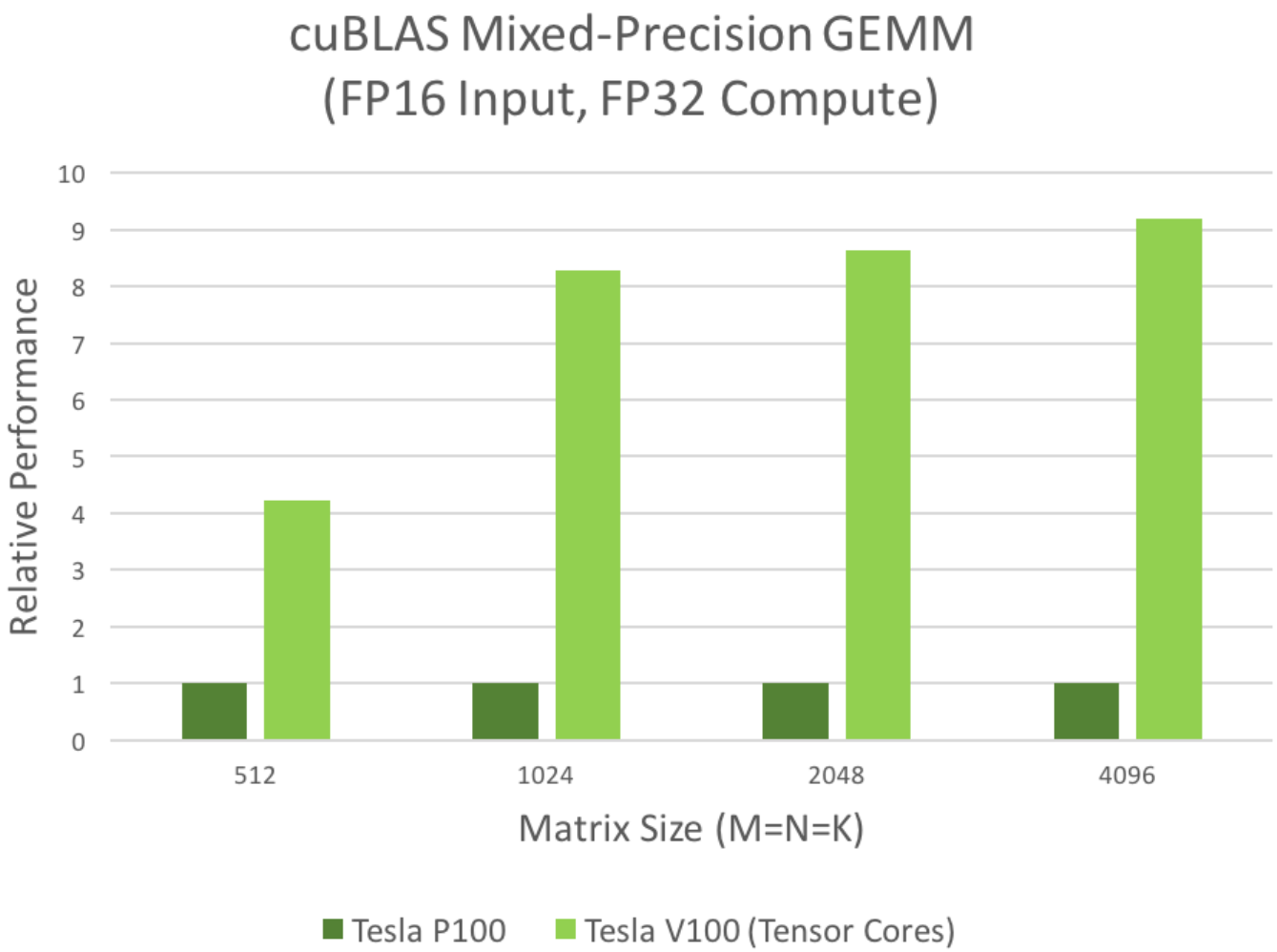


https://developer.nvidia.com/blog/parallelforall/wp-content/uploads/2017/05/image4.png

Tensor Cores in CUDA Libraries

- Two CUDA libraries that use Tensor Cores are cuBLAS and cuDNN
- cuBLAS uses Tensor Cores to speed up GEMM computations (GEMM is the BLAS term for a matrix-matrix multiplication)
  - Used for exemple in signal processing, fluid dynamics
  - As the data sizes of these applications grow exponentially, these applications require matching increases in processing speed
- cuDNN uses Tensor Cores to speed up both convolutions and recurrent neural networks (RNNs)
  - There are more and more convolutional layers $\rightarrow$ deeper networks
  - Need to process them quickly

Tensor Cores in cuBLAS (1/2)

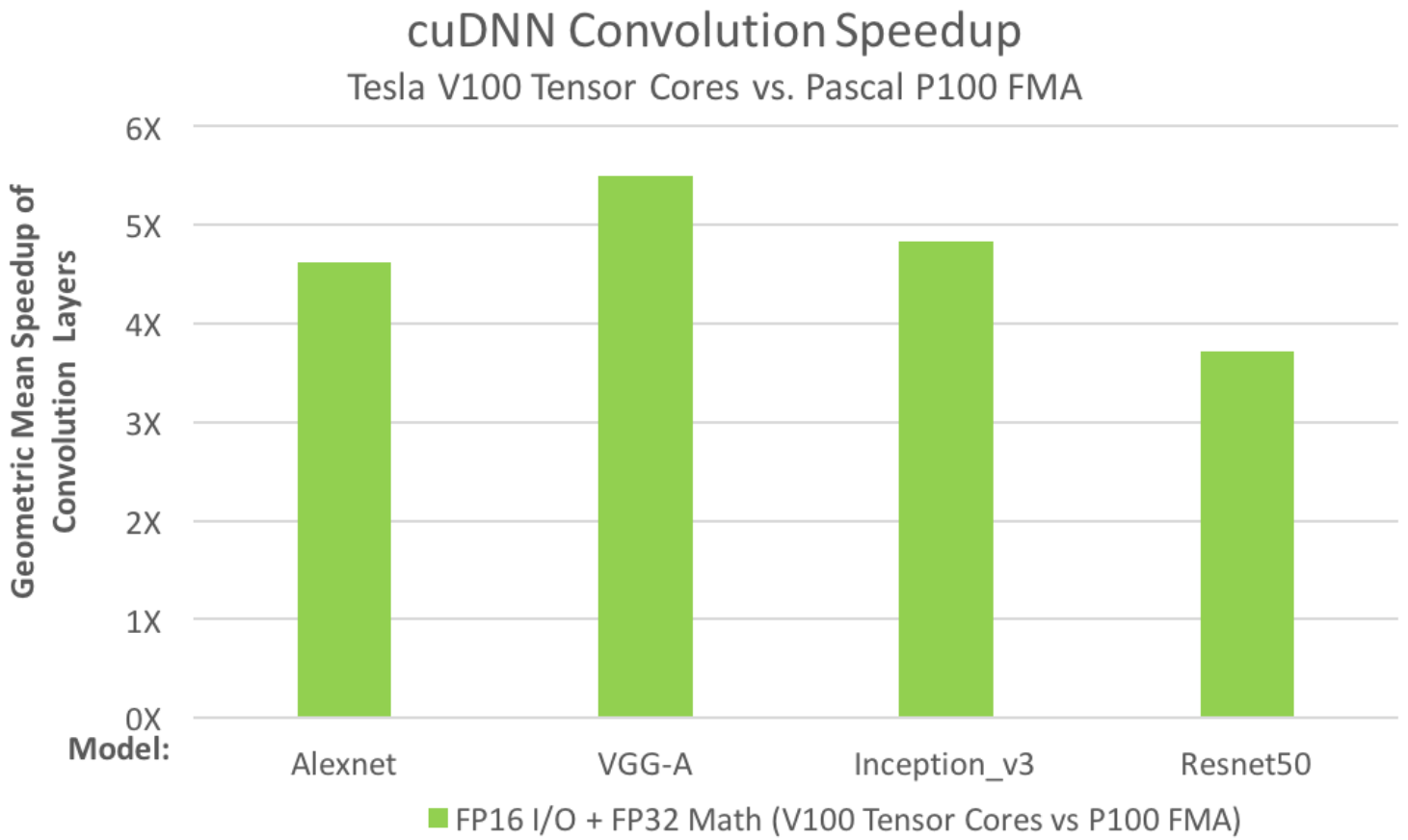


Performance comparison of matrix-matrix multiplication (GEMM)s https://developer-blogs.nvidia.com/wp-content/uploads/2017/10/tensor_core_cuBLAS_perf-1-e1508207263673.png

Tensor Cores in cuBLAS (2/2)

```
// First, create a cuBLAS handle:
cublasStatus_t cublasStat = cublasCreate(&handle);

// Set the math mode to allow cuBLAS to use Tensor Cores:
cublasStat = cublasSetMathMode(handle, CUBLAS_TENSOR_OP_MATH);

// Allocate and initialize your matrices (only the A matrix is shown):
size_t matrixSizeA = (size_t)rowsA * colsA;
T_ELEM_IN **devPtrA = 0;

cudaMalloc((void**)&devPtrA[0], matrixSizeA * sizeof(devPtrA[0][0]));
T_ELEM_IN A  = (T_ELEM_IN *)malloc(matrixSizeA * sizeof(A[0]));
memset( A, 0xFF, matrixSizeA* sizeof(A[0]));
status1 = cublasSetMatrix(rowsA, colsA, sizeof(A[0]), A, rowsA,
                          devPtrA[i], rowsA);
// ... allocate and initialize B and C matrices (not shown) ...

// Invoke the GEMM, ensuring k, lda, ldb, and ldc are all multiples of 8,
// and m is a multiple of 4:
cublasStat = cublasGemmEx(handle, transa, transb, m, n, k, alpha,
                          A, CUDA_R_16F, lda,
                          B, CUDA_R_16F, ldb,
                          beta, C, CUDA_R_16F, ldc, CUDA_R_32F, algo);
```

Tensor Cores in cuDNN (1/2)



Performance comparison of convolution `https://developer-blogs.nvidia.com/wp-content/uploads/2017/10/tensor_core_cudnn_speedup-1-e1508222018353.png`

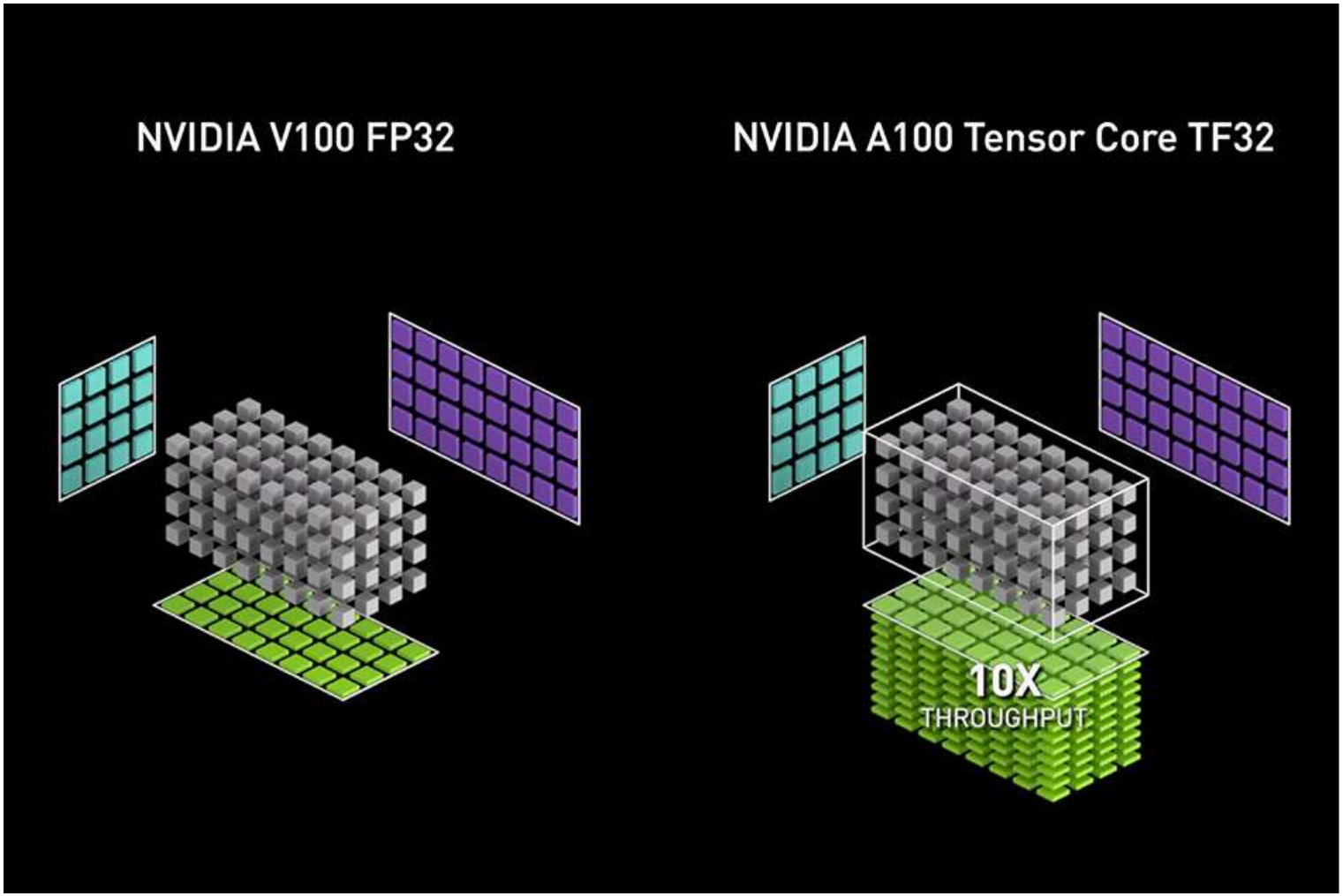
Tensor Cores in cuDNN (2/2)

```
// Create a cuDNN handle:
checkCudnnErr(cudnnCreate(&handle_));

// Create your tensor descriptors:
checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnIdesc ));
checkCudnnErr( cudnnCreateFilterDescriptor( &cudnnFdesc ));
checkCudnnErr( cudnnCreateTensorDescriptor( &cudnnOdesc ));
checkCudnnErr( cudnnCreateConvolutionDescriptor( &cudnnConvDesc ));

// Set tensor dimensions as multiples of eight
// (only the input tensor is shown here):
int dimA[] = {1, 8, 32, 32};
int strideA[] = {8192, 1024, 32, 1};

checkCudnnErr( cudnnSetTensorNdDescriptor(cudnnIdesc, getDataType(),
                                          convDim+2, dimA, strideA) );

// Allocate and initialize tensors (again, only the input tensor is shown)
checkCudaErr( cudaMalloc((void**)&(devPtrI), (insize) * sizeof(devPtrI[0]));
hostI = (T_ELEM*)calloc (insize, sizeof(hostI[0]) );
initImage(hostI, insize);
checkCudaErr( cudaMemcpy(devPtrI, hostI, sizeof(hostI[0]) * insize,
                         cudaMemcpyHostToDevice));
```

Tensor Cores in cuDNN (2/2)

```
// Set the compute data type (below as CUDNN_DATA_FLOAT):
checkCudnnErr( cudnnSetConvolutionNdDescriptor(cudnnConvDesc, convDim,
    padA, convstrideA, dilationA, CUDNN_CONVOLUTION, CUDNN_DATA_FLOAT));

// Set the math type to allow cuDNN to use Tensor Cores:
checkCudnnErr( cudnnSetConvolutionMathType(cudnnConvDesc,
                                           CUDNN_TENSOR_OP_MATH));

// Choose a supported algorithm:
cudnnConvolutionFwdAlgo_t algo =
                          CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM;

// Allocate your workspace:
checkCudnnErr( cudnnGetConvolutionForwardWorkspaceSize(handle_,
 cudnnIdesc, cudnnFdesc, cudnnConvDesc, cudnnOdesc, algo, &workSpaceSize));

if (workSpaceSize > 0) {cudaMalloc(&workSpace, workSpaceSize);}

// Invoke the convolution:
checkCudnnErr(cudnnConvolutionForward(handle_, (void*)(&alpha),
                  cudnnIdesc, devPtrI, cudnnFdesc, devPtrF, cudnnConvDesc,
                  algo, workSpace, workSpaceSize, (void*)(&beta),
                  cudnnOdesc, devPtrO));
```

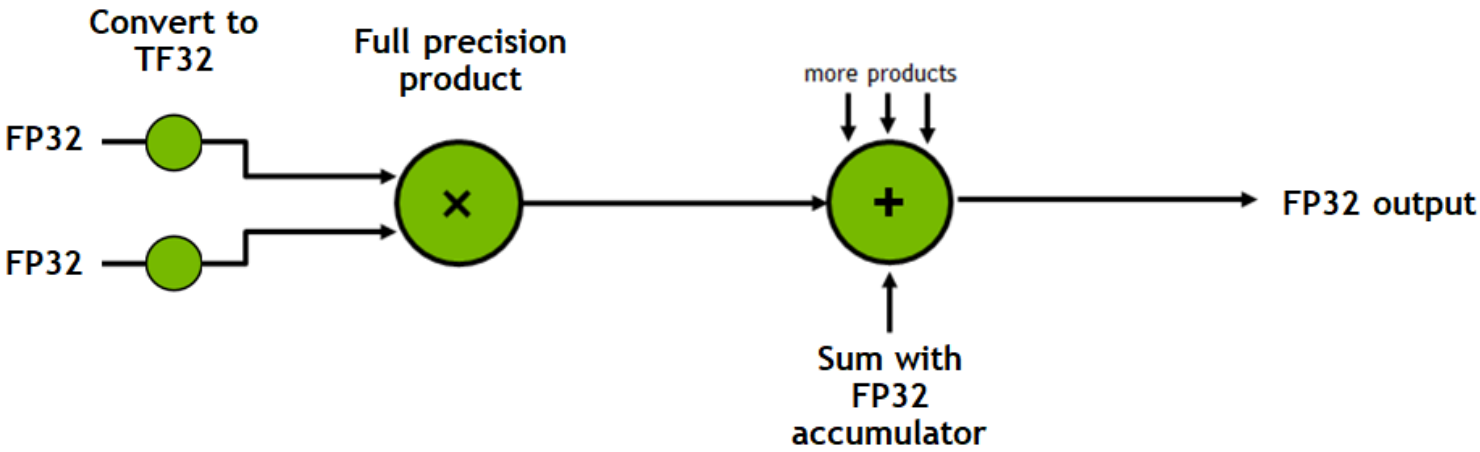
3rd Generation of Tensor Cores



https://developer-blogs.nvidia.com/wp-content/uploads/2021/01/AI_training_TF32_tensor_cores_Featured_Image.png

3rd Generation of Tensor Cores: Internals[1]

- TF32 is a compute mode added to Tensor Cores in the Ampere generation of GPU architecture
- Dot product computation is the building block for both matrix multiplies and convolutions

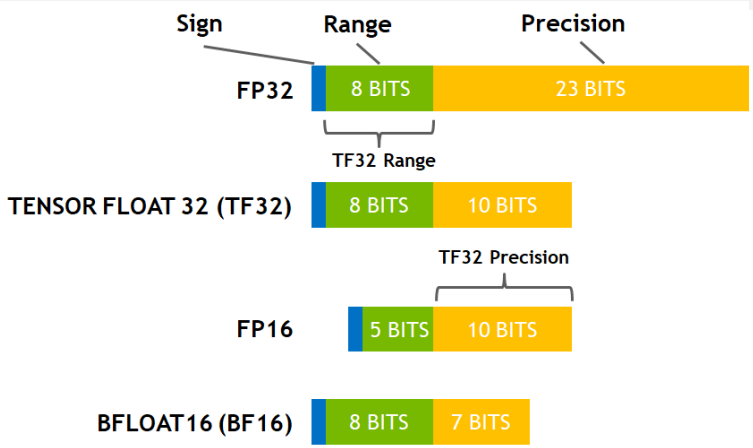


https://developer-blogs.nvidia.com/wp-content/uploads/2021/01/AI_training_TF32_tensor_cores_F1.png

- TF32 is only exposed as a Tensor Core operation mode, not a type
- All storage in memory and other operations remain completely in FP32, only convolutions and matrix-multiplications convert their inputs to TF32 right before multiplication

[1] https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

3rd Generation of Tensor Cores: Numerics[2]



https://developer-blogs.nvidia.com/wp-content/uploads/2021/01/AI_training_TF32_tensor_cores_F2.png

- bfloat (brain floating point): shortened version of the 32-bit single-precision floating point format (FP32)

- TF32 uses the same 10-bit mantissa as the half-precision (FP16) math
  $\rightarrow$ more than sufficient margin for the precision requirements of AI workloads
- TF32 adopts the same 8-bit exponent as FP32
  $\rightarrow$ it can support the same numeric range
- The combination makes TF32 a great alternative to FP32 for crunching through single-precision math, specifically the massive multiply-accumulate functions at the heart of deep learning and many HPC applications

[2] https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/

Mixed Precision[3]

- Mixed-precision training of deep neural networks
  - enables faster training
  - reduces memory requirements
- Possible to use larger batch sizes, larger models, or larger inputs
- Tensor Cores in Volta (FP16) provide an order of magnitude more throughput when compared to FP32

[3] https://developer.nvidia.com/gtc/2020/video/s22082-vid#:~:text=Mixed%2Dprecision%20training%20of%20deep,throughput%20when%20compared%20to%20FP32

Layout

Why Do We Use a Parallelism?[4]

- To fit very large models onto limited hardware
- To significantly speed up training
  - For example, to finish training that would take a year in hours

[4] https://huggingface.co/docs/transformers/v4.15.0/parallelism

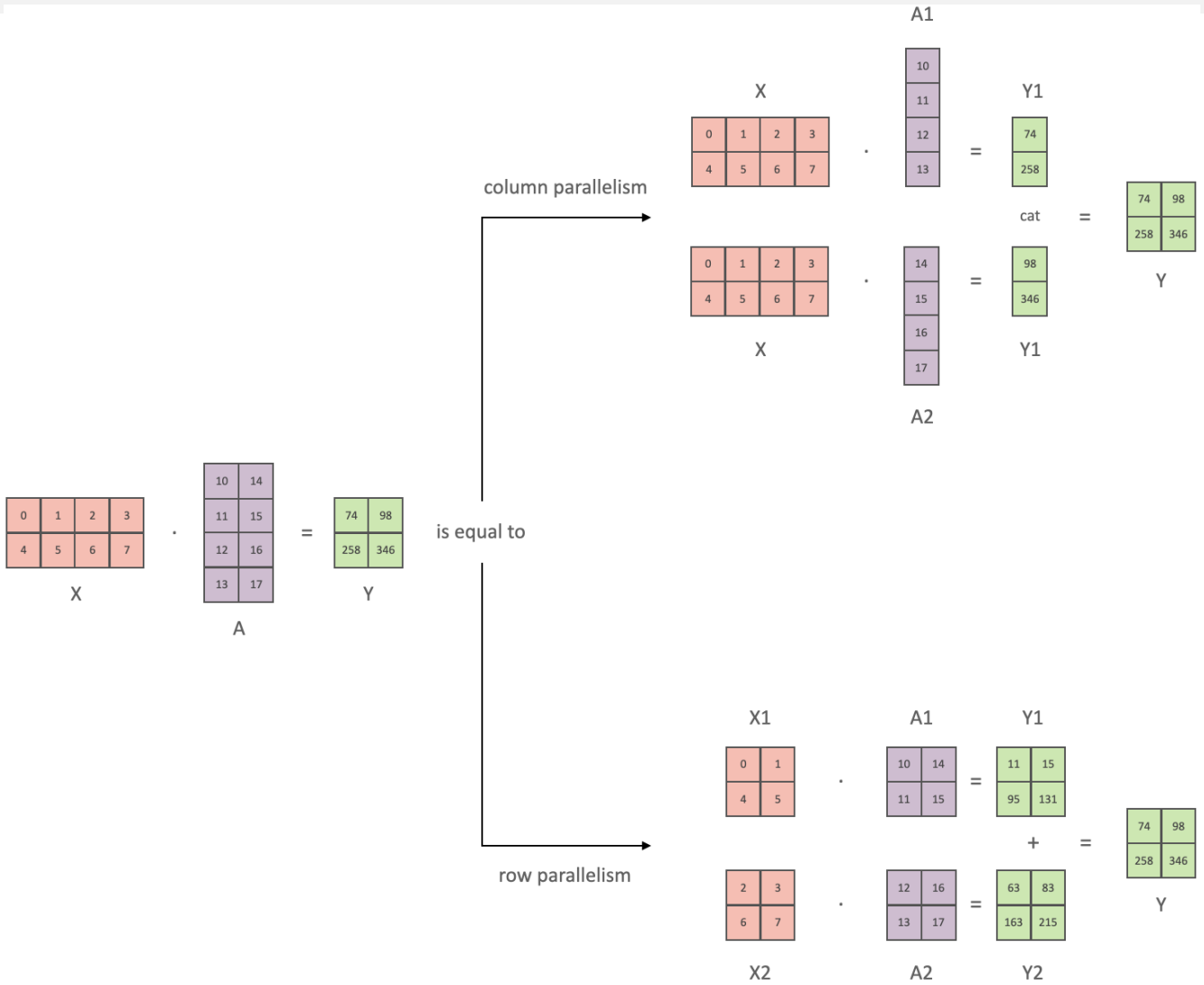
Different Types of Parallelism[5]

- ***Data Parallelism***
  - The same setup is replicated multiple times, and each being fed a slice of the data
- ***Pipeline Parallelism***
  - The model is split up vertically (layer-level) across multiple GPUs, so that only one or several layers of the model are places on a single gpu
- ***Tensor Parallelism***
  - See the next slide

[5] https://huggingface.co/docs/transformers/v4.15.0/parallelism

Tensor Parallelism[6]

- Splits a tensor along columns or rows
- Uses several GPUs simultaneously
- Requires regular synchronisation points, such as AllGather or AllReduce



https://cdn-lfs.huggingface.co/datasets/huggingface/documentation-images/db98c85824ac94c94508bf3a3e4e84c85154278a25943c14181e3b1918f62f55?response-content-disposition=inline%3B+filename*%3DUTF-8%27%27parallelism-tp-parallel_gemm.png%3B+filename%3D%22parallelism-tp-parallel_gemm.png%22%3B&response-content-type=image

Layout

Layout

Neural Compute Stick 2[7]

- Processor: Intel Movidius X Vision Processing Unit (VPU)
- It supports OpenVINO, a toolkit that accelerates solution development and streamlines deployment
- 16 powerful processing cores (called SHAVE cores) and a neural compute engine (a dedicated deep neural network accelerator)

https://www.intel.com/content/dam/develop/public/us/en/documents/ncs2-data-sheet.pdf

[7] https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html, https://www.mouser.fr/new/intel/intel-neural-compute-stick-2/

OpenVINO Toolkit[8]

- ***Model Optimizer***
  - It optimizes models for Intel architecture, converting models into a format (called an Intermediate Representation) compatible with the Inference Engine
- ***Intermediate Representation*** (IR)
  - Output: Model Optimizer
  - Input: Inference Engine
- ***Inference Engine***
  - Software libraries that run inference against the IR to produce inference results

https://www.intel.com/content/dam/develop/external/us/en/images/ncs2-transition-to-openvino-900x290-798445.png

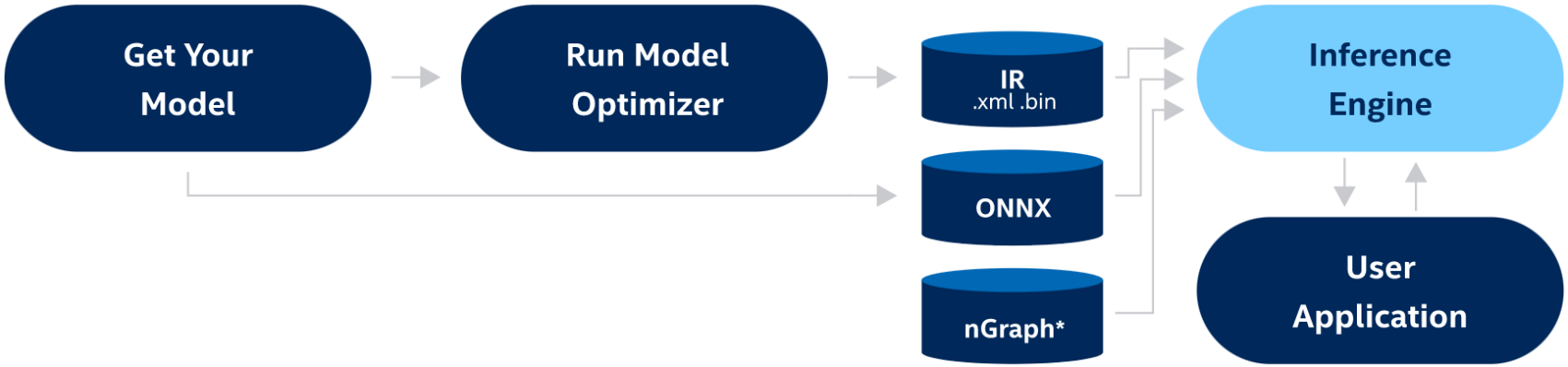
[8] https://docs.openvino.ai/latest/openvino_docs_get_started_get_started_windows.html#download-models

Available Models

- In total, 283 models
- For example
  - alexnet
  - googlenet
  - mobilenet
  - resnet
  - shufflenet
  - squeezenet
  - vgg16
  - vgg19
  - yolo

Main Steps[9]

1. Download the model(s)
2. Convert the model(s) to the intermediate representation
3. Prepare a video or a photo
4. Run the application
   - Possible to choose MYRIAD, CPU, GPU, or FPGA



https://docs.openvino.ai/latest/_images/BASIC_FLOW_IE_C.svg

[9] https://docs.openvino.ai/latest/openvino_docs_get_started_get_started_windows.html#download-models

Layout

Classification (1/2)

- Based on SqueezeNet topology

Example from Intel

```
classid probability label
------- ----------- -----
817     0.8295898   sports car, sport car
511     0.0961304   convertible
479     0.0439453   car wheel
751     0.0101318   racer, race car, racing car
436     0.0074234   beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon
656     0.0042267   minivan
586     0.0029869   half track
717     0.0018148   pickup, pickup truck
864     0.0013924   tow truck, tow car, wrecker
581     0.0006700   grille, radiator grille
```

https://actu.fr/ile-de-france/bouafle_78090/yvelines-bouafle-nouvel-arret-bus-rallier-mantes-cergy_30577954.html

```
classid probability label
------- ----------- -----
660     0.8061523   mobile home, manufactured home
705     0.0434265   passenger car, coach, carriage
757     0.0360107   recreational vehicle, RV, R.V.
874     0.0243835   trolleybus, trolley coach, trackless trolley
627     0.0205078   limousine, limo
803     0.0150223   snowplow, snowplough
675     0.0147705   moving van
867     0.0134430   trailer truck, tractor trailer, trucking rig, rig, articulated lorry, semi
829     0.0088348   streetcar, tram, tramcar, trolley, trolley car
654     0.0073128   minibus
```

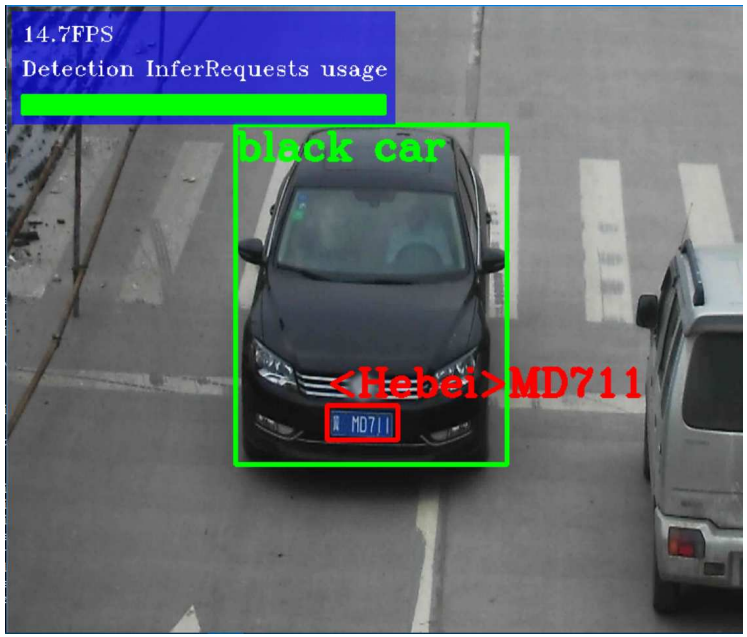
Classification (2/2)

https://upload.wikimedia.org/wikipedia/commons/f/ff/Air_France_A380-800%28F-HPJD%29_%285233706017%29.jpg

```
classid probability label
------- ----------- -----
404     0.9990234   airliner
812     0.0010996   space shuttle
908     0.0002387   wing
8       0.0000000   hen
7       0.0000000   cock
3       0.0000000   tiger shark, Galeocerdo cuvieri
1       0.0000000   goldfish, Carassius auratus
4       0.0000000   hammerhead, hammerhead shark
9       0.0000000   ostrich, Struthio camelus
5       0.0000000   electric ray, crampfish, numbfish, torpedo
```

Vehicle Application

- Vehicle detection
- Vehicle attributes recognition
- Vehicle number plate recognition



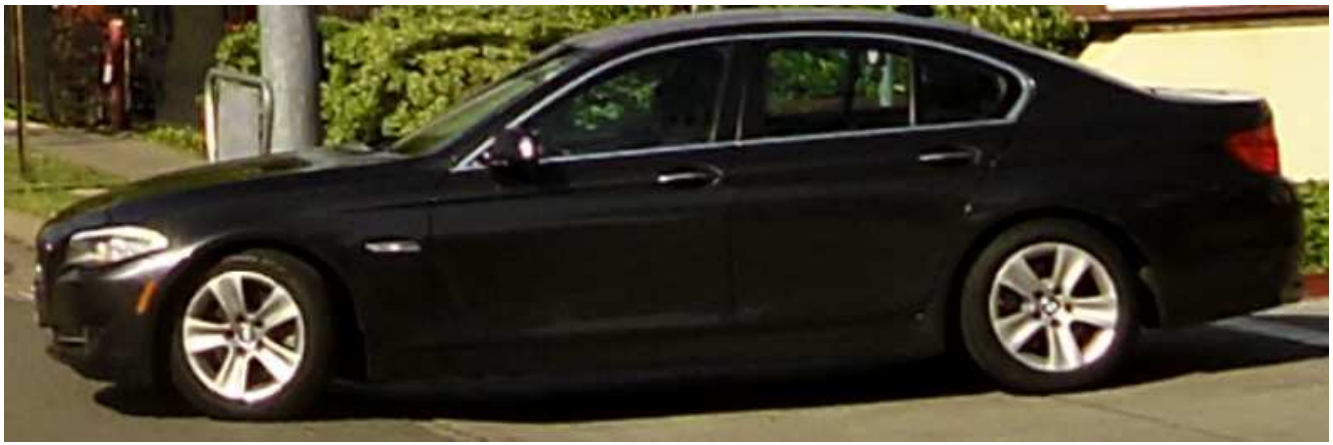
Example from Intel

http://www.olavsplates.com/foto_c/chn_q-p2138.jpg

Benchmark Application

- To estimate deep learning inference performance
- Based on SqueezeNet topology

- ***MYRIAD***: Intel Movidius Myriad X VPU
- ***CPU***: Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
- ***GPU***: Intel(R) HD Graphics 520 (iGPU)



Example from Intel

```
[Step 11/11] Dumping statistics report
Count:      16476 iterations
Duration:   60016.42 ms
Latency:    14.54 ms
Throughput: 274.52 FPS
```

MYRIAD

```
[Step 11/11] Dumping statistics report
Count:      9504 iterations
Duration:   60042.21 ms
Latency:    24.47 ms
Throughput: 158.29 FPS
```

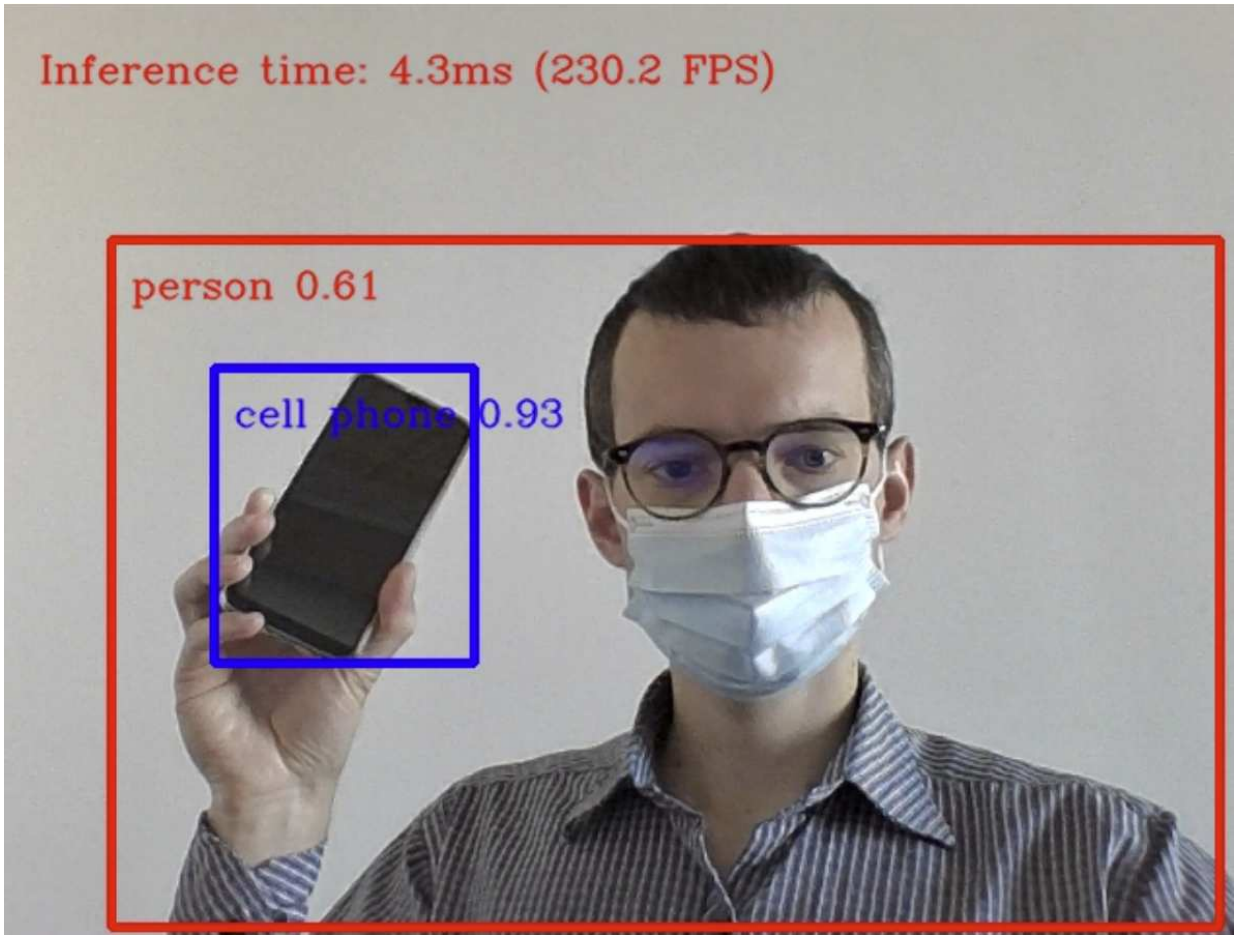
CPU

```
[Step 11/11] Dumping statistics report
Count:      20704 iterations
Duration:   60018.65 ms
Latency:    10.93 ms
Throughput: 344.96 FPS
```

GPU

Live Object Detection

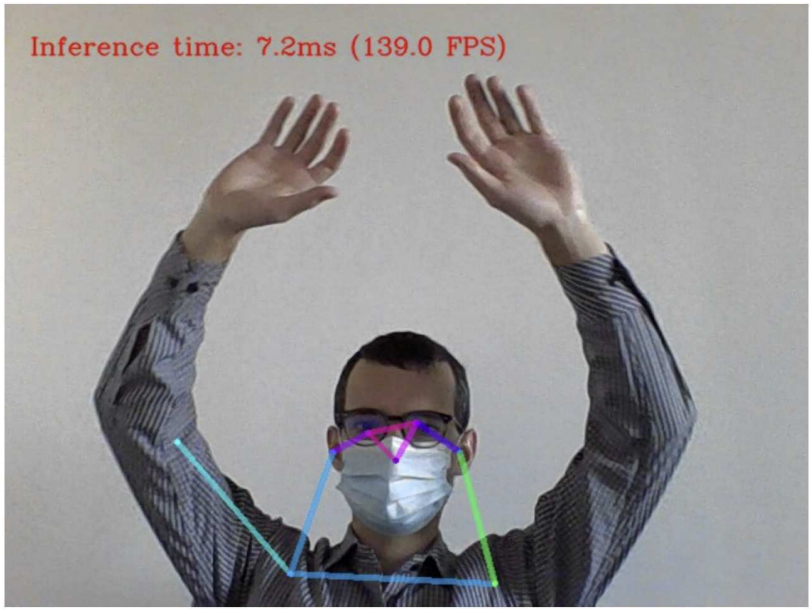
- SSDLite MobileNetV2 from Open Model Zoo
- COCO dataset[10]: 91 labels



[10] https://tech.amikelive.com/node-718/what-object-categories-labels-are-in-coco-dataset/ and https://github.com/amikelive/coco-labels

Live Human Pose Estimation

- OpenPose model `human-pose-estimation-0001` from Open Model Zoo

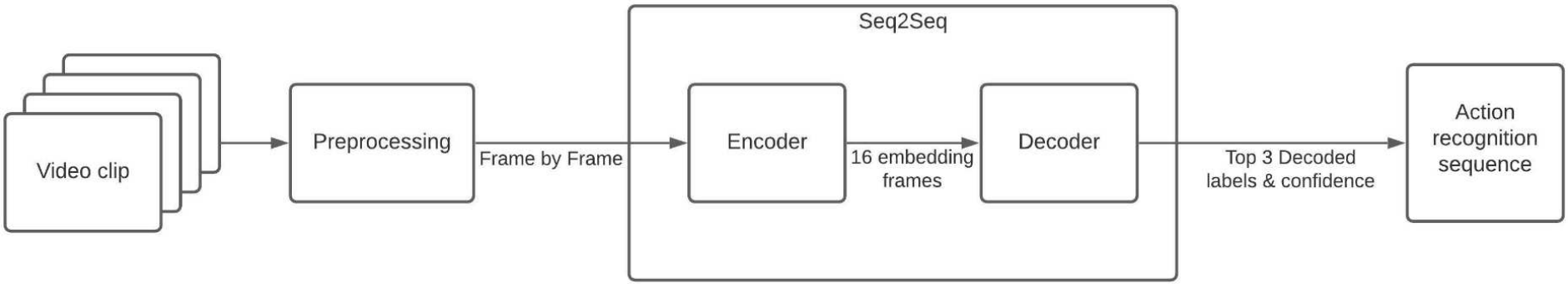


Webcam

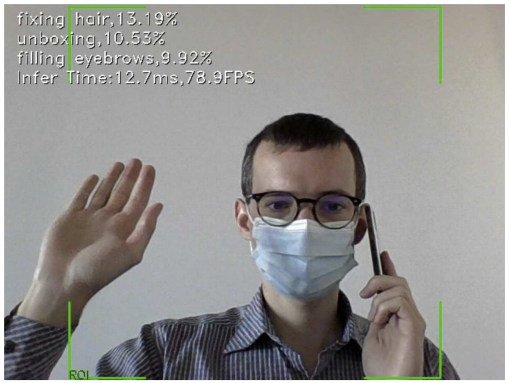
Video (Example from Intel)

Human Action Recognition

- Action Recognition Models from Open Model Zoo, specifically an Encoder and a Decoder
- Sequence to sequence system in order to identify the human activities for Kinetics-400 dataset[11]
- Video Transformer approach with ResNet34 encoder



Schema from tutorial 403-action-recognition-webcam

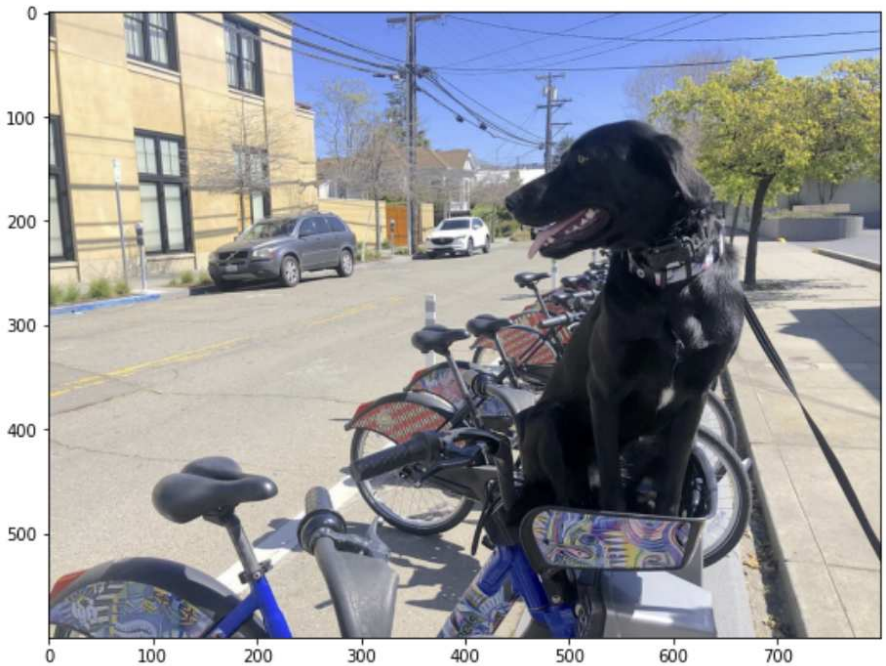


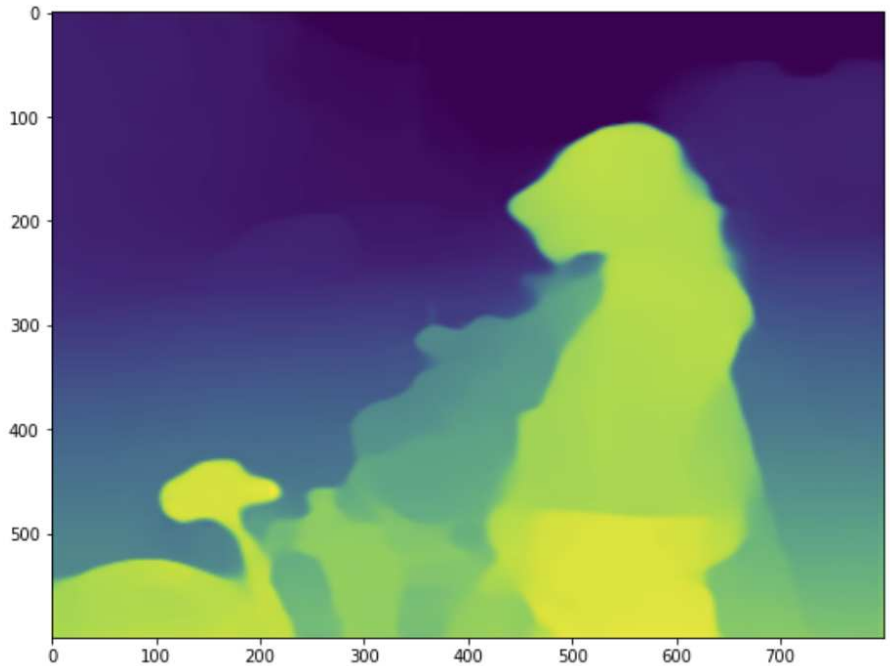
- Labels are not correctly assigned
- Maybe use Kinetics dataset with more labels

[11] https://arxiv.org/abs/1705.06950 and https://www.deepmind.com/open-source/kinetics

Monodepth Estimation

- Monocular Depth Estimation: task of estimating scene depth
- Neural network model called MiDaS developed by the Embodied AI Foundation[12]





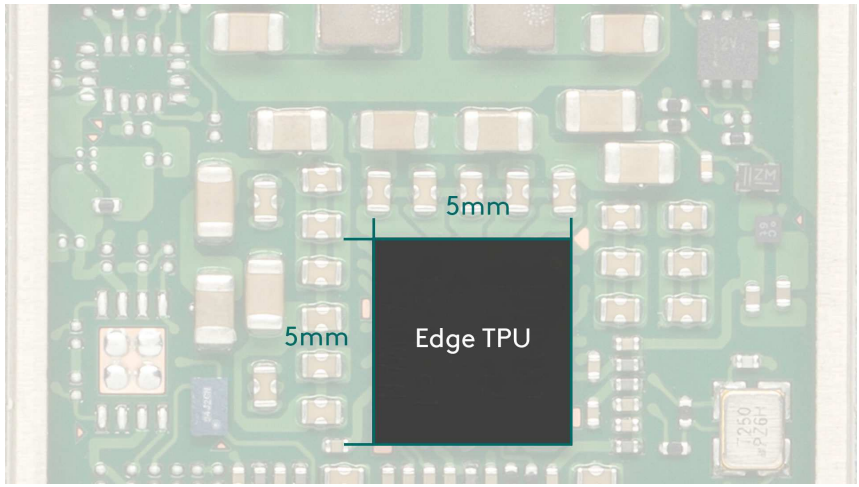
Example from Intel

[12] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler and V. Koltun, "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer," in IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2020.3019967.

Layout

General Facts

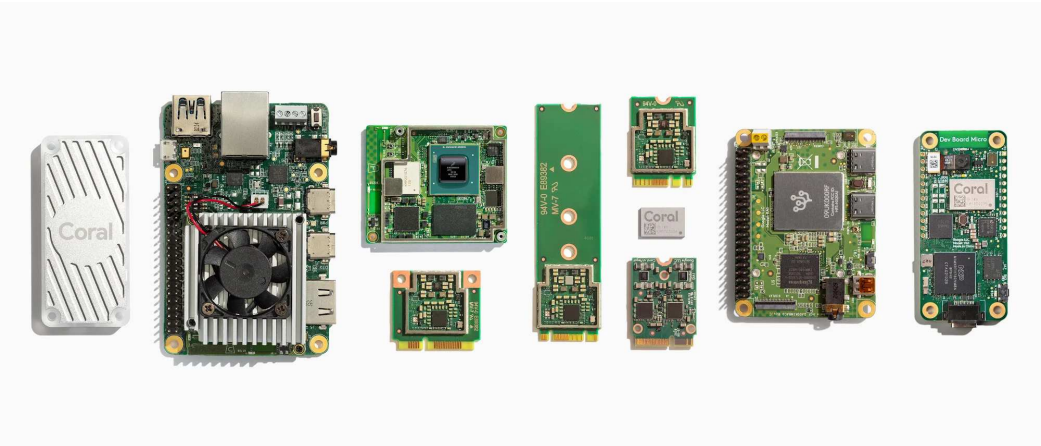
- `https://coral.ai/products/`
- Advanced neural network processing for low-power devices
- Hardware and software platform for building intelligent devices with fast neural network inferencing
- Based on the Coral Edge TPU coprocessor: small ASIC built by Google that's specially-designed to execute state-of-the-art neural networks at high speed, with a low power cost
- Three types of products: prototyping, production and sensing



`https://coral.ai/static/images/technology/technology-hero.jpg?=s0-rw`

User-Friendly

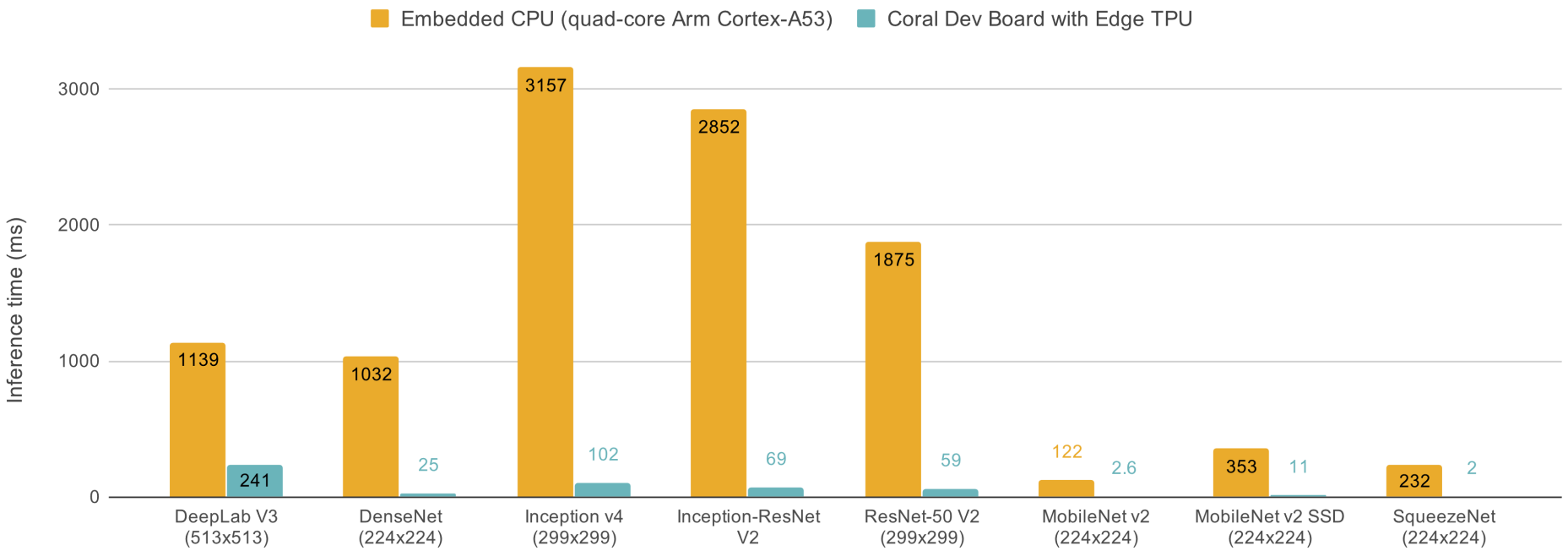
- The Edge TPU supports a variety of model architectures built with TensorFlow, including models built with Keras
- The workflow to create a model for the Edge TPU is based on TensorFlow Lite
- Pre-compiled models are available
- Model pipelining is possible
- ***Mendel Linux***: an open-source derivative of Debian Linux to ease development with our fully-integrated systems (the Dev Board, Dev Board Mini, and System-on-Module)
- ***coralmicro***: a real-time OS platform ased on FreeRTOS to simplify development for a microcontoller (MCU) board



https://lh3.googleusercontent.com/-ROH37d9aKorHo_VYWf8hCfukvbZolBaW2SHW1uDDn1G411r3MqemjxPZa9f44q8OwlfYIkGxSoj-GQbZGd2j7lxtyzSklIQVUWvo9r88mn8CzB-rcw=w2000-rw

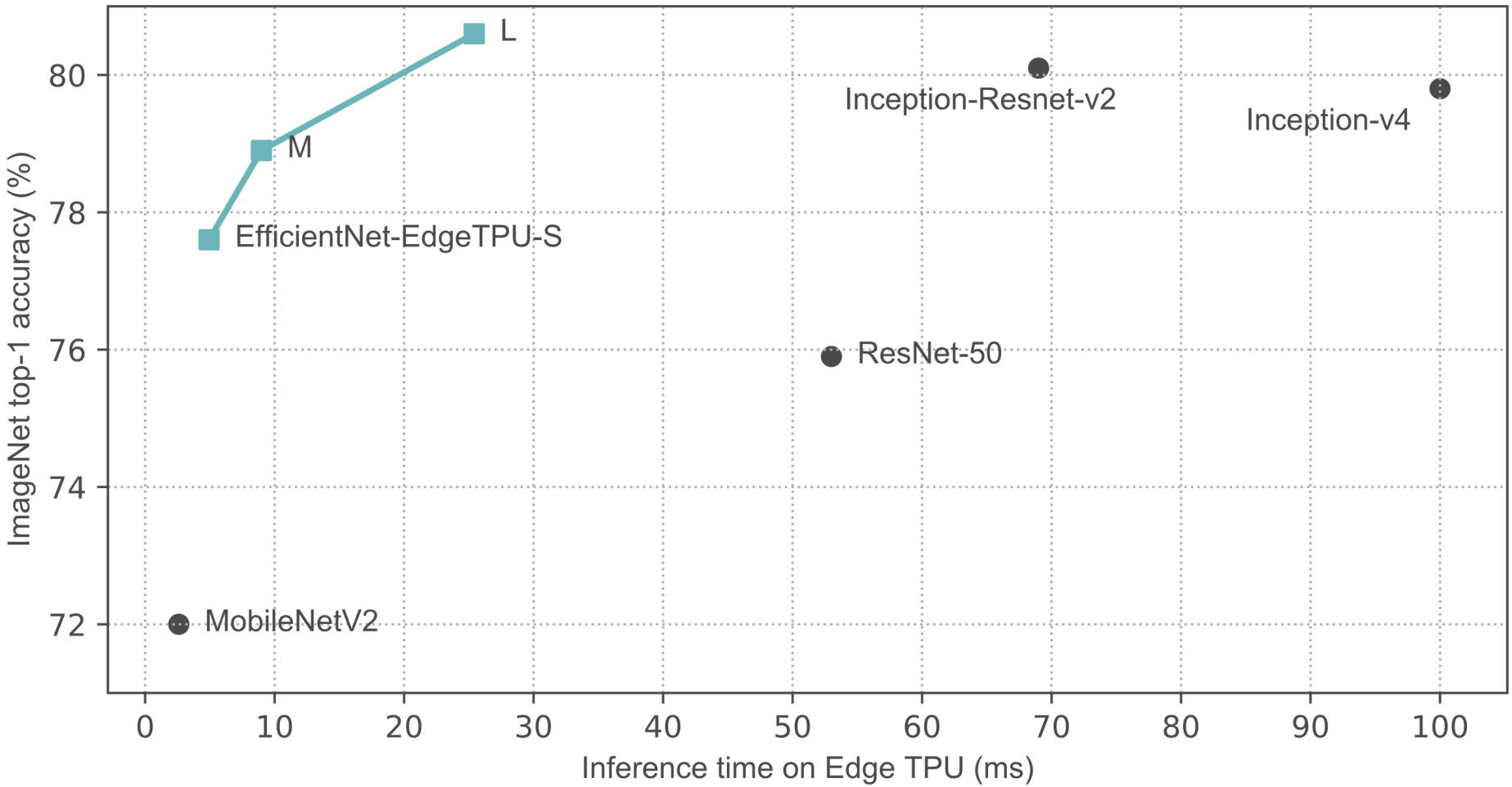
Performance (1/2)

- The Edge TPU is capable of performing 4 trillion operations (tera-operations) per second (TOPS), using 0.5 watts for each TOPS (2 TOPS per watt)
- The comparison of the inference time for several popular vision models in TensorFlow Lite format: a modern embedded CPU vs. on the Coral Dev Board



https://coral.ai/static/images/technology/edgetpu-benchmarks.svg?=s0-rw

Performance (2/2)



https://coral.ai/static/images/technology/edgetpu-efficientnet.svg?=s0-rw

Bibliography I

[1] *Fidle (Formation d'Introduction au Deep Learning)*.
https://fidle.cnrs.fr.

[2] *Google Cloud TPU System Architecture Documentation*.
https://cloud.google.com/tpu/docs/system-architecture-tpu-vm#targetText=Tensor%20Processing%20Units%20(TPUs)%20are,and%20leadership%20in%20machine%20learning.

[3] N. P. JOUPPI, D. H. YOON, G. KURIAN, S. LI, N. PATIL, J. LAUDON, C. YOUNG, AND D. PATTERSON, *A Domain-Specific Supercomputer for Training Deep Neural Networks*, Communucitaions of the ACM, 63 (2020), p. 67–78.
https://doi.org/10.1145/3360307.