Processor Design for ML/AI

CSN-221 CAM Autumn 2020-21: Project 1

Shreyas Dodamani
19114079

What are Processors?

Processors are operational units (circuits) that run the basic instructions inside a computer.

The primary functions of a processor are fetching, decoding, executing and writing values to memory in each instruction cycle. The two main parts of a processor are the Control Unit (CU) and the Arithmetic and Logic Unit (ALU). The ALU performs the mathematical calculations while the CU controls the flow of data and instructions within the computer.

The Central Processing Units (CPUs) that we are accustomed to in our everyday lives contain general purpose processors. For example, on our laptops, we may watch movies, browse the internet, edit files, or even perform simple arithmetic calculations that are required in our day to day lives. These calculations take meagre amounts of processing power to execute.

The technical term for one set of ALU, CU and its registers is a *core.* Instructions are queued for execution, which happens one by one in a single core. First, the Control Unit *fetches* the current instruction with the help of the program counter (which is the pointer to the memory location of the current instruction) and *decodes* the instruction. The ALU then *executes* the instruction.

To understand the working of the processor, let's use a simple example - suppose you want to multiply the numbers 2 and 4 and store the result 8 in the memory. In very crude terms, the processor would go through the following steps:

```
Step 1 (CU): Load the first number from data memory into ALU input A
Step 2 (CU): Load the second number from data memory into ALU input B
Step 3(ALU): Multiply the values at the two ALU inputs and store it
in the aluResult
Step 4 (CU): Store the value from aluResult in memory location C
```

Why is Machine Learning Heavy Duty?

To recap from the above introduction of neural networks, machine learning algorithms largely consist of linear algebraic computations, namely, matrix multiplication, addition, transpose, etc.

Let us understand the steps taken to multiply two matrices of dimensions $N \times N$.

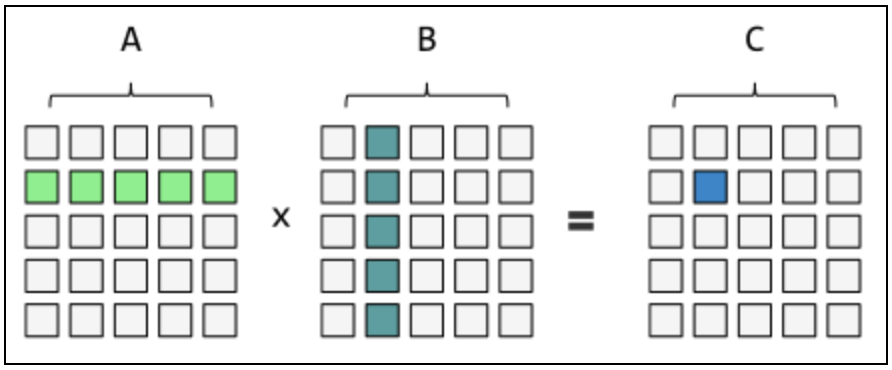


Figure : Matrix Multiplication

$$C_{i,j} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

To calculate one element of the resultant matrix, we must multiply each element of one row of matrix A by each element of one row of matrix B, and then take the sum of the individual products. Hence, to calculate $C_{i,j}$, we require an $O(n)$ operation.

There are n^2 elements in matrix C, hence, we must execute many $O(n)$ operations.

The final time complexity for computing one matrix product is $O(n^3)$, which is very expensive in terms of processing time. As the applications of neural networks become more complex, a general purpose CPU will be unable to handle such loads, and will either take ages to execute them, or crash.

How can we decrease the execution time?

You might have noticed a pattern in the sequence of operations of matrix multiplication. To calculate one element of the resultant matrix, you follow the following steps:

1. Multiply n pairs of numbers a_{ik} and b_{kj} .
2. Add the n resultant numbers to get $C_{i,j}$.

This specific operation is known as a **Multiply-Accumulate Operation** (**MAC**). Keep this in mind, as the specifications of some vendors' processors will be described with the help of this term later on.

For each element you would execute the same operation of addition n times and there are n^2 such elements.

Furthermore, each computation is independent of each other, so we don't necessarily need to compute them one after the other.

By having multiple processors - say m processors - we can split the workload into m different streams, which will get executed simultaneously. This would effectively reduce the processing time by m times, giving us our results faster.

This concept is termed as **parallel processing**. A sort of divide and conquer mechanism - we divide the large problem into smaller chunks, solve each chunk *simultaneously,* and combine the results at the end.

Another relevant term is **throughput**. It refers to how much data can be processed per unit time. The aim of parallel processors (and eventually processors designed for machine learning) is to improve throughput. The performance of a machine learning algorithm is directly proportional to the throughput of the machine that performs the computations. After all, the algorithms only tell the computer what to do, it is up to the computer how it does it.

AI Accelerators

This section elaborates on the developments made on a CPU in order to achieve parallel processing functionalities. AI accelerators are computer devices that have specifically been designed to run machine learning and neural network algorithms on. Many a time, they consist of multiple cores for parallel processing. Different types of AI Accelerators are:

Graphics Processing Unit (GPU)

A GPU is essentially an array of CPUs (or cores). These cores are connected to a main control super-unit which distributes the tasks to each core.

Field-Programmable Gate Array (FPGA)

An FPGA is an integrated circuit which is designed in such a way that it allows the user to configure the logic according to his/her own preference. It contains an array of *programmable logic blocks,* which can be interconnected in ways similar to logic gates, but provide for more complex implementations.

AI dedicated Application-Specific Integrated Circuit (ASIC)

An ASIC is simply an integrated circuit that has been designed for a particular use. An ASIC that has been designed for machine learning algorithms can increase efficiency by up to 10 times.

One thing common in all the accelerators is the presence of multiple processing units for increasing throughput.

Graphical Processing Unit (GPU)

The main difference between a CPU and a GPU is that the CPU is optimised for latency, while the GPU is optimised for throughput. CPU processes tasks in a serialised manner while a GPU can perform them parallelly.

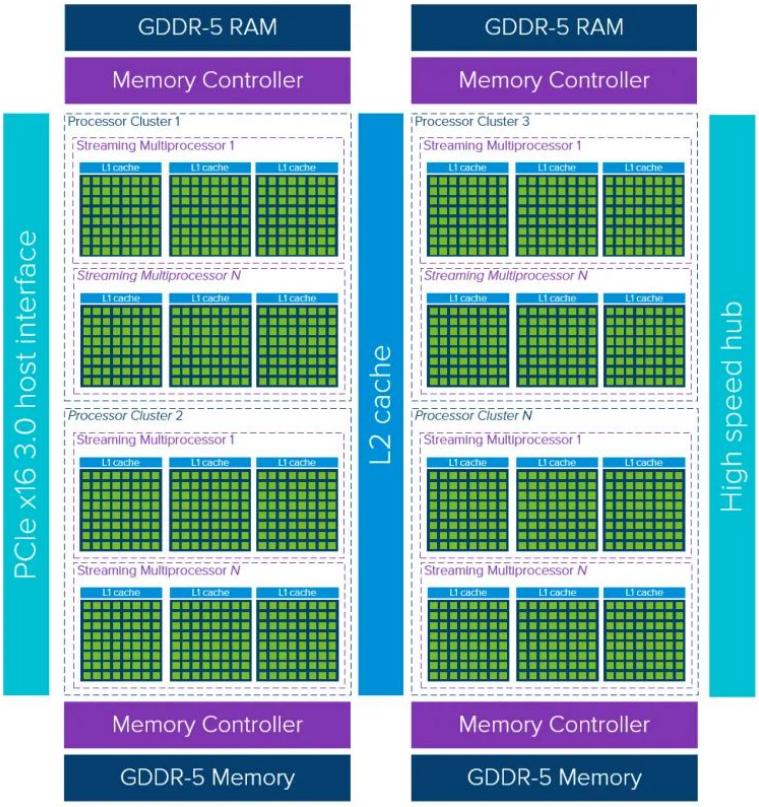


Figure 2 : GPU Architecture

We can see from the diagram that the architecture of a CPU is

very similar to the architecture of a GPU. The GPU consists of multiple Processor Clusters (PC) that contain multiple cores. Each cluster has a dedicated local layer-1 cache (GDDR-5 Memory) and a shared layer-2 cache between all the clusters.

However, unlike a CPU, a GPU works with fewer memory cache layers, because it is not concerned with latency, but only throughput.

FPGA

FPGAs are integrated circuits which are designed to be re-programmable using hardware description language such as Verilog HDL (VHDL). The fact that FPGAs are reprogrammable distinguishes them from ASICs. ASICs, as mentioned above, are integrated circuits that are manufactured to do predefined tasks. Although there are one-time programmable FPGAs, the dominant type are reprogrammable FPGAs.

If you make a mistake while programming the logic for an FPGA, then all you need to do is reprogram it to apply the correct logic. However, if you made a mistake while making the ASIC system, then you would have to rewire it (ie. re-manufacture it).

An FPGA is typically based on a matrix of configurable logic blocks (CLBs). CLBs are made out of 4 basic components:

1. Look up tables
2. Multiplexers
3. Full adder
4. D flip flops.

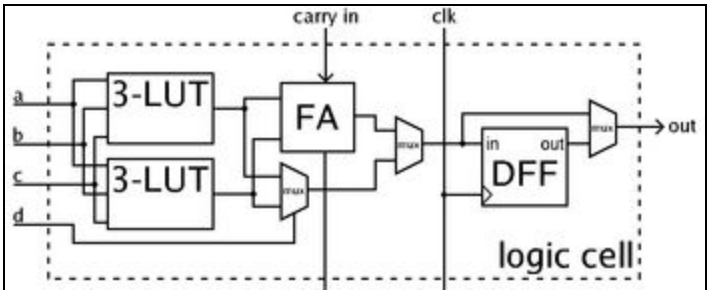


Figure 3 : CLB logic

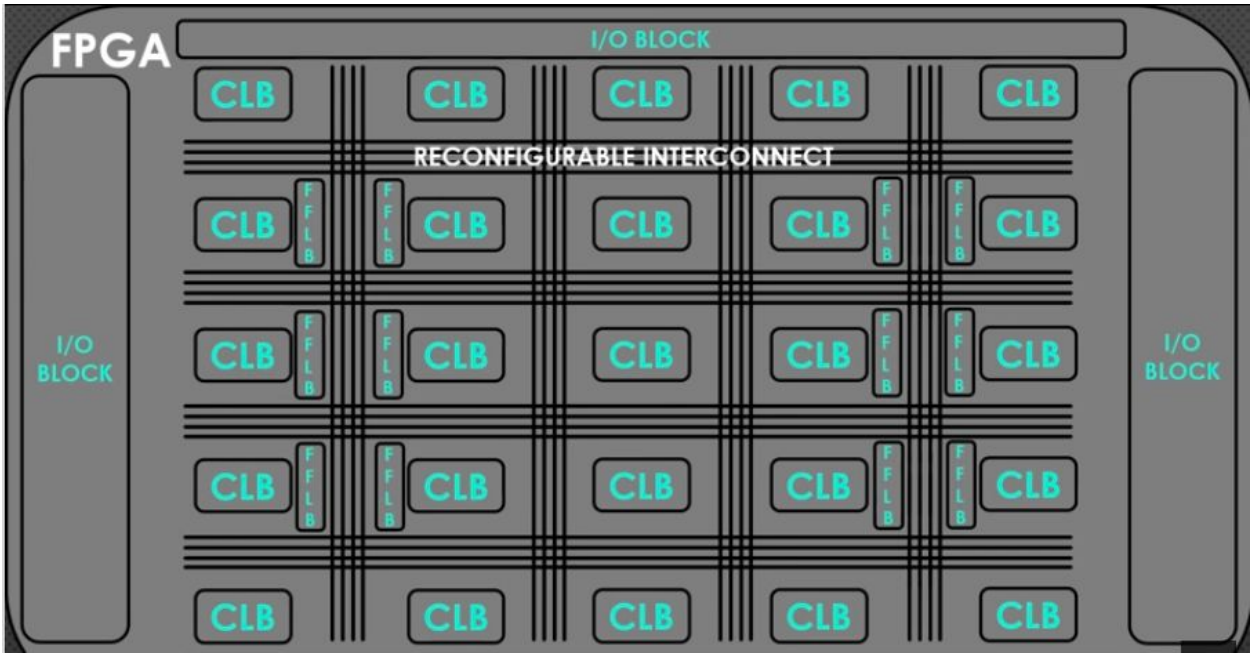


Figure : FPGA architecture

Reconfigurable interconnects allow CLBs to be connected with one another. Fixed functional logic blocks are blocks in the circuit with a predefined function. Block RAM serves as a common memory structure.

Google TPU v3

Tensor Processing Units (TPUs) are Application Specific Integrated Circuits (ASICs) developed by Google. TensorFlow is an open source library that is developed by Google for machine learning and neural networks. The Tensor processor is custom-built to smoothly run TensorFlow code.

Some definitions:

- **High-bandwidth memory (HBM):**

  High-bandwidth memory is standardized stacked memory technology that provides very wide channels for data, both within the stack and between the memory and logic.

- **Matrix Unit (MXU):**

  Matrix Unit is a central processing unit that implements an instruction set containing instructions that operate on two-dimensional arrays of data called *matrices,* compared to the scalar or vector processors, whose instructions operate on single data items (scalar), or a one-dimensional array of data items (vector).

Google has released three different versions of the TPU now. The latest version is TPU v3, which is an improvement over TPU v2. Comparing their architecture specifications:

TPU v2	**TPU v3**
8 GiB of HBM per core	16 GiB of HBM per core
One MXU for each TPU core	Two MXUs for each TPU core
Up to 4 TiB of memory in one TPU Pod	Up to 32 TiB of memory in a TPU Pod

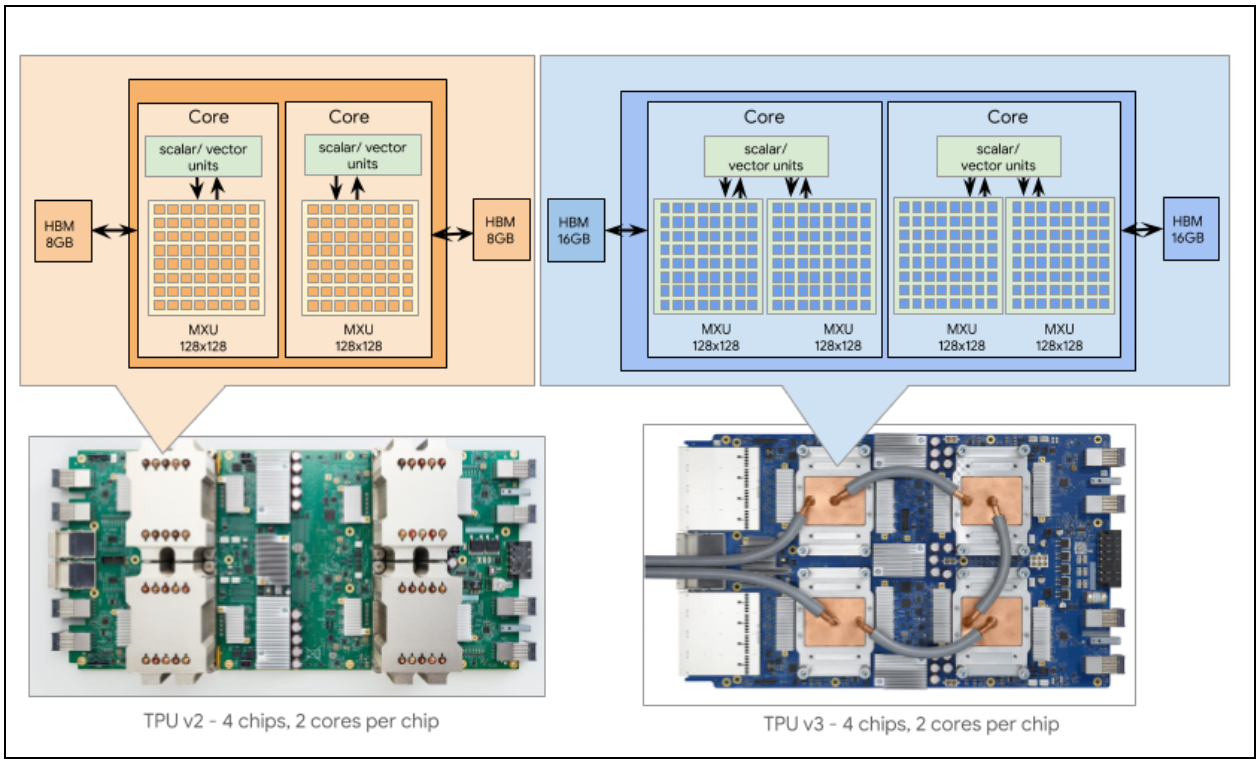


Figure : Google TPU v2 vs v3 comparison

$$1\ GiB\ (Gibibyte)\ =\ 2^{30}\ bytes$$

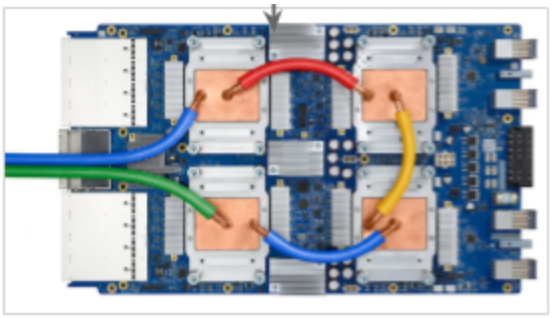
Why is the MXU so important?

As stated above, an MXU executes a matrix of operations at once. Matrix multiplication requires MAC operations in which each element of one array must be multiplied by an el which regular CPUs take $O(n^3)$ to execute.

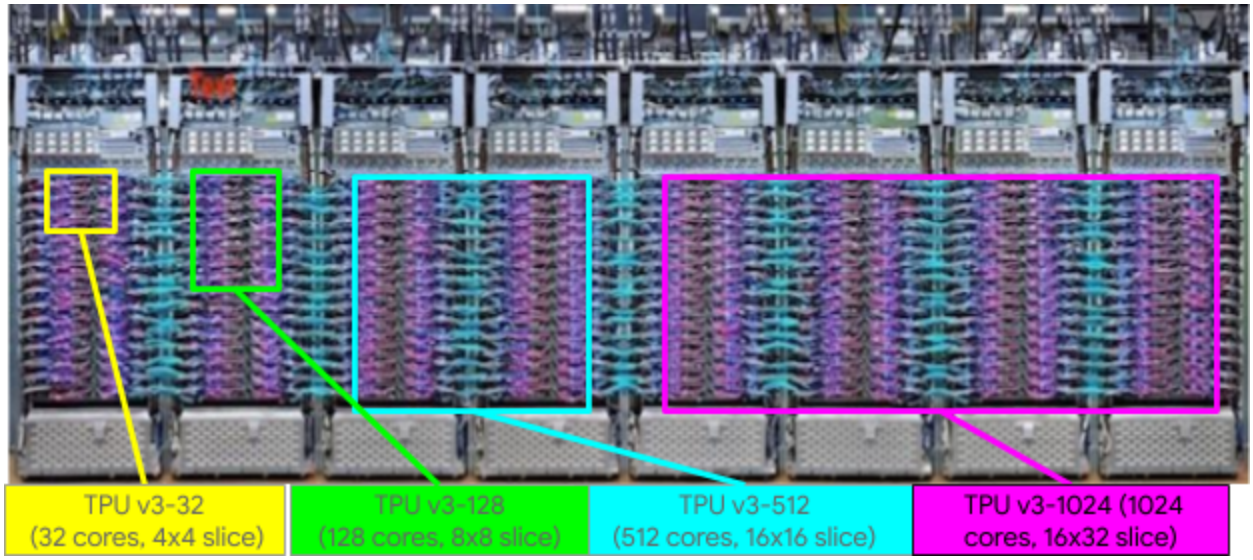
On the other hand, if we use matrix units, it will execute the multiplication of each element in the array in constant time. $O(n^3)$ to $O(n)$.

In the Google TPU, each MXU is capable of performing 16000 such MAC operations in each cycle. While the MXU inputs and outputs are 32-bit floating point values, the MXU performs multiplies at reduced *bfloat16* precision. Bfloat16 is a 16-bit floating point representation that provides better training and model accuracy than the IEEE half-precision representation.

Each TPU core executes computations independently of other TPU cores. However, they can also be interconnected to directly communicate with each other and share a global memory. Google refers to a cluster of interconnected TPU cores as a TPU pod.



Cloud TPU



Cloud TPU Pod

Google TPU v1

TPU versions 2 and 3 being newer versions, Google hasn't released a great deal of information on their architecture. However, we do have access to the inner workings of TPU v1.

The TPU is designed to be an accelerator. It is not the computer itself. We plug it into a host system, and the host will use the TPU to perform operations on the data it provides. The results are returned to the host via the same interface. This allows the main computer to handle the high-level jobs, and not worry about the low level heavy-duty calculations.

Let us understand the block diagrams of the core of the Google TPU:

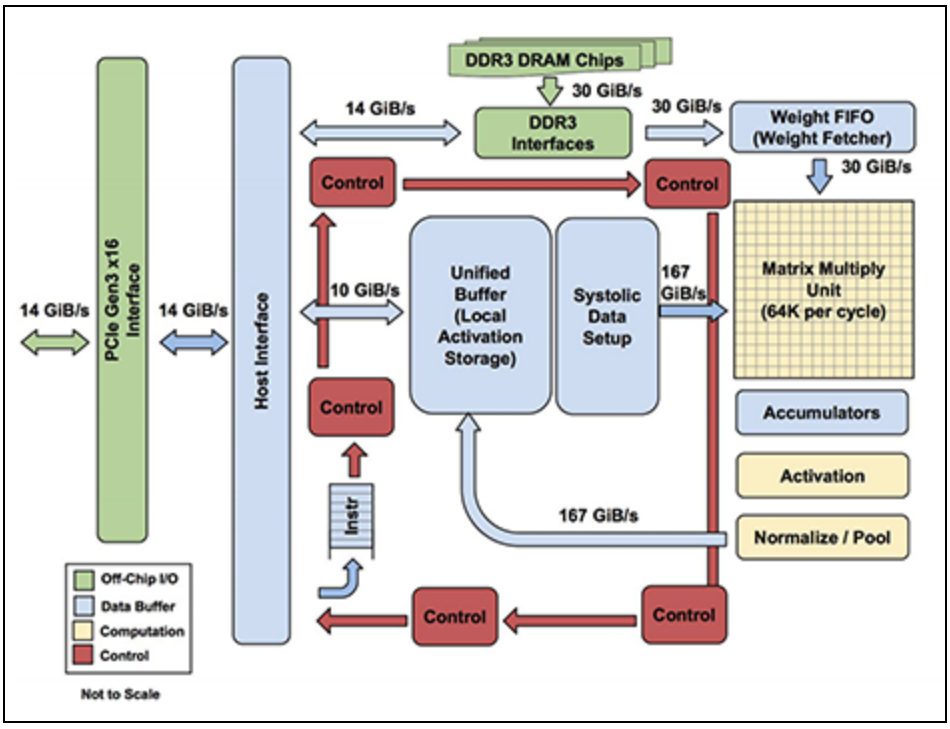


Figure : Google TPU v1 Architecture

The **Host Interface** will connect the TPU to the host machine. A neural network requires calculations of weights and activation values in the forward and backward propagation phases. Data shared through the interface consists of:

- Weights (through DDR3 Interface to DDR3 RAM).
- Activations (to Unified Buffer)
- Control Instructions (to the Control path - shown in red).

In the above list, **DDR3** RAM stands for Double Data Rate Type 3 RAM. It is a type of synchronous dynamic random-access memory with a high-bandwidth interface.

1. While training the model, and predicting outputs, weights are only uploaded once per batch (or layer of nodes, if you're a neural-network pro). Hence, they aren't frequently accessed and can be stored in the DDR3 RAM. When the weights are used in matrix multiplication, they are loaded into the weight fetcher (Weight FIFO). The host computer will have access to write directly to the DDR3, so the next set of weights can then immediately be loaded into the DDR3 RAM.

2. Unlike the weights, the activations are frequently read and updated. The unified buffer is directly connected to the MXU (defined previously), and together, the unified buffer and MXU take up more than half the size of the chip. The buffer is capable of storing up to 384 activation matrices of dimensions 256 by 256. Due to the frequent need to read and write activation values, the buffer and MXU are connected by a bus that can transfer data at the rate of 167 GiB/s.

   Accumulators then collect the results of the MXU, and transfer them to the **Activation Pipeline.** The activation pipeline applies standard neural network functions like ReLU, which aren't as computationally expensive as

matrix multiplication. Finally, these new activation values are written back to the unified buffer.

3. The final broad component of the TPU architecture is the control flow. The red path in the diagram indicates the flow of instructions from the host machine throughout the TPU. Instructions include any operation that you can think of performing, like, weight fetching, activation pipeline operations, MXU multiplication, and so on.

Now that you know how each component on the TPU works, it is time to understand the procedure of execution of the TPU:

1. The TPU is first connected to the host machine, and all buffers and RAMs are empty.
2. The host loads a TPU-compiled model, and pushes weights into the DDR3 RAM.
3. The host then populates the unified buffer with activation values.
4. A control signal is sent to the weight fetcher to load the weights into the weight FIFO unit.
5. Another control signal is sent to execute the MXU matrix multiplication.
6. Output from the MXU unit is fed into the activation pipeline which applies ReLU operations.
7. Steps 4 to 4 are repeated until we reach the last batch, or layer of nodes in the neural network.
8. Activation values of the last layer are sent back to the host machine.

Huawei Ascend

Huawei Ascend is a series of processors designed by Huawei with the purpose of improving performance in artificial intelligence. There are two processors in this suit:

- Ascend 310 AI Processor : Energy efficient and best for edge computing
- Ascend 910 AI Processor : High Performance, best for training ML models

The architectural specifications of each processor are as follows:

Ascend 310	**Ascend 910**
16 TOPS@INT8 8 TOPS@FP16	512 TOPS@INT8 256 TFLOPS@FP16
Power consumption: 8W	Max power consumption: 350W

Technical terms:

```
16 TOPS@INT8     - 16 Tera-operations per second on 8-bit integer values
16 TOPS@FP16     - 16 Tera-operations per second on 16-bit floating point
                   values
512 TOPS@INT8    - 512 Tera-operations per second on 8-bit integer values
256 TFLOPS@FP16 - 256 Tera-floating point operations per second on 16-bit
                   floating point values
```

Both Ascend 310 and Ascend 910 processors use the Huawei-developed Da Vinci architecture, however, only the Ascend 910 chip is an SoC processor.

An **SoC,** or **System-On-Chip** Processor is an integrated circuit that incorporates almost all of the functionalities of a computer onto a single silicon chip.

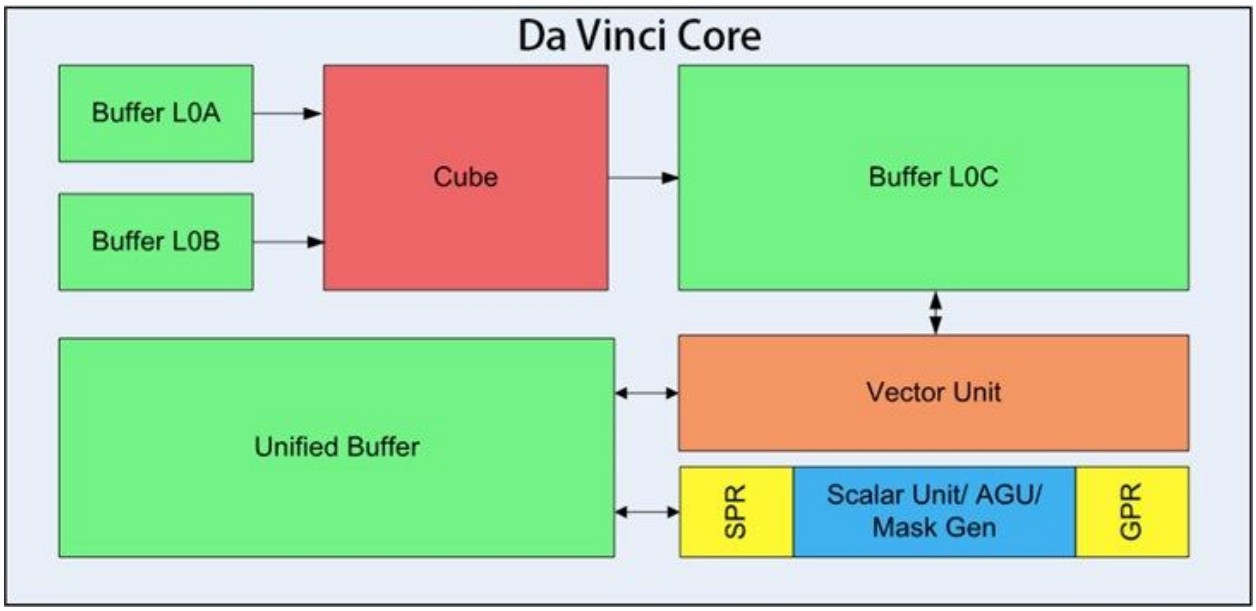
The Da Vinci Architecture

Before delving into the technical aspects of the Da Vinci architecture, please understand some of the AI computing data objects:

Type	Consists of:
Scalar	A single number
Vector	A set of 1-d ordered numbers, each identified by an index
Matrix	A set of 2-d ordered numbers, each identified by an index
Tensor	A set of n-d ordered numbers, each identified by an index

As mentioned previously, the majority (~99%) of AI calculation involves matrix multiplication (MAC Operations). These operations put high demand on the scalar, vector and matrix computing units which in turn are tolling on the hardware.

The core of the Da Vinci architecture is subdivided into multiple units, including a core 3D Cube, Vector unit (for vector calculation), Scalar unit (for scalar calculations), etc. Although the 3D cube is powerful beyond imagination, it is only able to execute matrix multiplication. Hence, the vector and scalar operation units are just as important.



The buffers L0A and L0B are used to store input matrix data, while the buffer L0C is used to store the output matrix data.

To understand how the 3D Cube unit accelerates matrix multiplication, think of it this way:

- To multiply two $N \times N$ matrices, we need to execute N^2 MAC operations.
- If the scalar unit executes it serially, then it will take N^2 cycles.
- If the vector unit executes it, then it will take N cycles.
- If the 3D cube unit executes it serially, then it will take 1 cycle.

For large values of N, we notice a drastic performance gain.

In the Da Vinci architecture, one side of the 3D cube is 16 units long. This means that it can execute $16 \times 16 \times 16$, or 4096 operations in one cycle.

NVIDIA Volta v100

Memory:

L1 Cache

The L1 cache is specific to a single core, having a measured **throughput** (maximum data that can be stored in it in one operation) of 108.3 bytes/cycle by a team of researchers at CITADEL, a high performance computing research firm based in Chicago. However, the theoretical upper bound is 256 bytes/cycle. The theoretical throughput is calculated by multiplying the LSU count per SM by the number of bytes that each LSU can load in one cycle per instruction. The **size** of the L1 data cache is 128 KiB.

L2 Cache

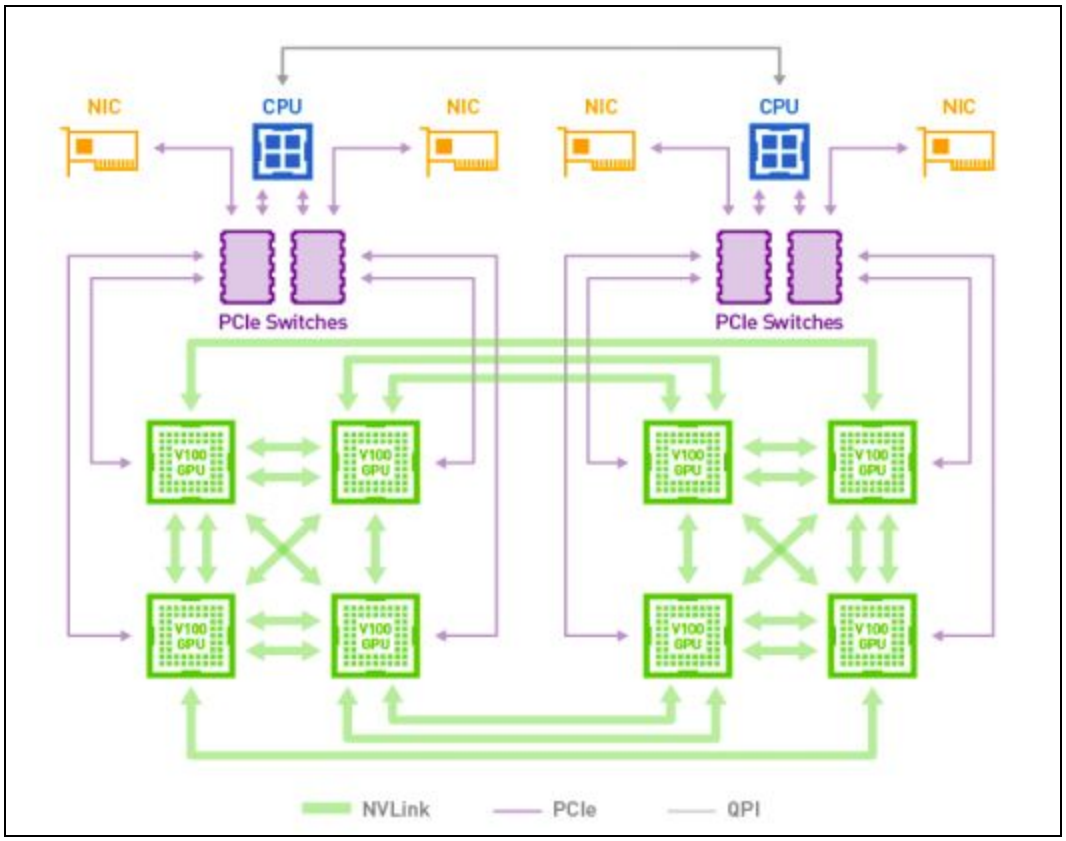
This cache is unified for data, instructions and constant memory. Its **size** is 6144 KiB. It has a cache line of 64 Bytes. A **cache line** is the unit of data transfer between the cache and main memory. The average **latency** of an L2 cache is 193 clock cycles. Here, the latency is defined as the delay before a transfer of data begins following an instruction for its transfer.

Instruction Caches

Instruction caches are temporary storage locations for instructions, to reduce the amount of overhead before retrieving the next sequence of instructions. Unlike data, instructions have to be loaded in each step of program execution. Hence, by storing recently run instructions in a common memory location, we can reuse them in case of loops, etc. This is useful in case of matrix multiplication when we need to perform the same kind of operations repeatedly.

Streaming Multiprocessors (SM)

Streaming Multiprocessor consists of 4 quadrants. Each quadrant can run 32 threads of instructions at the same time. They are feeding into all of the individual functional units. The tensor core is a functional unit just like any other unit.



NVLINK

It is the interconnection between processors (memory bus). Normal interconnects have to packetise data (1kB or 16 kB packets). The Nvidia Volta v100 is designed for memory size requests. You can read and write to neighbouring processors with single word data, and sustain **high throughput**. Volta uses one 128-bit word to encode each instruction, unlike previous NVIDIA processors that used 64-bit words to encode each instruction, and separate 64-bit words to specify the control signals. Where:

- At least 91 bits are used to encode the instruction
- At least 23 bits are used to encode the control signal

Each bus is able to carry 300 gigabytes of information per second.

Comparing Vendors

Chip	Google TPU v1	Huawei Ascend 910	Nvidia Volta v100
TDP (W)	75	350	300
On-Chip RAM (MB)	28	64	21.1
Peak FP32 (TFLOPs)	unknown	unknown	15.7
Peak FP16 (TFLOPs)	23 (INT6)	256	125
Memory bandwidth (GB/s)	30 (DDR3)	1200 (HBM2)	900 (HBM2)
I/O bandwidth (GB/s)	14	115	300

What the terms mean:

- TDP: Thermal Design Power, a measure of how much heat is produced by the processor
- On-Chip RAM: How much RAM is present on the processor
- Memory Bandwidth: Amount of memory that can be accessed in one call
- I/O Bandwidth: Amount of data that can be sent as input and output in one go by the processor

Information References

- https://forum.huawei.com/enterprise/en/huawei-da-vinci-ai-chip-architecture/thread/616780-895
- http://www.hisilicon.com/en/Products/ProductList/Ascend#:~:text=The%20massive%20boost%20in%20power,of%20its%20high%20computing%20power
- https://forum.huawei.com/enterprise/en/ai-chips-depend-on-architecture-innovation/thread/613226-895
- https://www.hotchips.org/hc31/HC31_1.11_Huawei.Davinci.HengLiao_v4.0.pdf
- https://cloud.google.com/tpu/docs/system-architecture
- https://arxiv.org/pdf/1804.06826.pdf
- https://medium.com/@antonpaquin/whats-inside-a-tpu-c013eb51973e
- https://medium.com/x8-the-ai-community/machine-learning-hardware-1dac168423fd
- http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture11.pdf
- https://en.wikipedia.org/wiki/AI_accelerator
- https://www.fonow.com/view/237148.html
- https://arxiv.org/pdf/1704.04760.pdf
- https://www.eetimes.eu/top-10-processors-for-ai-acceleration-at-the-end point/
- https://jameswhanlon.com/new-chips-for-machine-intelligence.html

Image References

- https://www.quora.com/Why-cant-you-multiply-a-2x2-matrix-with-a-3x2-matrix
- https://nielshagoort.com/2019/03/12/exploring-the-gpu-architecture/
- https://en.wikipedia.org/wiki/Field-programmable_gate_array
- https://developer.nvidia.com/blog/inside-volta/#:~:text=With%20640%20Tensor%20cores%2C%20Tesla,world's%20most%20powerful%20computing%20servers.
- https://www.fonow.com/view/237148.html
- https://medium.com/@antonpaquin/whats-inside-a-tpu-c013eb51973e