AI Accelerators 2

26 July 2025 09:51

What are Artificial Intelligence (AI) Accelerators?

An AI accelerator is high-performance specialized hardware that is optimized for AI workloads such as neural networks, machine learning, and other data-intensive or sensor-driven processes.

The three main types are

1. Central Processing Unit (CPU)
2. Graphics Processing Unit (GPU)
3. Field-Programmable Gate Arrays (FPGA)/Application-Specific Integrated Circuit (ASIC)

Where are they used

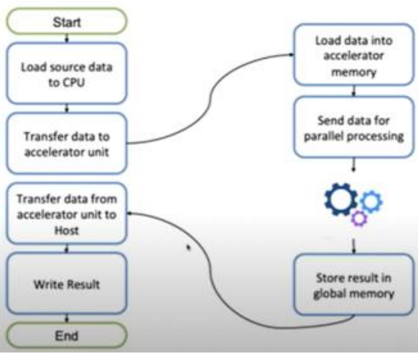
We can divide AI Accelerators into two groups (based on where we use them):

- Data centres
- Edge Devices

Data Centres

Edge

How do they work



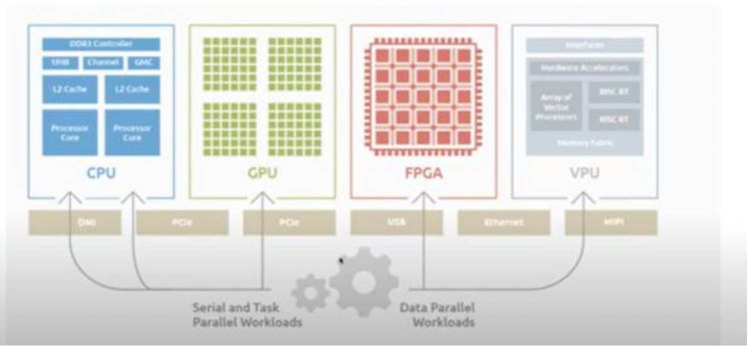
→ Ai accelerators should work in co-processing unit.

→ # this is how we develop program which could be run on AI accelerators.

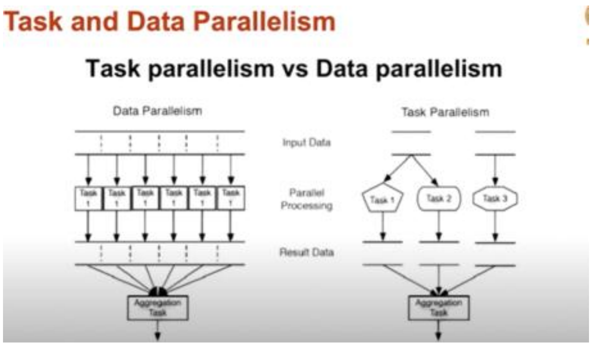
→ host – CPU
device – GPU

Not everything is to be offloaded on GPU.

One View of AI Accelerators



• Only compute intensive tasks which require Parallelism must be offloaded on GPUs

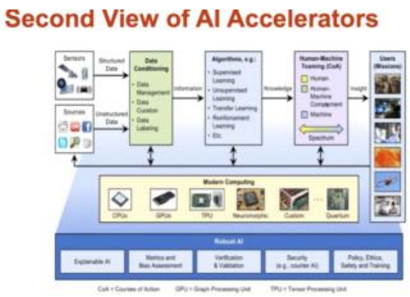


• TASK vs DATA Parallelism

Data Parallelism :- data is split & given to various tasks. } data level decomposition

Task Parallelism :- ↳ small section of data ↳ split that Particular data which is unique for each Particular task. } data & task level decomposition

(not necessary task 'll use same data not tasks 'll do same work.)



diff. level of complexities available in these accelerators can be used as diff. levels of Programming applications.

Gpus – For the Data Center   For the Edge

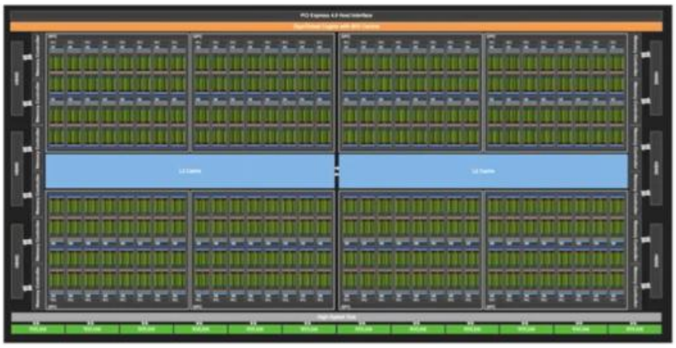


Figure 6. GA100 Full GPU with 128 SMs (A100 Tensor Core GPU has 108 SMs)

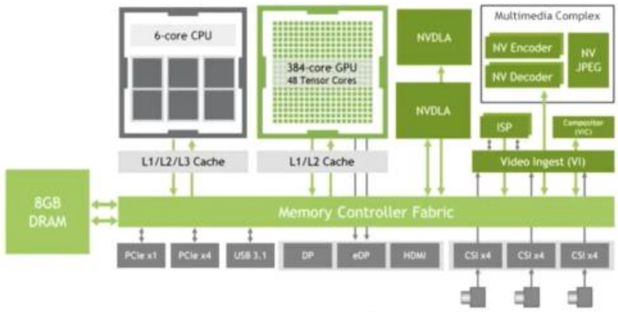


Figure 2: Block diagram of Jetson Xavier NX processor engines including high-speed I/O and memory fabric.

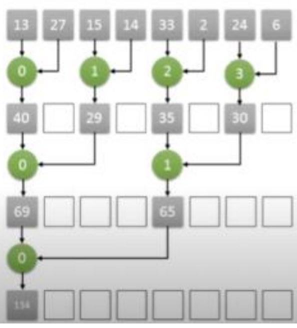
CPU vs GPU Processing

- Sequential reduction : ((((((13+27)+15)+14)+33)+2)+24)+6)

- Parallel reduction : ((13+27)+(15+14))+((33+2)+(24+6))

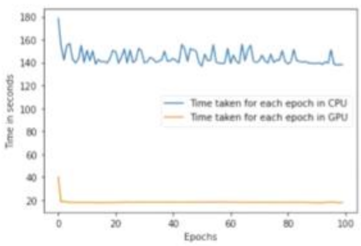
How does parallel reduction happens ?

- Parallel Reduction with a GPU



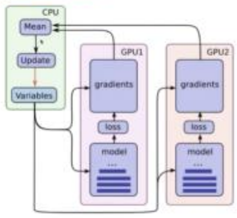
- Assuming N as the number of the elements in an array, we start N/2 threads, one thread for every two elements
- Each thread computes the sum of the corresponding two elements, storing the result at the position of the first one.
- Iteratively, each step:
  - the number of threads halved (for example, starting with 4, then 2, then 1)
  - doubles the step size between the corresponding two elements (starting with 1, then 2, then 4)
- after some iterations, the reduction result will be stored in the first element of the array.

CPU vs. GPU Processing – Training time (Classification Problem)



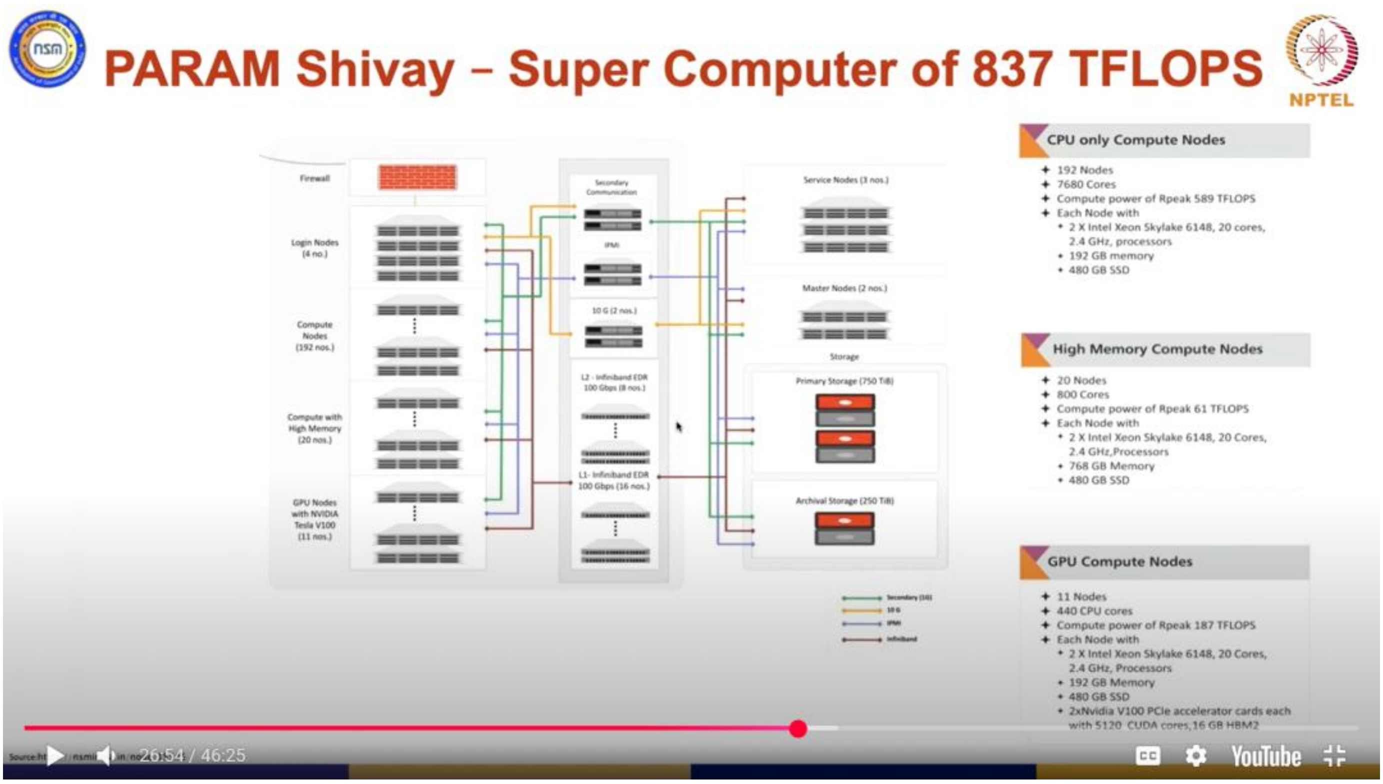
No of Epochs	Time taken for Epochs in CPU (sec)	Time taken for Epochs in GPU(sec)
1	154.26	18.21
2	142.15	18.84
3	154.84	17.92
4	156.78	17.85
5	142.98	17.77
6	139.54	17.69
7	143.02	17.58
8	154.93	17.72
9	139.99	17.76
10	150.66	17.65
11	141.59	17.72
12	150.30	17.65
13	138.58	17.51
14	142.64	17.62
15	140.51	17.52
16	141.09	17.65
17	139.49	17.61
18	143.42	17.61
...	...	...
79	141.08	17.75
80	138.68	17.77
81	140.72	17.88
82	151.33	17.78
83	141.76	17.74
84	141.02	17.80
85	140.29	17.80
86	140.89	17.74
87	139.63	17.73
88	139.36	17.53
89	139.15	17.56
90	139.25	17.41
91	139.62	17.44
92	138.27	17.44
93	140.55	17.72
94	139.36	17.83
95	151.21	17.97
96	138.70	18.07
97	137.66	17.57
98	138.35	17.51
99	138.08	17.61

Train a convolutional neural network on multiple GPU with TensorFlow.

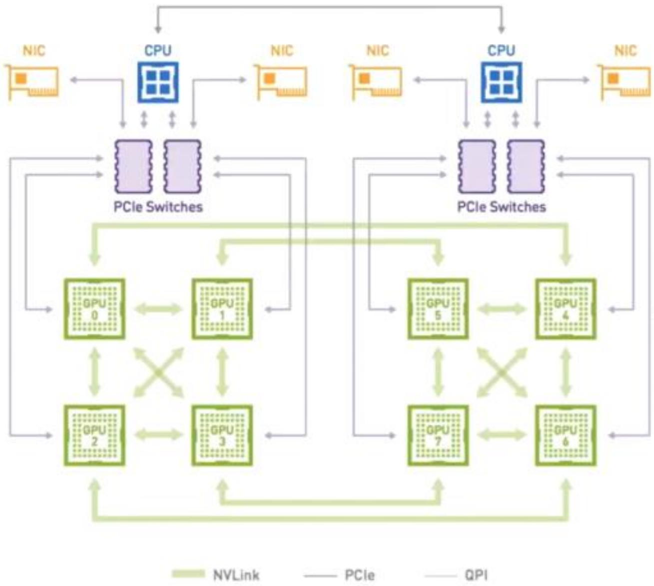


Not all tasks/applications are suitable to work good on gpu they have to be massively parallel computing reuirements
Cpu does certain specific portion which is not effectively done by gpu , as gpu is compute hungry type of processing units
You cannot depend on this data intensive and i/o intensive applications to be effectively run on gpu.

There can be single gpu/ muilti gpu or gpu clusters type of setup



DGX – I



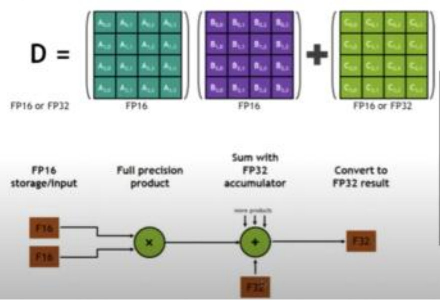
GPUs	8xTesla V100
GPU Memory	256 GB (32 GB/GPU)
CPU	Dual 20-core Intel Xeon E5-2698 v4 2.2 GHz
NVIDIA CUDA Cores	40,960
NVIDIA Tensor Cores (on V100 based systems)	5,120
System Memory	512 GB 2.133 GHz DDR4 RDIMM
Storage	4x1.92 TB SSD RAID-0
Network	Dual 10 GbE

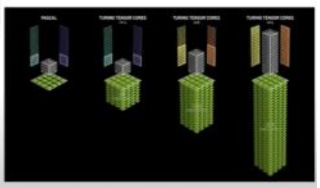
Dgx 1 server

- Has Nvlink propritery bus link specially developed by nvdia

DGX – I – Tensor Cores

- The Tesla V100 GPU contains 640 Tensor Cores: 8 per SM (Streaming Multiprocessor).
- Each Tensor Core provides a 4x4x4 matrix processing array which performs the operation D = A * B + C, where A, B, C and D are 4x4 matrices. The matrix multiply inputs A and B are FP16 matrices, while the accumulation matrices C and D may be FP16 or FP32 matrices.



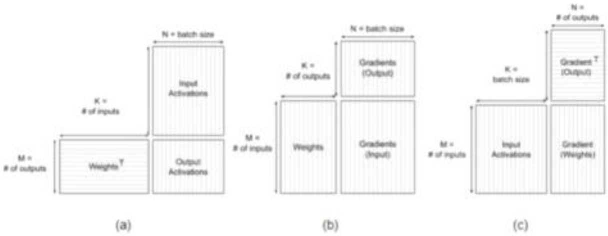


Why is tensor cores very important

DGX – I – Tensor Cores

- One arithmetic operation that holds high importance is matrix multiplication.
- Multiplying two 4×4 matrices involves 64 multiplications and 48 additions.
- Convolution and Multiplication are the areas where the new cores shine.
- Typically, the notion is that CUDA cores are slower, but offer more significant precision. Whereas a Tensor cores are lightning fast, however lose some precision along the way.
- All Tensor core does is that it accelerates the speed of matrix multiplication.
- Tensor Cores are able to multiply two fp16 matrices 4x4 and add the multiplication product fp32 matrix (size: 4x4) to accumulator (that is also fp32 4x4 matrix).
- *General Matrix Multiplication (GEMM)*
- Instead of needing to use many CUDA cores and more clocks to accomplish the same task it can be done in a single clock cycle causing a dramatic speed up in machine learning applications.

DGX – I – Tensor Cores



Equivalent matrix multiplies for (a) forward propagation, (b) activation gradient calculation, and (c) weight gradient calculation of a fully-connected layer.

- Tensor Cores in CUDA Libraries : cuBLAS uses Tensor Cores to speed up GEMM computations (GEMM is the BLAS term for a matrix-matrix multiplication); cuDNN uses Tensor Cores to speed up both convolutions and recurrent neural networks (RNNs)
- CUDA – WMMA (Warp Matrix Multiply-Accumulate)

Source NVIDIA

DGX – I – Tensor Cores

- DLSS -- Deep Learning Super Sampling
- Basic premise is simple: render a frame at low-ish resolution and when finished, increase the resolution of the end result so that it matches the native screen dimensions of the monitor (e.g. render at 1080p, then resize it to 1400p). That way you get the performance benefit of processing fewer pixels, but still get a nice looking image on the screen.

- Tensor core vs. Gpu cores performance

```
chpc@LAPTOP-TDCSOMDA:~/wmma_tensorcore_sample/project/project$ ./kernel
[*] Initializing Matrix...
[+]   A: 4096 x 4096
[+]   B: 4096 x 4096
[+]   C: 4096 x 4096
[*] Computing D = A * B + C on GPU without Tensor Cores...
[+] GPU(without Tensor Cores) Elapsed Time: 1409.558716 ms
[+] TFLOPS: 0.10
[*] Computing D = A * B + C on GPU with Tensor Cores...
[+] GPU(with Tensor Cores) Elapsed Time: 1386.487549 ms
[+] TFLOPS: 0.10
```