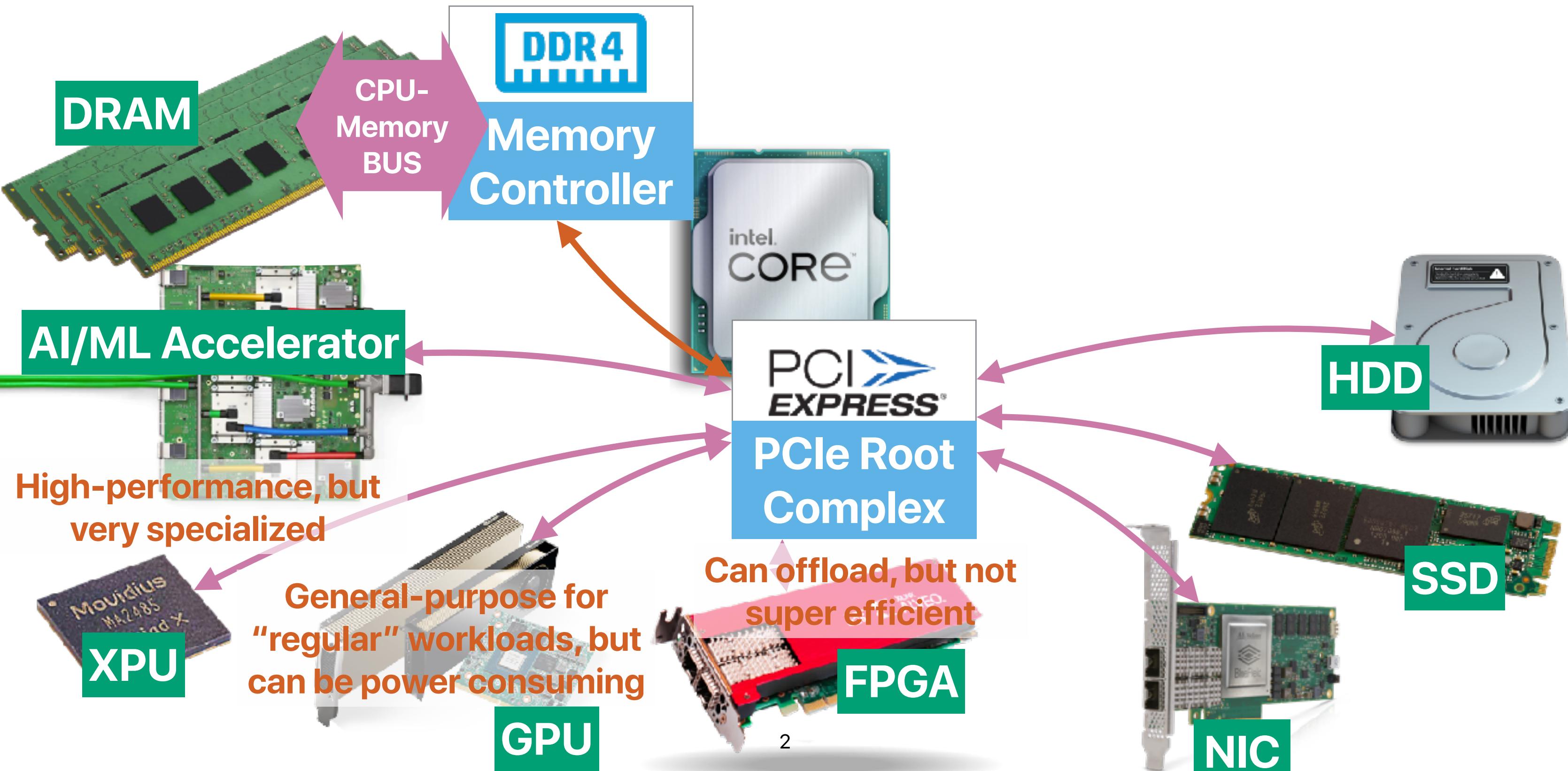


General purpose computing on hardware accelerators

Hung-Wei Tseng

Recap: The landscape of modern computers



**Can we take both advantages of
programmability and performance?**

Outline

- Neural Acceleration for General-Purpose Approximate Programs (Approximate computing on NPUs)
- Edge Tensor Processing Units
- Accelerating Applications using Edge Tensor Processing Units (GP Edge TPU)
- TCUDB: Accelerating Database with Tensor Processors
- RTNN: accelerating neighbor search using hardware ray tracing

Neural Acceleration for General-Purpose Approximate Programs

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, Doug Burger*

University of Washington and Microsoft*

In 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2012)

Recap: The concept of approximate computing on NPUs

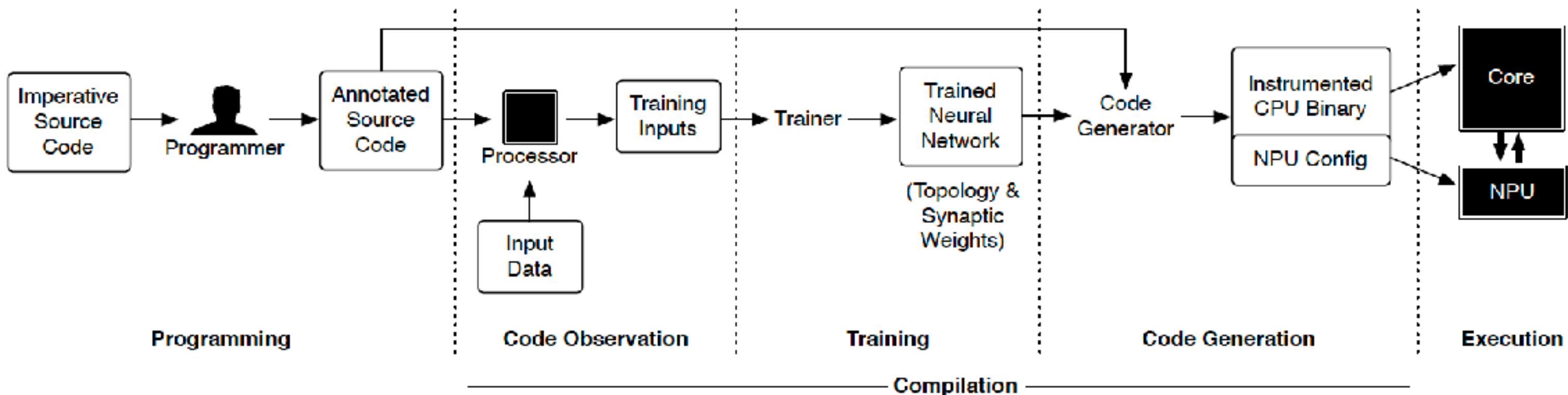
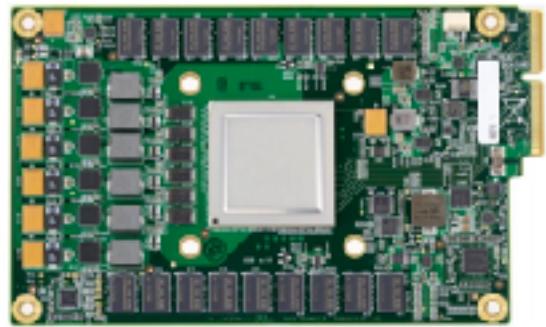


Figure 1: The Parrot transformation at a glance: from annotated code to accelerated execution on an NPU-augmented core.

Edge TPUs

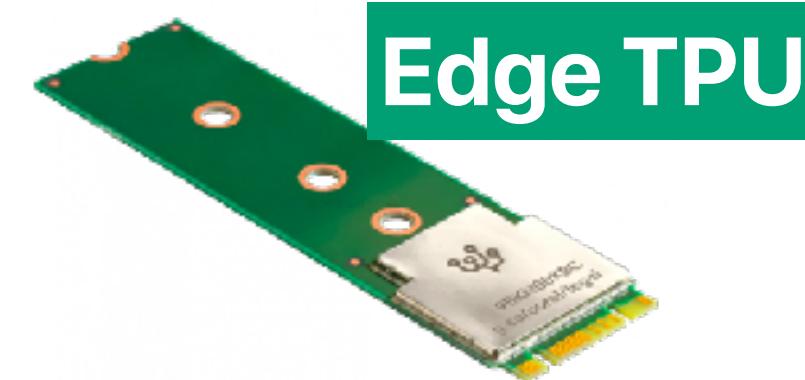
Variants of Google TPUs



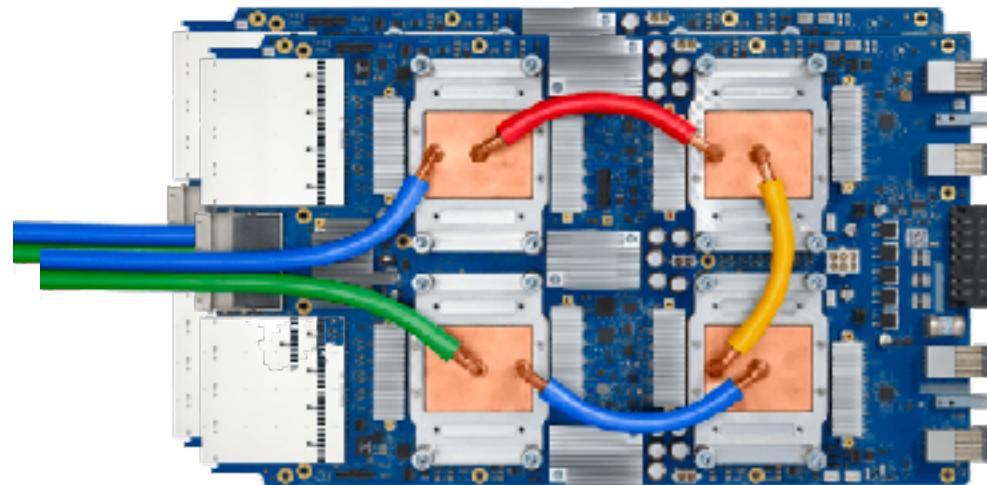
TPU v1



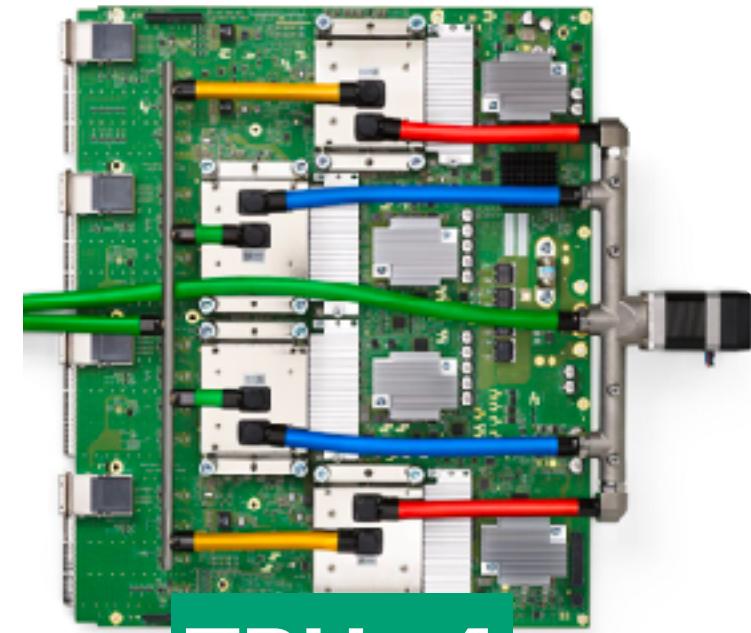
TPU v2



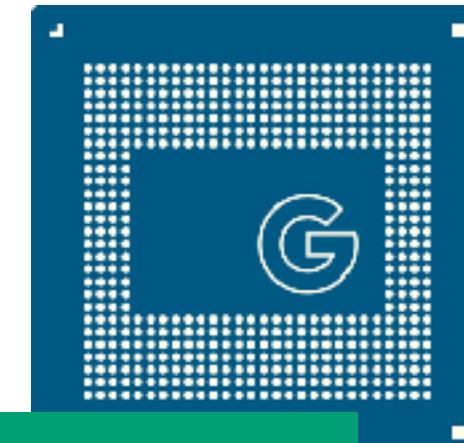
Edge TPU



TPU v3

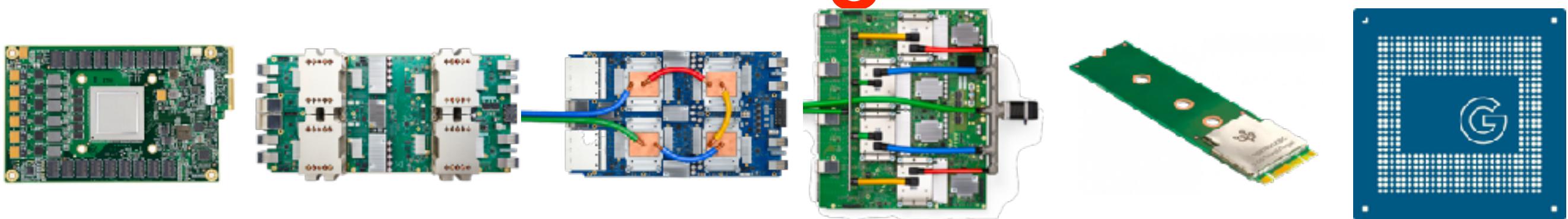


TPU v4



Google Tensor

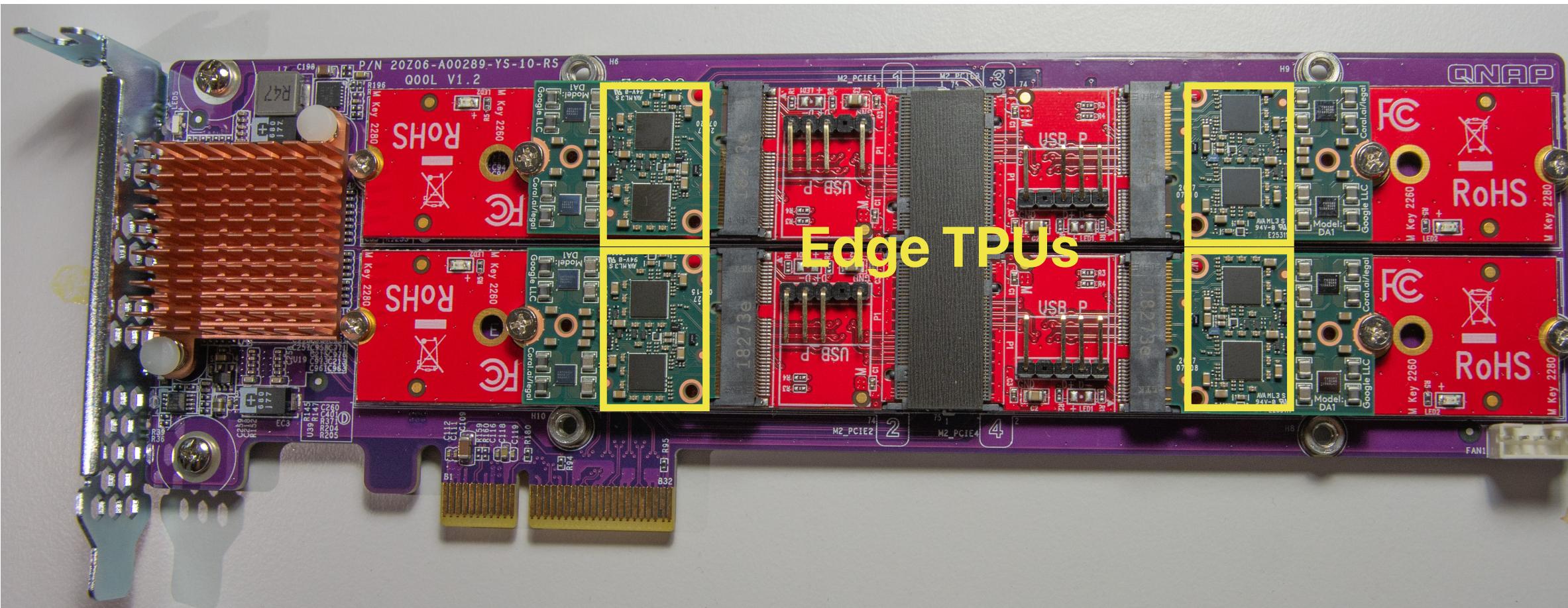
Variants of Google TPUs



	TPU v1	TPU v2	TPU v3	TPU v4	Edge TPU	Tensor
Year	2016	2017	2018	2021	2018	2021
Precision	INT8	BF16	BF16	BF16	INT8	?
Accessibility	Google Cloud Only	Google Cloud Only	Google Cloud Only	Google Cloud Only	Public Available	Only in Google Pixels
Programming Framework	Tensorflow/PyTorch	Tensorflow/PyTorch	Tensorflow/PyTorch	Tensorflow/PyTorch	Tensorflow/TFLite/C++	TFLite/NNAPI
TOPS	95	45	129	275	4	27
Power (Watt)	75	280	220	170	2	5
TOPS/W	1.27	0.16	0.59	1.62	2	5.4
Device Memory	24 MB	64 GB	128 GB	32 GB	8MB	N/A

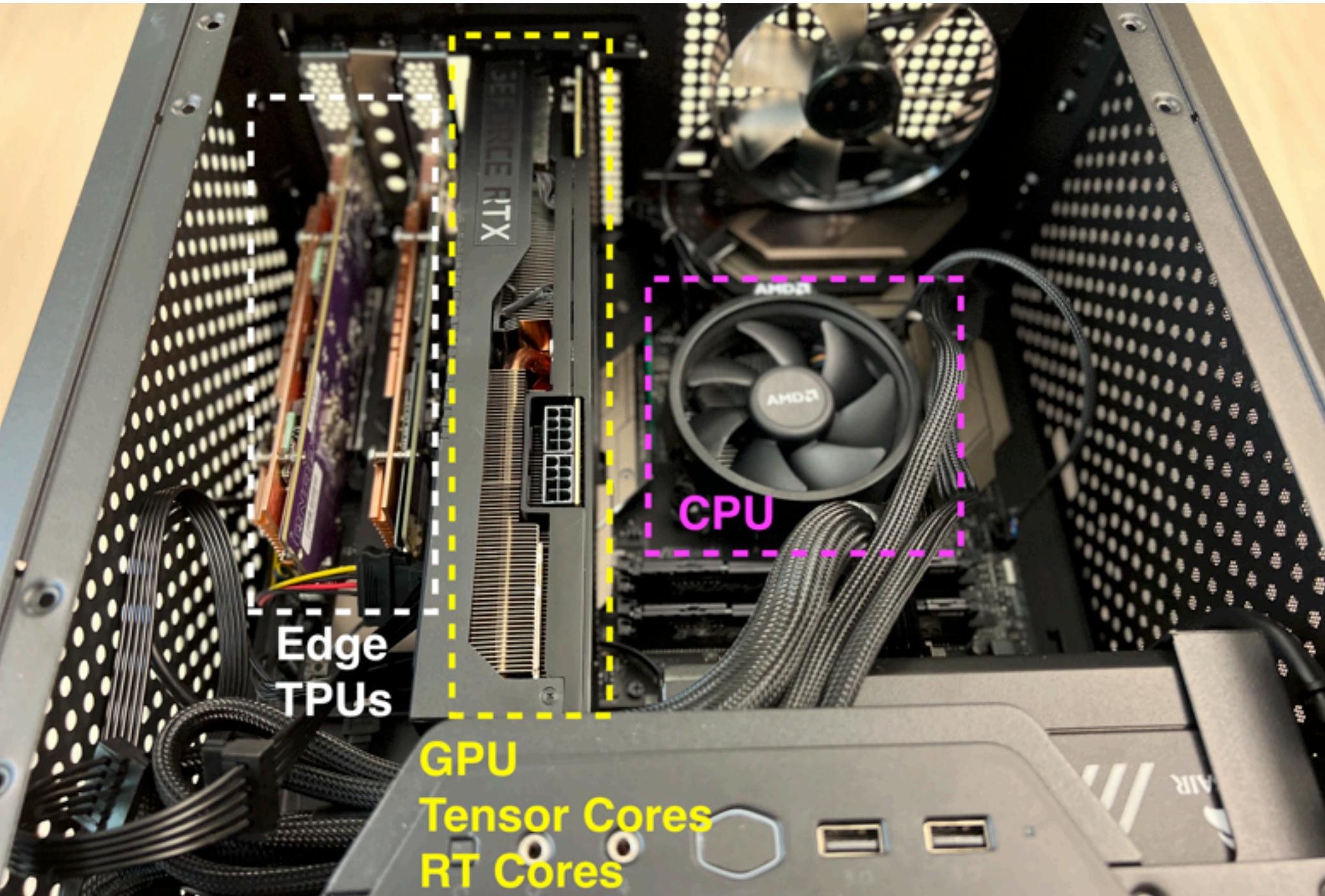
* https://www.reddit.com/r/GooglePixel/comments/1723547/just_how_fast_is_the_google_tensor_3/

Our Edge TPU Expansion Card



- Each TPU can deliver up to 4 Trillion Operations (TOPS) with 2 Watts of Power and cost USD 29
- Each card contains 4 TPUs — 16 TOPS
- GPUs can deliver 130 TOPS — GPU is still more powerful, but more power consuming, costs a lot more
- The only type of TPUs that you can order online — both in M.2 and USB

Heterogeneous computer with Edge TPU card



The concept of approximate computing on NPUs

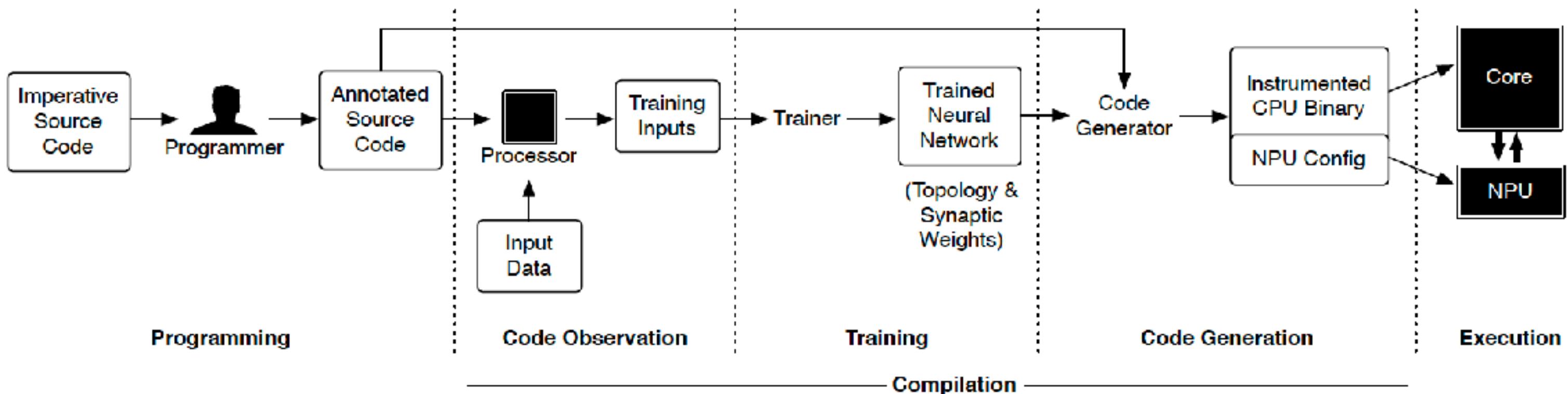
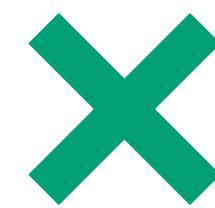


Figure 1: The Parrot transformation at a glance: from annotated code to accelerated execution on an NPU-augmented core.

Let's try a simple problem — binary MM

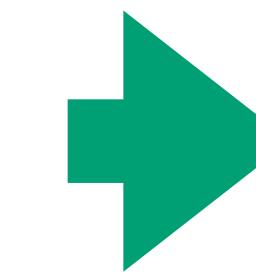
0	1	0	1
0	1	1	0
1	0	1	1
0	0	0	1



1	0
0	0
1	1
1	1



1	1
1	1
3	2
1	1



0	0
0	0
1	1
0	0

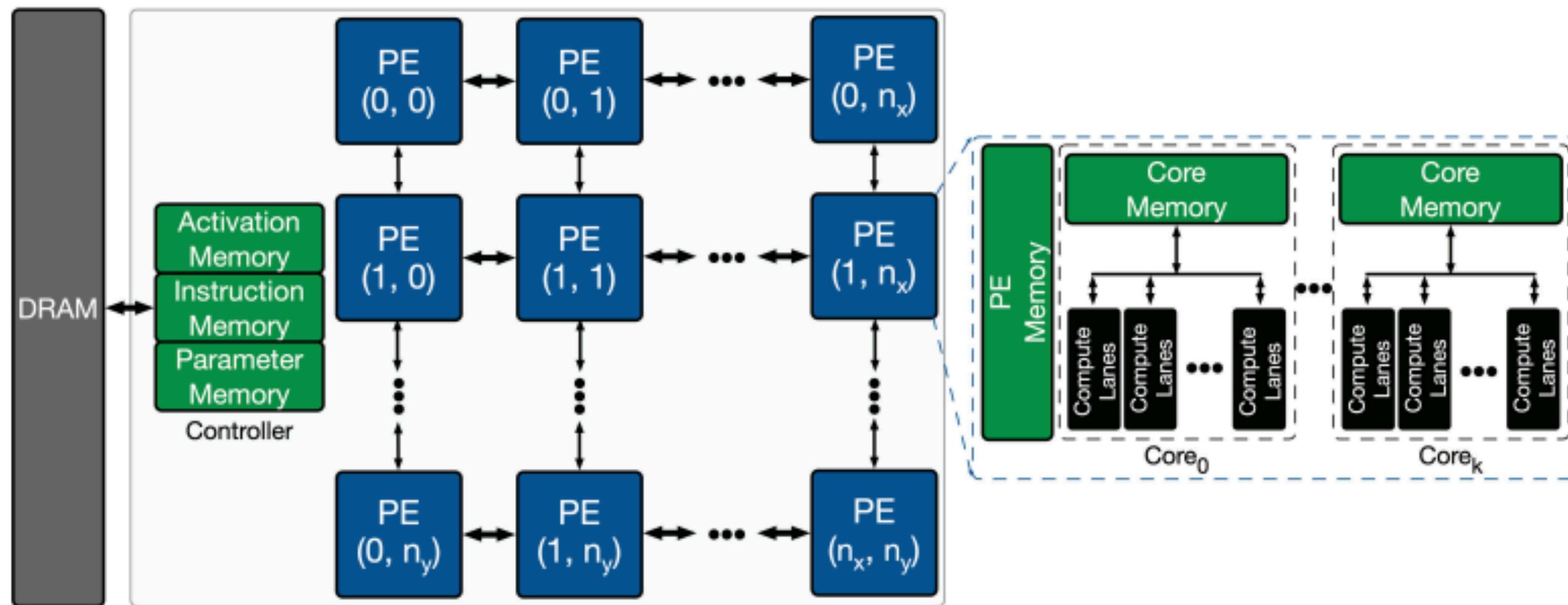


Figure 1: Overview of the template-based machine learning accelerator used for architecture exploration.

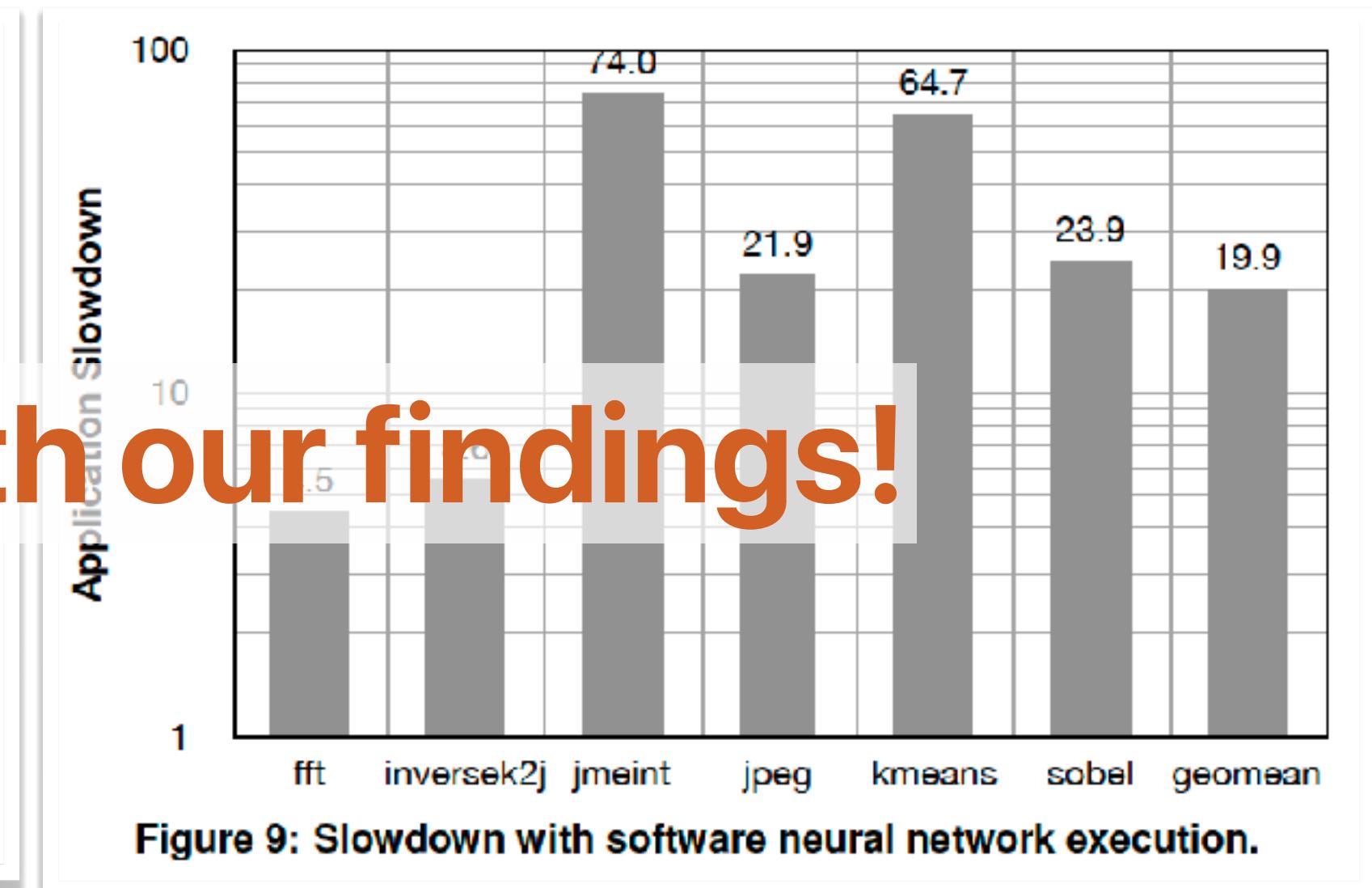
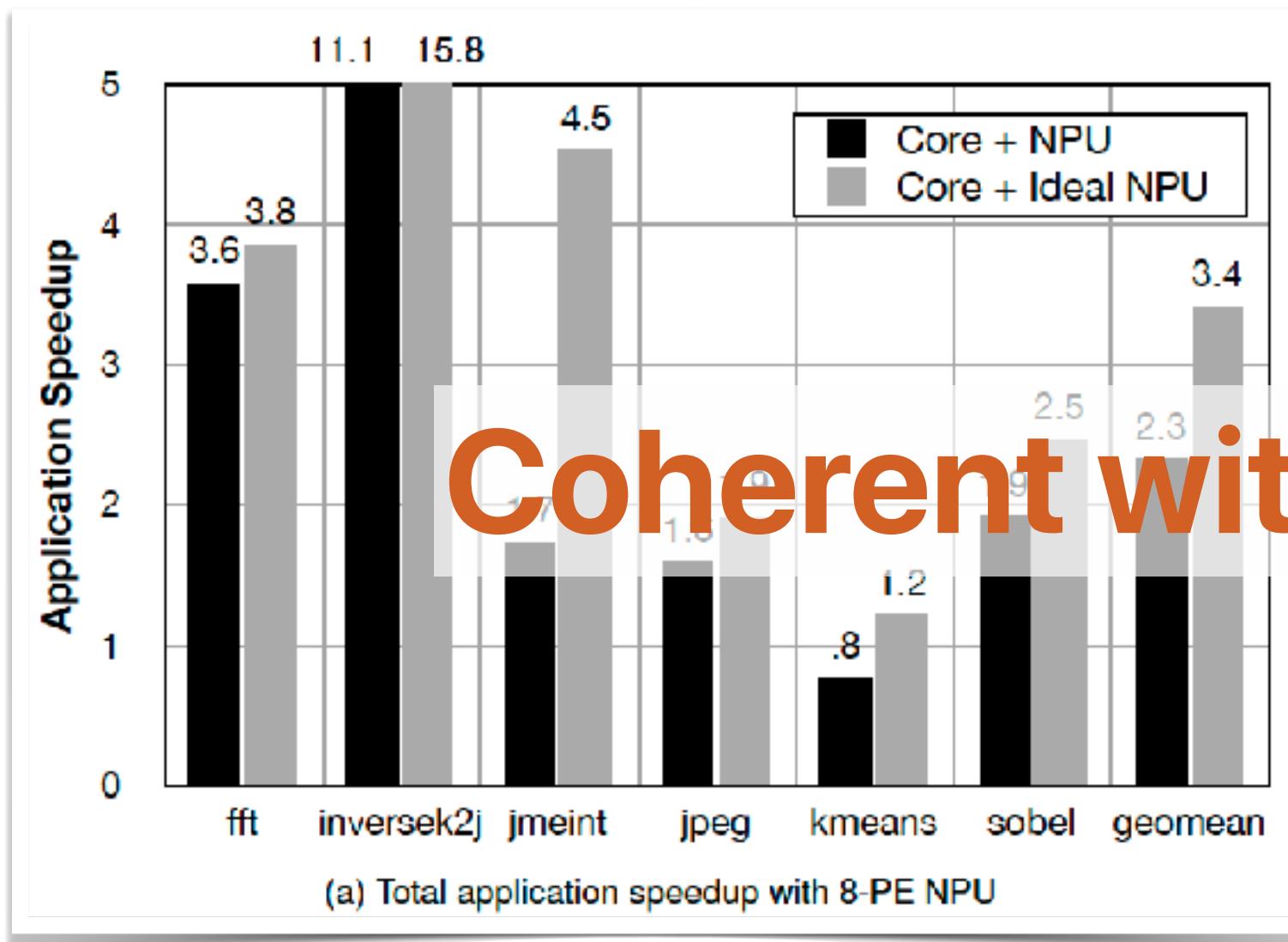
How are you going to apply the idea of
“approx. computing on NPUs” to an
AI/ML accelerator-equipped system?

Ideas?

Ideas?

- Format the input/output of the target algorithm as input tensors and labels
- Create training/testing datasets by running the algorithm
- Train a model using the generated datasets
- Run the algorithm as inferences on this model

The idea works more efficiently if accelerator exists



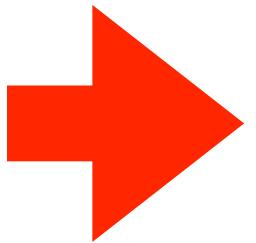
Why do you think NVIDIA succeed
in promoting GPGPU?

Why is NVIDIA successful?

Can we create another revolution with TPUs?

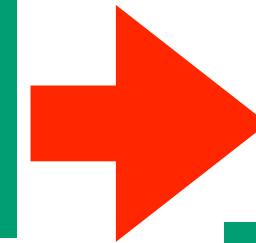


Graphics "Accelerator", 1999

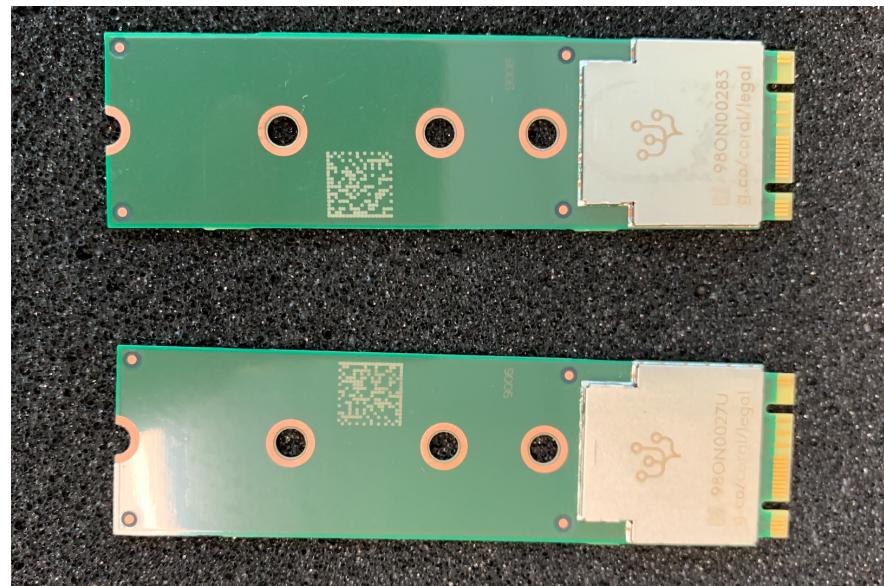


```
1 global__ void
2 vectorAdd(const float *A, const float *B, float *C, int numElements)
3 {
4     int i = blockDim.x * blockIdx.x + threadIdx.x;
5
6     if (i < numElements)
7     {
8         C[i] = A[i] + B[i];
9     }
10 }
```

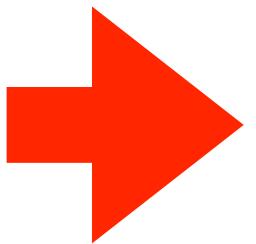
G80 (e.g., GTX 8800) with CUDA, 2007



General Purpose Computing on GPUs

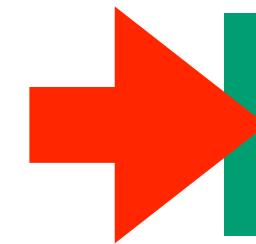


AI "Accelerator", 2013—



- Programming Language
- Compiler
- Runtime

?



General Purpose Computing on TPUs

Accelerating Applications using Edge Tensor Processing Units

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, Doug Burger*

University of Washington and Microsoft*

In 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2012)

Edge TPU Operators

Table 1. All operations supported by the Edge TPU and any known limitations

Operation name	Runtime version*	Known limitations
Add	All	
AveragePool2d	All	No fused activation function.
Concatenation	All	No fused activation function. If any input is a compile-time constant tensor, there must be only 2 inputs, and this constant tensor must be all zeros (effectively, a zero-padding op).
Conv2d	All	Must use the same dilation in x and y dimensions.
DepthwiseConv2d	≤12	Dilated conv kernels are not supported.
	≥13	Must use the same dilation in x and y dimensions.
ExponentCoss	≥15	
FullyConnected	All	Only the default format is supported for fully-connected weights. Output tensor is one-dimensional.
L2Normalization	All	
Logistic	All	
LSTM	≥14	Unidirectional LSTM only.
Maximum	All	
MaxPool2d	All	No fused activation function.
Mean	≤12	No reduction in batch dimension. Supports reduction along x- and/or y-dimensions only.
	≥13	No reduction in batch dimension. If a z-reduction, the z-dimension must be multiple of 4.
Minimum	All	
Mul	All	
Pack	≥15	No padding in batch dimension.
Pad	≤12	No padding in batch dimension. Supports padding along x- and/or y-dimensions only.
	≥13	No padding in batch dimension.

<https://coral.ai/docs/edgetpu/models-intro/#supported-operations>

PReLU	≥13	Alpha must be 1-dimensional (only the innermost dimension can be >1 size). If using Keras PReLU with 4D input (batch, height, width, channels), then shared_axes must be [1,2] so each filter has only one set of parameters.
Quantize	≥13	
ReduceMax	≥14	Cannot operate on the batch dimension.
ReduceMin	≥14	Cannot operate on the batch dimension.
ReLU	All	
ReLU6	All	
ReLUNTI	All	
Reshape	All	Certain reshapes might not be mapped for large tensor sizes.
ResizeBilinear	All	Input/output is a 3-dimensional tensor. Depending on input/output size, this operation might not be mapped to the Edge TPU to avoid loss in precision.
ResizeNearestNeighbor	All	Input/output is a 3-dimensional tensor. Depending on input/output size, this operation might not be mapped to the Edge TPU to avoid loss in precision.
Rsqrt	≥14	
Slice	All	
Softmax	All	Supports only 1-D input tensor with a max of 16,000 elements.
SpaceToDepth	All	
Split	All	No splitting in batch dimension.
Squeeze	≤12	Supported only when input tensor dimensions that have leading 1s (that is, no relayout needed). For example input tensor with [y][x][z] = 1,1,10 or 1,5,10 is ok. But [y][x][z] = 5,1,10 is not supported.
	≥13	None.
StridedSlice	All	Supported only when all strides are equal to 1 (that is, effectively a Stride op), and with ellipsis-axis-mask = 0, and new-axis-max = 0.
Sub	All	
Sum	≥15	Cannot operate on the batch dimension.
Squared-difference	≥14	
Tanh	All	
Transpose	≥14	
TransposeConv	≥13	

Programming Interface

Synopsis	Definition
<code>openctpu_dimension *openctpu_alloc_dimension(int dimensions, ...)</code>	This function allocates memory for the dimensions of a tensor. It takes the number of dimensions and their values as input and returns a pointer to a dynamically allocated array of dimension objects.
<code>openctpu_buffer_t *openctpu_create_buffer(openctpu_dimension *dimension, void *data, unsigned flags)</code>	This function creates a buffer object from raw data. It takes the dimension of the tensor, the raw data pointer, and creation flags as input and returns a pointer to the buffer object.
<code>int *openctpu_enqueue(void *(func) (void *), ...)</code>	This function enqueues a function call to the TPU. It takes a function pointer and other parameters as input and returns an enqueue handle.
<code>int *openctpu_invoke_operator(enum tpu_ops op, unsigned flags, ...)</code>	This function invokes a TPU operator. It takes the operator type, flags, and other parameters as input and returns an invoke handle.
<code>int *openctpu_sync()</code>	This function synchronizes all TPU operations. It returns a handle to the sync operation.
<code>int *openctpu_wait(int task_id)</code>	This function waits for a specific task to complete. It takes the task ID and returns a handle to the wait operation.

Type	Operator
Matrix-wise	<code>conv2D</code>
Matrix-vector	<code>FullyConnected</code>
Pair-wise	<code>sub</code> <code>add</code> <code>mul</code> <code>crop</code> <code>ext</code> <code>mean</code> <code>max</code> <code>tanh</code> <code>relu</code>
Single Matrix	<code>CTPU</code>

```
#include <stdio.h>
#include <stdlib.h>
#include <gptpu.h>

// The TPU kernel
void *kernel(openctpu_buffer *matrix_a,
             openctpu_buffer *matrix_b,
             openctpu_buffer *matrix_c)
{
    // invoke the TPU operator
    openctpu_invoke_operator(conv2D, SCALE, matrix_a, \
                             matrix_b, matrix_c);
    return 0;
}

int main(int argc, char **argv)
{
    float *a, *b, *c; // pointers for raw data
    openctpu_dimension *matrix_a_d, *matrix_b_d, *matrix_c_d;
    openctpu_buffer *tensor_a, *tensor_b, *tensor_c;
    int size; // size of each dimension

    // skip: data I/O and memory allocation/initialization

    // describe a 2-D tensor (matrix) object for a
    matrix_a_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for b
    matrix_b_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for c
    matrix_c_d = openctpu_alloc_dimension(2, size, size);

    // create/fill the tensor a from the raw data
    tensor_a = openctpu_create_buffer(matrix_a_d, a);
    // create/fill the tensor b from the raw data
    tensor_b = openctpu_create_buffer(matrix_b_d, b);
    // create/fill the tensor c from the raw data
    tensor_c = openctpu_create_buffer(matrix_c_d, c);

    // enqueue the matrix_mul TPU kernel
    openctpu_enqueue(kernel, tensor_a, tensor_b, tensor_c);
    // synchronize/wait for all TPU kernels to complete
    openctpu_sync();

    // skip: the rest of the program
    return 0;
}
```

An overview of GPTPU

Programming frontend

C/C++ program code with OpenCtpu extensions

OpenCtpu-compatible compiler

Host Program Binary

System Software

GPTPU System/API Library

GPTPU Runtime

Tensorizer

Edge TPU Kernel Modules/Drivers

Standard Library

Kernel Modules/
Drivers

Libraries for other accelerators
(e.g. GPUs)

Accelerator Runtime (e.g.,
CUDA/OpenCL)

AcceleratorDriver/Modules

Hardware



Our GPTPU framework

```
#include <stdio.h>
#include <stdlib.h>
#include <gptpu.h>

// The TPU kernel
void *kernel(openctpu_buffer *matrix_a,
             openctpu_buffer *matrix_b,
             openctpu_buffer *matrix_c)
{
    // invoke the TPU operator
    openctpu_invoke_operator(conv2D, SCALE, matrix_a, \
                            matrix_b, matrix_c);
    return 0;
}

int main(int argc, char **argv)
{
    float *a, *b, *c; // pointers for raw data
    openctpu_dimension *matrix_a_d, *matrix_b_d, *matrix_c_d;
    openctpu_buffer * tensor_a, * tensor_b, * tensor_c;
    int size; // size of each dimension

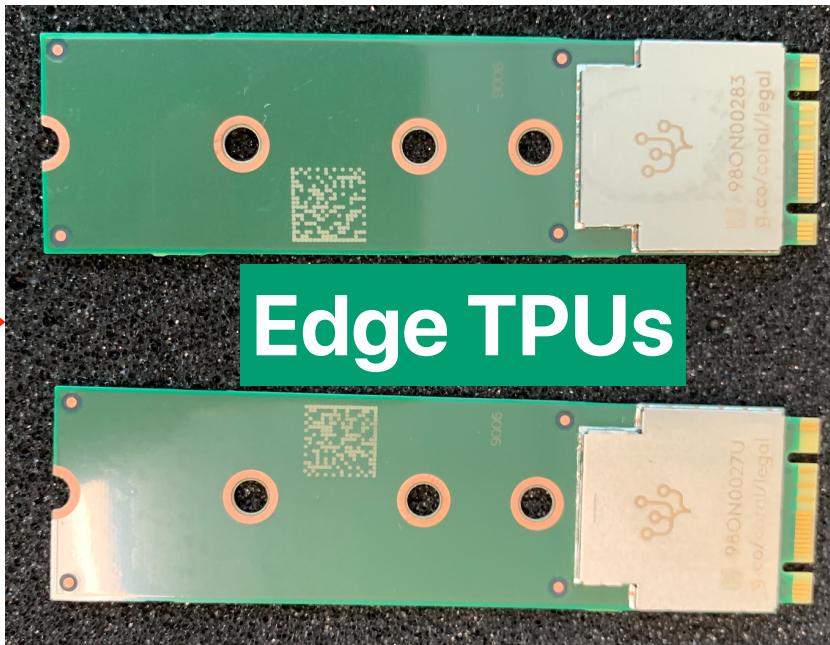
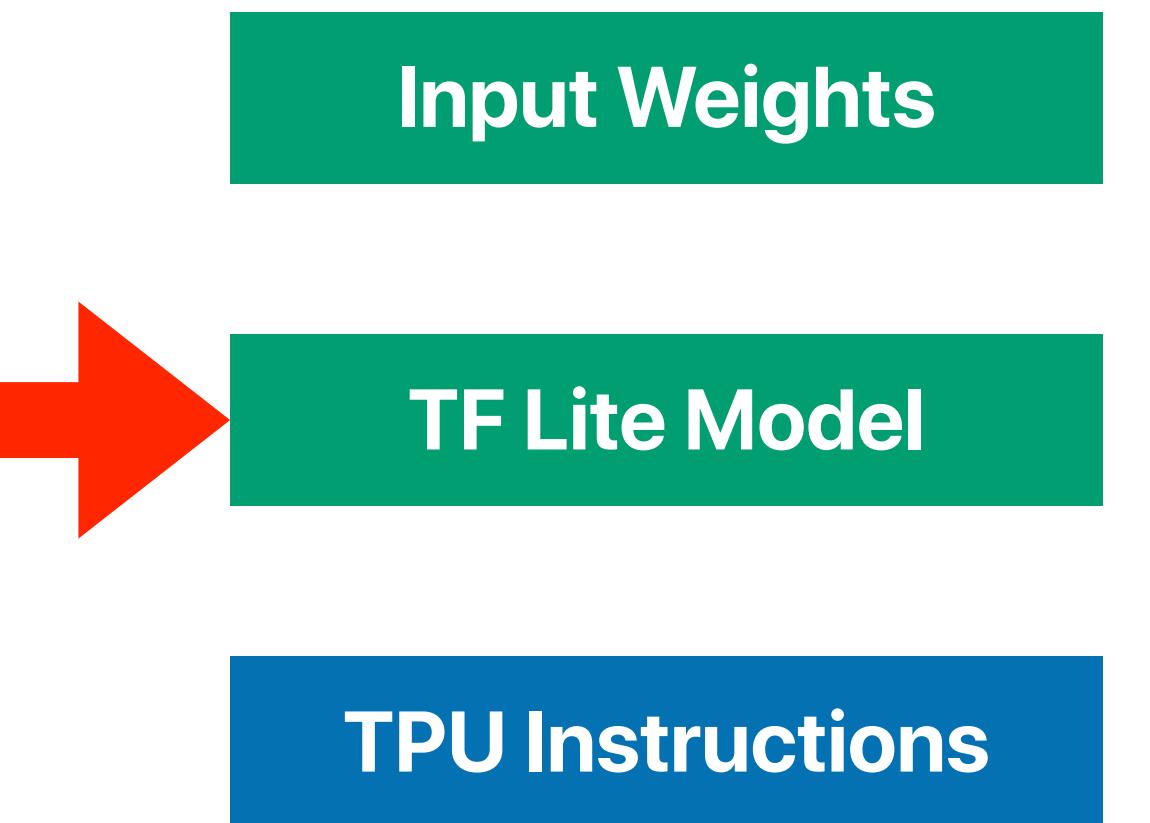
    // skip: data I/O and memory allocation/initialization

    // describe a 2-D tensor (matrix) object for a
    matrix_a_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for b
    matrix_b_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for c
    matrix_c_d = openctpu_alloc_dimension(2, size, size);

    // create/fill the tensor a from the raw data
    tensor_a = openctpu_create_buffer(matrix_a_d, a);
    // create/fill the tensor b from the raw data
    tensor_b = openctpu_create_buffer(matrix_b_d, b);
    // create/fill the tensor c from the raw data
    tensor_c = openctpu_create_buffer(matrix_c_d, c);

    // enqueue the matrix_mul TPU kernel
    openctpu_enqueue(kernel, tensor_a, tensor_b, tensor_c);
    // synchronize/wait for all TPU kernels to complete
    openctpu_sync();

    // skip: the rest of the program
    return 0;
}
```



The performance of each operator

Type	Operator	Throughput (op/s)	Input/Output Type
Matrix-wise	conv2D	160.45	2 matrices/1 matrix
Matrix-vector	FullyConnected	12,981.24	1 matrix, 1 vector/1 vector
Pair-wise	sub	98.02	2 matrices/1 matrix
	add	96.93	2 matrices/1 matrix
	mul	226.81	2 matrices/1 matrix
Single Matrix	crop	4,867.96	1 matrix/1 matrix
	ext	1,604.78	1 matrix/1 matrix
	mean	408.54	1 matrix/1 number
	max	477.08	1 matrix/1 vector
	tanh	3,232.31	1 matrix/1 matrix
	relu	11,194.26	1 matrix/1 matrix

GPTPU programming interface

Synopsis	Definition
<code>openctpu_dimension *openctpu_alloc_dimension(int dimensions, ...)</code>	This function allocates memory for an array of dimensions. It takes the number of dimensions and their values as arguments.
<code>openctpu_buffer_t *openctpu_create_buffer(openctpu_dimension *dimension, void *data, unsigned flags)</code>	This function creates a buffer on the TPU. It takes the dimension information and raw data as parameters.
<code>int *openctpu_enqueue(void *(func) (void *), ...)</code>	This function enqueues a task onto the TPU. It takes a function pointer and other parameters.
<code>int *openctpu_invoke_operator(enum tpu_ops op, unsigned flags, ...)</code>	This function invokes a TPU operator. It takes the operator type, flags, and other parameters.
<code>int *openctpu_sync()</code>	This function synchronizes all TPU tasks.
<code>int *openctpu_wait(int task_id)</code>	This function waits for a specific task to complete.

Type	Operator
Matrix-wise	<code>conv2D</code>
Matrix-vector	<code>FullyConnected</code>
Pair-wise	<code>sub</code> <code>add</code> <code>mul</code> <code>crop</code> <code>ext</code>
Single Matrix	<code>mean</code> <code>max</code> <code>tanh</code> <code>relu</code>

```
#include <stdio.h>
#include <stdlib.h>
#include <gptpu.h>

// The TPU kernel
void *kernel(openctpu_buffer *matrix_a,
             openctpu_buffer *matrix_b,
             openctpu_buffer *matrix_c)
{
    // invoke the TPU operator
    openctpu_invoke_operator(conv2D, SCALE, matrix_a, \
                             matrix_b, matrix_c);
    return 0;
}

int main(int argc, char **argv)
{
    float *a, *b, *c; // pointers for raw data
    openctpu_dimension *matrix_a_d, *matrix_b_d, *matrix_c_d;
    openctpu_buffer * tensor_a, * tensor_b, * tensor_c;
    int size; // size of each dimension

    // skip: data I/O and memory allocation/initialization

    // describe a 2-D tensor (matrix) object for a
    matrix_a_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for b
    matrix_b_d = openctpu_alloc_dimension(2, size, size);
    // describe a 2-D tensor (matrix) object for c
    matrix_c_d = openctpu_alloc_dimension(2, size, size);

    // create/fill the tensor a from the raw data
    tensor_a = openctpu_create_buffer(matrix_a_d, a);
    // create/fill the tensor b from the raw data
    tensor_b = openctpu_create_buffer(matrix_b_d, b);
    // create/fill the tensor c from the raw data
    tensor_c = openctpu_create_buffer(matrix_c_d, c);

    // enqueue the matrix_mul TPU kernel
    openctpu_enqueue(kernel, tensor_a, tensor_b, tensor_c);
    // synchronize/wait for all TPU kernels to complete
    openctpu_sync();

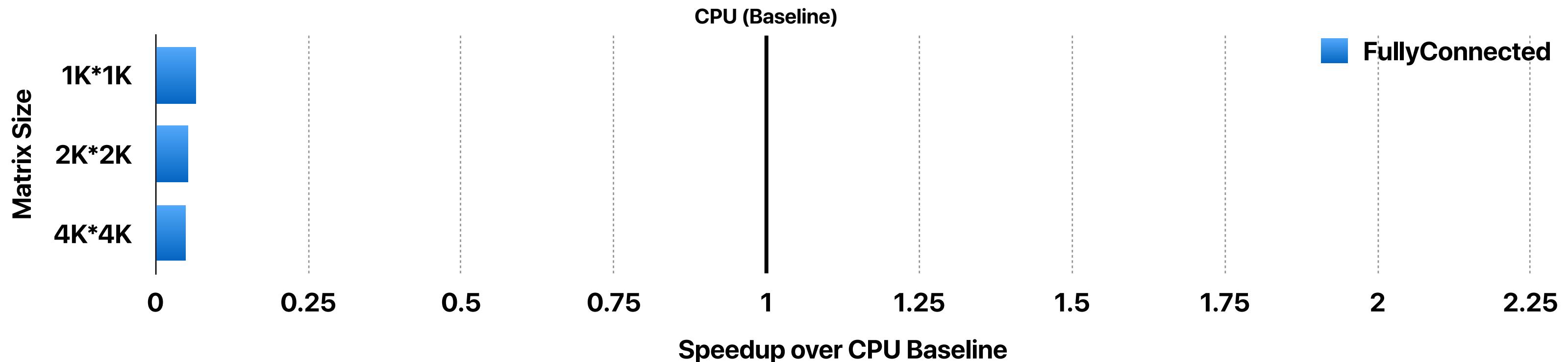
    // skip: the rest of the program
    return 0;
}
```

MM using FullyConnected

$$\begin{array}{ccccccccc}
 (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) & (0,7) & (0,8) \\
 (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) & (1,7) & (1,8) \\
 (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) & (2,7) & (2,8) \\
 (3,0) & (3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) & (3,7) & (3,8) \\
 (4,0) & (4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) & (4,7) & (4,8) \\
 (5,0) & (5,1) & (5,2) & (5,3) & (5,4) & (5,5) & (5,6) & (5,7) & (5,8) \\
 (6,0) & (6,1) & (6,2) & (6,3) & (6,4) & (6,5) & (6,6) & (6,7) & (6,8) \\
 (7,0) & (7,1) & (7,2) & (7,3) & (7,4) & (7,5) & (7,6) & (7,7) & (7,8) \\
 (8,0) & (8,1) & (8,2) & (8,3) & (8,4) & (8,5) & (8,6) & (8,7) & (8,8)
 \end{array} \times
 \begin{array}{ccccccccc}
 (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) & (0,7) & (0,8) \\
 (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) & (1,7) & (1,8) \\
 (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) & (2,7) & (2,8) \\
 (3,0) & (3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) & (3,7) & (3,8) \\
 (4,0) & (4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) & (4,7) & (4,8) \\
 (5,0) & (5,1) & (5,2) & (5,3) & (5,4) & (5,5) & (5,6) & (5,7) & (5,8) \\
 (6,0) & (6,1) & (6,2) & (6,3) & (6,4) & (6,5) & (6,6) & (6,7) & (6,8) \\
 (7,0) & (7,1) & (7,2) & (7,3) & (7,4) & (7,5) & (7,6) & (7,7) & (7,8) \\
 (8,0) & (8,1) & (8,2) & (8,3) & (8,4) & (8,5) & (8,6) & (8,7) & (8,8)
 \end{array} =
 \begin{array}{ccccccccc}
 (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) & (0,7) & (0,8) \\
 (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) & (1,7) & (1,8) \\
 (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) & (2,7) & (2,8) \\
 (3,0) & (3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) & (3,7) & (3,8) \\
 (4,0) & (4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) & (4,7) & (4,8) \\
 (5,0) & (5,1) & (5,2) & (5,3) & (5,4) & (5,5) & (5,6) & (5,7) & (5,8) \\
 (6,0) & (6,1) & (6,2) & (6,3) & (6,4) & (6,5) & (6,6) & (6,7) & (6,8) \\
 (7,0) & (7,1) & (7,2) & (7,3) & (7,4) & (7,5) & (7,6) & (7,7) & (7,8) \\
 (8,0) & (8,1) & (8,2) & (8,3) & (8,4) & (8,5) & (8,6) & (8,7) & (8,8)
 \end{array}$$

For square matrices with M columns M rows, takes M FullyConnected operations

On the real machine



Edge TPU performance

Table 1. Time per inference, in milliseconds (ms)

Model architecture	Desktop CPU ¹	Desktop CPU ¹ + USB Accelerator (USB 3.0) <i>with Edge TPU</i>	Embedded CPU ²	Dev Board ³ <i>with Edge TPU</i>
Unet Mv2 (128x128)	27.7	3.3	190.7	5.7
DeepLab V3 (513x513)	394	52	1139	241
DenseNet (224x224)	380	20	1032	25
Inception v1 (224x224)	90	3.4	392	4.1
Inception v4 (299x299)	700	85	3157	102
Inception-ResNet V2 (299x299)	753	57	2852	69
MobileNet v1 (224x224)	53	2.4	161	2.4
MobileNet v2 (224x224)	51	2.6	122	2.6
MobileNet v1 SSD	109	6.5	353	11

Does not make too much sense for general purpose computing

The performance of each operator

Type	Operator	Throughput (op/s)	Input/Output Type	Goodput (numbers/s)
Matrix-wise	conv2D	160.45	2 matrices/1 matrix	168,240,326.89
Matrix-vector	FullyConnected	12,981.24	1 matrix, 1 vector/1 vector	6,646,394.57
Pair-wise	sub	Convolution 2D is way more optimized than FullyConnected ³⁵		
	add	96.93	2 matrices/1 matrix	101,642,520.95
	mul	226.81	2 matrices/1 matrix	237,829,999.97
	crop	4,867.96	1 matrix/1 matrix	1,562,904,391.76
	ext	1,604.78	1 matrix/1 matrix	3,637,240,203.38
	mean	408.54	1 matrix/1 number	408.54
	max	477.08	1 matrix/1 vector	477.08
Single Matrix	tanh	3,232.31	1 matrix/1 matrix	3,389,324,502.67
	relu	11,194.26	1 matrix/1 matrix	11,738,029,769.15

Edge TPUs' "approximate" conv2D

Input: A

• (0,0)	• (0,1)	• (0,2)	• (0,3)	• (0,4)	• (0,5)
• (1,0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
• (2,	(2, 1)	(2, 2)	(2,	(2, 4)	(2, 5)
• (3,	(3, 1)	(3, 2)	(3,	(3, 4)	(3,
• (4,	(4, 1)	(4,	(4,	(4,	(4,
• (5,	(5, 1)	(5, 2)	(5,	(5,	(5,

size: $L*L$



**Kernel: B
size: $L*L$**

(0,0)	(0,1)	(0,2)
(0,3)	(0,4)	(0,5)
(0,6)	(0,7)	(0,8)

Result: C

$$C_{i,j} = \sum_{q=0}^L \sum_{p=0}^L A_{i \times L + p, j \times L + q} \cdot B'_{p,q}$$



(0,0)	(0,1)
(1,0)	(1,1)

(0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2, (2, 1) (2, 2)



(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
-------	-------	-------	-------	-------	-------	-------	-------	-------

(0,0)

(0,3) (0,4) (0,5) (1,3) (1,4) (1,5) (2, (2, 1) (2, 5)



(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
-------	-------	-------	-------	-------	-------	-------	-------	-------

(0,1)

(3, (3, 1) (3, 2) (4, (4, 1) (4, 2) (5, (5, 1) (5, 2)



(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
-------	-------	-------	-------	-------	-------	-------	-------	-------

(1,0)

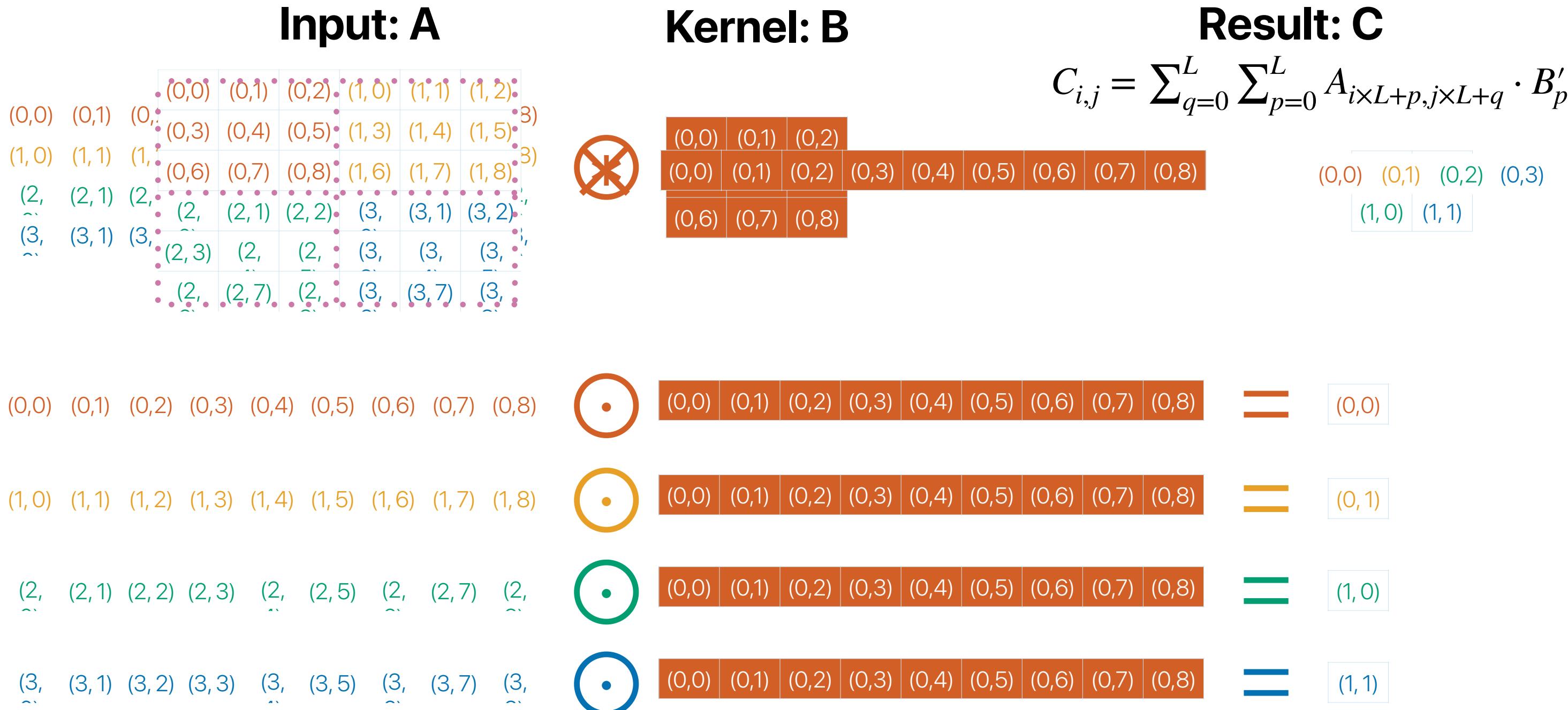
(3,3) (3, (3, 5) (4, (4, 1) (4, 5, 3) (5, (5, 1) (5, 5)



(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
-------	-------	-------	-------	-------	-------	-------	-------	-------

(1,1)

Edge TPUs' "approximate" conv2D

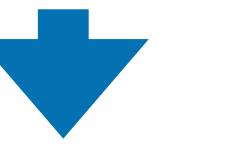


Map MM into conv2D

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	(7,8)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)	(8,8)

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	(7,8)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)	(8,8)

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)
(7,0)	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	(7,8)
(8,0)	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)	(8,8)



(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
(0,3)	(0,4)	(0,5)	(1,3)	(1,4)	(1,5)	(2,3)	(2,4)	(2,5)
(0,6)	(0,7)	(0,8)	(1,6)	(1,7)	(1,8)	(2,6)	(2,7)	(2,8)
(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)
(1,4)	(1,5)	(1,6)	(2,4)	(2,5)	(2,6)	(3,4)	(3,5)	(3,6)
(1,7)	(1,8)	(1,9)	(2,7)	(2,8)	(2,9)	(3,7)	(3,8)	(3,9)
(2,0)	(2,1)	(2,2)	(3,0)	(3,1)	(3,2)	(4,0)	(4,1)	(4,2)
(2,3)	(2,4)	(2,5)	(3,3)	(3,4)	(3,5)	(4,3)	(4,4)	(4,5)
(2,6)	(2,7)	(2,8)	(3,6)	(3,7)	(3,8)	(4,6)	(4,7)	(4,8)

(0,0)	(3,0)	(6,0)	(0,3)	(3,3)	(6,3)	(0,6)	(3,6)	(6,6)
(1,0)	(4,0)	(7,0)	(1,3)	(4,3)	(7,3)	(1,6)	(4,6)	(7,6)
(2,0)	(5,0)	(8,0)	(2,3)	(5,3)	(8,3)	(2,6)	(5,6)	(8,6)
(3,0)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)
(3,3)	(3,4)	(3,5)	(4,4)	(4,5)	(5,4)	(5,5)	(6,4)	(6,5)
(3,6)	(3,7)	(3,8)	(4,7)	(4,8)	(5,7)	(5,8)	(6,7)	(6,8)
(4,0)	(1,1)	(4,1)	(7,1)	(1,4)	(4,4)	(7,4)	(1,7)	(4,7)
(4,3)	(1,4)	(4,4)	(7,4)	(1,7)	(4,7)	(7,7)	(1,10)	(4,10)
(4,6)	(2,1)	(5,1)	(8,1)	(2,4)	(5,4)	(8,4)	(2,7)	(5,7)

(0,0)	(0,3)	(0,6)	(1,0)	(1,3)	(1,6)	(2,0)	(2,3)	(2,6)
(0,1)	(0,4)	(0,7)	(1,1)	(1,4)	(1,7)	(2,1)	(2,4)	(2,7)
(0,2)	(0,5)	(0,8)	(1,2)	(1,5)	(1,8)	(2,2)	(2,5)	(2,8)
(0,5)	(3,3)	(3,6)	(1,5)	(4,1)	(4,4)	(2,5)	(4,4)	(5,3)
(0,8)	(3,1)	(3,4)	(3,7)	(4,1)	(4,7)	(5,1)	(5,7)	(5,4)
(1,2)	(3,2)	(3,5)	(4,2)	(4,5)	(5,2)	(6,2)	(4,5)	(5,2)
(1,5)	(6,1)	(6,4)	(7,3)	(7,6)	(8,1)	(7,0)	(7,3)	(7,6)
(1,8)	(6,2)	(6,5)	(7,4)	(7,7)	(8,2)	(7,1)	(7,4)	(7,7)
(2,1)	(6,3)	(6,6)	(7,1)	(7,4)	(8,3)	(7,0)	(7,3)	(7,6)



“Theoretical” performance of different MM implementations

Type	Operator	Throughput (op/s)	Input/Output Type	Goodput (numbers/s)
Matrix-wise	conv2D	160.45	2 matrices/1 matrix	168,240,326.89
Matrix-vector	FullyConnected	12981.24	1 matrix, 1 vector/1 vector	6,646,394.57

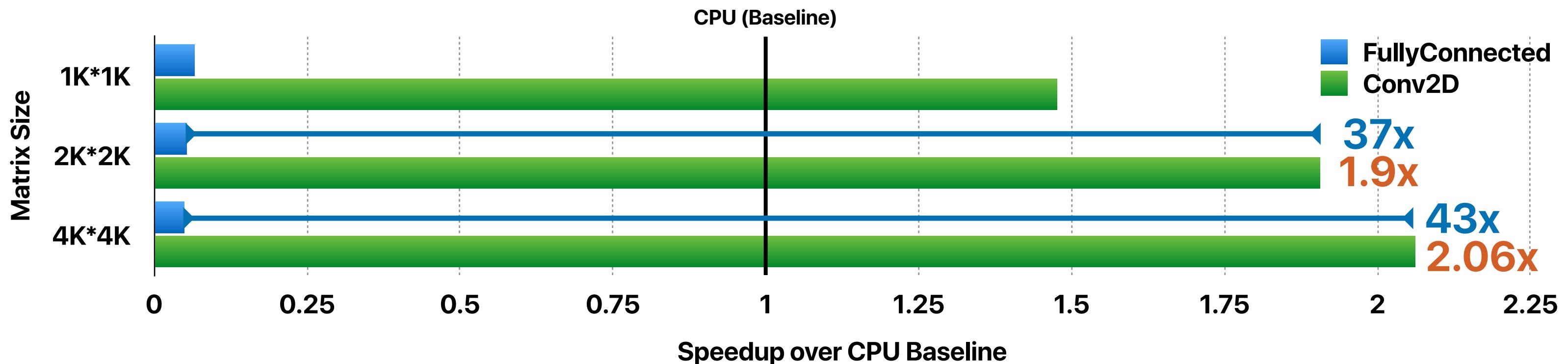
For square matrices with M columns M rows, takes M FullyConnected operations

$$\text{For } M=2048, \text{ it takes } 2048 \times \frac{1}{12981.24} = 0.15776613 \text{ Sec}$$

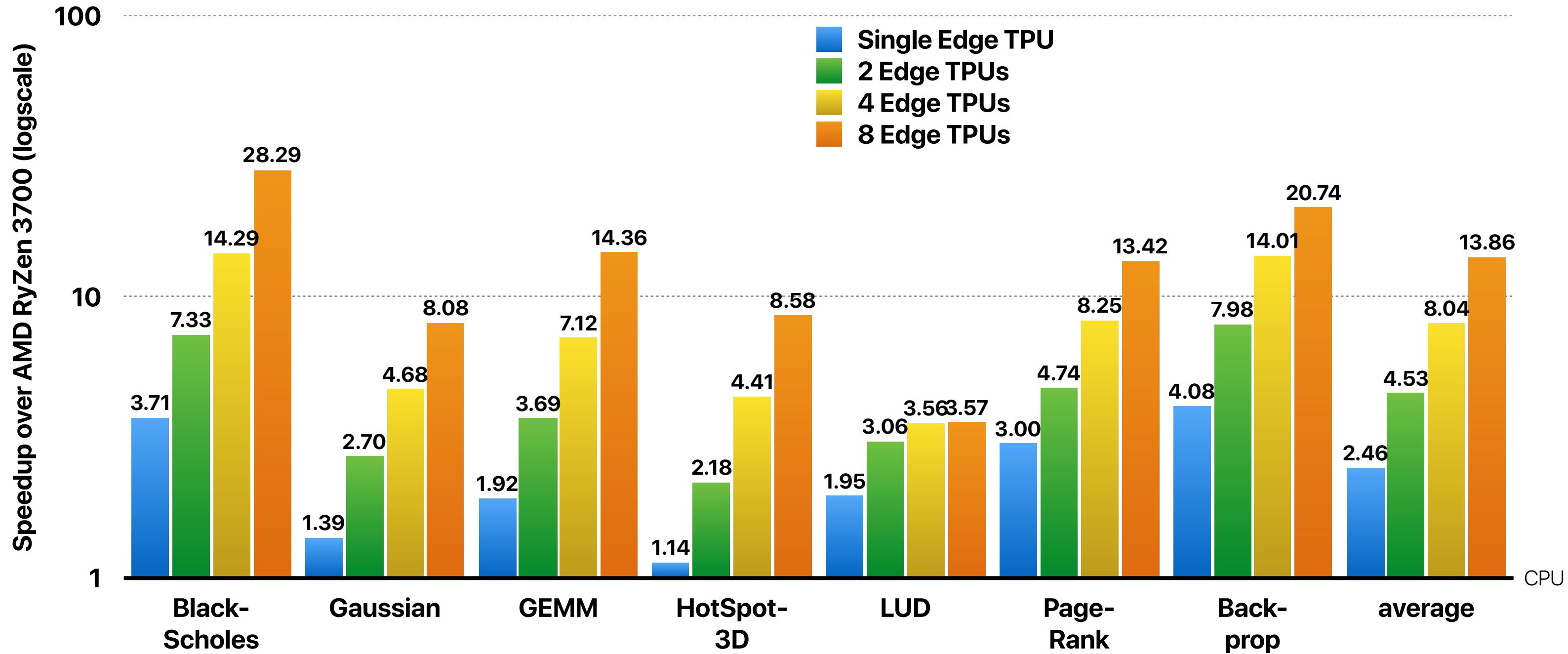
For square matrices with M columns M rows, takes 1 Conv2D operation if $M \leq 2048$

$$\text{For } M=2048, \text{ it takes } \frac{1}{160.45} = 0.00623247117 \text{ Sec}$$

On the real machine



Performance for general-purpose Workloads



Roogle Claimed ideas

- Setup and benchmarking SmartSSDs
- Parallel processing using matrix parallelism
- Benchmarking fault-tolerant mechanisms in machine learning
- Benchmarking and optimizing multimedia machine learning applications on heterogeneous computing resources
- Contact me if you've confirmed your project ideas but not seeing it in the slide — also send me your team member names
- Make an appointment through Google Calendar if you need consultant on projects

Roogle Other interesting ideas

- Can we use RT cores to accelerate other problems beyond identified ones?
- Can you improve the efficiency of existing problems on Tensor Cores (matrix decompositions, security algorithms)?
- Can you use other accelerated libraries (e.g., cuDNN, cuFFT) for applications beyond their domains?
- Can you think about some data representations that can make important compute kernels (e.g., machine learning algorithms, matrix multiplications) more efficient or secure?

Roogle Project Presentations

- Make an appointment on 2/1 and 2/6 through the Google Calendar
- 12 minute presentation with 3 minute Q & A
- Why & what & how!!! — considering you're giving a presentation at Apple's keynote
 - 6-minute why — why should everyone care about this problem? Why is this still a problem?
 - 4-minute what — what are you proposing in this project to address the problem?
 - 2-minute how — expected platforms/engineering efforts, milestones and workload distribution among members
 - Please reference this article to make a good presentation <https://cseweb.ucsd.edu/~swanson/GivingTalks.html>

Electrical Computer Science Engineering

277

つづく

