GAZI UNIVERSITY

FACULTY OF ENGINEERING – COMPUTER ENGINEERING

Cansu AYTEN – 171180010

CENG479 PARALLEL COMPUTER ARCHITECTURES AND PROGRAMMING

CURRENT CUDA ARCHITECTURES

JUNE 2022

CONTENTS

Page

1. GPU

The Graphics processing unit (GPU) is an electronic circuit that creates images for a display device. This circuit is customized to quickly manipulate memory to quickly create images. Processing images is difficult. Therefore, complex mathematical calculations are required when processing images. Parallelism is required for these calculations to be performed correctly. For example, if we consider a video game and assume that it is played at 60 frames per second, the formation of these frames requires the creation of shapes, colors, movements, and lighting in a very short time. The CPU may be weak for all of these processes. It does not have an architecture that can perform these operations. Therefore, a GPU is needed to perform these operations. In addition, GPUs render flawless, high-quality graphics in computer games. High-performance GPUs are used for 4K and high refresh rate technology. Apart from this, artificial intelligence is also used in machine learning fields to perform some operations quickly. For example, it can play a role in training neural networks in deep learning. Neural networks require computations for each node. Developers have realized that they can use GPUs to take advantage of more parallelism for this situation. Because the GPU provides more parallelism than the CPU. Thus, an increase in performance can be achieved. It takes advantage of the GPU's parallel processing technology to render graphics faster when editing videos from video editors. Although the GPU has become more programmable today, it is used in many areas, but it is generally responsible for creating 2D and 3D images, animations, and videos [1,2].

1.1 GPU vs CPU

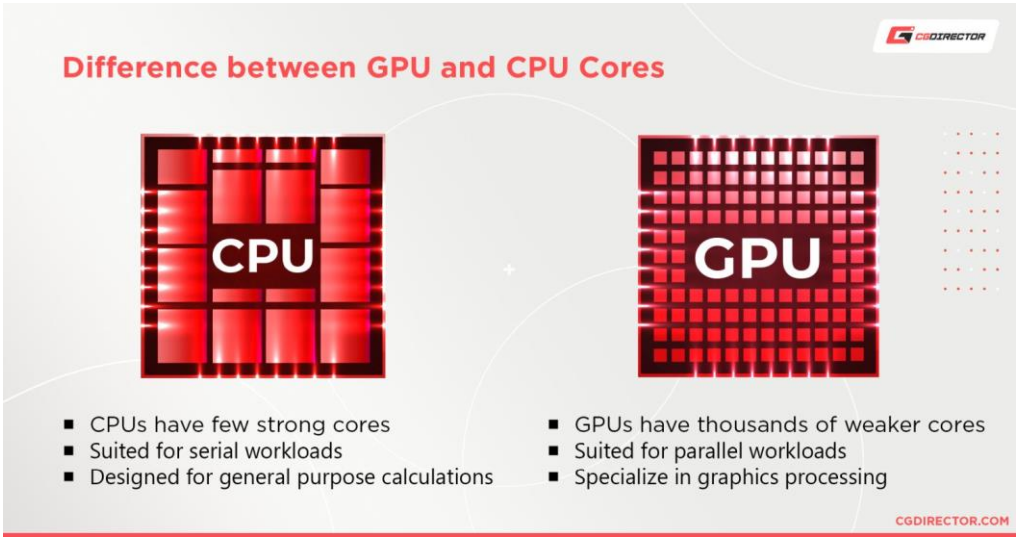

Figure 1: CPU vs GPU [3]

GPUs and CPUs are architecturally and functionally similar. For example, they contain components such as kernel and memory. In general, CPUs enable the computer to work by performing functions such as processing instructions, providing data flow, and collecting data, which are necessary for programs to run on the computer. Benefits can include flexibility and durability, cost and availability, access to memory, and contextual power. It is cost-effective and easy to find. Apart from this, designs are made so different CPUs can be found on motherboards. In terms of memory access, more complex calculations can be made with a larger linear instruction set, as it contains a local cache. But they are not capable of parallel processing. In cases where parallel processing is required, the data processing capacity of the CPU becomes full because the CPU does not have this capability [1,2].

The need for a GPU is because the CPU is underpowered by the development of graphics-intensive programs and applications, and therefore its performance is low. Unlike CPUs, GPUs work with a parallel processing method. In this method, multiple processors perform different parts of the same task. Thus, it can manage multiple tasks with multiple cores. These cores are weaker than CPU cores [1,2].

GPUs, on the other hand, are used to quickly process or render high-quality images or videos. While CPUs are designed for task parallelism, GPUs are designed for data parallelism and performing the same instruction on multiple data. Single instruction multiple data (SIMD) can be mentioned here. SIMD is the execution of a single instruction simultaneously on many data points in parallel. From this point of view, it can be said that GPUs are designed for SIMD. Since it can perform multiple calculations at the same time, it is faster in image processing than CPUs. With multi-core processors, multiple CPUs are combined on one chip. Thus, parallel computations can be performed. Also, because CPUs have higher clock speeds, they can perform a single computation faster than GPUs. GPUs are more expensive than CPUs in terms of cost [1,2].

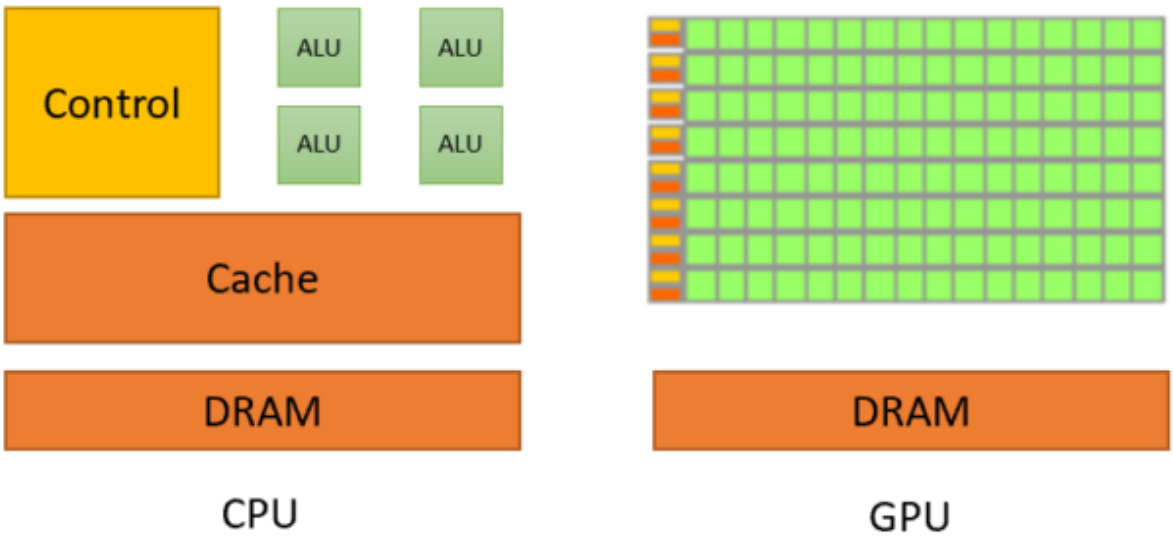

Figure 2: CPU and GPU architecture [4]

Scientific advances require more and more information and processes every day. This creates the need for computational power. In this regard, GPUs have been developed since CPUs will be weak and work slowly. Thus, parallelism was achieved. Figure 2 shows the main differences between CPU and GPU Architectures. In the GPU architecture, most parts are dedicated to data processing. It has more arithmetic logic units (ALUs) than CPUs. ALU is a digital circuit that performs arithmetic and logic operations. Unlike GPUs, in CPUs, some areas other than ALUs need to be divided into units such as control units and caches [4].

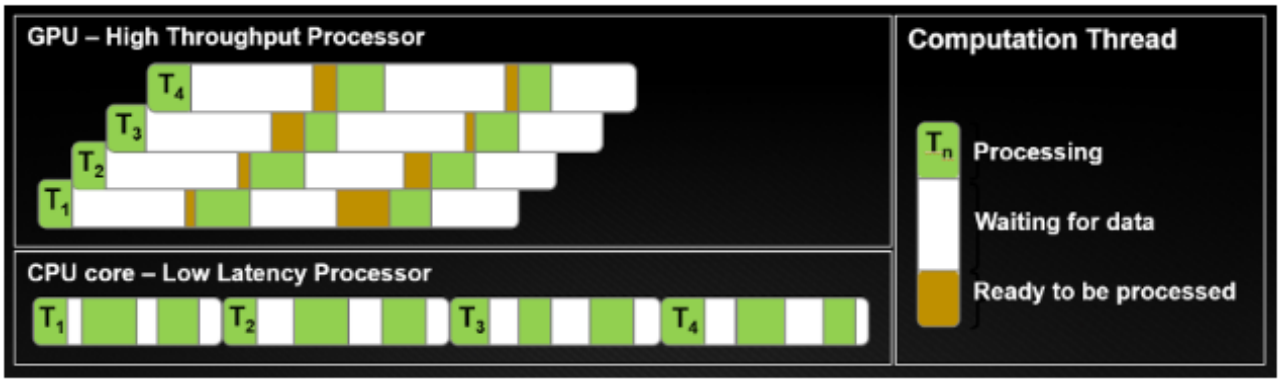

Figure 3: Low latency or high throughput. [4]

When Figure 3 is examined, while memory delays are minimized in each thread in CPUs, these delays are tried to be reduced by calculation in GPUs. The white parts of the image represent memory delays. The green parts are the parts where the work is done. CPUs try to do as many tasks as possible in a given time, minimizing memory latency. In doing so, it needs large caches and complex controllers that require low latency. GPUs, on the other hand, turn off instruction

and memory latency by computation. Looking at Figure 3, when T1 thread passes to the data waiting part, T2 thread starts to process. T3 continues to run when T2 goes into standby. While these are happening, it becomes ready to obtain and reprocess T1 data. In this way, the delays are tried to be hidden with simultaneous thread execution. In other words, it can be concluded that many threads should run on the GPU [4].

2. CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) that allows the software to use GPUs for general computations, thus contributing to performance. This technology from NVIDIA provides access to the GPU, accelerating parallelizable computations with GPU power. This is achieved by directly accessing the GPU's virtual instruction set and parallel computing elements. CUDA enables applications written using programming languages such as C, C++, Python, Fortran, and MATLAB to run using GPU power. Apart from that, it supports third-party units for accessing languages such as Perl, Java, Ruby, Lua, Common Lisp, Haskell, R, IDL, and Julia and Mathematica. NVIDIA's CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools, and the CUDA runtime, which enable rapid application development by harnessing the power of the GPU. It also includes many documents for developers, such as code examples, programming and user manuals, API references, as well as supporting deep learning, linear algebra, signal processing, and parallel algorithms. CUDA is a platform that can be used on all NVIDIA GPUs and is capable of deploying or scaling GPU configurations across all. Many applications developed using CUDA are deployed to GPUs in the cloud. For example, many programs such as Adobe programs, AUTODESK programs, Ansys, MathWorks, and Wolfram Mathematica have been developed using CUDA, which is one of the most performing solutions for processes such as high graphics requirements, graphic design, and numerical analysis [5,6].

There are also different APIs such as OpenCL that allow taking advantage of GPU resources and power. OpenCL, a competitor to CUDA, was released in 2009 by Apple and the Khronos Group. Although OpenCL was seen as a good solution for general computing, not limited to Intel/AMD CPUs with NVIDIA GPUs, NVIDIA GPUs did not perform with OpenCL as not well as CUDA. Apart from that, there are GPU models produced by NVIDIA's competitor, AMD. However, the combination of CUDA and NVIDIA GPUs is preferred in many areas. It also forms the basis of the world's fastest computers. CUDA makes it easier for developers to

use GPU resources instead of more skilled APIs like Direct3D or OpenGL. CUDA-supported GPUs also support frameworks such as OpenMP, OpenACC, OpenGL Compute Shader, C++ AMP, DirectCompute, and OpenCL [5,6,7].

The first GPUs were designed as graphics accelerators. In the 90s, GPUs were made programmable. In 1999, NVIDIA's first GPU was produced. They started using the high performance of this GPU for general-purpose computing. In 2003, a group of researchers led by Ian Buck worked to develop the C programming language with data-parallel structures. As a result of this study, the first programming model, Brook, was announced. Later, Ian Buck joined NVIDIA and released CUDA in 2006 for general GPU-based computing. It was compatible with Microsoft Windows and Linux. Mac OS X support was added with version 2.0 released in 2008. It works with all NVIDIA GPUs from the G8x series, including the GeForce, Quadro, and Tesla series. Apart from that, it is also compatible with most operating systems. Since the day it was developed, the CUDA platform has reached the present day by constantly improving itself on issues such as services and software development tools. It accelerates applications by using it in many fields such as image processing, deep learning, machine learning, computational finance, climate, weather, ocean modeling, data science and analytics, defense and intelligence, manufacturing/AEC, Media and entertainment, safety and security [5,6].

2.1 CUDA in Deep Learning

Deep learning requires great computing speed. For example, in 2016, the Google Brain and Google Translate teams did hundreds of TensorFlow runs in a week using GPUs to train Google Translate models. For this, they bought 2000 GPUs from NVIDIA. Using GPUs, they prevented the training process from taking months, allowing it to be completed within a week. Google used TPU (Tensor Processing Unit) for the production and distribution of Tensorflow translation models. TPU, on the other hand, is an integrated circuit specific to the AI accelerator application developed by Google, which will be used especially for neural network machine learning. Google started using TPUs internally in 2015. In 2018, it both added to the cloud infrastructure and made a small version of the chip available for third parties. [7,8].

Apart from Tensorflow, most frameworks use CUDA for GPU support, including Caffe2, CNTK, Databricks, H2O.ai, Keras, MXNet, PyTorch, Theano, and Torch. Deep learning space requires advanced next-generation GPUs such as the V100, which is 3x faster than the P100

due to the training workload. In order to take advantage of the GPU power, one or more libraries should be used, and libraries suitable for the algorithm to be used should be used. [7].

Figure 4: [7]

3. HISTORY OF CUDA

Thanks to the ability to get more performance with GPUs, the popularity of GPUs has increased day by day. Exploring the potential of GPUs and the high performance they provide has made them an important choice for computing. There were attempts to develop chip-scale parallel processors in the '90s, but limited transistor budgets only allowed for advanced single-core designs. Truly GPU computing actually started with non-programmable 3G-graphics accelerators. From the 1980s, many companies developed multi-chip 3D rendering engines, but the basic elements were only able to be combined on a single chip by the mid-1990s. It evolved from simple pixel drawing functions with these chips from 1994 to 2001 to implement the full 3D pipeline, transforms, lighting, rasterization, texturing, depth testing, and display functions. In 2001, NVIDIA released GeForce 3. GeForce 3 is the third generation of NVIDIA GeForce GPUs. In this generation, programmable pixel shading was possible, but the programmability of the chip was limited. Added separate programmable engines for vertex and geometry shading to make it more flexible and faster. Thus, the GeForce architecture has improved, increasing its efficiency. Later, GeForce 8800 was introduced by NVIDIA. This design included “unified shader architecture”. In this design, each shader core could be assigned to any task. Thus, an increase in performance could be achieved. With this design, GPUs became generalized computing tools. Although GPUs were originally designed for use in image processing, they can be programmed to meet the needs of other areas with the processing power they contain. That's why general-purpose GPU (GPGPU) programming was developed. The purpose of the

development of GPGPU is to perform non-graphical operations on graphics-optimized architectures. With GPGPU Acceleration, the parts that require intense computation in an application can be performed on the GPU, while the general-purpose calculations can be performed on the CPU, and very fast calculations can be made with supercomputation level parallelism. The GPU is then used for very complex calculations, while the CPU is used for sequential calculations in parallel. In short, it can be reached that it promises great hope, as higher performance is achieved by the combination of CPU and GPU processing power. It can now be said that all modern GPUs are GPGPUs. The CUDA model for GPGPU can provide application acceleration in many areas such as GPGPU AI, computational science, image processing, numerical analytics, and deep learning. At the time, GPGPU was thought to be the forerunner of technologies such as CUDA, OpenCL, and DirectCompute [9,10,11].

NVIDIA introduced the Tesla architecture in 2006. In 2007, CUDA, the first C-based development environment for GPUs, was introduced by NVIDIA. With CUDA, a more effective and easy programming model was provided after GPGPU. In 2010, the Fermi architecture instead of Tesla, was introduced by NVIDIA. Then, in 2012, NVIDIA introduced the Kepler architecture. Then, in 2014, the Maxwell architecture was introduced by NVIDIA. In 2016, Pascal architecture was introduced by NVIDIA instead of Maxwell [10].

4. CUDA-SUPPORTED GPU ARCHITECTURES

4.1 Tesla

In 2006, the Tesla GPU architecture was released by NVIDIA. The Tesla series is named after Nikola Tesla. Tesla is the codename for GPU microarchitecture. It is the first microarchitecture to implement the Unified shader model. Used with GeForce 8 Series, GeForce 9 Series, GeForce 100 Series, GeForce 200 Series, and GeForce 300 Series GPUs. It replaced the old fixed-pipeline microarchitectures when the GeForce 7 series was introduced. Figure 5 shows a block diagram of a GeForce 8800 GPU. There are 128 SP, and 16 SMs in eight independent processing units called texture/processor clusters (TPCs). The Tesla series starts with drivers optimized for the PCI Express port and GPU computing. With Tesla, developers began to treat the GPU as a multi-core processor [10,12,13].

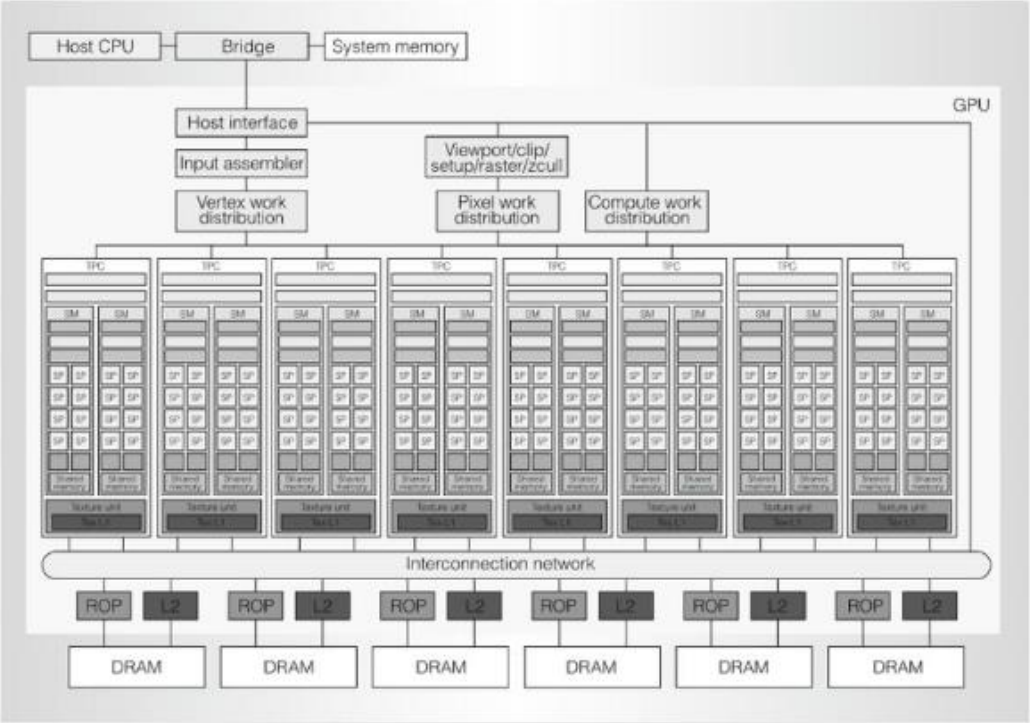

Figure 5: Tesla Architecture [15]

4.2 Fermi

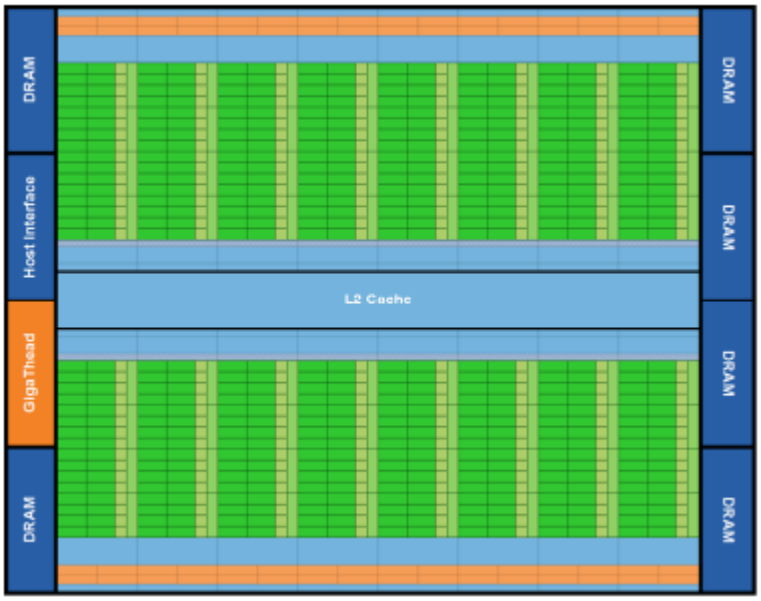

Figure 6: NVIDIA's Fermi GPU architecture consists of multiple streaming multiprocessors (SMs) [10]

In 2010, the Fermi architecture, which was offered by NVIDIA instead of Tesla, was released. The Fermi series is named after the Italian physicist Enrico Fermi. Fermi is the codename for the GPU microarchitecture. This microarchitecture is used in the GeForce 400 series and GeForce 500 series. Later, it was used in GeForce 600 series, GeForce 700 series, and GeForce 800 series with Kepler architecture. The Fermi architecture is managed by a programming

model that allows developers to focus on the design of the algorithm rather than the details of how the algorithm matches the hardware. In this way, the productivity of the developers' increases. In the CUDA software platform and the OpenCL framework, the computational elements of algorithms are called cores. An application or library function can contain more than one core. These kernels can be written in C to express parallelism directly rather than loops. There are additional keywords in C language to express it directly. After compilation, cores consist of many threads executing the same program in parallel. Threads can be thought of here as units that perform the looping operation. For example, while an image processing algorithm has a thread working on a pixel, there can be multiple threads working together – the kernel – on an entire image [10,15].

In Figure 6, NVIDIA's Fermi architecture, which includes multiple streaming multiprocessors (SM) consisting of 32 cores each, is given. each SM; It provides support for L2 cache, host interface, GigaThread scheduler, and multiple DRAM interfaces. Each Fermi Streaming Multiprocessor (SM) 32 CUDA cores –to perform floating-point and integer operations-, load/store units (LD/ST units) –to address memory operations for 16 threads per hour-, and four special Contains function units (SFU) -to execute transcendental mathematical instructions, a memory hierarchy, and warp schedulers-. It contains 64K local SRAM. It is split between cache and local memory [16].

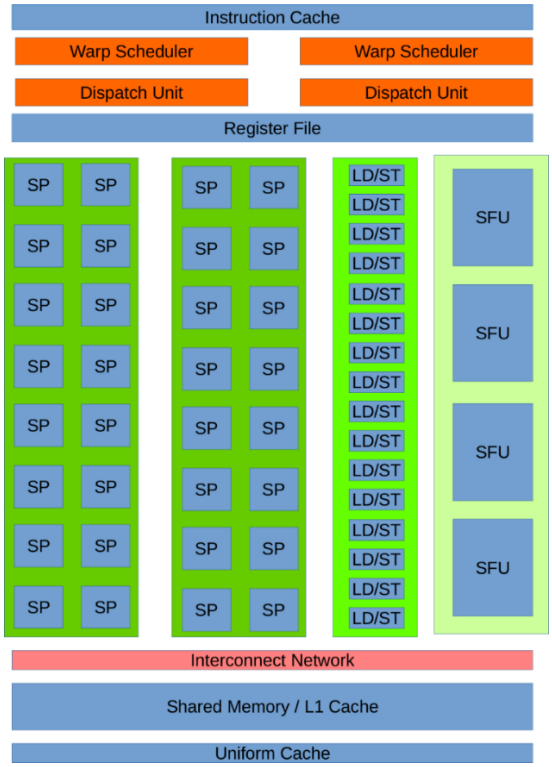

Figure 7: Fermi Architecture [16]

- Fermi cards provided support from CUDA 3.2 to CUDA 8. Removed in CUDA 9. Completely dropped in CUDA 10 [17].

4.3 Kepler

In 2012, the Kepler architecture, which was offered by NVIDIA instead of Fermi, was released. The Kepler series is named after Johannes Kepler. Kepler is the codename for the GPU microarchitecture. Most GeForce 600 series, most GeForce 700 series, and some GeForce 800M series GPUs were based on Kepler. This architecture is NVIDIA's first microarchitecture focused on energy efficiency. With Kepler, multiple CPU cores are enabled to start working on a GPU core at the same time. It consists of 15 SMs and six 64-bit memory controllers. Each SM contains 192 single-precision CUDA cores, 64 double-precision units, 32 SFUs, 32 LD/ST units, and 16 texture units [16,18].

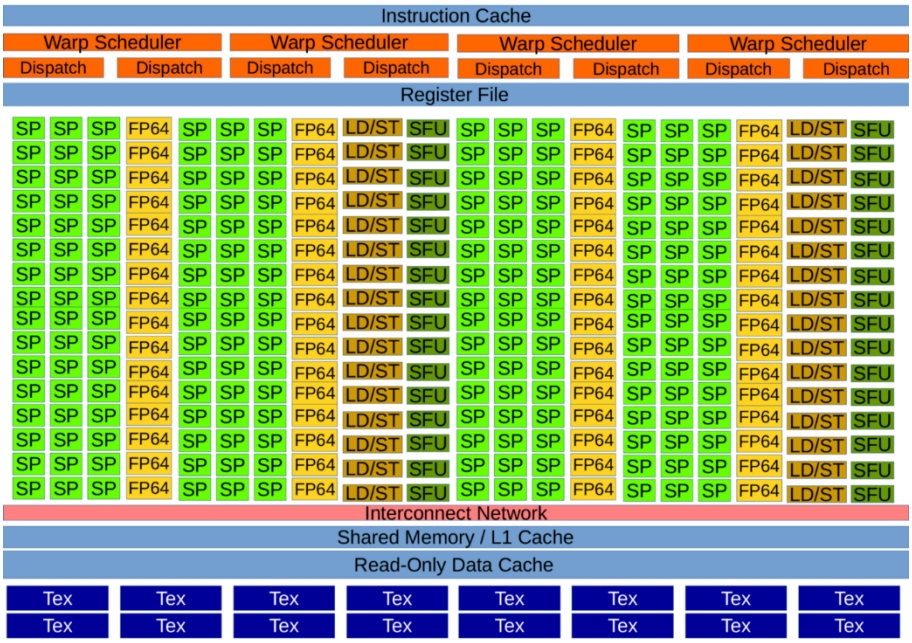

Figure 8: Kepler Architecture [16]

- Kepler cards provided support from CUDA 5 to CUDA 10. Removed in CUDA 11 [17].
- Can add support such as unified memory programming or dynamic parallelism [17].

4.4 Maxwell

In 2014, the Maxwell architecture was released by NVIDIA, replacing Kepler. The Maxwell series is named after James Clerk Maxwell. Maxwell is the codename for the GPU microarchitecture. This architecture was introduced in later models of the GeForce 700 series. In addition, SM design and CUDA Compute Capability 5.2, which aims to increase power efficiency, were introduced with this architecture. This architecture consists of up to 16 SMs

and four memory controllers. Each SM has been reconfigured to improve performance. each SM; is divided into a block of four 32 CUDA cores. Each of these consists of eight SFU and 8 LD/ST units. [16,19].

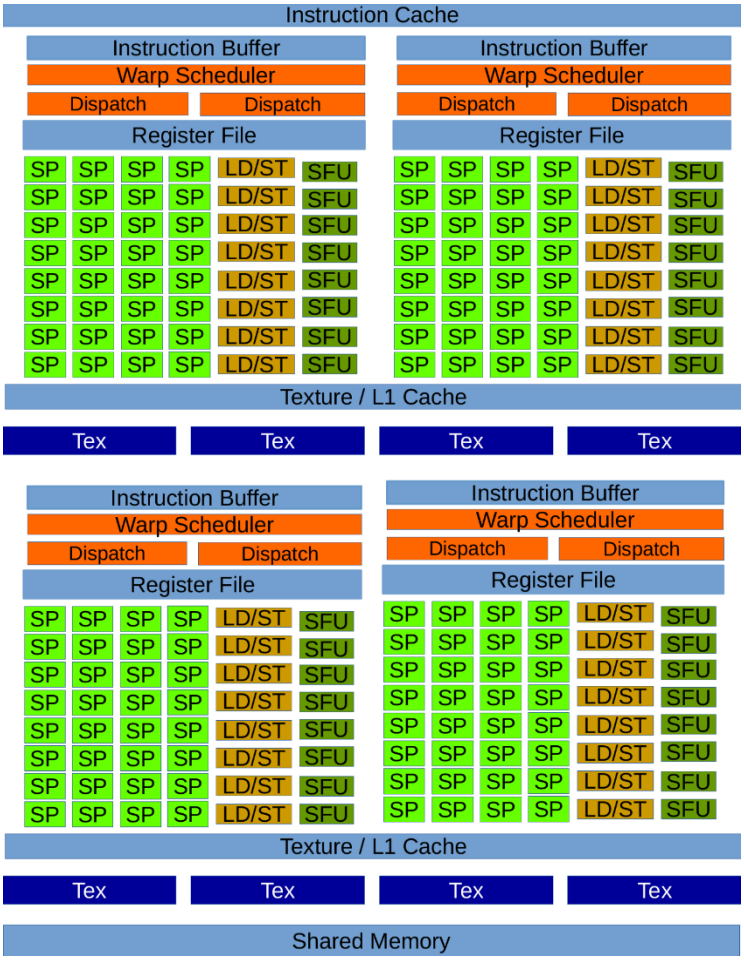

Figure 9: Maxwell Architecture [16]

- Maxwell cards were supported from CUDA 6 to CUDA 11. Removed as of CUDA 11 [17].

4.5 Pascal

In 2016, the Pascal architecture, which was presented by NVIDIA instead of Maxwell, was released. The Pascal series is named after the French mathematician and physicist Blaise Pascal. Pascal is the codename for the GPU microarchitecture. The architecture is primarily used in the GeForce 10 series, starting with the GeForce GTX 1080 and GTX 1070. There are up to 60 SMs and eight 512-bit memory controllers in the Pascal architecture. Each SM contains 64 CUDA cores and four texture units. It has the same number of registers as Kepler but provides more registers because it has more SMs. With the doubling of the amount of shared memory,

the code began to be executed more efficiently. CUDA cores can handle both 16- and 32-bit instructions [16,20].

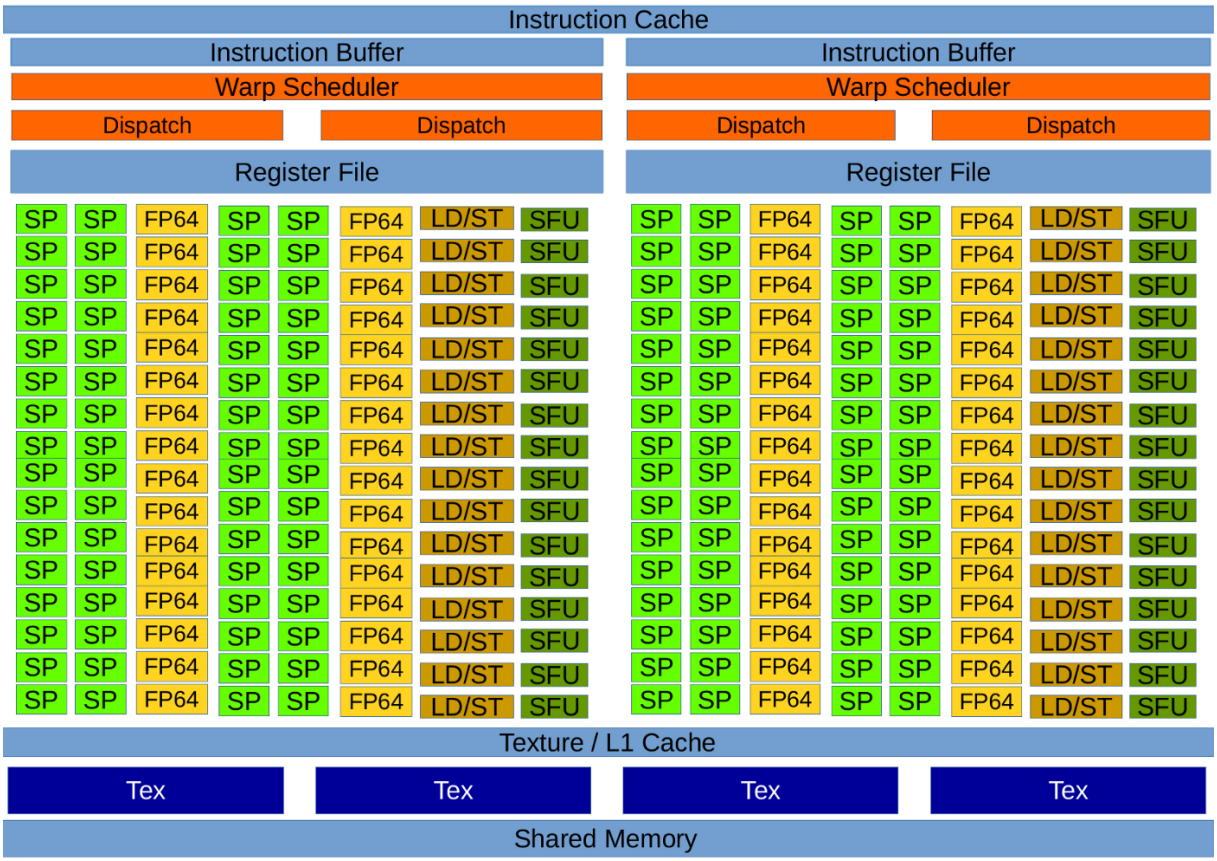

Figure 10: Pascal Architecture [16]

- Pascal supported in CUDA 8 and later [17].

4.6 Volta

In 2017, Volta architecture, which was introduced by NVIDIA instead of Pascal, was released. The Volta series is named after the Italian chemist and physicist Alessandro Volta. Volta is the codename for the GPU microarchitecture. Originally planned in 2013, this architecture was not released until 2017. It is NVIDIA's first chip with Tensor Cores specially designed to provide high deep learning performance compared to regular cores. This architecture was first used in the Tesla V100. The Volta architecture has up to 84 SMs and eight 512-bit memory controllers. Each SM has 64 FP32 CUDA cores, 64 INT32 CUDA cores, 32 FP64 CUDA Cores, 8 tensor cores for deep learning matrix arithmetic, 32 LD/ST units, and 16 SFUs. Each SM is divided into four blocks. Each of these has L0 instruction cache. Thus, it is aimed to increase the performance [16,21].

Figure 11: Volta Architecture [16]

- Volta supported in CUDA 9 and later [17].

4.7 Turing

Turing architecture was introduced by NVIDIA in 2018. The Turing series is named after the mathematician and computer scientist Alan Turing. Turing is the codename for the GPU microarchitecture. It was introduced in 2018 on Quadro RTX cards at SIGGRAPH 2018, and a week later on consumer GeForce RTX 20 series graphics cards at Gamescom. With this architecture, the first consumer products with real-time ray tracing capability were introduced. It has special artificial intelligence processors, namely Tensor Cores, and ray-tracing processors, namely RT cores. In 2019, NVIDIA introduced GeForce 16-series GPUs. These featured the new Turing architecture but did not have RT cores and Tensor cores. The main difference between Volta and Turing architectures is the target market, explained by NVIDIA CEO Jen-Hsun Huang. In other words, the target markets of Turing architecture and Volta architecture are different. While Volta is designed for large-scale training with up to eight GPUs, the fastest HBM2 and other features for data centers. Turing was designed with three areas in mind: Pro visualization, video gaming, and image generation that uses the Tensor Core [16,22].

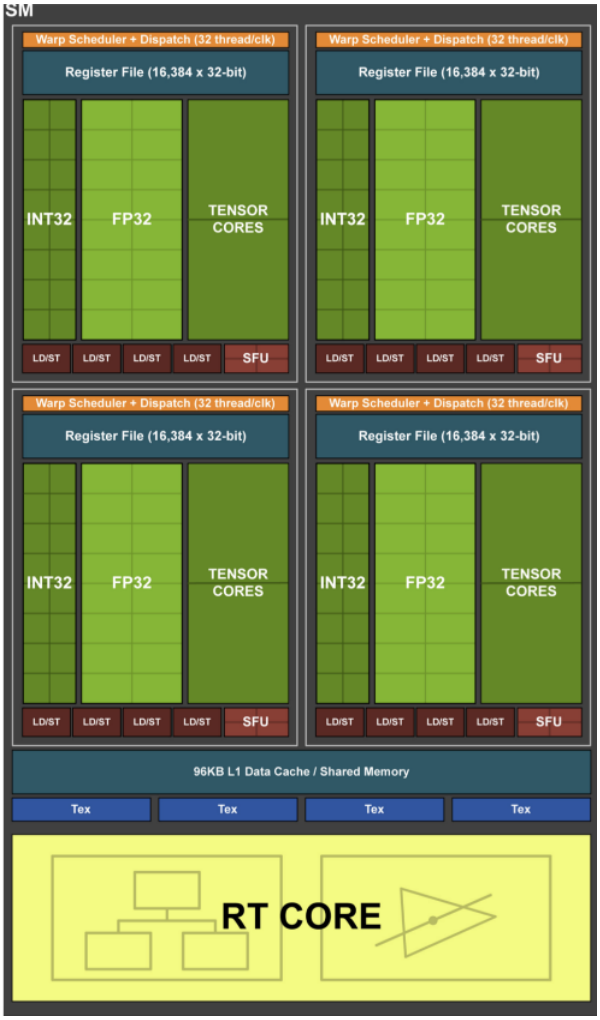

Figure 12: Turing Architecture [16]

- Turing supported in CUDA 10 and later [17].

4.8 Ampere

In 2020, the Ampere architecture was introduced by NVIDIA, replacing both Volta and Turing. It is one of the newer CUDA architectures. The Ampere series is named after the French mathematician and physicist André-Marie Ampère. Ampere is the codename for the GPU microarchitecture. At the GeForce Special Event in 2020, the next-generation GeForce 30 series consumer GPUs were introduced by NVIDIA. In Ampere architecture, each SM has a 192 KB of unified shared memory and L1 data cache. It also includes 40MB of L2 cache to boost performance. This cache is split into two, resulting in higher bandwidth. Each SM consists of four blocks. Each of these blocks goes into L0 cache - to cache data - 6 INT32 CUDA cores, 16 FP32 CUDA cores, 8 FP64 CUDA cores, 8 LD/ST cores, a Tensor core for matrix multiplication, and a 16K 32-bit register file includes [16,23].

Figure 13: Ampere Architecture [16]

- Ampere supported in CUDA 11.1 and later [17].

4.9 Hopper

Figure 14: GH100 Full GPU with 144 SMs [25]

NVIDIA introduced the Hopper architecture in March 2022. The Hopper series is named after the American computer scientist and the United States Navy Rear Admiral Grace Hopper. Hopper GPU microarchitecture and H100 GPU were officially announced by NVIDIA at GTC 2022 in March 2022 [24].

With the improvements made to the A100, the H100 has been designed, which has made the design of the A100 more efficient. H100 with InfiniBand interconnect provides 30 times better performance than A100. NVIDIA H100 GPU, new fourth-generation Tensor Cores –fast matrix calculations-, a new transformer engine - enables to deliver up to 9x faster AI training and up to 30x faster AI-, the new NVLink Network interconnect and Secure MIG has. Thus, an increase in performance is achieved. The H100 is the first asynchronous GPU. Extends all global-to-shared asynchronous transfers to all address spaces, adds support for tensor memory patterns. It enables applications to build an end-to-end asynchronous pipeline. Thus, data can be moved in or out of the chip, or data movement can be hidden by computation [25].

Fewer CUDA threads are required when managing memory bandwidth with the New Tensor Memory Accelerator. General-purpose calculations can be made with other CUDA threads. With the H100, the CUDA thread group hierarchy has changed with the thread block cluster. A cluster is a block that guarantees concurrent programming and provides efficiency for threads in the SM. Can be run efficiently asynchronously with the cluster in Tensor Memory Accelerator and the Tensor Cores [25].

New fourth-generation Tensor Cores are up to 6x faster than A100 GPUs with acceleration per SM, and features such as more SM counts. With distributed shared memory, direct SM to SM communication is allowed on issues such as loading and storage. With the HBM3 memory subsystem, a bandwidth increase of 2 times is achieved compared to the previous generation. The H100 is the world's first GPU with HBM3 memory. It offers 3TB/s of memory bandwidth. The architecture includes 8 GPCs, 72 TPCs (9 TPCs/GPC), 2 SMs/TPC, 144 SMs per full GPU. It has 128 FP32 CUDA Cores for each SM and 18432 FP32 CUDA Cores for each full GPU. Each SM contains 4 fourth-generation Tensor Cores. Contains 6 HBM3 and 12 512-bit memory controllers along with 60 MB of L2 cache. Also features fourth-generation NVLink technology [25].

The GH100 GPU with 144 SMs is given in Figure 14. The H100 SXM5 GPU has 132 SMs. H100 GPUs are designed to boost performance in AI, HPC, and data centers for data analytics [25].

Figure 15: GH100 Streaming Multiprocessor [25]

- Hopper supported in CUDA 12 and later [17].

5. CUDA LIBRARIES

5.1 Math Libraries

Math libraries accelerated by GPU power can be used in applications in many fields. Examples; molecular dynamics, computational fluid dynamics, computational chemistry, medical imaging, and seismic exploration.

5.1.1 cuBLAS

CUDA Basic Linear Algebra Subroutines library.

The cuBLAS library enables basic linear algebra subroutines to be used in applications accelerated by GPU power. It includes plugins like batched operations, execution across multiple GPUs, and mixed and low precision execution. Applications using cuBLAS will

achieve better performance, benefiting from regular performance improvements and new GPU architectures. It uses tensor cores to speed up low and mixed-precision matrix multiplication. This library is included in both NVIDIA HPC SDK and CUDA Toolkit [26].

5.1.2 cuFFT

CUDA Fast Fourier Transform library

It is a Fast Fourier Transform library consisting of two components, cuFFT and cuFFTW. This library provides FFT applications that are up to 10x faster and use GPU power compared to CPU alternatives. It can be used when creating applications in areas such as deep learning, computer vision, computational physics, molecular dynamics, quantum chemistry, and seismic and medical imaging. Applications using cuFFT will achieve better performance, benefiting from regular performance improvements and new GPU architectures. It has features such as 1D, 2D, and 3D transforms of complex and real data types, half, single and double precision transforms, streamed asynchronous execution [27].

5.1.3 cuRAND

CUDA Random Number Generation library

The cuRAND library provides GPU-powered accelerated, high-performance, and high-quality random number generation (RNG). In doing so, it uses hundreds of processor cores in NVIDIA GPUs. It has features such as four high-quality RNG algorithms and distribution options, a flexible usage model. This library is included in both NVIDIA HPC SDK and CUDA Toolkit [28].

5.1.4 cuSOLVER

CUDA based collection of dense and sparse direct solvers.

cuSOLVER is a collection based on the cuBLAS and cuSPARSE libraries. It provides features like factorization, triangulation for dense matrices. Since this library contains a collection of dense and sparse direct linear solvers and Eigen solvers, it can be used in applications in many fields. Examples are Computer Vision, CFD, Computational Chemistry, and Linear Optimization. cuSOLVER uses 11 DMMA Tensor Cores. This library is included in both NVIDIA HPC SDK and CUDA Toolkit [29].

5.1.5 cuSPARSE

CUDA Sparse Matrix library

This library provides basic linear algebra subroutines for handling sparse matrices in applications that use GPU power and are faster than CPU alternatives. It can be used in applications such as machine learning, computational fluid dynamics, seismic exploration, and computational sciences. Applications using cuSPARSE will achieve better performance, benefiting from regular performance improvements and new GPU architectures. This library is included in both NVIDIA HPC SDK and CUDA Toolkit [30].

5.1.6 cuTENSOR

The cuTENSOR library is a first-of-its-kind GPU-powered tensor linear algebra library. With this library, tensor contraction, reduction and elementwise process can be done. It can be used to accelerate applications in areas such as deep learning training and inference, computer vision, quantum chemistry and computational physics. Applications using cuTENSOR will achieve better performance, benefiting from regular performance improvements and new GPU architectures [31].

5.1.7 AmgX

AmgX is one of the cutting-edge libraries. Parallelism-optimized methods have features such as flexible selection in solver generation, complex solver, and preconditioner generation. Provides MPI and OpenMP support [32].

5.2 Parallel Algorithm Libraries

There are GPU-accelerated libraries for parallel algorithms. It is used for many operations in C++ and provides high performance. It can be used when working on topics such as relationships in natural sciences, logistics, and travel planning.

5.2.1 Thrust

A library of parallel algorithms and data structures for C++ developers. It provides a flexible interface that increases developer productivity. It has STL-like templated interfaces to algorithms and data structures. Sort, scan, transform, and reduction operations can be performed faster than CPUs. A tested version of Thrust is available in the CUDA Toolkit [33].

5.3 Image and Video Libraries

CUDA and GPU components and image and video decoding, encoding, and processing can be done with the libraries in this section. These libraries are GPU accelerated libraries.

5.3.1 nvJPEG

It is a GPU-accelerated library that includes process such as decoding, encoding and transcoding for JPEG format images by achieving high performance with GPU power. It provides high throughput and low latency compared to CPU. This library can be used in applications such as image classification, object detection and image segmentation in computer vision. It features hybrid decoding using CPU and GPU, color space conversion to RGB, BGR, RGBI, BGRI, and YUV. There is an updated version in the CUDA Toolkit [34].

5.3.2 nvJPEG2000

It is a GPU-accelerated library that includes process such as decoding, encoding and transcoding for JPEG 2000 format images by achieving high performance with GPU power. It provides high throughput and low latency compared to CPU [34].

5.4 Communication Libraries

5.4.1 NVSHMEM

NVIDIA GPU is a parallel programming interface that provides efficient and scalable communication for clusters. It has features such as combining multiple GPU memories with partitioned global address space that is accessed, interoperability with MPI and other OpenSHMEM applications, and support for x86 and POWER9 processors. Enables long-running cores that include both communication and computation. Reduces overheads by limiting application performance with scaling. With asynchronous communication, computation and interleaving of communications are made easy. Thus, an increase in performance is achieved. It can be used in applications in fields such as image processing, machine learning, and scientific computing. [35].

5.4.2 NCCL

It is optimized to achieve high bandwidth and low latency over high-speed interconnects with PCIe and NVLink. It has features such as removing the need for optimization on specific machines, ease of programming – because it has a simple API accessible from a variety of programming languages – and compatibility with most multi-GPU parallelization models. It provides operations such as all-gather, all-reduce, broadcast, reduce, reduce-scatter as well as point-to-point send and receive. Deep learning frameworks such as Caffe2, Chainer, MxNet, PyTorch, and TensorFlow have integrated NCCL to accelerate training. This library can be downloaded either as part of the NVIDIA HPC SDK. [36].

5.5 Deep Learning Libraries

There are GPU-accelerated libraries for deep learning applications.

5.5.1 cuDNN

cuDNN is a GPU accelerated library. The cuDNN library is used in many cases for deep neural network computations. cuDNN provides highly tuned implementations for operations such as forward and backward convolution, pooling, normalization, and activation layers. This library accelerates deep learning frameworks such as Caffe2, Chainer, Keras, MATLAB, MxNet, PaddlePaddle, PyTorch and TensorFlow. Performance gains in all deep learning frameworks when CUDA and cuDNN versions are developed and updated. This performance may vary depending on how well multiple GPUs and multiple nodes are scaled [37].

5.5.2 TensorRT

TensorRT is an SDK for high-performance deep learning inference. Provides a deep learning inference optimizer and runtime for TensorRT inference applications. Thus, it helps to provide high efficiency [38].

5.5.3 DeepStream SDK

DeepStream SDK is a video extraction library. Since there are many cameras and sensors in the world, there is a large amount of data. This library is used to process data in order to quickly develop applications and services where the data will be used. DeepStream has a stream analysis toolset for multi-sensor processing, video, audio, and image processing. Powerful and flexible SDK provides many features such as low-code programming, real-time analysis, and managed artificial intelligence services with the use of a user interface [39].

REFERENCES

1. Patwardhan, R. (2021). CPU vs. GPU | Best Use Cases For Each https://www.weka.io/blog/cpu-vs-gpu/
2. Gillis, A.S. graphics processing unit (GPU) https://www.techtarget.com/searchvirtualdesktop/definition/GPU-graphics-processing-unit
3. Glawion, A. (2022). CUDA Cores vs. Stream Processors (And other GPU Cores Explored). https://www.cgdirector.com/cuda-cores-vs-stream-processors/
4. Gupta, P. (2020). CUDA Refresher: Reviewing the Origins of GPU Computing https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/
5. CUDA Zone. (n.d). https://developer.nvidia.com/cuda-zone
6. CUDA. (2022). *Wikipedia* https://en.wikipedia.org/wiki/CUDA
7. Heller, M. (2018). What is CUDA? Parallel programming for GPUs https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html
8. Tensor Processing Unit. (2022). *Wikipedia* https://en.wikipedia.org/wiki/Tensor_Processing_Unit
9. GPGPU Definition. (n.d). https://www.heavy.ai/technical-glossary/gpgpu
10. Glaskowsky, P. N. (2009). NVIDIA's Fermi: The First Complete GPU Computing Architecture https://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf
11. Graphics processing unit. (2022). *Wikipedia* https://en.wikipedia.org/wiki/Graphics_processing_unit#Stream_processing_and_general_purpose_GPUs_(GPGPU)
12. Tesla (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Tesla_(microarchitecture)

13. Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J. (2008). NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro, 28(2), 39–55. doi:10.1109/mm.2008.31
14. Turgut, Ç. Ergin, O. (2017). GPU Önbelleklerinde Yerelliğe Bağlı Dinamik Yazma Politikası https://slideplayer.biz.tr/slide/12070497/
15. Fermi (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Fermi_(microarchitecture)
16. PD, Adrian. (2020). How the hell are GPUs so fast? A HPC walk along Nvidia CUDA-GPU architectures. From zero to nowadays. https://towardsdatascience.com/how-the-hell-are-gpus-so-fast-a-e770d74a0bf
17. Shimoni, A. (2020) Matching CUDA arch and CUDA gencode for various NVIDIA architectures https://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/
18. Kepler (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Kepler_(microarchitecture)
19. Maxwell (microarchitecture). (2022) *Wikipedia* https://en.wikipedia.org/wiki/Maxwell_(microarchitecture)
20. Pascal (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Pascal_(microarchitecture)
21. Volta (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Volta_(microarchitecture)
22. Turing (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Turing_(microarchitecture)
23. Ampere (microarchitecture). (2022). *Wikipedia* https://en.wikipedia.org/wiki/Ampere_(microarchitecture)
24. Hopper (microarchitecture). (2022). https://en.wikipedia.org/wiki/Hopper_(microarchitecture)
25. Andersch, M. Palmer, G. Krashinsky, R. Stam, N. Mehta, V. Brito, G. Ramaswamy, S. (2022). NVIDIA Hopper Architecture In-Depth https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

26. cuBLAS. (n.d)
https://developer.nvidia.com/cublas
27. cuFFT. (n.d)
https://developer.nvidia.com/cufft
28. cuRAND. (n.d)
https://developer.nvidia.com/curand
29. cuSOLVER. (n.d)
https://developer.nvidia.com/cusolver
30. cuSPARSE. (n.d)
https://developer.nvidia.com/cusparse
31. cuTENSOR (n.d.)
https://developer.nvidia.com/cutensor
32. AmgX. (n.d)
https://developer.nvidia.com/amgx
33. Thrust. (n.d)
https://developer.nvidia.com/thrust
34. nvJPEG Libraries. (n.d)
https://developer.nvidia.com/nvjpeg
35. NVSHMEM. (n.d).
https://developer.nvidia.com/nvshmem
36. NVIDIA NCCL. (n.d.).
https://developer.nvidia.com/nccl
37. NVIDIA cuDNN. (n.d).
https://developer.nvidia.com/cudnn
38. NVIDIA TensorRT. (n.d)
https://developer.nvidia.com/tensorrt
39. DeepStream SDK. (n.d).
https://developer.nvidia.com/deepstream-sdk