Accelerating Applications using Edge Tensor Processing Units

Kuan-Chieh Hsu, Hung-Wei Tseng
University of California, Riverside
Riverside, California, USA

PRESENTER:

SUSHIL RAJ REGMI

UNIVERSITY OF NORTH TEXAS

NN accelerators

- Specialized processor optimized to handle specific neural network workloads
- Google's Edge Tensor Processing Units (Edge TPUs) and Apple's Neural Engines
- Power/energy efficiency is better than conventional vector processors (e.g. GPUs)
- Takes tensor/matrices as inputs, generates tensors/matrices as outputs
- Not used in general computing application

Using NN accelerators for non-AI/ML workloads brings few challenges

Challenges

- Microarchitectures and instructions of NN accelerators optimized for NN workloads.
- Trade accuracy with area/energy-efficiency, produce undesirable result
- Programming interface of existing NN accelerators are specialized for AI/ML applications with very few details about hardware/software interfaces
- Tensor algorithms are time consuming, compute kernels are made for scalar/vector processing, makes application unable to take advantage of tensor operators without revisiting algorithms

General-Purpose Computing on Edge TPUs

- Full stack system architecture that enables General-Purpose Computing on Edge TPUs.
- Provides programming interface, a runtime system, compiler and libraries.
- Edge TPU-specific C/C++ extension OpenCTPU
- Tensorizer, evaluates input data and transforms it into ML models Edge TPU can operate upon
- Uses Edge TPUs

Characterizing Edge TPU instructions

- Edge TPU performance numbers available only in TOPS (tera operations for second) and IPS (inferences per second)
- Cannot be used for general-purpose software design
- Result per second (RPS) - amount of final result values an Edge TPU can generate within a second
- Operation per second (OPS) - number of operations done in second

Operator	OPS (ops per second)	RPS (results per second)	Description
conv2D	10268.80	168240326.89	2D Convolution on a matrix
FullyConnected	51924.96	6646394.57	Input vector multiplies a weight matrix
sub	6273.28	82871343.60	Pair-wise subtraction on two matrices
add	6203.52	98293633.48	Pair-wise addition on two matrices
mul	14515.84	216469999.54	Pair-wise multiplication on two matrices
crop	4867.96	1562904391.76	Remove all unwanted elements outside of a sub-matrix from a given 2D matrix and return the sub-matrix
ext	1604.78	3637240203.38	Pad a matrix to the target dimensionality and return the padded matrix
mean	408.54	408.54	Count the average value of all elements in the matrix
max	477.08	477.08	Find the maximum value within a matrix
tanh	3232.31	2148232470.28	Perform tanh function on a matrix pair-wisely
ReLu	11194.26	4043196115.38	Leave only non-zero values on a matrix pair-wisely

Table 1: The maximum OPS and RPS for each Edge TPU operator/instruction

Edge TPU data and model formats

- TPU instructions take two types of data inputs
  - Tensor used for input datasets
  - Model generated and compiled by TFLite framework
- GPTPU runtime library translates one of the instruction inputs as model for the Edge TPU (model creation overhead)
- Need to create different model for different inputs
- Use python based TFLite compiler for model generation
- Translating 2k x 2k matrix into model takes 2.7 seconds
- TPU model encoding, and compiler code are not available

Edge TPU data and model formats

- Reverse-engineered model formats by creating models with different inputs, dimensions, and value ranges
  - **Header :** 120-byte for model format version.
  - Last 4 bytes for size of data section
  - **Data section :** binary-encoded 8-bit integers stored in row major order
  - **Metadata section** : describes data-section dimension in terms of rows and columns, scaling factor used when rescaling raw data into 8-bit integers

Conclusion on observation

- Edge TPUs are optimized for operation on sub-matrices of 128x128.
- If input data doesn't align with the required dimension, the compiler adds zero padding
- Model stores data as 8-bit integers

Overview of GPTPU system

- Heterogenous computing system stack
  - Application layer
  - Compiler layer
  - Host System
  - System Interconnect
  - Hardware Accelerators

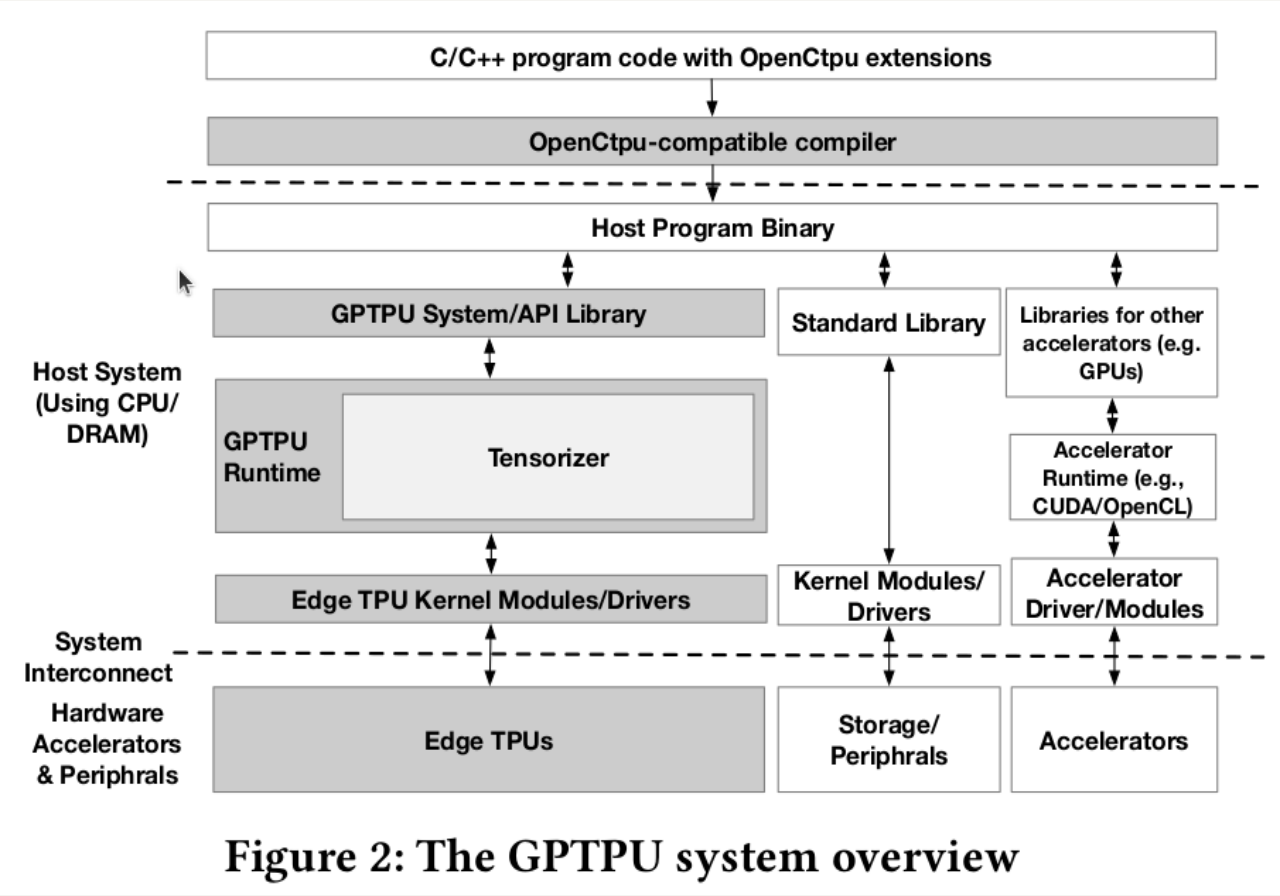


Figure 2: The GPTPU system overview

Overview of GPTPU system

- OpenCtpu
  - Programming language front-end
  - Create a host program for TPU tasks
  - Coordinates the use of computing resources and data exchanges

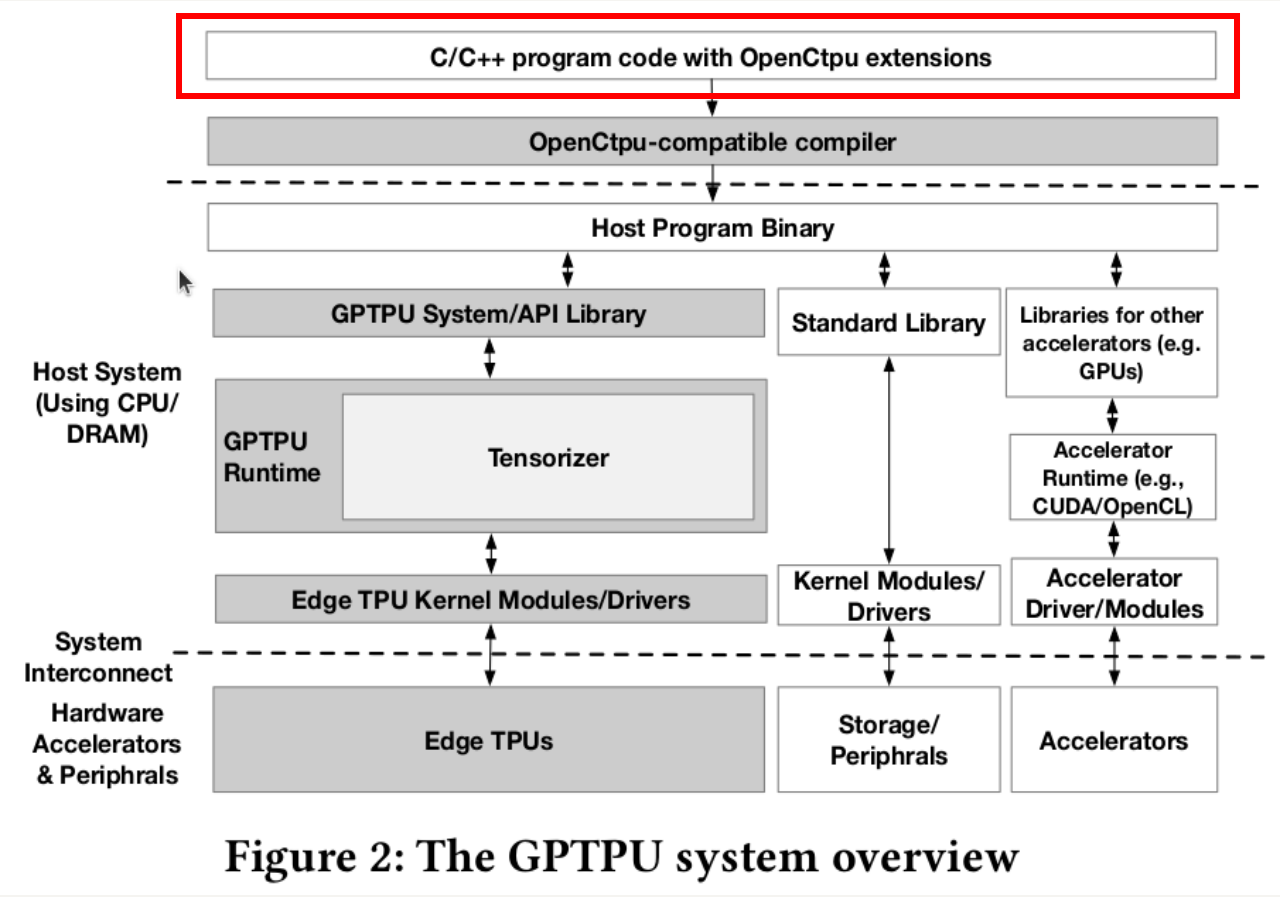


Figure 2: The GPTPU system overview

OpenCtpu

- Shares similarities with popular GPU programming models like CUDA and OpenCL
- TPU program
  - Kernel function describing computation for TPUs
  - Input/output data buffers for TPU kernels
  - Enqueuing kernel functions
  - Use openctpu_invoke_operator for Edge TPU operations in kernel function

```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;
}
```

Figure 3: An OpenCtpu code sample

GPTPU library and runtime system

- Coordinates available TPU hardware
- Schedules TPU operations from programmer defined TPU tasks
- Prepare inputs/outputs for TPU operation

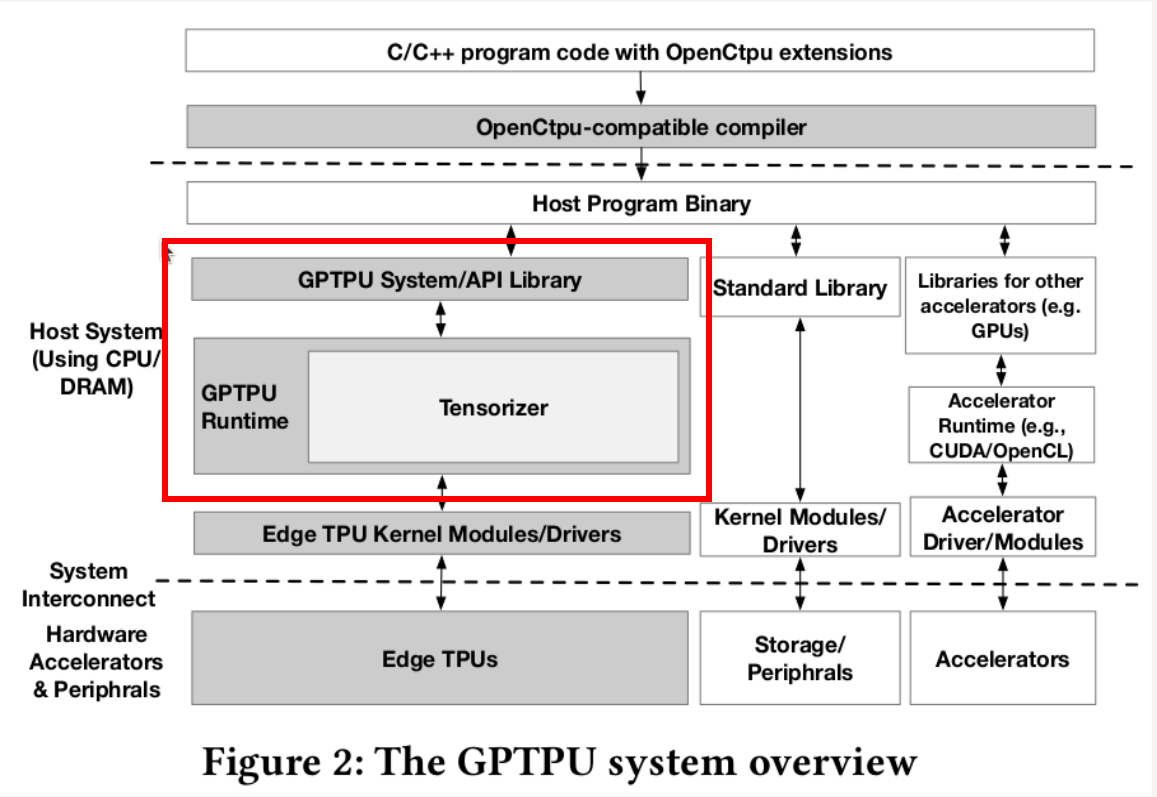
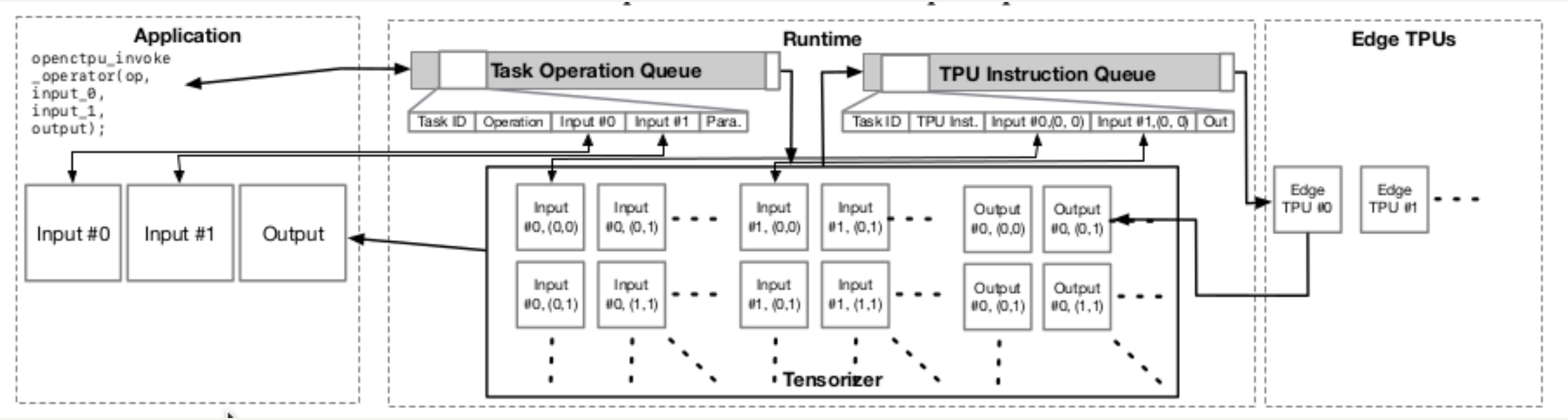


Figure 2: The GPTPU system overview

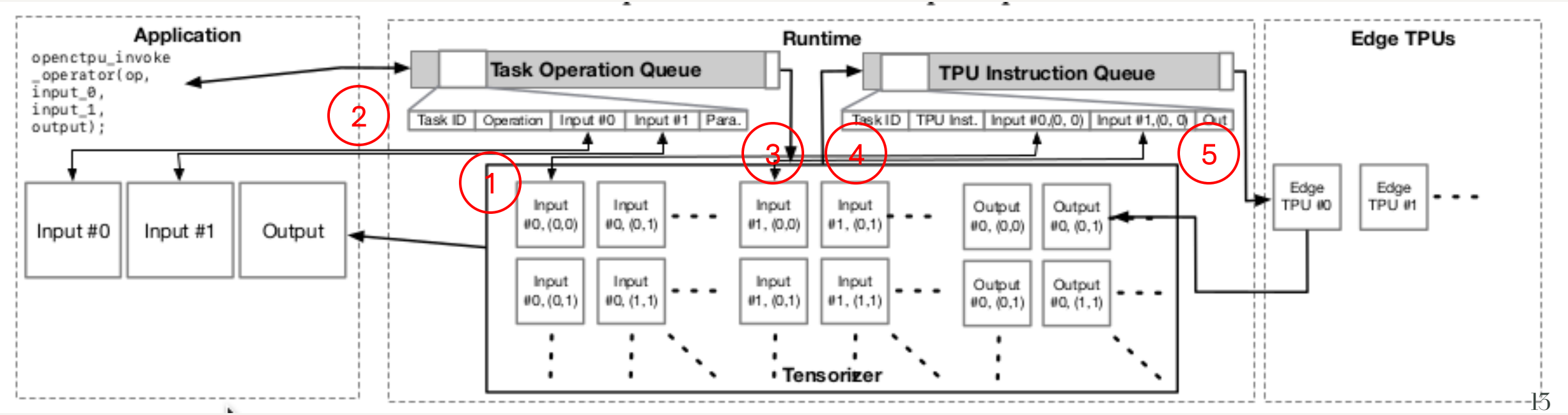
GPTPU library and runtime system: Task Scheduling

- Task-scheduling policy: dataflow-based algorithm on front-end OPQ and back-end IQ



GPTPU library and runtime system: Task Scheduling

- Task-scheduling policy: dataflow-based algorithm on front-end OPQ and back-end IQ
  1. When the program calls the openctpu_enqueue function, initiates new task ID
  2. Calls openctpu_invoke_operator create OPQ entry with task ID created
  3. Issue entries in OPQ to Tensorizer for each task (for each operator)
  4. Tensorizer divides a task into instructions in IQ
  5. Schedules to Edge TPU based on task ID



Tensorizer

Transforms and optimizes programmer-requested operations into instructions

1. Mapping operators into instructions
2. Data transformation
3. Overhead of model creation

Tensorizer

Mapping operators into instructions

- Dynamically partitioning tasks into Edge TPU instructions working on optimal data sizes/shapes(128x128)

- For pair-wise/element-wise operators (add,sub,mul,tanh,relu)
  - Divides input data into tensors and models that contain sub-matrices
  - Rewrites the operator into set of Edge TPU instructions where each works on sub-matrix
  - Collects the result in the corresponding memory locations

- For matrix-wise operator (mean and max), it generates CPU code to aggregate the received values from the results of instruction

Tensorizer

Data transformation

- Rescales values into fixed point number (i.e 8-bit) to make applicable with Edge TPU
- Determine scaling factor based on:
  - Sequence of operators
  - Number of operators
  - Range of input data

$$S = \frac{1}{max(|output_{max}|, |output_{min}|)}$$

Where, output max = expected maximum output value
Output min = expected minimum output value

- GPTPU applies different formula for different types of operators

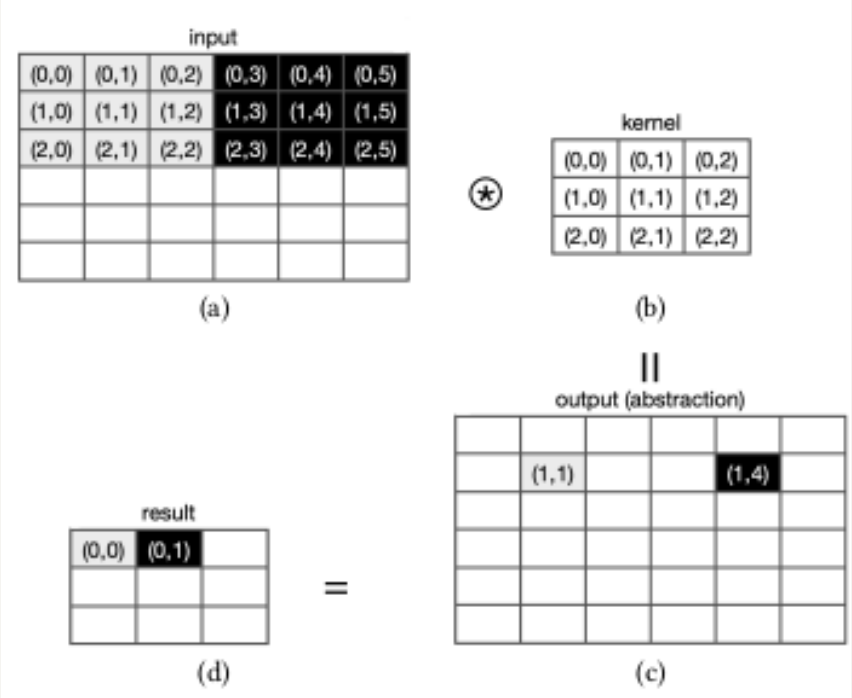
Tensorizer

Overhead of model creation

- Dynamically create model from arbitary input data
- C-based tensorizer reduce the latency of generating a model (2k x 2k matrix) to 1.8 ms – 1500x speedup than TFLite compiler.

Optimizing applications for GPTPU: GEMM

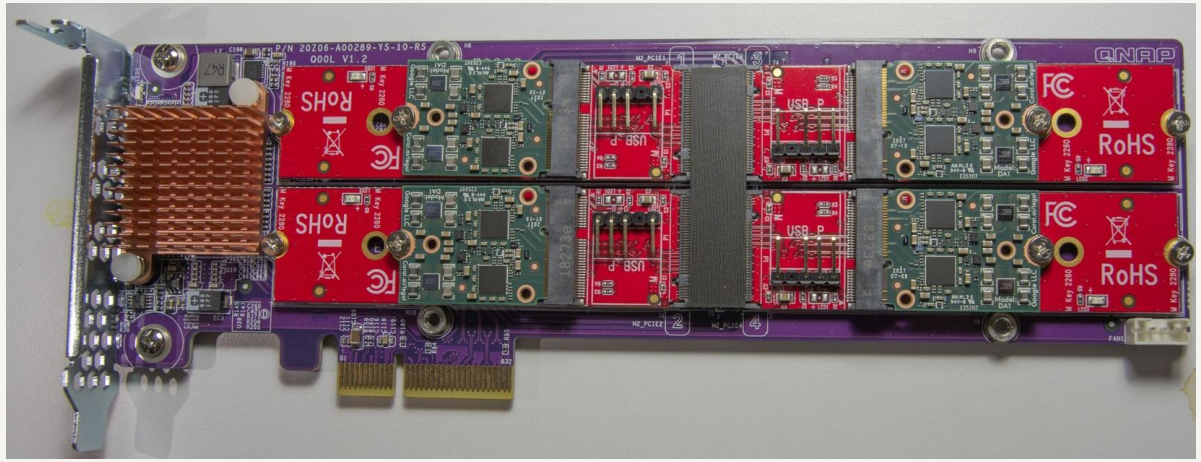
- FullyConnected operator: select either matrix and iterate through a column or row of another matrix
- Conv2D operator: take one of its inputs as kernel, multiplies each kernel element with an element mapping to corresponding locations
- Conv2D operator is preferred over FullyConnected operator for GEMM
- GPTPU provide GEMM operation through optimized library function, tpuGemm similar to cublasGemm function



Experiment

Experiment: System Platform

- Uses Edge TPU in PCIe M.2 form factors, for lower latency and better bandwidth
- Custom built quad-Edge TPU PCIe expansion
- AMD Ryzen 3700X CPU, 64 GB DDR4
- Can host upto 8 x M.2 Edge TPUS

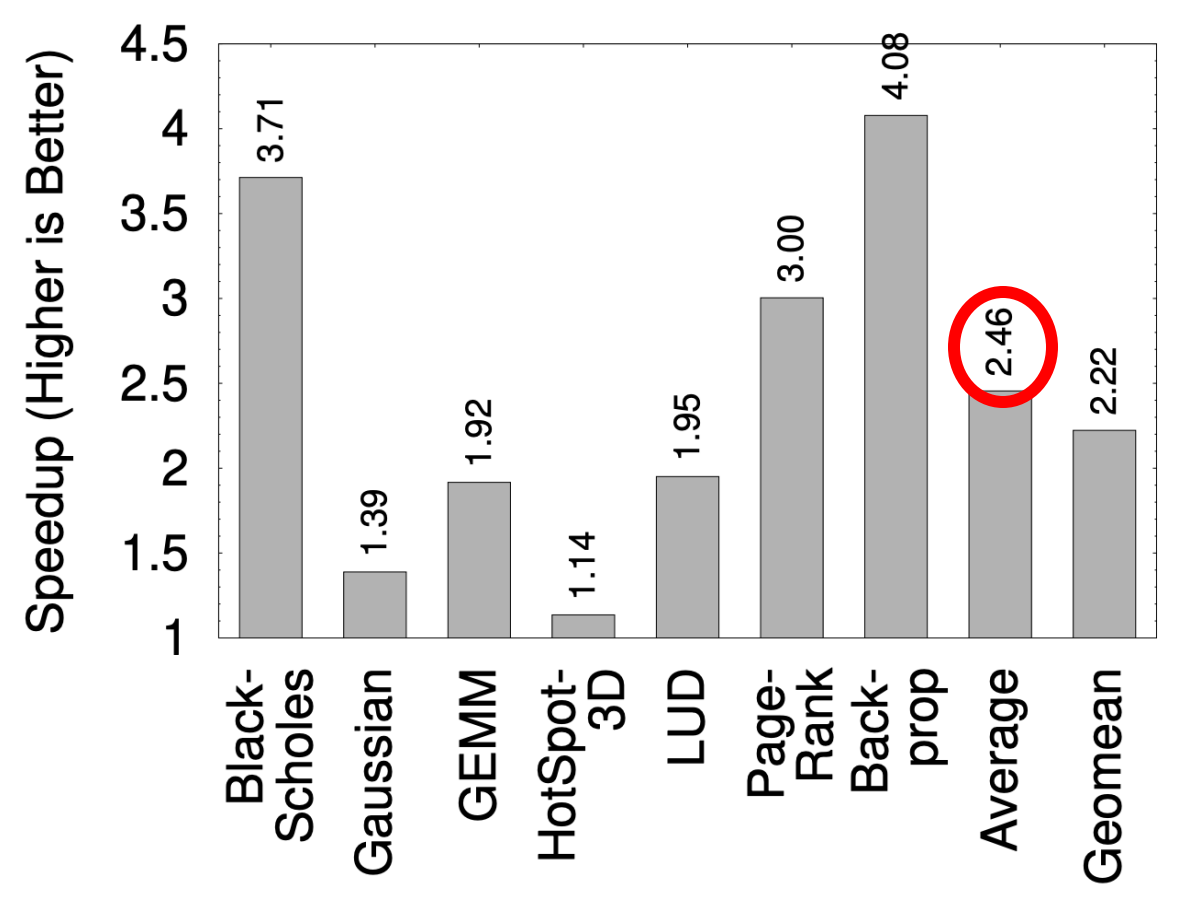


Experiment: Benchmark application

Benchmark	Input Matrices	Data Size	Category	Baseline Implementation
Backprop	1×8K×8K	512MB	Pattern Recognition	[76, 77]
BlackScholes	1×256M×9	9GB	Finance	[78]
Gaussian	1×4K×4K	64MB	Linear Algebra	[76, 77]
GEMM	2×16K×16K	1GB	Linear Algebra	[71, 72, 79]
HotSpot3D	8×8K×8K	2GB	Physics Simulation	[76, 77]
LUD	1×4K×4K	64MB	Linear Algebra	[76, 77]
PageRank	1×32K×32K	4GB	Graph	[80]

Experiment: Single core performance: GPTPU vs CPU

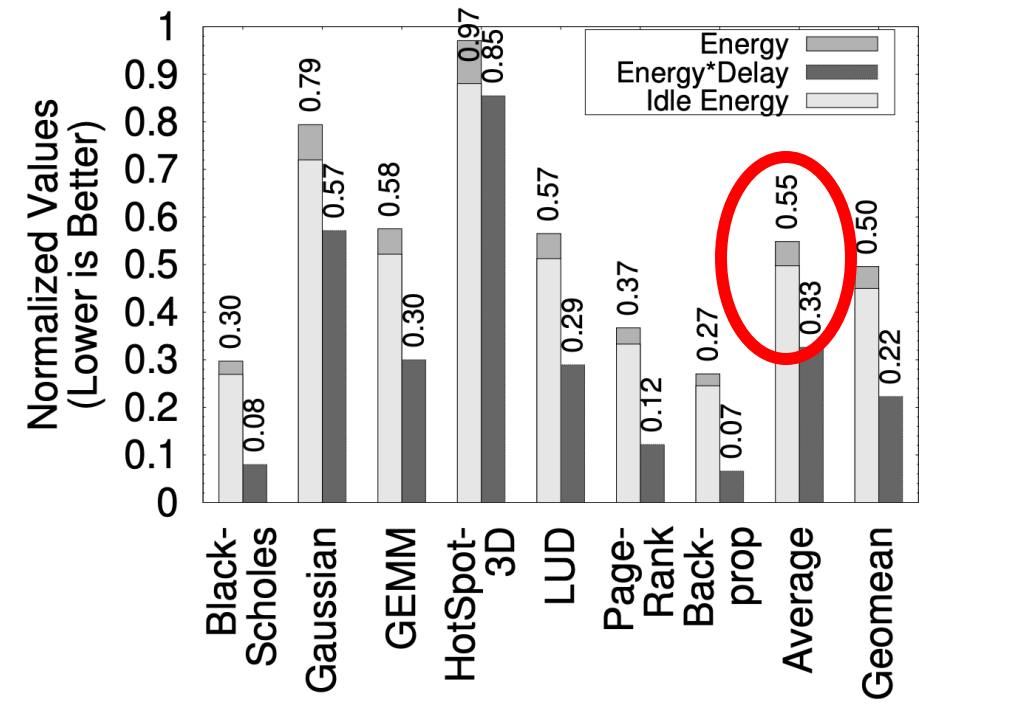
Speedup



- On average GPTPU is 2.46 times faster than CPU

Experiment: Single core performance: GPTPU vs CPU

Energy consumption and energy delay



- GPTPU consumes only 5% of the active energy and 51% of idle energy that a CPU consumes
- GPTPU has 67% energy-delay improvement over CPU

Experiment: Single core performance: GPTPU vs CPU

- GPTPU sacrifices accuracy but only to a limited degree
- MAPEs and RMSEs for GPTPU applications
- MAPE is always less than 1 % across all applications
- Largest RMSE, 0.98%, acceptable
- High error rates in default datasets than on synthetic inputs

MAPEs

Benchmark	default	$-2^{7} \leq x < 2^{7}$	$-2^{15} \leq x < 2^{15}$	$-2^{31} \leq x < 2^{31}$
Backprop	0.12%	0.17%	0.10%	0.11%
Blackscholes	0.18%	0.18%	0.18%	0.18%
Gaussian	0.00%	0.00%	0.00%	0.00%
GEMM	0.89%	0.90%	0.90%	0.90%
HotSpot	0.50%	0.49%	0.46%	0.46%
LUD	0.00%	0.00%	0.00%	0.00%
PageRank	0.61%	0.73%	0.73%	0.73%
Average	0.33%	0.35%	0.34%	0.34%

(a)

RMSE

Benchmark	default	$-2^{7} \leq x < 2^{7}$	$-2^{15} \leq x < 2^{15}$	$-2^{31} \leq x < 2^{31}$
Backprop	0.14%	0.17%	0.12%	0.12%
Blackscholes	0.33%	0.33%	0.33%	0.33%
Gaussian	0.00%	0.00%	0.00%	0.00%
GEMM	0.98%	0.91%	0.91%	0.91%
HotSpot	0.64%	0.64%	0.59%	0.59%
LUD	0.00%	0.00%	0.00%	0.00%
PageRank	0.41%	0.91%	0.91%	0.91%
Average	0.41%	0.42%	0.41%	0.41%

(b)

Experiment: GPTPU-GEMM vs 8-bit CPU GEMM

Comparison of GPTPU with state-of-the-art FBGEMM low-precision CPU matrix-multiplication library(use 8-bit input)

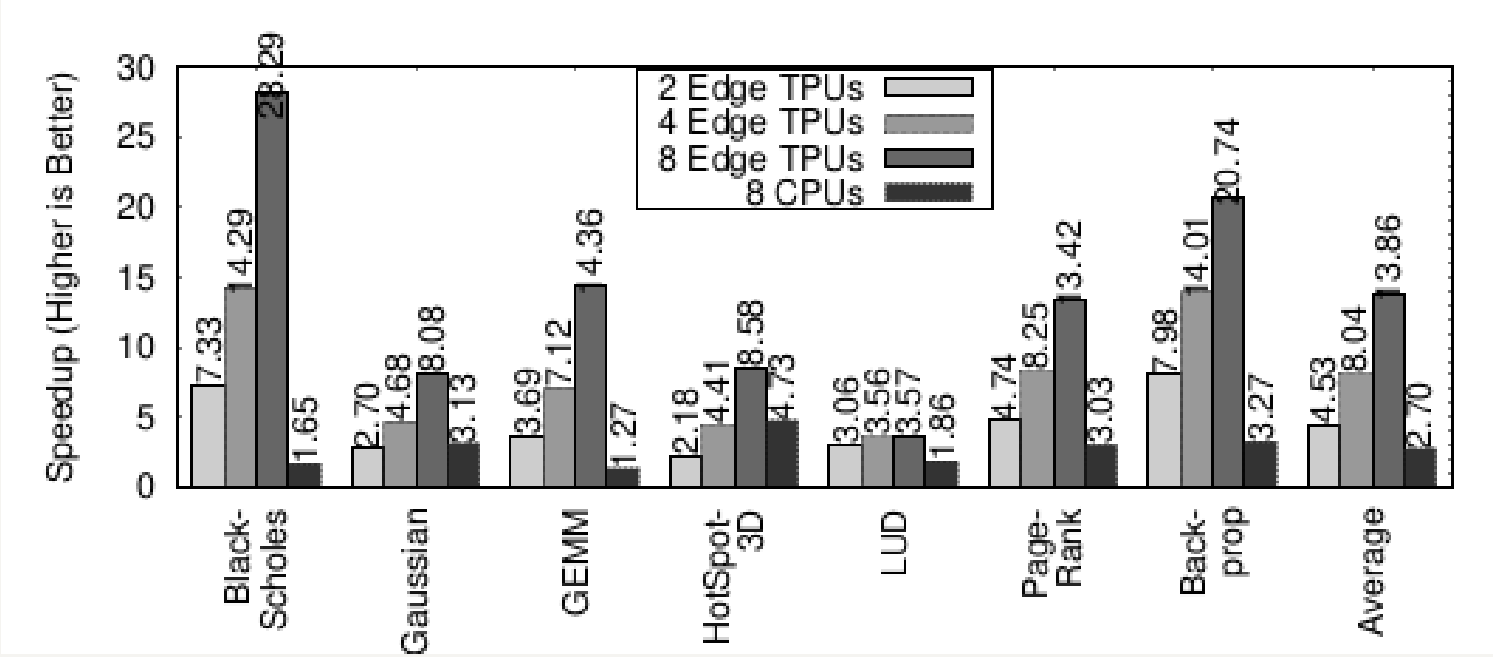
Range of Values		0–2	0–4	0–8	0–16	0–32	0–64	0–128
Speedup over FBGEMM		1.26	1.27	1.28	1.22	1.28	1.27	1.28
RMSE	FBGEMM	0.00	0.00	0.00	0.00	0.47	0.87	0.97
	TPUGEMM	0.00	0.00	0.00	0.00	0.00	0.00	0.01

GPTPU outperforms FBGEMM on high performance CPU cores with 1.22x to 1.28x across all configurations

FBGEMM's RMSE is poor after matrix entry value exceeds 16

Experiment: Parallel processing with multiple Edge TPUs

Speed up on adding more Edge TPUs

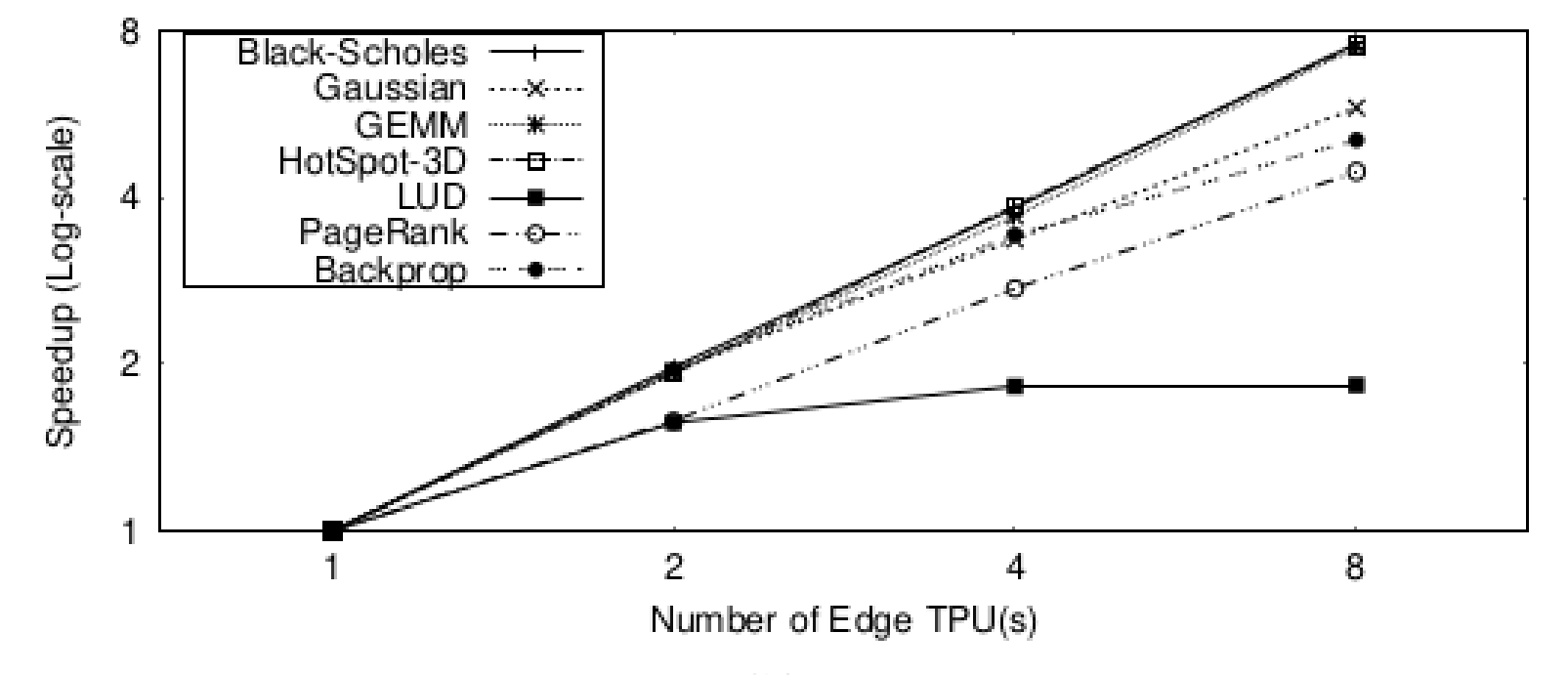


Baseline:
Single-core CPU

GPTPU achieves 13.86 x speedup on average

Experiment: Parallel processing with multiple Edge TPUs

Log scale performance with up to 8 Edge TPU

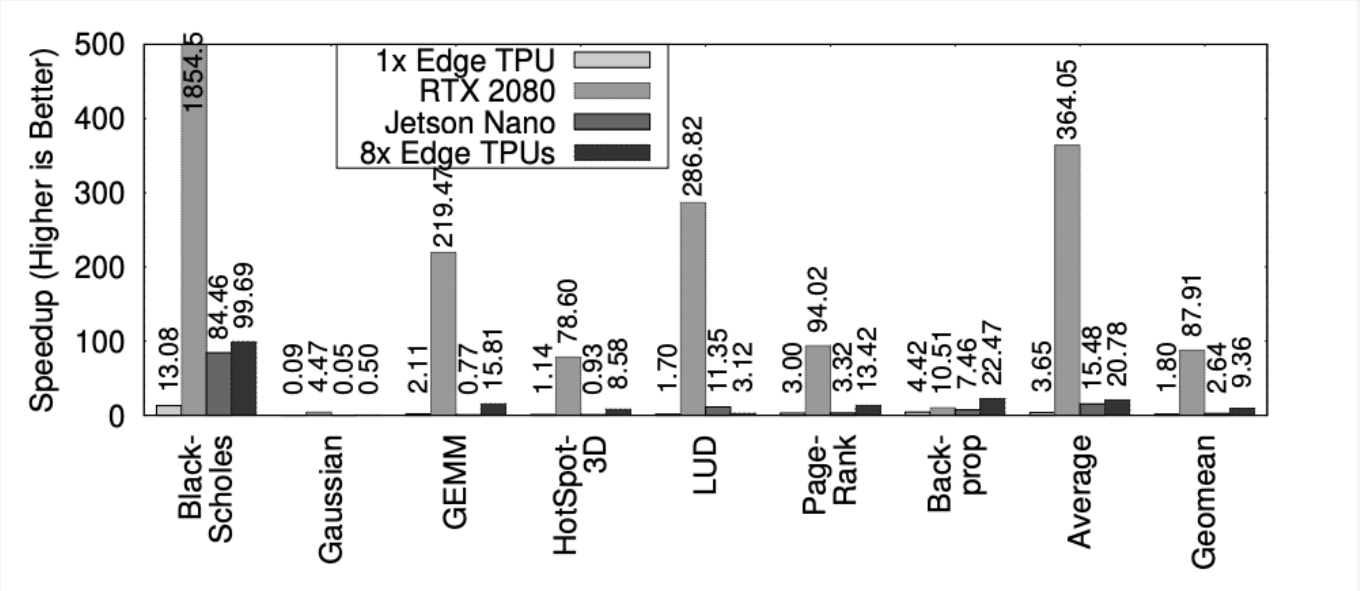


Good performance scaling for 6 out of 7 applications

Exception is LUD: it partitions matrices into four sub-matrices so it is difficult to scale the performance

Experiment: Comparison with GPUs

Comparison with GTX 2080 and NVIDIA's embedded Jetson Nano



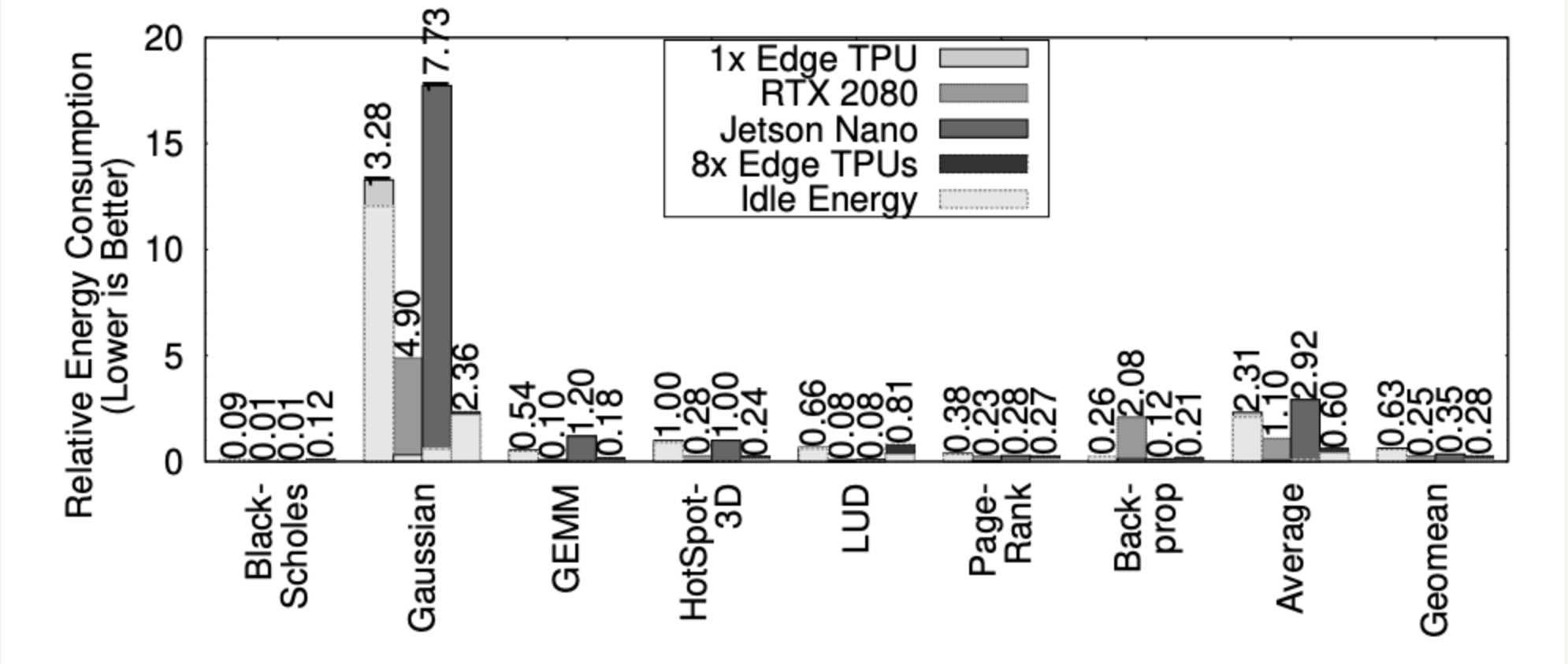
Baseline:
Single-core CPU

GTX 2080 GPU is 69x faster than an Edge TPU

Embedded GPU is 2.30x faster than an Edge TPU

8x Edge TPUs is 2.48x faster than Jetson Nano

Experiment: Energy Consumption



8x Edge TPU system can save energy by 40 % from the CPU baseline

GTX 2080 consumes 14x the energy of 1x Edge TPU on average (exclude idle)

Jetson Nano consumes 23.55x more energy than 1x Edge TPU

Contribution

- Presents full stack system architecture for General-Purpose Computing on Edge TPUs
- Includes powerful programming interface i.e., OpenCtpu similar to that of CUDA/OpenCL
- Characterize the capability and previously unidentified architectural design of Edge TPUs
- Introduces Tensorizer, a module that dynamically maps operators to neural network models and Edge TPU instructions for efficient use of the underlying accelerators.

Limitations

- Latency overhead introduced due to input to model conversion could be significant for application that need to frequently create new model
- Focuses only on Edge TPU which may not generalize to other accelerators
- High error rates for real datasets