Domain specific computing architectures - II

Asif Ali Khan

Fall Semester 2024

Department of Computer Systems Engineering

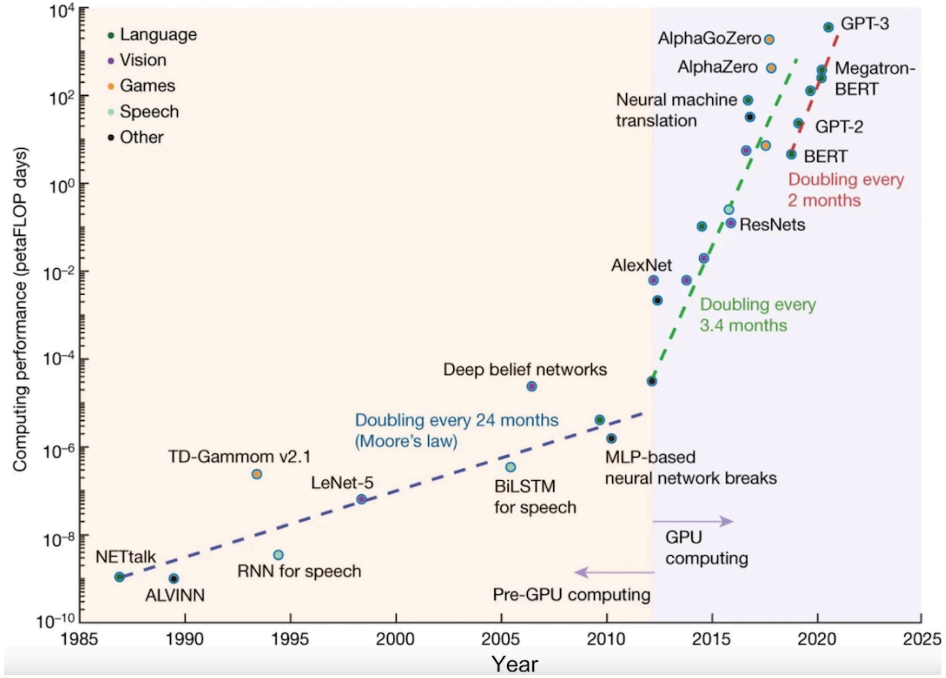
UET Peshawar, Pakistan

Oct 24, 2024

Recap: Programmability of Von Neumann systems

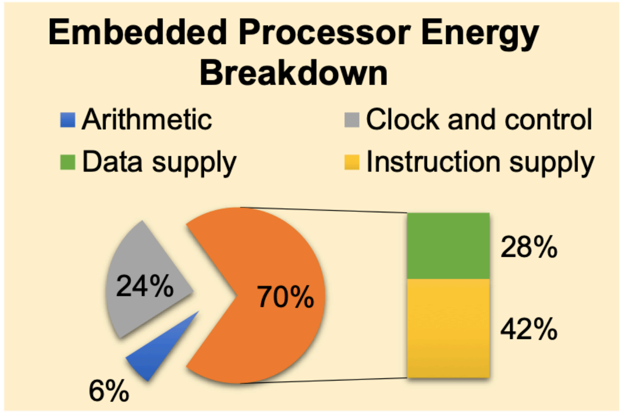
- ISA is the set of instruction the processor can understand and is the interface between h/w and s/w
- High level languages are typically used to program these systems
- Compilers transform this hardware-agnostic representation to machine code
- A compiler typically has front-end, middle-end, and back-end

Recap: why are DSAs needed?



Aguirre et al, Nature Communications, 2024

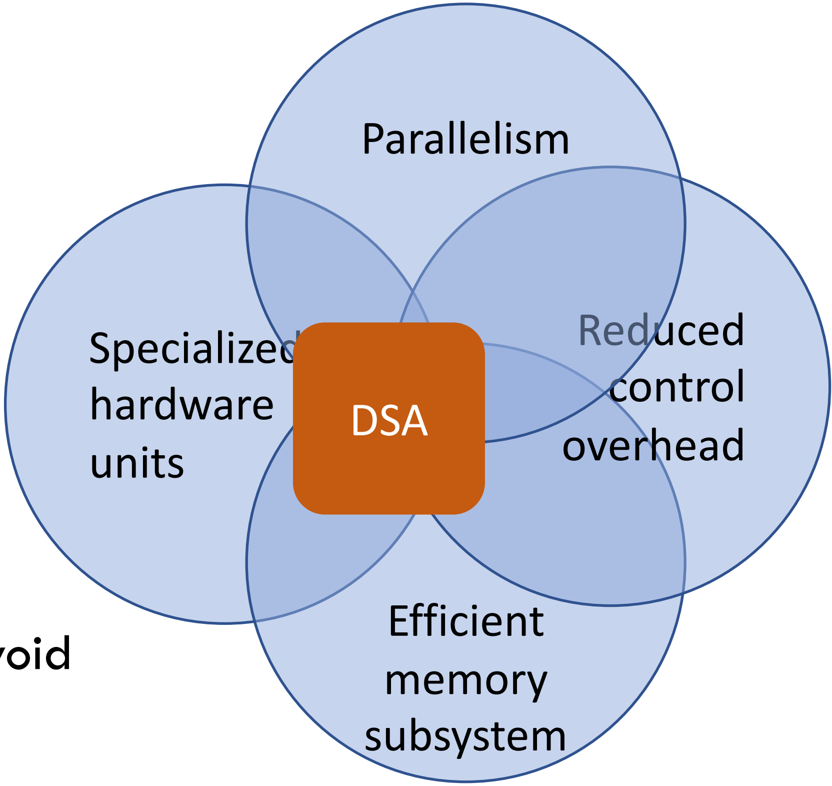
Traditional general-purpose (GP) computing can not fulfill the compute-demands of these applications due to their inefficiencies



Dally et al, Efficient embedded computing, IEEE'08

DSAs: Important aspects

- ❑ Specialization can be in:
  - ❑ Hardware
  - ❑ Data
- ❑ Parallelism can be decided as per the domain requirements
- ❑ Efficient local memories can be utilized to avoid long data access latencies
- ❑ The control overhead of over 90% in CPUs can be significantly reduced



DSAs: Specialization

- Specialization can be in:
  - Data
  - Hardware

DSAs: Specialization

- Specialization can be in:
  - Data
  - Hardware

- Data specialization: Instead of using the general-purpose 32/64 bits FP, use specialized types for different domains (for ML, 1-8 bits)

DSAs: Specialization

- Specialization can be in:
  - Data
  - Hardware

- Data specialization: Instead of using the general-purpose 32/64 bits FP, use specialized types for different domains (for ML, 1-8 bits)

- Hardware specialization: Specialized hardware modules, e.g., multiply-and-accumulate (MAC) unit, adder tree, other special function units (e.g., for softmax)

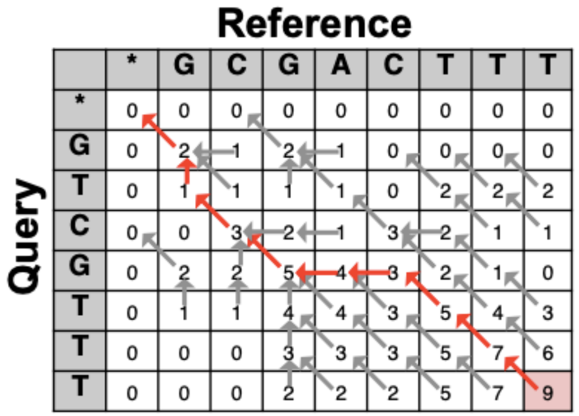
DSAs: Specialization

$$I(i,j) = \max \begin{cases} H(i, j-1) - o \\ I(i, j-1) - e \end{cases}$$

$$D(i,j) = \max \begin{cases} H(i-1, j) - o \\ D(i-1, j) - e \end{cases}$$

$$H(i,j) = \max \begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1, j-1) + W(r_i, q_j) \end{cases}$$

- ❑ Smith-Waterman algorithm for sequence alignment



	*	G	C	G	A	C	T	T	T
*	0	0	0	0	0	0	0	0	0
G	0	2	1	2	1	0	0	0	0
T	0	1	1	1	1	0	2	2	2
C	0	0	3	2	1	3	2	1	1
G	0	2	2	5	4	3	2	1	0
T	0	1	1	4	4	3	5	4	3
T	0	0	0	3	3	3	5	7	6
T	0	0	0	2	2	2	5	7	9

DSAs: Specialization

$$
\begin{aligned}
I(i,j) &= \max \begin{cases} H(i,j-1) - o \\ I(i,j-1) - e \end{cases} \\
D(i,j) &= \max \begin{cases} H(i-1,j) - o \\ D(i-1,j) - e \end{cases} \\
H(i,j) &= \max \begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1) + W(r_i, q_j) \end{cases}
\end{aligned}
$$

- Smith-Waterman algorithm for sequence alignment
- 15 loads/stores, 35 arithmetic/logic ops

DSAs: Specialization

$$
\begin{aligned}
I(i,j) &= \max \begin{cases} H(i,j-1) - o \\ I(i,j-1) - e \end{cases} \\
D(i,j) &= \max \begin{cases} H(i-1,j) - o \\ D(i-1,j) - e \end{cases} \\
H(i,j) &= \max \begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1) + W(r_i,q_j) \end{cases}
\end{aligned}
$$

- Smith-Waterman algorithm for sequence alignment
- 15 loads/stores, 35 arithmetic/logic ops
- Intel Xeon 14nm:
  - 37 cycles (latency), 81nj (energy)

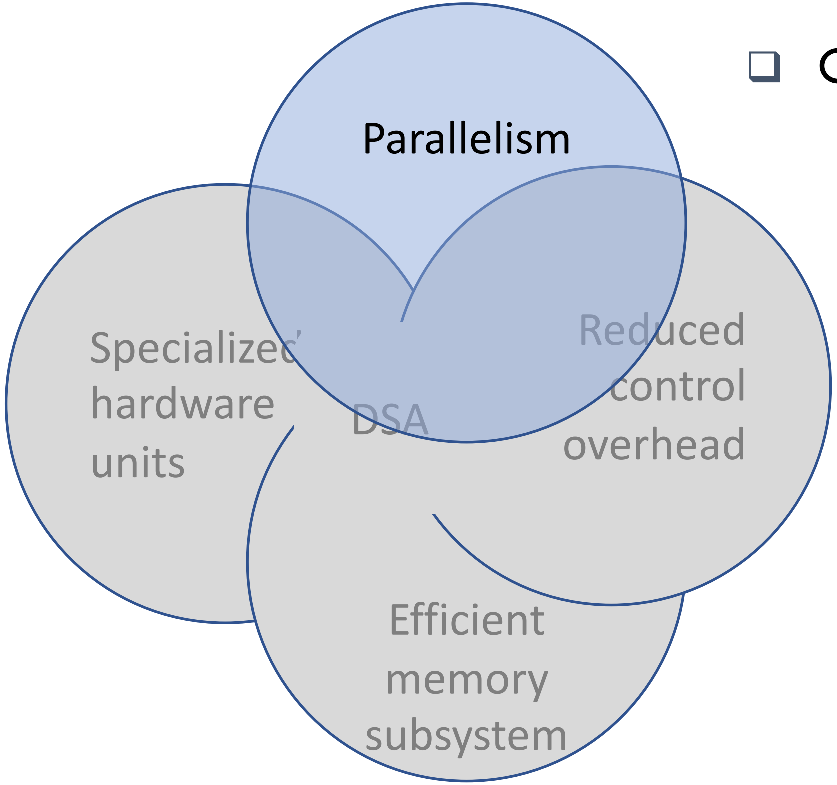
DSAs: Specialization

$$
\begin{aligned}
I(i,j) &= \max \begin{cases} H(i,j-1) - o \\ I(i,j-1) - e \end{cases} \\
D(i,j) &= \max \begin{cases} H(i-1,j) - o \\ D(i-1,j) - e \end{cases} \\
H(i,j) &= \max \begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1) + W(r_i, q_j) \end{cases}
\end{aligned}
$$

Turakhia, Y., Bejerano, G. and Dally, W.J., Darwin: A Genomics Co-processor Provides up to 15,000 X Acceleration on Long Read Assembly. ASPLOS, 2018

- Smith-Waterman algorithm for sequence alignment
- 15 loads/stores, 35 arithmetic/logic ops
- Intel Xeon 14nm:
  - 37 cycles (latency), 81nj (energy)
- Darwin accelerator:
  - Latency: 1 cycle (37x speedup)
  - Energy: 3.1pj (26,000x reduction)

DSAs: Parallelism

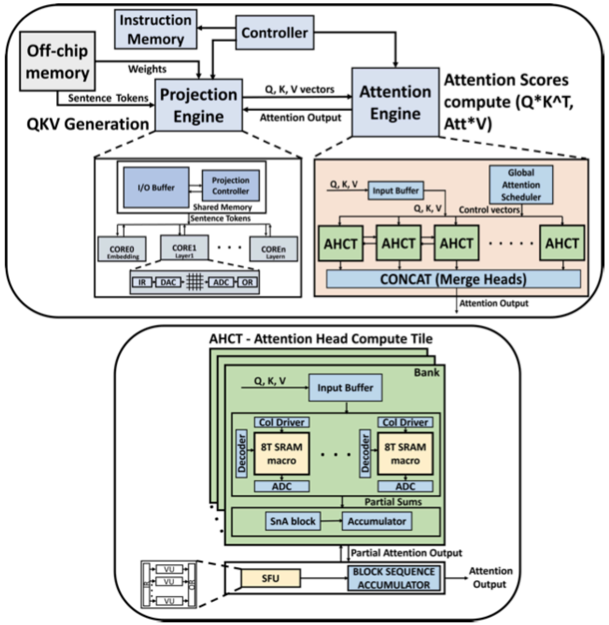


- One of the fundamental principals of DSAs

DSAs: Parallelism

- One of the fundamental principals of DSAs
- Parallelism is usually hierarchal, e.g., multiple MAC units per processing element (PE), multiple PEs per module, multiple modules per system

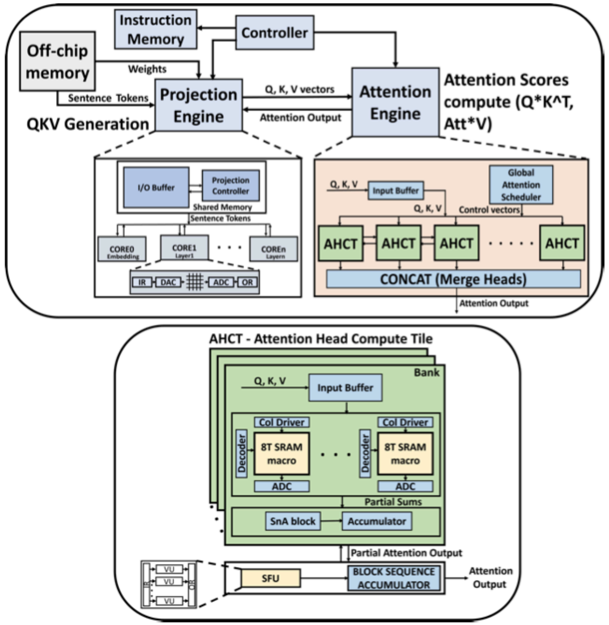
DSAs: Parallelism



S. Sridharan et al., "X-Former: In-Memory Acceleration of Transformers", IEEE TVLSI 2023

- ❑ One of the fundamental principals of DSAs
- ❑ Parallelism is usually hierarchal, e.g., multiple MAC units per processing element (PE), multiple PEs per module, multiple modules per system

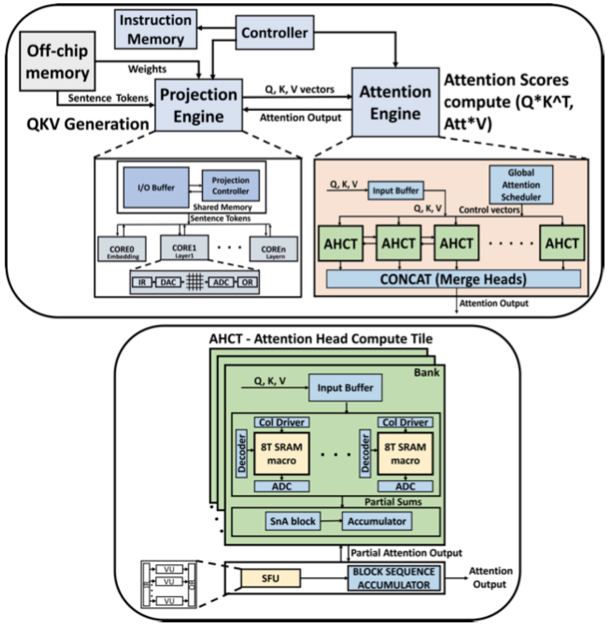
DSAs: Parallelism



S. Sridharan et al., "X-Former: In-Memory Acceleration of Transformers", IEEE TVLSI 2023

- ❑ One of the fundamental principals of DSAs
- ❑ Parallelism is usually hierarchal, e.g., multiple MAC units per processing element (PE), multiple PEs per module, multiple modules per system
- ❑ Memory organization is key to performance

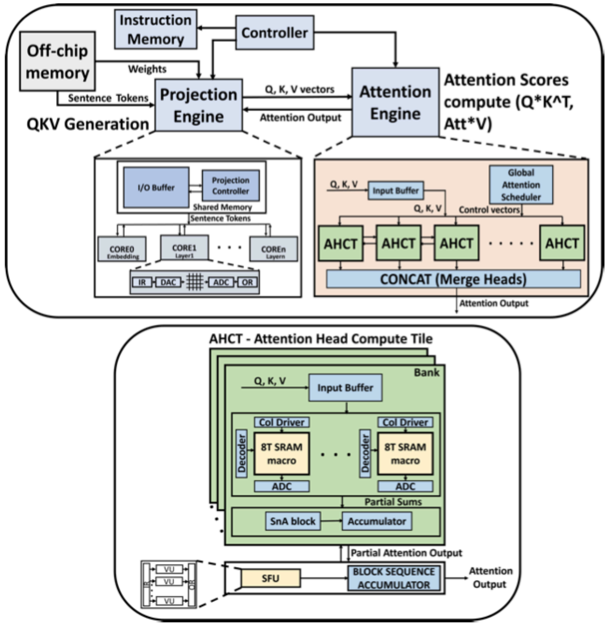
DSAs: Parallelism



S. Sridharan et al., "X-Former: In-Memory Acceleration of Transformers", IEEE TVLSI 2023

- One of the fundamental principals of DSAs
- Parallelism is usually hierarchal, e.g., multiple MAC units per processing element (PE), multiple PEs per module, multiple modules per system
- Memory organization is key to performance
- Parallel PEs must exploit locality

DSAs: Parallelism



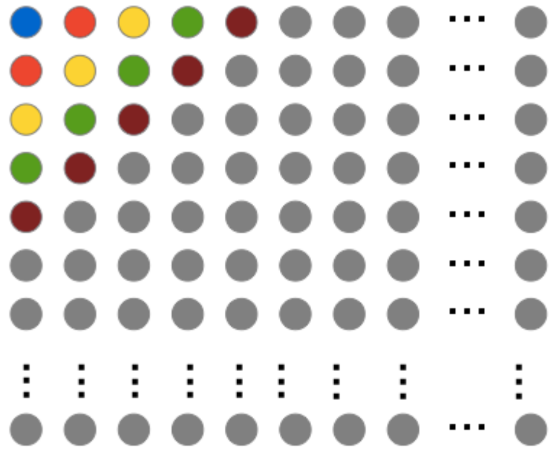
S. Sridharan et al., "X-Former: In-Memory Acceleration of Transformers", IEEE TVLSI 2023

- ❑ One of the fundamental principals of DSAs
- ❑ Parallelism is usually hierarchal, e.g., multiple MAC units per processing element (PE), multiple PEs per module, multiple modules per system
- ❑ Memory organization is key to performance
- ❑ Parallel PEs must exploit locality
- ❑ There should be ideally no cross-PE dependencies (we will see this in the UPMEM example)

DSAs: Parallelism in Darwin

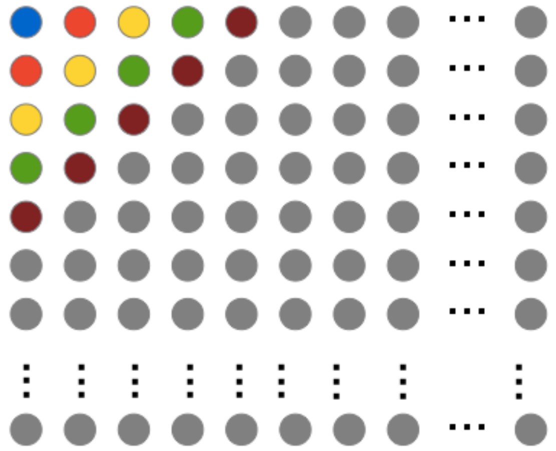
- Outer-loop: 64 PEs running alignment instances in parallel
  - No cross-PEs communication

DSAs: Parallelism in Darwin



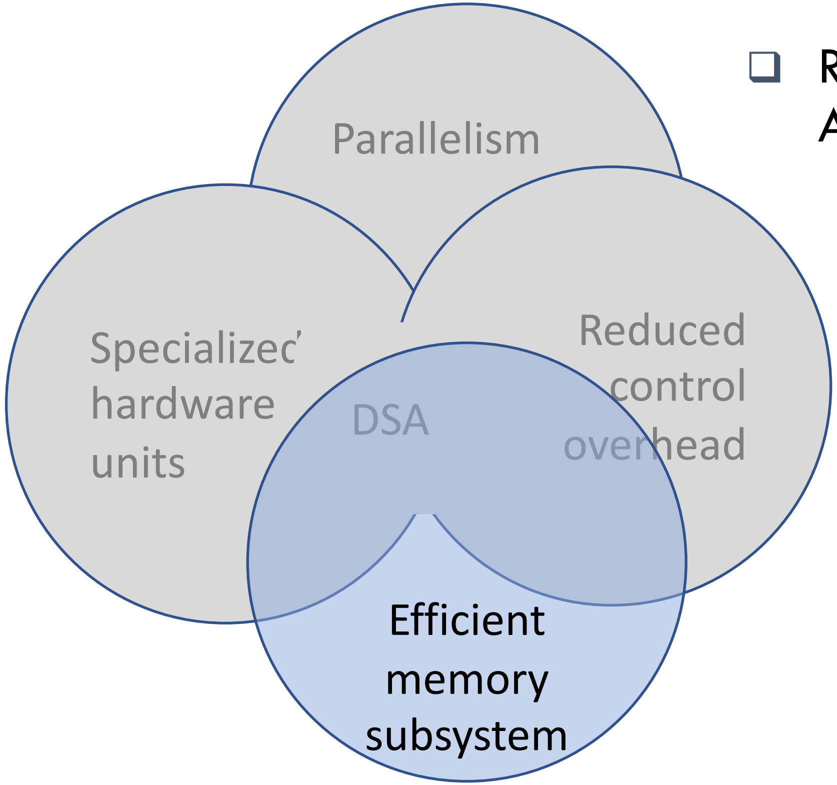
- Outer-loop: 64 PEs running alignment instances in parallel
  - No cross-PEs communication
- Inner-loop: In each one of them, 64 special function units compute H, I, Ds in parallel
  - Only neighbors communicate

DSAs: Parallelism in Darwin



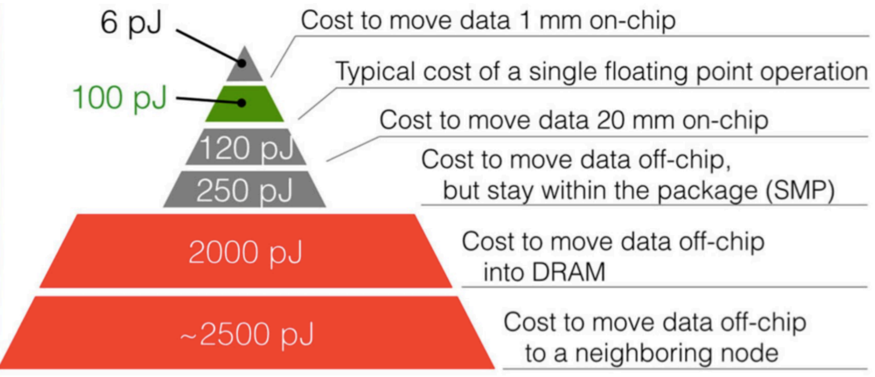
- Outer-loop: 64 PEs running alignment instances in parallel
  - No cross-PEs communication
- Inner-loop: In each one of them, 64 special function units compute H, I, Ds in parallel
  - Only neighbors communicate
- Speedup:
  - Specialization: 37x
  - Parallelization: 4034x
  - Total speedup: 150,000x

DSAs: Optimized memory



- Recall from the last lecture:
  Arithmetic is free; memory is expensive

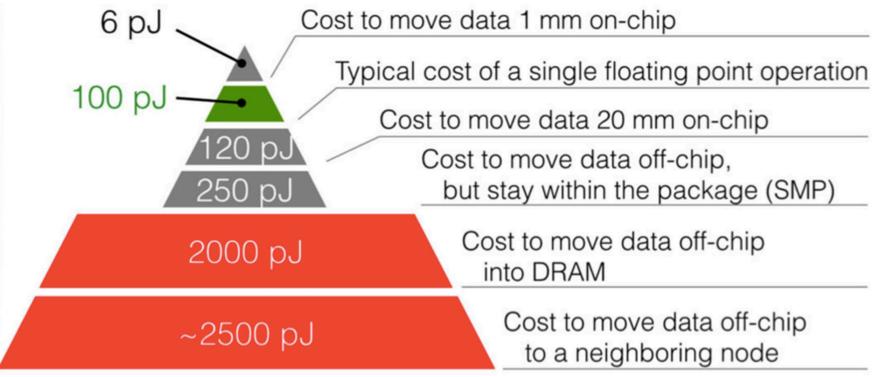
DSAs: Optimized memory


6 pJ
100 pJ
120 pJ
250 pJ
2000 pJ
~2500 pJ
Cost to move data 1 mm on-chip
Typical cost of a single floating point operation
Cost to move data 20 mm on-chip
Cost to move data off-chip, but stay within the package (SMP)
Cost to move data off-chip into DRAM
Cost to move data off-chip to a neighboring node

You, Y., et al. Fast LSTM by dynamic decomposition on cloud and distributed systems. Knowledge and Information Systems, 2020

- ❑ Recall from the last lecture that arithmetic is almost free; memory is expensive
- ❑ Data movement is prohibitively expansive

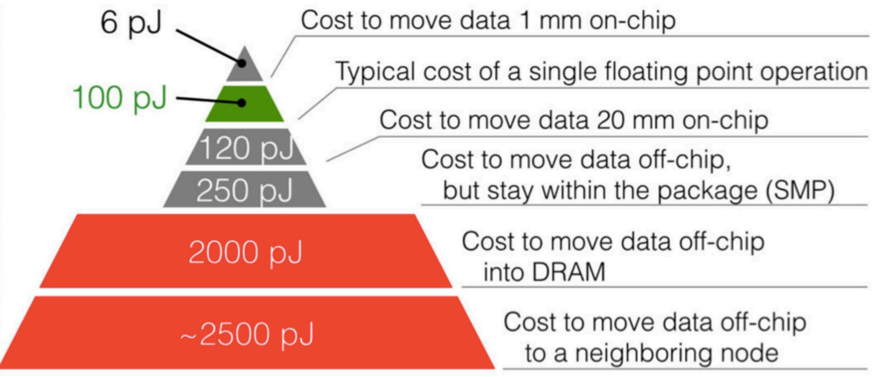
DSAs: Optimized memory


6 pJ
Cost to move data 1 mm on-chip
100 pJ
Typical cost of a single floating point operation
120 pJ
Cost to move data 20 mm on-chip
250 pJ
Cost to move data off-chip, but stay within the package (SMP)
2000 pJ
Cost to move data off-chip into DRAM
~2500 pJ
Cost to move data off-chip to a neighboring node

You, Y., et al. Fast LSTM by dynamic decomposition on cloud and distributed systems. Knowledge and Information Systems, 2020

- Recall from the last lecture that arithmetic is almost free; memory is expensive
- Data movement is prohibitively expansive
- Key: Use many small, local memory to
  - Minimize distance
  - Maximize reuse

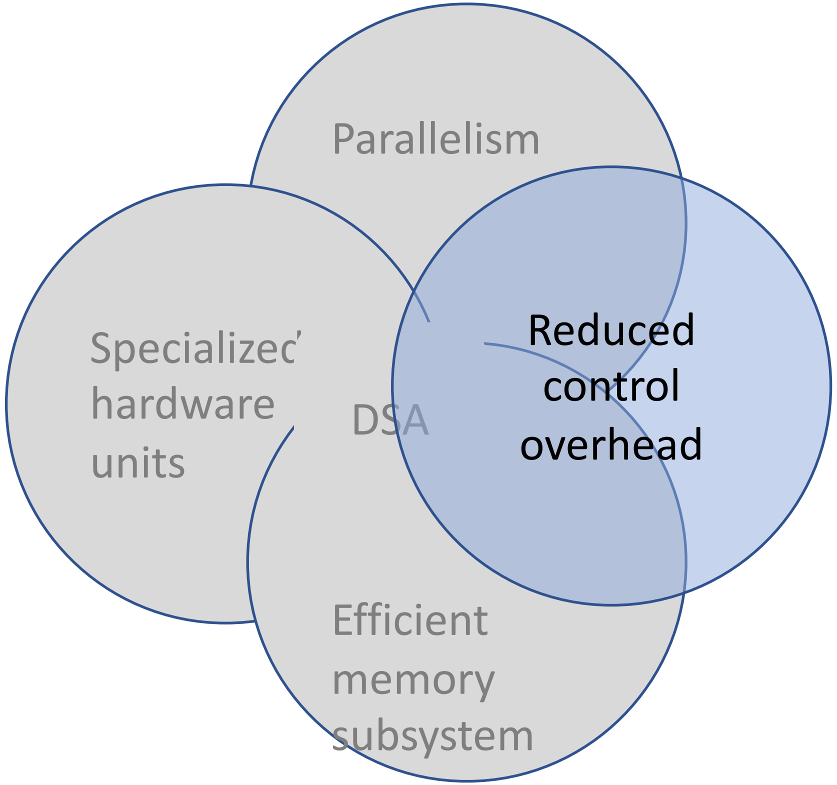
DSAs: Optimized memory


6 pJ
Cost to move data 1 mm on-chip
100 pJ
Typical cost of a single floating point operation
120 pJ
Cost to move data 20 mm on-chip
250 pJ
Cost to move data off-chip, but stay within the package (SMP)
2000 pJ
Cost to move data off-chip into DRAM
~2500 pJ
Cost to move data off-chip to a neighboring node

You, Y., et al. Fast LSTM by dynamic decomposition on cloud and distributed systems. Knowledge and Information Systems, 2020

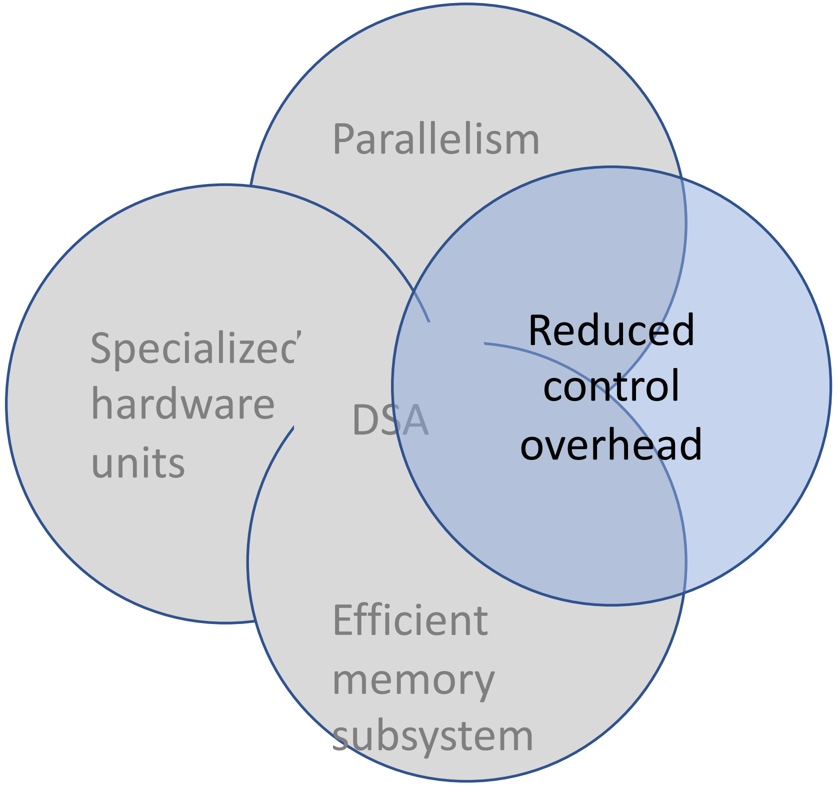
- Recall from the last lecture that arithmetic is almost free; memory is expensive
- Data movement is prohibitively expansive
- Key: Use many small, local memory to
  - Minimize distance
  - Maximize reuse
- Other optimizations:
  - Data compression
    - Increase the effective bandwidth/ capacity

DSAs: Reduced overhead



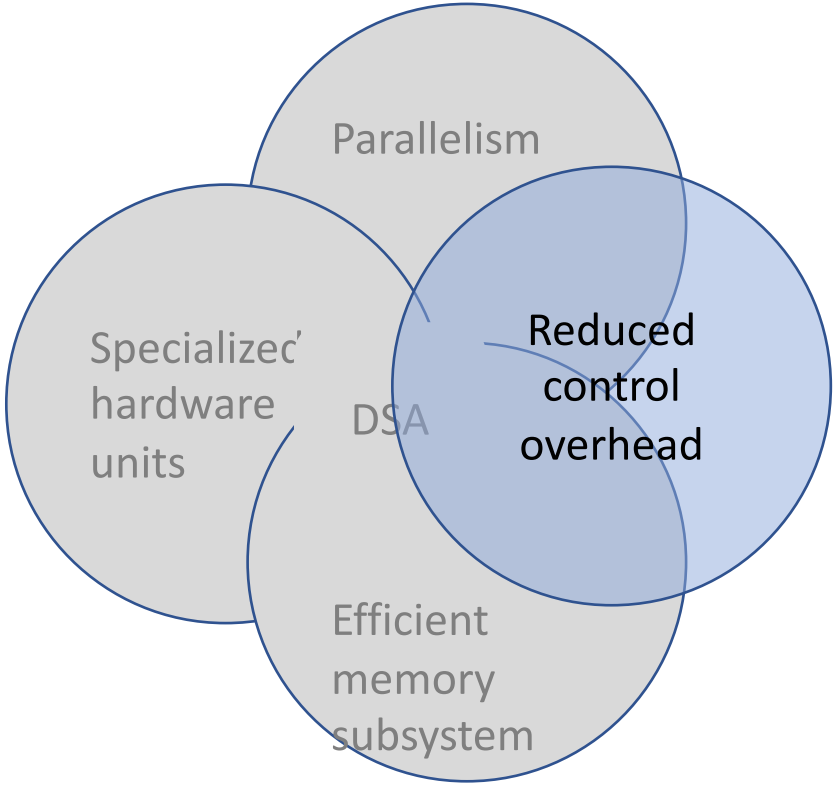
- Specialized hardware reduces overhead

DSAs: Reduced overhead



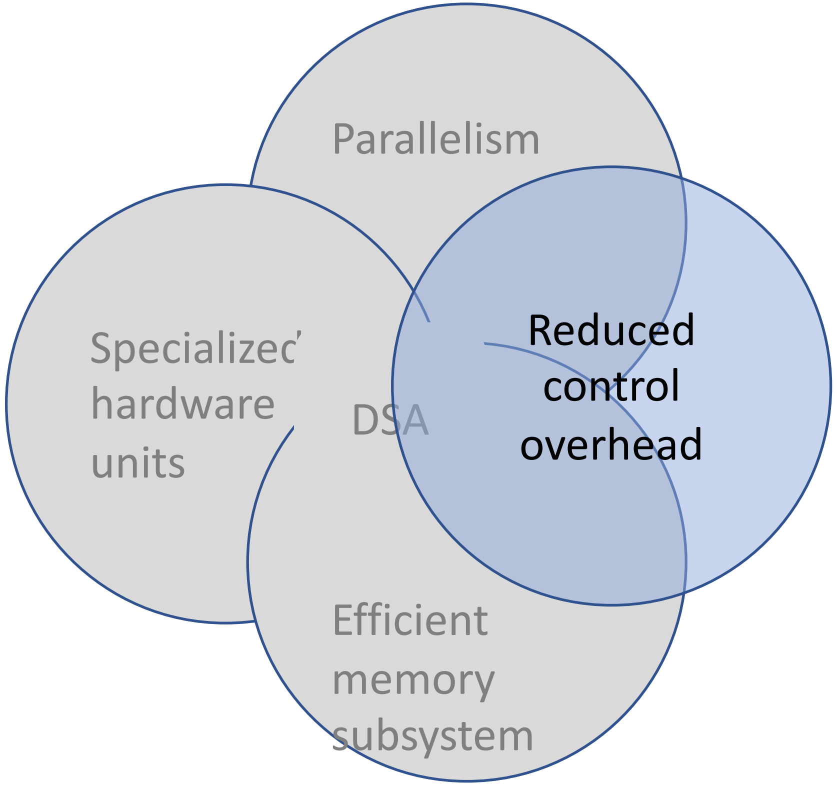
- Specialized hardware reduces overhead
- In-order general purpose CPU spends >90% energy on overhead (instruction fetch, decode, data, control etc)

DSAs: Reduced overhead



- Specialized hardware reduces overhead
- In-order general purpose CPU spends >90% energy on overhead (instruction fetch, decode, data, control etc)
- In OOO processor, this is over 99%

DSAs: Reduced overhead



- Specialized hardware reduces overhead
- In-order general purpose CPU spends >90% energy on overhead (instruction fetch, decode, data, control etc)
- In OOO processor, this is over 99%
- Example:
  - ARM A-15, integer add: 250pj
  - 32-bit CMOS adder: 68fj (4000x less)

GPU architecture and programming

- Extreme throughput-oriented processors

GPU architecture and programming

- Extreme throughput-oriented processors
- The basic computing unit of a GPU is called a CUDA core

GPU architecture and programming

- Extreme throughput-oriented processors
- The basic computing unit of a GPU is called a CUDA core
- CUDA cores are combined to form streaming multi-processors (SMs)

GPU architecture and programming

- Extreme throughput-oriented processors
- The basic computing unit of a GPU is called a CUDA core
- CUDA cores are combined to form streaming multi-processors (SMs)
- SMs are organized into grids

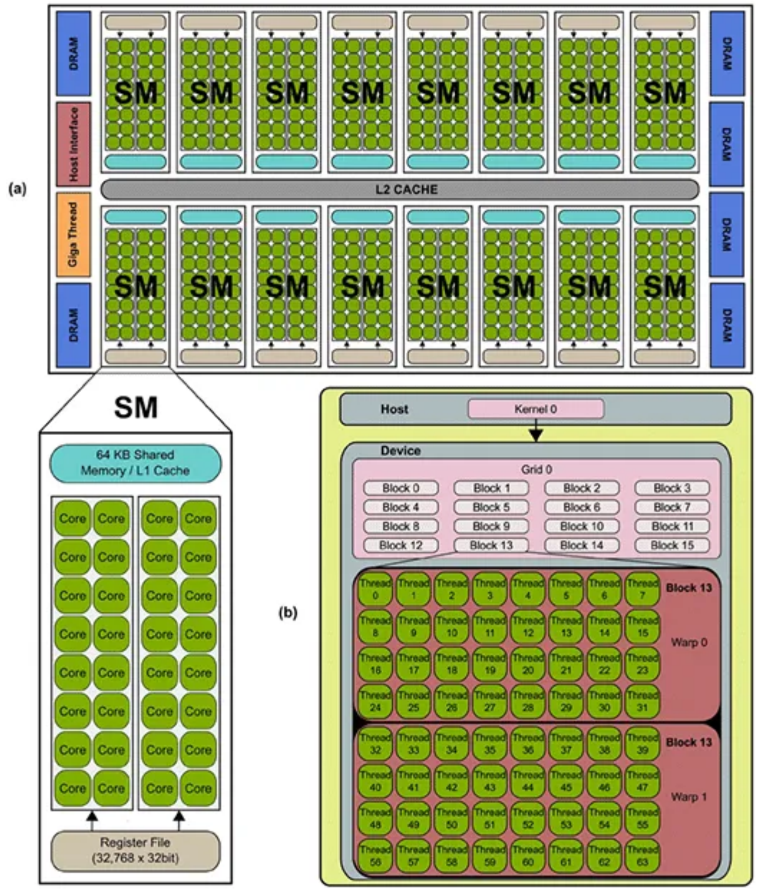
GPU architecture and programming

- ❑ Extreme throughput-oriented processors
- ❑ The basic computing unit of a GPU is called a CUDA core
- ❑ CUDA cores are combined to form streaming multi-processors (SMs)
- ❑ SMs are organized into grids
- ❑ Cores within an SM communicate via local (shared) memory

GPU architecture and programming

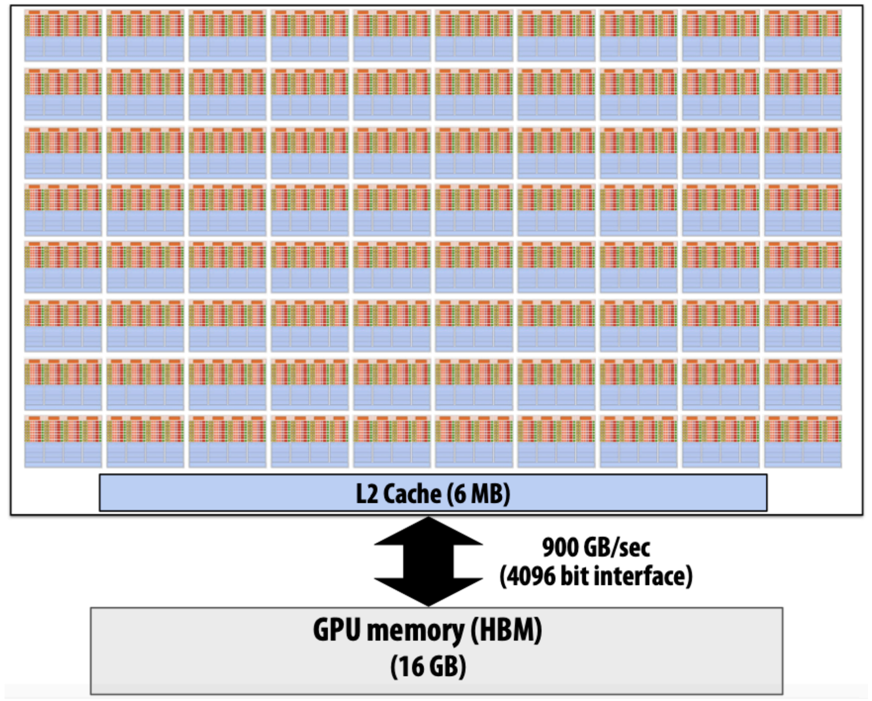
- Extreme throughput-oriented processors
- The basic computing unit of a GPU is called a CUDA core
- CUDA cores are combined to form streaming multi-processors (SMs)
- SMs are organized into grids
- Cores within an SM communicate via local (shared) memory
- SMs communicate via global (device) memory

GPU architecture and programming



GPU architecture and programming

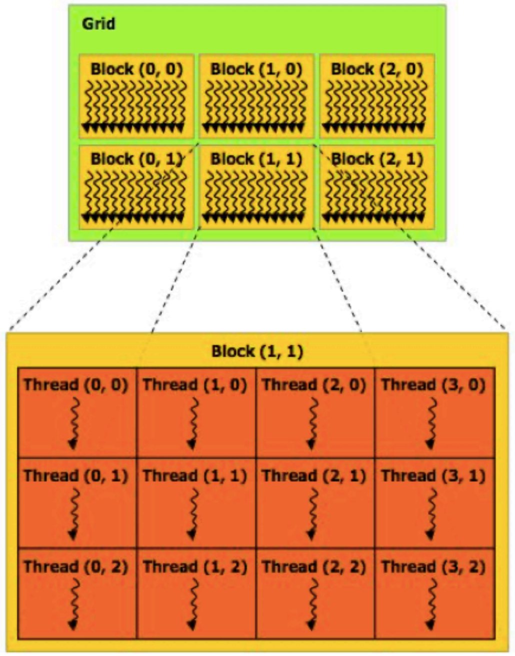
- Nvidia V100 GPU
- 80 SM cores
- 64 FP32 ALUs per SM core
- 5120 FP32 ALUs per board

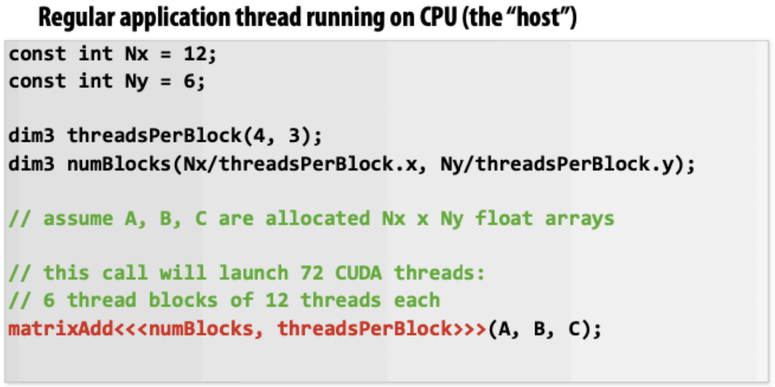


Source: *Standford CS149, Fall 2022*

GPU architecture and programming

- CUDA programming language is used to program Nvidia GPUs



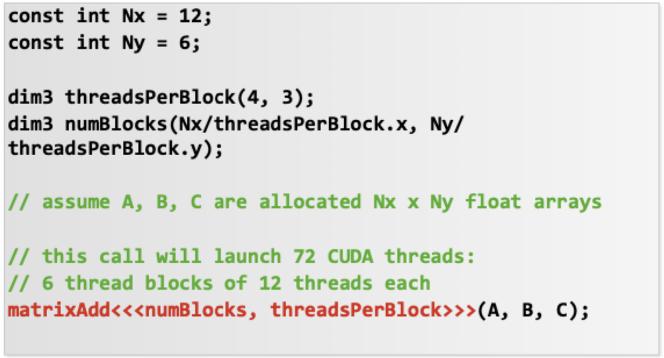

Regular application thread running on CPU (the "host")

```
const int Nx = 12;
const int Ny = 6;

dim3 threadsPerBlock(4, 3);
dim3 numBlocks(Nx/threadsPerBlock.x, Ny/threadsPerBlock.y);

// assume A, B, C are allocated Nx x Ny float arrays

// this call will launch 72 CUDA threads:
// 6 thread blocks of 12 threads each
matrixAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

GPU architecture and programming

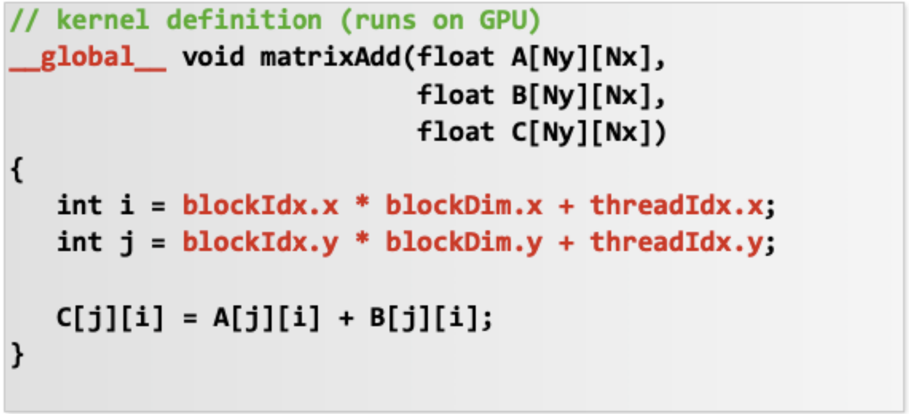
- CUDA programming language is used to program Nvidia GPUs

Regular application thread running on CPU (the "host")



```
const int Nx = 12;
const int Ny = 6;

dim3 threadsPerBlock(4, 3);
dim3 numBlocks(Nx/threadsPerBlock.x, Ny/
threadsPerBlock.y);

// assume A, B, C are allocated Nx x Ny float arrays

// this call will launch 72 CUDA threads:
// 6 thread blocks of 12 threads each
matrixAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

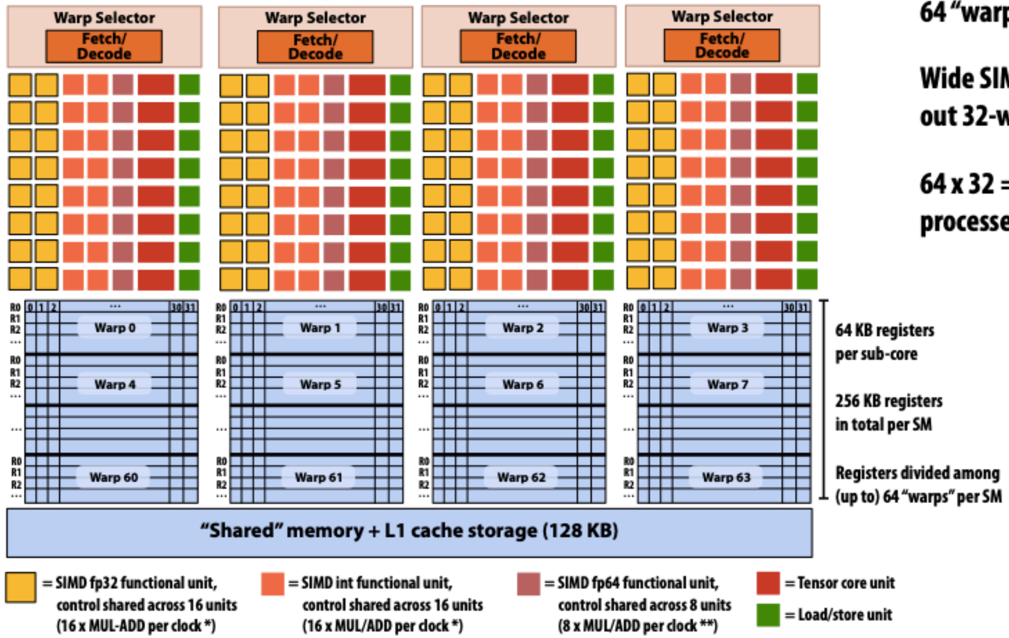
CUDA kernel definition



```
// kernel definition (runs on GPU)
__global__ void matrixAdd(float A[Ny][Nx],
                          float B[Ny][Nx],
                          float C[Ny][Nx])
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;

   C[j][i] = A[j][i] + B[j][i];
}
```

GPU architecture: V100 SM unit and programming

This is one NVIDIA V100 streaming multi-processor (SM) unit



Warp Selector
Fetch/ Decode
R0
R1
R2
0
1
2
30
31
Warp 0
Warp 1
Warp 2
Warp 3
Warp 4
Warp 5
Warp 6
Warp 7
Warp 60
Warp 61
Warp 62
Warp 63
64 KB registers per sub-core
256 KB registers in total per SM
Registers divided among (up to) 64 "warps" per SM
"Shared" memory + L1 cache storage (128 KB)
= SIMD fp32 functional unit, control shared across 16 units (16 x MUL-ADD per clock *)
= SIMD int functional unit, control shared across 16 units (16 x MUL/ADD per clock *)
= SIMD fp64 functional unit, control shared across 8 units (8 x MUL/ADD per clock **)
= Tensor core unit
= Load/store unit

64 "warp" execution contexts per SM

Wide SIMD: 16-wide SIMD ALUs (carry out 32-wide SIMD execute over 2 clocks)

64 x 32 = up to 2048 data items processed concurrently per "SM" core

Source: Standford CS149, Fall 2022

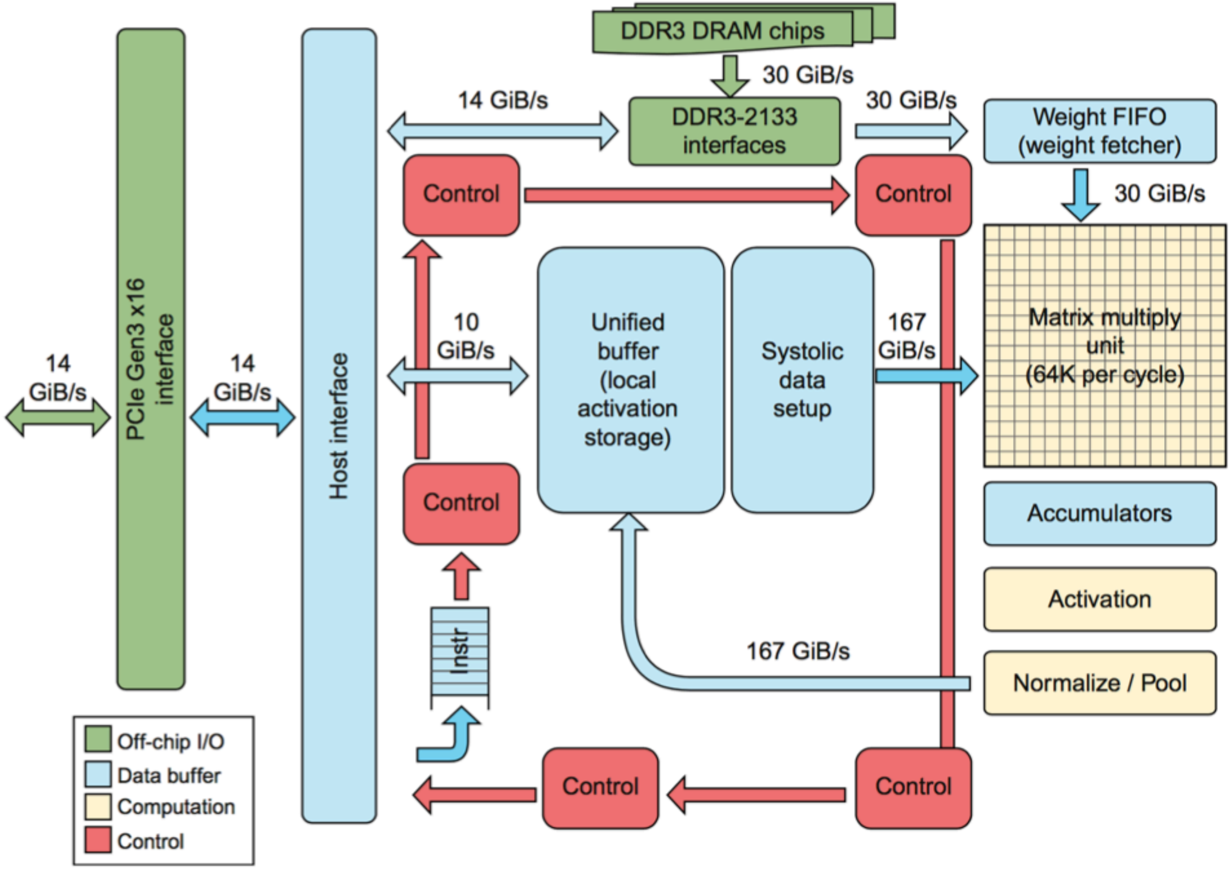
Tensor processing unit (TPU)

- ❑ Google’s ASIC for deep neural networks
- ❑ Specifically designed for inference
- ❑ Is programmed using Google’s DSL named TensorFlow

Key idea:

- ❑ Dedicated modules for matrix-matrix multiplication (matmul), and other functions
  - ❑ 256 x 256 MAC units
- ❑ Scratchpad memories

Tensor processing unit (TPU)



TPU’s ISA

- TPU follows the CISC type, typically 10-20 cycles per instruction (CPI)

TPU’s ISA

- TPU follows the CISC type, typically 10-20 cycles per instruction (CPI)
- Reduced overhead, i.e., no program counter, branch instructions etc.

TPU’s ISA

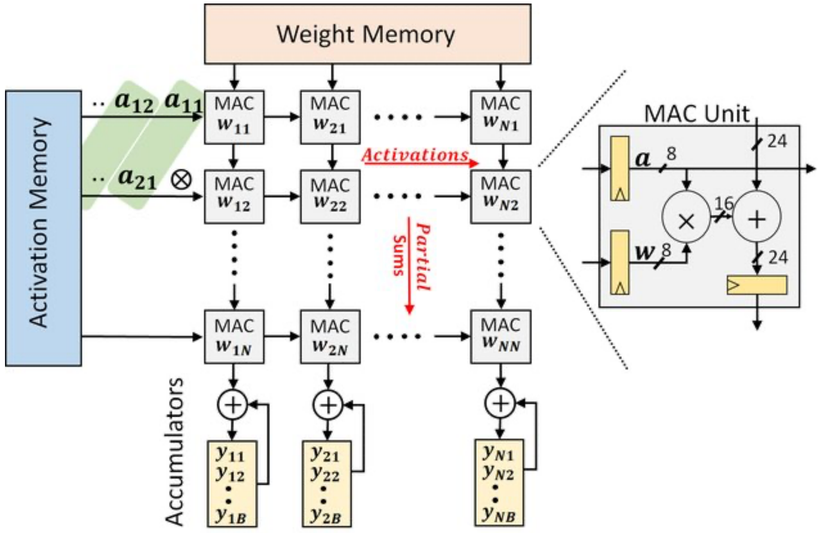
- TPU follows the CISC type, typically 10-20 cycles per instruction (CPI)
- Reduced overhead, i.e., no program counter, branch instructions etc.
- A handful (around a dozen) of instructions in total

TPU's ISA

- TPU follows the CISC type, typically 10-20 cycles per instruction (CPI)
- Reduced overhead, i.e., no program counter, branch instructions etc.
- A handful (around a dozen) of instructions in total
- Some key instructions are:
  - Read_Host_Memory: CPU memory → Unified Buffer (UB)
  - Read_Weights: Weight memory → Weight FIFO
  - MatrixMatrixMultiply/Convolve: Perform MM, MV etc; input: UB, output: accumulator
  - Activate: Performs non-linear activations, e.g., ReLU; input: accumulator, output: UB
  - Write_Host_Memory: UB → CPU memory

TPU's microarchitecture & summary

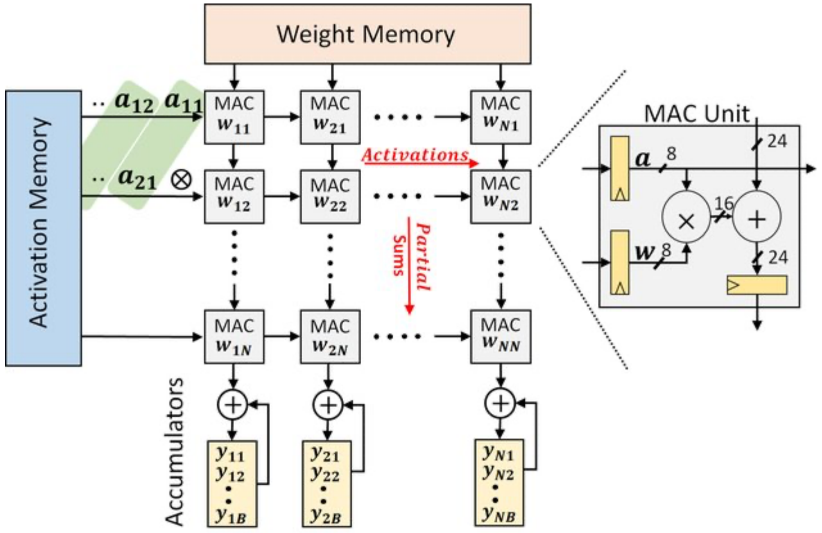
- TPU is based on systolic arrays
  - More details:
    Kung, "Why systolic architectures?, Computers, 1982



TPU's microarchitecture & summary

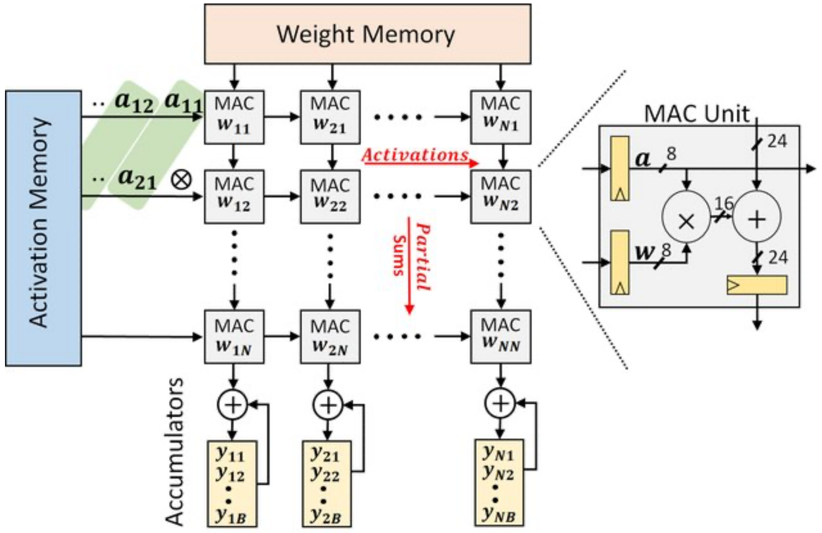
- TPU is based on systolic arrays
  - More details:
    Kung, "Why systolic architectures?, Computers, 1982

- Uses dedicated memories
  - Dedicated buffers, accumulators



TPU's microarchitecture & summary

- TPU is based on systolic arrays
  - More details:
    Kung, "Why systolic architectures?, Computers, 1982

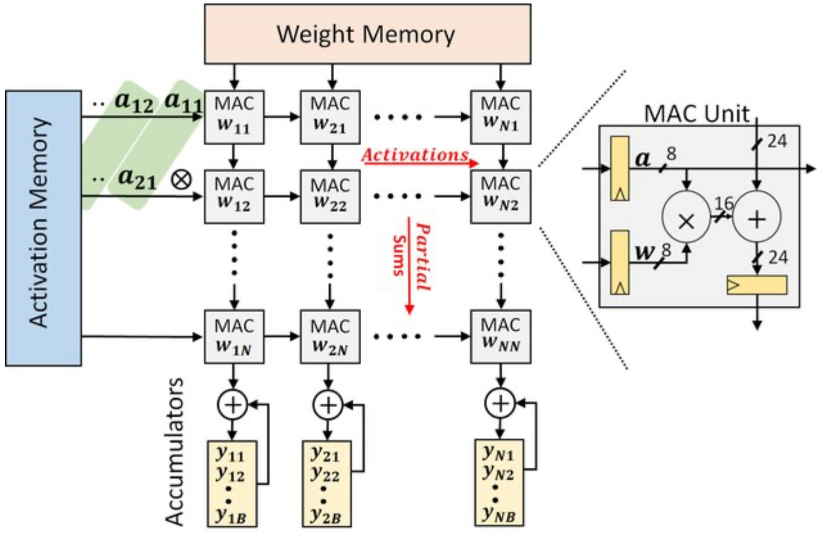


- Uses dedicated memories
  - Dedicated buffers, accumulators

- Invest resources in arithmetic units and memories (way more resources than a server-class CPU)

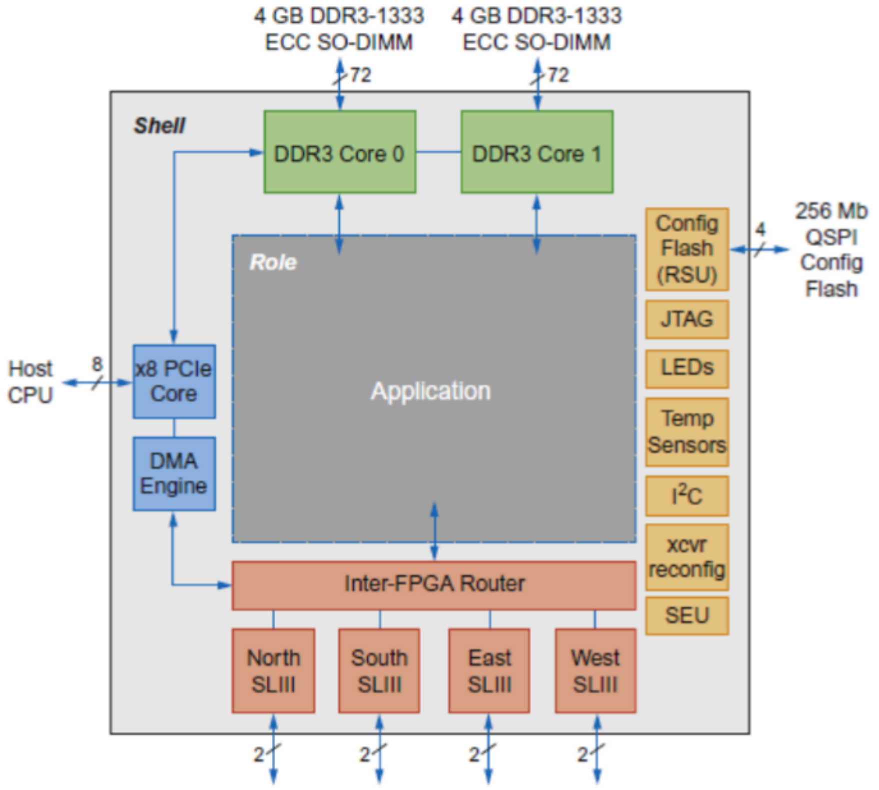
TPU's microarchitecture & summary

- TPU is based on systolic arrays
  - More details:
    Kung, "Why systolic architectures?, Computers, 1982
- Uses dedicated memories
  - Dedicated buffers, accumulators
- Invest resources in arithmetic units and memories (way more resources than a server-class CPU)
- Easiest form of parallelism (SIMD, systolic arrays), and specialization (8b data)



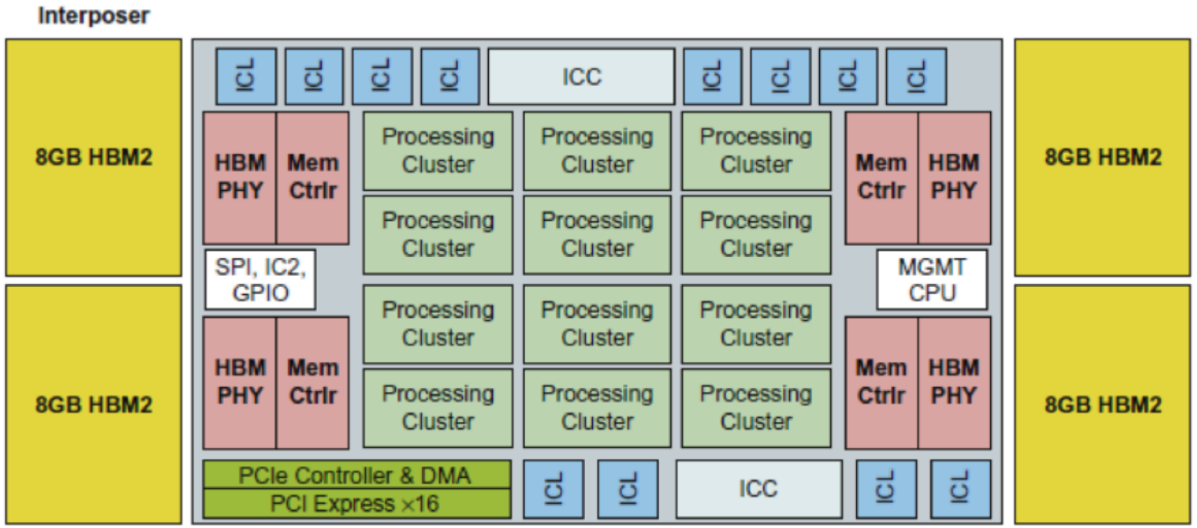
Microsoft's Catapult and Brainwave

- FPGA based DSA for AI inference (edge as well as cloud)
- Catapult: 32MB of Flash memory, PCIe connectivity
- Brainwave: Has dedicated deep-learning processing units (DPUs) soft-cores



Intel's Crest

- Specifically designed for DNNs training
- 16-bit Fixed Point
- Multiple matmul units, operating on 32 x 32 size matrices
- Employs HBM + SRAM



Apple’s neural processing units (NPUs) and neural engine

- Your assignment to read on it and answer the following:
  - How is the micro-architecture?
  - How is it programmed?
  - What kind of parallelism is it using?
  - How does it compare to Google’s TPU in the above aspects, and/or otherwise?

Thank you!

asif.ali@uetpeshawar.edu.pk