The Golden Age of Computer Architecture

Hung-Wei Tseng

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

Main thread

```
for(i = 0 ; i < NUM_OF_THREADS ; i++)
{
  tids[i] = i;
  pthread_create(&thread[i], NULL, threaded_vadd, &tids
}
for(i = 0 ; i < NUM_OF_THREADS ; i++)
  pthread_join(thread[i], NULL);
```

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!

Again — how many values are possible?

- Consider the given program. You can safely assume the caches are coherent. How many of the following outputs will you see?

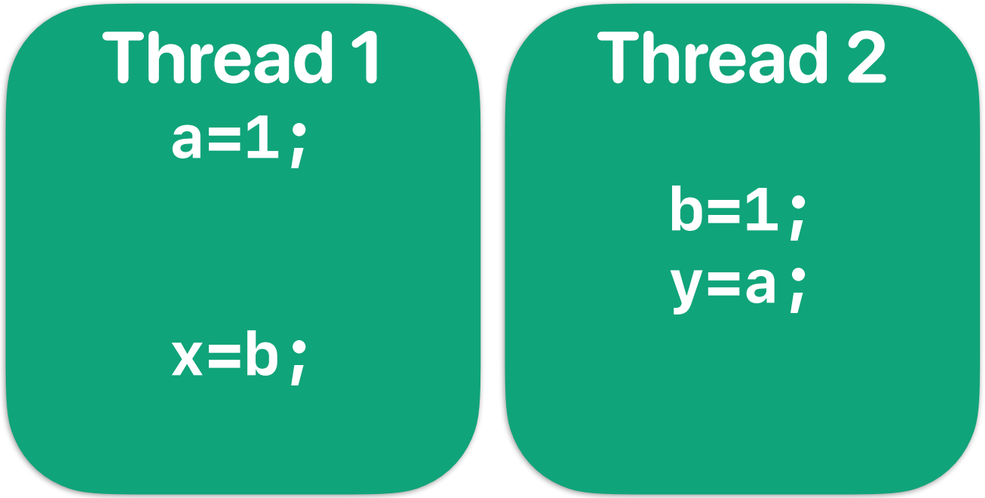
① (0, 0)
② (0, 1)
③ (1, 0)
④ (1, 1)

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

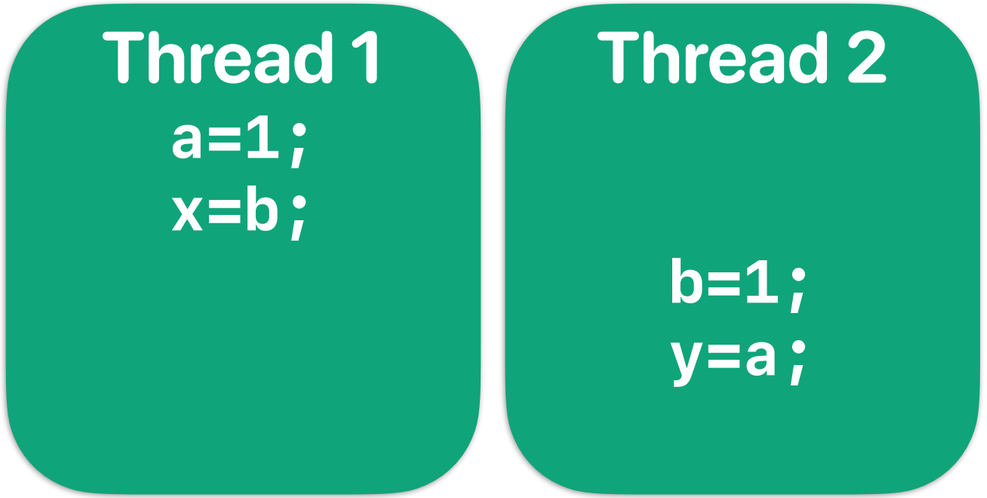
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

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;
}
```

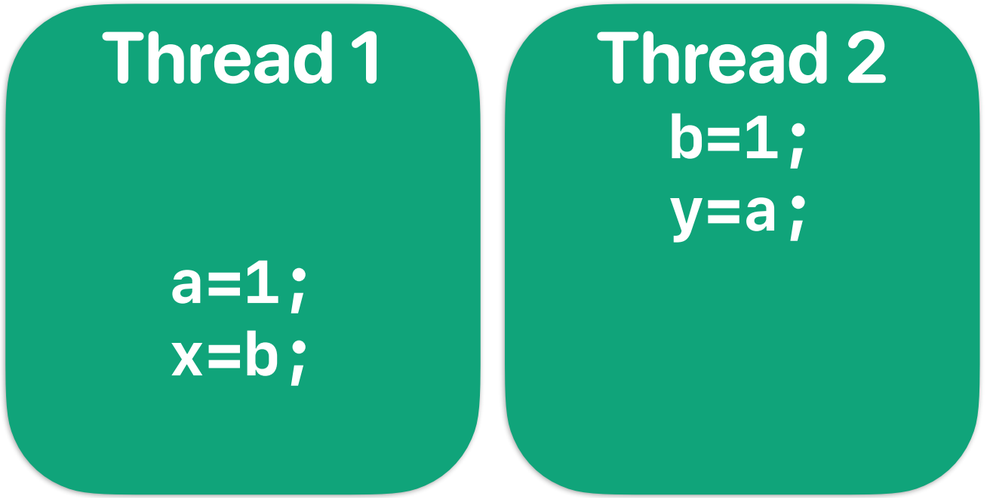
Possible scenarios



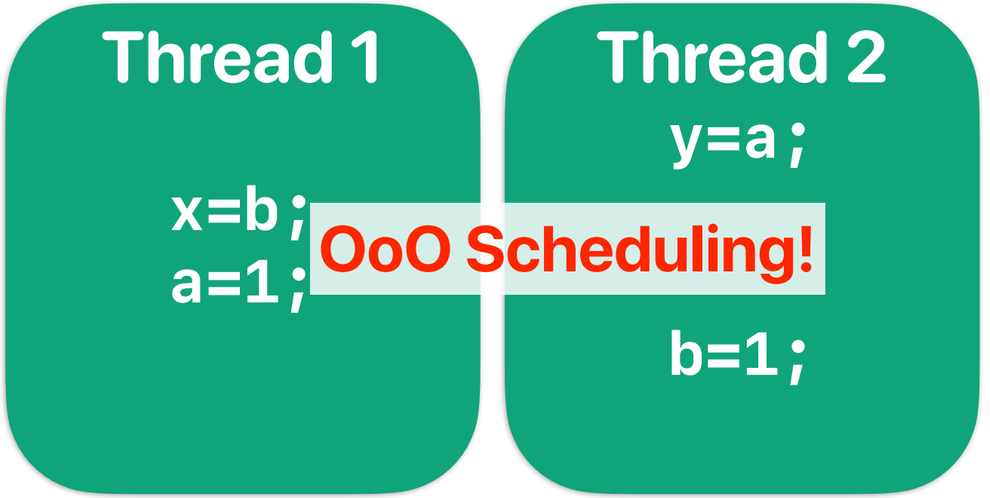
(1,1)



(0,1)



(1,0)



(0,0)

Take-aways of parallel programming

- Processor behaviors are non-deterministic
  - You cannot predict which processor is going faster
  - You cannot predict when OS is going to schedule your thread
- Cache coherency only guarantees that everyone would eventually have a coherent view of data, but not when
- Cache consistency is hard to support

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

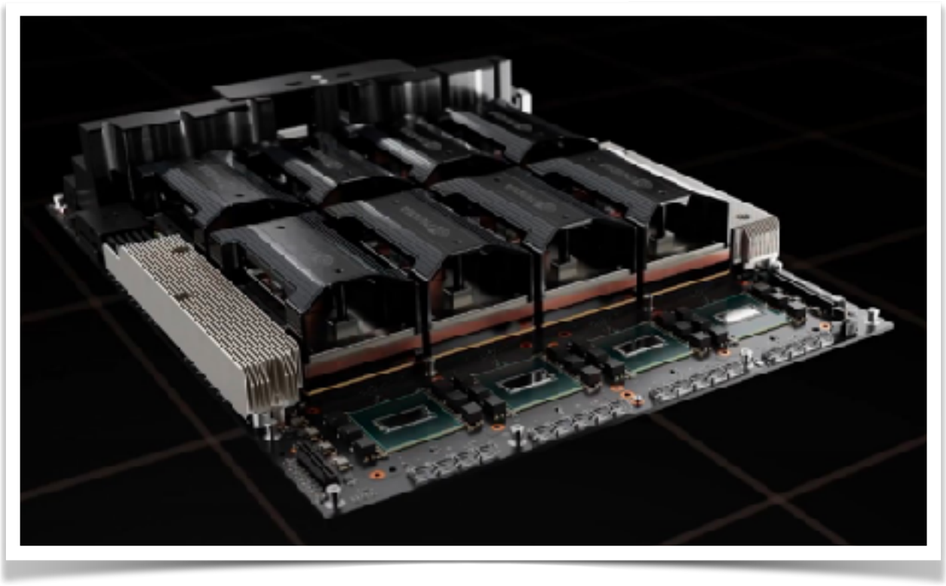
Dark!

=100W!

If you can add power budget…

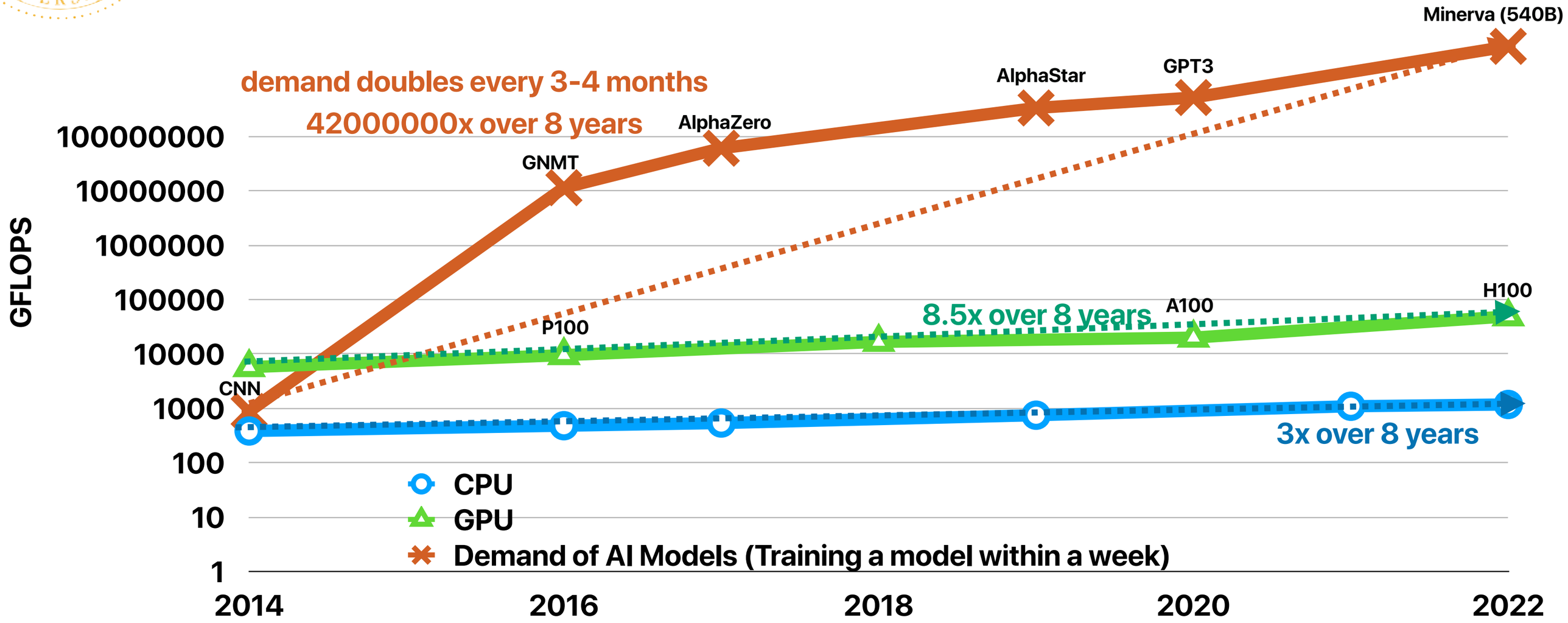
NVIDIA Accelerator Specification Comparison			
	H100	A100 (80GB)	V100
FP32 CUDA Cores	16896	6912	5120
Tensor Cores	528	432	640
Boost Clock	~1.78GHz (Not Finalized)	1.41GHz	1.53GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	1.75Gbps HBM2
Memory Bus Width	5120-bit	5120-bit	4096-bit
Memory Bandwidth	3TB/sec	2TB/sec	900GB/sec
VRAM	80GB	80GB	16GB/32GB
FP32 Vector	60 TFLOPS	19.5 TFLOPS	15.7 TFLOPS
FP64 Vector	30 TFLOPS	9.7 TFLOPS (1/2 FP32 rate)	7.8 TFLOPS (1/2 FP32 rate)
INT8 Tensor	2000 TOPS	624 TOPS	N/A
FP16 Tensor	1000 TFLOPS	312 TFLOPS	125 TFLOPS
TF32 Tensor	500 TFLOPS	156 TFLOPS	N/A
FP64 Tensor	60 TFLOPS	19.5 TFLOPS	N/A
Interconnect	NVLink 4 18 Links (900GB/sec)	NVLink 3 12 Links (600GB/sec)	NVLink 2 6 Links (300GB/sec)
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

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

Mis-matching AI/ML demand and general-purpose processing



https://ourworldindata.org/grapher/artificial-intelligence-training-computation

Outline

- The Golden Age of Computer Architecture

Solutions/trends in dark silicon era

Trends in the Dark Silicon Era

- Aggressive dynamic voltage/frequency scaling
- Throughout oriented — slower, but more
- Just let it dark — activate part of circuits, but not all
- From general-purpose to domain-specific — ASIC

Aggressive dynamic frequency/ voltage scaling

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:

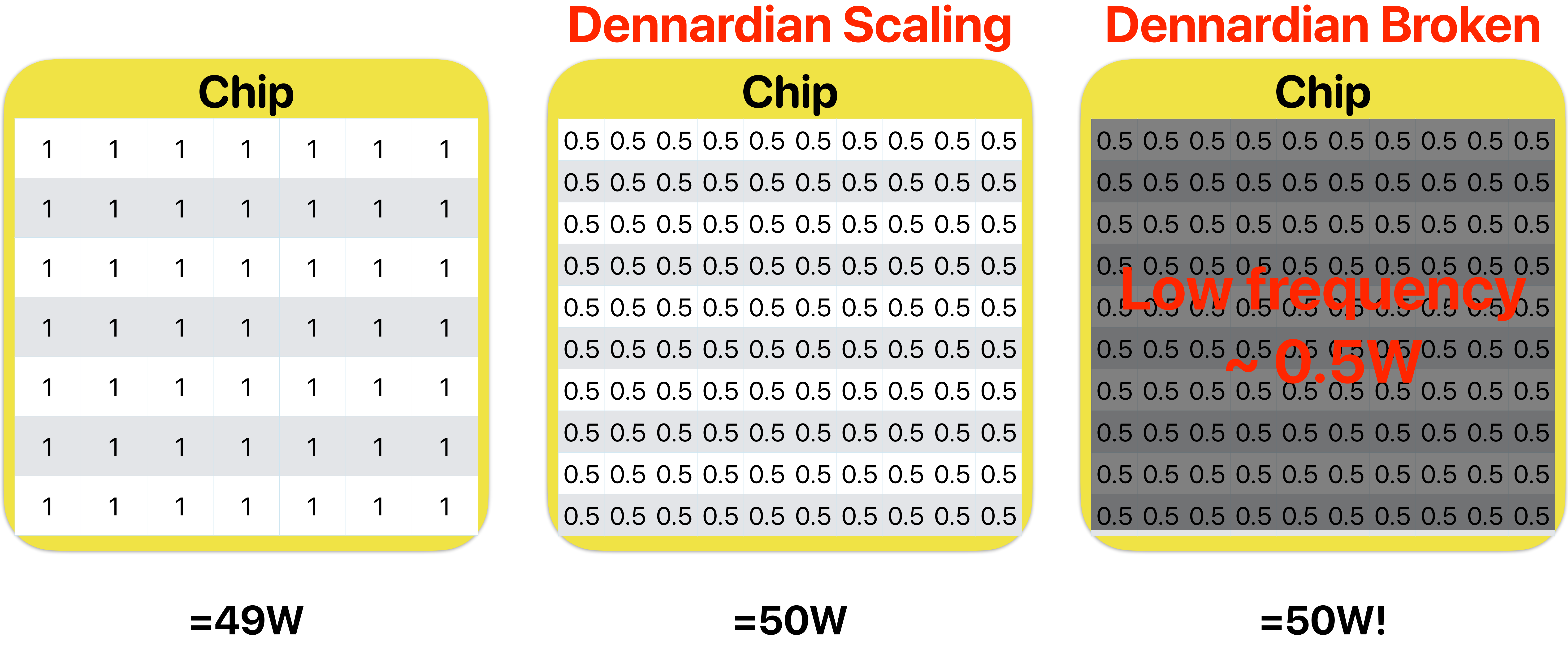
$$P_{density} = \frac{P}{area}$$

Lower the frequency can reduce active power

Moore's Law allows higher frequencies as transistors are smaller

Moore's Law makes this smaller

Or we have to dim down all transistors



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
```

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!

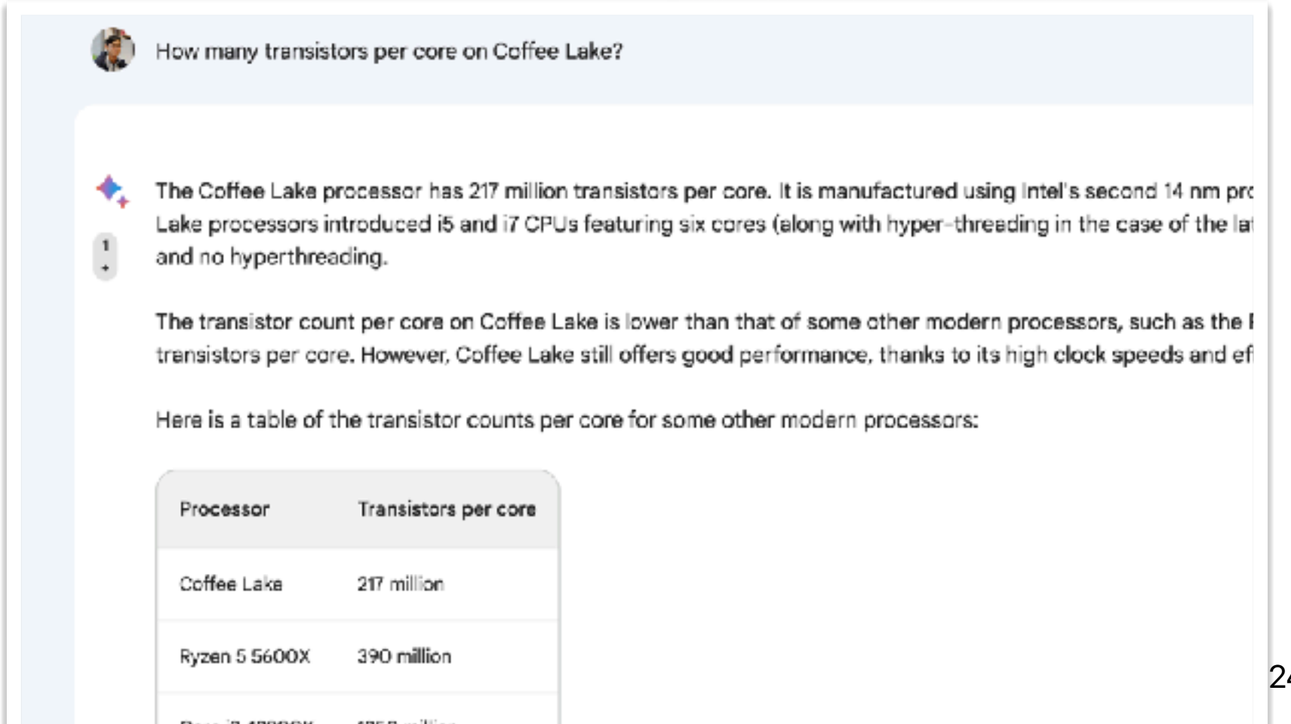
Trends in the Dark Silicon Era

- Aggressive dynamic voltage/frequency scaling
  - Potentially hurts performance and increases **energy consumption**
  - Trade-offs between UX/Power/Energy
- Throughout oriented — slower, but more
- Just let it dark — activate part of circuits, but not all
- From general-purpose to domain-specific — ASIC

Slower, but more

Transistor counts

Microarchitecture	Transistor Count	Issue-width	Year
Alder Lake	325 M	5x ALU, 7x Memory	2021
Coffee Lake	217 M	4x ALU, 4x Memory	2017
Sandy Bridge	290 M	3x ALU, 3x Memory	2011
Nehalem	182.75 M	3x ALU, 3x Memory	2008



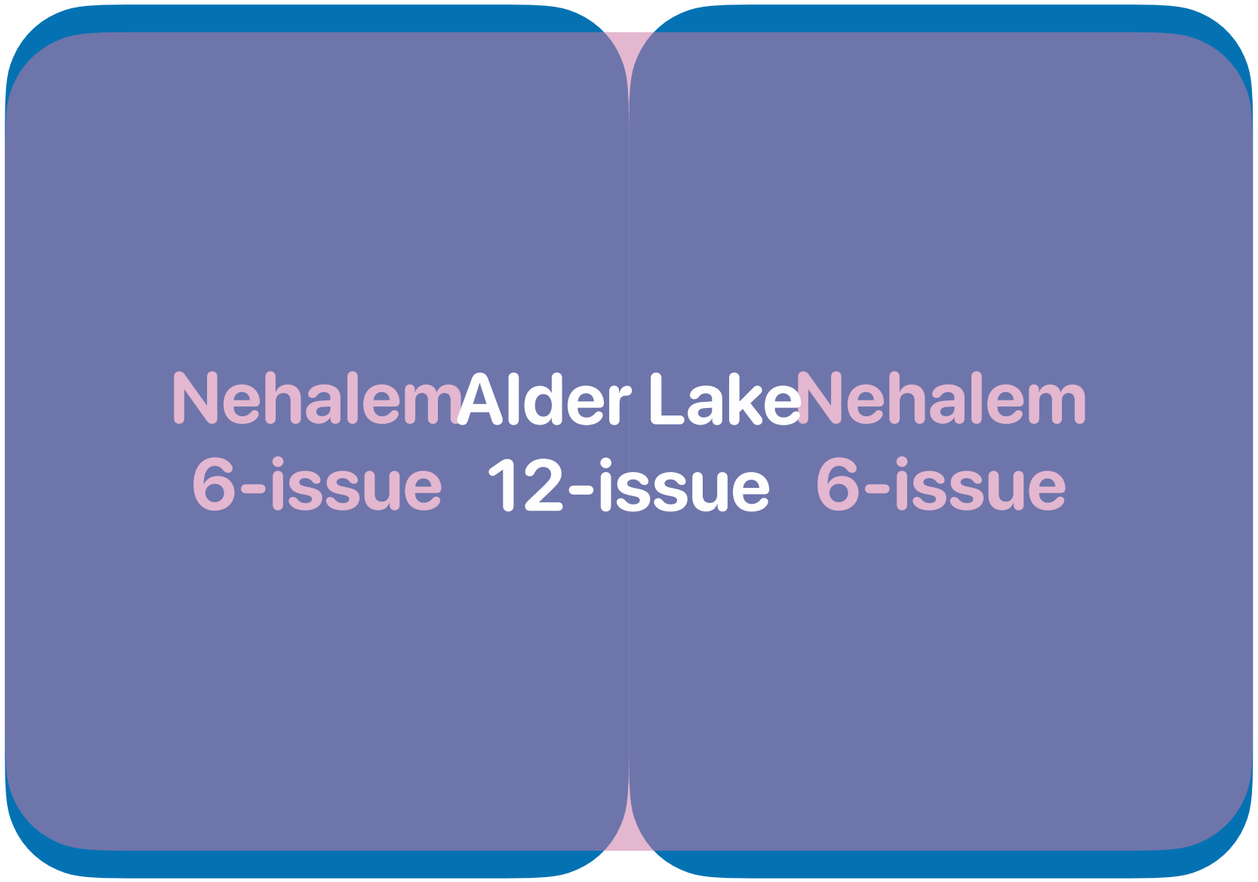
How many transistors per core on Coffee Lake?

The Coffee Lake processor has 217 million transistors per core. It is manufactured using Intel's second 14 nm pro
Lake processors introduced i5 and i7 CPUs featuring six cores (along with hyper-threading in the case of the lat
and no hyperthreading.

The transistor count per core on Coffee Lake is lower than that of some other modern processors, such as the I
transistors per core. However, Coffee Lake still offers good performance, thanks to its high clock speeds and ef

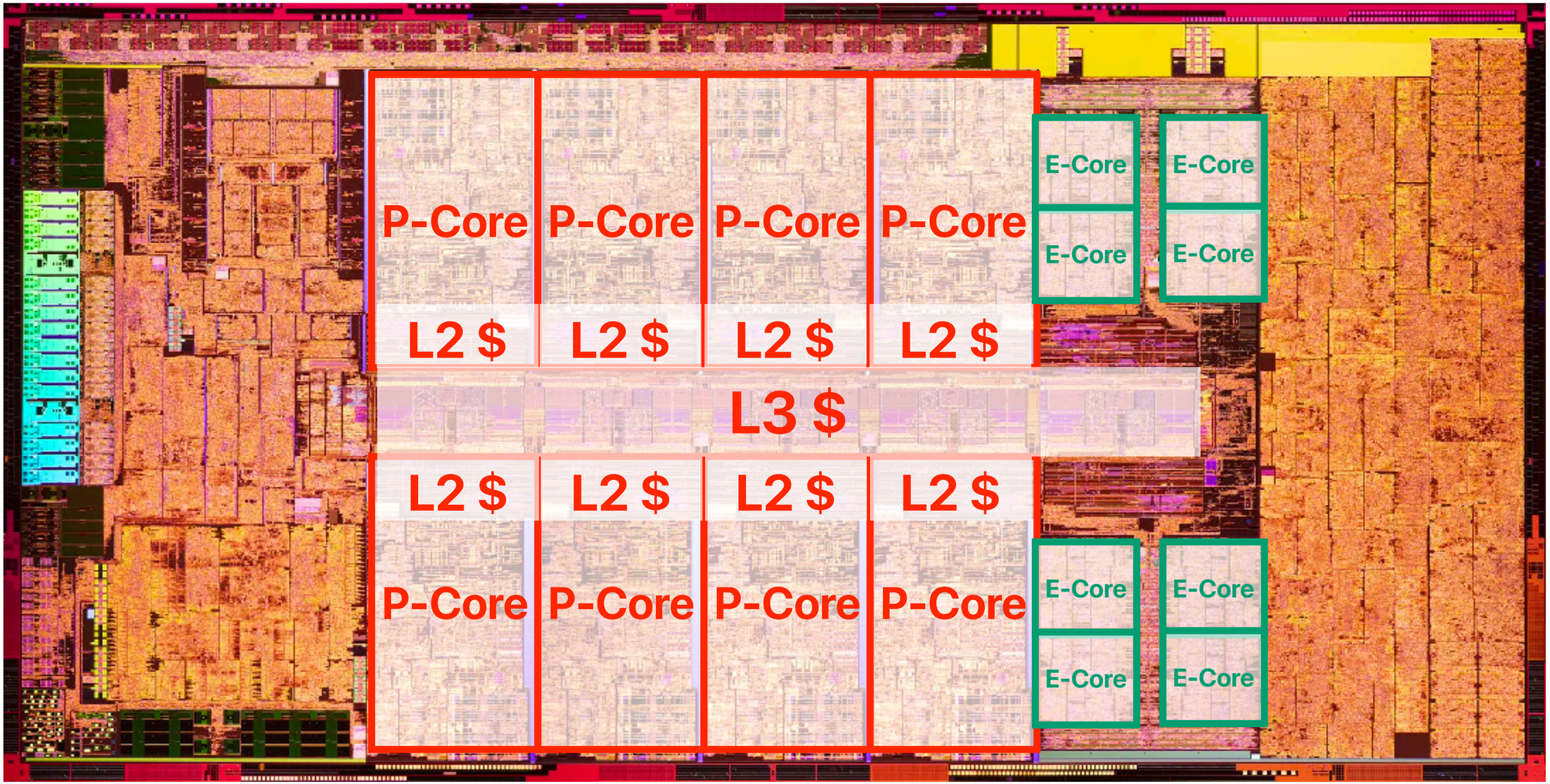
Here is a table of the transistor counts per core for some other modern processors:

Processor	Transistors per core
Coffee Lake	217 million
Ryzen 5 5600X	390 million

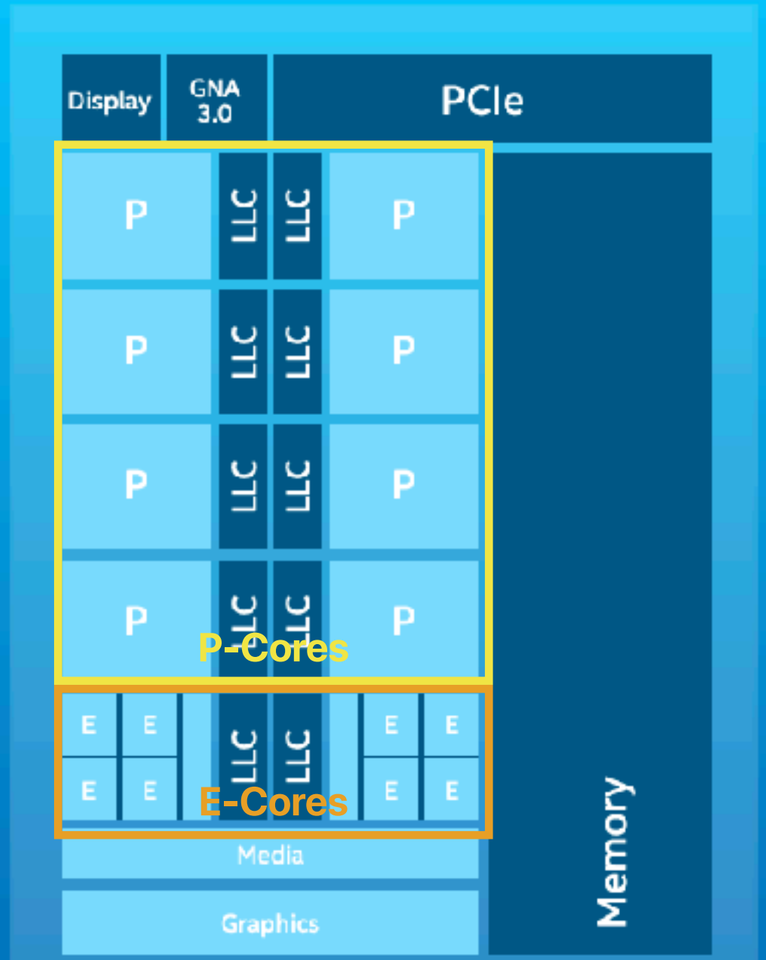


Based on https://en.wikipedia.org/wiki/Transistor_count

Intel Alder Lake



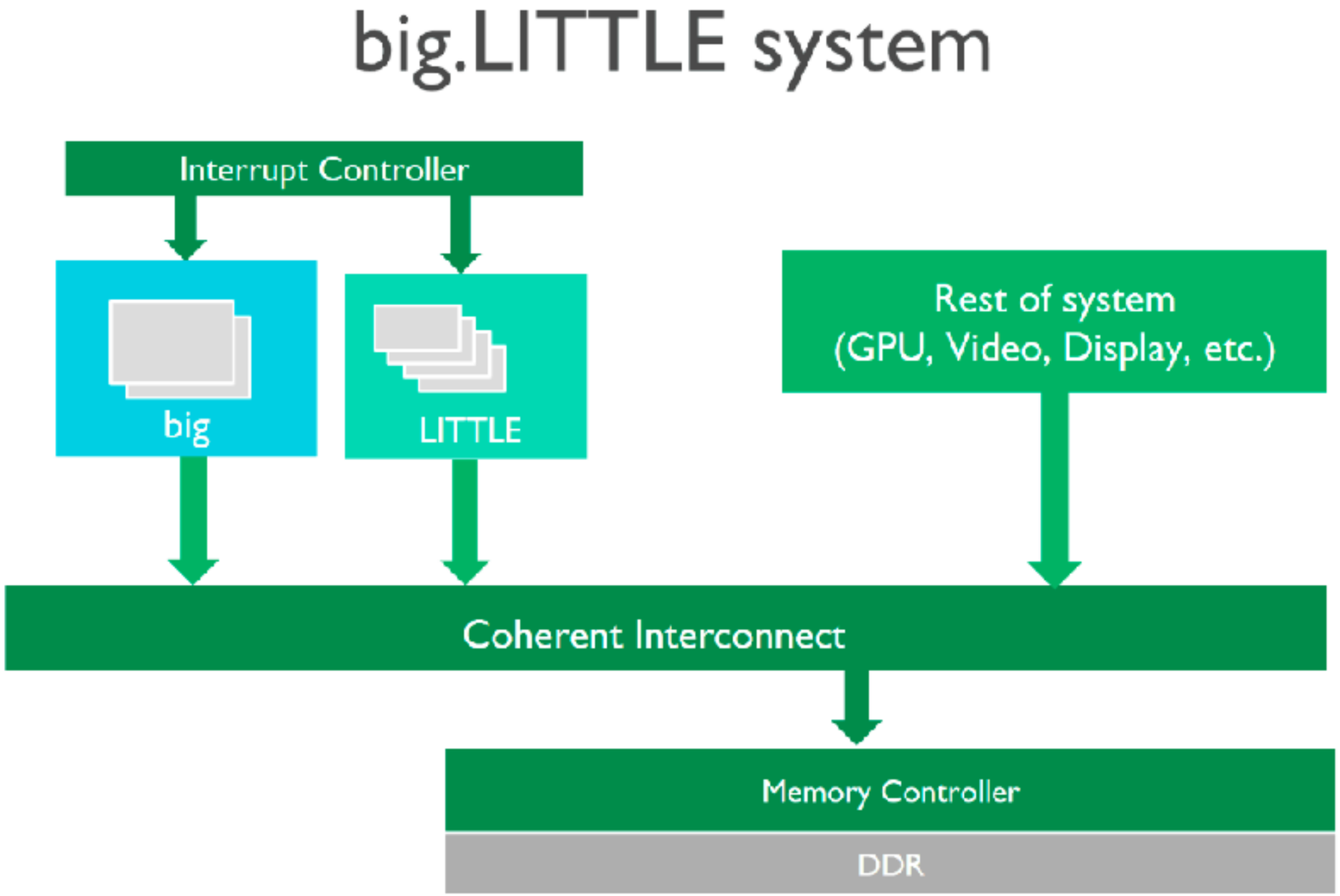
Single ISA heterogeneous CMP in Intel Processors



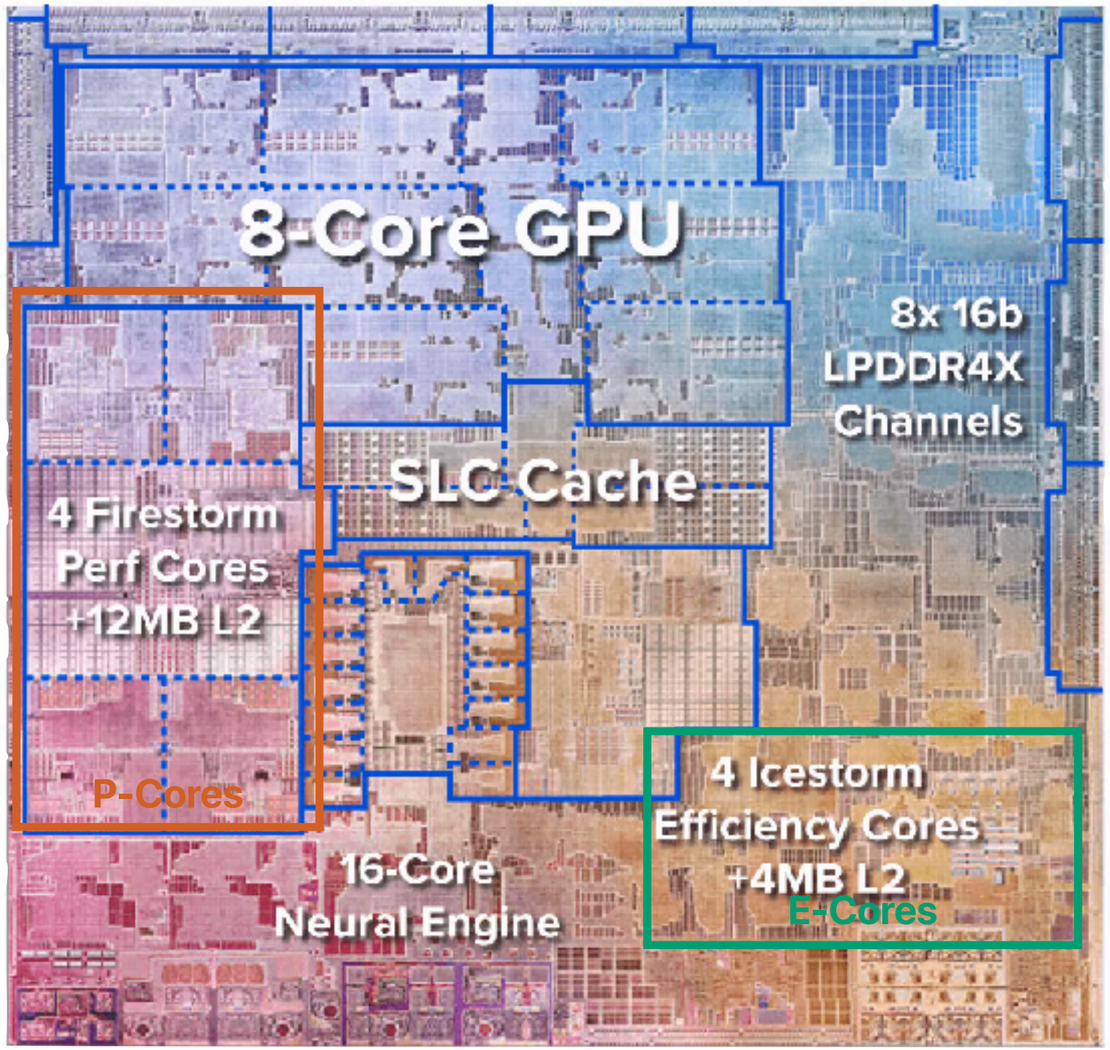
In a game-changing advance for core performance, 12th Gen Intel® Core™ desktop processors power a revolutionary approach to the x86 architecture. Its Performance-cores—or "P-cores"—are optimized for single & lightly-threaded performance, while its Efficient-cores—or "E-cores"—are optimized for scaling 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 ARM's big.LITTLE architecture



Single ISA heterogeneous CMP in Apple's M1



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

Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction

Rakesh Kumar, Keith I. Farkas*, Norman P. Jouppi*, Parthasarathy Ranganathan*, Dean M. Tullsen

UCSD and HP Labs*

In the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36., 2003
MICRO Test-of-time award, 2021

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

Single ISA heterogeneous CMP (big.Little)

- Regarding "Single-ISA Heterogeneous Multi-Core Architectures", how many of the following statements is/are correct?
  - ① You need to recompile and optimize the binary for each core architecture to exploit the thread-level parallelism in this architecture
  - ② For a program with limited thread-level parallelism, single ISA heterogeneous CMP would deliver better than or at least the same level of performance as homogeneous CMP built with older-generation cores
  - ③ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with older-generation cores
  - ④ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with newer-generation cores

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

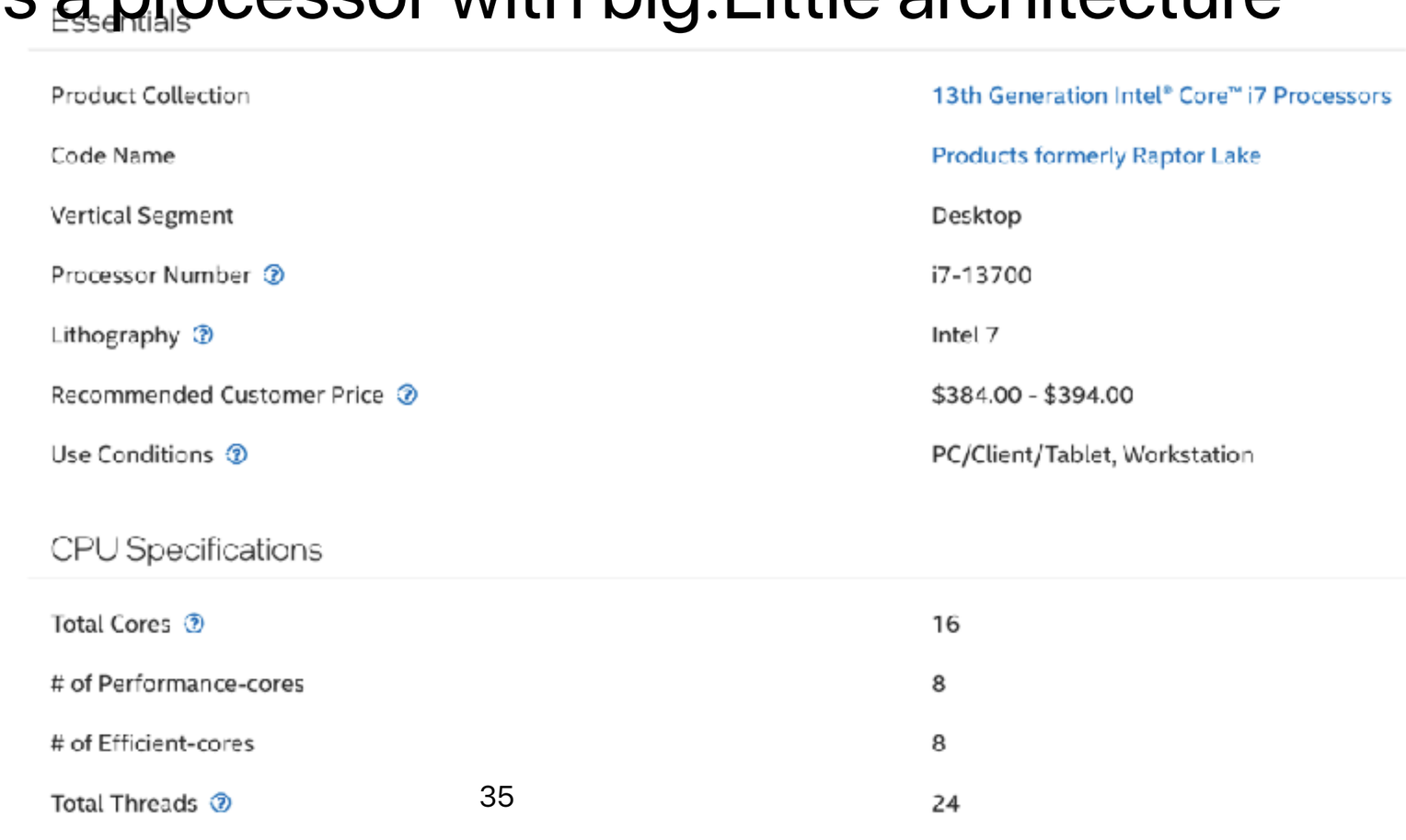
Single ISA heterogeneous CMP

- Regarding “Single-ISA Heterogeneous Multi-Core Architectures”, how many of the following statements is/are correct?
  - ① You need to recompile and optimize the binary for each core architecture to exploit the thread-level parallelism in this architecture
  - ② For a program with limited thread-level parallelism, single ISA heterogeneous CMP would deliver better than or at least the same level of performance as homogeneous CMP built with older-generation cores
  - ③ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with older-generation cores
  - ④ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with newer-generation cores

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

Demo

- We can use taskset command in linux to control the task allocation.
- Core i7 13700 is a processor with big.Little architecture



Essentials

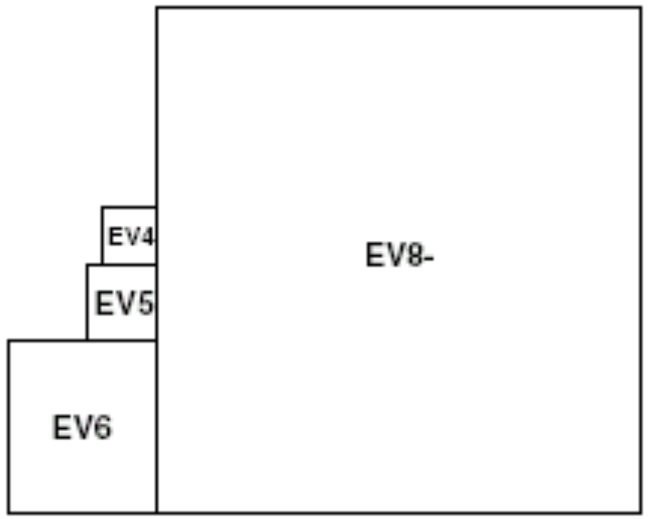
Product Collection	13th Generation Intel® Core™ i7 Processors
Code Name	Products formerly Raptor Lake
Vertical Segment	Desktop
Processor Number	i7-13700
Lithography	Intel 7
Recommended Customer Price	$384.00 - $394.00
Use Conditions	PC/Client/Tablet, Workstation

CPU Specifications

Total Cores	16
# of Performance-cores	8
# of Efficient-cores	8
Total Threads	24

What the paper says?

Processor	EV5	EV6	EV6+
Issue-width	4	6 (OOO)	6 (OOO)
I-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
D-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
Branch Pred.	2K-gshare	hybrid 2-level	hybrid 2-level
Number of MSHRs	4	8	16
Number of threads	1	1	4
Area (in mm^2)	5.06	24.5	29.9



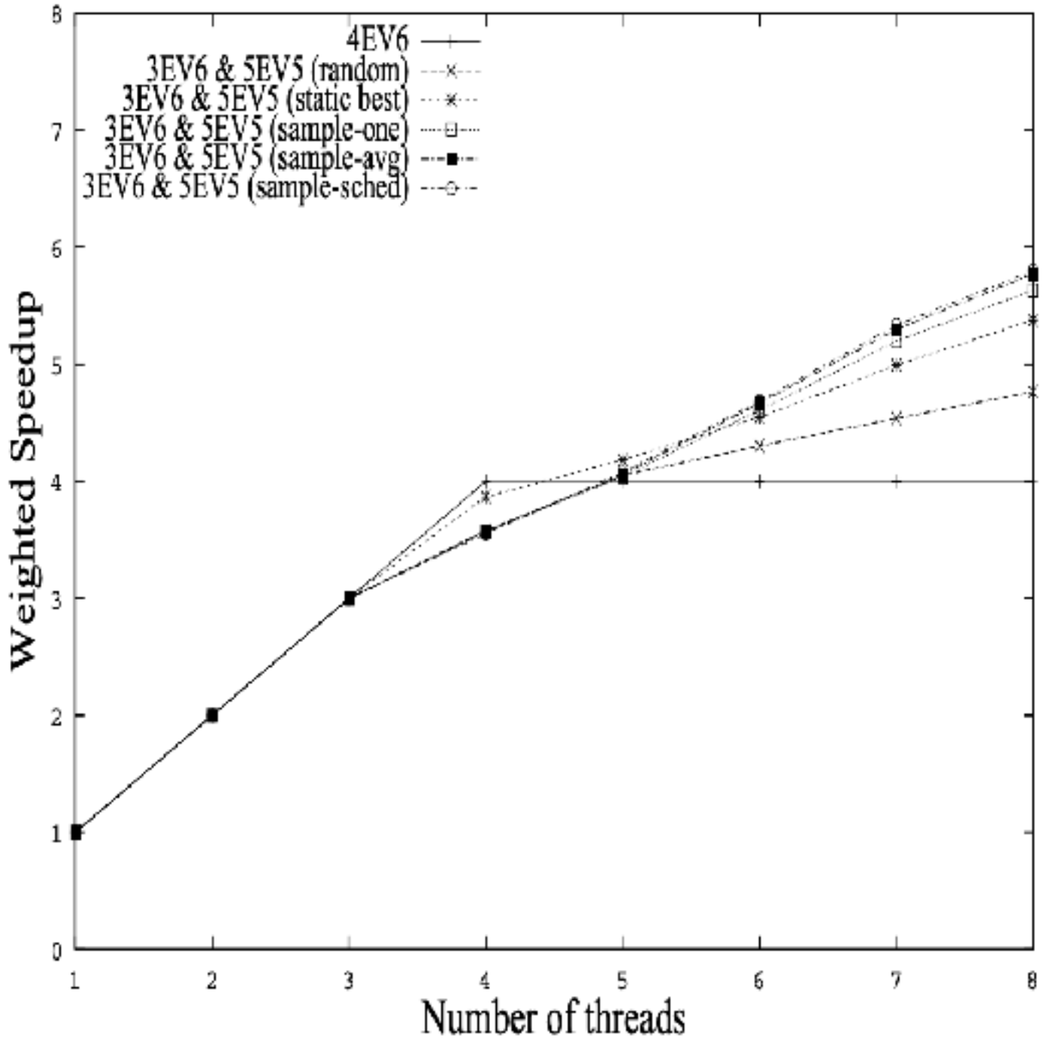
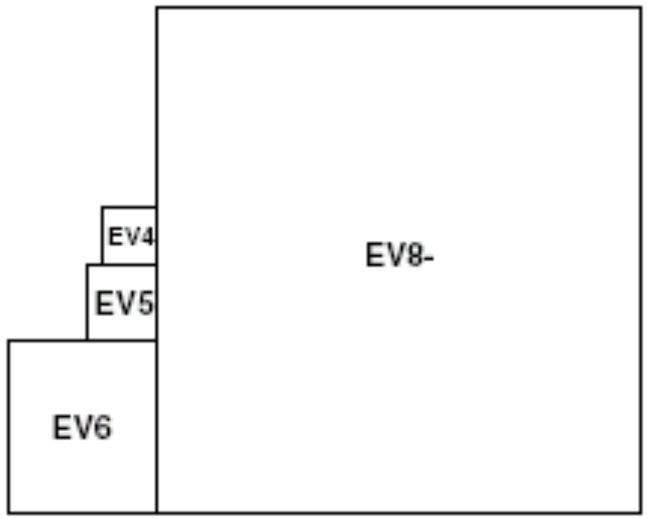


Figure 4. *Three strategies for evaluating the performance an application will realize on a different core.*

What the paper says?

Processor	EV5	EV6	EV6+
Issue-width	4	6 (OOO)	6 (OOO)
I-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
D-Cache	8KB, DM	64KB, 2-way	64KB, 2-way
Branch Pred.	2K-gshare	hybrid 2-level	hybrid 2-level
Number of MSHRs	4	8	16
Number of threads	1	1	4
Area (in mm^2)	5.06	24.5	29.9



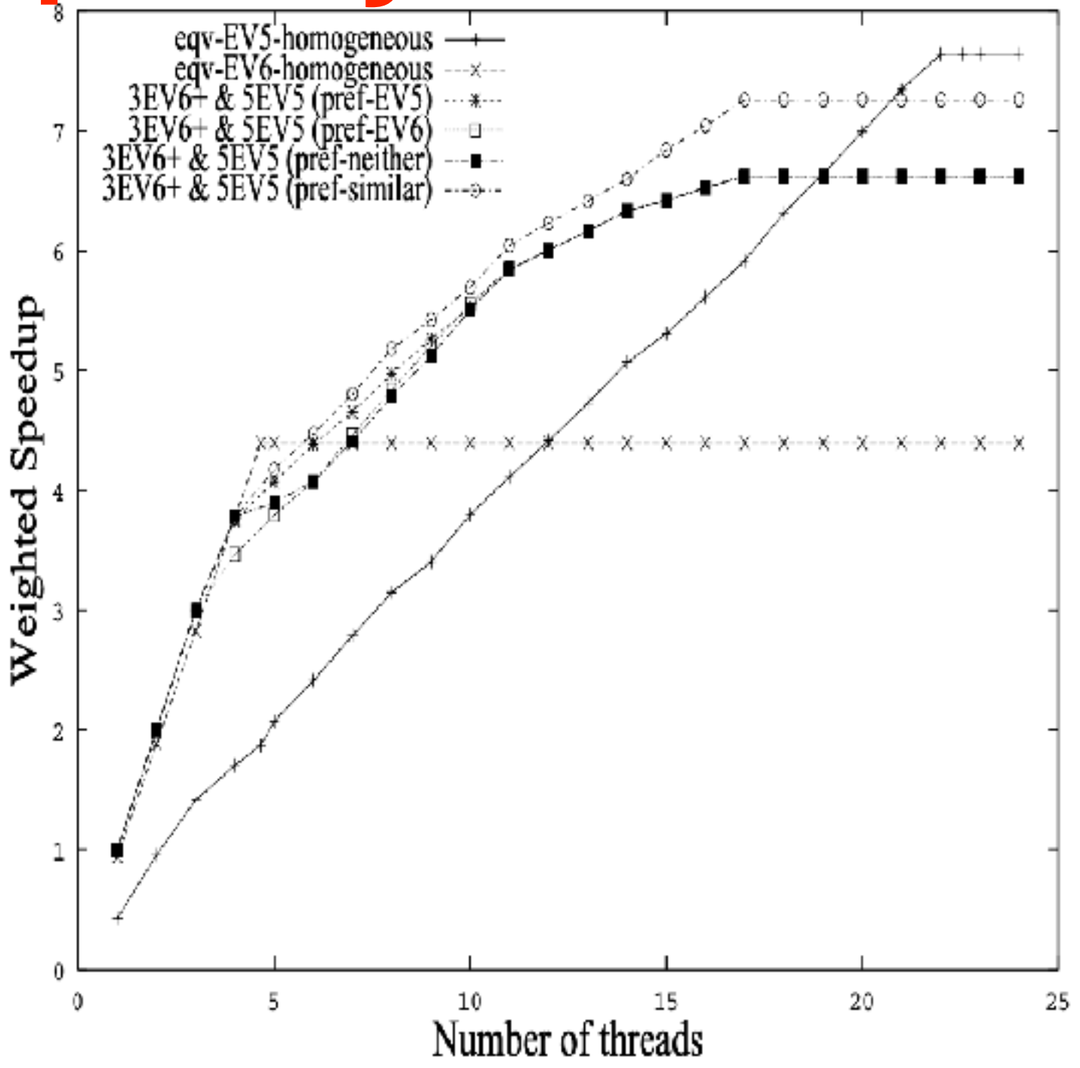


Figure 7. *Performance of heuristics for a heterogeneous architecture with multithreaded cores.*

Single ISA heterogeneous CMP

- Regarding "Single-ISA Heterogeneous Multi-Core Architectures", how many of the following statements is/are correct?
  - ① You need to recompile and optimize the binary for each core architecture to exploit the thread-level parallelism in this architecture
  - ② For a program with limited thread-level parallelism, single ISA heterogeneous CMP would deliver better than or at least the same level of performance as homogeneous CMP built with older-generation cores
  - ③ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with older-generation cores
  - ④ For a program with rich thread-level parallelism, single ISA heterogeneous CMP would deliver better or at least the same level of performance than homogeneous CMP built with newer-generation cores

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

- Corollary #4: Exploiting more parallelism from a program is the key to performance gain in modern architectures

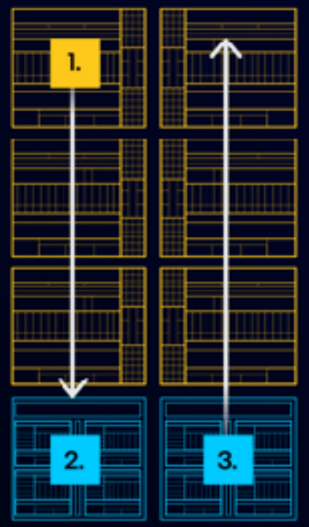
$$Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1 - f_{parallelizable})}$$

Intel Thread Director

Scheduling Evolution

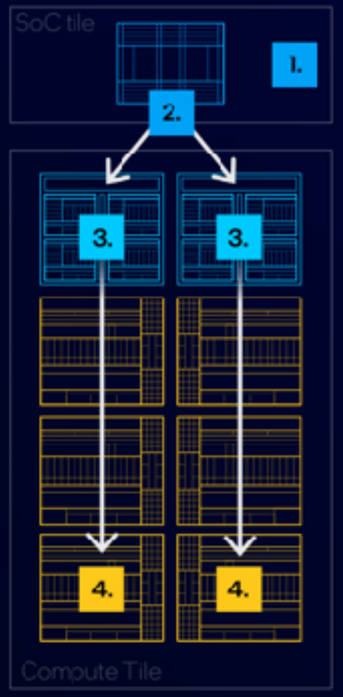
Raptor Lake

1. Higher demand work to **P-cores**
2. Lower demand to **E-cores**
3. Periodically move **E-core** threads to **P-core**



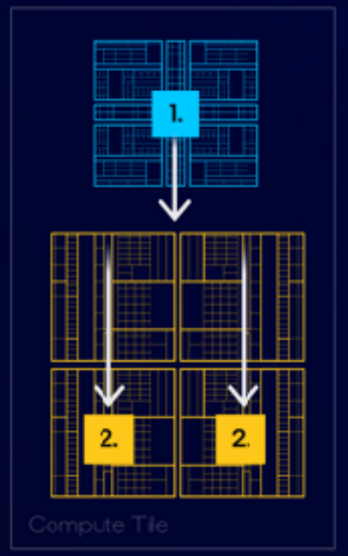
Meteor Lake

1. Contain work on **SoC E-cores**
2. Exceeding capacity? move to **compute tile**
3. Use compute **E-cores** if the work fits there
4. Higher demand work to **P-cores**



Lunar Lake

1. Use **E-cores** if the work fits there
2. Exceeding capacity? Move work to **P-cores**



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

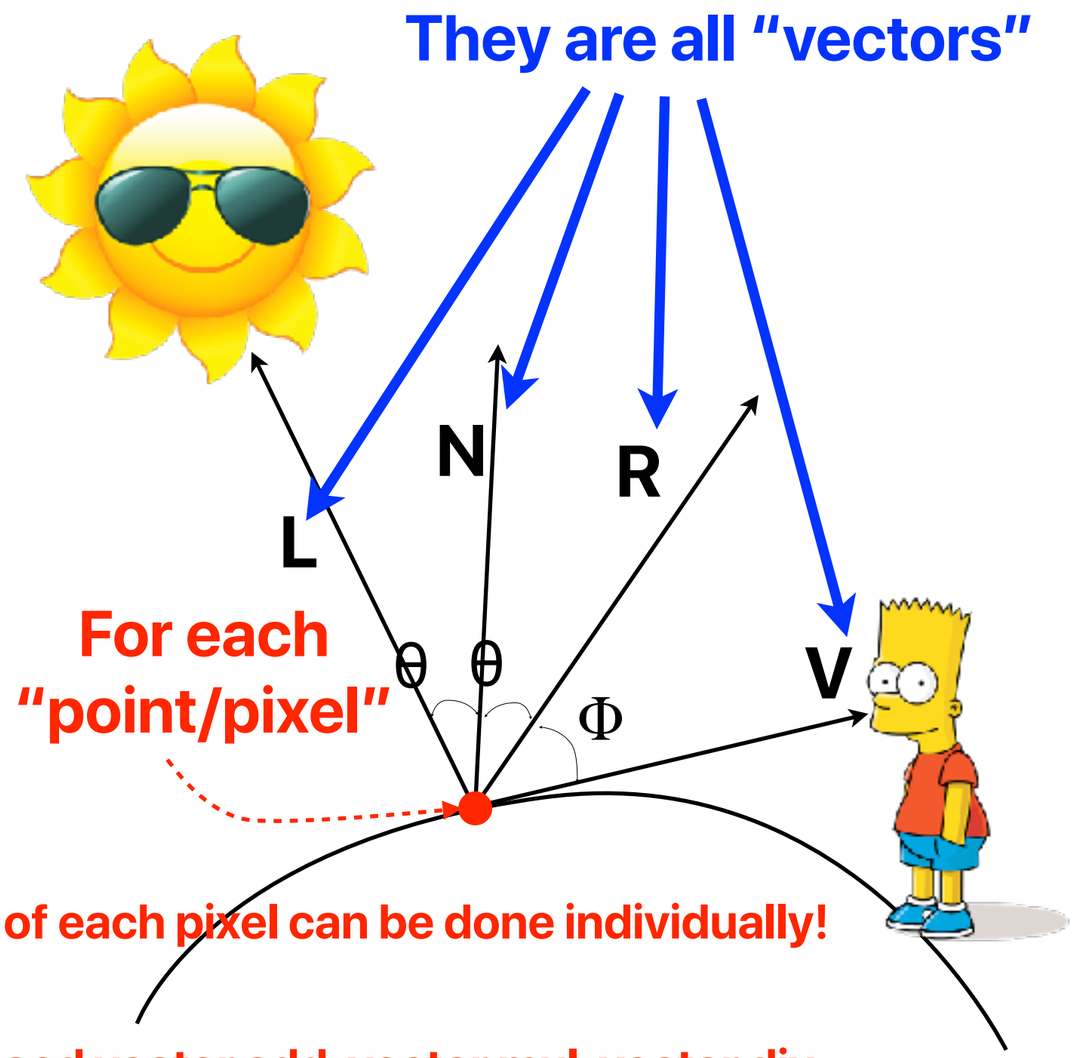
Essentials

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

Basic concept of shading



$$I_{amb} = K_{amb} \cdot M_{amb}$$

$$I_{diff} = K_{diff} \cdot M_{diff} \cdot (N \cdot L)$$

$$I_{spec} = K_{spec} \cdot M_{spec} \cdot (R \cdot V)^n$$

$$I_{total} = I_{amb} + I_{diff} + I_{spec}$$

```
void main(void)
{
    // normalize vectors after interpolation
    vec3 L = normalize(o_toLight);
    vec3 V = normalize(o_toCamera);
    vec3 N = normalize(o_normal);

    // get Blinn-Phong reflectance components
    float Iamb = ambientLighting();
    float Idif = diffuseLighting(N, L);
    float Ispe = specularLighting(N, L, V);

    // diffuse color of the object from texture
    vec3 diffuseColor = texture(u_diffuseTexture, o_texcoords)

    // combination of all components and diffuse color of the
    resultingColor.xyz = diffuseColor * (Iamb + Idif + Ispe);
    resultingColor.a = 1;
}
```

The work of each pixel can be done individually!

We only need vector add, vector mul, vector div.

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

What GPU architectures should look like?

- Based on the original target of GPU design, what do you think would be reasonable design decisions that GPU architectures make?
  - ① The architecture should render pixels as fast as possible
  - ② The architecture does not need a branch unit
  - ③ The architecture should contain an array of powerful ALUs and functional units that can perform a rich set of operations
  - ④ The architecture should offer very large bandwidth of memory accesses
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

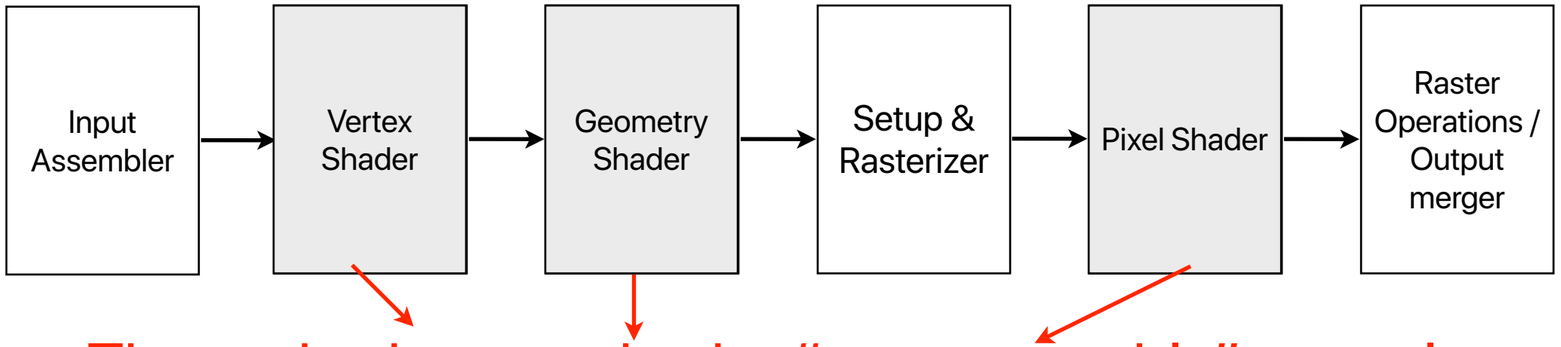
What GPU architectures should look like?

- Based on the original target of GPU design, what do you think would be reasonable design decisions that GPU architectures make?
  - ① The architecture should render pixels as fast as possible
  - ② The architecture does not need a branch unit
  - ③ The architecture should contain an array of powerful ALUs and functional units that can perform a rich set of operations
  - ④ The architecture should offer very large bandwidth of memory accesses
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

GPU (Graphics Processing Unit)

- Originally for displaying images
- HD video: 1920*1080 pixels * 60 frames per second
  - Therefore, GPU is not latency-oriented by design!
  - Even for 120 frames, you still have 8ms latency to get everything done!
- Graphics processing pipeline

1 GHz can give you 8000000 cycles!!!



These shaders need to be "programmable" to apply different rendering effects/algorithms (Phong shading, Gouraud shading, and etc...)

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

What GPU architectures should look like?

- Based on the original target of GPU design, what do you think would be reasonable design decisions that GPU architectures make?
  - ① The architecture should render pixels as fast as possible
  - ② The architecture does not need a branch unit
  - ③ The architecture should contain an array of powerful ALUs and functional units that can perform a rich set of operations
  - ④ The architecture should offer very large bandwidth of memory accesses

  A. 0

  B. 1

  C. 2

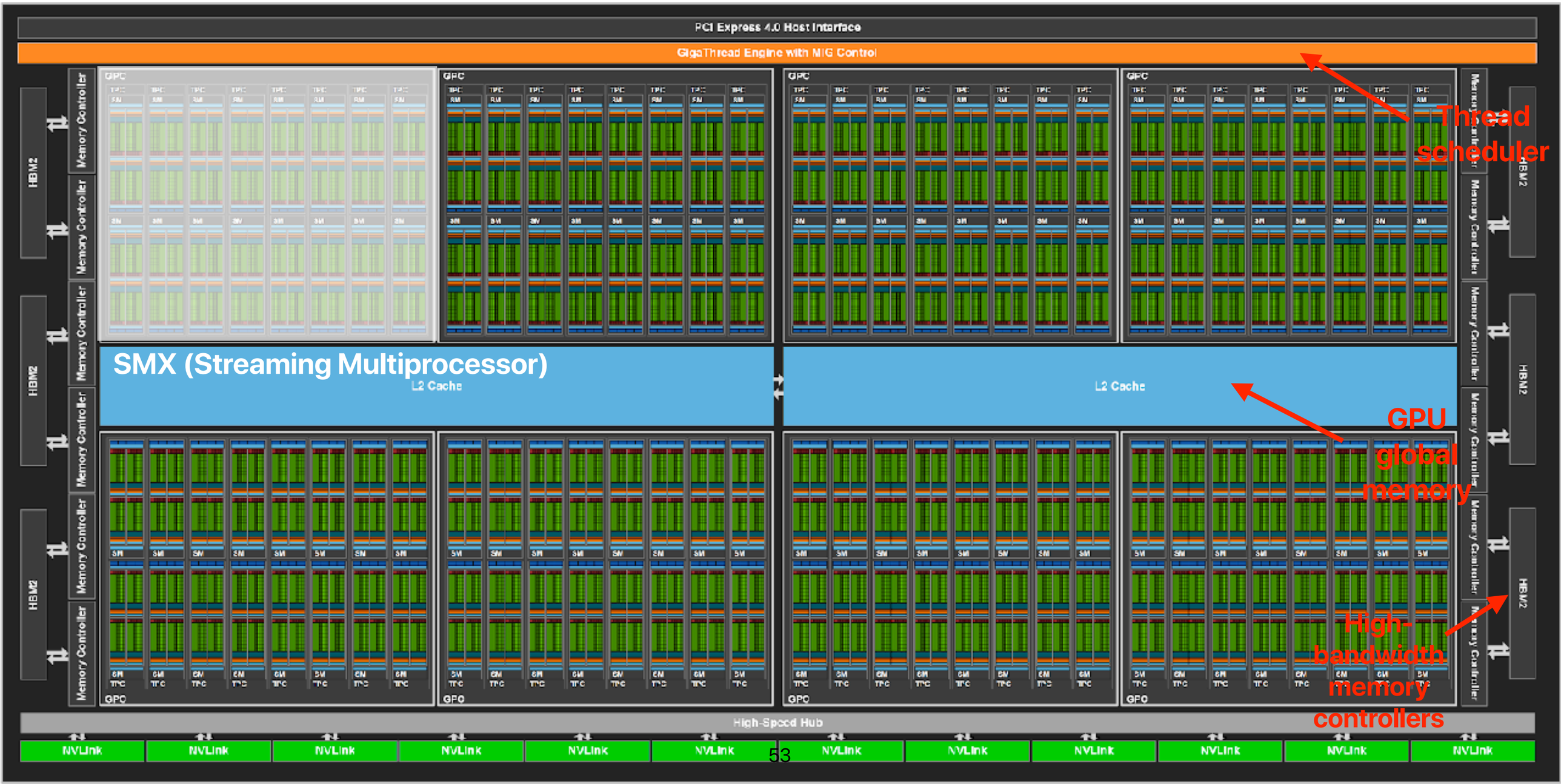
  D. 3

  E. 4

GPU is also the concept of “slower, but more”

https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

GPU Architecture



Inside an SM

A total of 16*4 = 64 FP32 cores
A total of 16*4 = 64 INT32 cores
A total of 16*4 = 16 FP64 cores

- All of these can only perform the same operation at the same time, but each of these is named as a “thread” in CUDA
- You can only use either FP32, FP64, INT32 and “Tensor Cores” at the same time

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

How about static power?

Moore's Law allows higher frequencies as transistors are smaller

Moore's Law makes this smaller

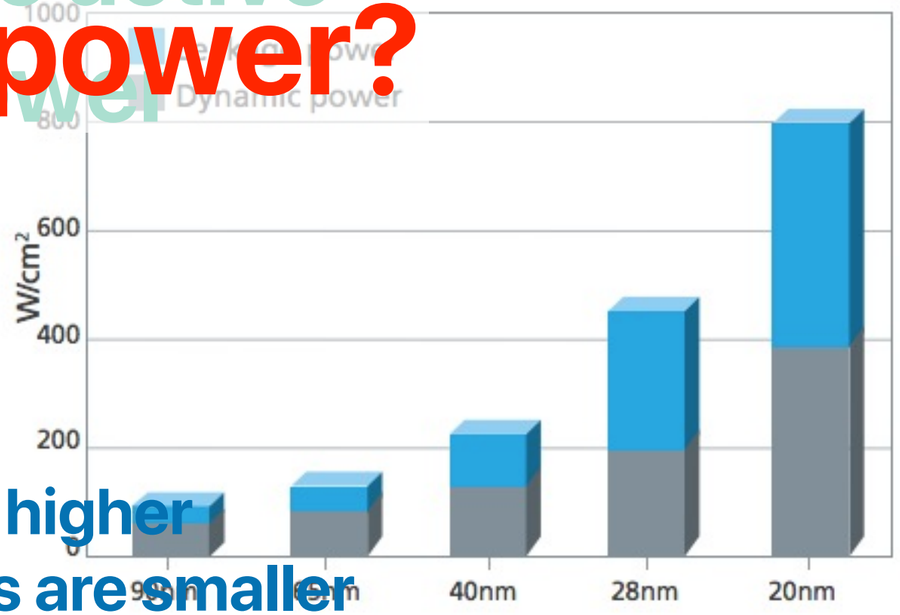


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Trends in the Dark Silicon Era

- Aggressive dynamic voltage/frequency scaling
  - Potentially hurts performance and increases **energy consumption**
  - Trade-offs between UX/Power/Energy
- Throughout oriented — slower, but more
  - Hurts latency
  - Limited applications — not every workload reveals strong data-level parallelism
- Just let it dark — activate part of circuits, but not all
- From general-purpose to domain-specific — ASIC

Just let it dark

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

Inside an SM

A total of 16*4 = 64 FP32 cores
A total of 16*4 = 64 INT32 cores
A total of 16*4 = 16 FP64 cores

- All of these can only perform the same operation at the same time, but each of these is named as a “thread” in CUDA
- You can only use either FP32, FP64, INT32 or “Tensor Cores” at the same time

Trends in the Dark Silicon Era

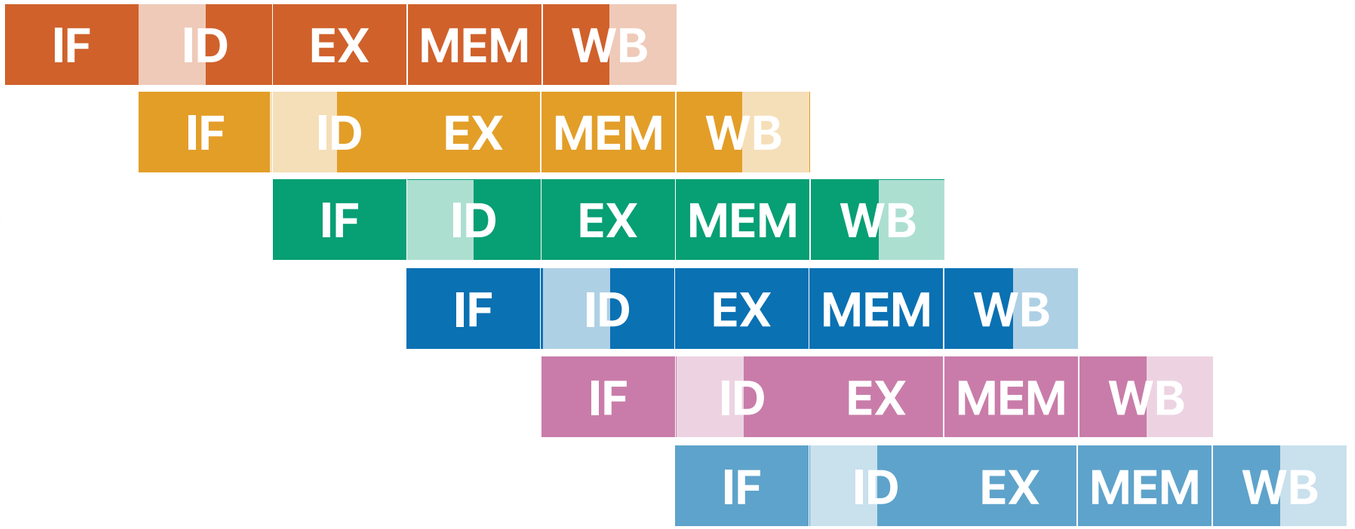
- Aggressive dynamic voltage/frequency scaling
  - Potentially hurts performance and increases **energy consumption**
  - Trade-offs between UX/Power/Energy
- Throughout oriented — slower, but more
  - Hurts latency
  - Limited applications — not every workload reveals strong data-level parallelism
- Just let it dark — activate part of circuits, but not all
  - Increases manufacturing costs
- From general-purpose to domain-specific — ASIC

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

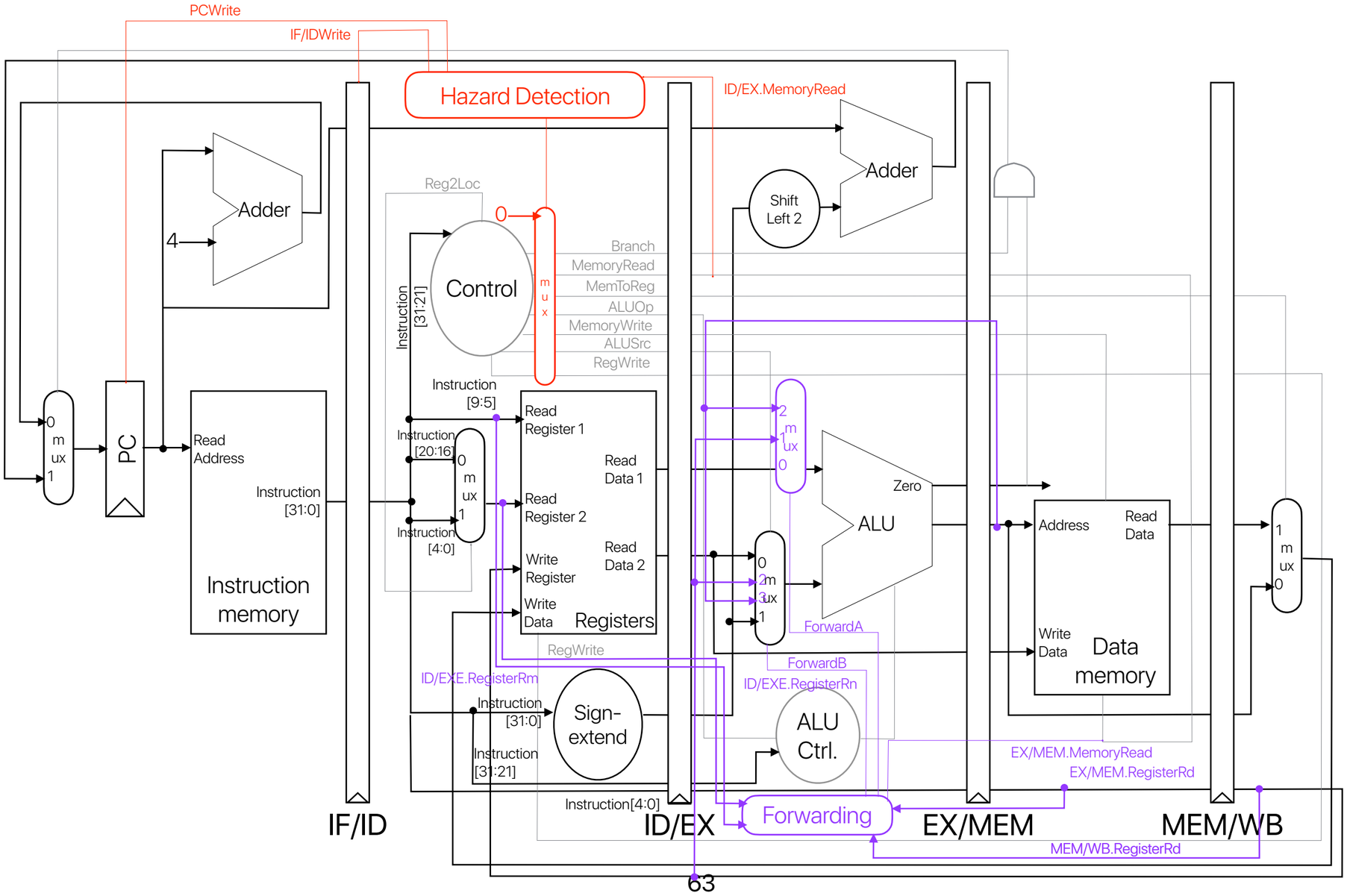
Say, we want to implement a[i] += a[i+1]*20

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

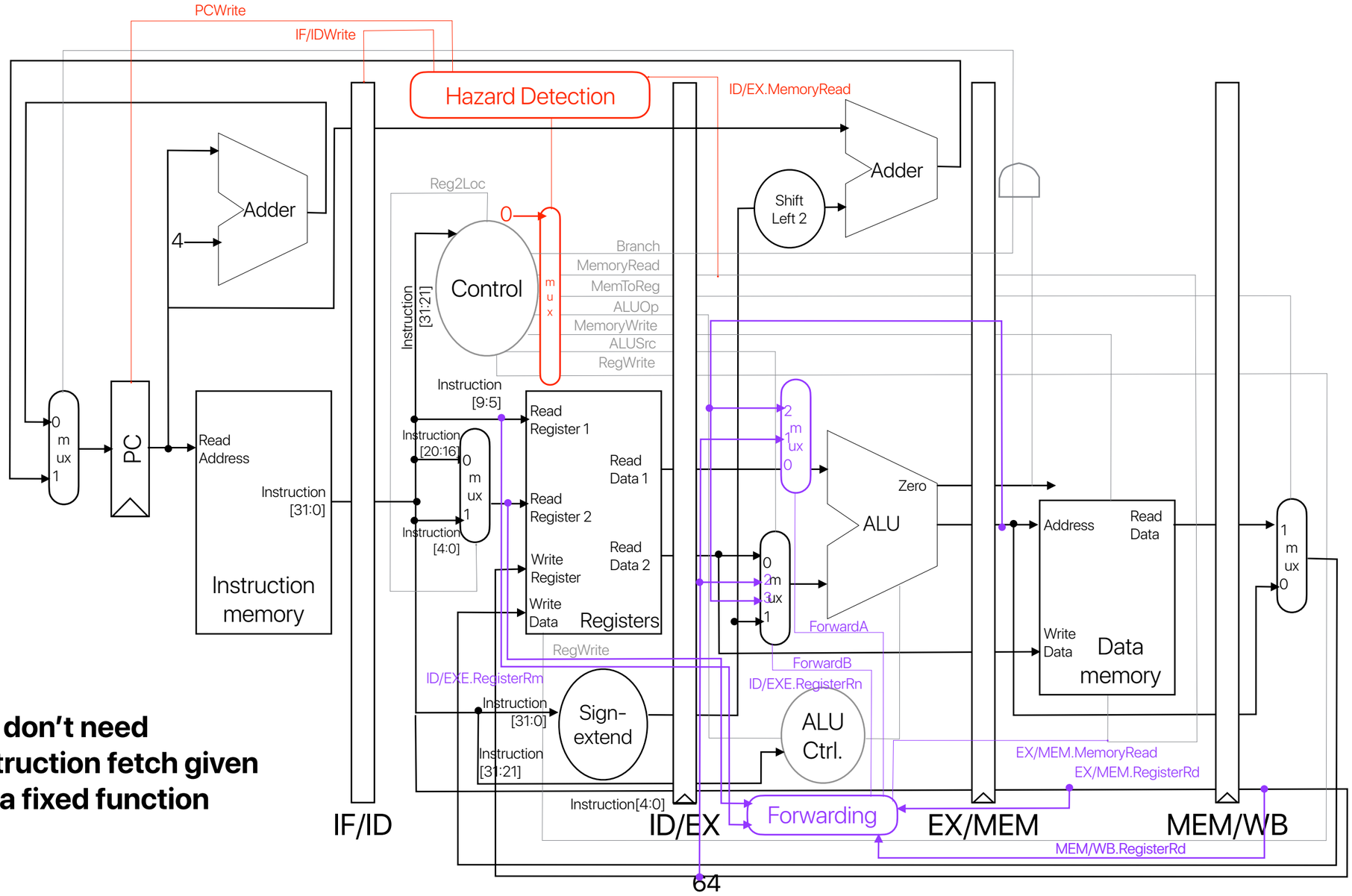
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



This is what you need for these instructions



Specialize the circuit

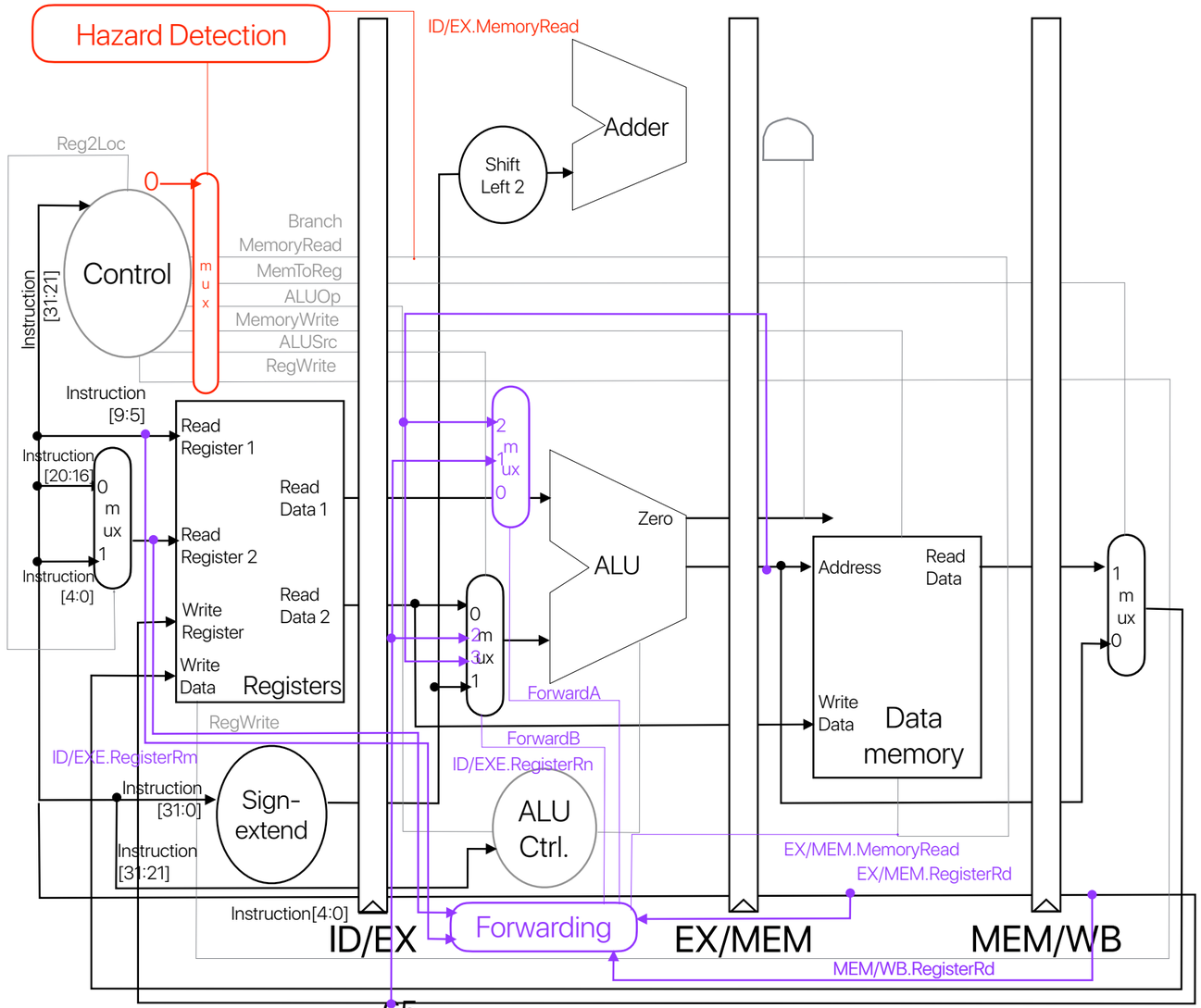


We don't need instruction fetch given it's a fixed function

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

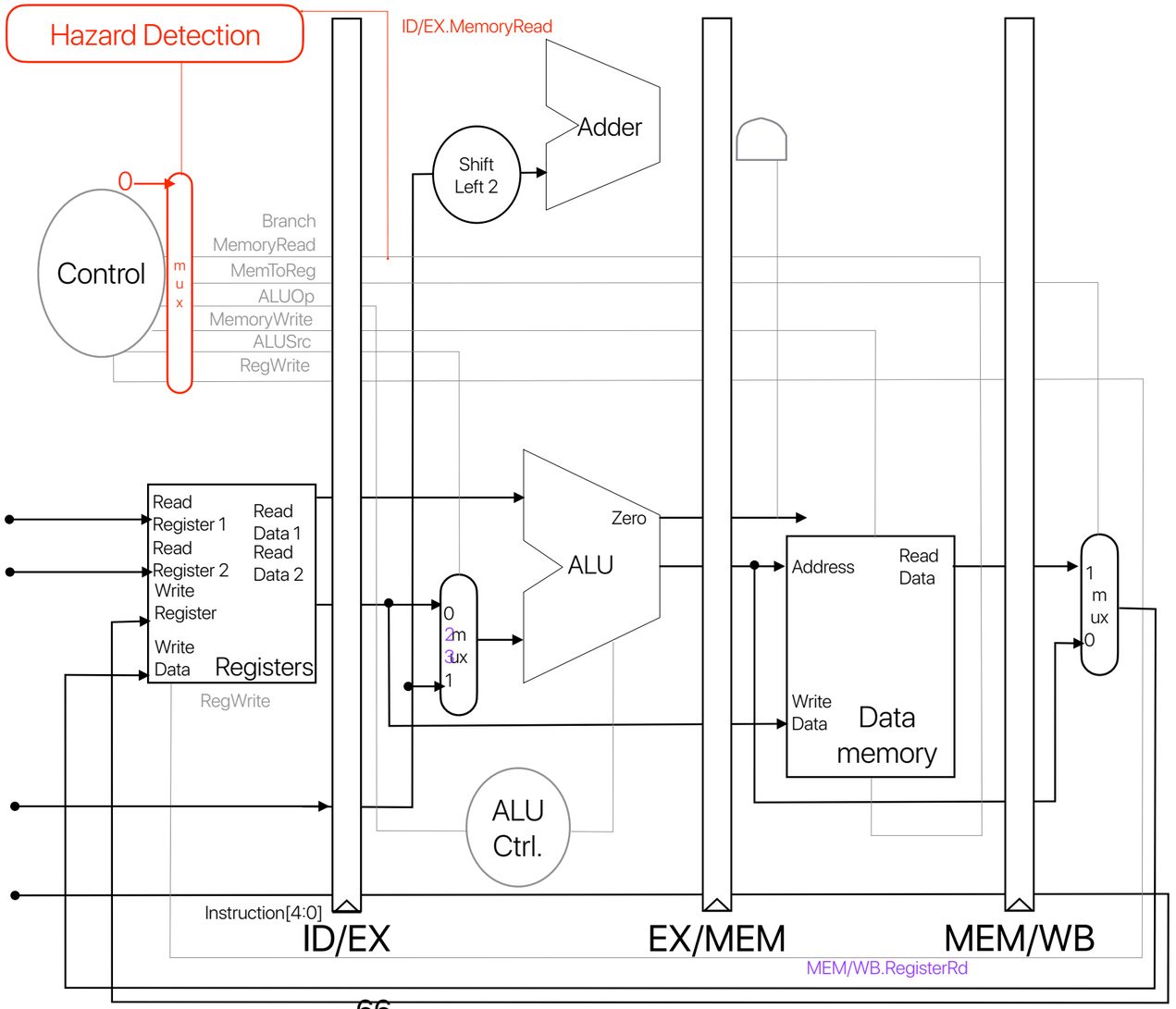


Specialize the circuit

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

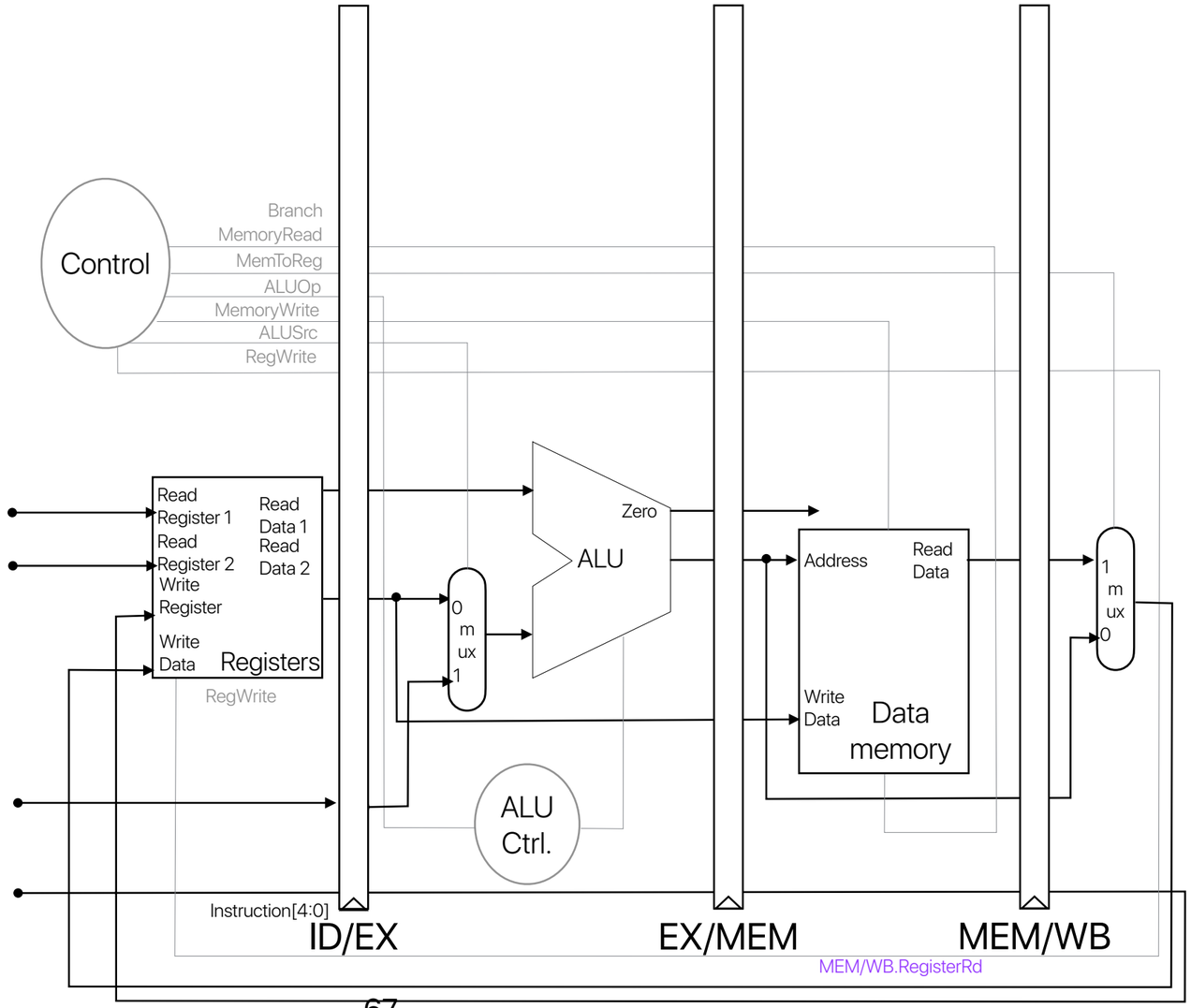


Specialize the circuit

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

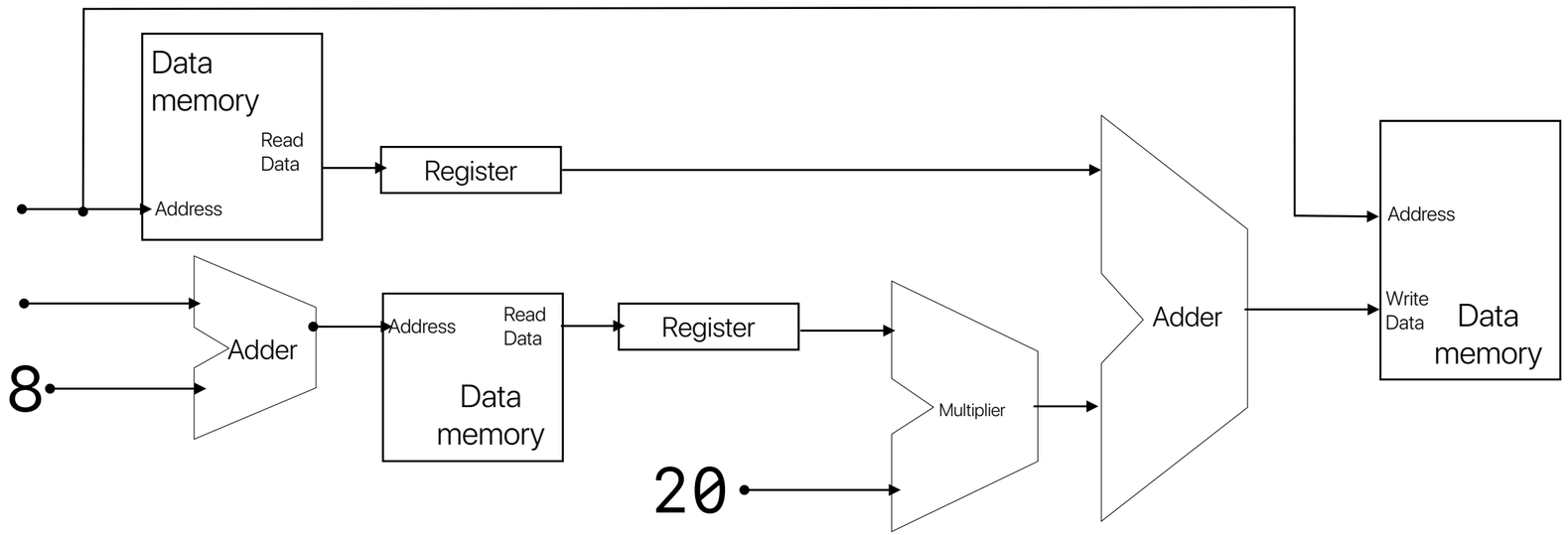
We don’t need these many registers, complex control, decode

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



Rearranging the datapath

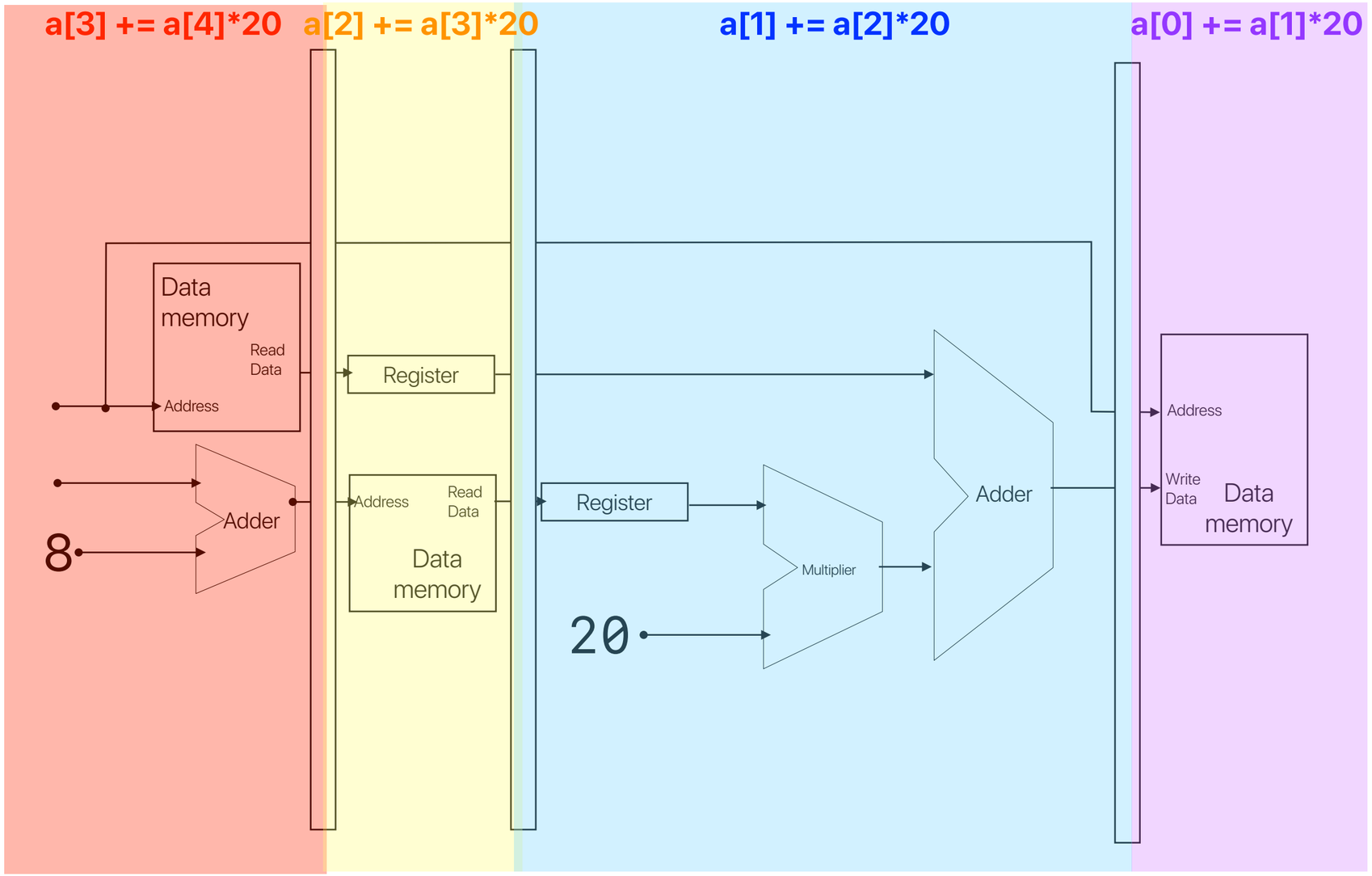
```
ld    X1, 0(X0)
ld    X2, 8(X0)
add   X3, X31, #20
mul   X2, X2, X3
add   X1, X1, X2
sd    X1, 0(X0)
```



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



In-Datacenter Performance Analysis of a Tensor Processing Unit

N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Na- garajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Wal- ter, W. Wang, E. Wilcox, and D. H. Yoon
Google Inc.

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

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.

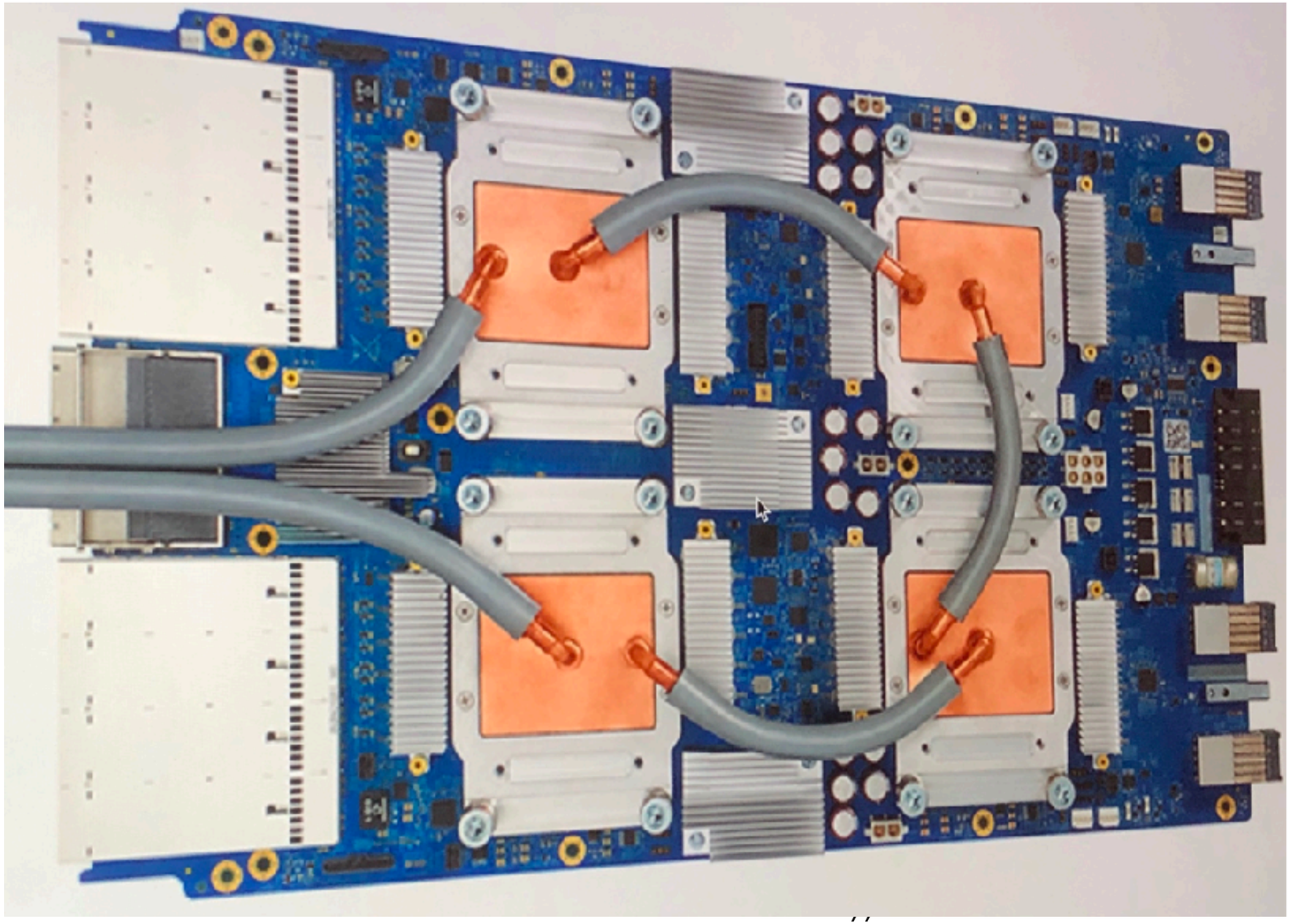
peak performance.

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

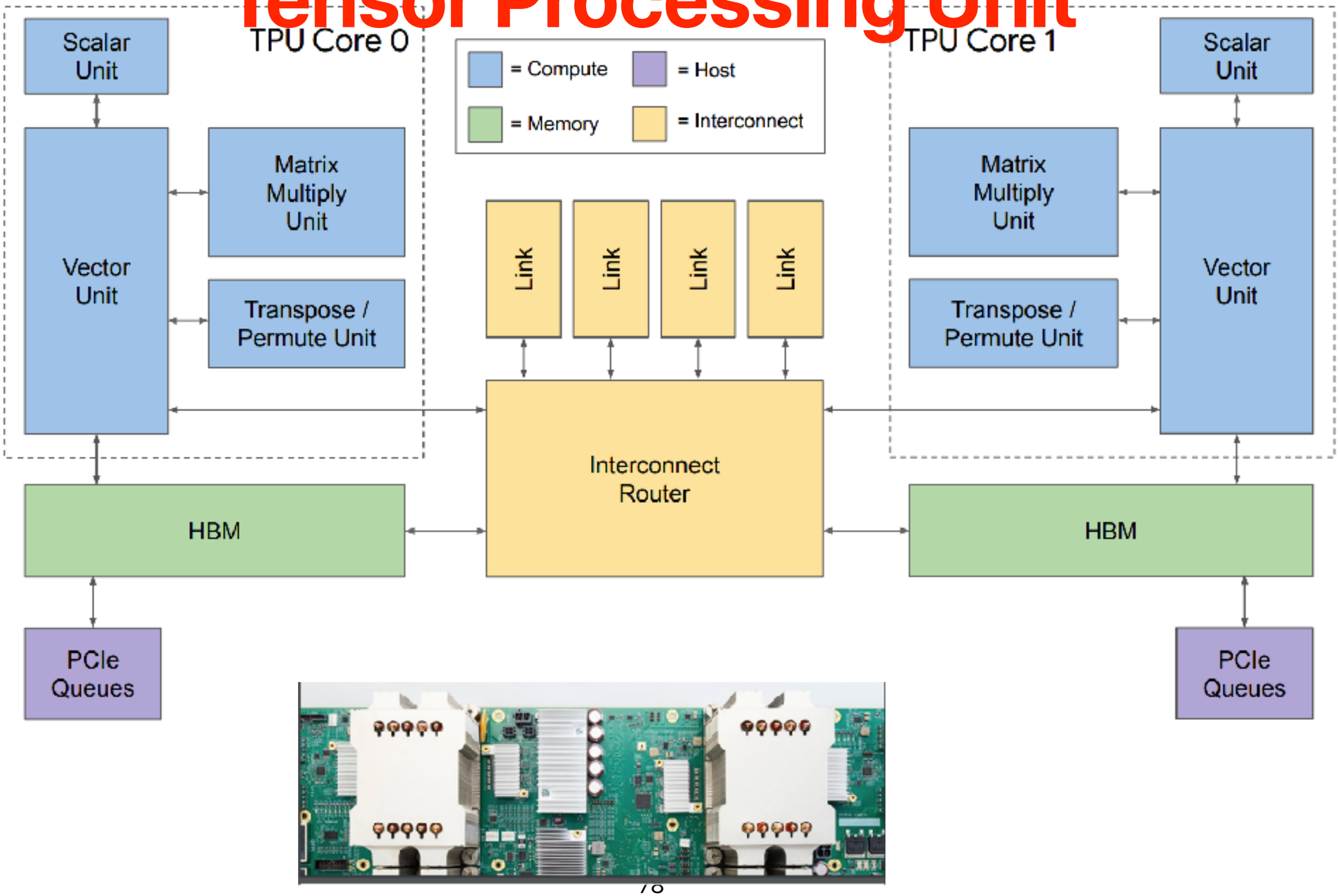
takes to shift a tile in). This unit is designed for dense matrices. Sparse architectural support was omitted for time-to-deployment reasons. The weights for the matrix unit are staged through an on-

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

What TPU looks like

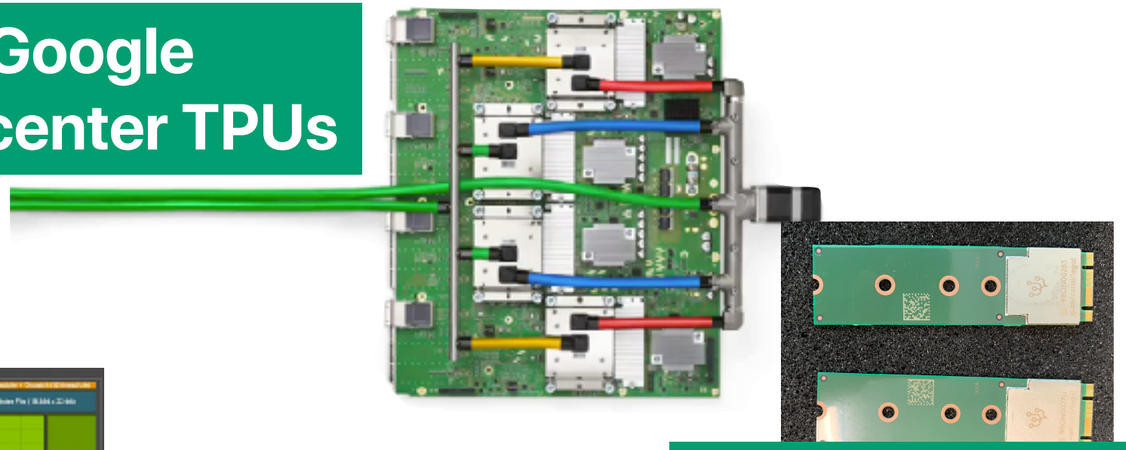


Tensor Processing Unit



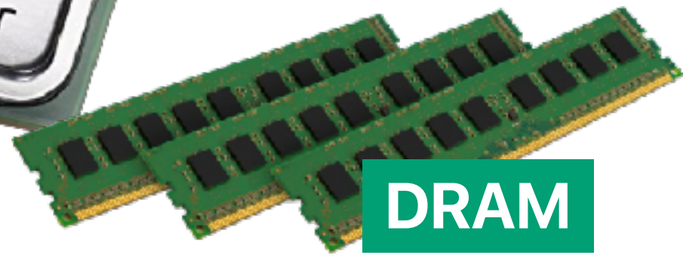
The “landscape” of modern computers

Google Datacenter TPUs



Google Edge TPUs

CPU

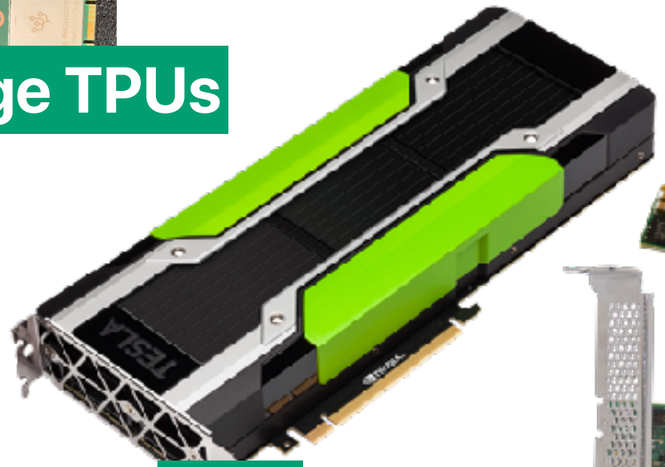


DRAM

AI/ML Accelerators

NVIDIA Tensor Core Units

Apple Neural Engines

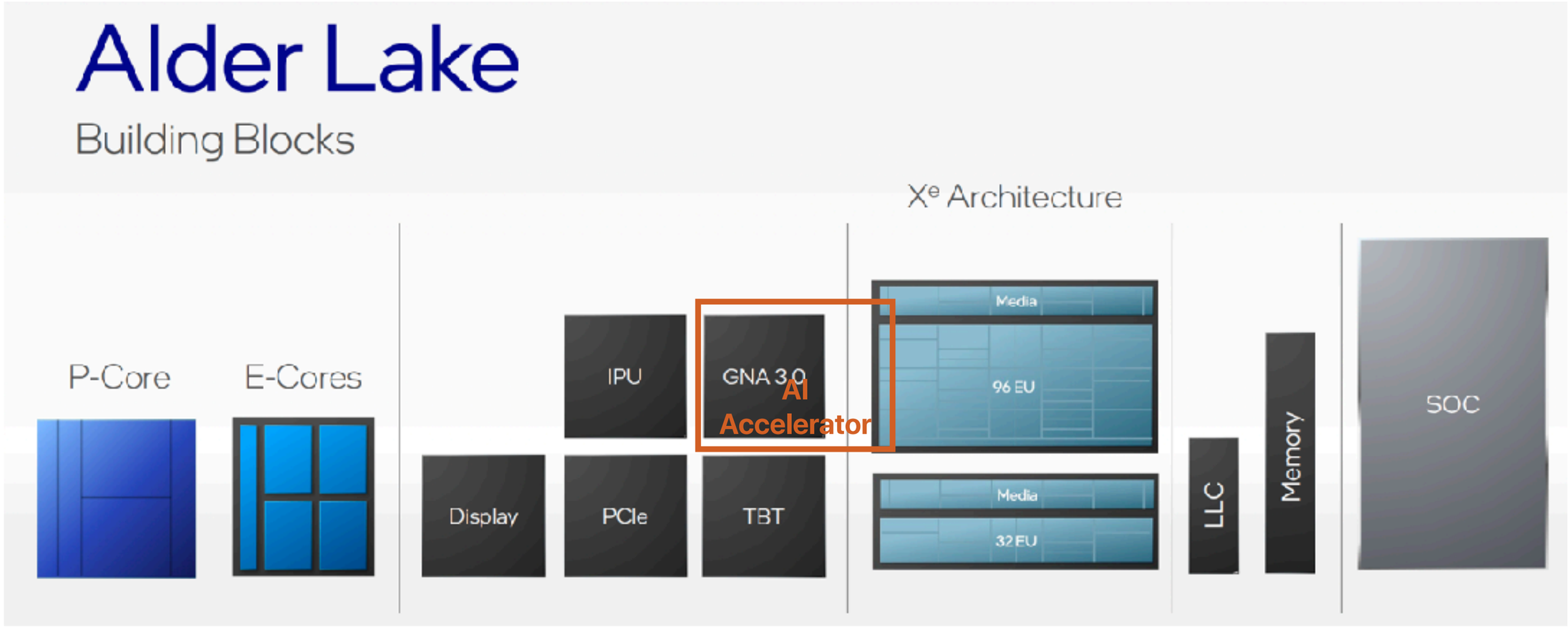


GPU

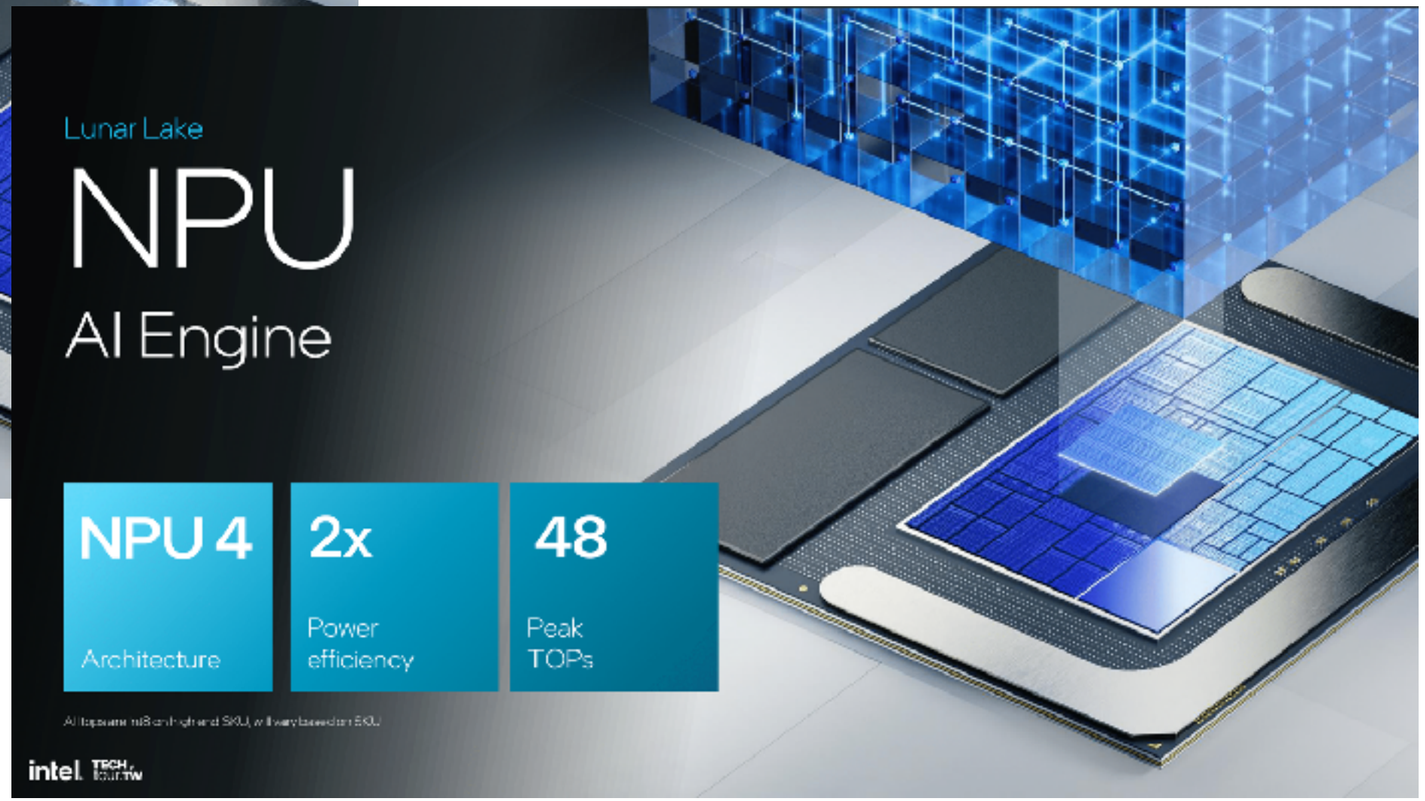
SSD

NIC

Modern processors also contain “accelerators”




Lunar Lake
CPU
AI Engine
P-core & E-core
CPU architecture
VNNI & AMX
AI instructions
5
peak TOPs
intel.

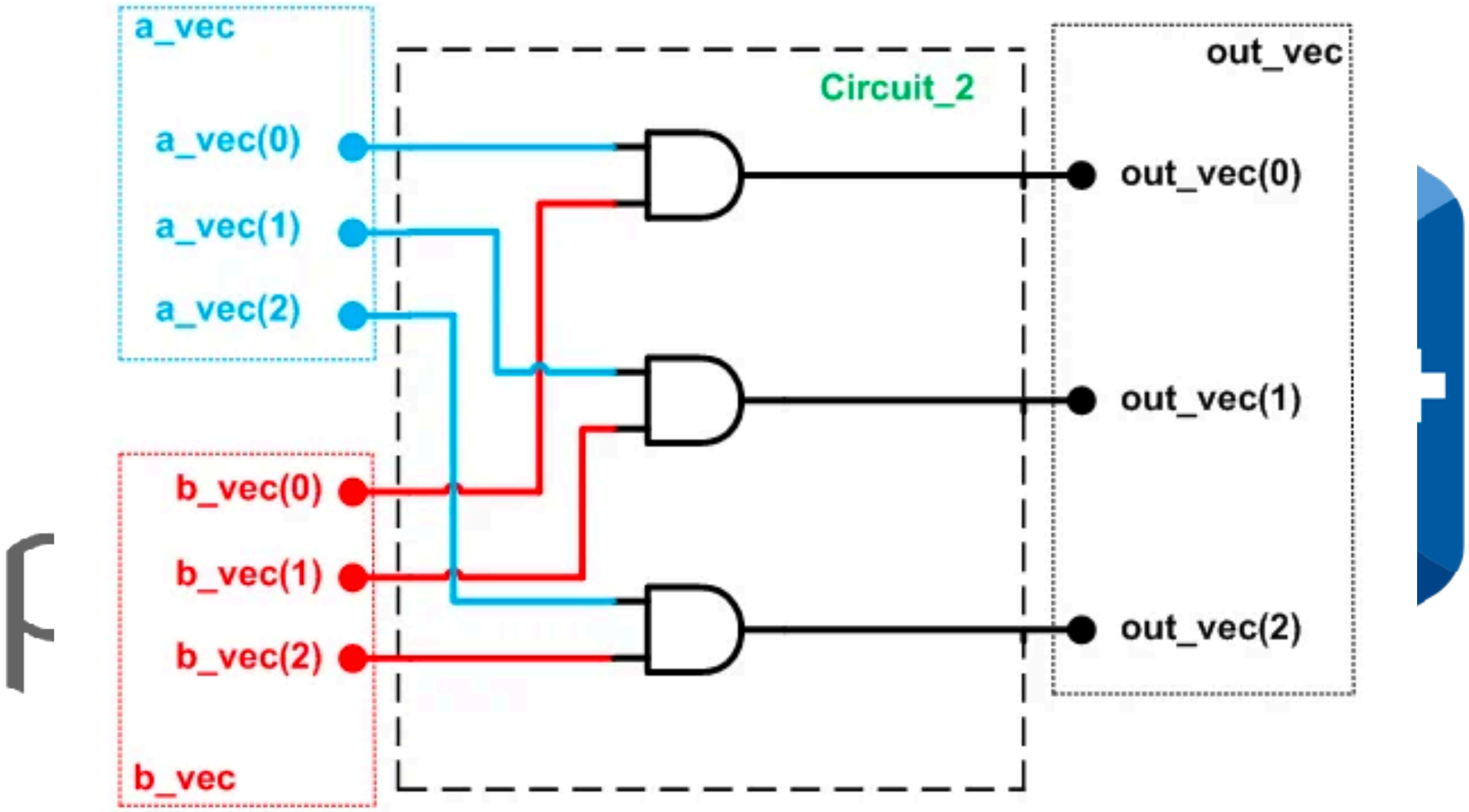

Lunar Lake
NPU
AI Engine
NPU 4
Architecture
2x
Power efficiency
48
Peak TOPs
intel.

Trends in the Dark Silicon Era

- Aggressive dynamic voltage/frequency scaling
  - Potentially hurts performance and increases **energy consumption**
  - Trade-offs between UX/Power/Energy
- Throughout oriented — slower, but more
  - Hurts latency
  - Limited applications — not every workload reveals strong data-level parallelism
- Just let it dark — activate part of circuits, but not all
  - Increases manufacturing costs
- From general-purpose to domain-specific — ASIC
  - Not flexible
  - Hard to design

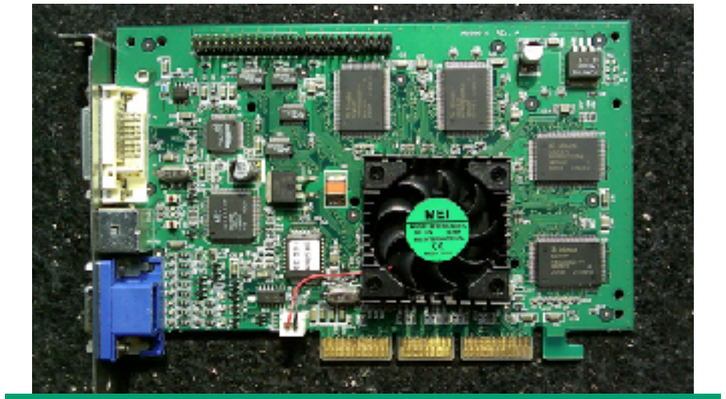
Conclusion: A New Golden Age

Is building domain specific accelerators the only avenue?

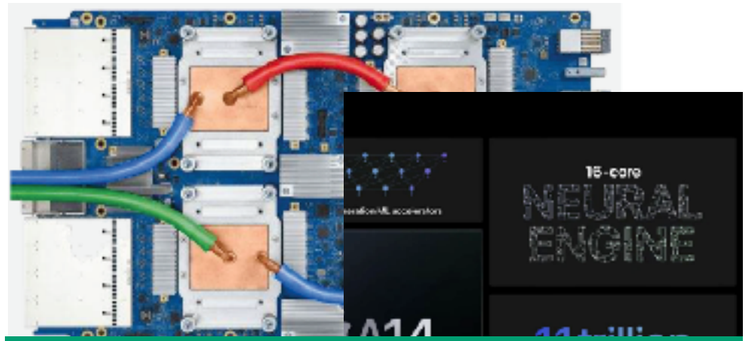


The evolutionary cycle of computing

A combination of special-purposed logical units



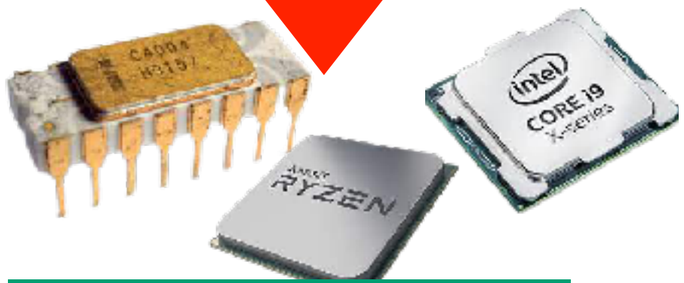
Graphics "Accelerator"



AI/ML/NN/RayTracing "Accelerators"

Special-purpose

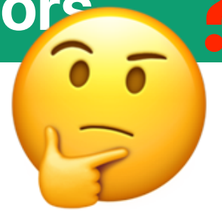
General-purpose



General-purpose microprocessors

General Purpose Computing On GPUs

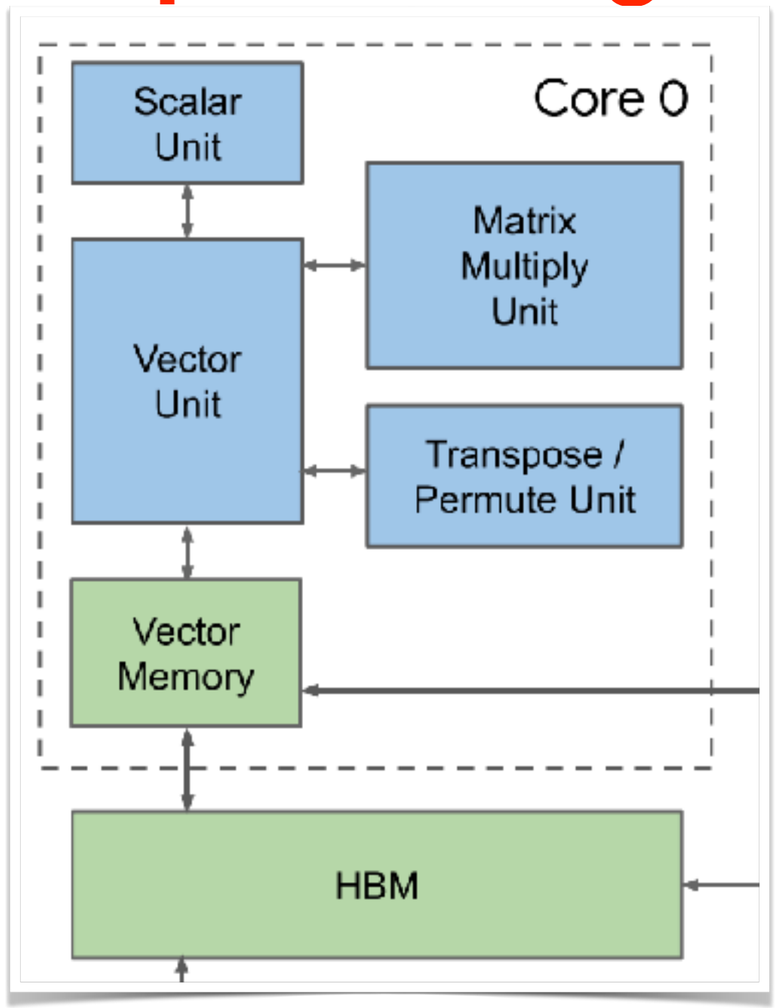
General Purpose Computing On Modern Accelerators



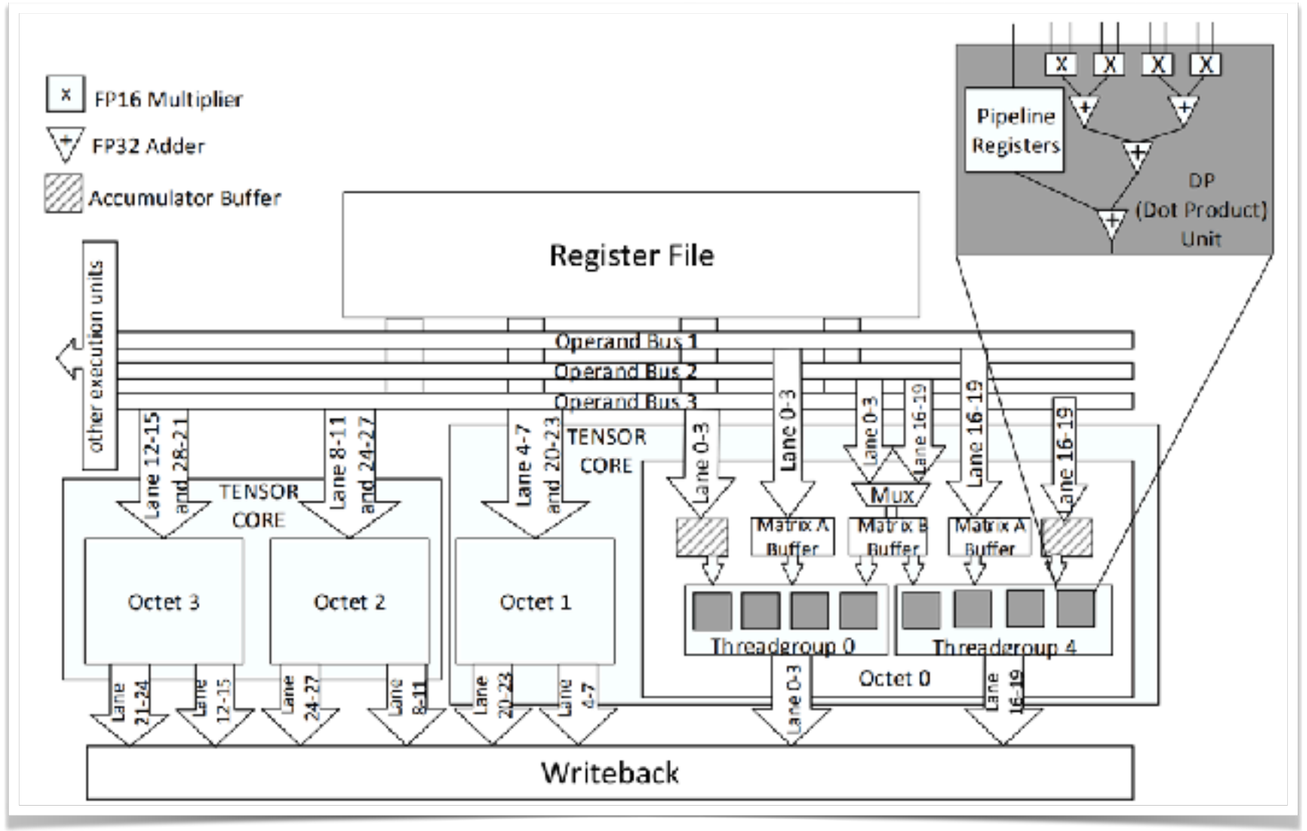
ESCAL

UC RIVERSIDE

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

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

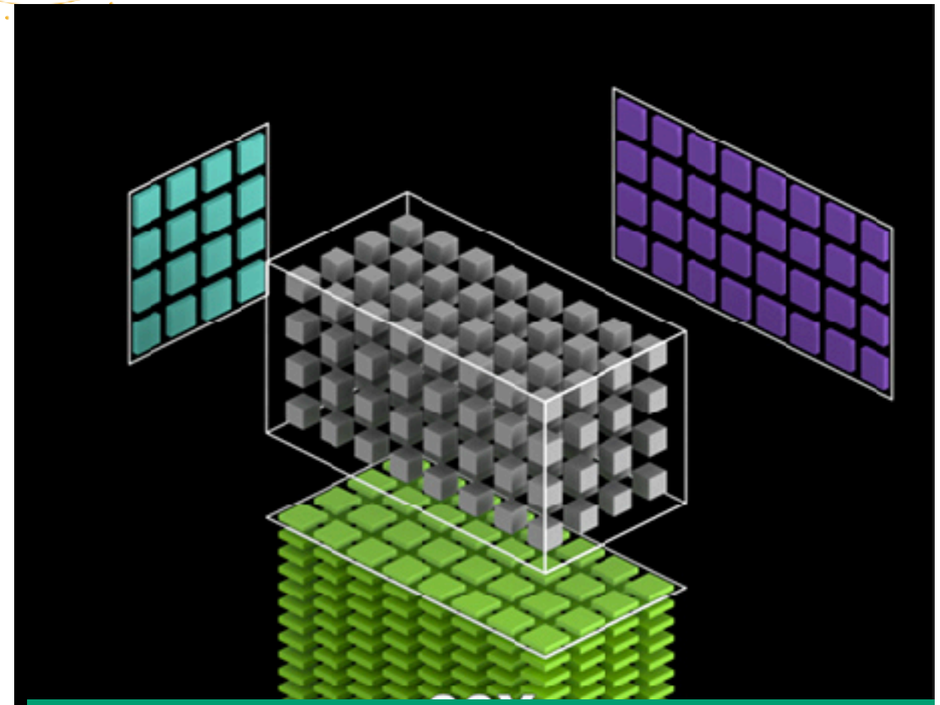
Customer/ ProductID	i9-12900	i7-12700	660p	RX5000	RyZen 5800G	RTX3080	980Pro
Abe	1	1	2	0	0	1	0
Bob	0	0	0	0	1	0	1
Cindy	0	2	2	0	0	1	0
Diana	0	0	0	1	1	0	1

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

How can Tensor Cores faster despite higher algorithm complexity?



Each tensor core operation performs 8x4x4 MMA in one cycle
~ 256 FP operations in conventional scalar models

	RTX 3090
CUDA Cores	10496
Tensor Cores	328

FLOPS of 8K by 8K matrix multiplications =

$$8192 \times 8192 \times 8192 \times 2$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{10496} = 104{,}755{,}300 \ CUDA \ core \ cycles$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{328 \times 256} = 13{,}094{,}413 \ Tensor \ core \ cycles$$

Tensor cores are 8x faster

TCUDB: Accelerating Database with Tensor Processors

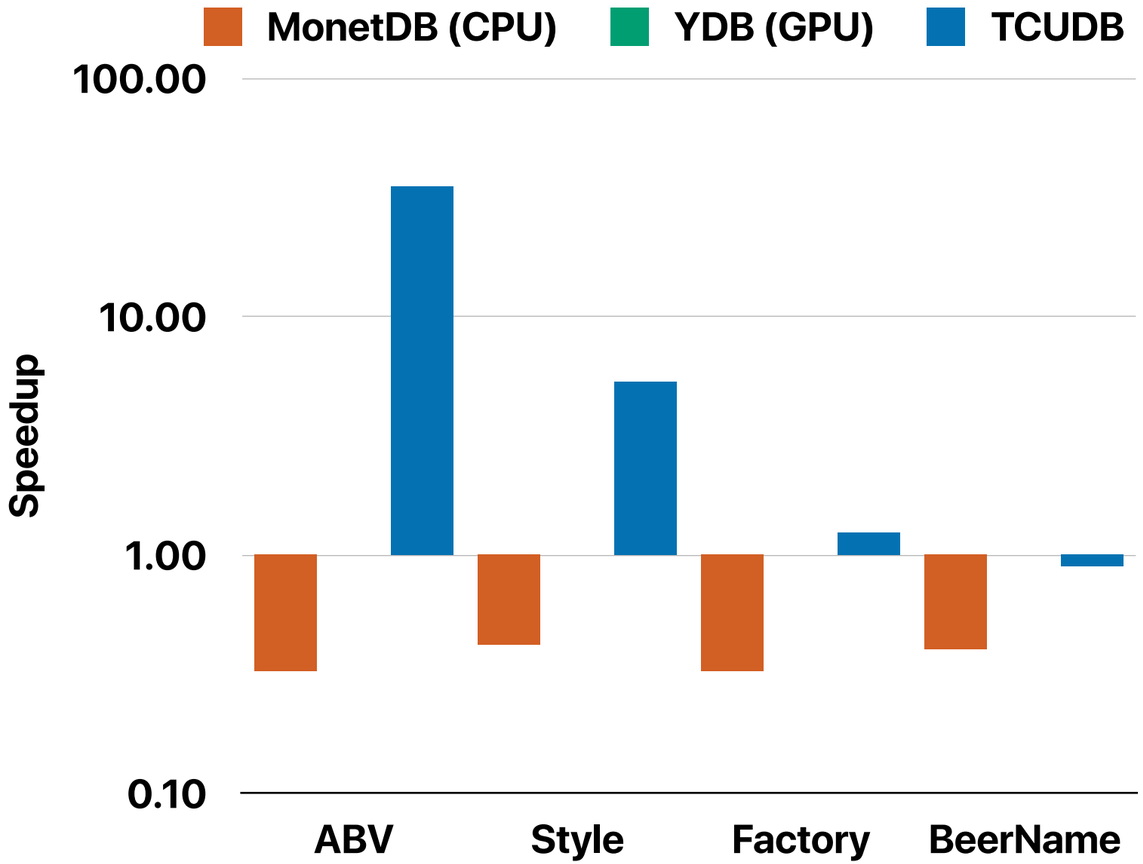
Yu-Ching Hu, Yuliang Li* and Hung-Wei Tseng
University of California, Riverside and Megagon Labs*
In The ACM SIGMOD/PODS 2022 International Conference on Management of Data.
https://github.com/escalab/TCUDB

Megagon Labs

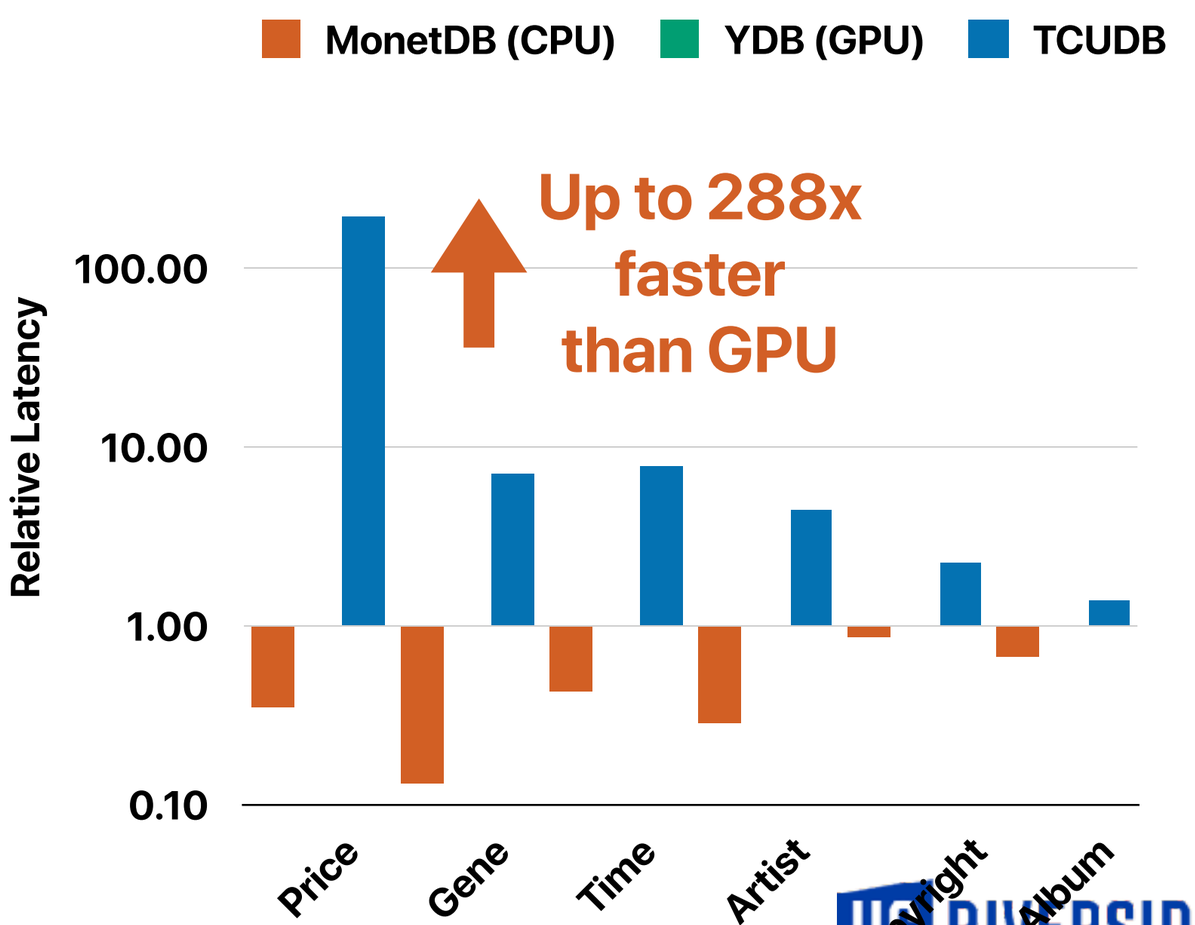
ESCAL

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;

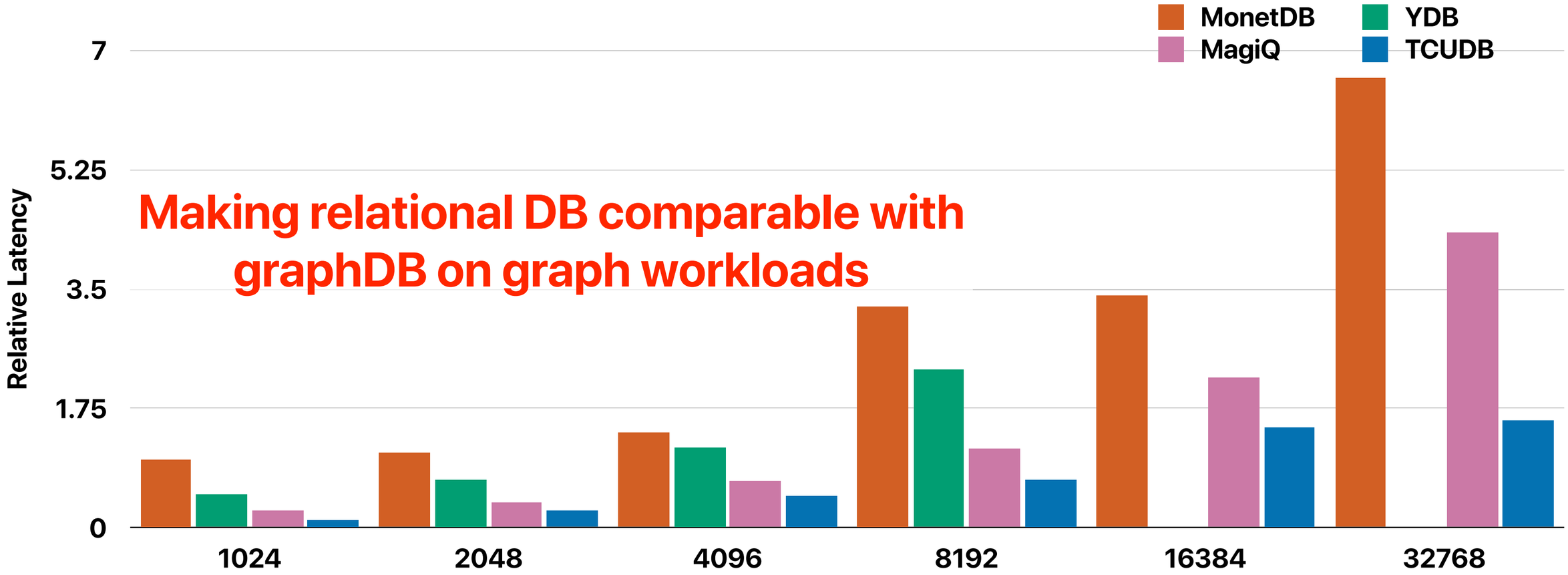


- SELECT TABLE_A.ID, TABLE_A.SONG, TABLE_B.ID, TABLE_B.SONG FROM TABLE_A, TABLE_B WHERE TABLE_A.ARTIST = TABLE_B.ARTIST;



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

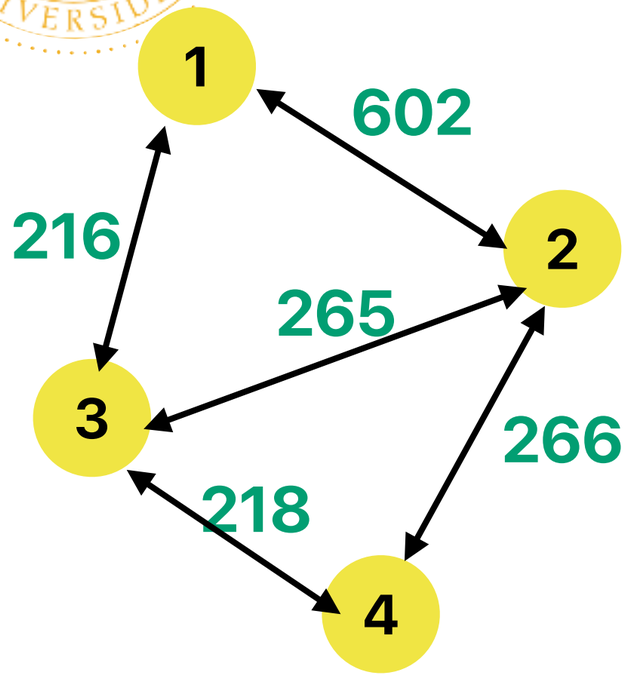
sum(| 1 | 0 | 602 | 216 | 0 | ⊙ | 2 | 602 | 0 | 265 | 266 |)

	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

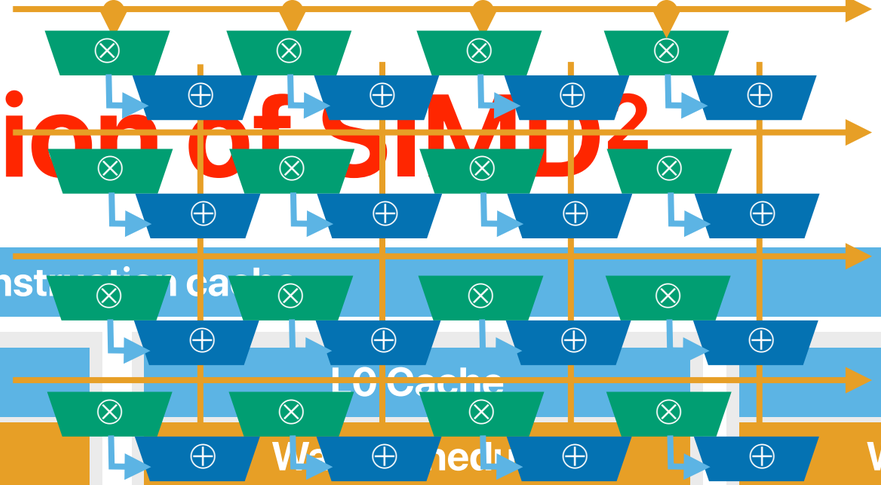
In The International Symposium on Computer Architecture (ISCA). 2022

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 SIMD²



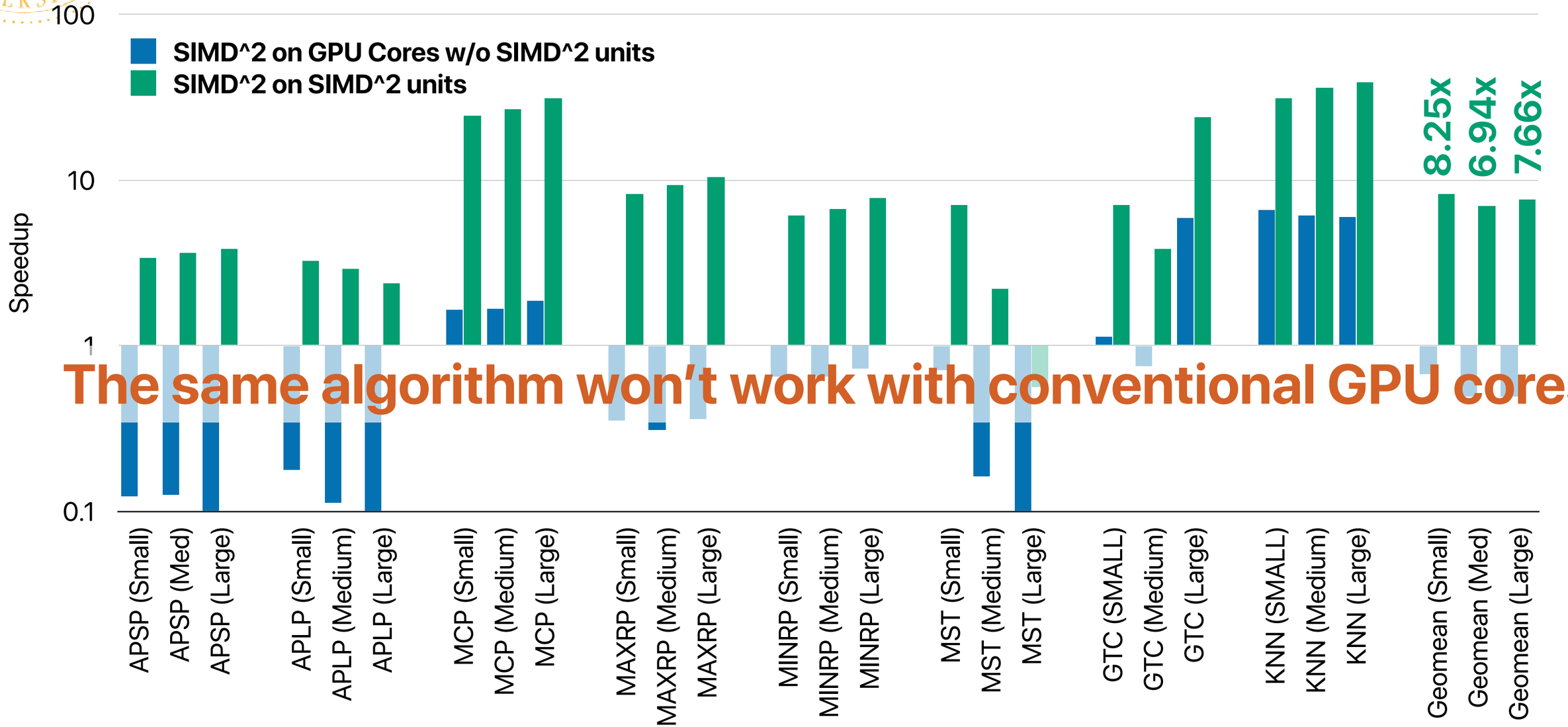
L1 instruction cache

L0 Cache	L0 Cache	L0 Cache	L0 Cache
Warp Scheduler	Warp Scheduler	Warp Scheduler	Warp Scheduler
Dispatch Unit	Dispatch Unit	Dispatch Unit	Dispatch Unit
Register File	Register File	Register File	Register File
INT32 FP32 FP32 FP64 ×8	INT32 FP32 FP32 FP64 ×8	INT32 FP32 FP32 FP64 ×8	INT32 FP32 FP32 FP64 ×8
SIMD² Cores	SIMD² Cores	SIMD² Cores	SIMD² Cores
Core	Core	Core	Core

L1 Data Cache/Shared Memory

TEX	TEX	TEX	TEX

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
- Being a “programmer” is easy. You need to know architecture a lot to be a “performance programmer”
  - Branch prediction
  - Cache
- 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.

One more thing...

Format of the final

- Cumulative, don’t forget your midterm and midterm review
- Multiple choices (15—20 questions)
  - They’re like your clicker/midterm multiple choices questions
- Free-answer questions * 4 questions
  - Two of them are MS/CSE comprehensive examine questions
- Open-ended questions *4 questions
  - Explain your answer clearly, precisely and “correctly”.
  - You need to cite papers to prove your correctness.
  - If you include “incorrect” or “irrelevant” 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

Sample Final

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

CMP advantages

- How many of the following are advantages of CMP over traditional superscalar processor
  - ① CMP can provide better energy-efficiency within the same area
  - ② CMP can deliver better instruction throughput within the same die area (chip size)
  - ③ CMP can achieve better ILP for each running thread
  - ④ CMP can improve the performance of a single-threaded application without modifying code
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

What about “linked list”

- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the
  code snippet determines the
  performance?

```
A.do {
B.     number_of_nodes++;
C.     current = current->next;
D.} while ( current != NULL );
```

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 ++;
    }
}
```

Why can’t we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  - ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
  - ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
  - ③ The branch outcome cannot be decided until the comparison result of ALU is not out
  - ④ The next instruction needs the branch instruction to write back its result

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

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];
    }
  }
}
```

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

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];
    }
  }
}
```

MS' "Configurable Clouds"

- Regarding MS' configurable clouds that are powered by FPGAs, please identify how many of the following are correct
  - ① Each FPGA is dedicated to one machine
  - ② Each FPGA is connected through a network that is separated from the data center network
  - ③ FPGA can deliver shorter average latency for AES-CBC-128-SHA1 encryption and decryption than Intel's high-end processors
  - ④ FPGA-accelerated search queries are always faster than a pure software-based datacenter

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

Summary of Optimizations

- Regarding the following cache optimizations, how many of them would help improve miss rate?
  - ① Non-blocking/pipelined/multibanked cache
  - ② Critical word first and early restart
  - ③ Prefetching
  - ④ Write buffer

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

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

3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and A, B, C: associativity, block size, capacity
  How many of the following are correct?
  - ① Increasing associativity can reduce conflict misses
  - ② Increasing associativity can reduce hit time
  - ③ Increasing block size can increase the miss penalty
  - ④ Increasing block size can reduce compulsory misses

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

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

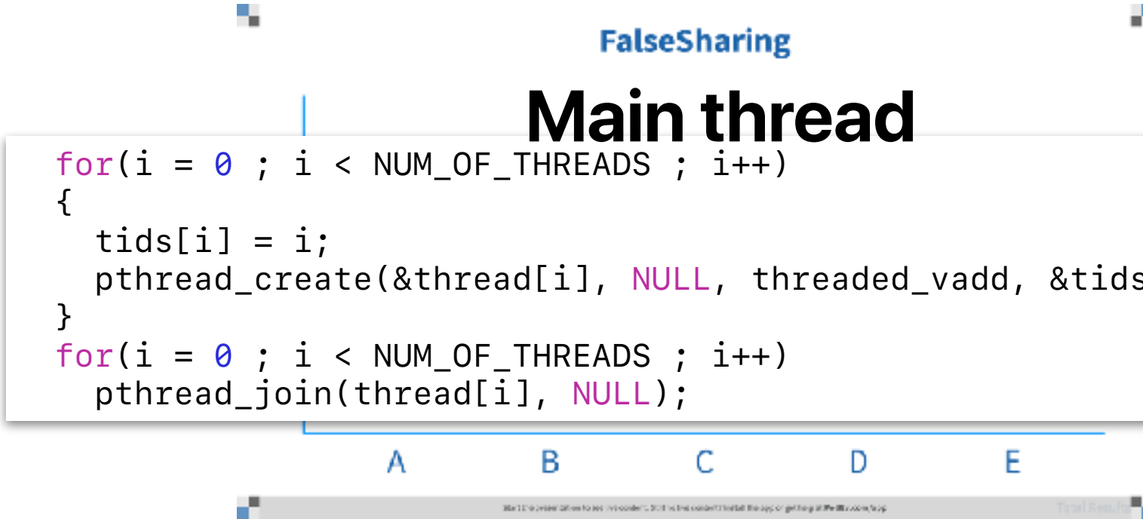
Version L

```
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



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: 4 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

```
uint64_t* conflict(uint64_t * data, uint64_t size, uint64_t arg1) {

    uint64_t sum=0;

    for(uint i = 0; i < arg1; i++)
    {
        for(uint x = i; x < size; x+=arg1) {
            sum += data[x];
        }
    }
    data[0] = sum;
    return data;
}
```

- On an intel Core i3 12100F, what values of `arg1` and `size` can —
  - Lead to 100% miss rate?
  - Lead to 50% miss rate?
  - Lead to almost 0% miss rate?

Reverse caching

```
uint64_t* conflict(uint64_t * data, uint64_t size, uint64_t arg1) {

    uint64_t sum=0;

    for(uint i = 0; i < arg1; i++)
    {
        for(uint x = i; x < size; x+=arg1) {
            sum += data[x];
        }
    }
    data[0] = sum;
    return data;
}
```

- If the size is 8192 and arg is 16, cache miss rate is 100%, what's the cache size, block size & associativity? Why?

Cache simulation

- The processor has a 8KB, 256B blocked, 2-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?

Latency numbers every programmer should know

- Review this and explain why
- https://colin-scott.github.io/personal_website/research/interactive_latency.html

Open-ended questions

SMT v.s. CMP

- Both CMP & SMT exploit thread-level or task-level parallelism. Assuming both application X and application Y have similar instruction combination, say 60% ALU, 20% load/store, and 20% branches. Consider two processors:

  P1: CMP with a 2-issue pipeline on each core. Each core has a private L1 32KB D-cache

  P2: SMT with a 4-issue pipeline. 64KB L1 D-cache

  Which one do you think is better?

  A. P1
  B. P2

Other open-ended questions

- Given the instruction front-end is decoupled from the backend of the pipeline ALUs, do you think ISA still affect performance?
  - Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA) (HPCA '13). https://minds.wisconsin.edu/handle/1793/64923
  - Ashish Venkat and Dean M. Tullsen. 2014. Harnessing ISA diversity: design of a heterogeneous-ISA chip multiprocessor. In Proceeding of the 41st annual international symposium on Computer architecuture (ISCA '14). http://www.cs.virginia.edu/venkat/papers/isca2014.pdf
- Reading the following papers and think of good use cases of simultaneous multithreading
  - Weifeng Zhang, Dean M. Tullsen, Brad Calder. Accelerating and Adapting Precomputation Threads for Efficient Prefetching. In 13th International Symposium on High Performance Computer Architecture, January, 2007.
  - A. Roth and G. S. Sohi. Speculative data-driven multithreading. Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, 2001, pp. 37-48, doi: 10.1109/HPCA.2001.903250.
  - J. D. Collins et al.. Speculative precomputation: long-range prefetching of delinquent loads. Proceedings 28th Annual International Symposium on Computer Architecture, Gothenburg, Sweden, 2001, pp. 14-25, doi: 10.1109/ISCA.2001.937427.

Other open-ended questions

- We're now living in a world with heterogeneous cores and simultaneous multithreading, what do you think an operating system scheduler should do to optimize —
  - Throughput of applications
  - Average response time
  - Energy consumption
  - Power consumption
  - Allan Snavely and Dean M. Tullsen. 2000. Symbiotic jobscheduling for a simultaneous multithreaded processor. In Proceedings of the ninth international conference on Architectural support for programming languages and operating systems (ASPLOS IX). https://doi.org/10.1145/378993.379244
- What features in modern processor architecture enable the potential of "Meltdown and Spectre" attacks? Should we live without those features? How to solve these security issues?
  - M. D. Hill, J. Masters, P. Ranganathan, P. Turner and J. L. Hennessy, "On the Spectre and
  - Meltdown Processor Security Vulnerabilities," in IEEE Micro, vol. 39, no. 2. http://pages.cs.wisc.edu/~markhill/papers/ieeemicro19_spectre_meltdown_2019_01_30

Other open-ended questions

- If you're asked to design a machine learning hardware, what will you do?
- If you're asked to build an Xeon Phi type processor where each core also has many-way SMT, are you going to give the processor more cache or better branch predictor?
- Can we focus on improving the throughput of computing instead of latency? Can you give an example on what type of applications will not work well in this way
- Pros and cons for branch prediction using perceptrons?
- What compiler optimizations would not be effective given OoO execution hardware?

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?

Announcements

- **iEVAL** started and ends on 6/07/2024
  - Submit the prove of your participation in iEVAL through Gradescope
  - It can become a full credit reading quiz (it helps to amortize the penalty of another least performing one)
- **Assignment 5** is **due 6/09/2024**
  - The same programming assignment as Assignment 3 but you need to speedup by 4x on **Gradescope** this time
  - The Gradescope/AWS server gets busy and more "non-deterministic" toward the deadline
- **Final exam**
  - 6/14 8a-11a @ **BOYHL 1471**
  - Closed book, no cheatsheet — the same rules as the midterm
  - Two questions can be used as CSMS comprehensive examine questions — one is memory-hierarchy related, and the other is OoO scheduling and code optimization


Computer
Science &
Engineering
203
ありがとう
Prof. Usagi