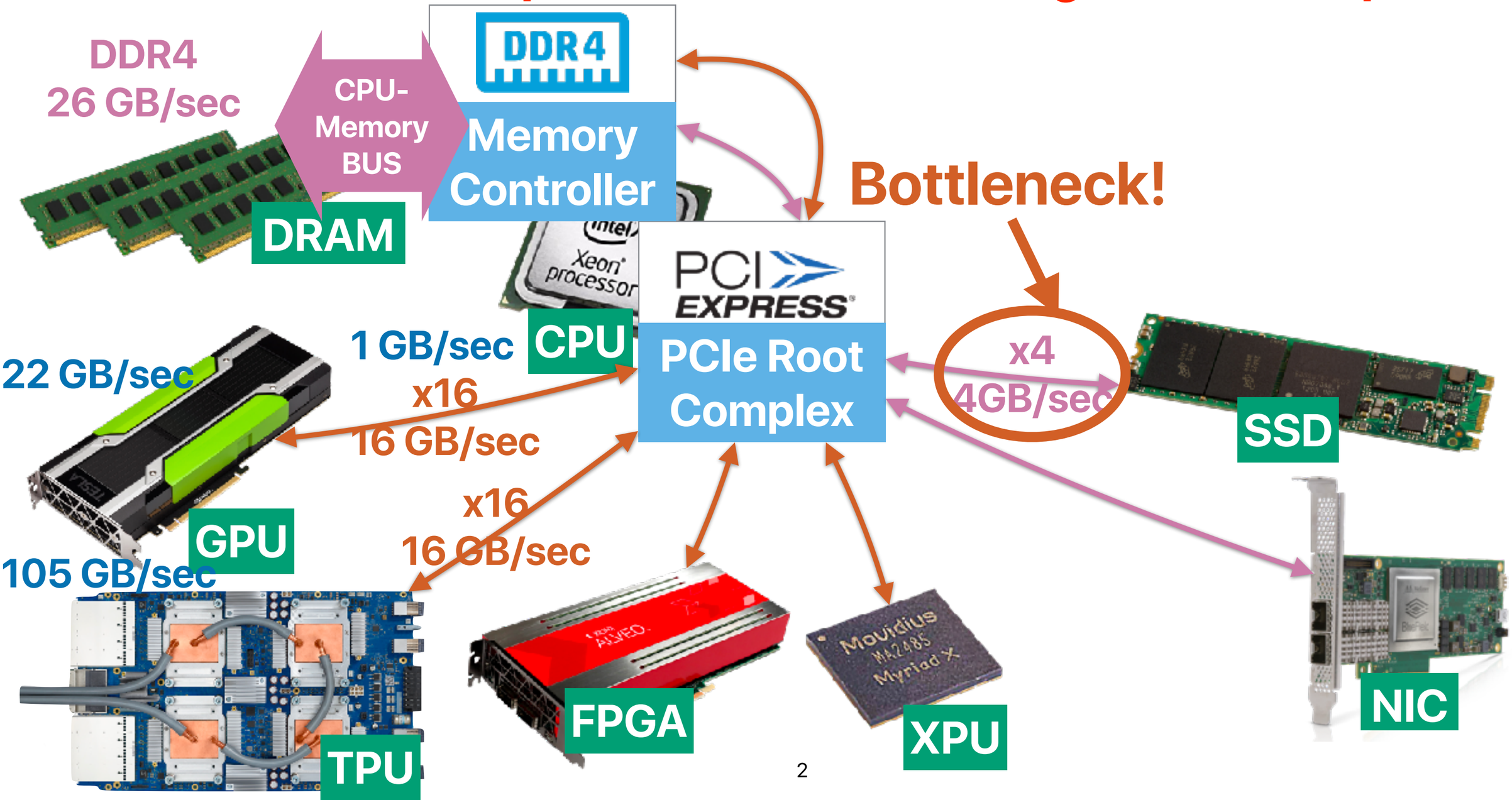
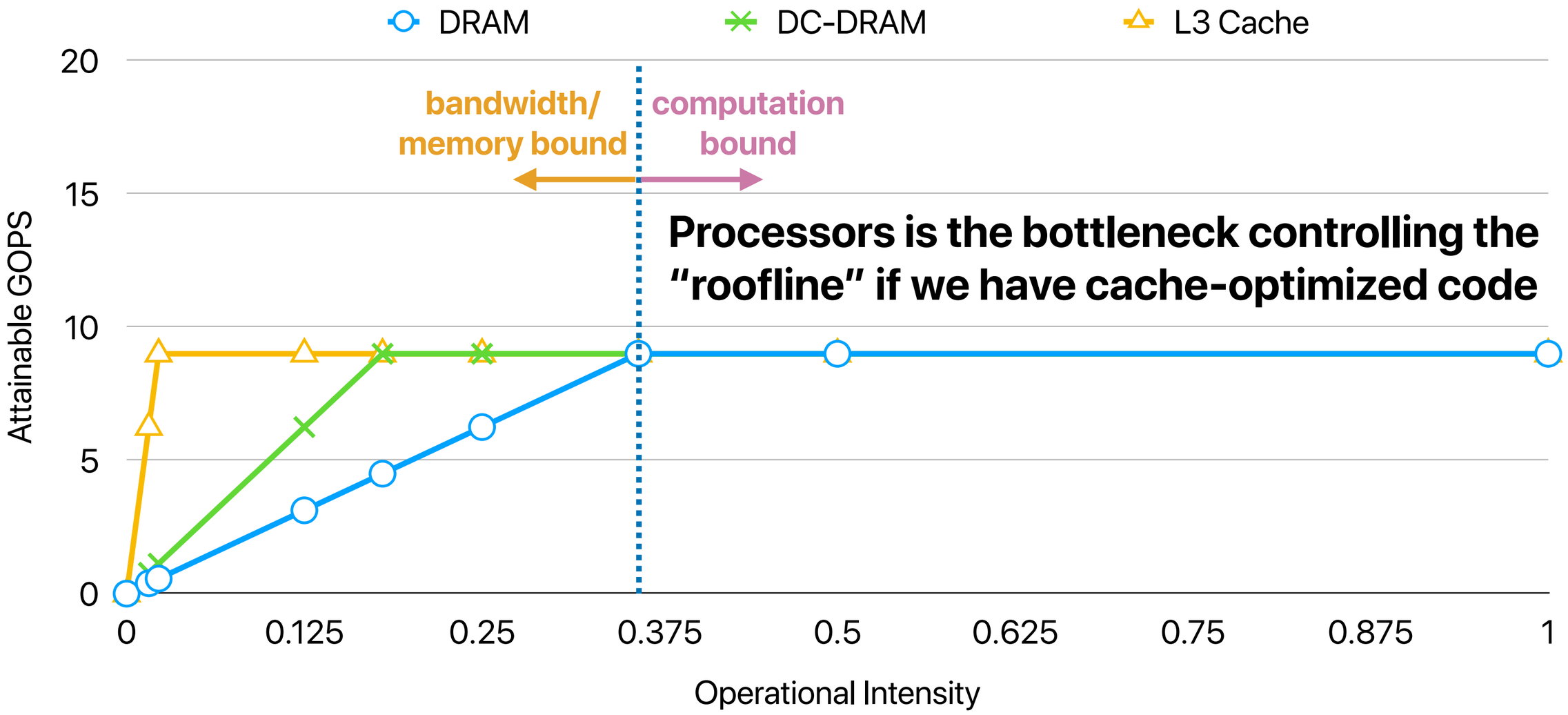
Modern Heterogeneous Computers: (2) Computing Units

Hung-Wei Tseng

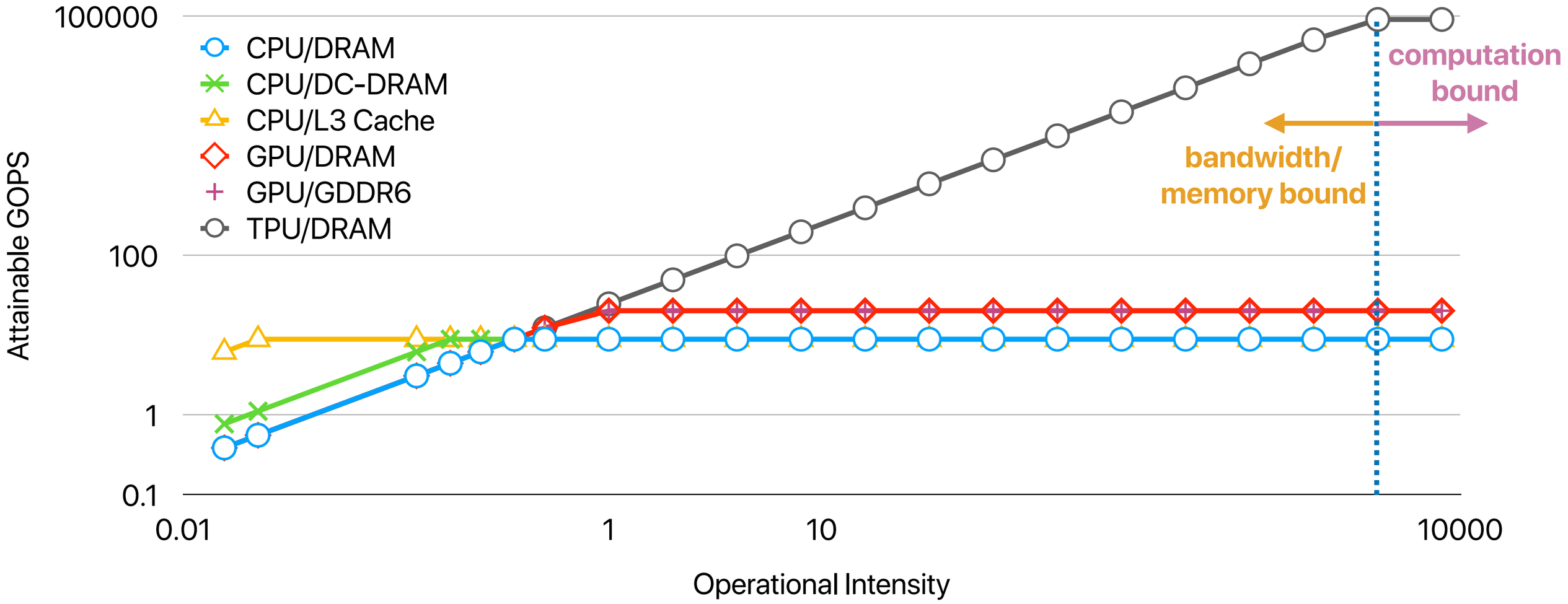
Review: The "data path" in modern heterogeneous computers



Recap: The roofline of CPU



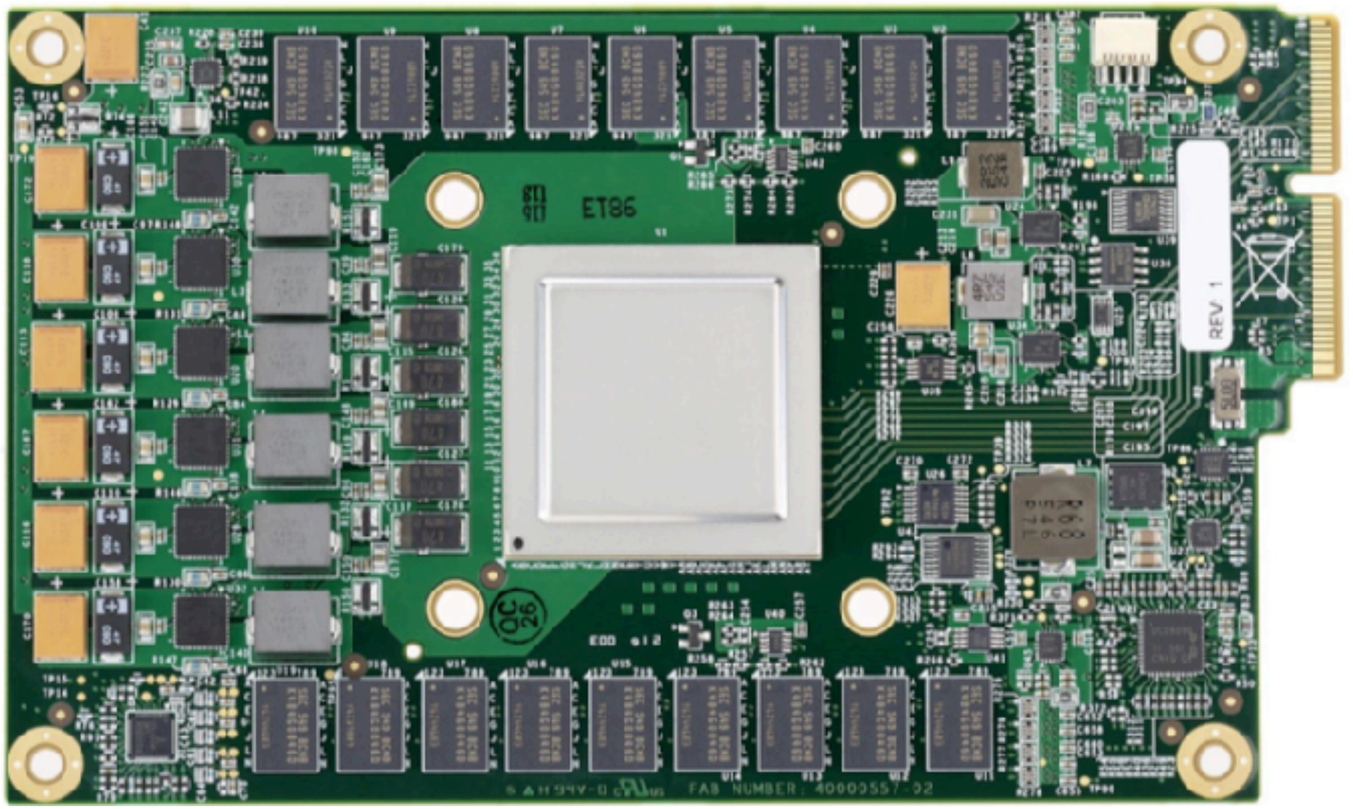
Recap: the roofline after using hardware accelerators



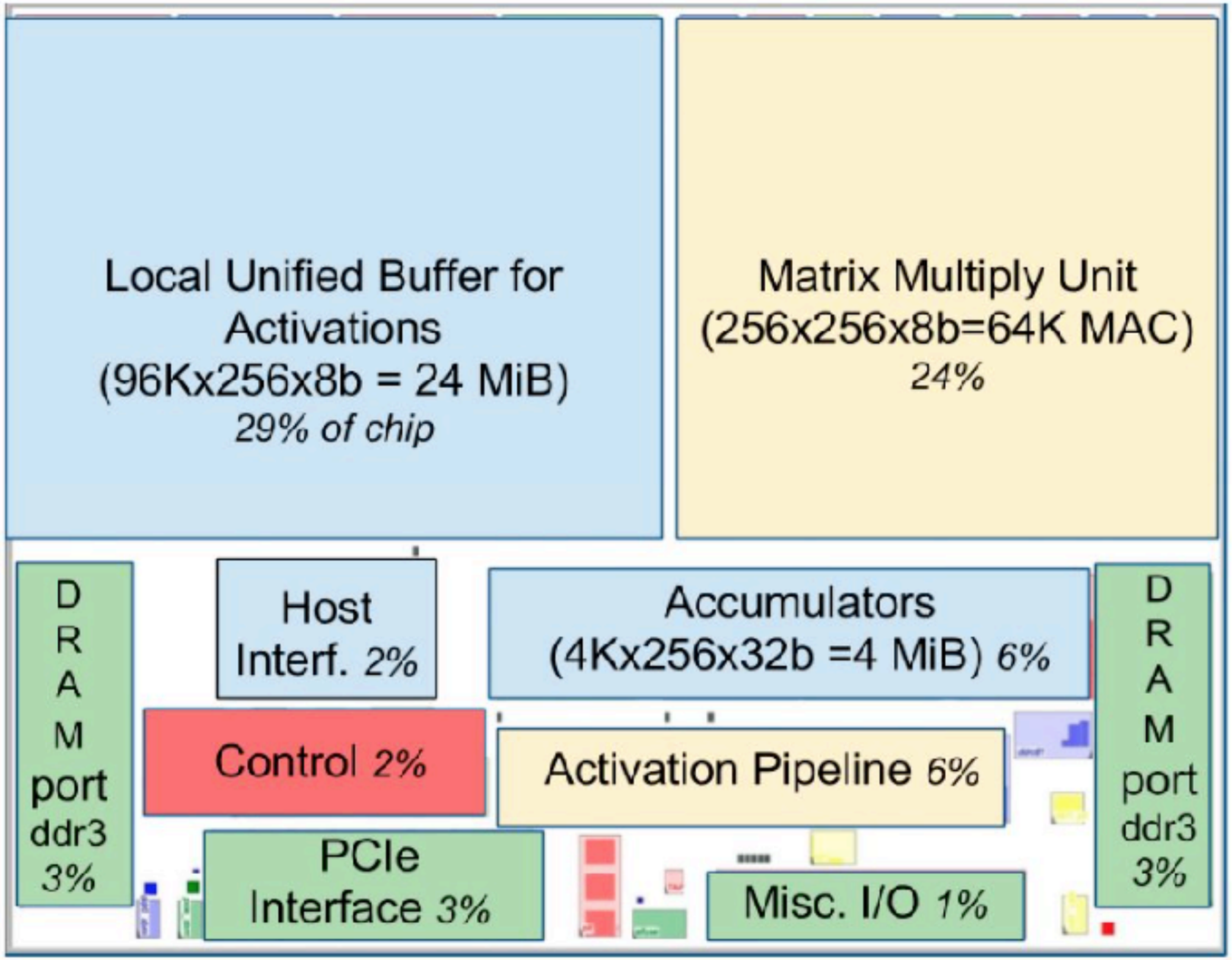
Example of a hardware accelerator — Tensor Processing Unit

- N. Jouppi, C. Young, N. Patil and D. Patterson. Motivation for and Evaluation of the First Tensor Processing Unit. In IEEE Micro, vol. 38, no. 3, pp. 10-19. 2018
- Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In ISCA '17. 2017.

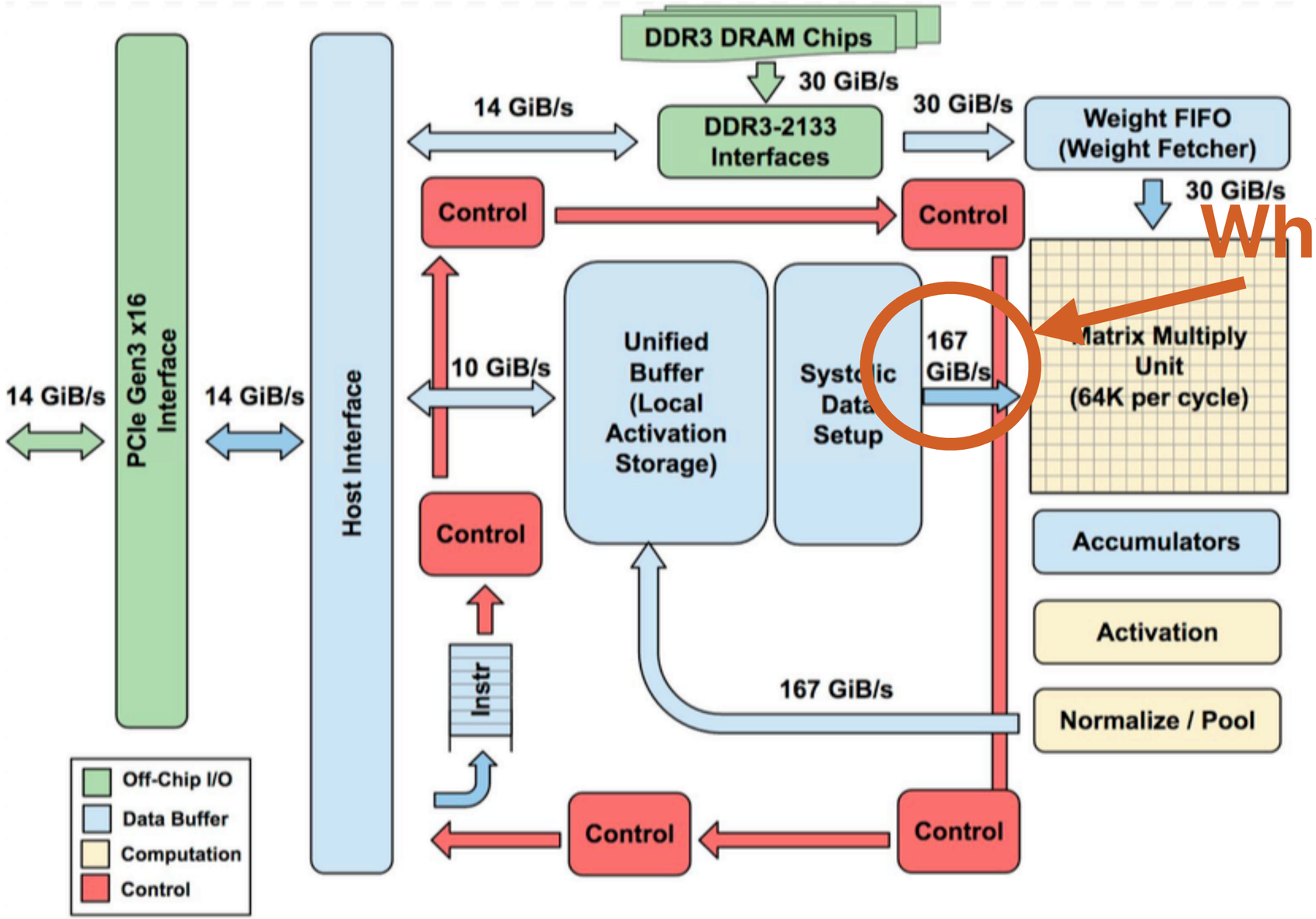
What TPU looks like



TPU Floorplan



TPU Block diagram



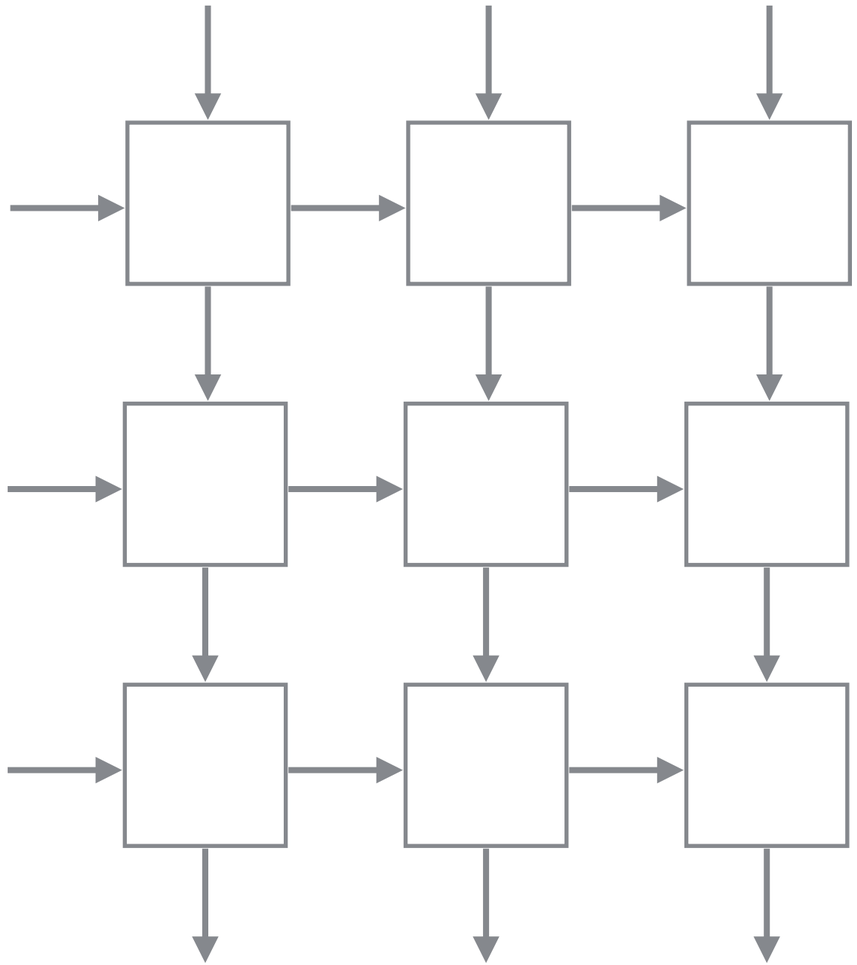
Systolic Arrays used by AI/ML Accelerators

B

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

A

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



H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

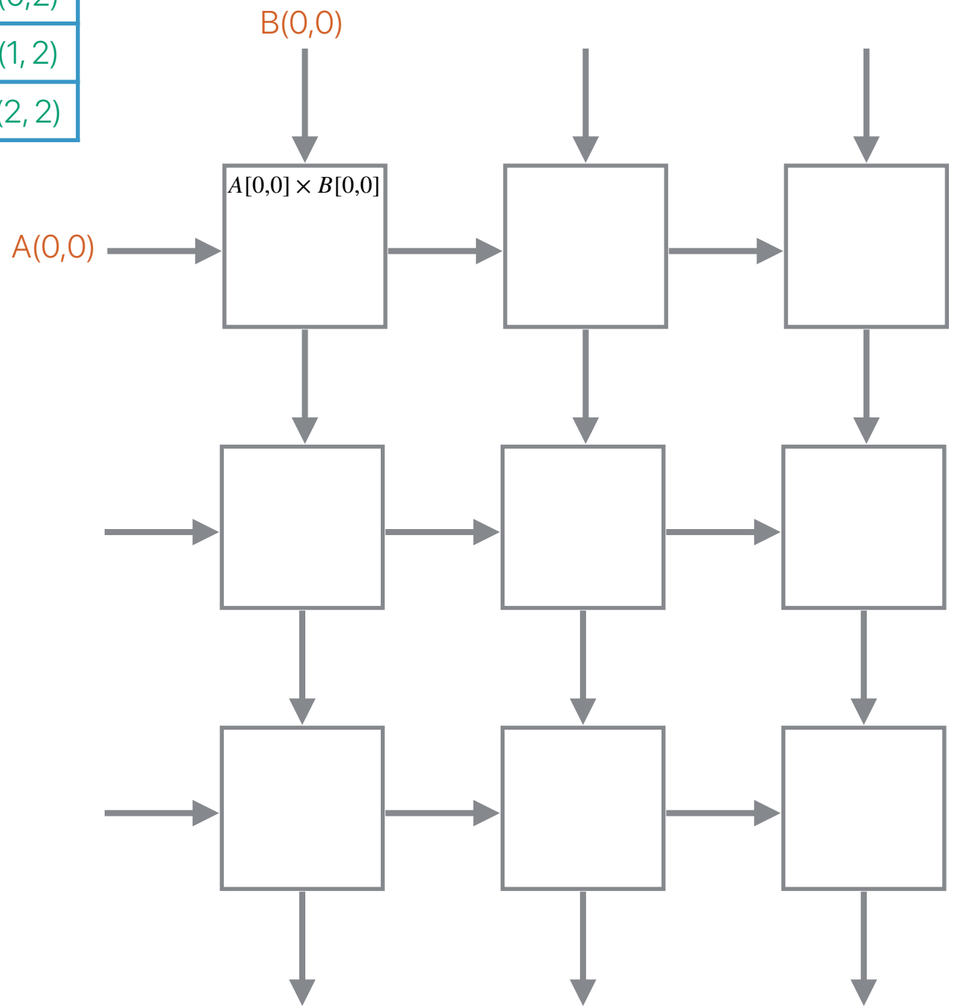
#1

B

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

A

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



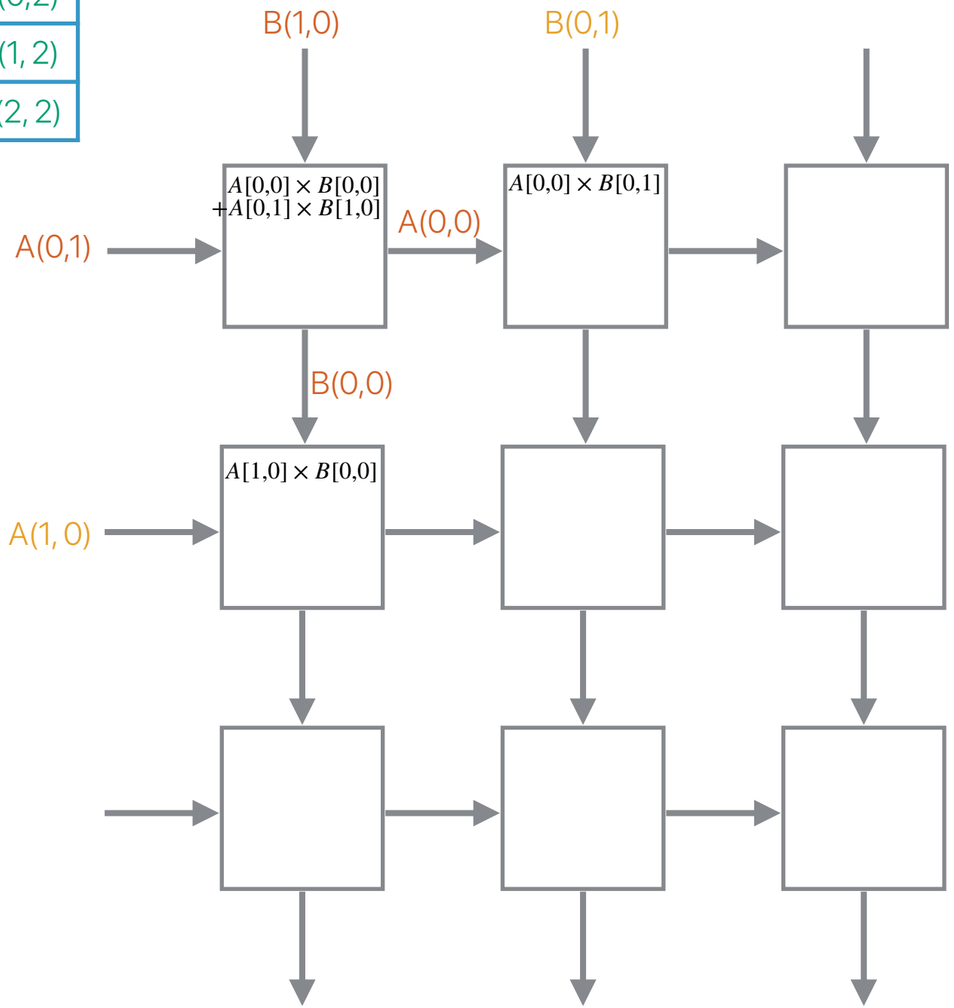
H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

#2

B

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



A

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

H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

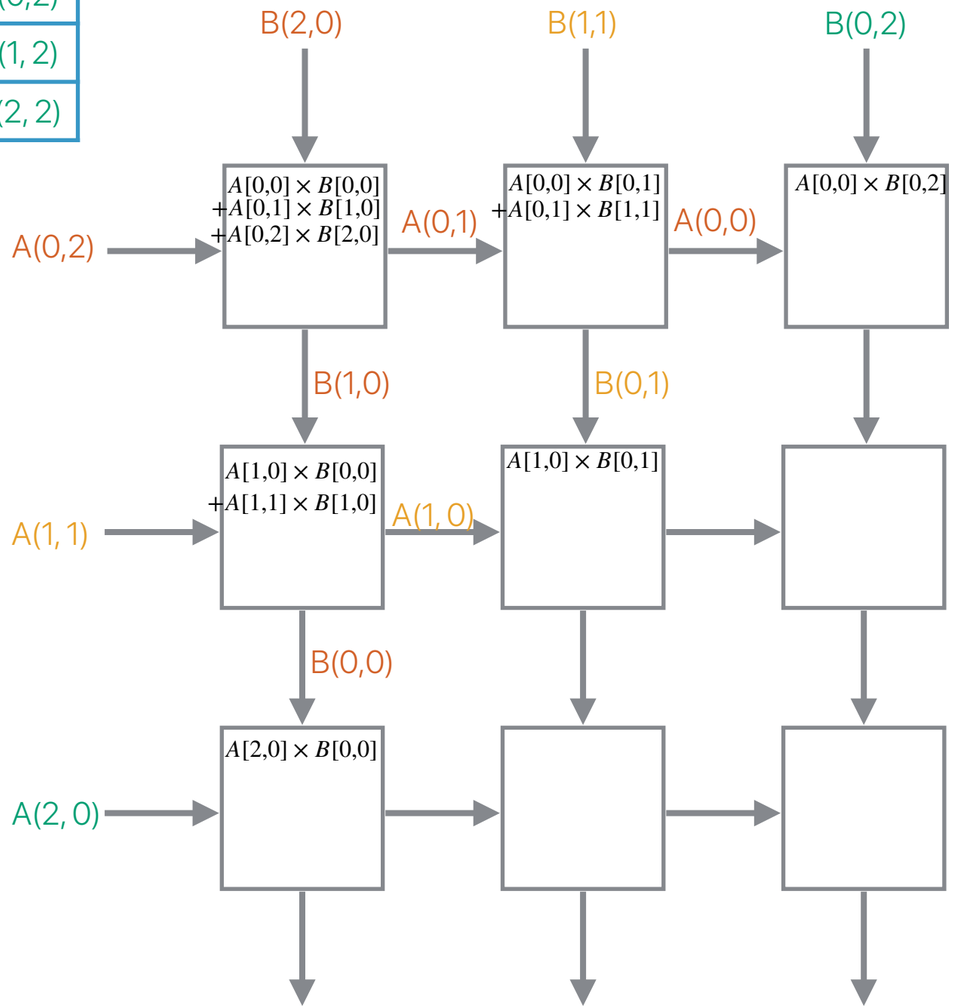
#3

B

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

A

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



H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

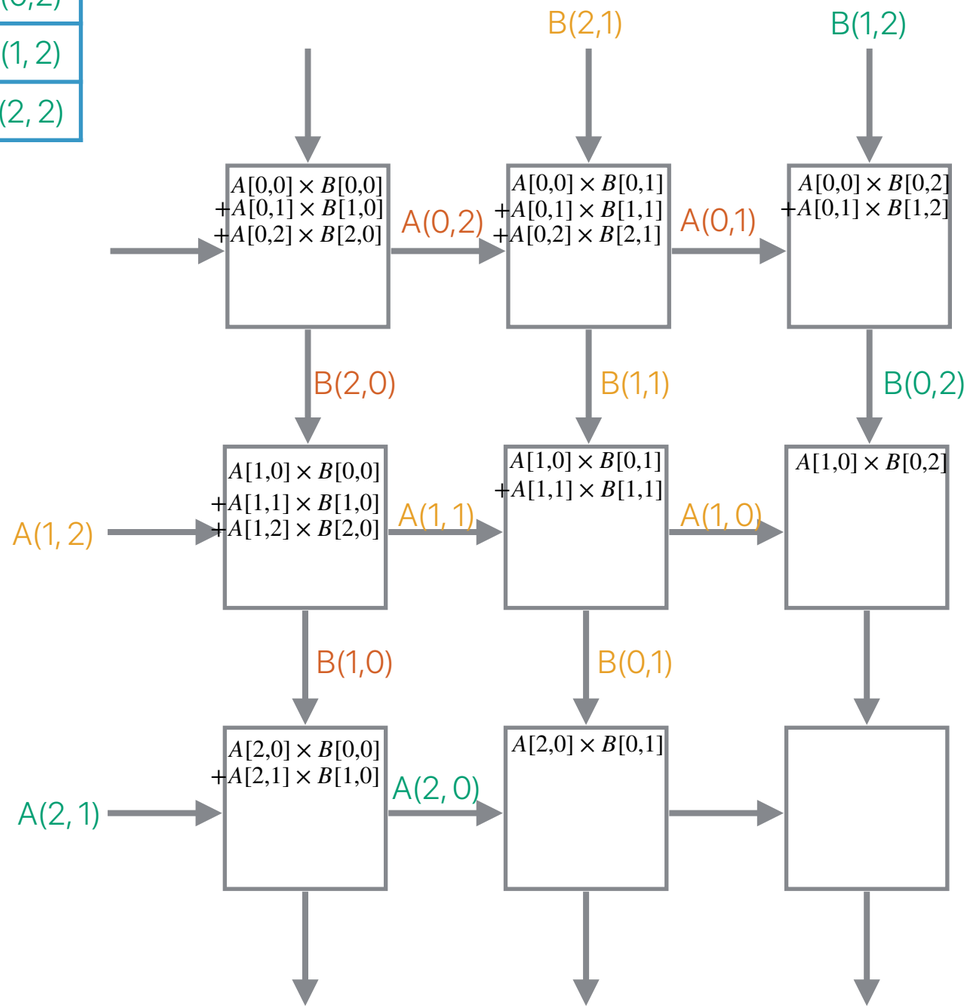
#4

B

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

A

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



H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

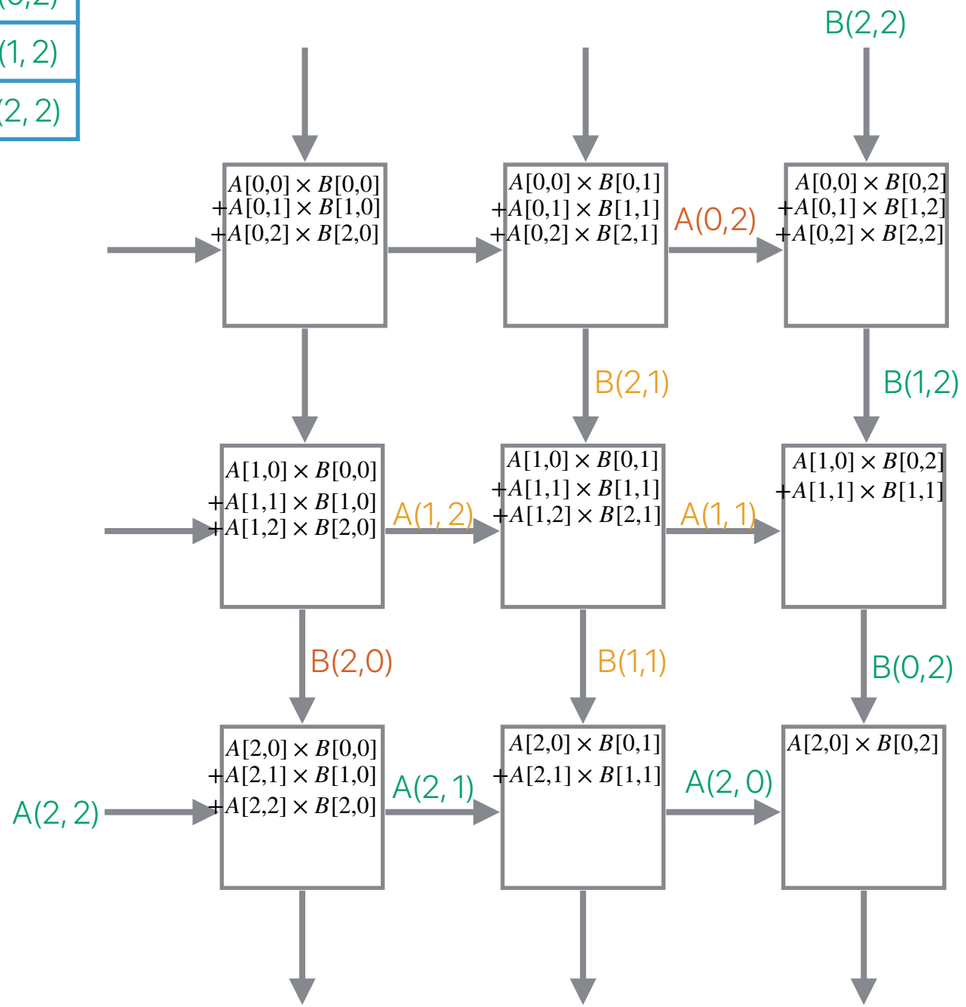
#5

B

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

A

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



H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

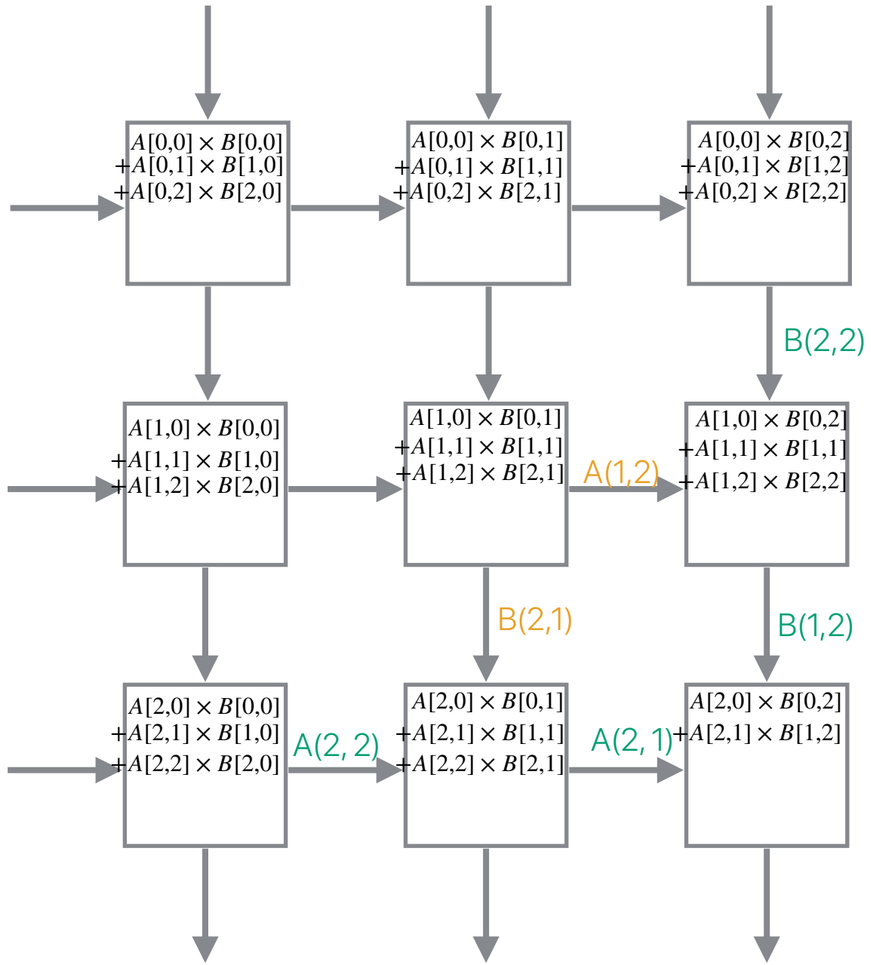
#6

B

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

A

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



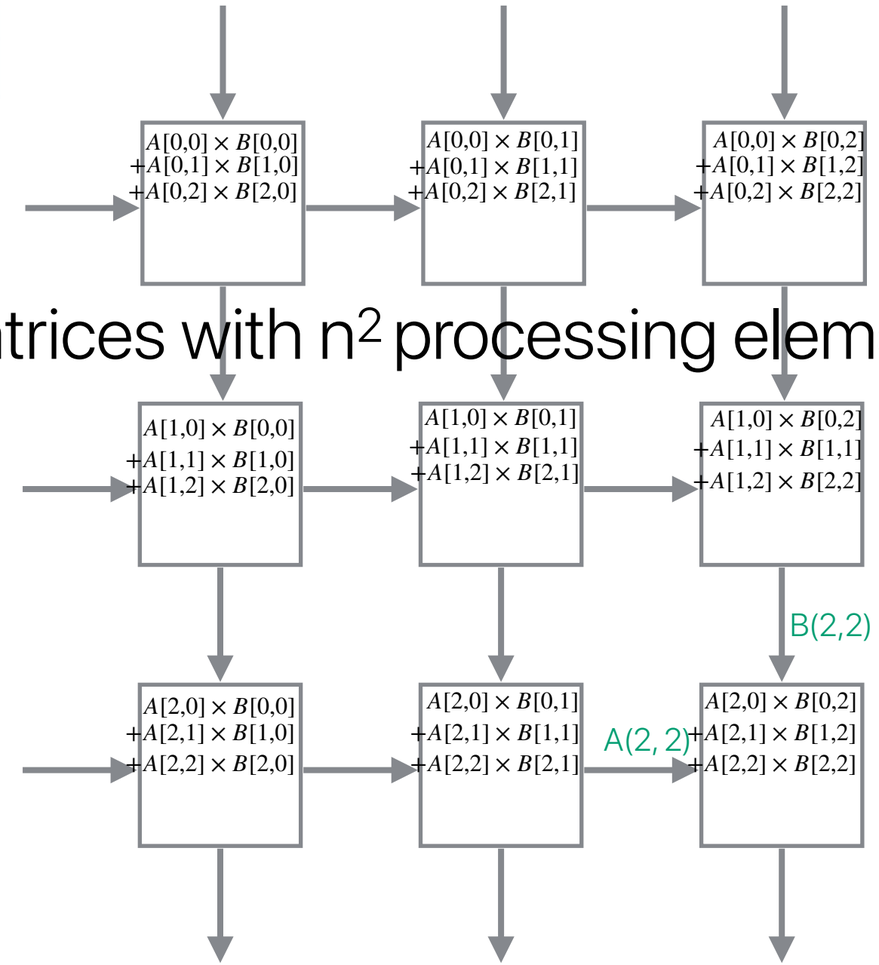
H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Systolic Arrays used by AI/ML Accelerators

#7

B

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



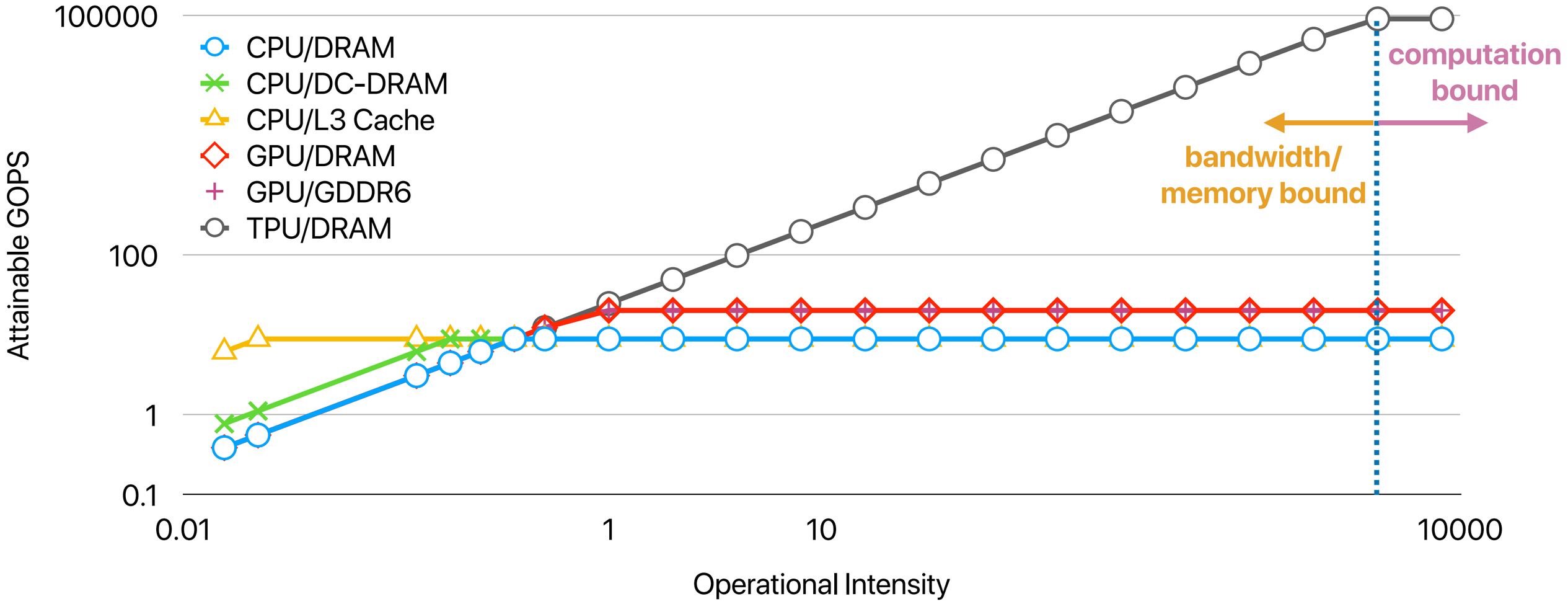
For n^2 matrices with n^2 processing elements: $3 \times n - 2$

A

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

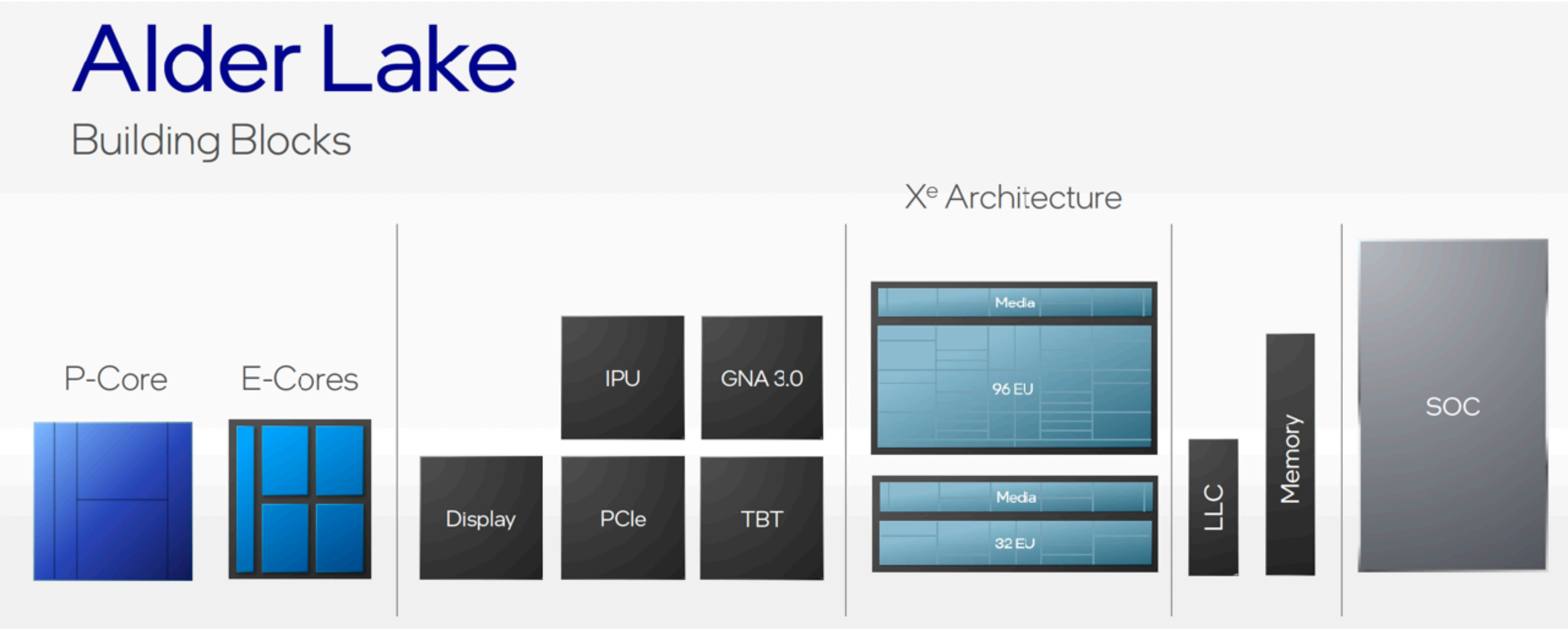
H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

The roofline after using hardware accelerators

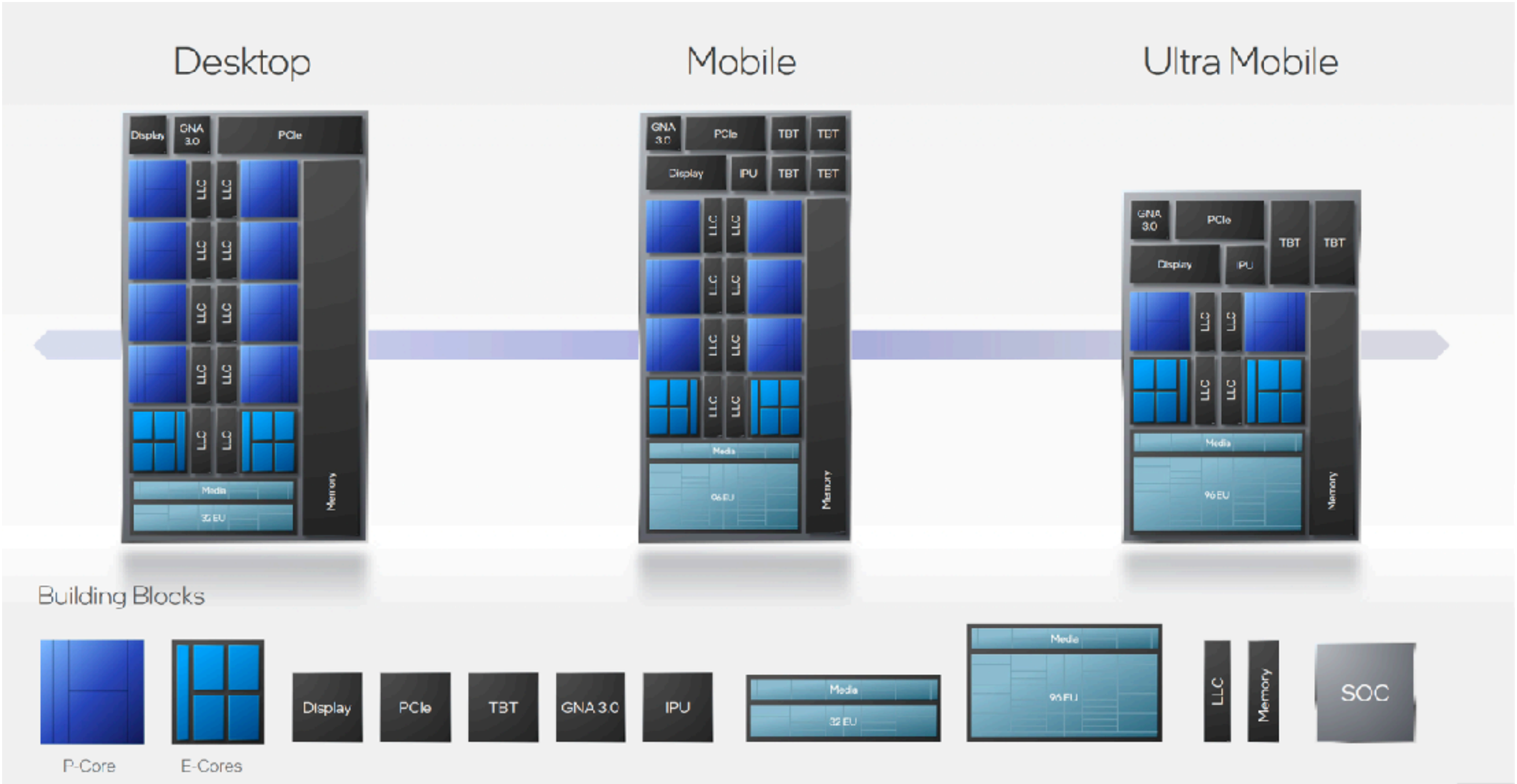


Example of heterogeneous “processors”

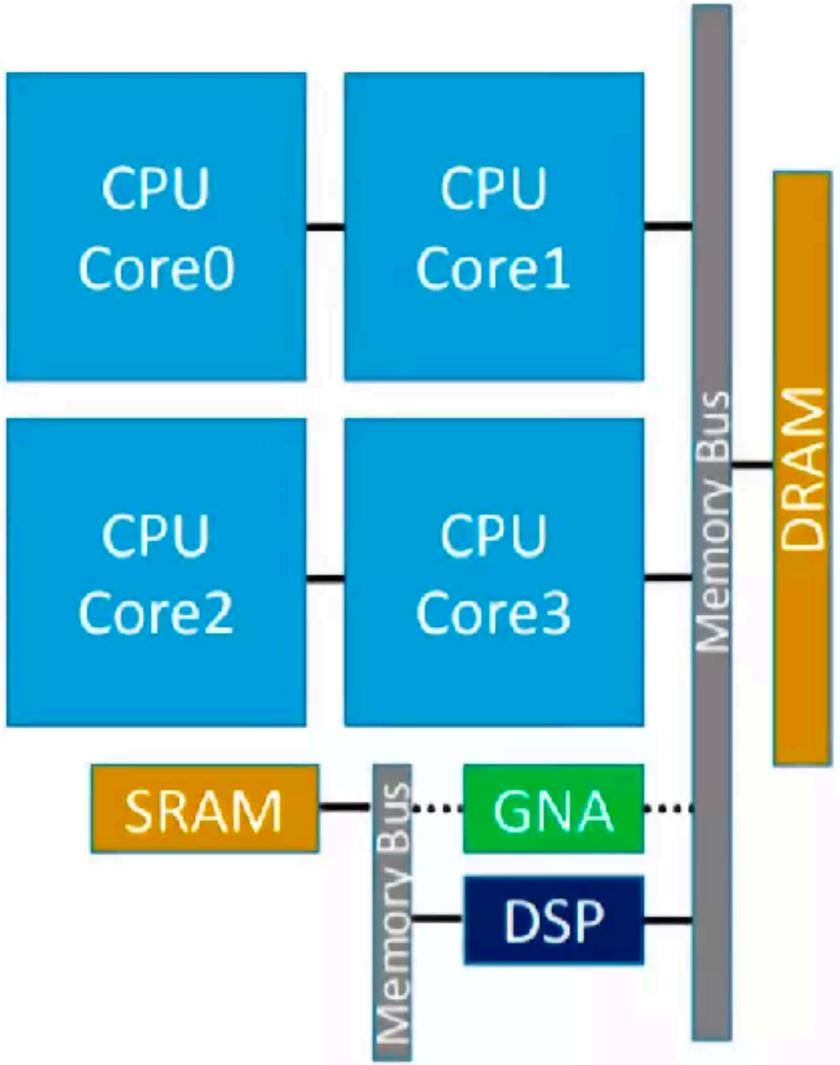
The intel Alder Lake



The intel Alder Lake



intel processor architecture



The NVIDIA Hooper Architecture

NVIDIA RTX 3090

3:00

Which one do you prefer?
A. Single chip, heterogeneous processor (e.g., NVIDIA/intel) or
B. standalone accelerators? (e.g., TPU)

Single-chip or standalone ones?

Is single-chip, heterogeneous processor a good idea?

- Potentially reduce the system interconnect bandwidth demand (still need to use intra-chip interconnect)
- Reduce the total system size and cost
- The design of the memory controller is complicated. For example, CPUs are more latency sensitive and want pairs of scalar values. GPUs/accelerators are more bandwidth demanding and want pairs of vectors
  - J. Power, M. D. Hill and D. A. Wood, "Supporting x86-64 address translation for 100s of GPU lanes," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014, pp. 568-578
- The power consumption per-chip is still limiting the overall performance
  - You can have only moderate processor cores with moderate GPU cores
  - You can have performance processor cores with wimpy graphic cores
  - You have to dynamically switch each part on/off
- Or you have to increase power consumption — NVIDIA's H100 goes up to 700W!!!

Is system-wide heterogeneous processing a better idea?

- Easier for heat dissipation
  - Each processor can be more powerful
    - gamers/datacenters still prefers discrete GPUs
    - Cloud TPUs are standalone ones
- System size is larger, cost of ownership can be higher
- Data movement going through system interconnects

3:00

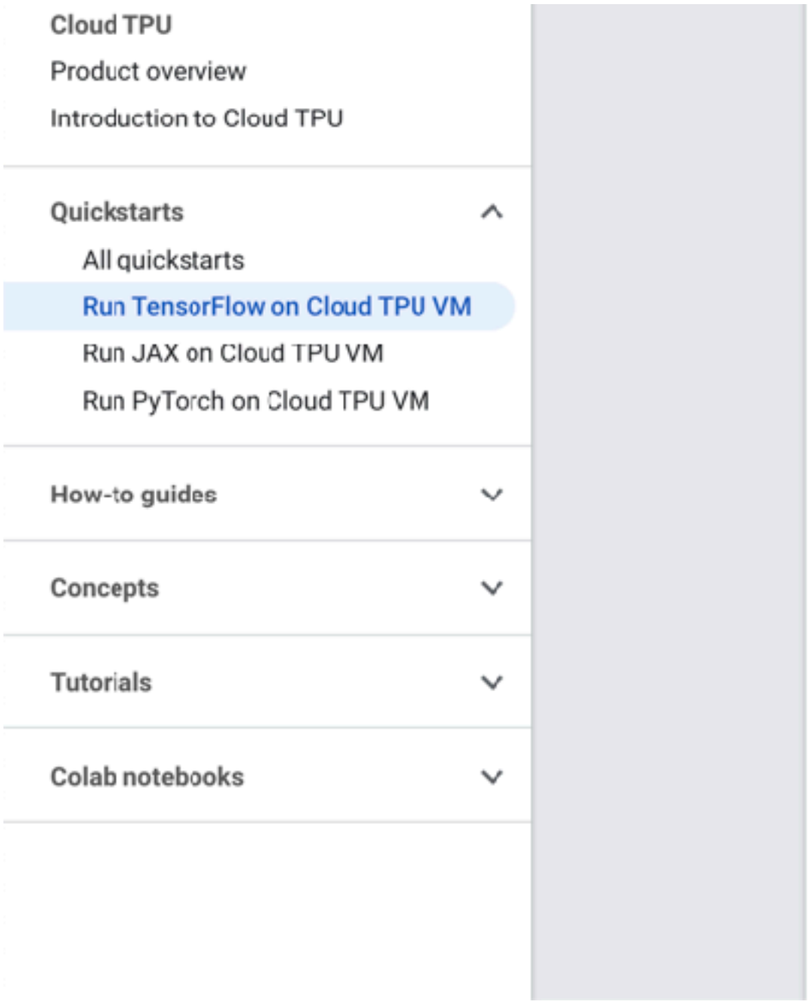
If you design an accelerator, how're you going to let programmer access accelerator's features?

From the TPU paper

Pitfall: Being Ignorant of Architecture History when Designing a DSA.

Ideas that didn't fly for general-purpose computing might be ideal for DSAs. For the TPU, three important architectural ideas date from the early 1980s: systolic arrays,[5] decoupled-access/execute,[4] and CISC instructions. The first reduced the area and power of the large matrix multiply unit, the second fetches weights concurrently during operation of the matrix multiply unit, and the third better utilizes the limited bandwidth of the PCIe bus for delivering instructions. History-aware architects could have a competitive edge in this DSA era.

TPU Programming model — domain specific languages



Note: For single TPUs, Slices, and Pods, pass your TPU name to `TPUClusterResolver()`.

```
import tensorflow as tf
print("Tensorflow version " + tf.__version__)

tpu = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='your-tpu-name')  # TPU detection
print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])

tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)
strategy = tf.distribute.experimental.TPUStrategy(tpu)

@tf.function
def add_fn(x,y):
    z = x + y
    return z

x = tf.constant(1.)
y = tf.constant(1.)
z = strategy.run(add_fn, args=(x,y))
print(z)
```

Programming interface — NV's WMMA

```
__global__ void WMMAF16TensorCore(half *A, half *B, float *C, float *D) {

     int ix = (blockIdx.x * blockDim.x + threadIdx.x)/WARP_SIZE;
     int iy = (blockIdx.y * blockDim.y + threadIdx.y);

     wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a_frag;
     wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b_frag;
     wmma::fragment<wmma::accumulator, M, N, K, float> ab_frag;
     wmma::fragment<wmma::accumulator, M, N, K, float> c_frag;

     wmma::fill_fragment(ab_frag, 0.0f);

     // AB = A*B

     int a_col, a_row, b_col, b_row, c_col, c_row;

     a_row = ix * M;
     b_row = iy * N;

     for (int k=0; k<K_TOTAL; k+=K) {
          a_col = b_col = k;

          if (a_row < M_TOTAL && a_col < K_TOTAL && b_row < K_TOTAL && b_col <
N_TOTAL) {
               // Load the inputs
               wmma::load_matrix_sync(a_frag, A + a_col + a_row * M_TOTAL,
M_TOTAL);
               wmma::load_matrix_sync(b_frag, B + b_col + b_col * K_TOTAL,
K_TOTAL);

               // Perform the matrix multiplication
               wmma::mma_sync(ab_frag, a_frag, b_frag, ab_frag);
          }
     }

     // D = AB + C
     c_col = b_row;
     c_row = a_row;

     if (c_row < M_TOTAL && c_col < N_TOTAL) {
          wmma::load_matrix_sync(c_frag, C + c_col + c_row * N_TOTAL, N_TOTAL,
wmma::mem_row_major);

          for (int i = 0; i < c_frag.num_elements; i++) {
               c_frag.x[i] = ab_frag.x[i] + c_frag.x[i];
          }
          // Store the output
          wmma::store_matrix_sync(D + c_col + c_row * N_TOTAL, c_frag,
N_TOTAL, wmma::mem_row_major);
     }
}
```

intel GNA

```
// Open selected  device
status = Gna2DeviceOpen(deviceIndex);
CleanupOrError(status, deviceIndex, nullptr, GNA2_DISABLED, GNA2_DISABLED,
               "main", "Gna2DeviceOpen()");

/* Calculate model memory parameters for GnaAlloc. */
int buf_size_weights = Gna2RoundUpTo64(sizeof(weights));
// note that buffer alignment to 64-bytes is required by GNA HW
int buf_size_inputs = Gna2RoundUpTo64(sizeof(inputs));
int buf_size_biases = Gna2RoundUpTo64(sizeof(biases));
int buf_size_outputs = Gna2RoundUpTo64(H * B * 4); // (4 out vectors, H elems in each one,
4-byte elems)

auto rw_buffer_size = Gna2RoundUp(buf_size_inputs + buf_size_outputs, 0x1000);
auto bytes_requested = rw_buffer_size + buf_size_weights + buf_size_biases;

// Allocate GNA memory (obtains pinned memory shared with the device)
uint32_t bytes_granted;
void* memory = nullptr;
status = Gna2MemoryAlloc(bytes_requested, &bytes_granted, &memory);

/* Prepare model memory layout. */
auto model_memory = reinterpret_cast<uint8_t*>(memory);

auto rw_buffers = model_memory;

auto pinned_inputs = reinterpret_cast<int16_t*>(rw_buffers);
memcpy_s(pinned_inputs, buf_size_inputs, inputs, sizeof(inputs)); // puts the inputs into
the pinned memory
rw_buffers += buf_size_inputs; // fast-forwards current pinned memory pointer to the next
free block

auto pinned_outputs = reinterpret_cast<int32_t*>(rw_buffers);
rw_buffers += buf_size_outputs; // fast-forwards the current pinned memory pointer by the
space needed for outputs

model_memory += rw_buffer_size;
auto weights_buffer = reinterpret_cast<int16_t*>(model_memory);
memcpy_s(weights_buffer, buf_size_weights, weights, sizeof(weights)); // puts the weights
into the pinned memory
model_memory += buf_size_weights; // fast-forwards current pinned memory pointer to the
next free block

auto biases_buffer = reinterpret_cast<int32_t*>(model_memory);
memcpy_s(biases_buffer, buf_size_biases, biases, sizeof(biases)); // puts the biases into
the pinned memory
model_memory += buf_size_biases; // fast-forwards current pinned memory pointer to the
next free block

/* Prepare neural network topology,
 * Single FullyConnectedAffine layer in this example. */
```

```
/* Prepare and initialize FullyConnectedAffine operation operands with GNA API helpers */
auto inputTensor = Gna2TensorInit2D(W, B, Gna2DataTypeInt16, pinned_inputs);
auto outputTensor = Gna2TensorInit2D(H, B, Gna2DataTypeInt32, pinned_outputs);
auto weightTensor = Gna2TensorInit2D(H, W, Gna2DataTypeInt16, weights_buffer);
auto biasTensor = Gna2TensorInit1D(H, Gna2DataTypeInt32, biases_buffer);
auto activationTensor = Gna2TensorInitDisabled();

/* Create single FullyConnectedAffine operation (layer) */
auto operation = Gna2Operation{};
status = Gna2OperationInitFullyConnectedAffine(&operation, customAlloc,
                                               &inputTensor, &outputTensor, &weightTensor,
&biasTensor,
                                               &activationTensor);

/* Create data-flow model with single operation (layer) */
Gna2Model model = {1, &operation};
uint32_t modelId = GNA2_DISABLED;
status = Gna2ModelCreate(deviceIndex, &model, &modelId);

// Create request configuration used for queueing inference requests
uint32_t configId = GNA2_DISABLED;
status = Gna2RequestConfigCreate(modelId, &configId);

// Set model input data buffer, operation 0, operand 0, for this sample
status = Gna2RequestConfigSetOperandBuffer(configId, 0, 0, pinned_inputs);

// Set model output data buffer, operation 0, operand 1, for this sample
status = Gna2RequestConfigSetOperandBuffer(configId, 0, 1, pinned_outputs);

// [optional] Set acceleration mode automatic (software emulation used if no hardware
detected)
status = Gna2RequestConfigSetAccelerationMode(configId, Gna2AccelerationModeAuto);

// Enqueue inference request (non-blocking call)
uint32_t requestId; // this gets filled with the actual id later on
status = Gna2RequestEnqueue(configId, &requestId);

/
**********************************************************************************
********
 * Offload effect: other calculations can be done on CPU here, while model inference runs
on GNA HW *

**********************************************************************************
********/

// Wait for inference request completion (blocks until the results are ready)
uint32_t timeout = 1000;
status = Gna2RequestWait(requestId, timeout);
CleanupOrError(status, deviceIndex, memory, modelId, configId,
               "main", "Gna2RequestWait(1000)");
```

3:00

With hardware accelerators, what inspirations do you have in mind in application design?

Inspirations?

Inspirations?

- Using hardware accelerators for applications outside their original domains
  - Using TPUs/Tensor Cores/Ray Tracing hardware for “other things”
- Transforming an application as a problem in the target domain
  - Making an ML model for a specific algorithm problem
- Each of these could be a potential project idea or a topic you select as presentation

Other references

- Using TPUs to accelerate
  - FFT
    - Lu, Tianjian, Thibault Marin, Yue Zhuo, Yi-Fan Chen, and Chao Ma. "Nonuniform fast Fourier transform on TPUs." In 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), pp. 783-787. IEEE, 2021.
    - Lu, Tianjian, Thibault Marin, Yue Zhuo, Yi-Fan Chen, and Chao Ma. "Accelerating MRI reconstruction on TPUs." In 2020 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-9. IEEE, 2020.
  - Database operations
    - Pedro Holanda and Hannes Mühleisen. 2019. Relational Queries with a Tensor Processing Unit. In Proceedings of the 15th International Workshop on Data Management on New Hardware (DaMoN'19). Association for Computing Machinery, New York, NY, USA, Article 19, 1–3. DOI:https://doi.org/10.1145/3329785.3329932
  - Quantum simulation
    - Morningstar, Alan, Markus Hauru, Jackson Beall, Martin Ganahl, Adam GM Lewis, Vedika Khemani, and Guifre Vidal. "Simulation of quantum many-body dynamics with Tensor Processing Units: Floquet prethermalization." arXiv preprint arXiv:2111.08044 (2021).
    - Pederson, Ryan, John Kozlowski, Ruyi Song, Jackson Beall, Martin Ganahl, Markus Hauru, Adam GM Lewis, Shrestha Basu Mallick, Volker Blum, and Guifre Vidal. "Tensor Processing Units as Quantum Chemistry Supercomputers." arXiv preprint arXiv:2202.01255 (2022).
  - General-purpose matrix algebra
    - Kuan-Chieh Hsu and Hung-Wei Tseng. Accelerating Applications using Edge Tensor Processing Units. In The International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2021.

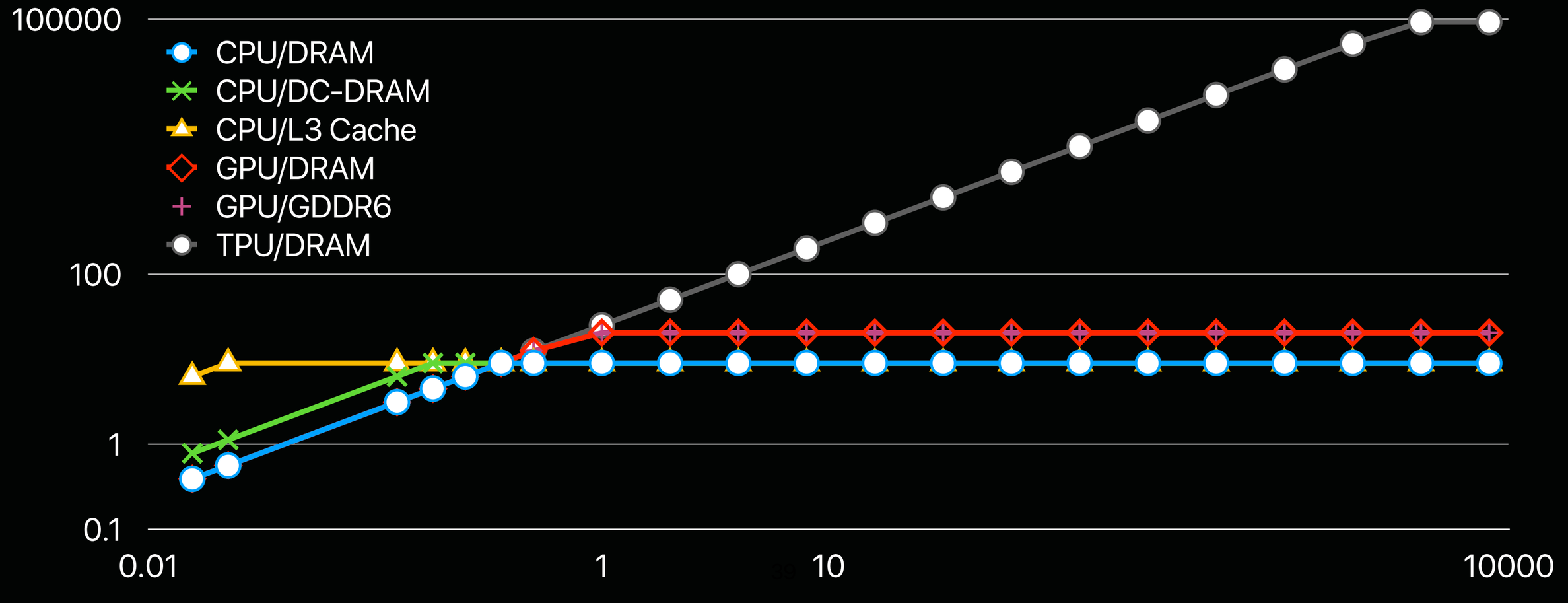
Other references

- Using Tensor Cores to accelerate
  - FFT
    - B. Li, S. Cheng and J. Lin, "tcFFT: A Fast Half-Precision FFT Library for NVIDIA Tensor Cores," in 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 2021 pp. 1-11. doi: 10.1109/Cluster48925.2021.00035
    - Durrani, Sultan and Chughtai, Muhammad Saad and Hidayetoglu, Mert and Tahir, Rashid and Dakkak, Abdul and Rauchwerger, Lawrence and Zaffar, Fareed and Hwu, Wen-mei. Accelerating Fourier and Number Theoretic Transforms using Tensor Cores and Warp Shuffles," 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2021, pp. 345-355, doi: 10.1109/PACT52795.2021.00032.
  - Database operations
    - Abdul Dakkak, Cheng Li, Jinjun Xiong, Isaac Gelado, and Wen-mei Hwu. 2019. Accelerating reduction and scan using tensor core units. In Proceedings of the ACM International Conference on Supercomputing (ICS '19). Association for Computing Machinery, New York, NY, USA, 46–57. DOI:https://doi.org/10.1145/3330345.3331057
    - Yu-Ching Hu, Yuliang Li and Hung-Wei Tseng. TCUDB: Accelerating Database with Tensor Processors. In the 2022 ACM SIGMOD/PODS International Conference on Management of Data, SIGMOD 2022, 2022
    - Dong He, Supun Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos and Matteo Interlandi. Query Processing on Tensor Computation Runtimes. https://doi.org/10.48550/arxiv.2203.01877
  - Crypyto
    - Lee, Wai-Kong, Hwajeong Seo, Zhenfei Zhang, and Seong Oun Hwang. "TensorCrypto: High Throughput Acceleration of Lattice-based Cryptography Using Tensor Core on GPU." IEEE Access (2022).

Other references

- Using Ray Tracing Hardware for
  - J. Salmon and S. McIntosh-Smith, "Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC," 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2019, pp. 19-29, doi: 10.1109/PMBS49563.2019.00008.
  - Yuhao Zhu, RTNN: Accelerating Neighbor Search Using Hardware Ray Tracing. In PPoPP 2022
- Training an algorithm as a neural network model
  - Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45). IEEE Computer Society, USA, 449–460. DOI:https://doi.org/10.1109/MICRO.2012.48

What do you learn from the "roofline" model?



Lessons learned from accelerators?

Lessons learned from accelerators?

- Accelerators “lift up” the roofline
  - Applications/compute kernels with higher arithmetic densities may be feasible
    - NN is feasible after GPGPU
    - Trade “complexity” with parallelism
  - Applications are more likely to be memory-bound
  - Your software should try to avoid frequent memory access
    - Try to use memory closer to the processing elements
  - The hardware design must not ignore the importance of memory bandwidth
- The most “efficient” system design must land on the “turning point” of your roofline model
  - TPU’s 167GB/sec memory bandwidth is an example

Electrical
Computer Science
Engineering

277

つづく