The New Golden Age of Computer Architecture

Hung-Wei Tseng

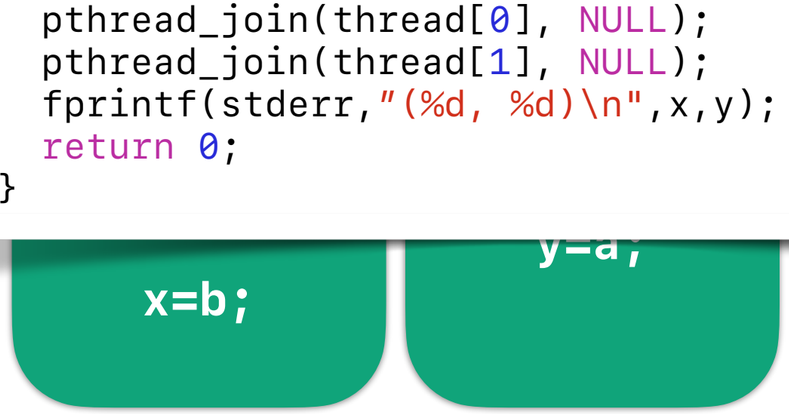
Recap: 4Cs of cache misses

- 3Cs:
  - Compulsory, Conflict, Capacity
- Coherency miss:
  - A “block” invalidated because of the sharing among processors.
- True sharing
  - Processor A modifies X, processor B also want to access X.
- False sharing
  - Processor A modifies X, processor B also want to access Y. However, Y is invalidated because X and Y are in the same block!

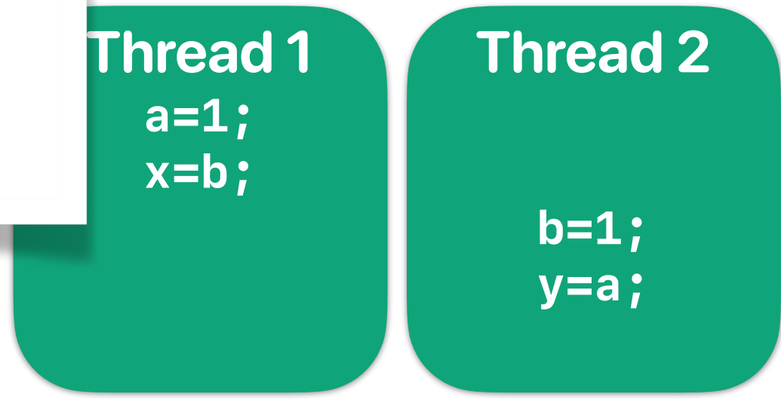
Recap: Possible scenarios

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

volatile int a,b;
volatile int x,y;
volatile int f;
void* modifya(void *z) {
  a=1;
  x=b;
  return NULL;
}
void* modifyb(void *z) {
  b=1;
  y=a;
  return NULL;
}
```

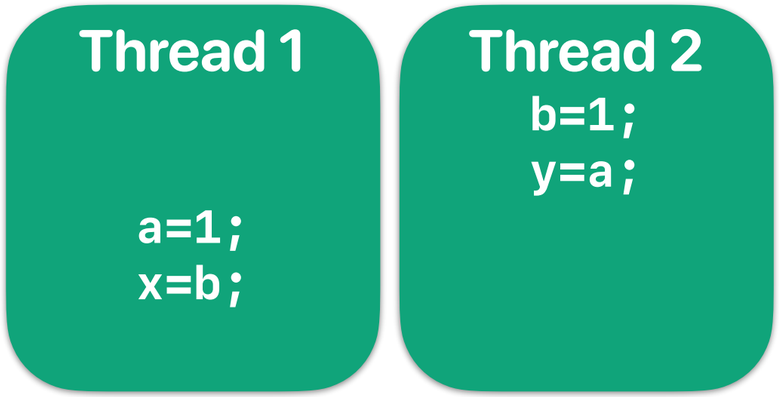
```c
int main() {
  int i;
  pthread_t thread[2];
  pthread_create(&thread[0], NULL, modifya, NULL);
  pthread_create(&thread[1], NULL, modifyb, NULL);
  pthread_join(thread[0], NULL);
  pthread_join(thread[1], NULL);
  fprintf(stderr,"(%d, %d)\n",x,y);
  return 0;
}
```



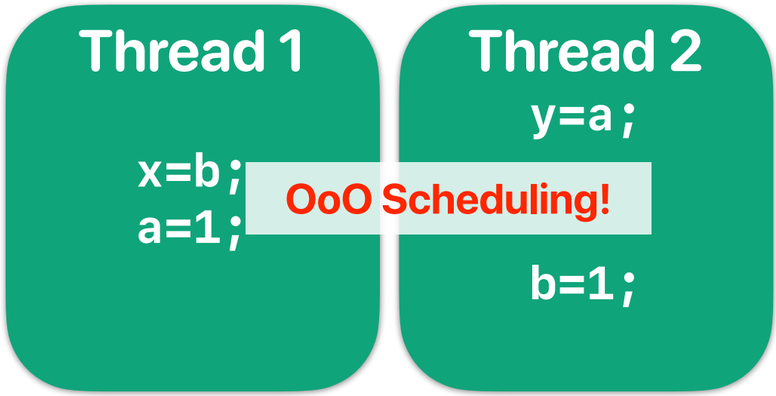
(1,1)



(0,1)



(1,0)

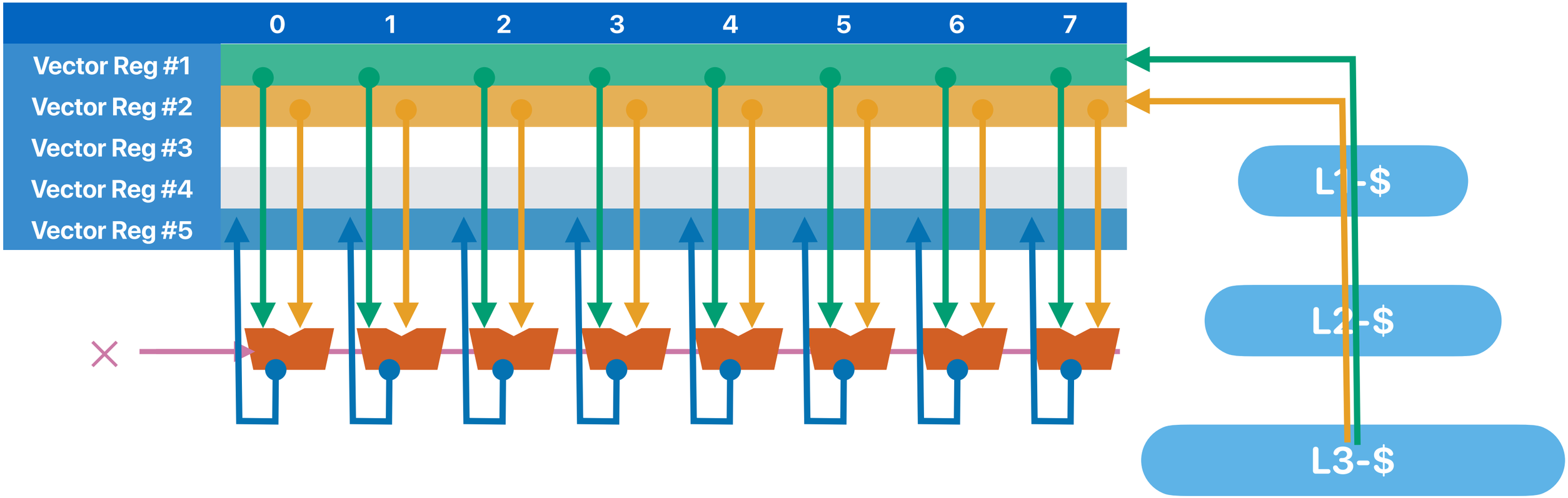


(0,0)

Recap: Parallelisms

- Instruction-level parallelism — perform various, independent instructions simultaneously
  - Pipeline
  - OoO/Superscalar
- Thread-level parallelism — perform independent computation streams (composed of many instructions or SIMD instructions)
  - Multicore/SMT processors

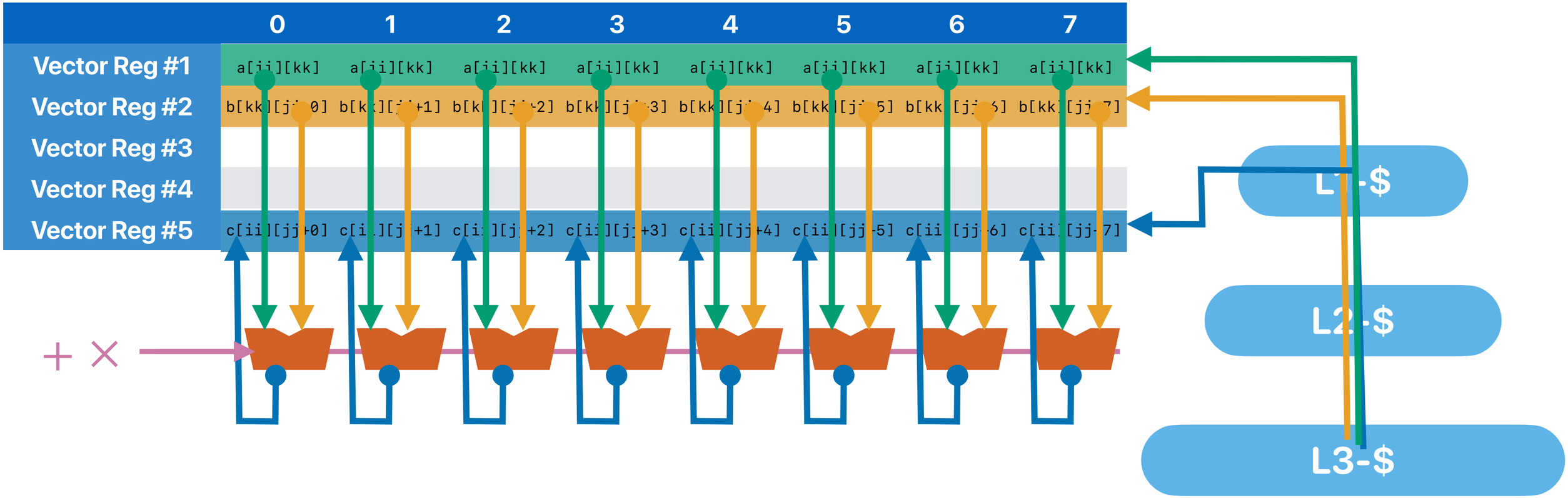
Recap: Vector processing architecture



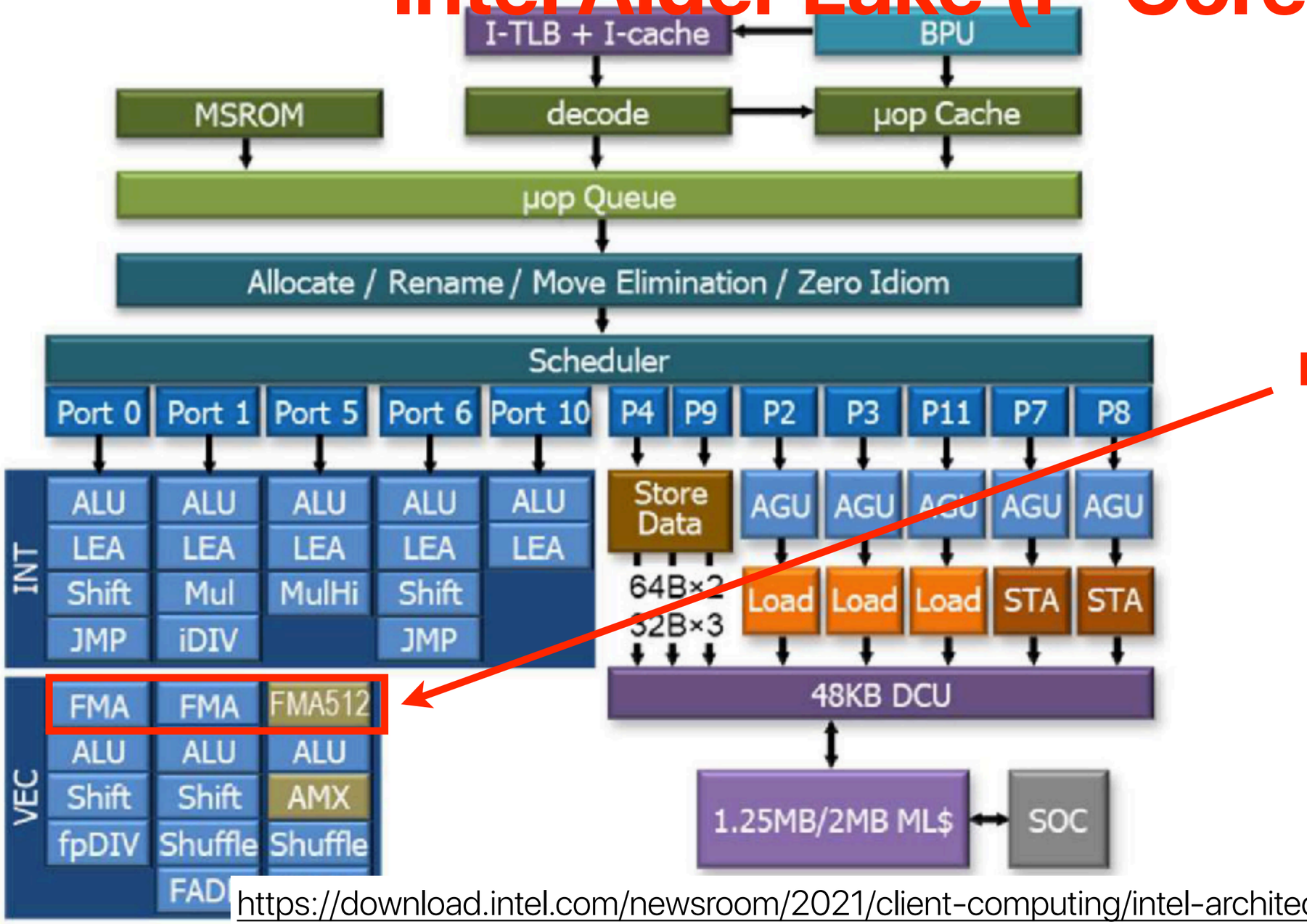
Vectorized matrix multiplications

```
#define VECTOR_WIDTH 8 // VECTOR_WIDTH is 8 as we're using AVX-512
void vector_blockmm(double **a, double **b, double **c, \
uint64_t tile_size) {
  int i,j,k, ii, jj, kk, x;
  __m512d va, vb, vc;
  for(i = 0; i < ARRAY_SIZE; i+=tile_size) {
    for(j = 0; j < ARRAY_SIZE; j+=tile_size) {
      for(k = 0; k < ARRAY_SIZE; k+=tile_size) {
        for(ii = i; ii < i+tile_size; ii++) {
          for(jj = j; jj < j+tile_size; jj+=VECTOR_WIDTH) {
             vc = _mm512_load_pd(&c[ii][jj]); // load c[ii][jj] to c[ii][jj+7] to vc[0-7]
             for(kk = k; kk < k+tile_size; kk++) {
                va = _mm512_broadcastsd_pd(&a[ii][kk]);// load a[ii][kk] to va[0-7]
                vb = _mm512_load_pd(&b[kk][jj]);  // load b[kk][jj] to b[kk][jj+7] to vb[0-7]
                vc = _mm512_add_pd(vc,_mm512_mul_pd(va,vb));    // vc += va * vb
             }
             _mm512_store_pd(&c[ii][jj],vc);      // store vc to c[ii][jj] to c[ii][jj+4]
          }
        }
      }
    }
  }
}
```

Vector processing architecture



Intel Alder Lake (P-Core)



AVX Vector Processing Units

https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

NVIDIA CUDA GPU function

```
__global__ void gpu_matrix_mult(D_TYPE *a,D_TYPE *b,
D_TYPE *c, int size)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    D_TYPE sum = 0;
    for(int i = 0; i < size; i++) {
            sum += a[row * size + i] * b[i * size + col];
    }
    c[row * size + col] = sum;
}
```

Recap: Parallelisms

- Instruction-level parallelism — perform various, independent instructions simultaneously
  - Pipeline
  - OoO/Superscalar
- Data-level parallelism — perform the same operation on multiple data elements in parallel
  - SIMD instructions
  - Compute within an SM in GPUs
- Thread-level parallelism — perform independent computation streams (composed of many instructions or SIMD instructions)
  - Multicore/SMT processors
  - Compute using multiple SMs in GPUs

Power consumption & power density

- $P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$
  - α: average switches per cycle
  - C: capacitance
  - V: voltage
  - f: frequency, usually linear with V
  - N: the number of transistors
- $P_{leakage} \sim N \times V \times e^{-V_t}$
  - N: number of transistors
  - V: voltage
  - V_t: threshold voltage where transistor conducts (begins to switch)
- Power density:

$$P_{density} = \frac{P}{area}$$

Lower the frequency can reduce active power

Moore's Law allows higher frequencies as transistors are smaller

Moore's Law makes this smaller

Recap: What happens if power doesn't scale with process technologies?

- If we are able to cram more transistors within the same chip area (Moore's law continues), but the power consumption per transistor remains the same. Right now, if put more transistors in the same area because the technology allows us to. How many of the following statements are true?
  - ① The power consumption per chip will increase
  - ② The power density of the chip will increase
  - ③ Given the same power budget, we may not able to power on all chip area if we maintain the same clock rate **— even if we put more cores, we cannot have all of them on!**
  - ④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area **— or we have to operate all of them at a slower speed**

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

GPUs still suffer from broken Dennard Scaling

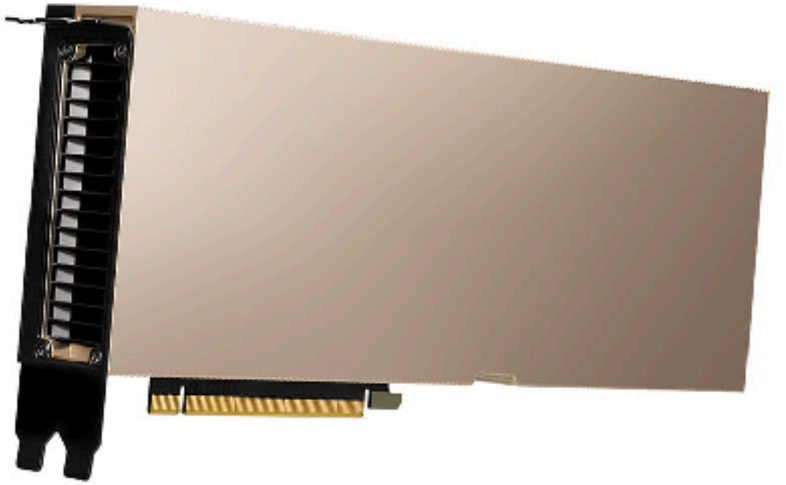
NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
GPU	GH100 (814mm2)	GA100 (826mm2)	GV100 (815mm2)
Transistor Count	80B	54.2B	21.1B
TDP	700W	400W	300W/350W
Manufacturing Process	TSMC 4N	TSMC 7N	TSMC 12nm FFN
Interface	SXM5	SXM4	SXM2/SXM3
Architecture	Hopper	Ampere	Volta

1.46x

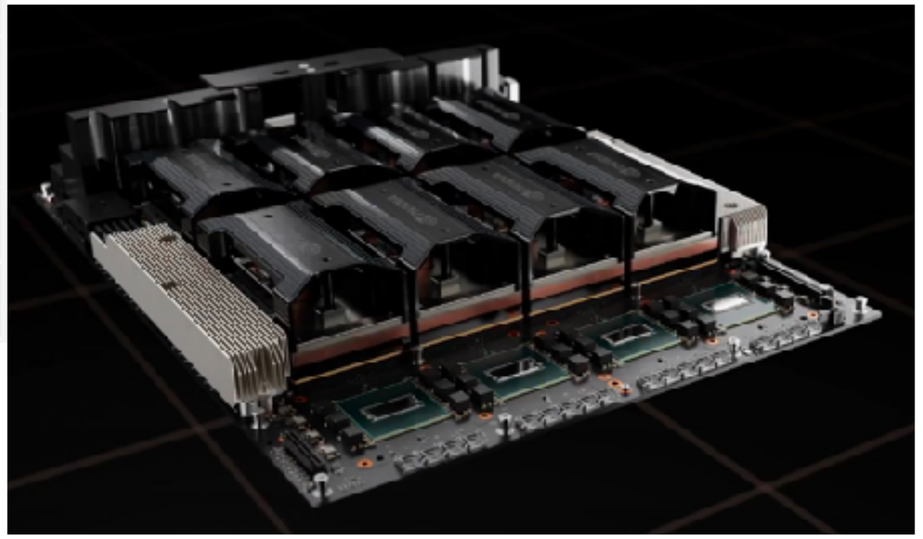
1.75x

8.75 W/ B Transistors

7.38 W/ B Transistors



https://www.workstationspecialist.com/product/nvidia-tesla-a100/



https://www.servethehome.com/wp-content/uploads/2022/03/NVIDIA-GTC-2022-H100-in-HGX-H100.jpg

Recap: Power consumption to light on all transistors

Chip

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

=49W

Dennardian Scaling

Chip

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

=50W

Dennardian Broken

Chip

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

On ~ 50W

Off ~ 0W

Dark!

=100W!

Recap — Take-aways: parallel programming

- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache coherency may create unexpected cache invalidations/misses if you do it wrong
- Processor behaviors are non-deterministic
  - You cannot predict which processor is going faster
  - You cannot predict when OS is going to schedule your thread
  - You cannot predict when the processor is going to schedule an instruction
- Cache consistency is hard to support
- **Even if we can address all programming challenges, multi-core performance has stopped to scale due to the Dark Silicon problem**

Outline

- Challenges and state-of-the-art solutions in the dark silicon era
- The new golden age of Computer Architecture

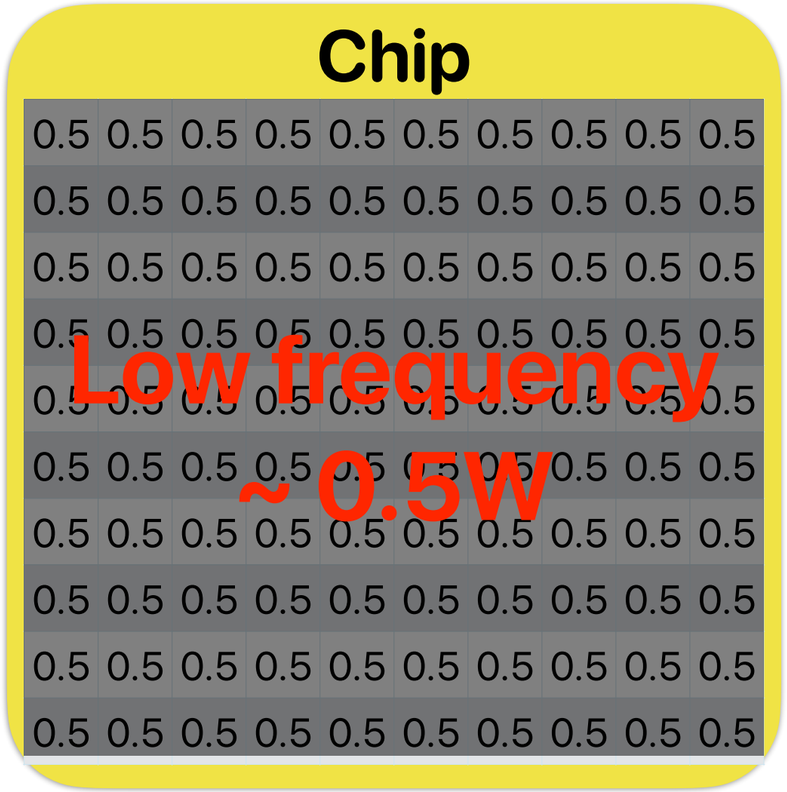
Challenges and state-of-the-art solutions in the dark silicon era

Dark silicon problem

- We are given the same area, twice amount of transistors
- We are given the same power budget
- We are given a certain applications
- How can we maximize the performance for these applications within the given constraints?

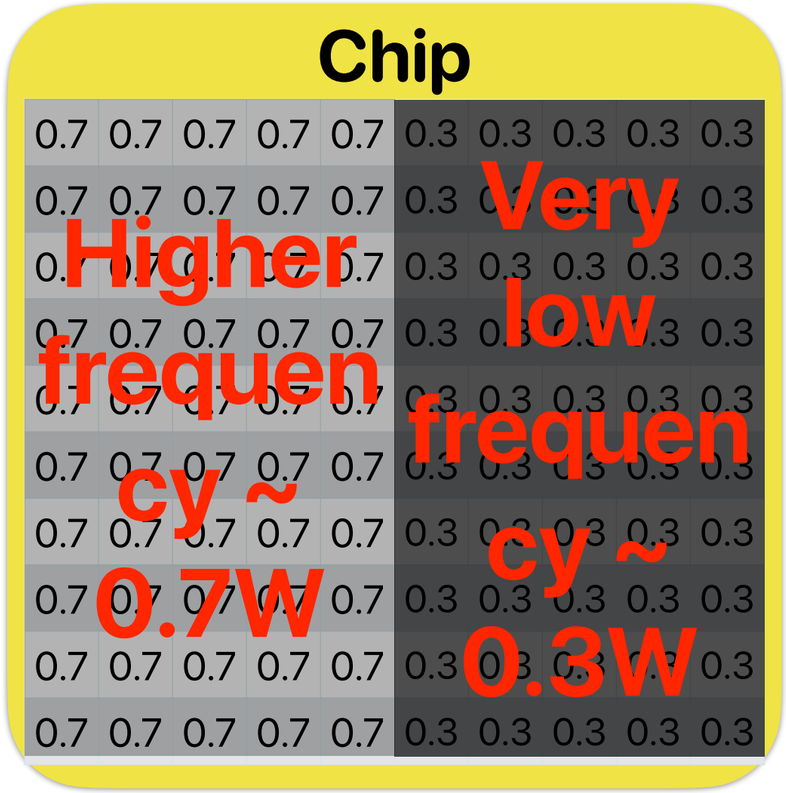
Given the same power budget, maximize the efficiency per chip

Slowdown all of them



=50W!

Some faster, some slower

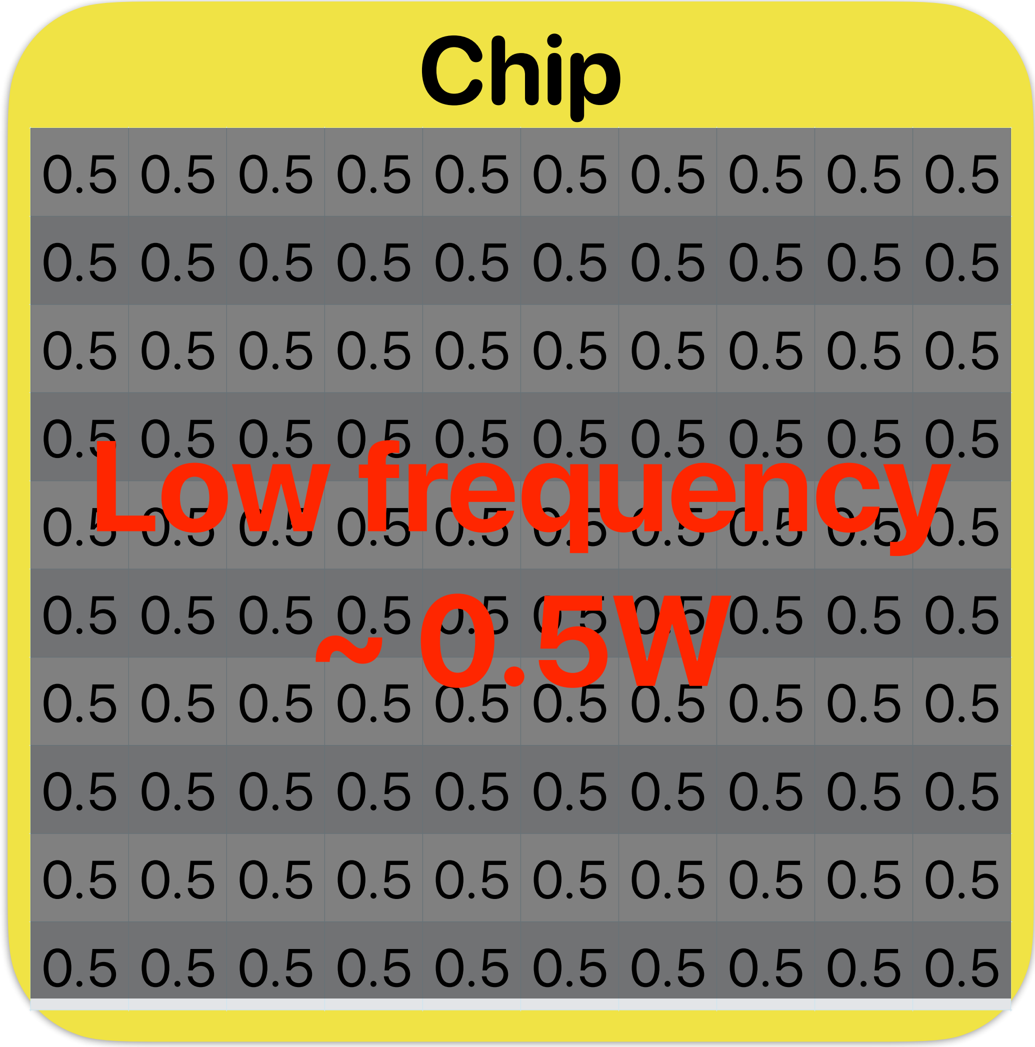


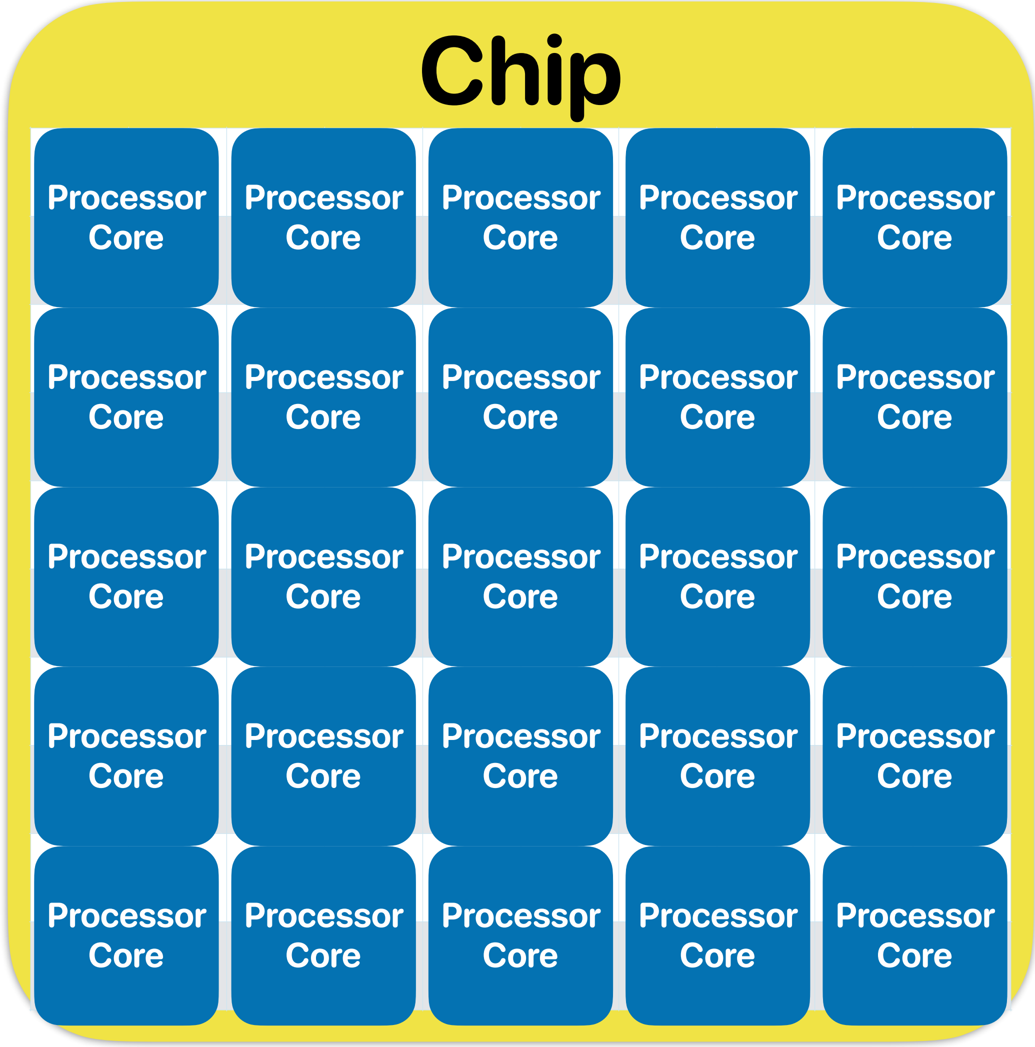
=50W!

Some at top speed, some are not functioning

Given the same power budget, maximize the efficiency per chip

Slowdown all of them





All of them are smaller, simpler, lower-frequency, lower-power cores

=50W!

More cores per chip, slower per core

Export comparison	Intel® Xeon® Platinum 849...	Intel® Xeon® Platinum 846...	Intel® Xeon® Gold 6448H ...	Intel® Xeon® Platinum 844...	Intel® Xeon® Gold 6434H ...
Total Cores	60	48	32	16	8
Total Threads	120	96	64	32	16
Max Turbo Frequency	3.50 GHz	3.80 GHz	4.10 GHz	4.00 GHz	4.10 GHz
Processor Base Frequency	1.90 GHz	2.10 GHz	2.40 GHz	2.90 GHz	3.70 GHz
Cache	112.5 MB	105 MB	60 MB	45 MB	22.5 MB
Intel® UPI Speed	16 GT/s	16 GT/s	16 GT/s	16 GT/s	16 GT/s
Max # of UPI Links	4	4	3	4	3
TDP	350 W	330 W	250 W	270 W	195 W

https://ark.intel.com/content/www/us/en/ark/products/series/228622/4th-generation-intel-xeon-scalable-processors.html

Xeon Phi

Essentials

Product Collection	Intel® Xeon Phi™ 72x5 Processor Family
Code Name	Products formerly Knights Mill
Vertical Segment	Server
Processor Number	7295
Off Roadmap	No
Status	Launched
Launch Date	Q4'17
Lithography	14 nm

Performance

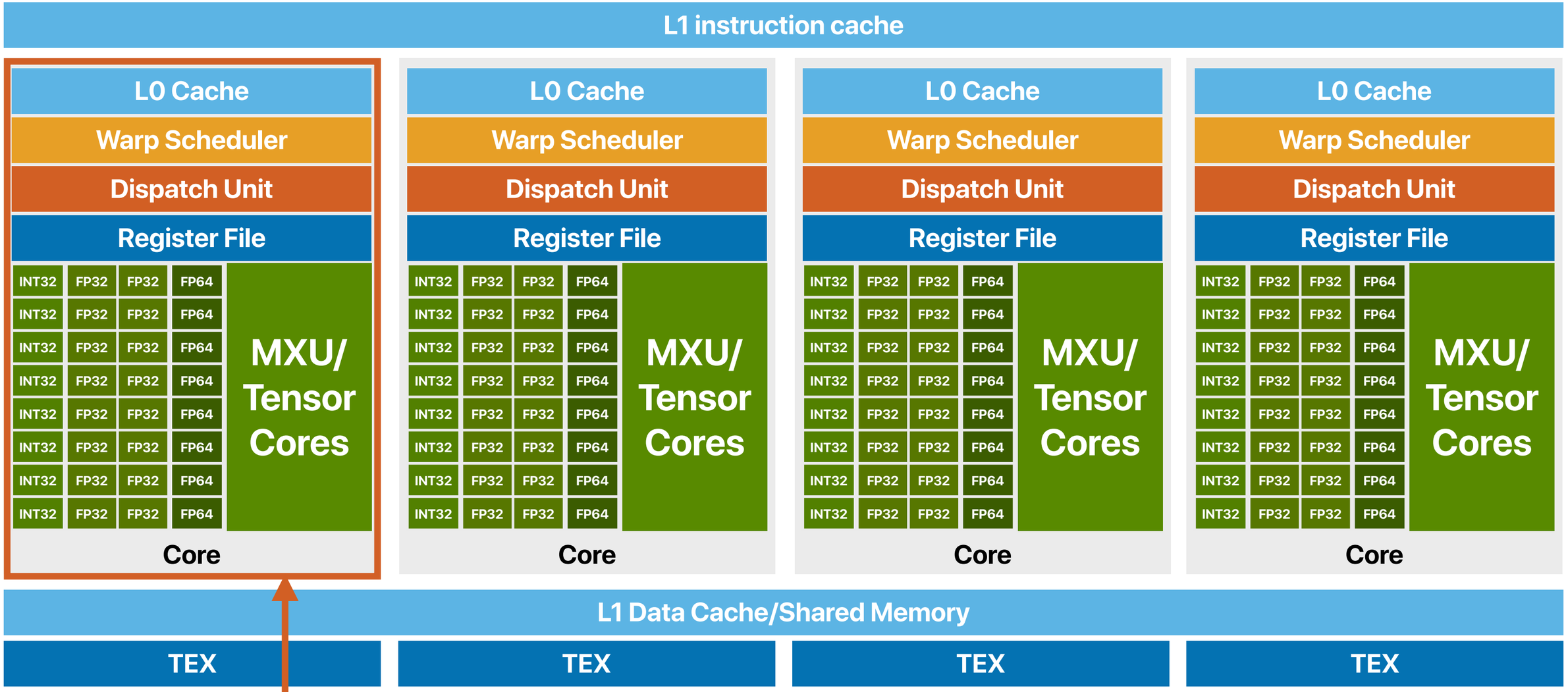
# of Cores	72
# of Threads	72
Processor Base Frequency	1.50 GHz
Max Turbo Frequency	1.50 GHz
Cache	36 MB L2 Cache
TDP	320 W

What's the "appropriate" GPU architecture

- Lots of ALUs to process pixels in parallel — 2M pixels in HD resolution, very regular workloads
  - Vector processing model
- Simple operations
  - The ALUs only supports very few instructions
  - Almost no branches
- Deadline driven and throughput-oriented rather than latency oriented
  - High-bandwidth but also "higher-latency" memory
  - ALUs can be slower

GPU also follows the idea of slower, but more!

GPU Architecture



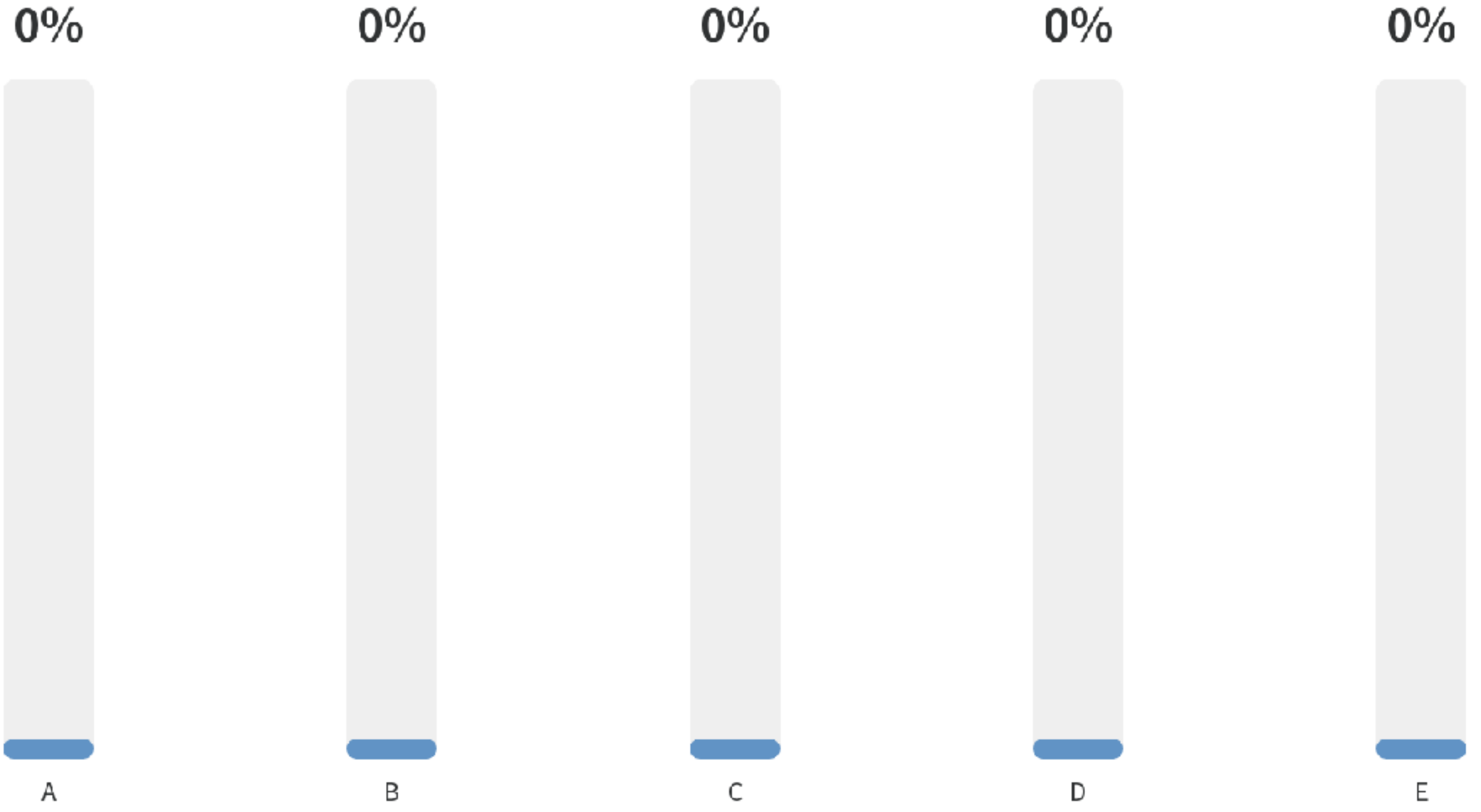
https://www.pollev.com/hungweitseng close in 1:30

Lower frequency

- Regarding lower the operating frequency when running applications, how many of the following statements are correct?
  - ① Lowering the frequency helps the same battery capacity to carry out more tasks
  - ② Lowering the frequency helps reducing the heat generation
  - ③ Lowering the frequency helps reducing the electricity bill/total energy consumption of running the same amount of tasks
  - ④ A CPU operating at 30% of the peak frequency can still consume more than 50% of the peak power

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

0

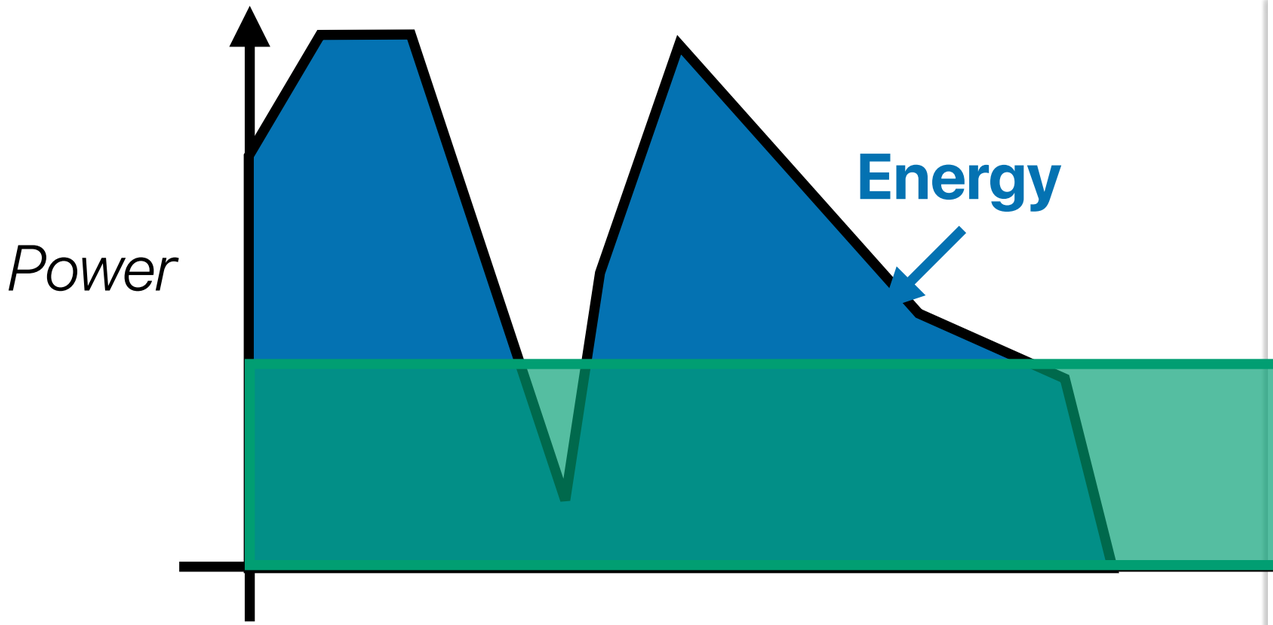


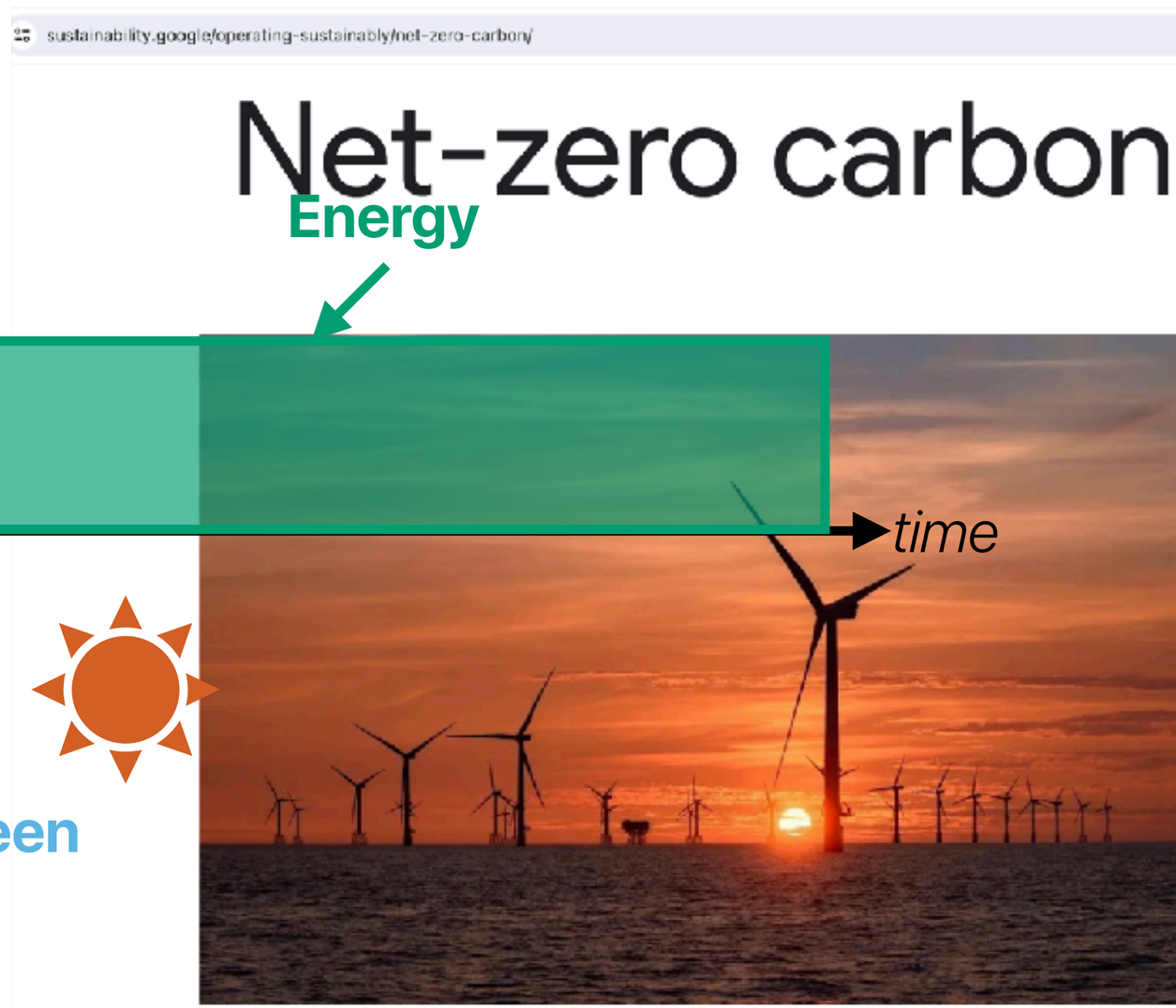
Power v.s. Energy

- Power is the direct contributor of “heat”
  - Packaging of the chip
  - Heat dissipation cost
  - Dynamic power
  - Leakage power
- Energy = $Power \times Execution\_Time$
  - The electricity bill and battery life is related to energy!
  - Lower power does not necessary means better battery life if the processor slow down the application too much

Power/Energy/Carbon footprint

The Green can be more if power is not low enough





If we run the task when there is no green energy— more carbon footprint!

Recap: Demo — changing the max frequency and performance

- Change the maximum frequency of the intel processor — you learned how to do this when we discuss programmer's impact on performance
- LIKWID a profiling tool providing power/energy information
  - likwid-perfctr -g ENERGY [command_line]
  - Let's try blockmm and popcount and see what's happening!

Lower frequency

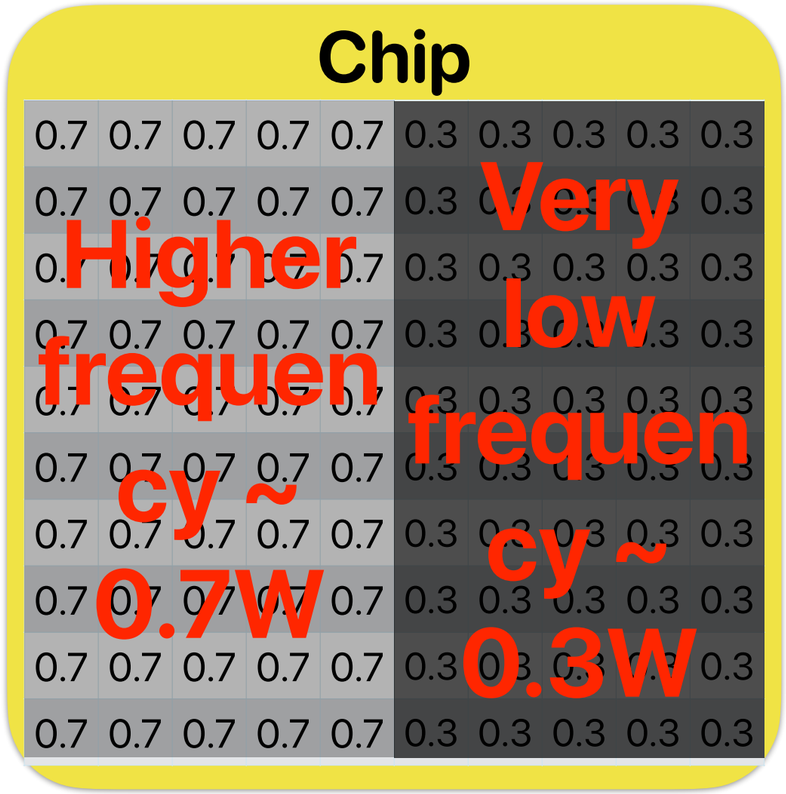
- Regarding lower the operating frequency when running applications, how many of the following statements are correct?
  - ① Lowering the frequency helps the same battery capacity to carry out more tasks
  - ② Lowering the frequency helps reducing the heat generation
  - ③ Lowering the frequency helps reducing the electricity bill/total energy consumption of running the same amount of tasks
  - ④ A CPU operating at 30% of the peak frequency can still consume more than 50% of the peak power
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

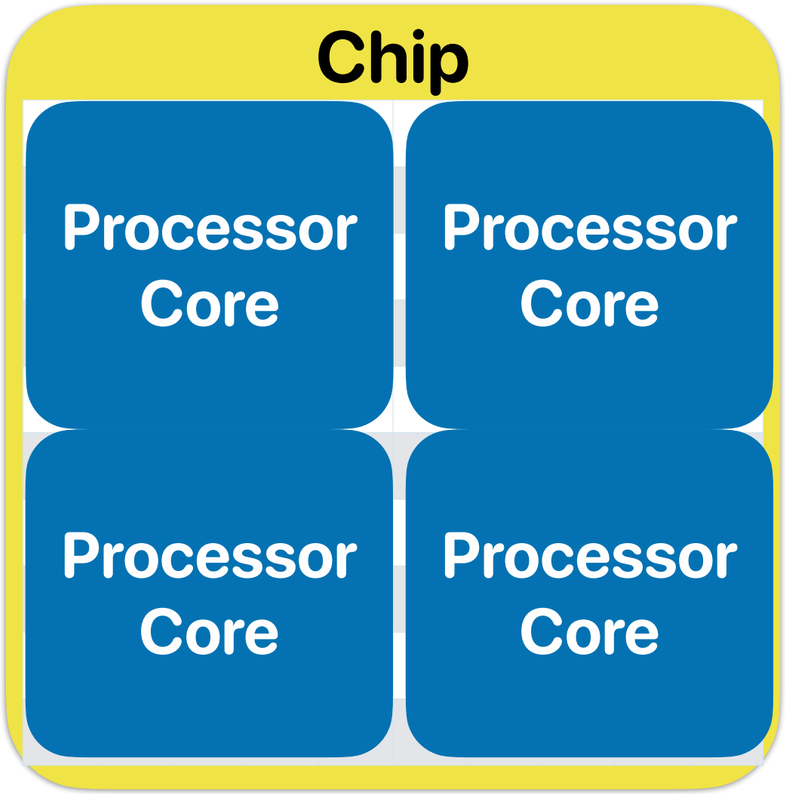
Take-aways: the new golden age of computer architectures

- Challenges and SOTA solutions in the dark silicon era
  - GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads

Given the same power budget, maximize the efficiency per chip

Some faster, some slower





Aggressively adjust the frequency of processor cores — If only one program is compute-intensive, having it running at full speed, others at lower frequency or turning off

Modern processor's frequency

```
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i7-12700KF
Stepping:                        2
CPU MHz:                         1226.409
CPU max MHz:                     5000.0000
CPU min MHz:                     800.0000
BogoMIPS:                        7219.20
```

	i7-12700K
	Intel 7
	$450.00 - $460.00
	PC/Client/Tablet, Workstation

CPU Specifications

Total Cores	12
# of Performance-cores	8
# of Efficient-cores	4
Total Threads	20
Max Turbo Frequency	5.00 GHz
Intel® Turbo Boost Max Technology 3.0 Frequency ‡	5.00 GHz
Performance-core Max Turbo Frequency	4.90 GHz
Efficient-core Max Turbo Frequency	3.80 GHz
Performance-core Base Frequency	3.60 GHz
Efficient-core Base Frequency	2.70 GHz
Cache	25 MB Intel® Smart Cache

https://www.intel.com/content/www/us/en/products/sku/134594/intel-core-i712700k-processor-25m-cache-up-to-5-00-ghz/specifications.html

Modern processor's frequency

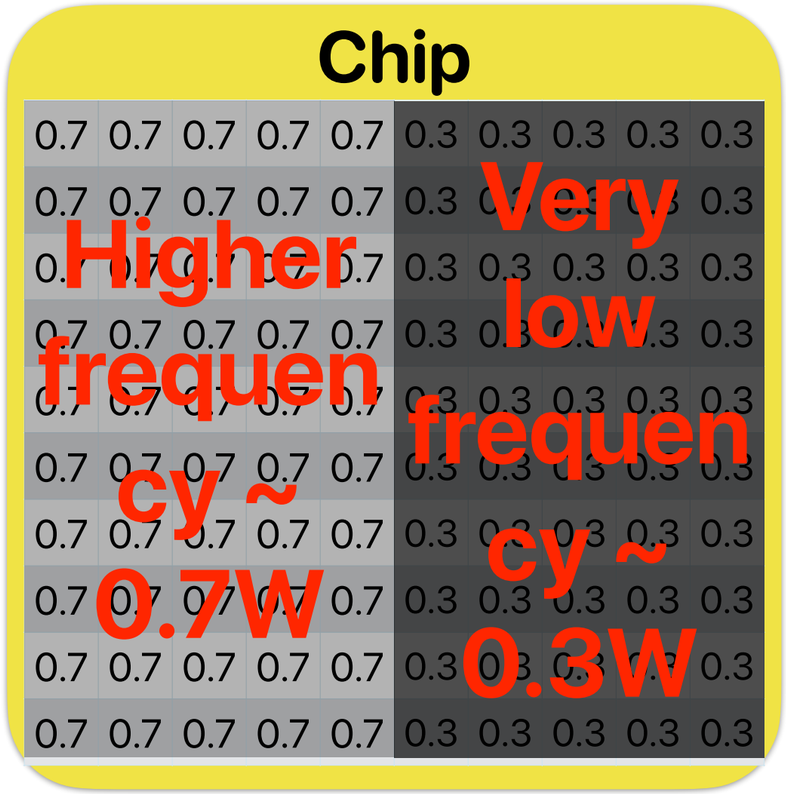
```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   48 bits physical, 48 bits virtual
CPU(s):                          12
On-line CPU(s) list:             0-11
Thread(s) per core:              2
Core(s) per socket:              6
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      25
Model:                           80
Model name:                      AMD Ryzen 5 5500
Stepping:                        0
Frequency boost:                 enabled
CPU MHz:                         3600.000
CPU max MHz:                     3600.0000
CPU min MHz:                     1400.0000
```

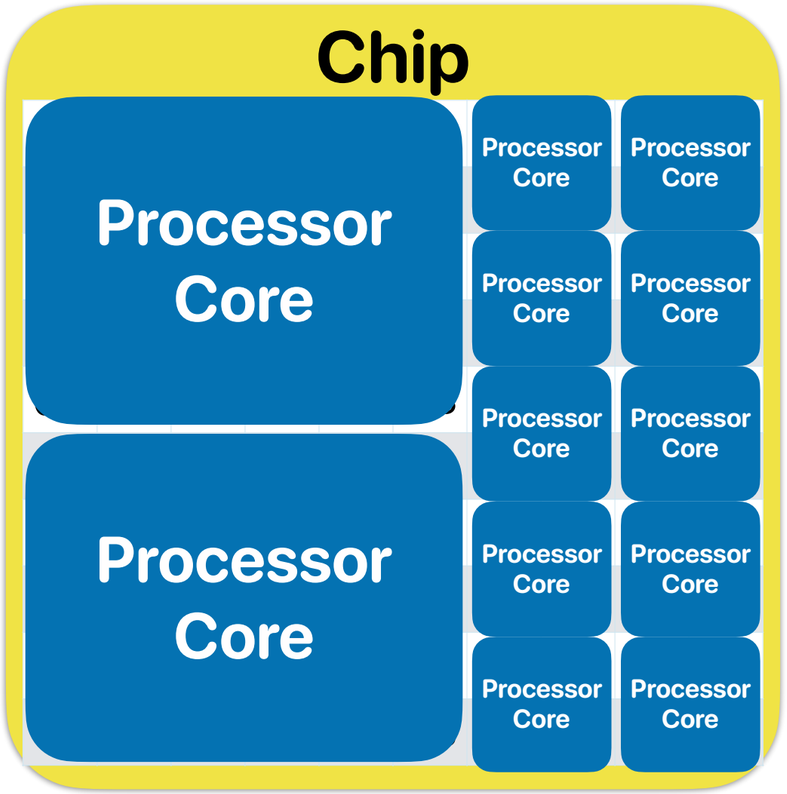
Take-aways: the new golden age of computer architectures

- Challenges and SOTA solutions in the dark silicon era
  - GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads
  - Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great

Given the same power budget, maximize the efficiency per chip

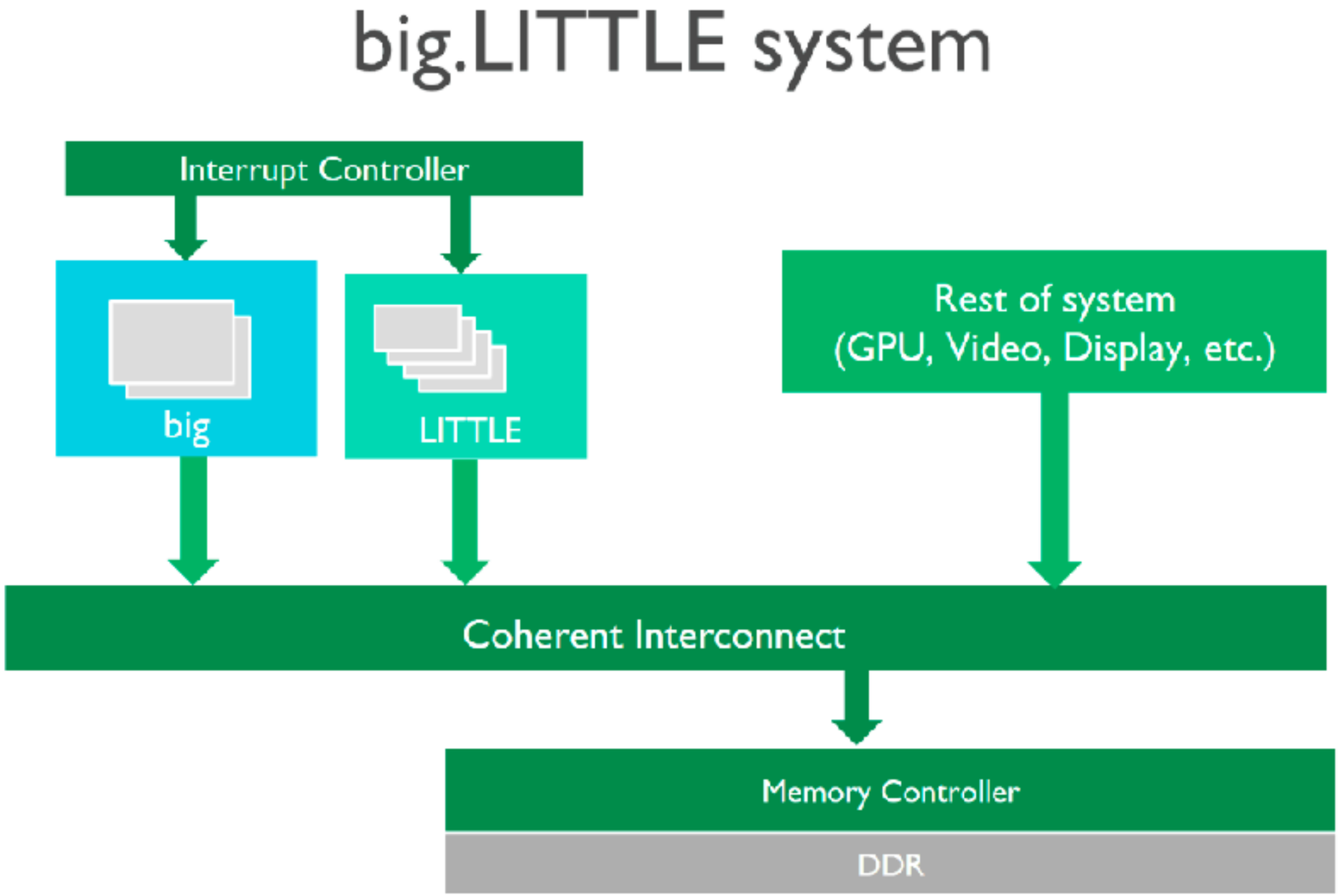
Some faster, some slower



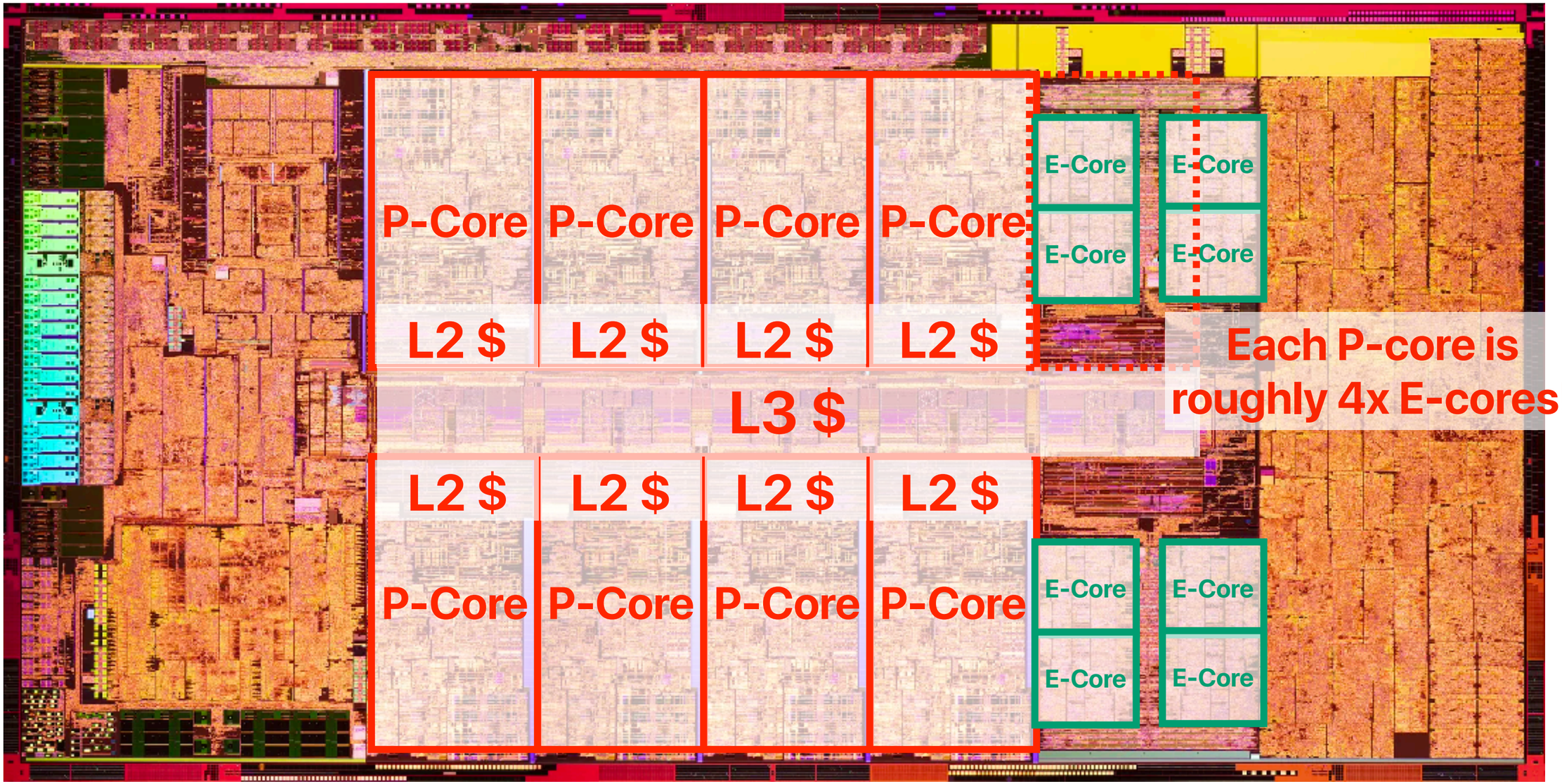


Some powerful cores for latency sensitive workloads, many small cores for highly parallelizable workloads

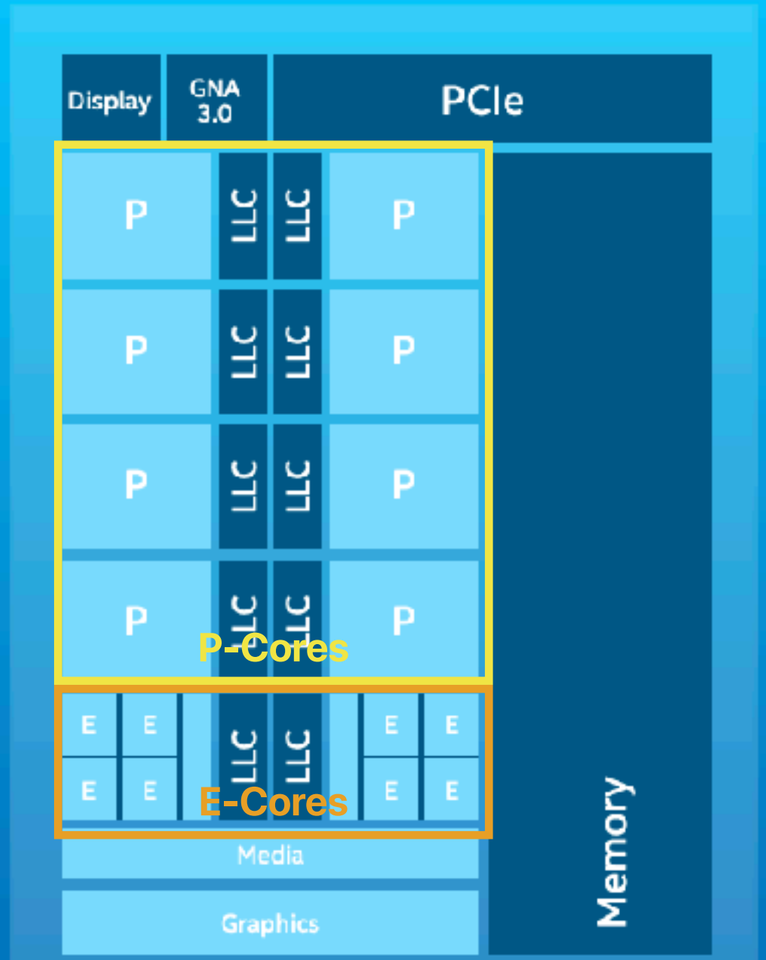
Single ISA heterogeneous CMP in ARM's big.LITTLE architecture



Intel Alder Lake



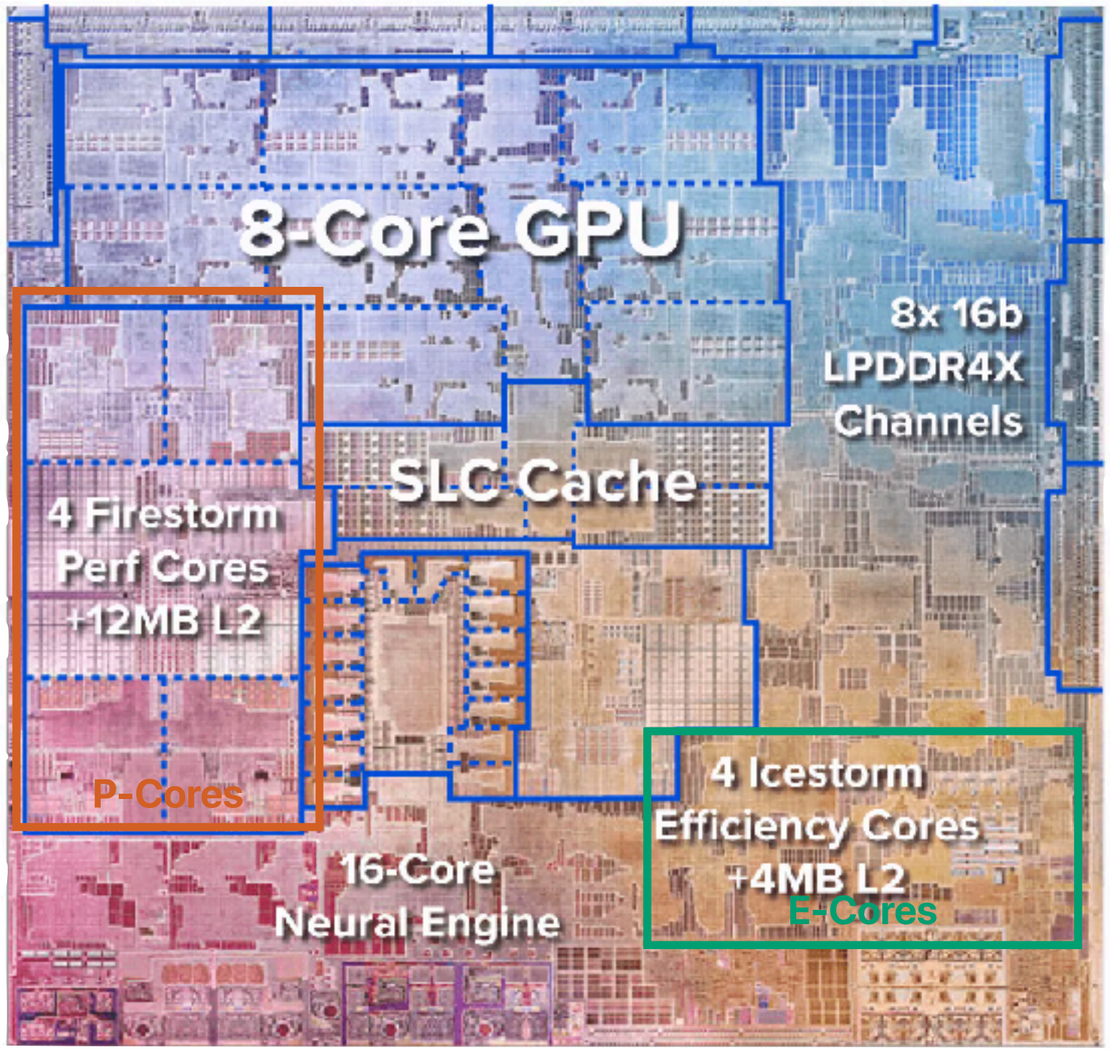
Single ISA heterogeneous CMP in Intel Processors



In a game-changing advance for core performance, 12th Gen Intel® Core™ desktop processors power a revolutionary approach to the x86 architecture. Its Performance-cores—or "P-cores"—are optimized for single & lightly-threaded performance, while its Efficient-cores—or "E-cores"—are optimized for scaling highly-threaded workloads. Intel® Thread Director helps to monitor and analyze performance data in real-time to seamlessly place the right application thread on the right core and optimize performance per watt.[1] That means gamers, creators, and professionals can harness both intelligence and power to enhance the experiences that matter most.

https://download.intel.com/newsroom/2021/client-computing/12th-gen-processor-product-brief.pdf

Single ISA heterogeneous CMP in Apple's M1



https://www.toptal.com/apple/apple-m1-processor-compatibility-overview

Single ISA heterogeneous CMP (big.Little)

- Assume we have two processor core architectures:
  (1) P-core: higher performance, but also higher power consumption and larger in size and
  (2) E-core: lower performance, but also lower power consumption and smaller in size.
  If we can build three types of multi-processors, all has the same area/size and power budget:
  (**big Only**) 4x P-cores only.
  (**Little Only**) 24x E-cores only and
  (**big.Little**) 2x P-cores and 12 E-cores. Please identify the correct expectations of their performance
  - ① For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Little Only**
  - ② For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Big Only**
  - ③ For a program with rich thread-level parallelism, **big.Little** would deliver better or at least the same level of performance than **Little Only**
  - ④ For a program with rich thread-level parallelism, s **big.Little** would deliver better or at least the same level of performance than **big Only**

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Single ISA heterogeneous CMP (big.Little)

- Assume we have two processor core architectures:
  (1) P-core: higher performance, but also higher power consumption and larger in size and
  (2) E-core: lower performance, but also lower power consumption and smaller in size.
  If we can build three types of multi-processors, all has the same area/size and power budget:
  (**big Only**) 4x P-cores only.
  (**Little Only**) 24x E-cores only and
  (**big.Little**) 2x P-cores and 12 E-cores. Please identify the correct expectations of their performance
  - ① For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Little Only**
  - ② For a program with limited thread-level parallelism, **big.Little** would deliver better than or at least the same level of performance as **Big Only**
  - ③ For a program with rich thread-level parallelism, **big.Little** would deliver better or at least the same level of performance than **Little Only**
  - ④ For a program with rich thread-level parallelism, s **big.Little** would deliver better or at least the same level of performance than **big Only**

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Take-aways: the new golden age of computer architectures

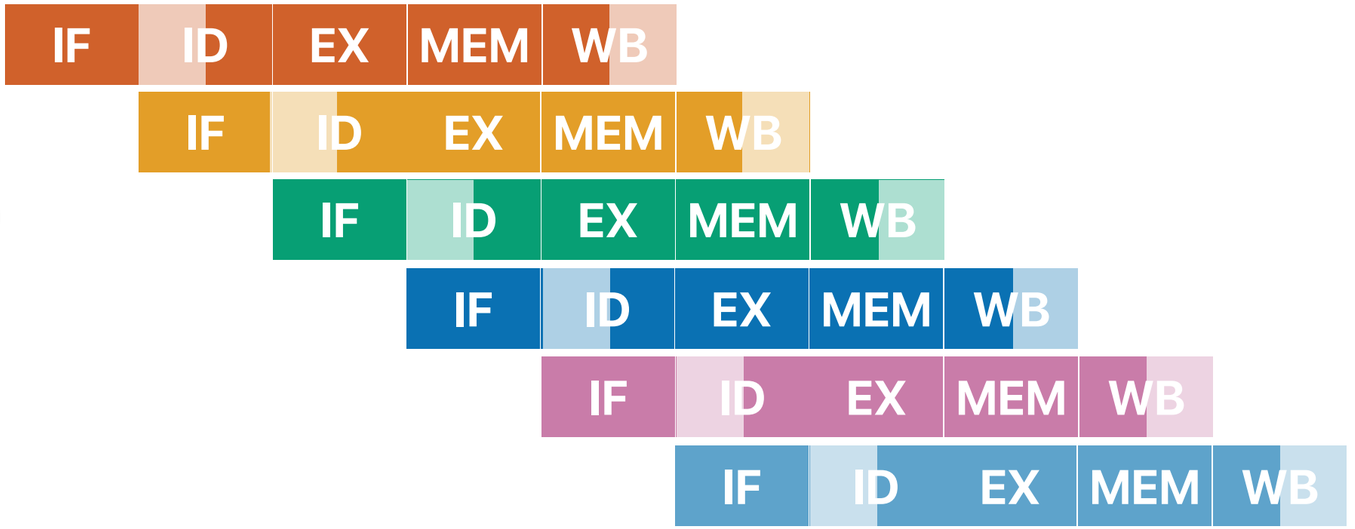
- Challenges and SOTA solutions in the dark silicon era
  - GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads
  - Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great
  - Single ISA, heterogeneous CMPs (e.g., big.Little cores, Intel/Apple's P-cores/E-cores) find a balance the trade-offs of general-purpose workloads, but won't be ideal if your applications go to either extreme of throughput (massive parallelism) or latency

The Rise of ASICs/Accelerators — A New “Golden” Age of Architects

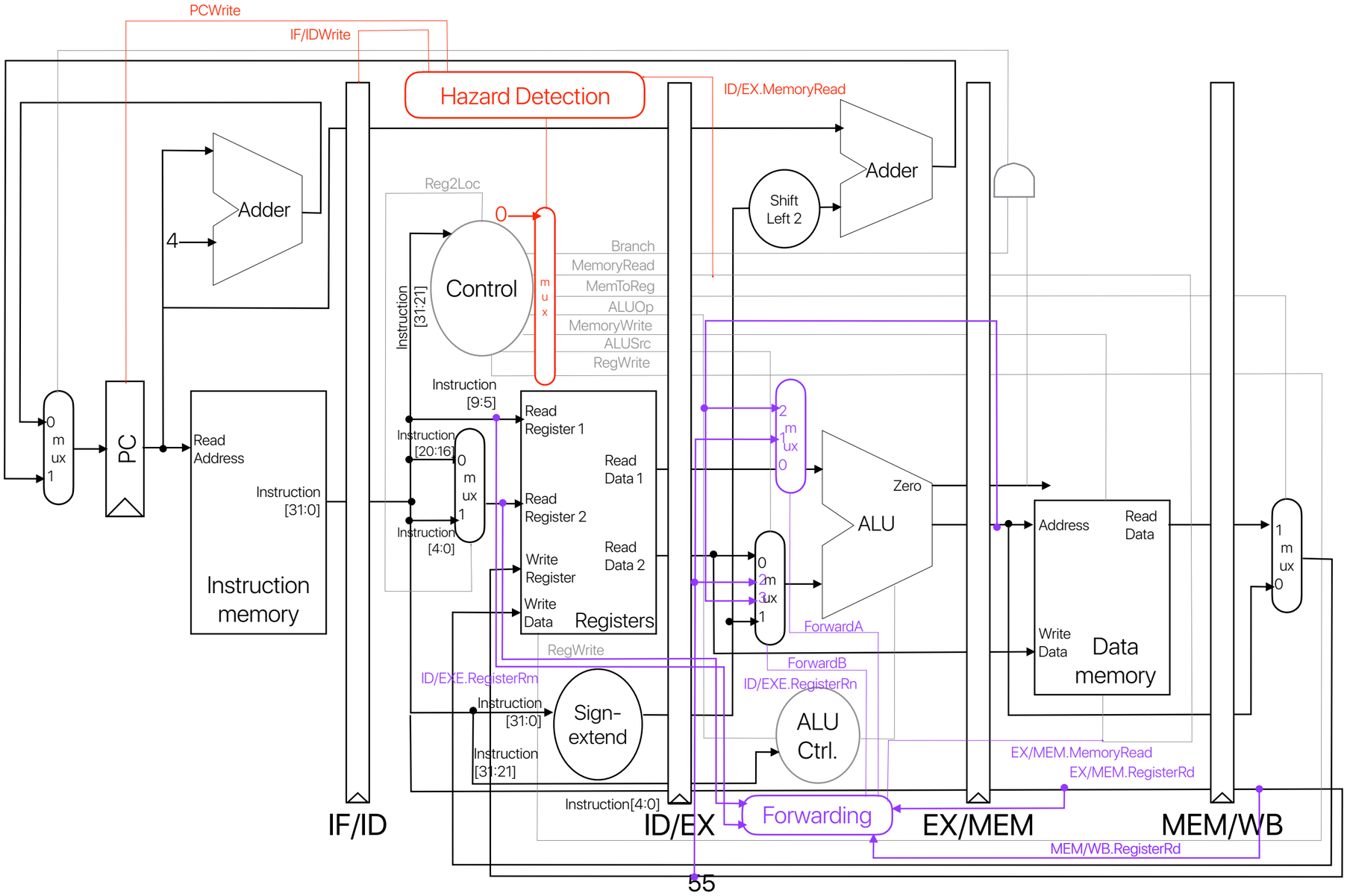
Say, we want to implement a[i] += a[i+1]*20

- This is what we need in RISC-V in each iteration

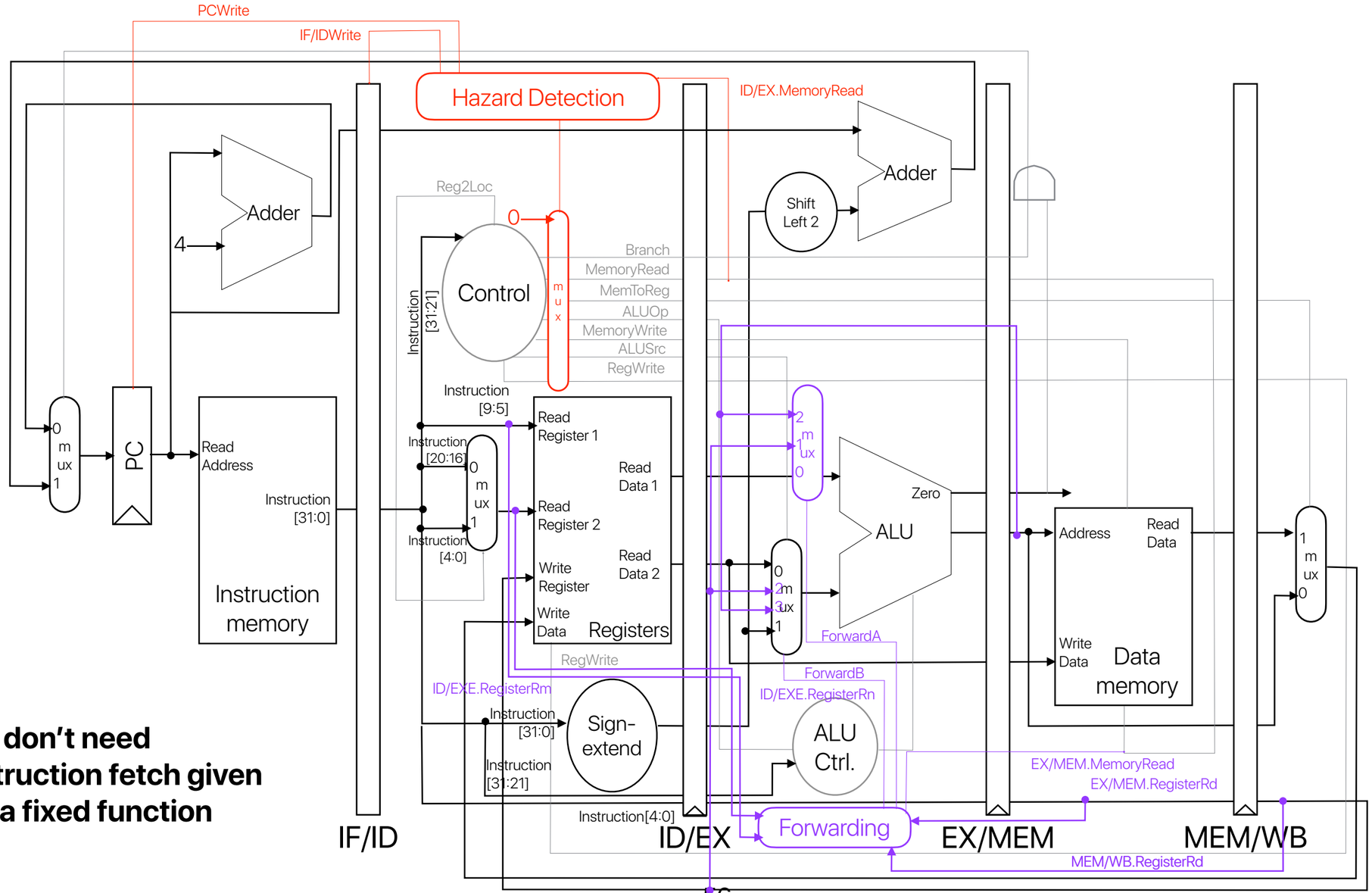
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



This is what you need for these instructions



Specialize the circuit

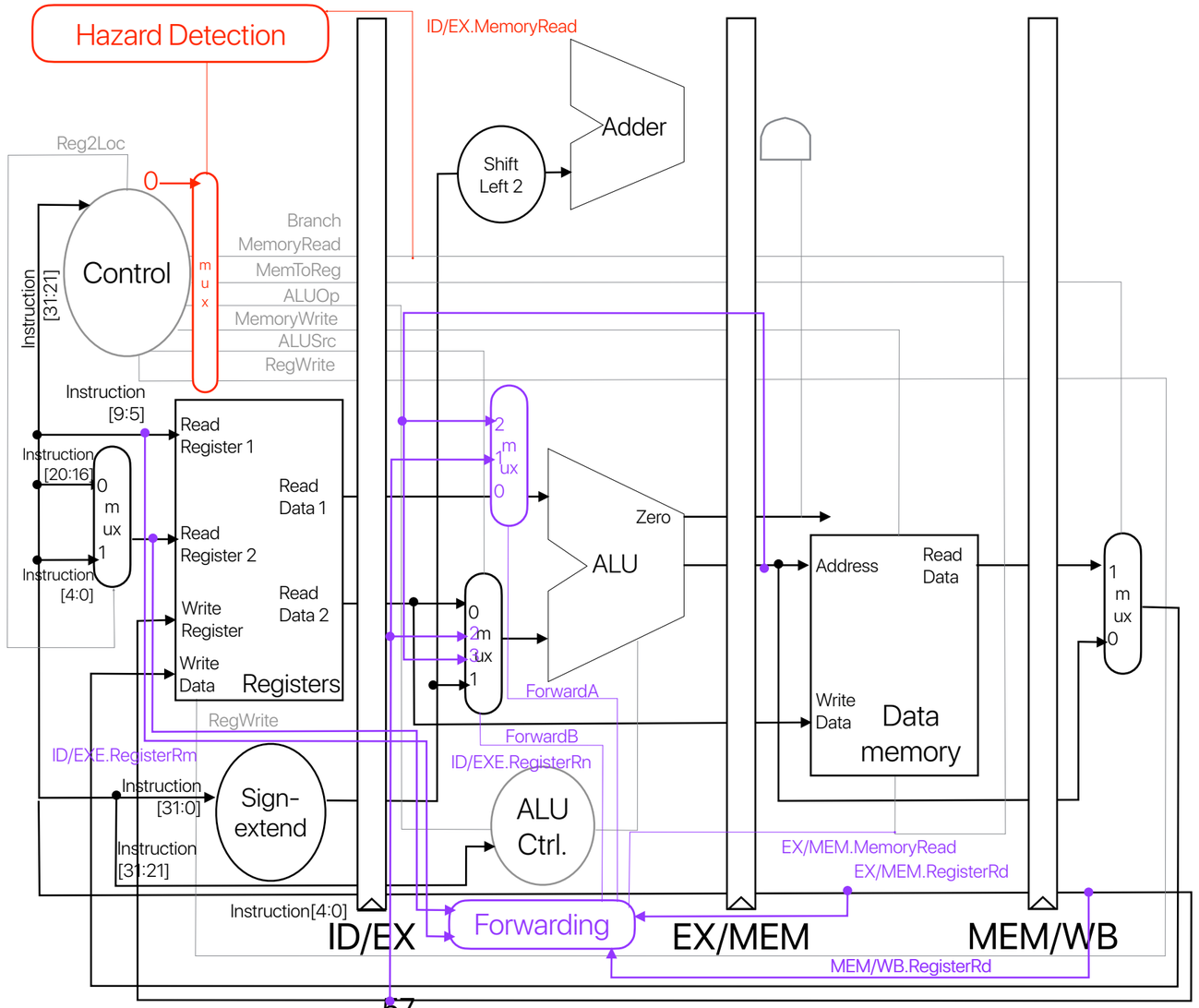


We don't need instruction fetch given it's a fixed function

Specialize the circuit

We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function

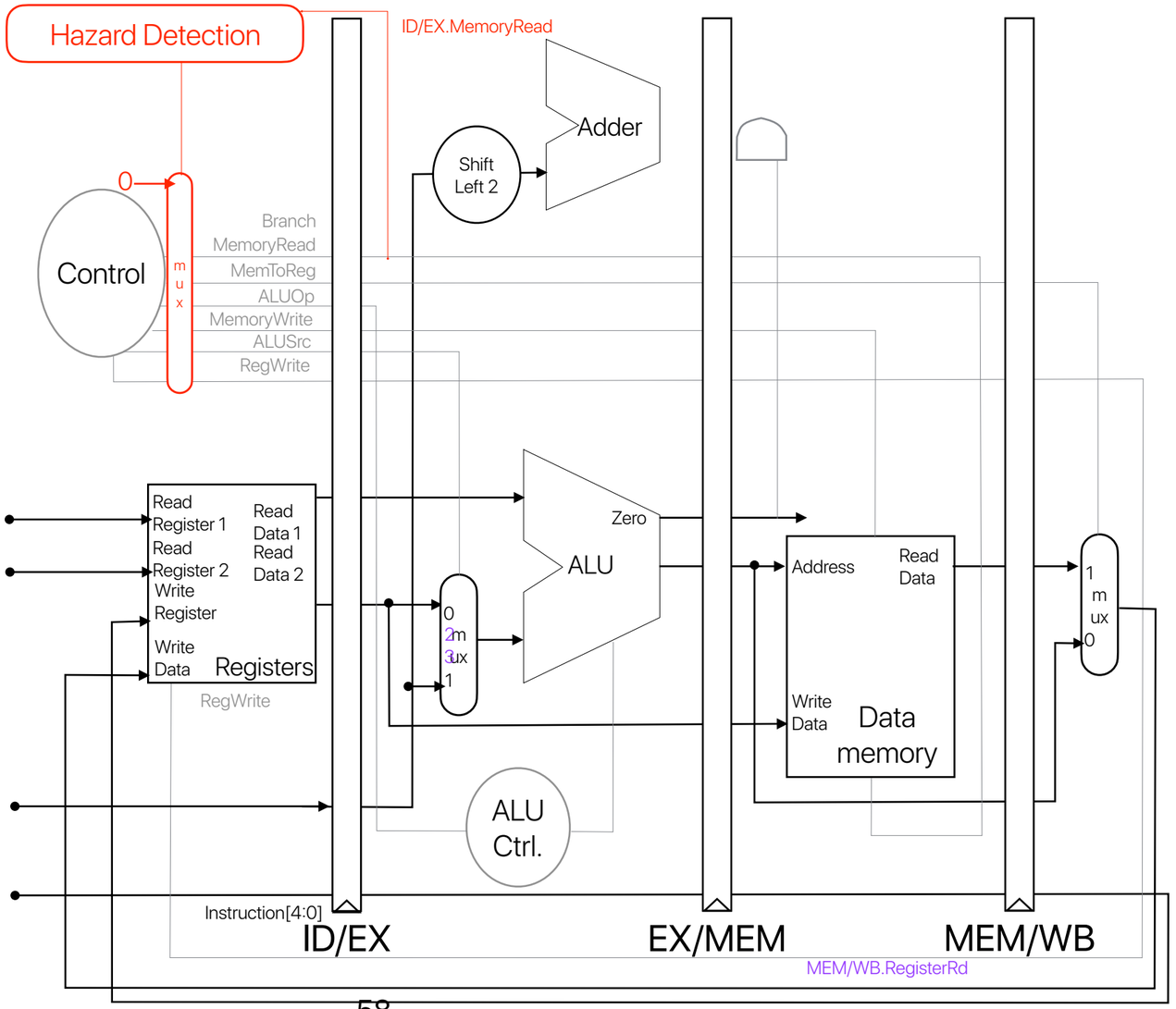


Specialize the circuit

We don't need ALUs, branches, hazard detections…

We don't need these many registers, complex control, decode

We don't need instruction fetch given it's a fixed function

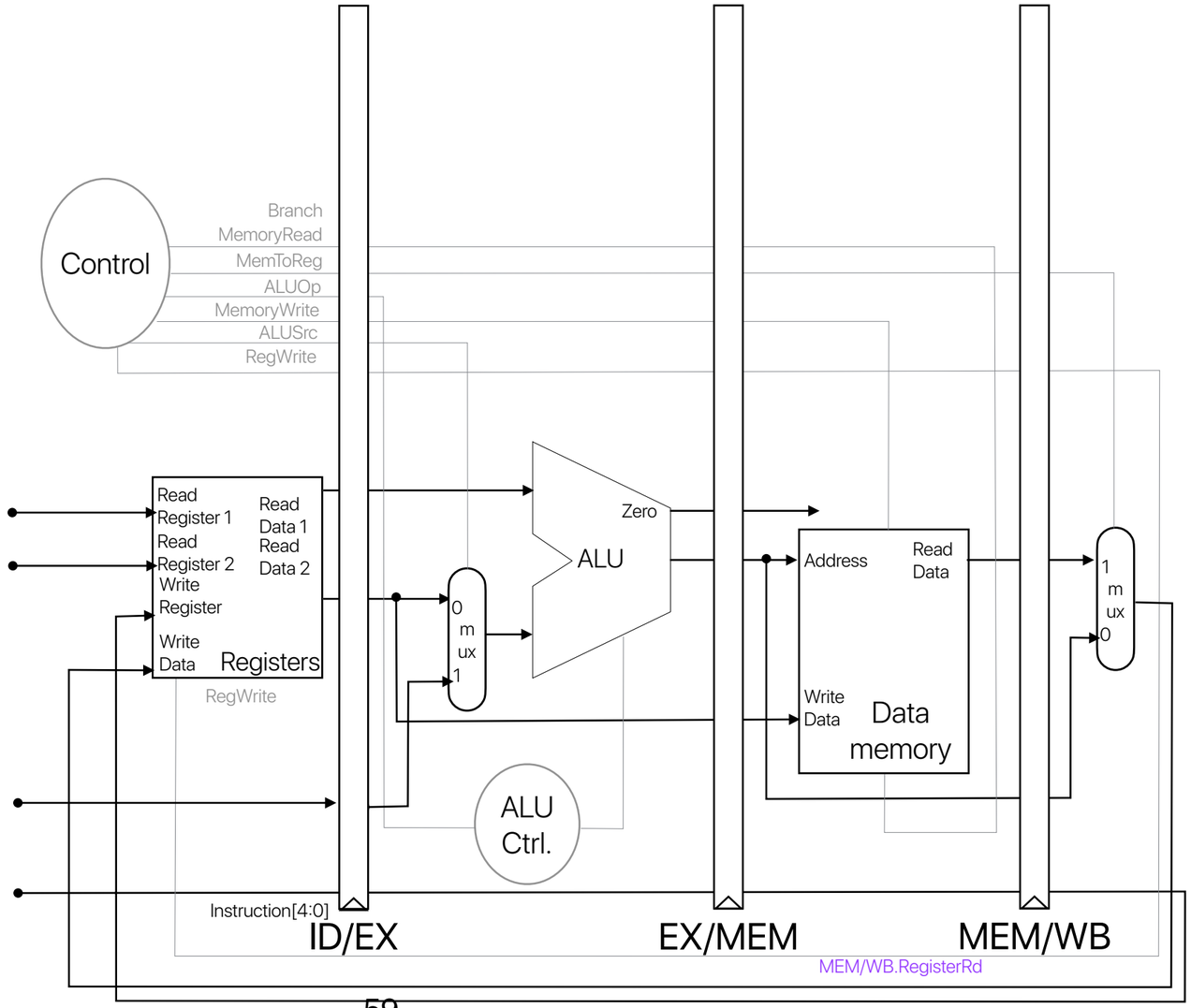


Specialize the circuit

We don’t need big ALUs, branches, hazard detections…

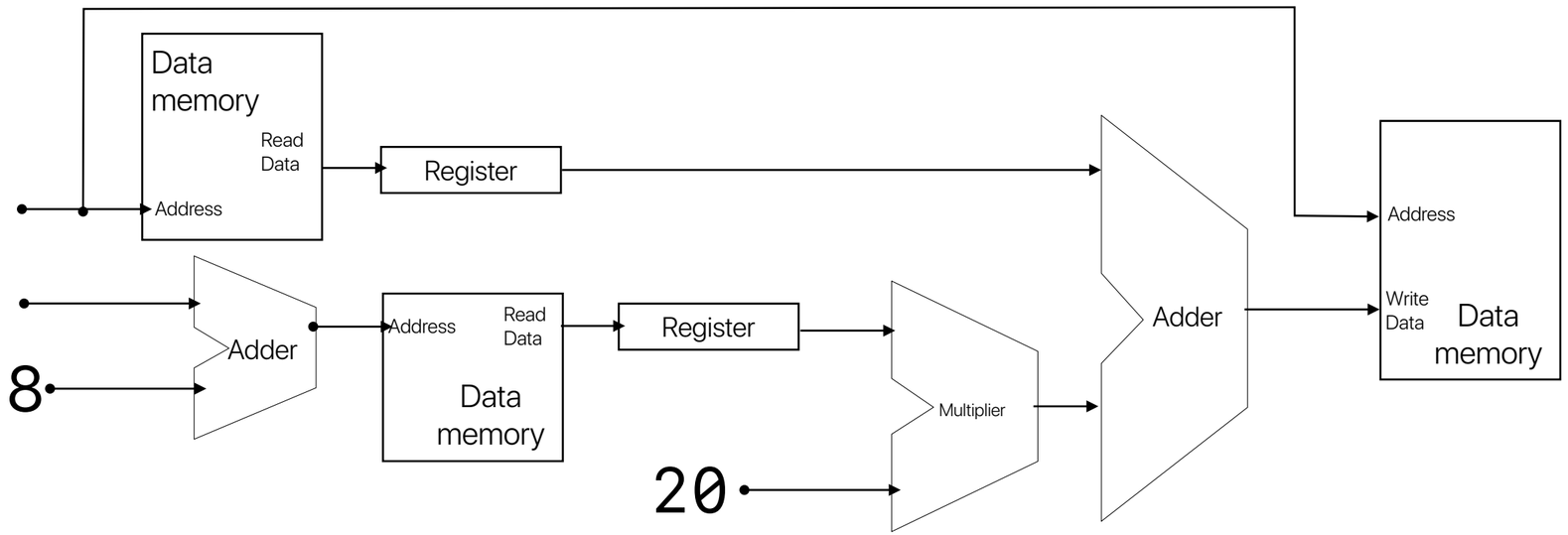
We don’t need these many registers, complex control, decode

We don’t need instruction fetch given it’s a fixed function



Rearranging the datapath

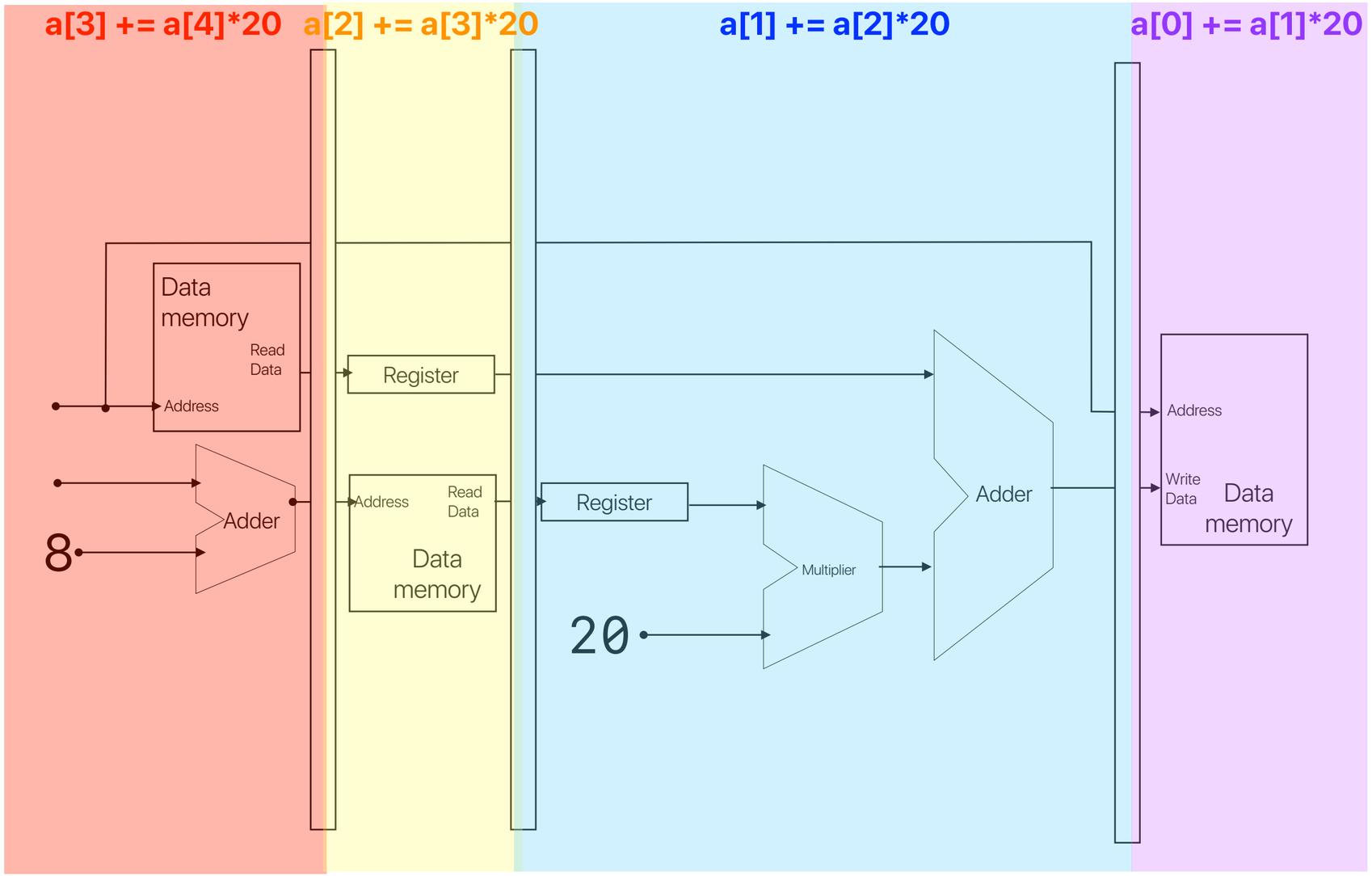
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



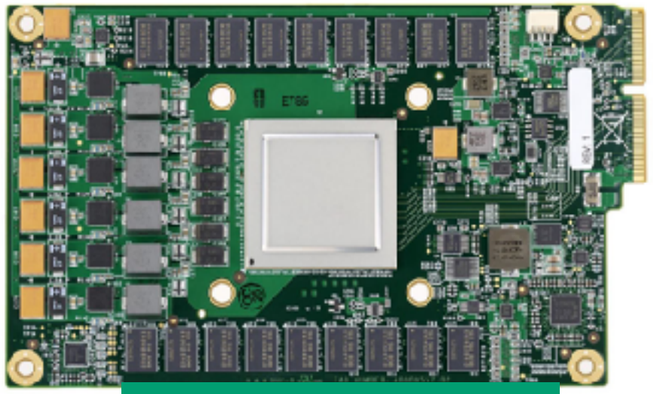
The pipeline for a[i] += a[i+1]*20

Each stage can still be as fast as the pipelined processor

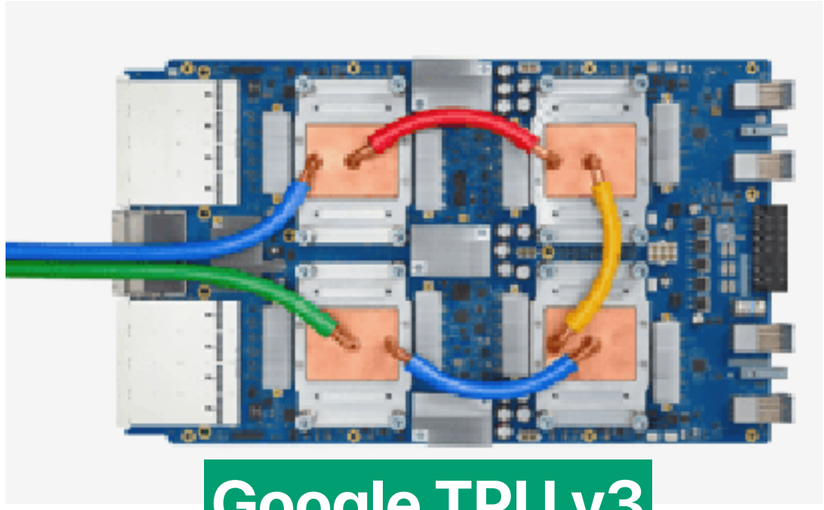
But each stage is now working on what the original 6 instructions would do



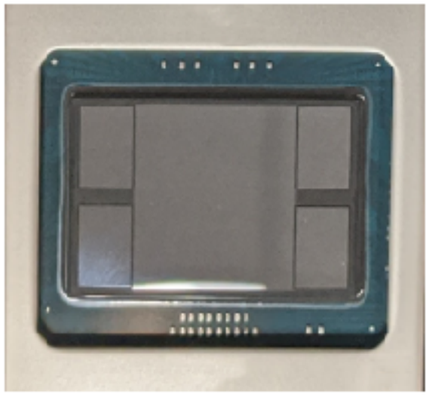
Tensor Processing Units

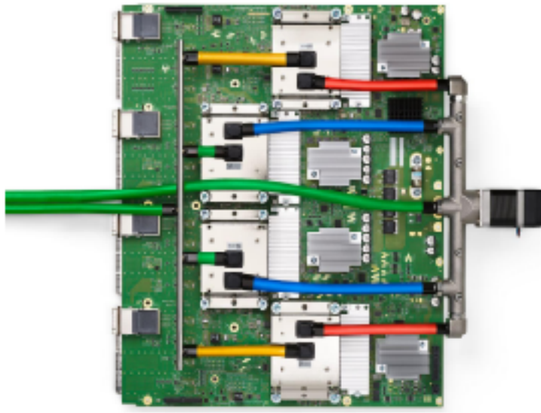


Google TPU v1

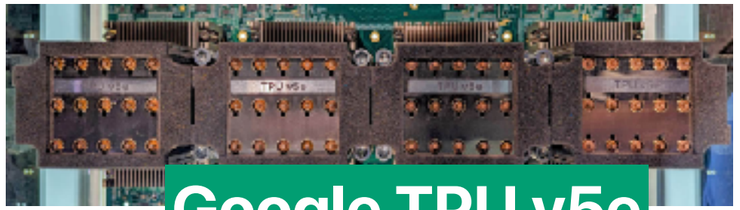


Google TPU v3





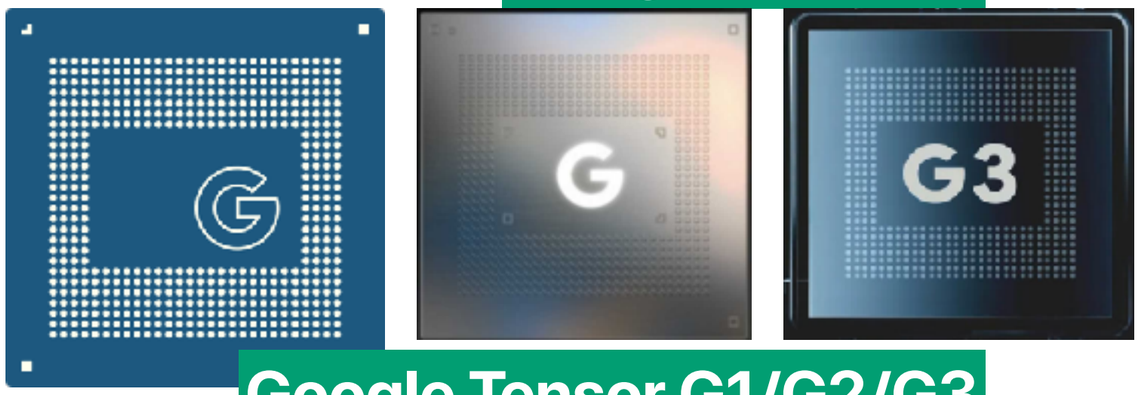
Google TPU v4



Google TPU v5e

Edge TPU

Google TPU v2



Google Tensor G1/G2/G3

https://www.pollev.com/hungweitseng close in 1:30

TPU (Tensor Processing Unit)

- Regarding TPUs, please identify how many of the following statements are correct.
  - ① TPU is optimized for highly accurate matrix multiplications
  - ② TPU is designed for dense matrices, not for sparse matrices
  - ③ TPU can reduce the "IC" in the performance equation
  - ④ TPU features an instruction set architecture that facilitates pipelining

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

TPU (Tensor Processing Unit)

- Regarding TPUs, please identify how many of the following statements are correct.
  - ① TPU is optimized for highly accurate matrix multiplications
  - ② TPU is designed for dense matrices, not for sparse matrices
  - ③ TPU can reduce the "IC" in the performance equation
  - ④ TPU features an instruction set architecture that facilitates pipelining

A. 0
B. 1
C. 2
D. 3
E. 4

- *Pitfall: Being ignorant of architecture history when designing a domain-specific architecture.*

Ideas that didn't fly for general-purpose computing may be ideal for domain-specific architectures. For the TPU, three important architectural features date back to the early 1980s: systolic arrays [31], decoupled-access/execute [54], and CISC instructions [41]. 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.

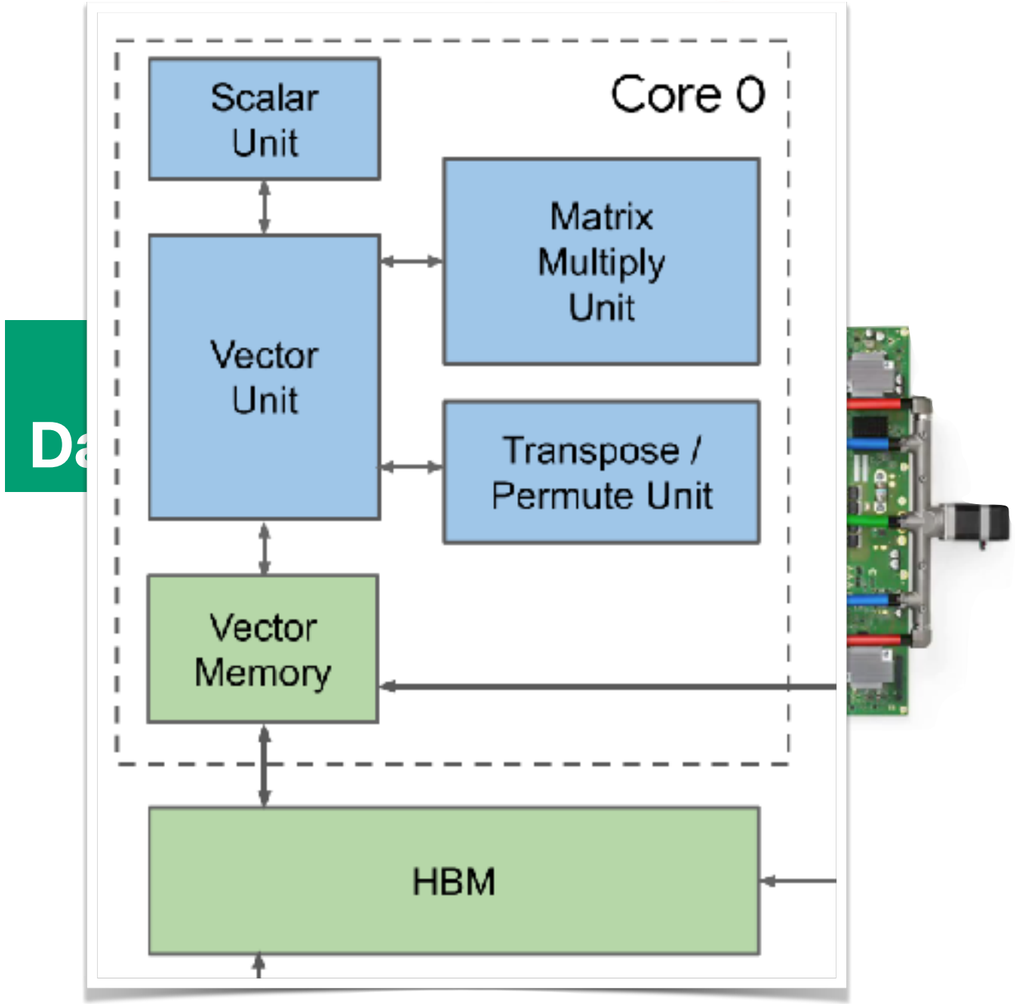
peak performance.

When using a mix of 8-bit weights and 16-bit activations (or vice versa), the Matrix Unit computes at half-speed, and it computes at a quarter-speed when both are 16 bits. It reads and

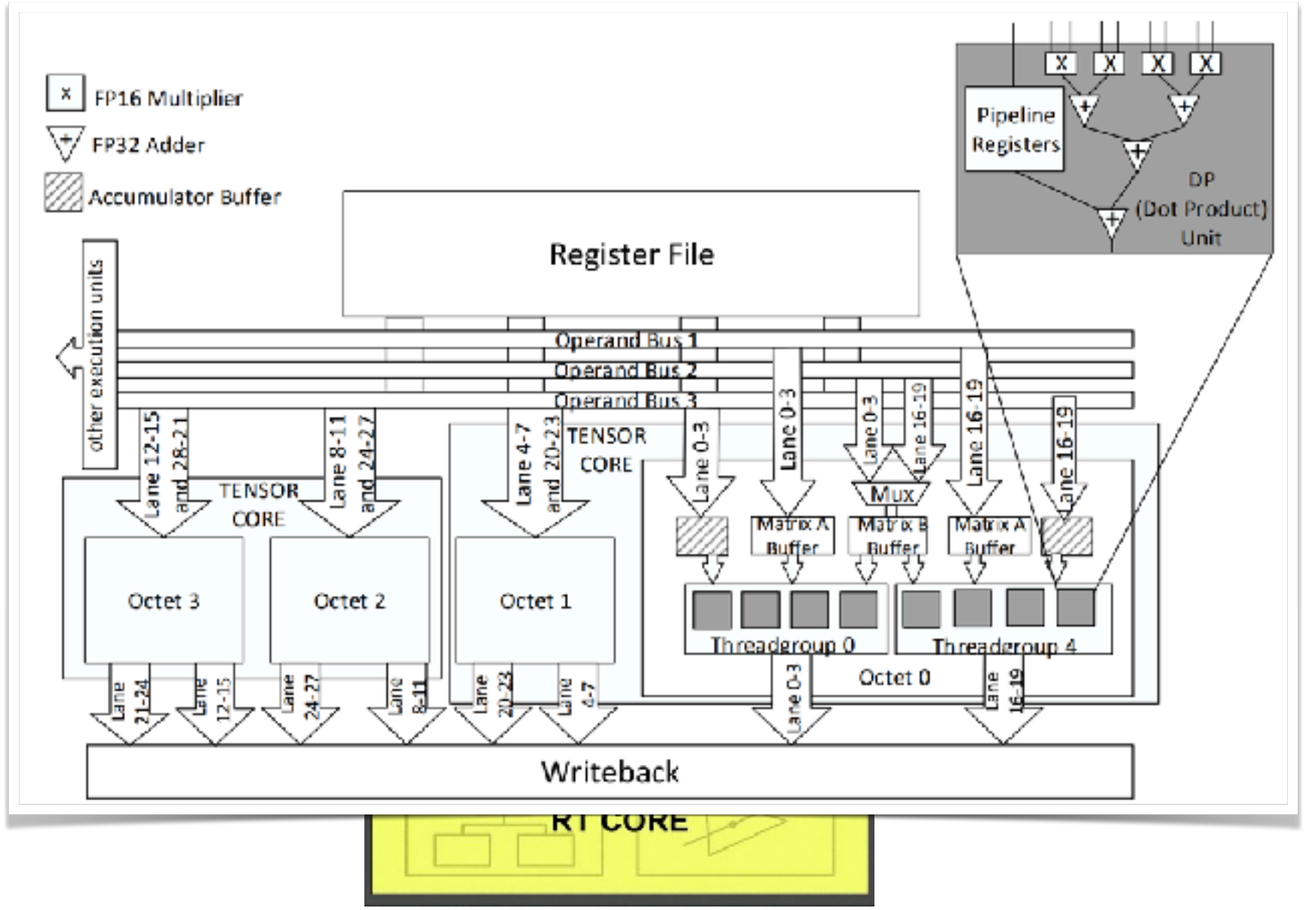
takes to shift a tile in). This unit is designed for dense matrices. Sparse architectural support was omitted for time-to-deployment reasons. The weights for the matrix unit are staged through an on-

As instructions are sent over the relatively slow PCIe bus, TPU instructions follow the CISC tradition, including a repeat field. The average clock cycles per instruction (CPI) of these CISC instructions is typically 10 to 20. It has about a dozen

Matrix processing — the core of AI/ML accelerators



T. Norrie et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," in IEEE Micro, vol. 41, no. 2, pp. 56-63



M. Raihan, N. Goli and T. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs, ISPASS 2019

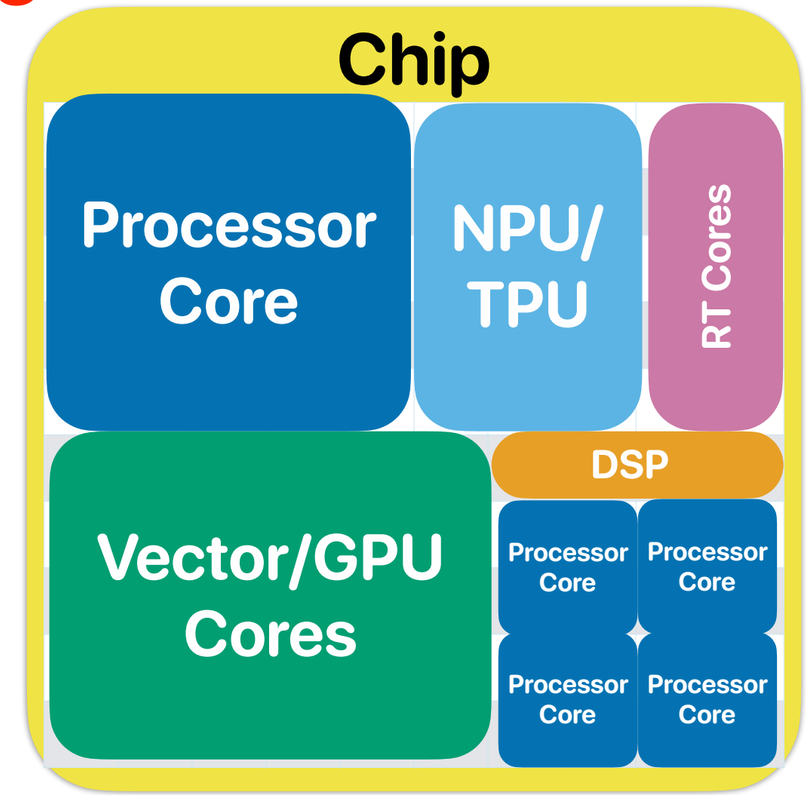
TensorFlow — the standard TPU programming interface

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='')
tf.config.experimental_connect_to_cluster(resolver)
# This is the TPU initialization code that has to be at the beginning.
tf.tpu.experimental.initialize_tpu_system(resolver)
print("All devices: ", tf.config.list_logical_devices('TPU'))

a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])

with tf.device('/TPU:0'):
  c = tf.matmul(a, b)

print("c device: ", c.device)
print(c)
```

Only very high-level mathematical/domain-specific functions are exposed to the user!

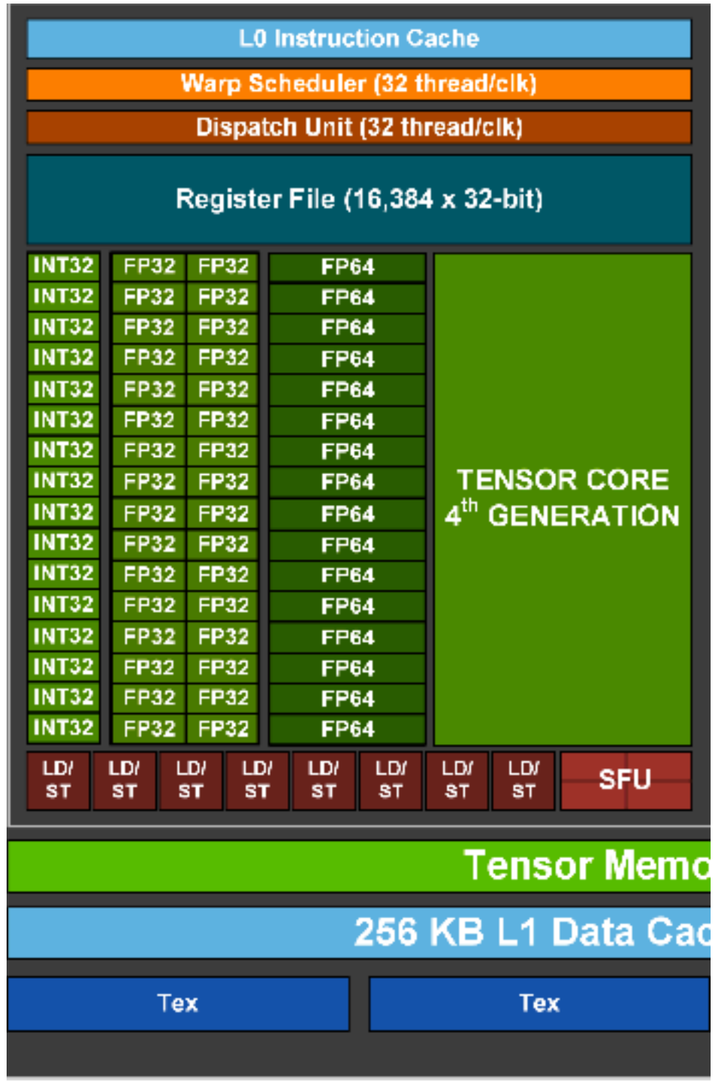
Given the same power budget, maximize the efficiency per chip

Some at top speed, some are not functioning.



Turn off unnecessary ones when we don't need specialized functions!

NVIDIA's Tensor Cores



What are Tensor Cores?

Tesla V100's Tensor Cores are programmable matrix-multiply-and-accumulate units that can deliver up to 125 Tensor TFLOPS for training and inference applications. The Tesla V100 GPU contains 640 Tensor Cores: 8 per SM. Tensor Cores and their associated data paths are custom-crafted to dramatically increase floating-point compute throughput at only modest area and power costs. Clock gating is used extensively to maximize power savings.

Each Tensor Core provides a 4x4x4 matrix processing array which performs the operation **D** = **A** * **B** + **C**, where **A**, **B**, **C** and **D** are 4×4 matrices as Figure 1 shows. The matrix multiply inputs **A** and **B** are FP16 matrices, while the accumulation matrices **C** and **D** may be FP16 or FP32 matrices.

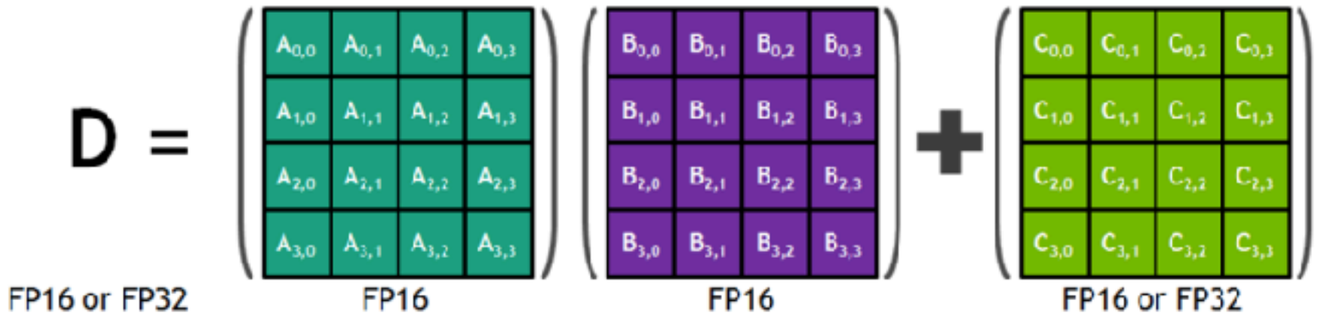
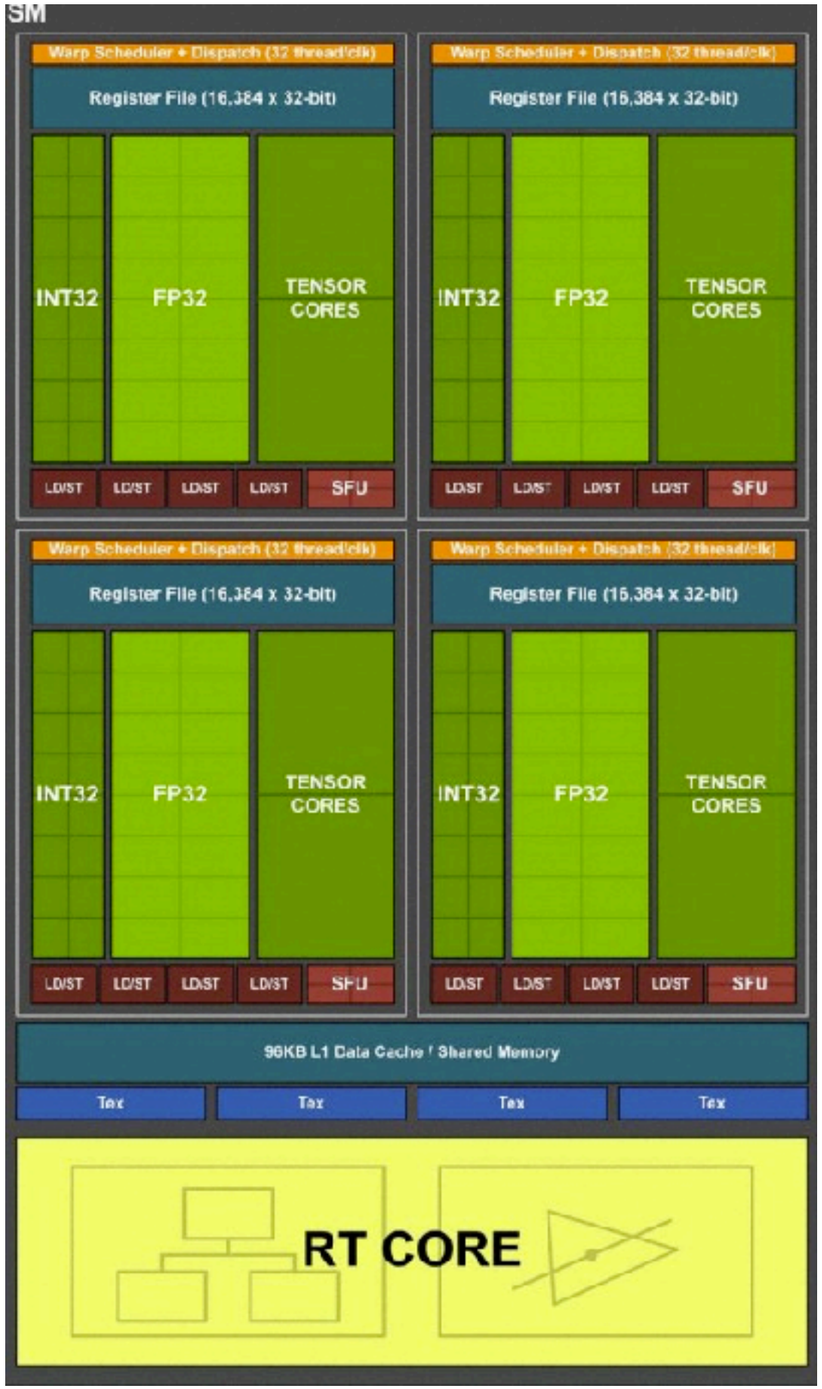


Figure 1: Tensor Core 4x4x4 matrix multiply and accumulate.

https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/



NVIDIA GPU =
CUDA Cores (vector)+
Tensor Cores (matrix) +
RT Cores (tree intersection)+
DPX Units (dynamic programming)

Programming in Turing Architecture

Use tensor cores

```
cublasErrCheck(cublasSetMathMode(cublasHandle, CUBLAS_TENSOR_OP_MATH));
```

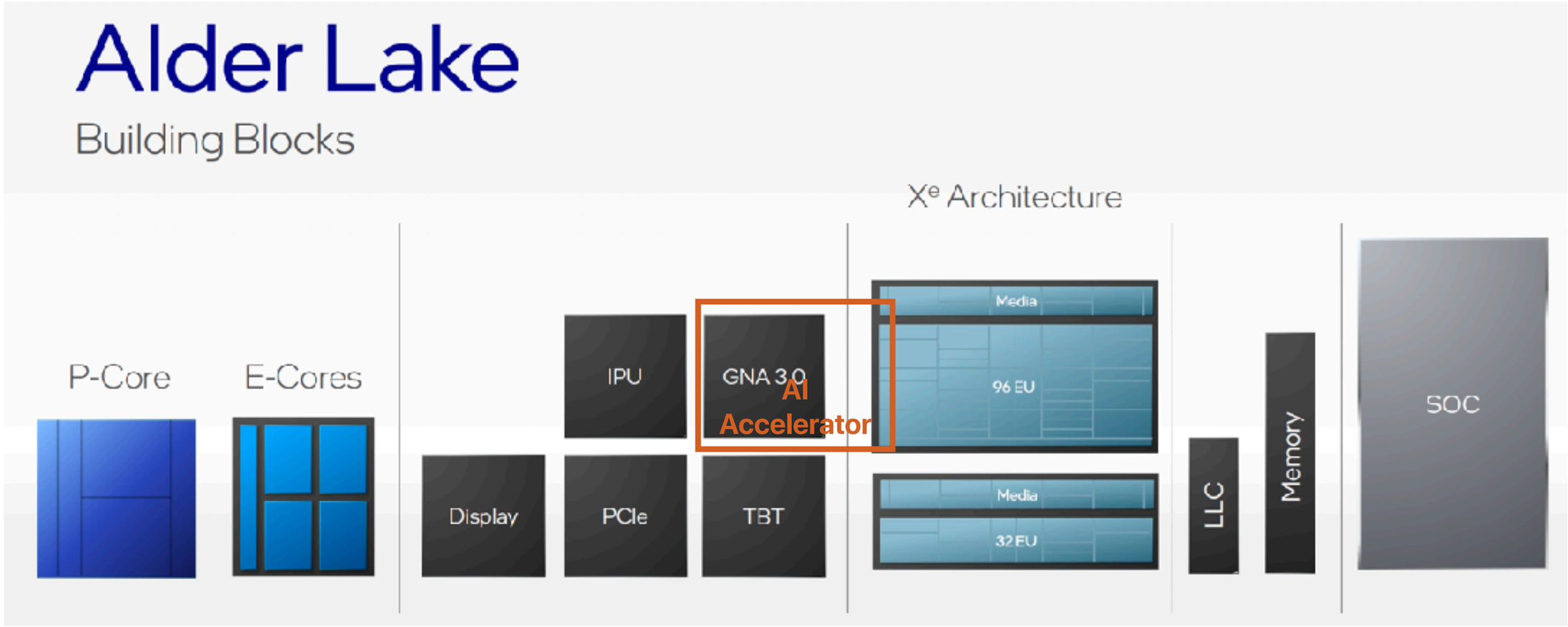
Make them 16-bit

```
convertFp32ToFp16 <<< (MATRIX_M * MATRIX_K + 255) / 256, 256 >>> (a_fp16, a_fp32,
MATRIX_M * MATRIX_K);
   convertFp32ToFp16 <<< (MATRIX_K * MATRIX_N + 255) / 256, 256 >>> (b_fp16, b_fp32,
MATRIX_K * MATRIX_N);
```

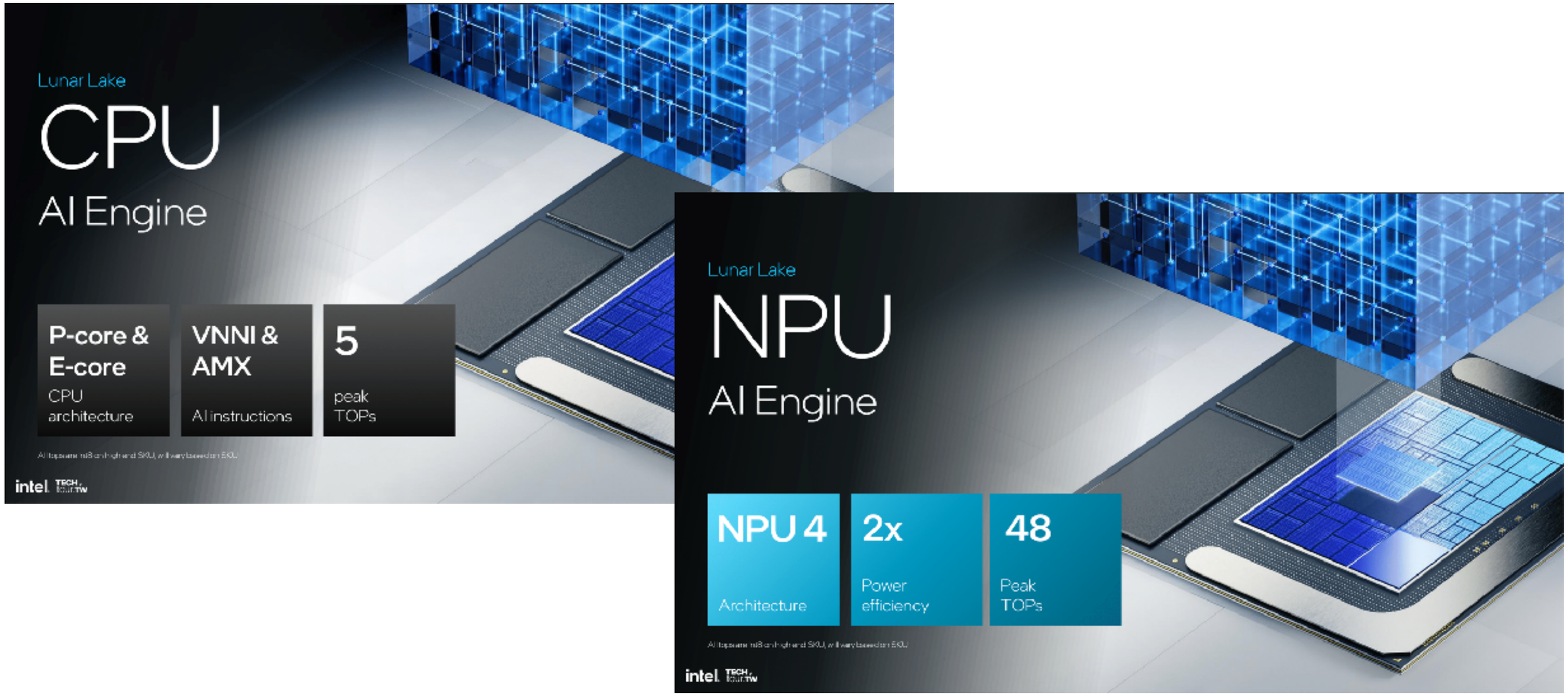
```
cublasErrCheck(cublasGemmEx(cublasHandle, CUBLAS_OP_N, CUBLAS_OP_N,
                 MATRIX_M, MATRIX_N, MATRIX_K,
                 &alpha,
                 a_fp16, CUDA_R_16F, MATRIX_M,
                 b_fp16, CUDA_R_16F, MATRIX_K,
                 &beta,
                 c_cublas, CUDA_R_32F, MATRIX_M,
                 CUDA_R_32F, CUBLAS_GEMM_DFALT_TENSOR_OP));
```

call Gemm

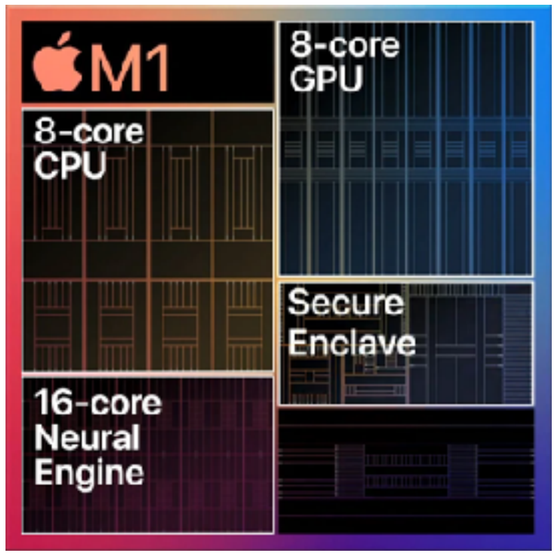
Modern processors also contain “accelerators”

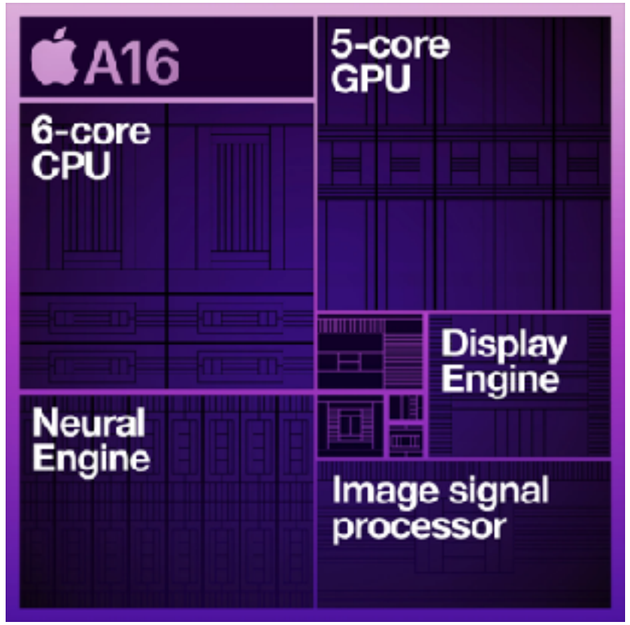


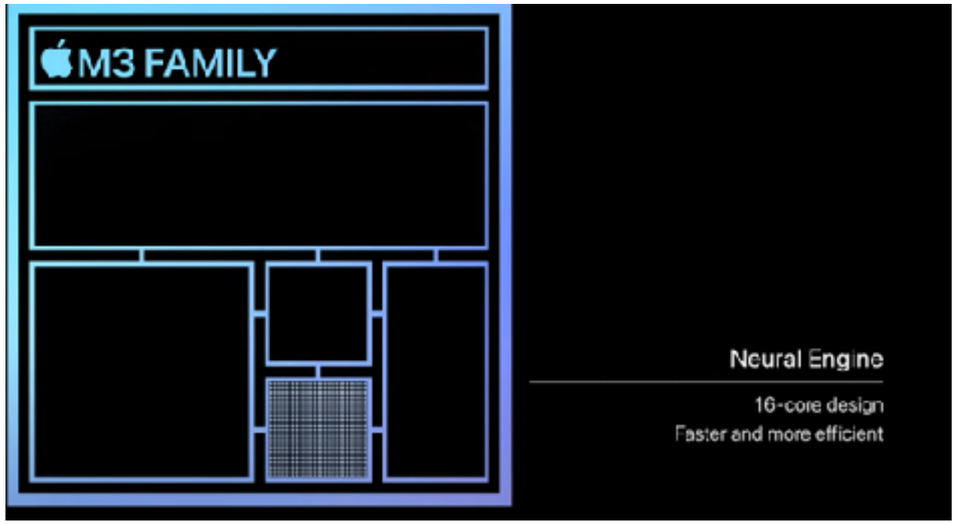
Intel’s Neural Processing Unit

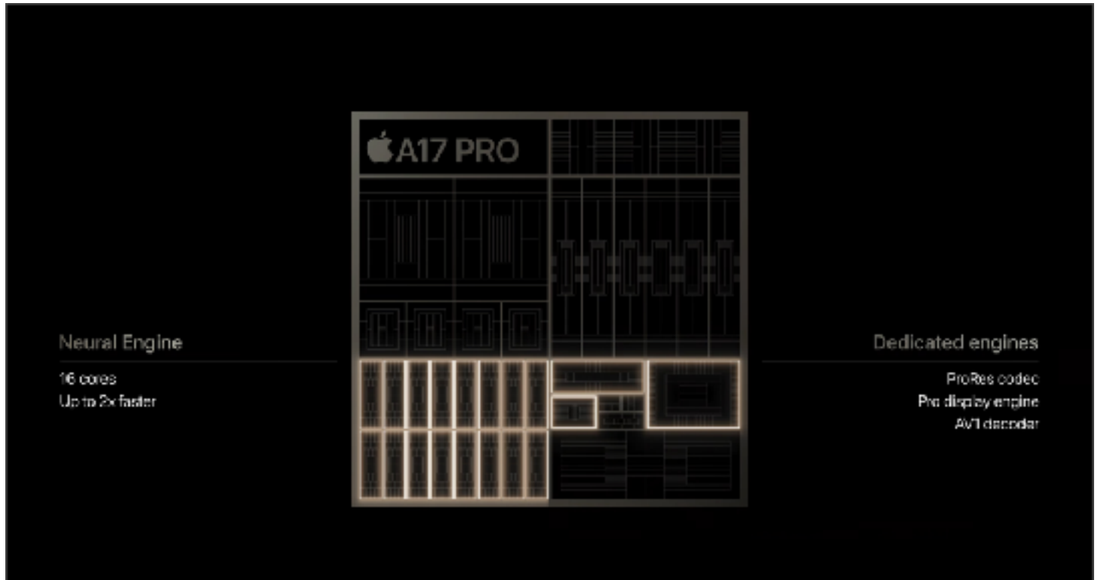


Apple’s Neural Engine



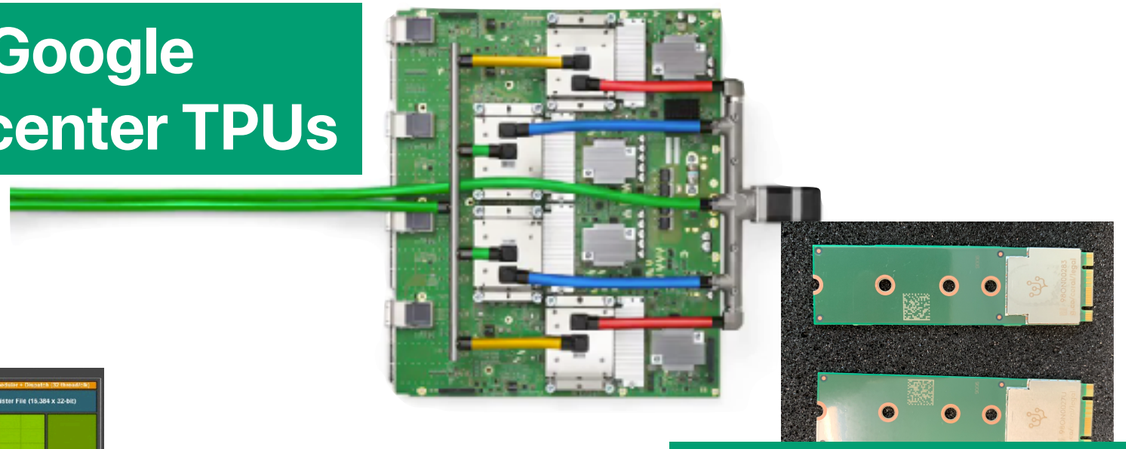






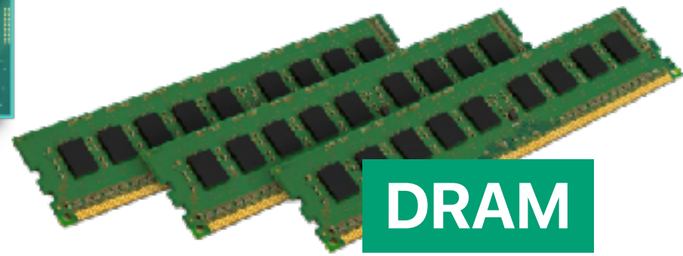
The “landscape” of modern computers

Google Datacenter TPUs



Google Edge TPUs

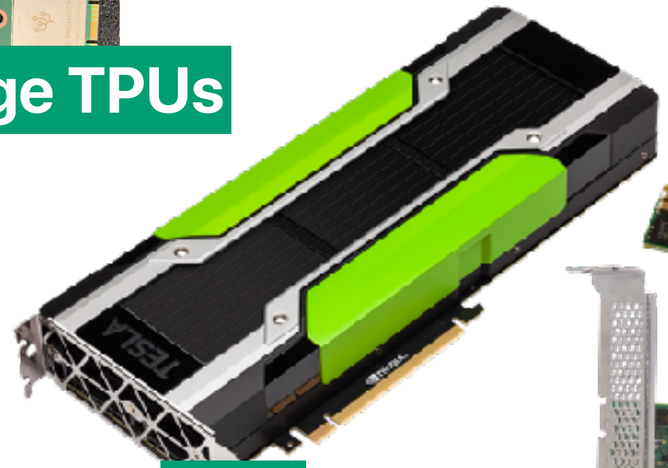
CPU



DRAM

NVIDIA Tensor Core Units

AI/ML Accelerators



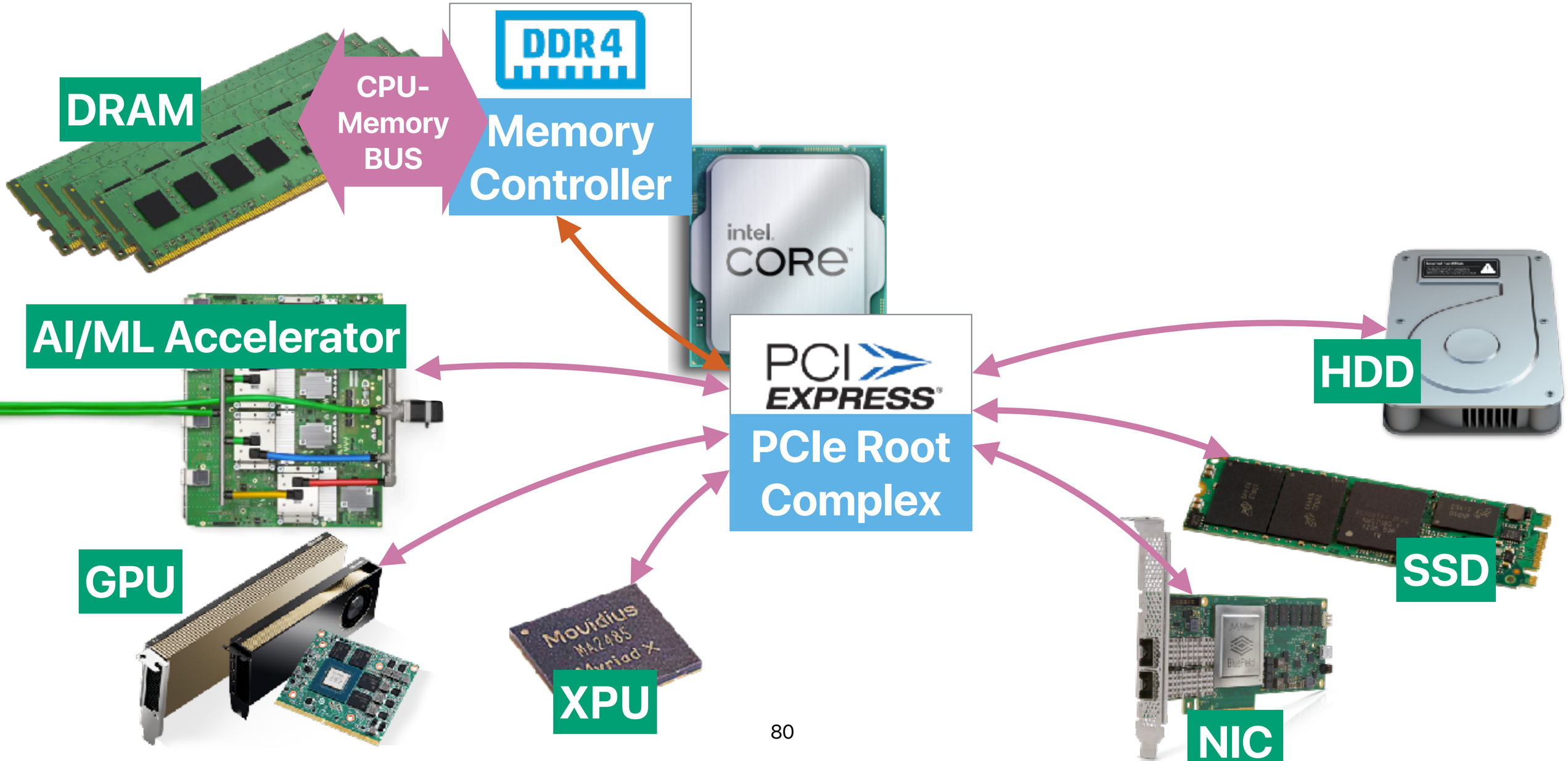
GPU

SSD

Apple Neural Engines

NIC

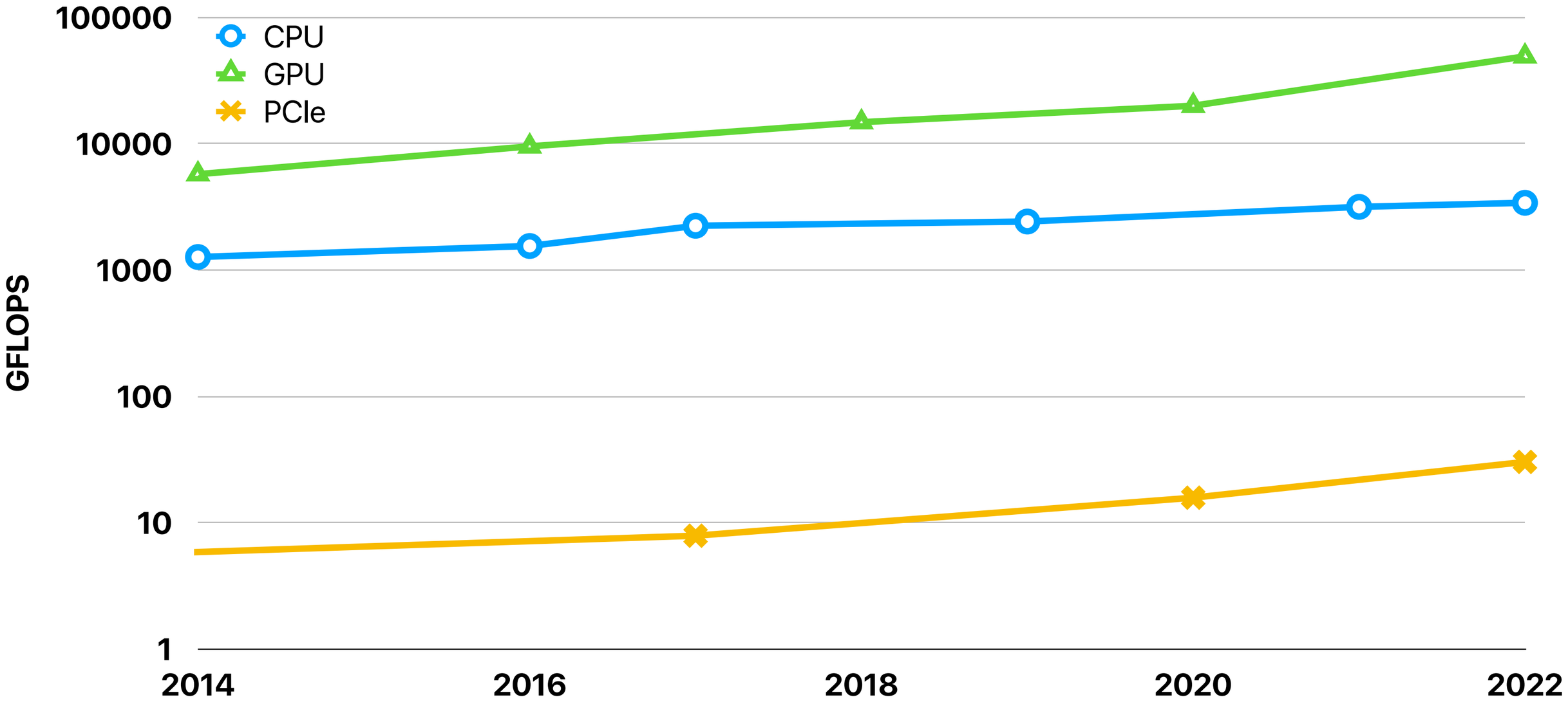
No free lunch! — Data movement as an overhead



Data movement overhead before calling GPUs

```
// Allocate memory space on the device
D_TYPE *d_a, *d_b, *d_c;
cudaMalloc((void **) &d_a, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);
cudaMalloc((void **) &d_b, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);
cudaMalloc((void **) &d_c, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE);

// copy matrix A and B from host to device memory
cudaMemcpy(d_a, a, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyHostToDevice);

unsigned int grid_rows = (ARRAY_SIZE + BLOCK_SIZE - 1) / BLOCK_SIZE;
unsigned int grid_cols = (ARRAY_SIZE + BLOCK_SIZE - 1) / BLOCK_SIZE;
dim3 dimGrid(grid_cols, grid_rows);
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

// Launch kernel
gpu_block_matrix_mult<<<dimGrid, dimBlock>>>(d_a, d_b, d_c, ARRAY_SIZE);

// Transefr results from device to host
cudaMemcpy(c, d_c, sizeof(D_TYPE)*ARRAY_SIZE*ARRAY_SIZE, cudaMemcpyDeviceToHost);
cudaThreadSynchronize();
// time counting terminate
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

// compute time elapse on GPU computing
cudaEventElapsedTime(&gpu_elapsed_time_ms, start, stop);
```

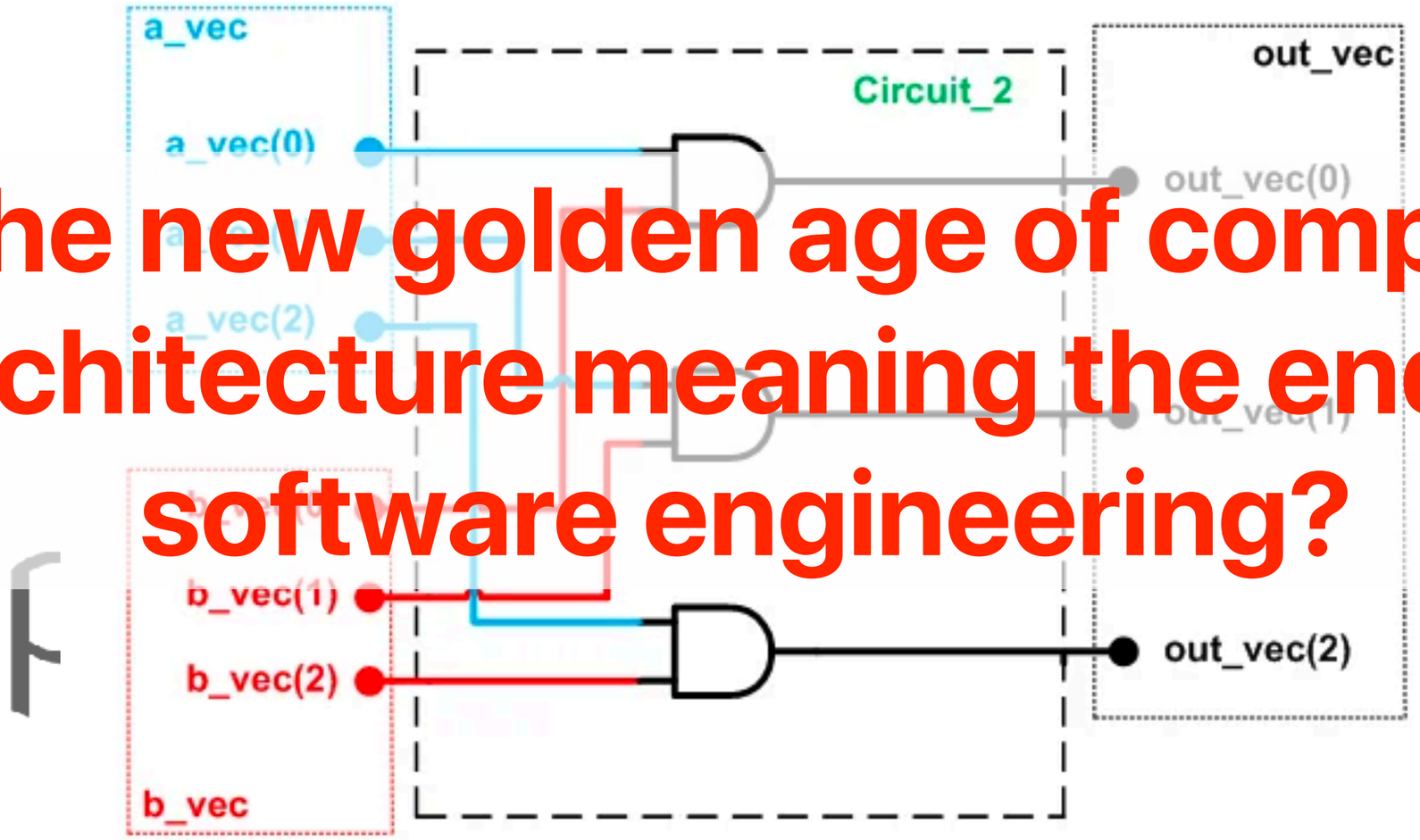
The “speed” of PCIe compared to computing



Take-aways: the new golden age of computer architectures

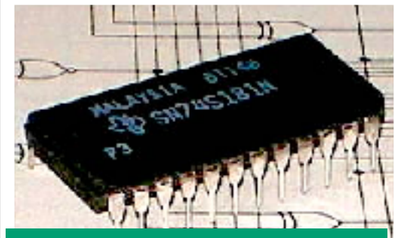
- Challenges and SOTA solutions in the dark silicon era
  - GPUs/many-core processors improve the **throughput** per-chip through providing massive parallelism where each processing element operates at a lower speed, but not-ideal for latency-sensitive workloads
  - Aggressive dynamic frequency/voltage scaling on CMP to accommodate the demand of **latency**-sensitive, parallelism-limited applications, but the area-efficiency of the slower cores is not great
  - Single ISA, heterogeneous CMPs (e.g., big.Little cores, Intel/Apple's P-cores/E-cores) find a balance the trade-offs of general-purpose workloads, but won't be ideal if your applications go to either extreme of throughput or latency
  - Domain-specific accelerators or application-specific ICs make more efficient use of chip areas to fulfill the latency or throughput demand of applications, but sacrifices flexibility and application designers are now also hardware designers

Conclusion: A New Golden Age



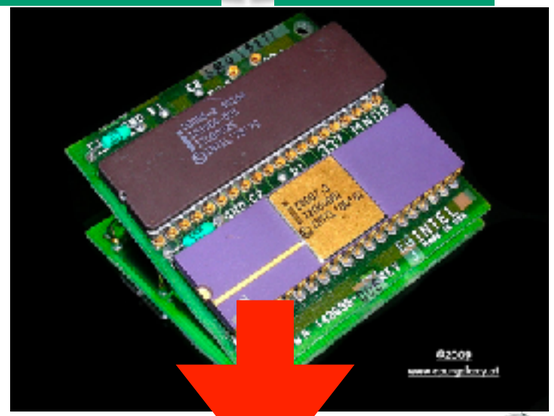
Is the new golden age of computer architecture meaning the end of software engineering?

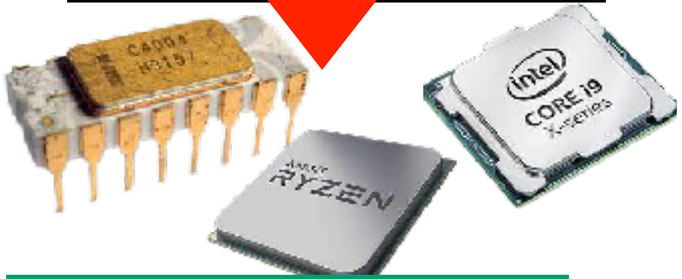
The evolutionary cycle of computing



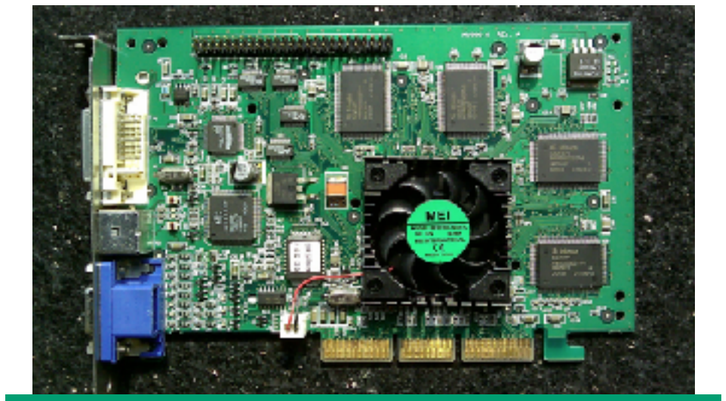
74181 Integer ALU

8087 FPU



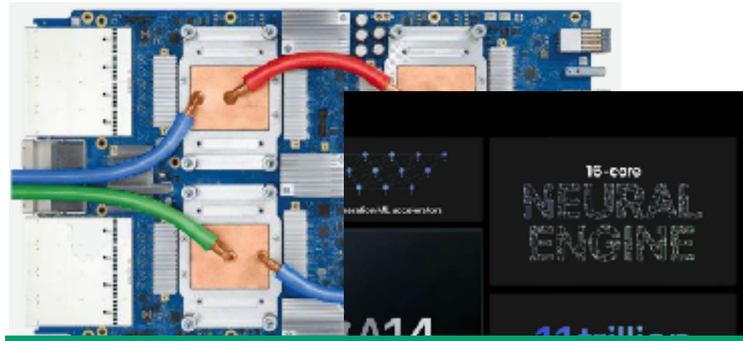


General-purpose microprocessors



Graphics "Accelerator"

General Purpose Computing On GPUs

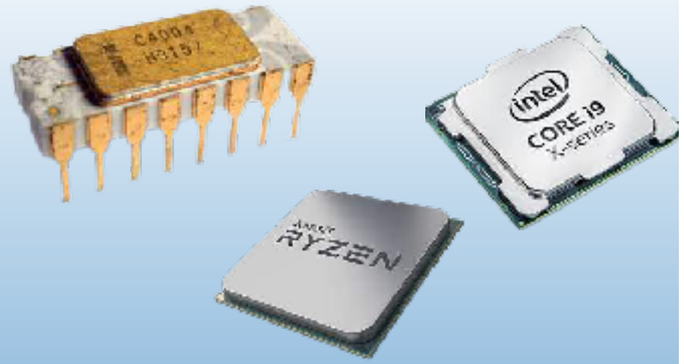


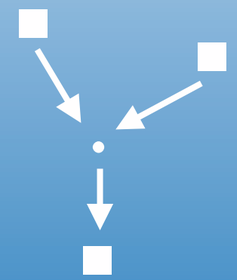
AI/ML/NN "Accelerators"

General Purpose Computing On Modern Accelerators

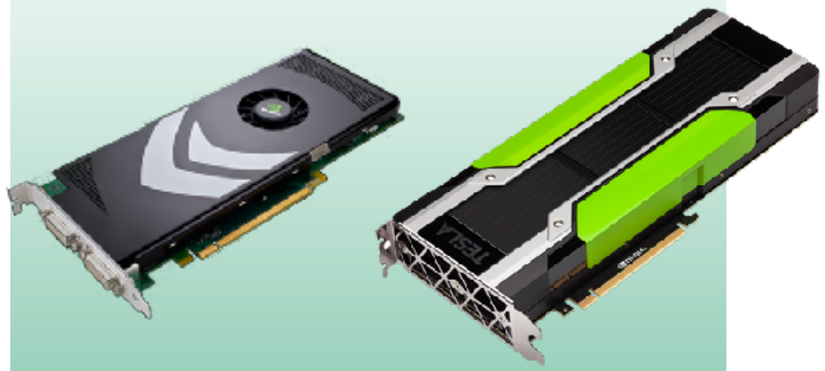
https://en.wikipedia.org/wiki/74181#/media/File:SN74S181N.JPG
https://upload.wikimedia.org/wikipedia/commons/4/4f/Intel_8087.jpg

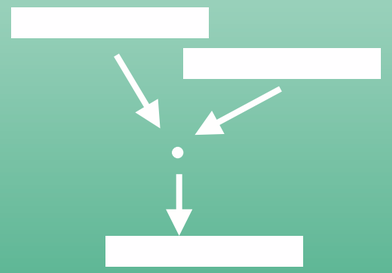
The evolutionary cycle of computing



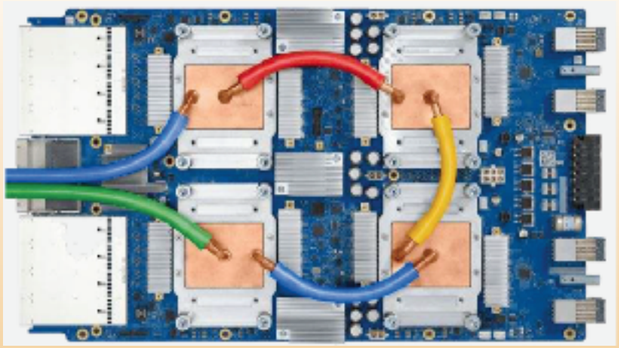


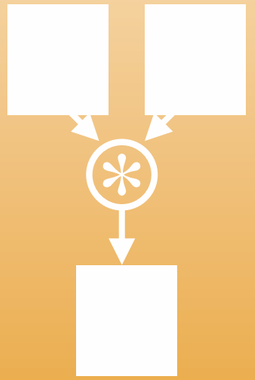
General-purpose microprocessors





General-purpose on Graphic Processing Units

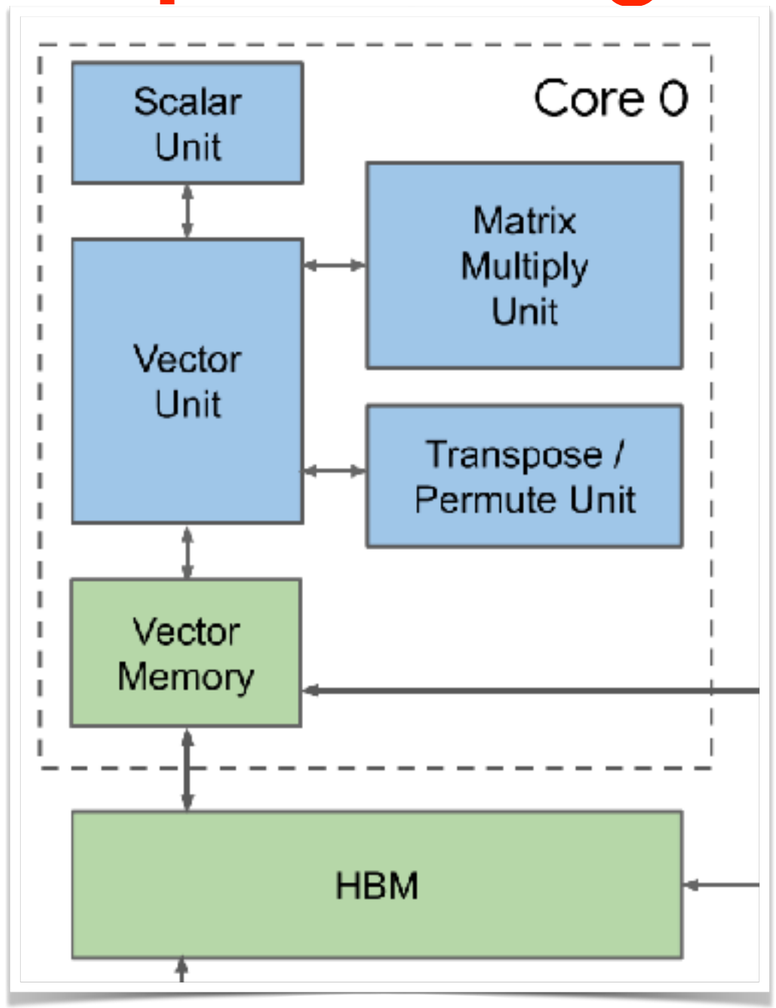


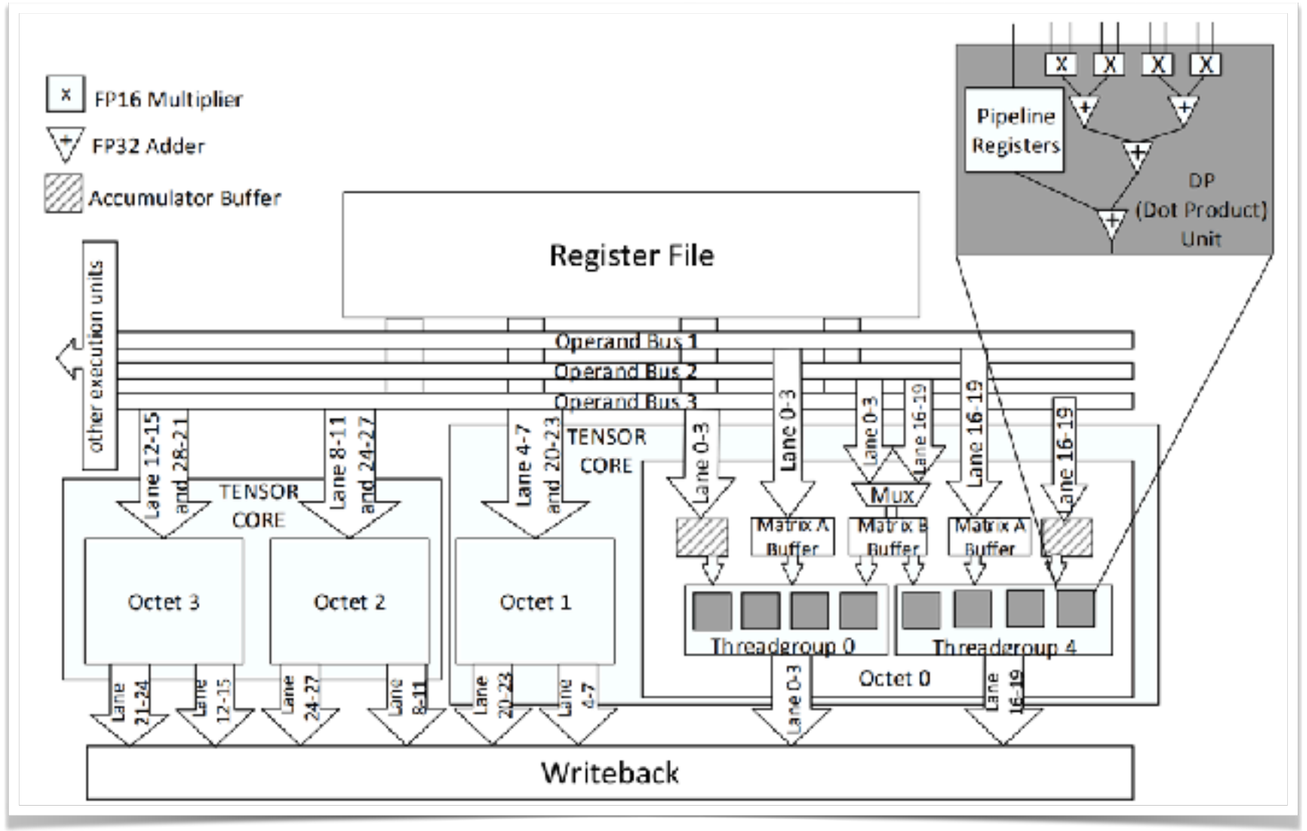


General-purpose on AI/ML accelerators

https://en.wikipedia.org/wiki/74181#/media/File:SN74S181N.JPG
https://upload.wikimedia.org/wikipedia/commons/4/4f/Intel_8087.jpg

Matrix processing — the core of AI/ML accelerators





T. Norrie et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," in IEEE Micro, vol. 41, no. 2, pp. 56-63

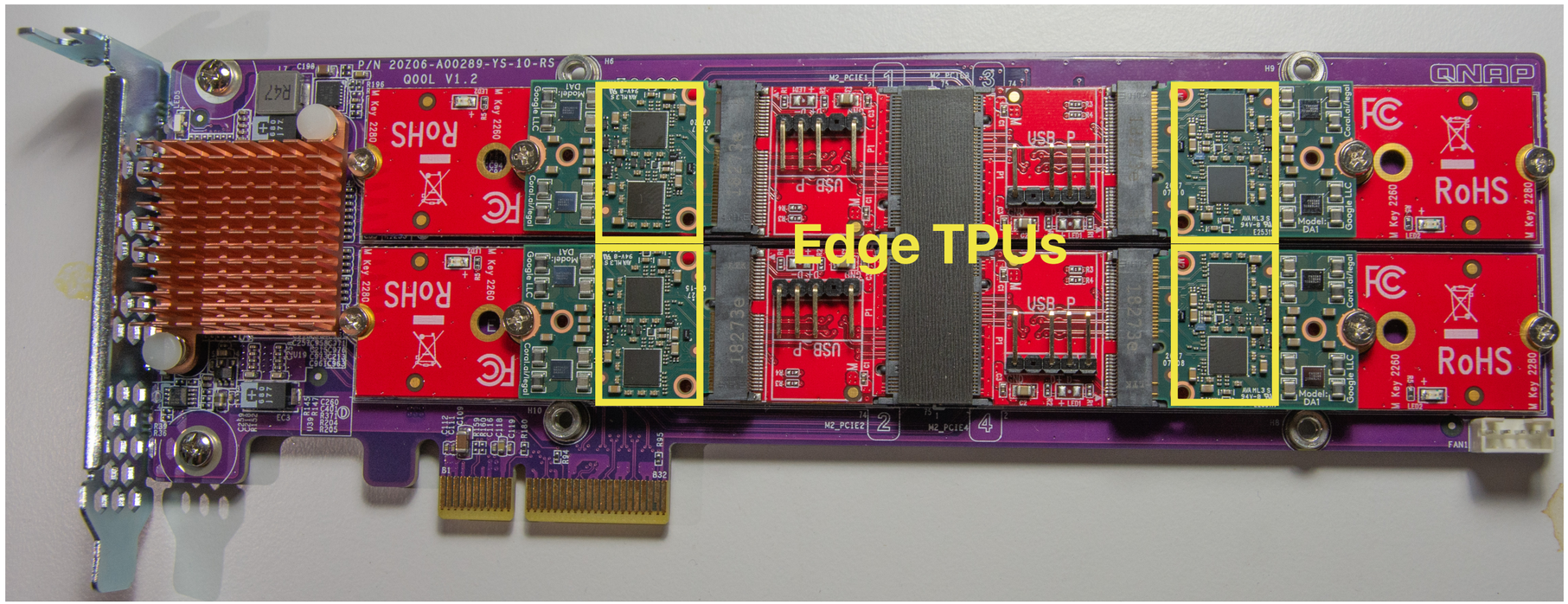
M. Raihan, N. Goli and T. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs, ISPASS 2019

Accelerating Applications using Edge Tensor Processing Units

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

ESCAL

Our GPTPU 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

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;
}
```

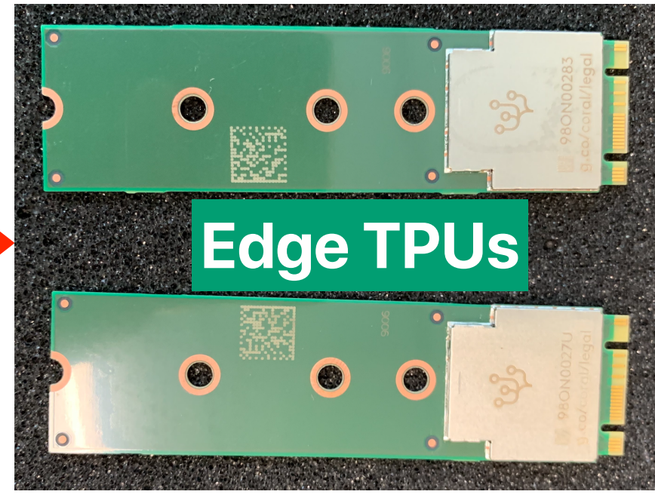
Our GPTPU framework

```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;
}
```

Input Weights

TF Lite Model

TPU Instructions



Who is my most loyal customer? — conventional version

```
SELECT A.customer, B.brand, SUM(A.Quantity * B.Price) AS value FROM A INNER JOIN B WHERE ON
A.ProductID = B.ProductID GROUP BY A.customer, B.brand;
```

Compare and sum

A

Customer	ProductID	Brand	Quant.
Abe	i9-12900	Intel	1
Abe	i7-12700	Intel	1
Abe	RTX3080	NVIDIA	1
Abe	660p	Intel	2
Bob	RyZen 5800G	AMD	1
Bob	980Pro	Samsung	1
Cindy	RTX3080	NVIDIA	1
Cindy	i7-12700	Intel	2
Cindy	660p	Intel	2
Diana	RyZen 5800G	AMD	1
Diana	RX5000	AMD	1
Diana	980Pro	Samsung	1

B

Brand	ProductID	Price
Intel	i9-12900	600
Intel	i7-12700	400
Intel	660p	100
AMD	RX5000	500
AMD	RyZen 5800G	200
NVIDIA	RTX3080	1200
Samsung	980Pro	100

Joined A*B

Customer	ProductID	Brand	Quant.	Price	Value
Abe	i9-12900	Intel	1	600	600
Abe	i7-12700	Intel	1	400	400
Abe	RTX3080	NVIDIA	1	1200	1200
Abe	660p	Intel	2	100	200
Bob	RyZen 5800G	AMD	1	400	400
Bob	980Pro	Samsung	1	100	100
Cindy	RTX3080	NVIDIA	1	1200	1200
Cindy	i7-12700	Intel	2	400	800
Cindy	660p	Intel	2	100	200
Diana	RyZen 5800G	AMD	1	400	400
Diana	RX5000	AMD	1	500	500
Diana	980Pro	Samsung	1	100	100

Customer	Brand	Value
Abe	Intel	1200
Abe	NVIDIA	1200
Bob	AMD	400
Bob	Samsung	100
Cindy	Intel	1000
Cindy	NVIDIA	1200
Diana	AMD	900
Diana	Samsung	100

Memory copy

Compare & Element-wise Memory copy

Vector multiplications

Who is my most loyal customer? — Matrix Version

```
SELECT A.customer, B.brand, SUM(A.Quantity * B.Price) AS value FROM A INNER JOIN B WHERE ON
A.ProductID = B.ProductID GROUP BY A.customer, B.brand;
```

A

Customer	ProductID	Brand	Quant.
Abe	i9-12900	Intel	1
Abe	i7-12700	Intel	1
Abe	RTX3080	NVIDIA	1
Abe	660p	Intel	2
Bob	RyZen 5800G	AMD	1
Bob	980Pro	Samsung	1
Cindy	RTX3080	NVIDIA	1
Cindy	i7-12700	Intel	2
Cindy	660p	Intel	2
Diana	RyZen 5800G	AMD	1
Diana	RX5000	AMD	1
Diana	980Pro	Samsung	1

B

Brand	ProductID	Price
Intel	i9-12900	600
Intel	i7-12700	400
Intel	660p	100
AMD	RX5000	500
AMD	RyZen 5800G	200
NVIDIA	RTX3080	1200
Samsung	980Pro	100

Customer	Intel	AMD	NVIDIA	Samsung
Abe	1200	0	1200	0
Bob	0	400	0	100
Cindy	1000	0	1200	0
Diana	0	900	0	100

Matrix multiplications

Left

Customer/ ProductID	i9-12900	i7-12700	660p	RX5000	RyZen 5800G	RTX3080	980Pro
Abe	1	1	2	0	0	1	0
Bob	0	0	0	0	1	0	1
Cindy	0	2	2	0	0	1	0
Diana	0	0	0	1	1	0	1

Right

ProductID/ Brand	Intel	AMD	NVIDIA	Samsung
i9-12900	600	0	0	0
i7-12700	400	0	0	0
660p	100	0	0	0
RX5000	0	500	0	0
RyZen 5800G	0	400	0	0
RTX3080	0	0	1200	0
980Pro	0	0	0	100

Memory copy

TCUDB: Accelerating Database with Tensor Processors

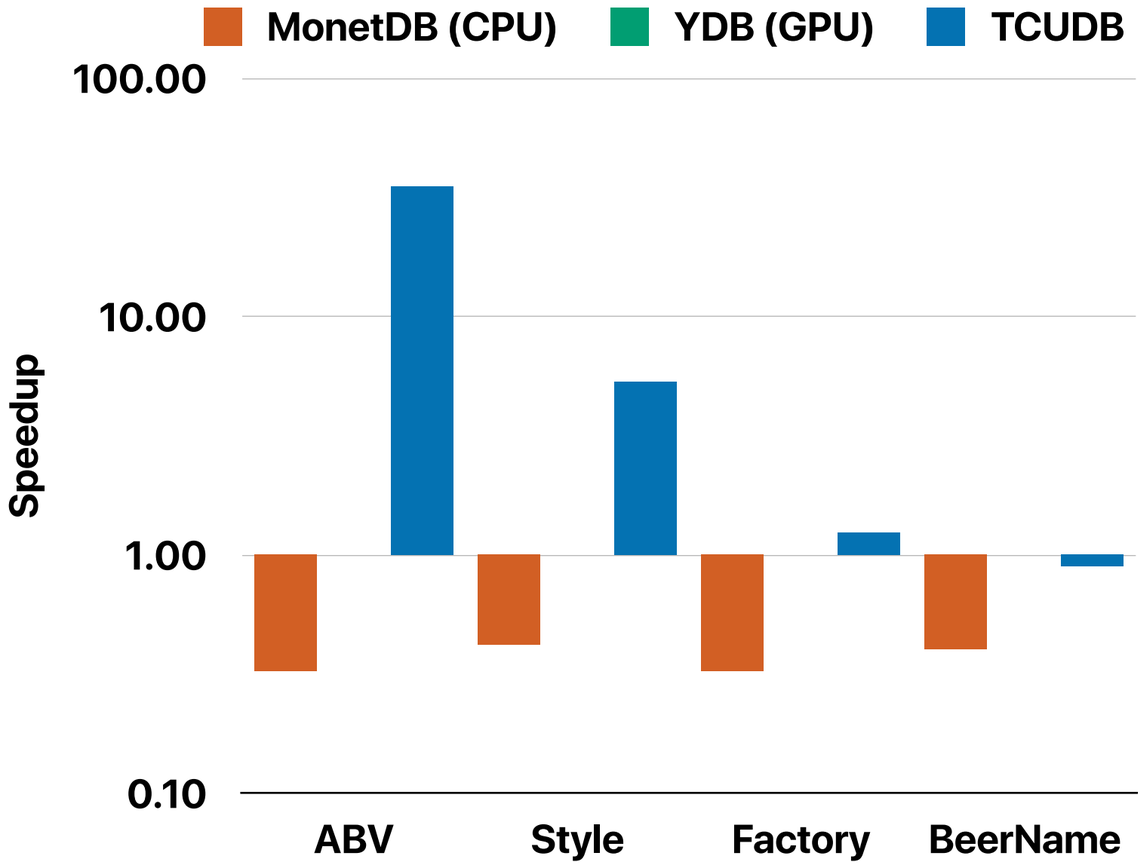
Yu-Ching Hu, Yuliang Li* and Hung-Wei Tseng
University of California, Riverside and Megagon Labs*
In The ACM SIGMOD/PODS 2022 International Conference on Management of Data.
https://github.com/escalab/TCUDB

Megagon Labs

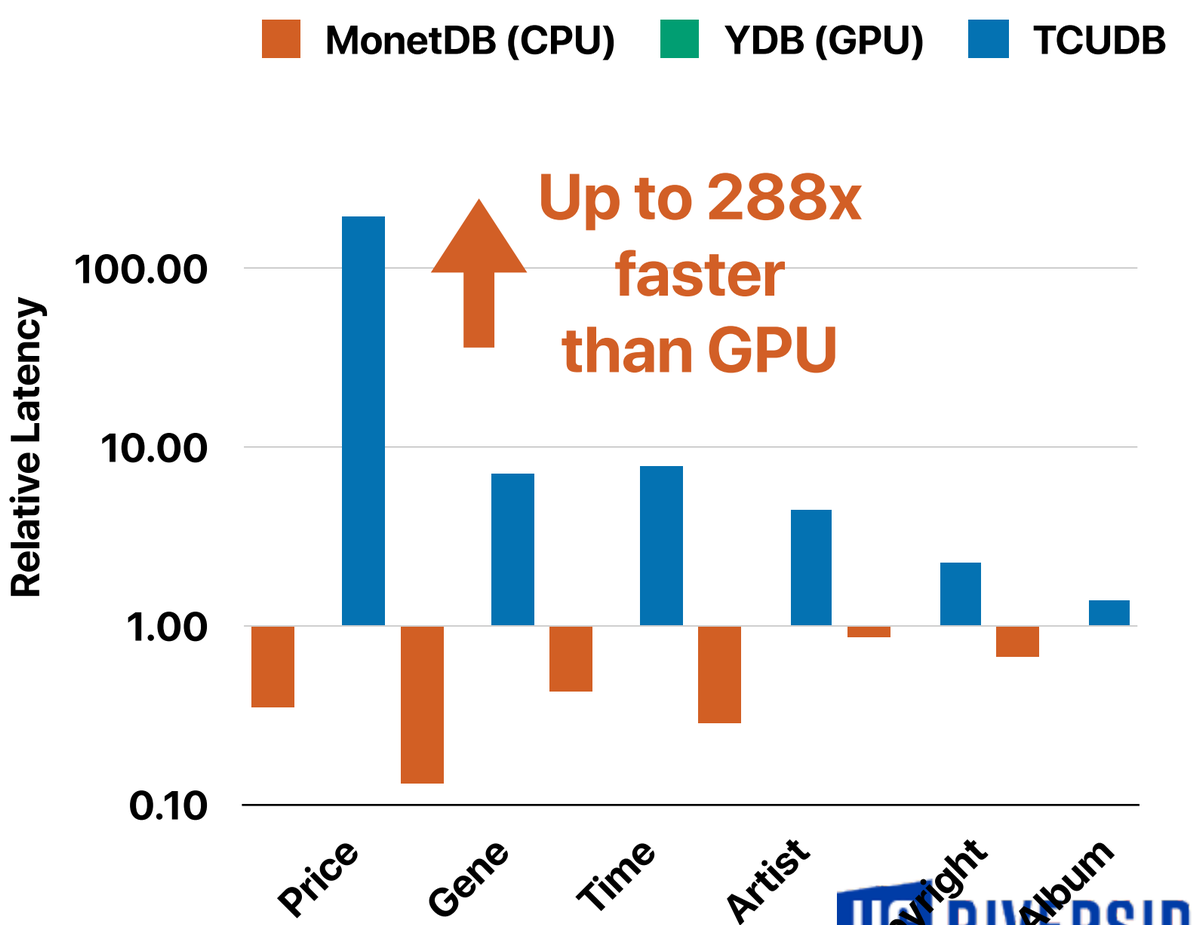
ESCAL

Entity matching

- `SELECT TABLE_A.ID, TABLE_A.BEER_NAME, TABLE_B.ID, TABLE_B.BEER_NAME FROM TABLE_A, TABLE_B WHERE TABLE_A.ABV = TABLE_B.ABV;`

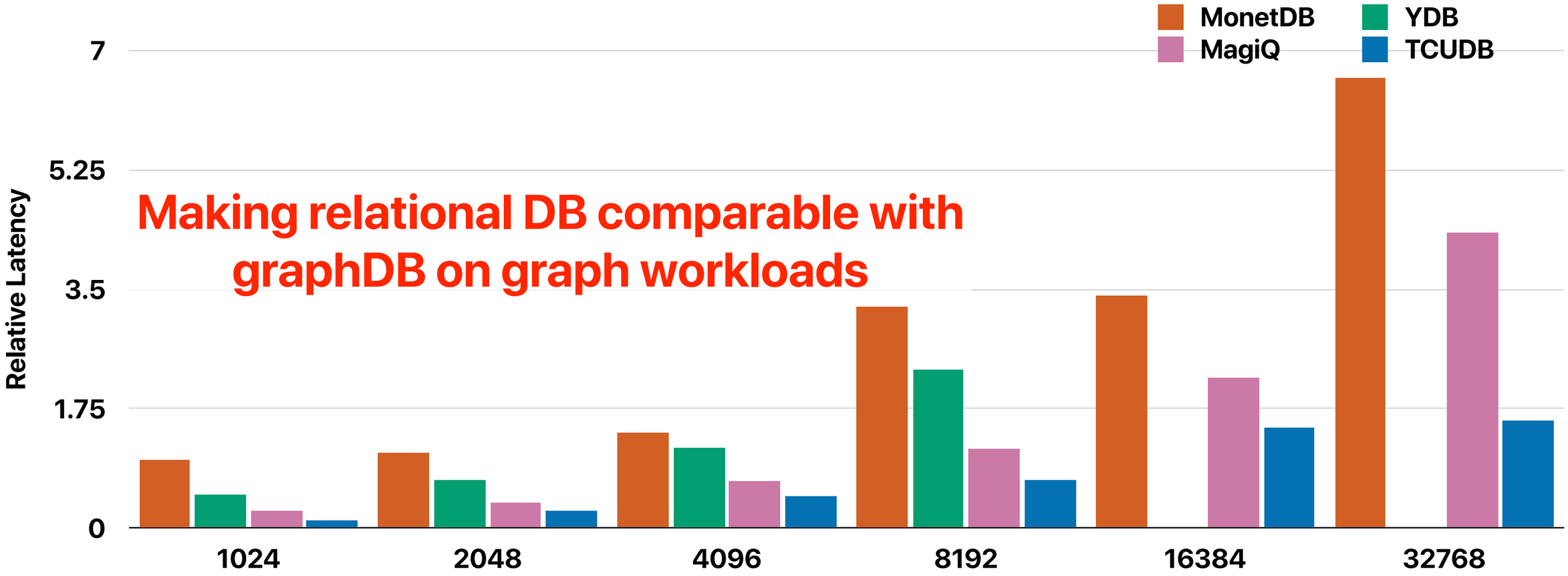


- `SELECT TABLE_A.ID, TABLE_A.SONG, TABLE_B.ID, TABLE_B.SONG FROM TABLE_A, TABLE_B WHERE TABLE_A.ARTIST = TABLE_B.ARTIST;`



PageRank

- SELECT SUM(@alpha * PAGERANK.rank / OUTDEGREE.DEGREE) + (1-@alpha)/ @num_node FROM PAGERANK, OUTDEGREE WHERE PAGERANK.ID = OUTDEGREE.ID;



Recap: Matrix Multiplications

	1	2	3	4
1	0	602	216	0
2	602	0	265	266
3	216	265	0	218
4	0	266	218	0

×

	1	2	3	4
1	0	602	216	0
2	602	0	265	266
3	216	265	0	218
4	0	266	218	0

sum(| 1 | 0 | 602 | 216 | 0 | ⊙ |

2
602
0
265
266

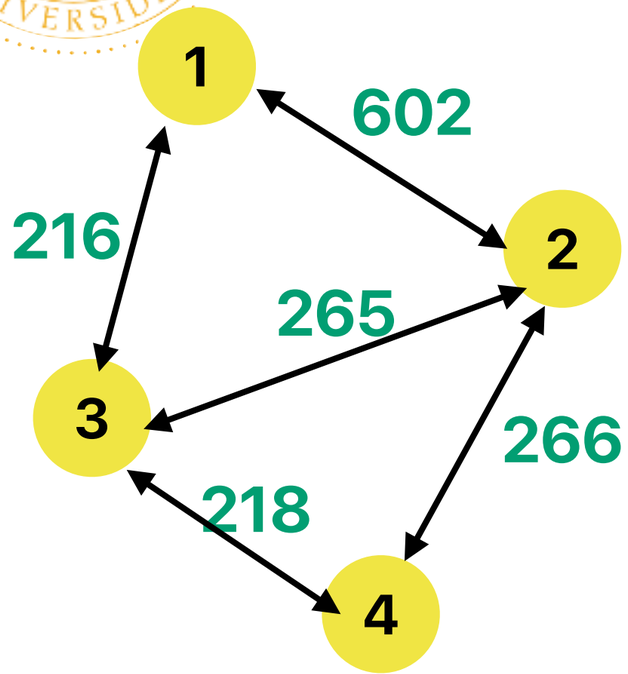
)

	1	2	3	4
1				
2	57240			
3				
4				

1st OP

2nd OP

Recap: All-Pair-Shortest-Path



	1	2	3	4
1	—	602	216	∞
2	602	—	265	266
3	216	265	—	218
4	∞	266	218	—

	1	2	3	4
1	—	602	216	∞
2	602	—	265	266
3	216	265	—	218
4	∞	266	218	—

min(1 — 602 216 ∞ + | 2: 602, —, 265, 266) , 602)

	1	2	3	4
1				
2	481			
3				
4				

1st OP

2nd OP

Similarity of matrix problems

	1st OP	2nd OP
Matrix Multiplications	+	×
All pair shortest path	min	+
Critical path	max	+
Minimum reliability paths	min	×
Maximum reliability paths	max	×
Minimum spanning tree	min	max
Maximum capacity paths	max	min
Transitive and reflexive closure	or	and
L2 Distance	+	$\|a-b\|^2$

SIMD²: A Generalized Matrix Instruction Set for Accelerating Tensor Computation beyond GEM

Yunan Zhang, Po-An Tsai and Hung-Wei Tseng

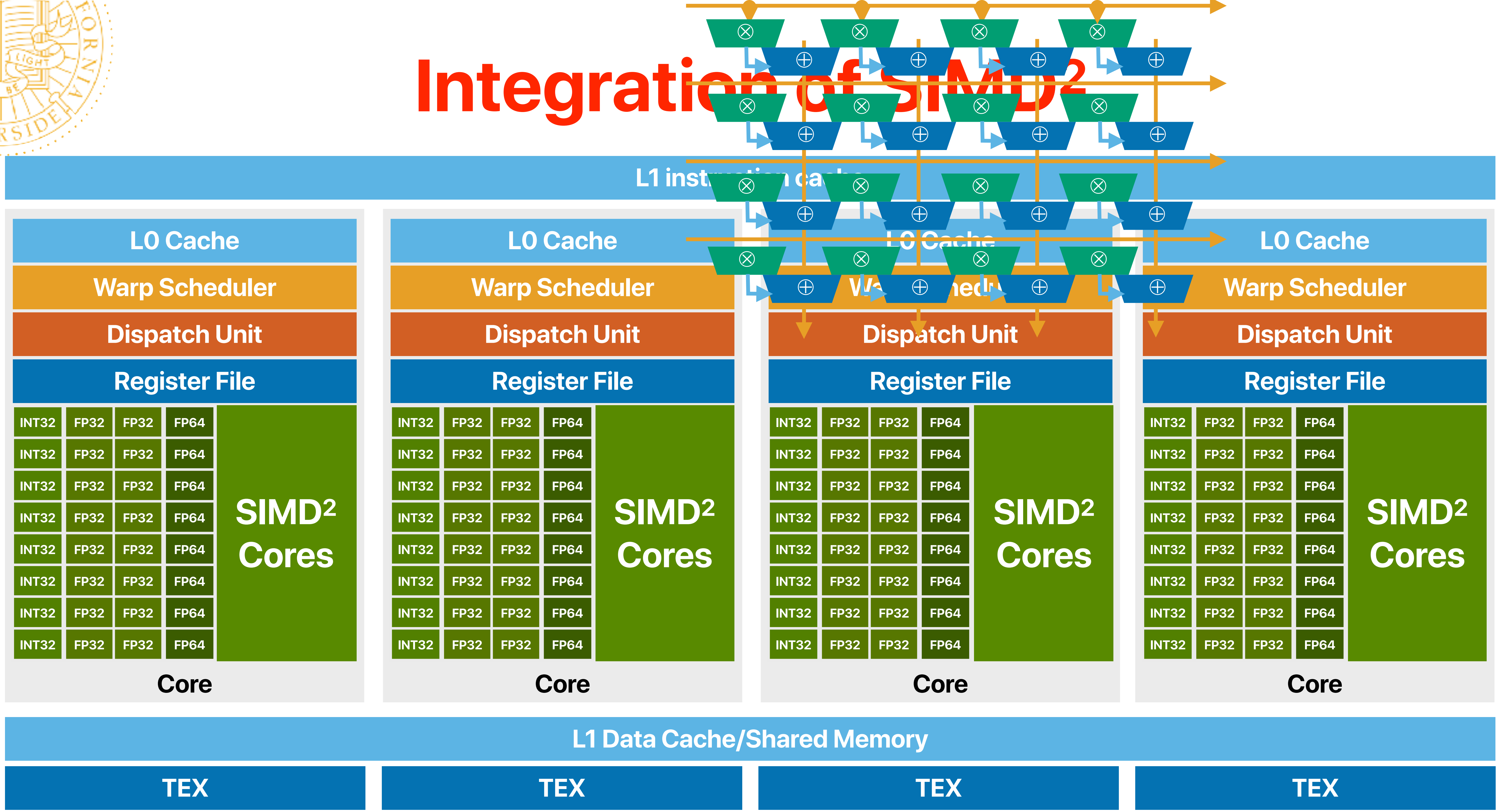
In The International Symposium on Computer Architecture (ISCA). 2022

https://github.com/escalab/SIMD2

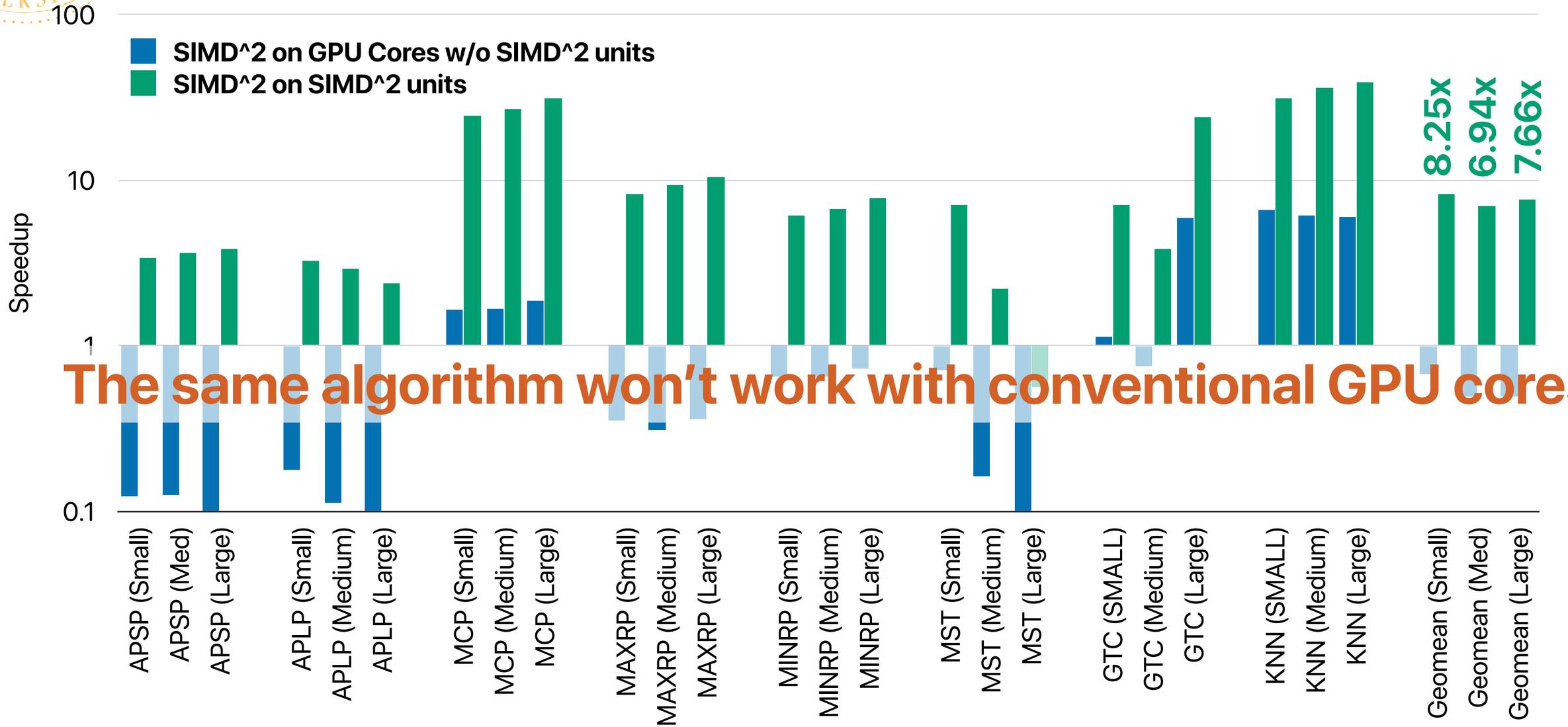
SIMD² instructions

	1st OP	2nd OP
Matrix Multiplications	+	×
All pair shortest path	min	+
Critical path	max	+
Minimum reliability paths	min	×
Maximum reliability paths	max	×
Minimum spanning tree	min	max
Maximum capacity paths	max	min
Transitive and reflexive closure	or	and
L2 Distance	+	$\|a-b\|^2$

Integration of SIMD^2



SIMD² performance



The same algorithm won't work with conventional GPU cores

Final words

Conclusion

- Computer architecture is now more important than you could ever imagine in the dark silicon era + the "new" golden age of computer architecture
- Just being a "programmer" is easy. You need to know architecture a lot to be a "performance programmer"
  - Optimizing locality and footprint for caching
  - Make your code or data more branch prediction friendly
  - Exploiting more instruction-level parallelism via carefully revisiting the critical path of execution
- Multicore era — to get your multithreaded program correct and perform well, you need to take care of coherence and consistency
- We're now in the "dark silicon era"
  - Single-core isn't getting any faster
  - Multi-core doesn't scale anymore
  - We will see more and more ASICs
  - You need to write more "system-level" programs to use these new ASICs.
- Can we be more creative in using ASICs?

Thank you all for this great quarter!
Let’s take a group photo now!


Computer
Science &
Engineering
142
ありがとう
Prof. Usagi

One more thing...

Sample Final

Format of the online final

- Multiple choices (15 questions)
  - They’re like your clicker/midterm multiple choices questions
  - Cumulative, don’t forget your midterm and midterm review
- Problem sets * 3 problem sets,
- Open-ended * 5 questions in total
  - Explain your answer clearly and “correctly”. If you include “incorrect” information in your answer, it’s going to hurt your grade
  - May not have a standard answer. You need to understand the concepts to provide a good answer
  - You should review your assignments carefully

Multiple choices

How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

A. 1
B. 2
C. 3
D. 4
E. 5

The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

① `movl (%rdi), %ecx`
② `addl %ecx, %eax`
③ `addq $4, %rdi`
④ `cmpq %rdx, %rdi`
⑤ `jne .L3`
⑥ `ret`

A. (1) & (2)
B. (2) & (3)
C. (3) & (4)
D. (4) & (5)
E. None of the pairs can be reordered

Pros/Cons of SMT

- How many of the following statements about SMT is/are correct?
  - ① SMT makes processors with deep pipelines more tolerable to mis-predicted branches
  - ② SMT can improve the throughput of a single-threaded application
  - ③ SMT processors can better utilize hardware during cache misses comparing with superscalar processors with the same issue width
  - ④ SMT processors can have higher cache miss rates comparing with superscalar processors with the same cache sizes when executing the same set of applications.
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Linked list v.s. arrays

- We can use either a linked list or an array to store a list of data, compare the performance of the array (version A) and linked list (version B) implementations that can achieve the same outcome as below. Assume we have a processor with a reasonably good branch predictor and unlimited fetch/issue width, please identify the correct statements.
  - ① If the dataset is large, the A will outperform the B
  - ② If the dataset is small, there is very little performance difference between A and B
  - ③ If the dataset is small, B will outperform A as A has more branch instructions
  - ④ If the dataset is small, A will outperform B as A has fewer data dependencies

A. 0
B. 1
C. 2
D. 3
E. 4

A

```
for(i=0;i<size;i++)
{
    if(node[i].next)
        number_of_nodes++;
}
```

B

```
while(node)
{
    node = node->next;
    number_of_nodes++;
}
```

Why is C better than B?

- How many of the following statements explains the reason why B outperforms C with compiler optimizations
  - ① C has lower dynamic instruction count than B
  - ② C has significantly lower branch mis-prediction rate than B
  - ③ C has significantly fewer branch instructions than B
  - ④ C has better CPI than B

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

C

```
inline int popcount(uint64_t x) {
      int c = 0;
      int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
      while(x)      {
          c += table[(x & 0xF)];
          x = x >> 4;
      }
      return c;
}
```

B

```
inline int popcount(uint64_t x) {
    int c = 0;
    while(x)      {
      c += x & 1;
      x = x >> 1;
      c += x & 1;
      x = x >> 1;
      c += x & 1;
      x = x >> 1;
      c += x & 1;
      x = x >> 1;
    }
    return c;
}
```

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  - ① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
  - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
  - ③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
  - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way

Power & Energy

- Regarding dynamic frequency scaling, how many of the following statements are correct?
  - ① Lowering the frequency helps extending the battery life
  - ② Lowering the frequency helps reducing the heat generation
  - ③ Lowering the frequency helps reducing the electricity bill
  - ④ A CPU operating at 25% of the peak frequency can still consume more than 50% of the peak power
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Why is D better than C?

- How many of the following statements explains the main reason why B outperforms C with compiler optimizations
  - ① D has lower dynamic instruction count than C
  - ② D has significantly lower branch mis-prediction rate than C
  - ③ D has significantly fewer branch instructions than C
  - ④ D can incur fewer memory accesses than C

A. 0
B. 1
C. 2
D. 3
E. 4

C

```
inline int popcount(uint64_t x) {
      int c = 0;
      int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
      while(x)      {
          c += table[(x & 0xF)];
          x = x >> 4;
      }
      return c;
}
```

D

```
inline int popcount(uint64_t x) {
      int c = 0;
      int table[16] = {0, 1, 1, 2, 1,
2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
      for (uint64_t i = 0; i < 16; i++)
      {
          c += table[(x & 0xF)];
          x = x >> 4;
      }
      return c;
 }
```

Demo revisited

- Why the performance is better when option is not “0”
  - ① The amount of dynamic instructions needs to execute is a lot smaller
  - ② The amount of branch instructions to execute is smaller
  - ③ The amount of branch mis-predictions is smaller
  - ④ The amount of data accesses is smaller

A. 0
B. 1
C. 2
D. 3
E. 4

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```

What if the code look like this?

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 64KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

What data structure is performing better

	Array of objects	object of arrays

	`struct grades`
	`{`
	`  int id;`
	`  double *homework;`
	`  double average;`
	`};`
		`struct grades`
		`{`
		`  int *id;`
		`  double **homework;`
		`  double *average;`
		`};`

average of each homework	`for(i=0;i<homework_items; i++)`
	`{`
	`gradesheet[total_number_students].homework[i] = 0.0;`
	`  for(j=0;j<total_number_students;j++)`
	`gradesheet[total_number_students].homework[i]`
	`+=gradesheet[j].homework[i];`
	`  gradesheet[total_number_students].homework[i] /=`
	`(double)total_number_students;`
	`}`
		`for(i = 0;i < homework_items; i++)`
		`{`
		`  gradesheet.homework[i][total_number_students] = 0.0;`
		`  for(j = 0; j <total_number_students;j++)`
		`  {`
		`  gradesheet.homework[i][total_number_students] +=`
		`gradesheet.homework[i][j];`
		`  }`
		`  gradesheet.homework[i][total_number_students] /=`
		`total_number_students;`
		`}`

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance?
  A. Array of objects
  B. Object of arrays

Cache coherency

- Assuming that we are running the following code on a CMP with a cache coherency protocol, how many of the following outputs are possible? (a is initialized to 0 as assume we will output more than 10 numbers)

thread 1	thread 2
`while(1)` `  printf("%d ",a);`	`while(1)` `  a++;`

① 0 1 2 3 4 5 6 7 8 9

② 1 2 5 9 3 6 8 10 12 13

③ 1 1 1 1 1 1 1 64 100

④ 1 1 1 1 1 1 1 1 1 100

A. 0

B. 1

C. 2

D. 3

E. 4

Performance comparison

- Comparing implementations of thread_vadd — L and R, please identify which one will be performing better and why

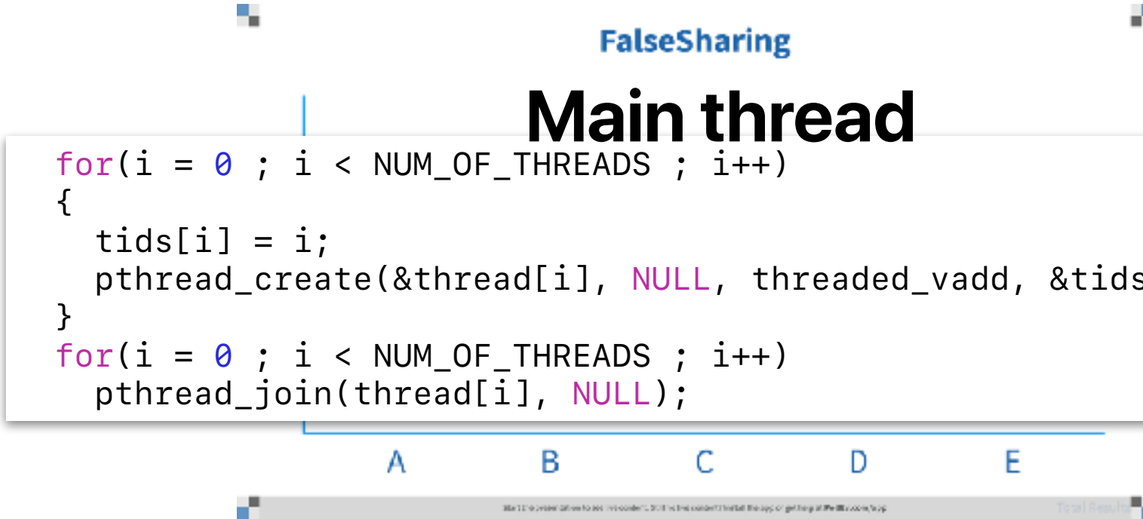
Version L

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid;i<ARRAY_SIZE;i+=NUM_OF_THREADS)
  {
        c[i] = a[i] + b[i];
  }
  return NULL;
}
```

Version R

```
void *threaded_vadd(void *thread_id)
{
  int tid = *(int *)thread_id;
  int i;
  for(i=tid*(ARRAY_SIZE/NUM_OF_THREADS);i<(tid+1)*(ARRAY_SIZE/NUM_OF_THREADS);i++)
  {
      c[i] = a[i] + b[i];
  }
  return NULL;
}
```

A. L is better, because the cache miss rate is lower
B. R is better, because the cache miss rate is lower
C. L is better, because the instruction count is lower
D. R is better, because the instruction count is lower
E. Both are about the same



Free-answer questions

Branch prediction performance

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

- What's the branch prediction accuracy if the processor uses?
  - A 2-bit local predictor
  - A (4, 2) global history predictor
- Can you rewrite the code to generate the same result but eliminate branch X?

Register renaming

- For the following C code:

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

- Assume we have a dual-fetch, dual-issue, out-of-order pipeline where
  - INT ALU takes 1 cycle
  - MEM pipeline: 5 cycles in total
  - BR takes 1 cycle to resolve
- If the loop is taken twice, how many cycles it takes to issue all instructions?
- If the loop is taken 100 times, what's the average CPI?

Cache simulation

- The processor has a 48KB, 64B blocked, 12-way L1 cache. Consider the following code:

```
for(i=0;i<256;i++) {
    a[i] = b[i] + c[i];
// load a[i] and load b[i], store to c[i]
// &a[0] = 0x10000, &b[0] = 0x20000, &c[0] = 0x30000
}
```

- What's the total miss rate? How many of the misses are compulsory misses? How many of the misses are conflict misses?
- How can you improve the cache performance of the above code through changing hardware?
- How can you improve the performance **without** changing hardware?

Open-ended questions

SMT v.s. CMP

- An SMT processor is basically a SuperScalar processor with multiple instruction front-end. Assume within the same chip area, we can build an SMT processor supporting 4 threads, with 6-issue pipeline, 64KB cache or — a CMP with 4x 2-issue pipeline & 16KB cache in each core. Please identify how many of the following statements are/is correct when running programs on these processors.
  - ① If we are just running one program in the system, the program will perform better on an SMT processor
  - ② If we are running 4 applications simultaneously, the cache miss rates will be higher in the SMT processor
  - ③ If we are running 4 applications simultaneously, the branch mis-prediction will be higher in the SMT processor
  - ④ If we are running one program with 4 parallel threads, the cache miss rates will be higher in the SMT processor
  - ⑤ If we are running one program with 4 parallel threads simultaneously, the branch mis-prediction will be longer in the SMT processor

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

Other open-ended questions (cont.)

- What's Amdahl's Law implication on parallelism? How does that guide future software design?
- What's Dark Silicon Problem? What are the new design trends in addressing the problem?
- If the OoO pipeline is highly optimized, do we still care about the ISA design?
- What's volatile? What's inline? Why we need them? The pros and cons?
- What software/compiler optimization techniques are still useful or becoming less beneficial in the era of GPU/vectorized processing?
- Revisit the code to accelerate an application this originally single-threaded on SMT/CMP?
- What software programmer should do to optimize application performance in the era of hardware accelerators

Announcements

- Final Review tomorrow during the lecture
- **Course evaluation** started and ends on 9/05/2024
  - Submit the prove of your participation in course evaluation through Gradescope
  - It can become a full credit reading quiz (it helps to amortize the penalty of another least performing one) — we will drop a total of three now
- **Assignment 5** is **due 9/05/2024**
  - The due date is earlier to allow publication of solutions before the deadline
- **Final exam**
  - 9/6 3p-6p @ **PCH 121**
  - Closed book, no cheatsheet — the same rules as the midterm
  - Pizza party after the examine

Computer Science & Engineering

142

ありがとう