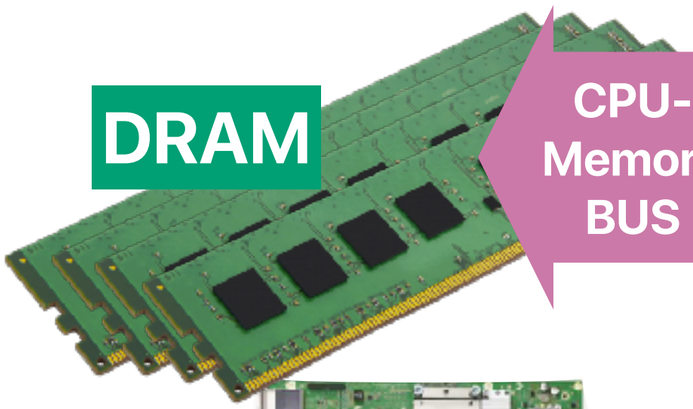
General purpose computing on hardware accelerators (2)

Hung-Wei Tseng

Recap: The landscape of modern computers



CPU-Memory BUS

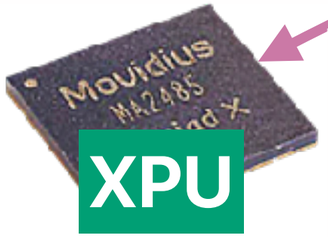
DDR4

Memory Controller

High-performance, but very specialized

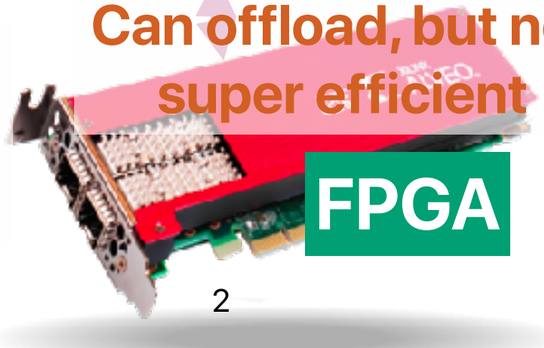
PCI EXPRESS

PCIe Root Complex



General-purpose for "regular" workloads, but can be power consuming

Can offload, but not super efficient



Recap: The concept of approximate computing on NPUs

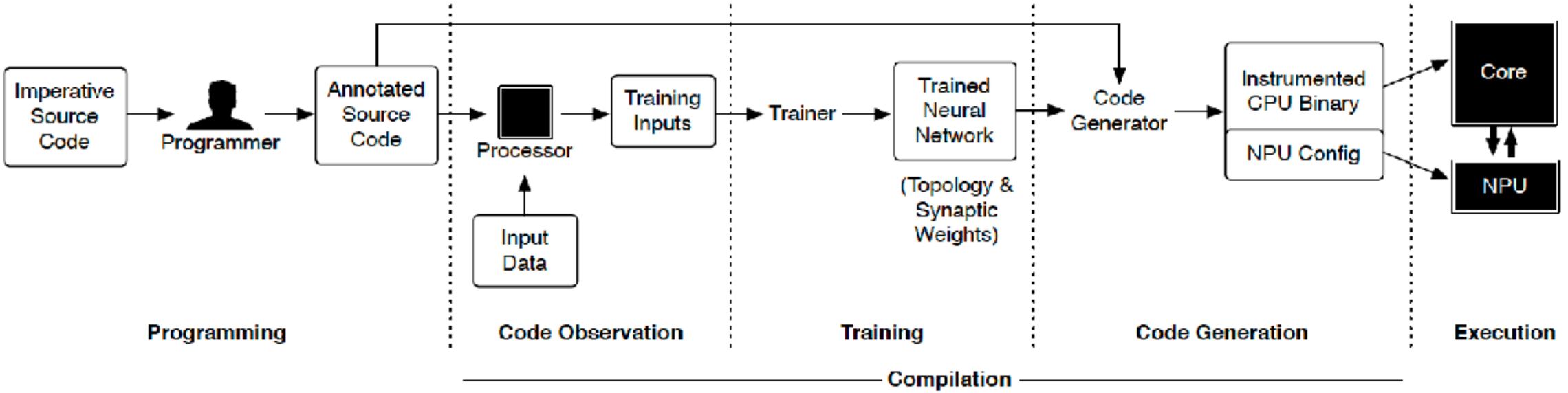
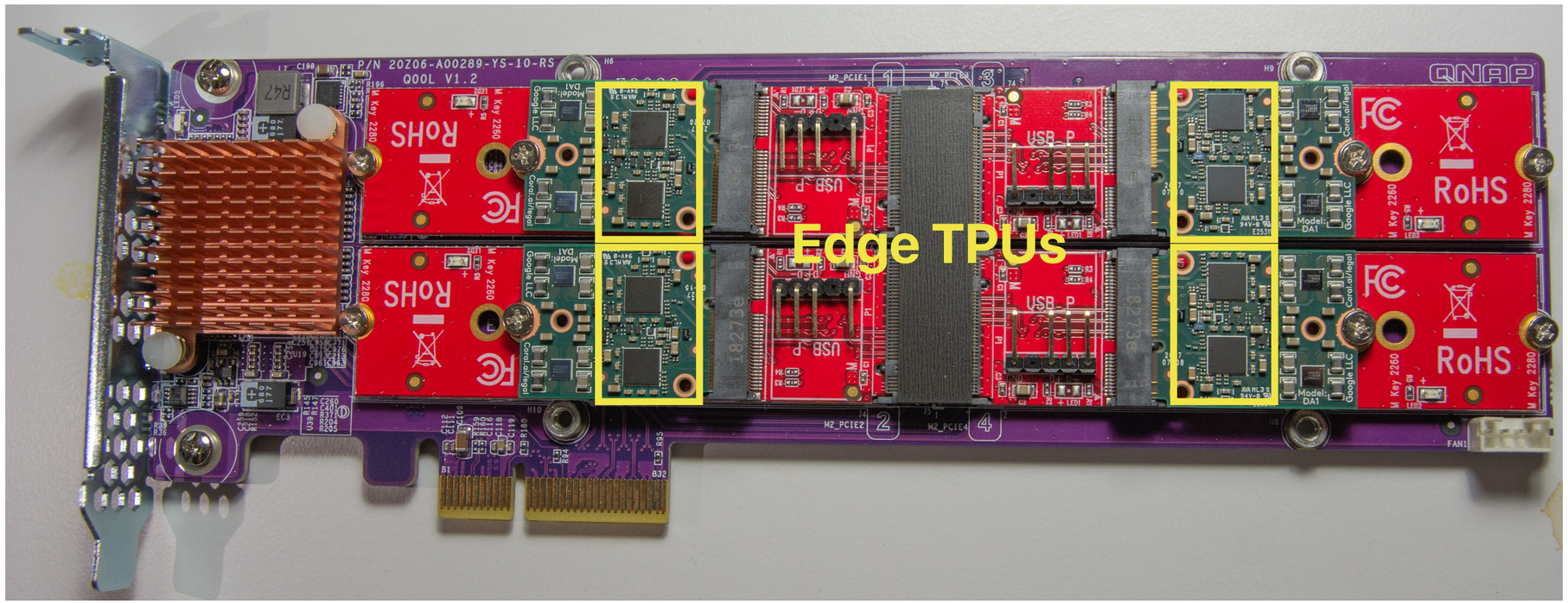


Figure 1: The Parrot transformation at a glance: from annotated code to accelerated execution on an NPU-augmented core.

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

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

Let’s try a simple problem — binary MM

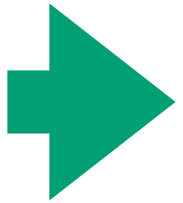
0	1	0	1
0	1	1	0
1	0	1	1
0	0	0	1

×

1	0
0	0
1	1
1	1

=

1	1
1	1
3	2
1	1



0	0
0	0
1	1
0	0

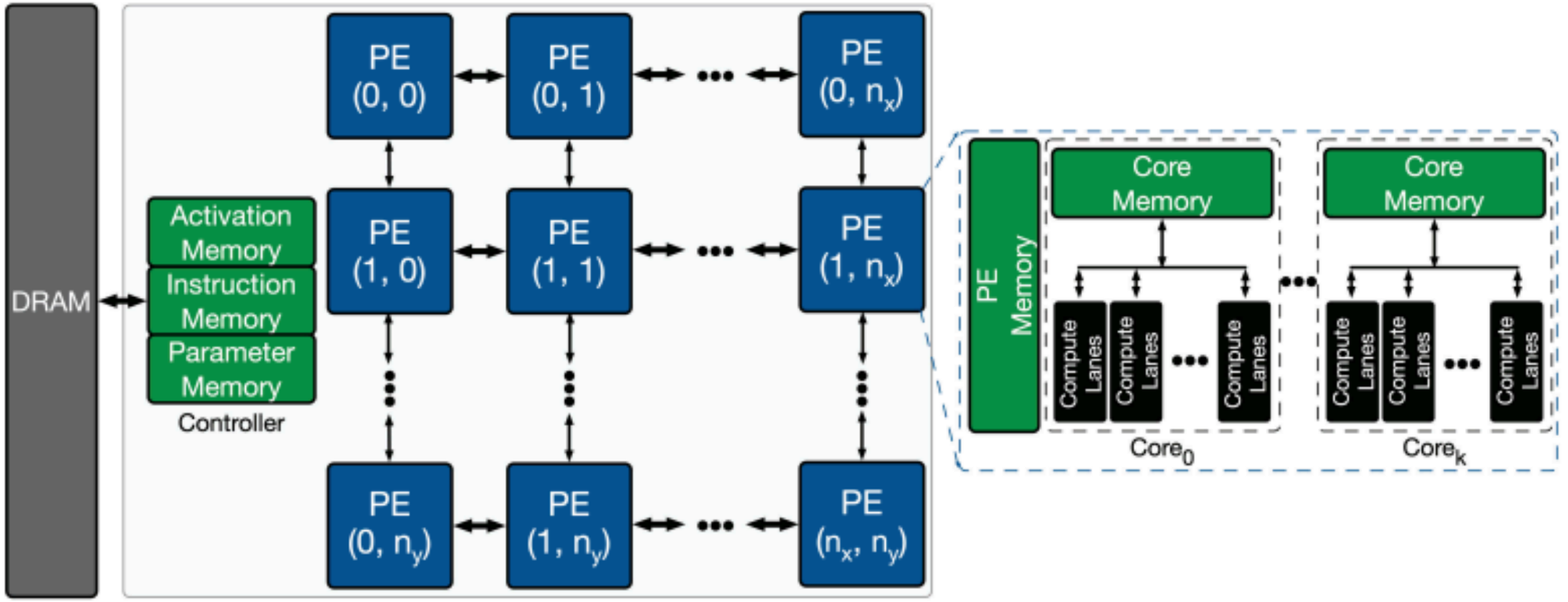
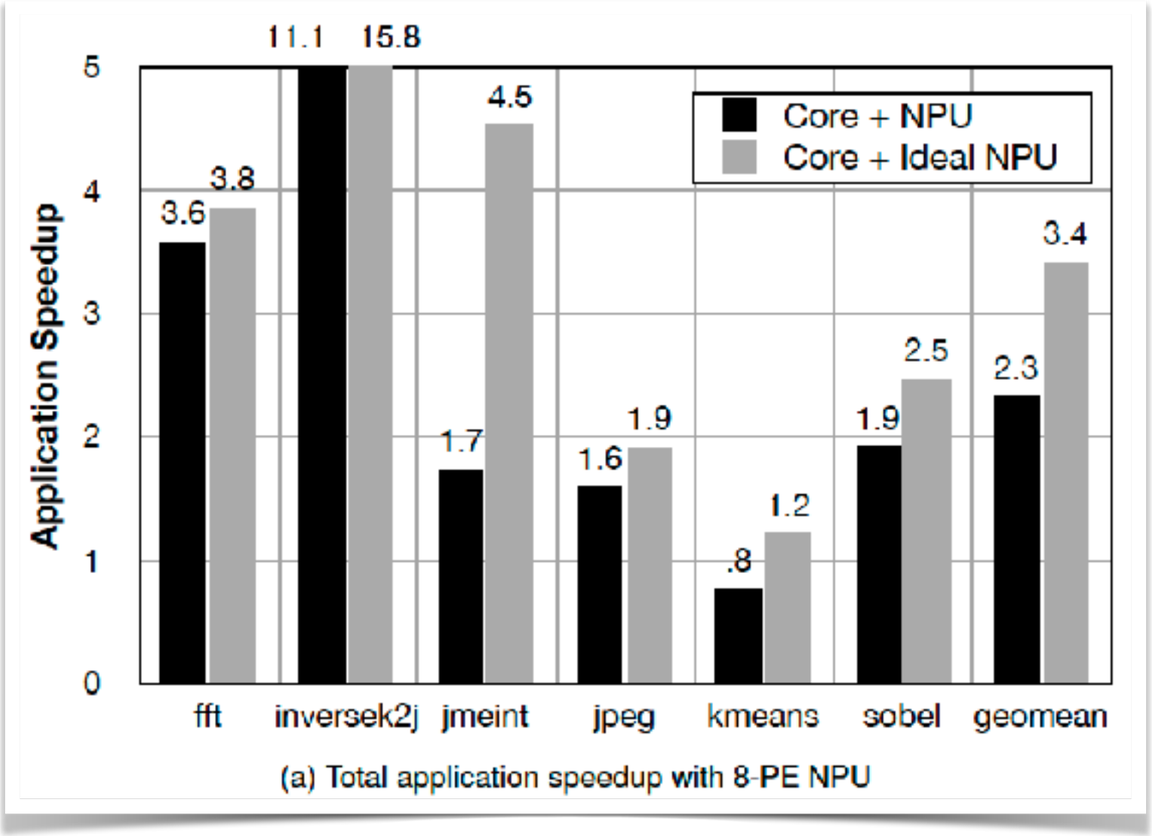


Figure 1: Overview of the template-based machine learning accelerator used for architecture exploration.

Ideas?

The idea works more efficiently if accelerator exists



(a) Total application speedup with 8-PE NPU

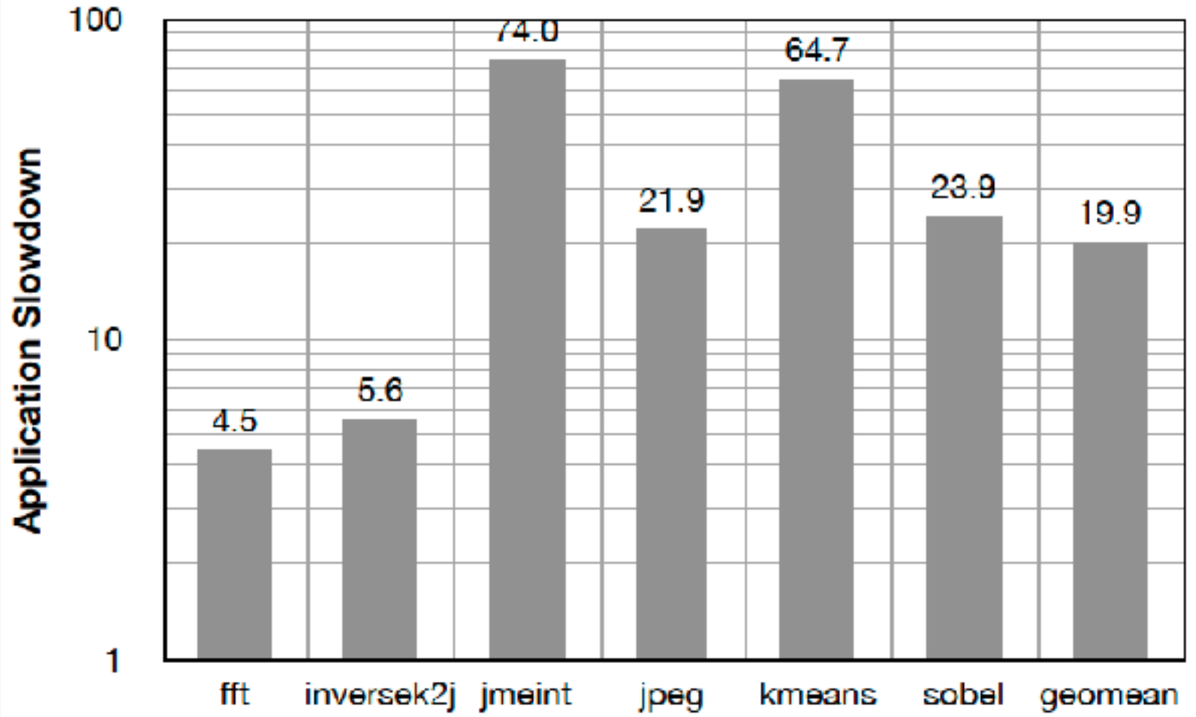
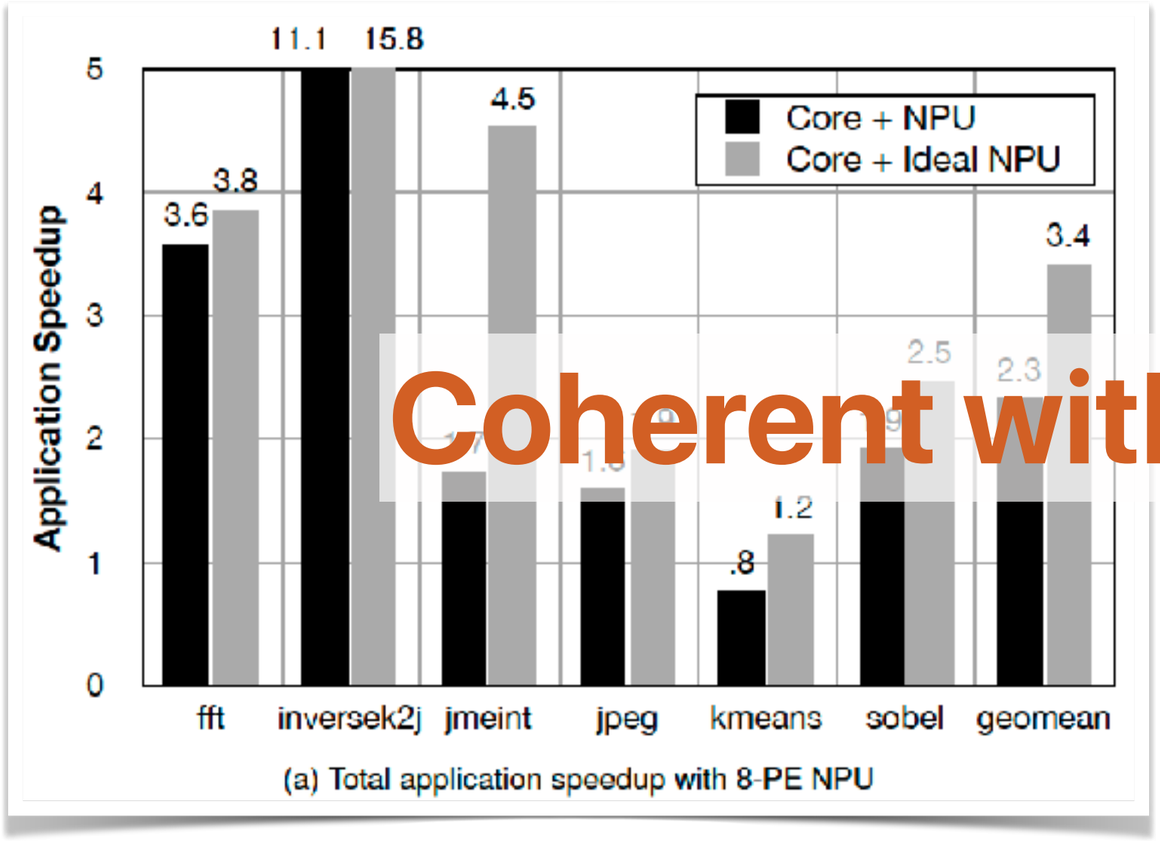


Figure 9: Slowdown with software neural network execution.

The idea works more efficiently if accelerator exists



(a) Total application speedup with 8-PE NPU

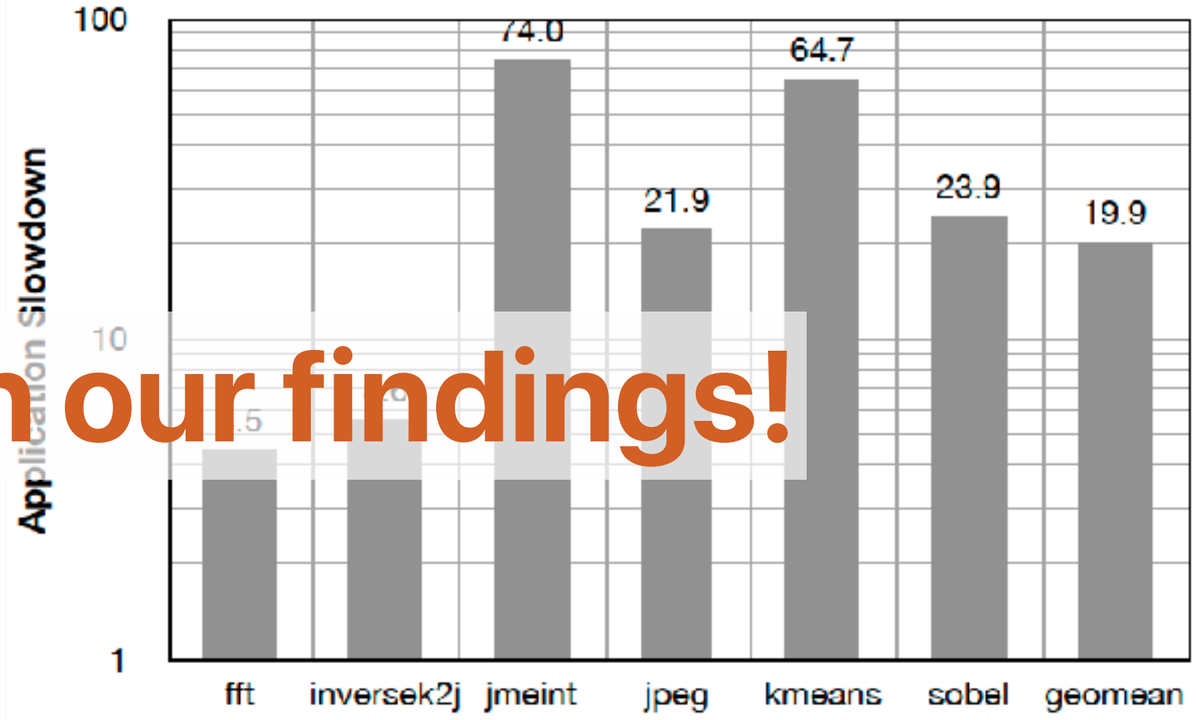


Figure 9: Slowdown with software neural network execution.

Coherent with our findings!

3:00

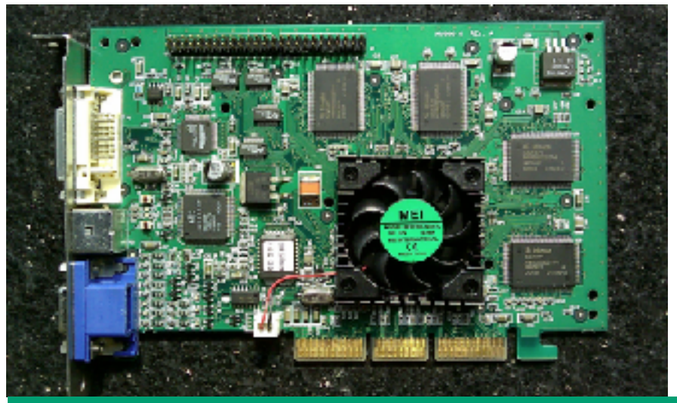
What’s the limitation of Approx. computing on NPUs?

3:00

Why do you think NVIDIA succeed in promoting GPGPU?

Why is NVIDIA successful?

Can we create another revolution with TPUs?



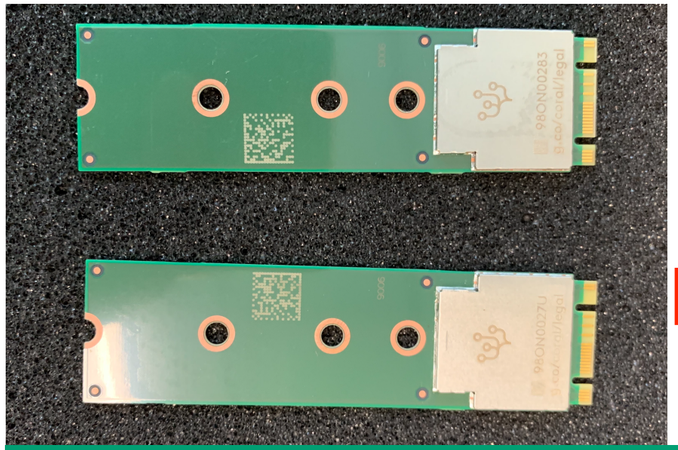
Graphics "Accelerator", 1999

```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}
```

- CUDA
- OpenCL

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

General Purpose Computing on GPUs



AI "Accelerator", 2013—

- Programing Language
- Compiler
- Runtime

?

General Purpose Computing on TPUs

Accelerating Applications using Edge Tensor Processing Units

Kuan-Chieh Hsu and Hung-Wei Tseng.
University of California, Riverside
In The International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2021.
https://github.com/escalab/GPTPU

Edge TPU Operators

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

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.
ExpandDims	≥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	≥13	No packing 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.
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	
ReLUN1To1	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 Slice op), and with ellipsis_axis_mask == 0, and new_axis_mask == 0.
Sub	All	
Sum	≥13	Cannot operate on the batch dimension.
Squared-difference	≥14	
Tanh	All	
Transpose	≥14	
TransposeConv	≥13	

Programming Interface

Synopsis	De
openctpu_dimension *openctpu_alloc_dimension(int dimensions, ...)	Thi inp tha
openctpu_buffer_t *openctpu_create_buffer(openctpu_dimension *dimension, void *data, unsigned flags)	Thi inf pro
int *openctpu_enqueue(void *(*func)(void *), ...)	Thi nu
int *openctpu_invoke_operator(enum tpu_ops op, unsigned flags, ...)	Thi fl
int *openctpu_sync()	Thi
int *openctpu_wait(int task_id)	Thi

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

```c
#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_a, * tensor_a;
    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

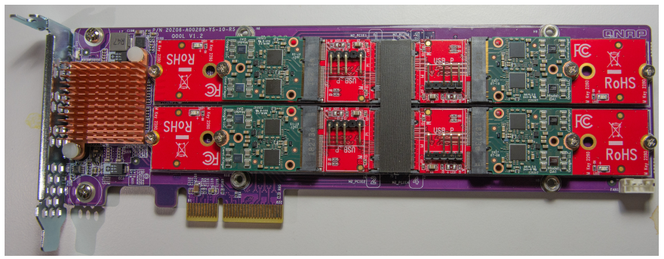
Kernel Modules/ Drivers

Libraries for other accelerators (e.g. GPUs)

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

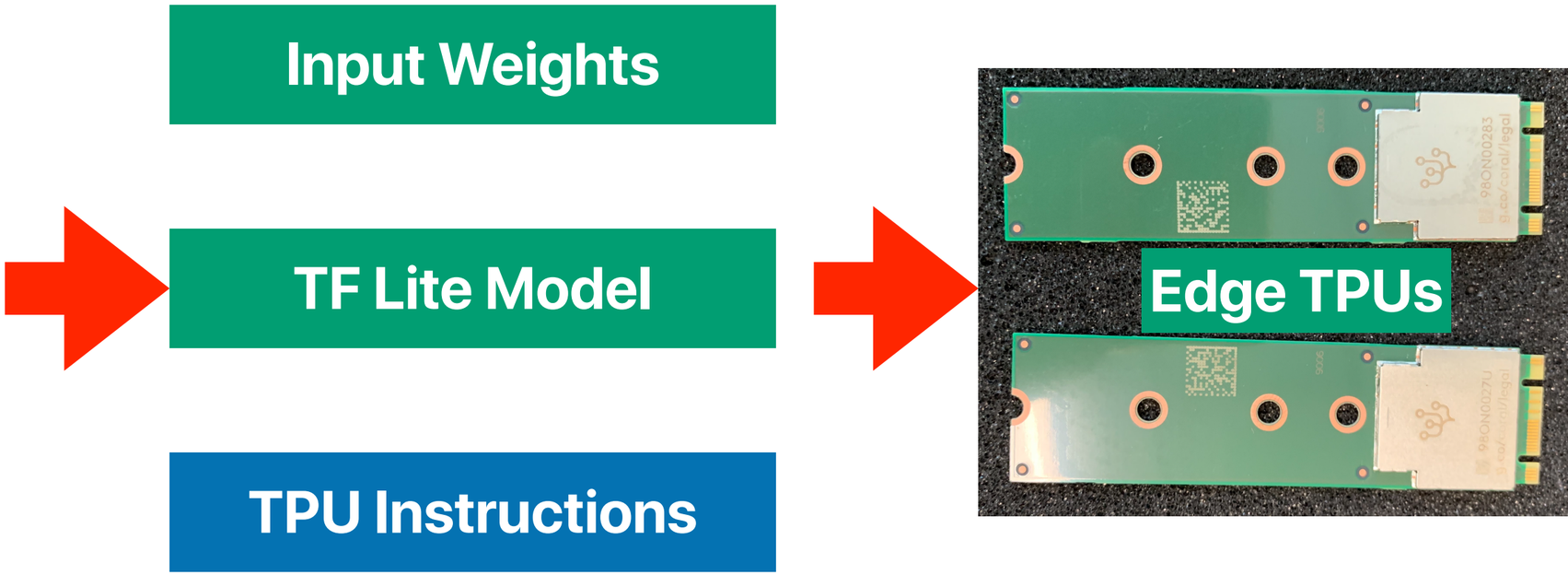
AcceleratorDriver/Modules

Hardware



Our GPTPU framework

```c
#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_a, * tensor_a;
    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

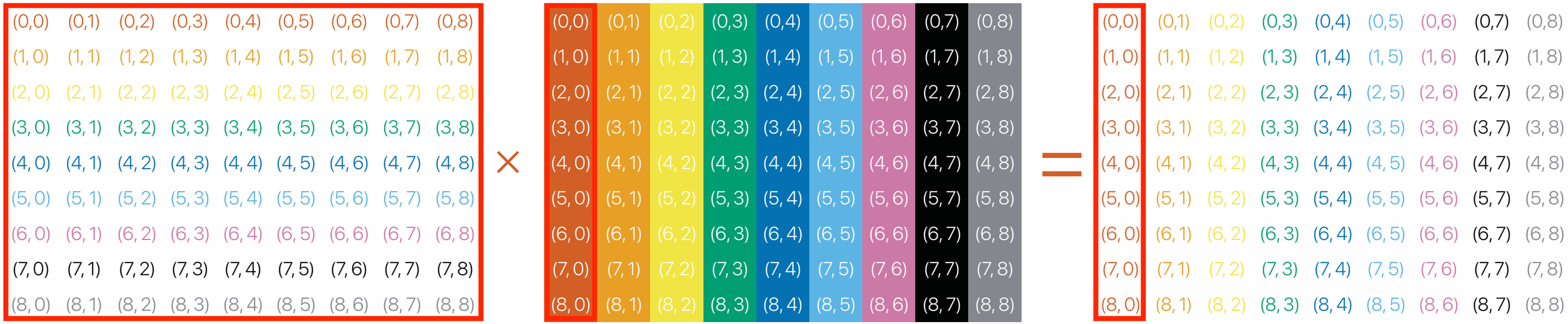
GPTPU programming interface

Synopsis	De
`openctpu_dimension *openctpu_alloc_dimension(int dimensions, ...)`	Thi inp tha
`openctpu_buffer_t *openctpu_create_buffer( openctpu_dimension *dimension, void *data, unsigned flags)`	Thi inf pro
`int *openctpu_enqueue(void *(*func)(void *), ...)`	Thi nu
`int *openctpu_invoke_operator(enum tpu_ops op, unsigned flags, ...)`	Thi fl
`int *openctpu_sync()`	Thi
`int *openctpu_wait(int task_id)`	Thi

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

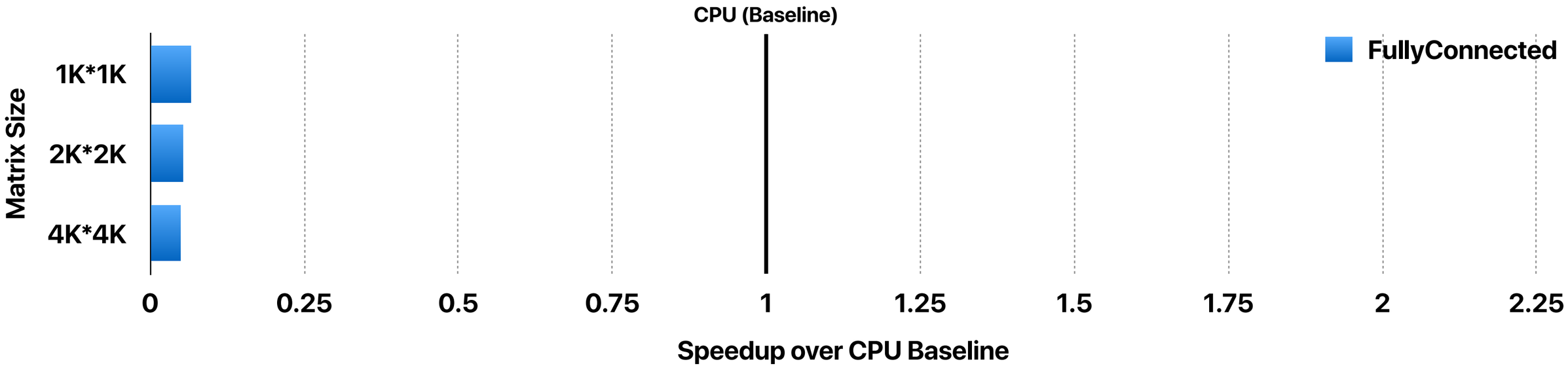
```c
#include <stdio.h>
#include <stdlib.h>
#include <gptpu.h>

// 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_a, * tensor_a;
    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



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) *with Edge TPU*	Embedded CPU [2]	Dev Board [3] *with Edge TPU*
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	164	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			35
	add	96.93	2 matrices/1 matrix	101,642,520.95
	mul	226.81	2 matrices/1 matrix	237,829,999.97
Single Matrix	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
	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

Convolution 2D is way more optimized than FullyConnected

Edge TPUs' "approximate" conv2D

Input: A

size: *L*L*

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

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

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

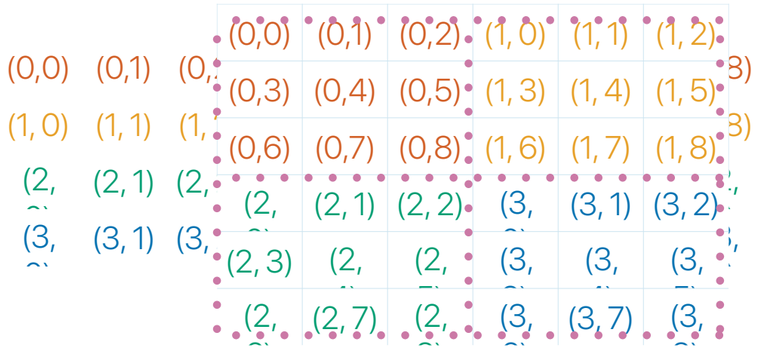
(0,0)	(0, 1)
(1, 0)	(1, 1)

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

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

Edge TPUs' "approximate" conv2D

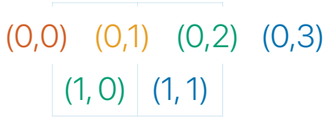
Input: A

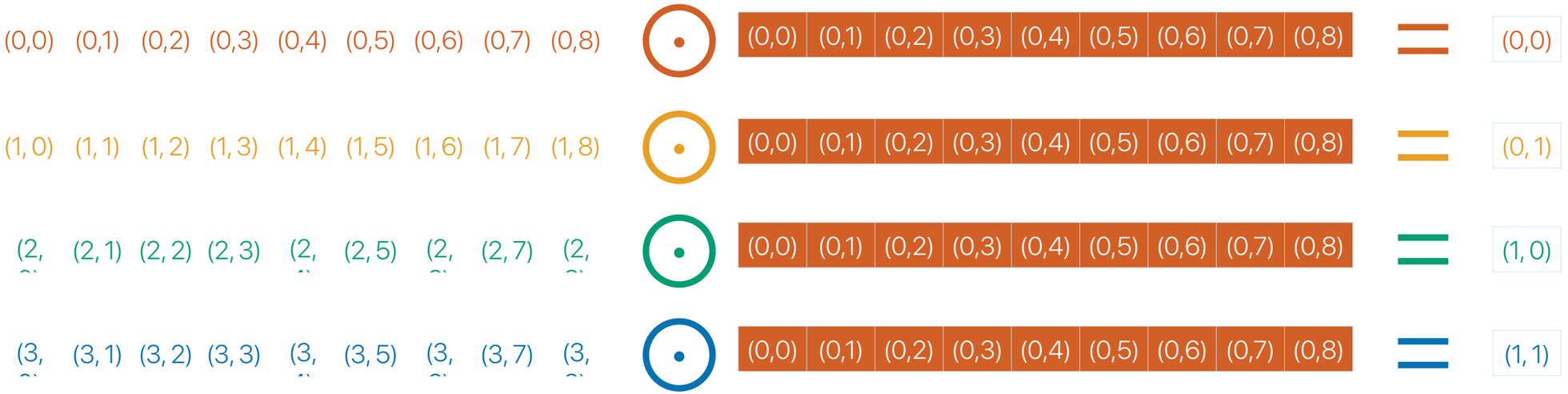


Kernel: B

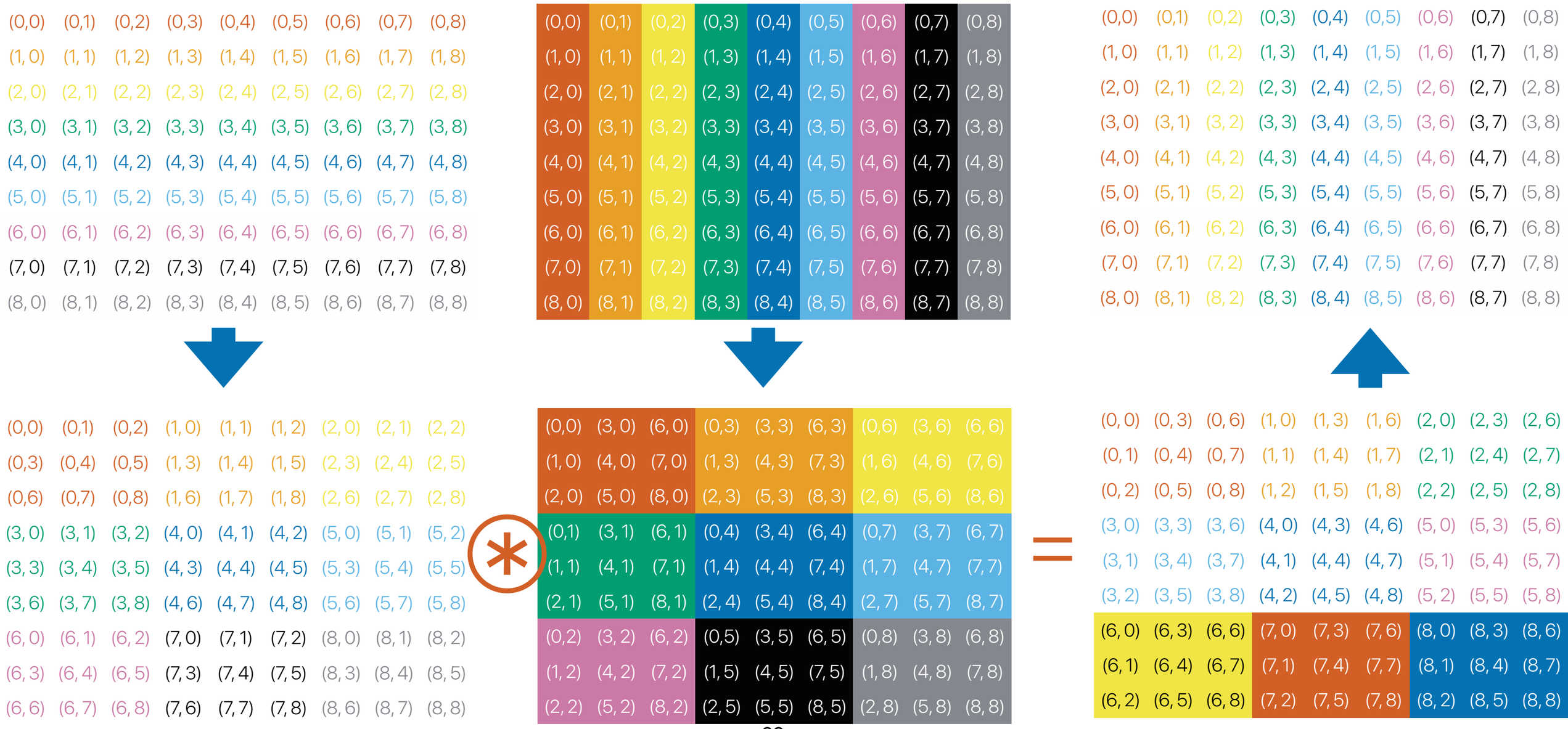
Result: C

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





Map MM into conv2D



"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

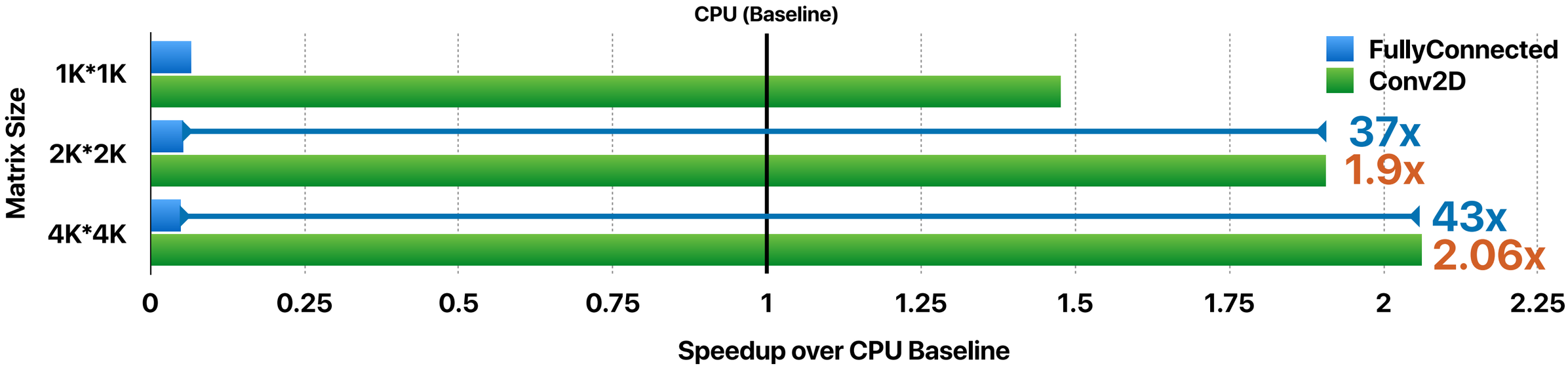
For square matrices with ***M*** columns ***M*** rows, takes ***M*** FullyConnected operations

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

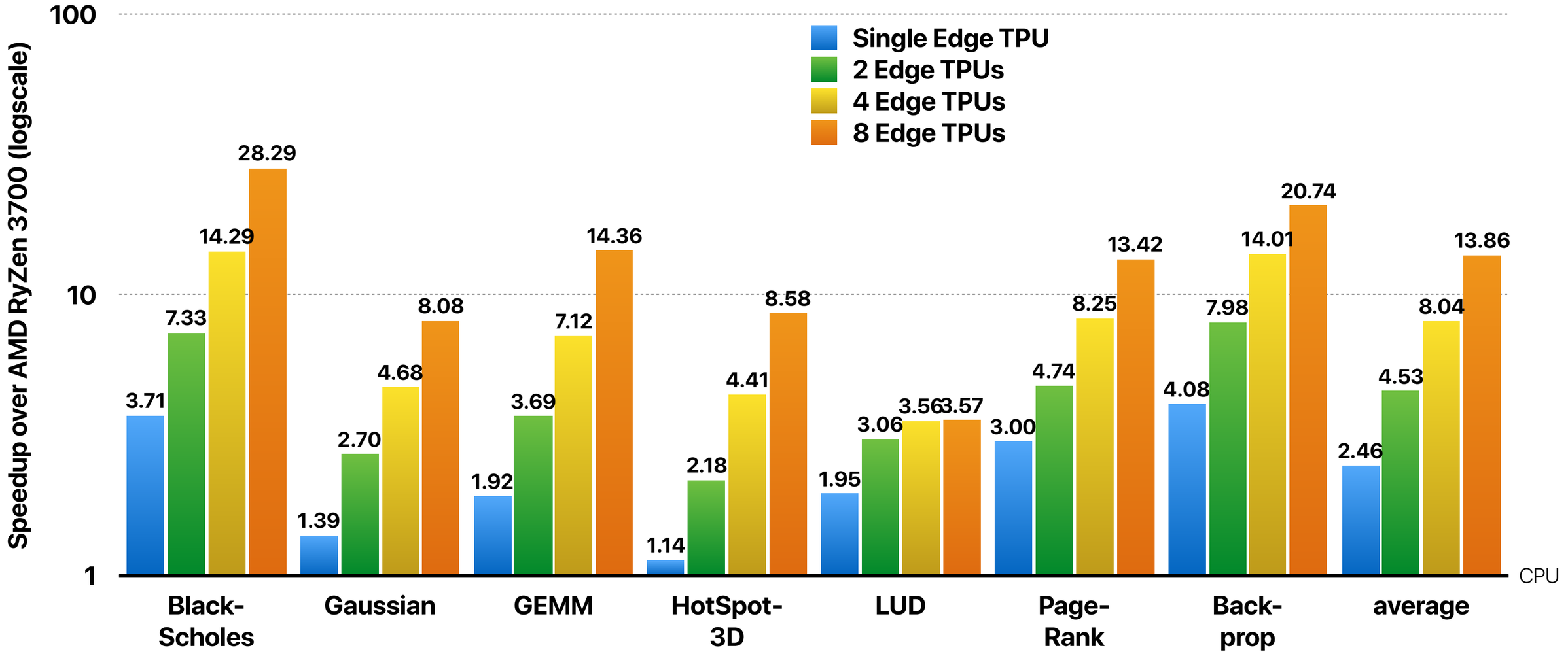
For square matrices with ***M*** columns ***M*** rows, takes ***1*** Conv2D operation if M <=2048

For ***M***=2048, it takes $\frac{1}{160.45} = 0.006232471 17 Sec$

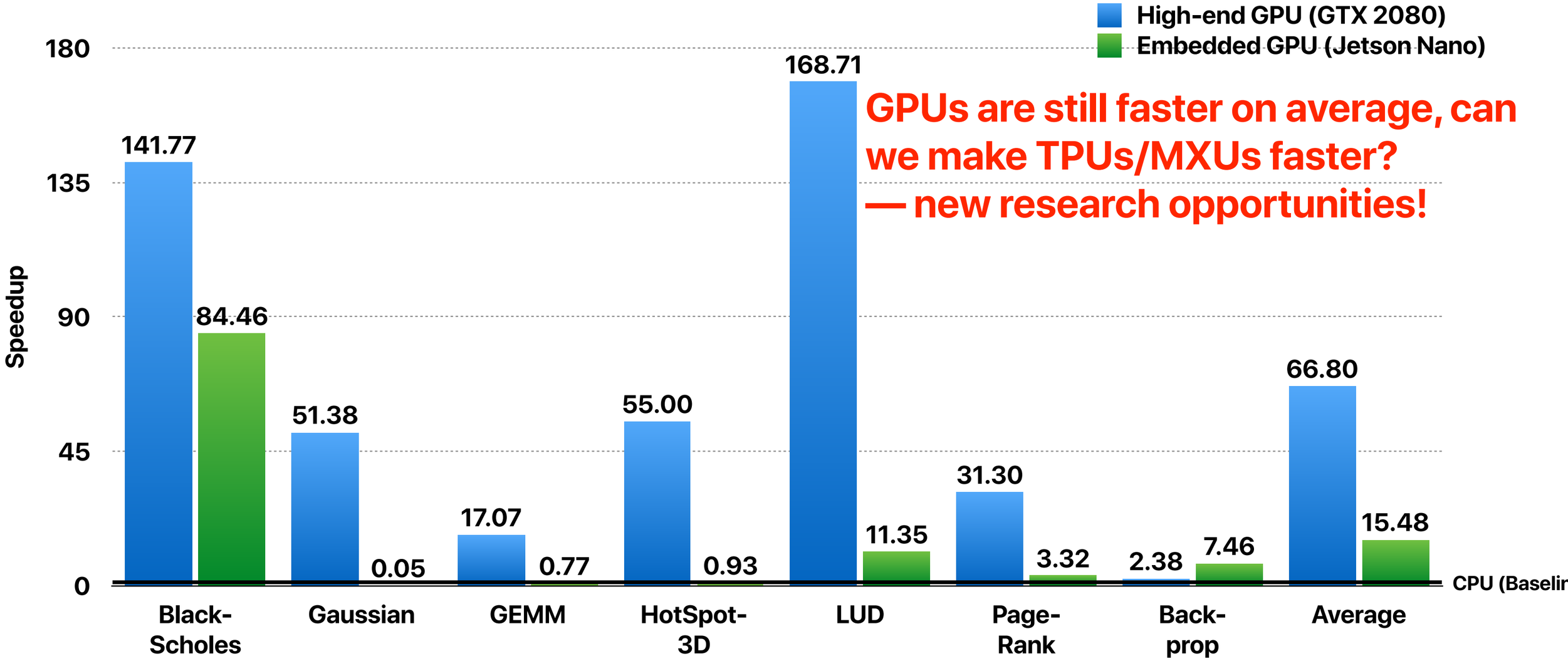
On the real machine



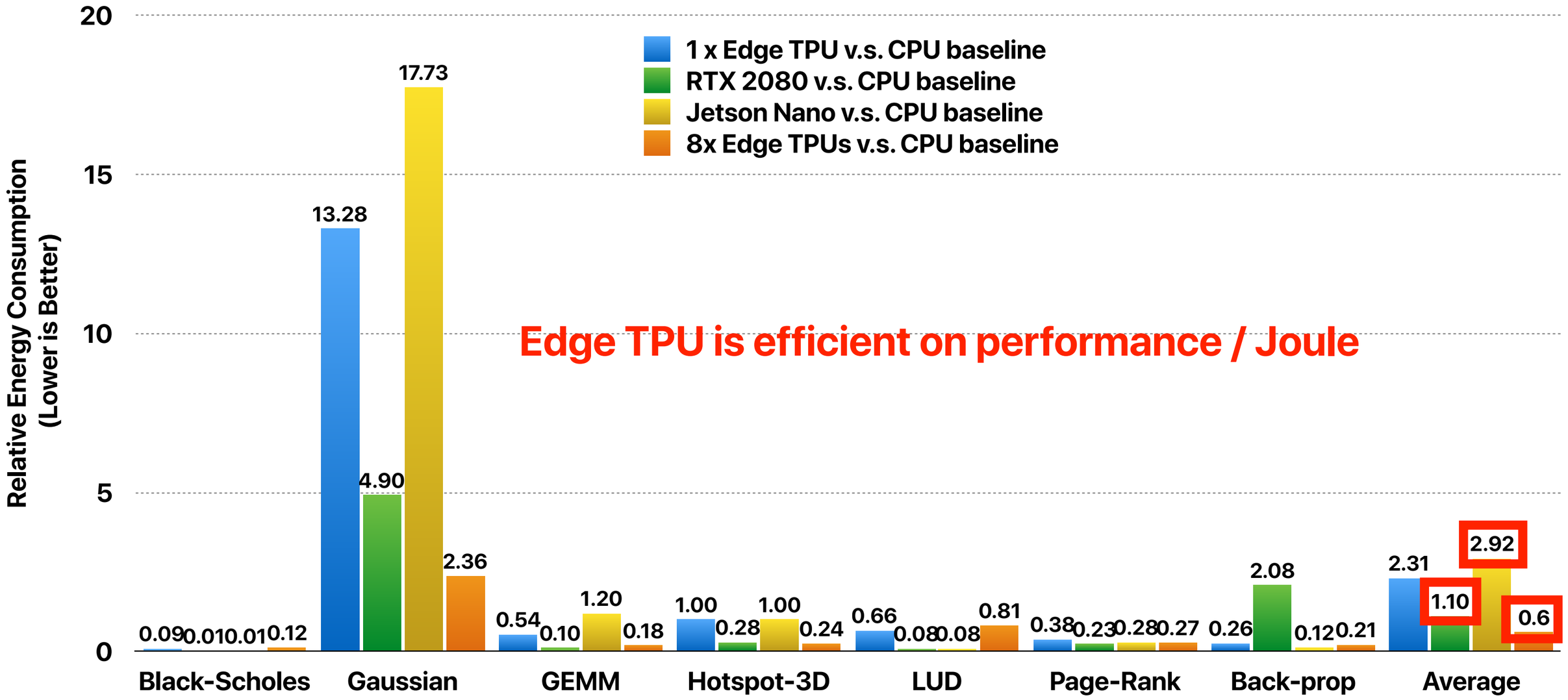
Performance for general-purpose Workloads



Compare 8 Edge TPUs with GPUs



Compare to GPUs



3:00

What’s the main challenges of applying the concept of “GPTPU” in general?

Challenges of GPTPU?

Lessons learned from developing "GPTPU"

- Reverse engineering is interesting but not worth the time
- Developing software frameworks on hardware with steady support
- The accessibility to low-level and backward compatible APIs
- Estimating the performance gain of design using the right/appropriate metrics
- Algorithm implementations should fit the most efficient operations
- The precision support is still below the demand of applications beyond AI/ML

Roogle Project Presentations

- Make an appointment on 4/29 and 5/1 through the Google Calendar
- 15 minute presentation with 3 minute Q & A
- Why & what & how!!! — considering you're giving a presentation at Apple's keynote
  - 7-minute why — why should everyone care about this problem? Why is this still a problem?
  - 5-minute what — what are you proposing in this project to address the problem?
  - 3-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