

## Computer Architecture

# GPU

Dr Jeff Drobman

website → [drjeffsoftware.com/classroom.html](https://drjeffsoftware.com/classroom.html)

email → [jeffrey.drobman@csun.edu](mailto:jeffrey.drobman@csun.edu)

# GPU Preview

## ❖ General

## ❖ CUDA

## ❖ zyBook Ch 9

## ❖ GPU Products

### ❑ Apple

### ❑ AMD

- Vega (see new AMD **RDNA3** slides in *Roadmap* file)
- Radeon/Navi

### ❑ Intel

- Xe
- Max GPU

### ❑ Nvidia

- Cuda
- GeForce

### ❑ ARM

- Mali

# Section



New GPU Slides

# Video Frames (Mpixels)

- Supports 12/10 bits RAE RGB
  - Supports images sizes
    - 3840x2160
    - 2560x1440
    - 1080p(1920x1080)
    - 720p(1280x720)
- | ❖ Mpixels |
|-----------|
| • 8.3     |
| • 3.7     |
| • 2.1     |
| • 0.9     |

# GPU vs CPU

While CPUs focused on increasing single-thread performance, GPUs focused on increasing throughput. This is apparent in terms of architecture.

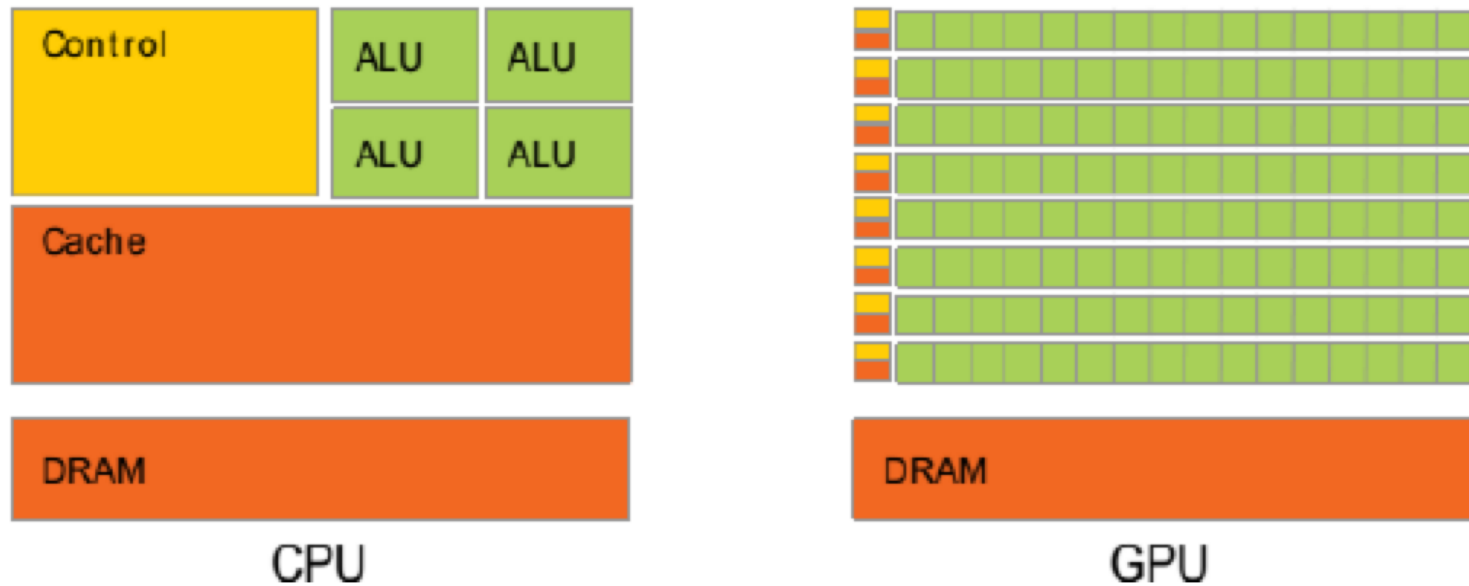


image: [Fig. 1. CPU vs GPU Architecture](#) ↗

Wide GPU cores (AMD CU, Nvidia SM) can deal with dozens of SIMD operations at once, with each values corresponding to things like polygons vertices. These SIMD lanes can be kept occupied because the workload is embarrassingly parallel. A CPU would have a very hard time mapping a C program to more than a few lanes at once, by contrast.

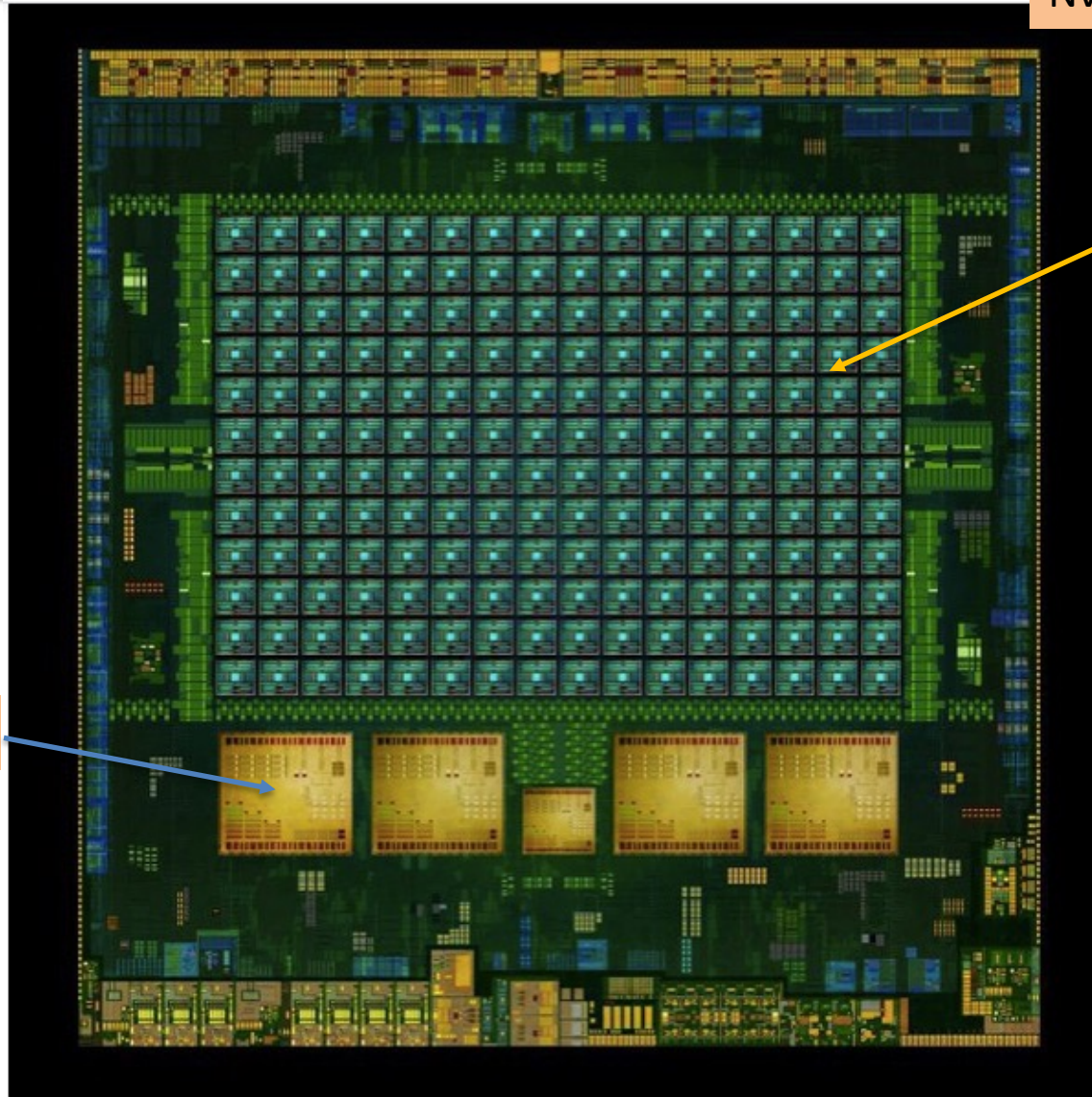
Trying to make a CPU more GPU-like, or vice versa, would only result in a decrease in performance. Both units are optimized for their particular jobs.

# SoC = CPU + GPU

Nvidia

GPU cores

CPU cores



*This SOC has four CPU cores (ARM Cortex) and 192 GPU cores (Kepler).*

# GPU “Cores”



Victor Eijkhout · Follow

To quote Grace Hopper: "Hardware is the part of a computer that you can kick". · 5y

**If a GPU has around 1000 to 2000 cores, then why does a CPU still have only 12 cores?**

A GPU does not have a thousand cores. It has a couple of dozen, which have a bunch of SIMD lanes. (Those are the “warps”: threads in a warp are really SIMD lanes in a single processor.) All this “massively parallel” stuff is just NVidia marketing.

Oh, and how many cores does a CPU have? The Intel Xeon Phi “Knights Landing” has 68 cores (depends slightly on the model) no matter how you count it. Each core has 4 hyperthreads, so that’s kinda 272 cores (in fact, that is what many hardware discovery tools will report) and each of those is 8-wide SIMD, so it has 2000+ cores if you use the NVidia way of counting.

That’s why NVidia coined the term “SIMT” Single Instruction Multiple Thread: a GPU executes only a single (sub)program, but it spreads it over many threads. Like having just one core, but with an enormous SIMD width.

SIMD  $\leftrightarrow$  SIMT?

# GPU “Cores”



Larry Seiler · [Follow](#)



PhD in Computer Science, Massachusetts Institute of Technology (Graduated 1985) · 3y

**What is the difference between CPU cores and GPU cores in terms of computation?**

GPUs were originally designed to run a large number of processing streams at once — first tens, then hundreds, now sometimes thousands. Originally the processing streams were fixed function, but now they are programs with branching, random memory access and other general purpose capabilities. And they aren't just used for graphics: “compute shaders” are used for a wide variety of processing.

The key difference remains that CPUs are primarily good for data-dependent algorithms that have parallelism but not huge amounts, whereas GPUs are primarily good for algorithms that have a huge amount of inherent parallelism and that minimize the amount of data-dependent control flow. Getting the best bandwidth is of secondary importance to minimizing the latency.

# GPU Cache



Joe Zbiciak · 52m

They do, in fact, have L1/L2 caches in the streaming multiprocessor (SM) units (Nvidia Ampere). The diagram below doesn't show the shared L2. It's 40MB.



# Section



## General GPU

# GPU

**What is the reason for the big difference in performance between graphics processing units (GPUs) and central processing units (CPUs) when it comes to rendering videos or playing video games, but not so much when it comes to web browsing?** ...



**Jeff Drobman**

Lecturer at California State University, Northridge (2016–present) · 🕒 16m · 💰

GPU's are designed for graphics rendering to produce a million-pixel frame 30–60 times per second. that involves a lot of functions (3D, shading, etc.) which can be given their own threads. in fact, CPU cores typically handle 1–2 threads, while GPU cores can handle 1000's.

High Thread Count!

# CPU/GPU Units Rel



Brett Bergan · [Follow](#)

Building PC's for 25 years · Updated Jul 6



**Does an entire computer chip get rendered useless if just one of its billions of transistors gets damaged or placed incorrectly?**

Ironically, the majority of CPU/GPU chips produced are dies that have flaws. And yes, the flawed sections of the chip are useless.

Have you heard of the Ryzen 5 5600X?

These are chips with a fatal flaw in one of the CPU cores. A perfect chip is called a Ryzen 7 5800X. But if the die has a flaw in one of the cores, the pair of cores affected is simply cut off from the rest of the chip and used as a six core processor instead of an eight-core processor.

This is even more common in GPU production. RTX 3000 dies have thousands of cores. The functional unit is called a streaming multiprocessor (SM). Each SM has one RT Core and 128 so called CUDA cores along with a certain amount of "local" VRAM along with Integer units, floating point units and tensor cores.

If one part of the SM is damaged the entire SM is junk. So it is simply removed from the system.

## Graphics processing unit

From Wikipedia, the free encyclopedia

*For an expansion card that contains a graphics processing unit, see [Video cards](#).*

*"GPU" redirects here. For other uses, see [GPU \(disambiguation\)](#).*

A **graphics processing unit** (**GPU**) is a specialized [electronic circuit](#) designed to rapidly manipulate and alter [memory](#) to accelerate the creation of [images](#) in a [frame buffer](#) intended for output to a [display device](#). GPUs are used in [embedded systems](#), [mobile phones](#), [personal computers](#), [workstations](#), and [game consoles](#). Modern GPUs are very efficient at manipulating [computer graphics](#) and [image processing](#). Their highly [parallel structure](#) makes them more efficient than general-purpose [central processing units](#) (CPUs) for [algorithms](#) that process large blocks of data in parallel. In a personal computer, a GPU can be present on a [video card](#) or embedded on the [motherboard](#). In certain CPUs, they are embedded on the CPU die.<sup>[1]</sup>

The term "GPU" was coined by [Sony](#) in reference to the [PlayStation](#) console's [Toshiba](#)-designed [Sony GPU](#) in 1994.<sup>[2]</sup> The term was popularized by [Nvidia](#) in 1999, who marketed the [GeForce 256](#) as "the world's first GPU".<sup>[3]</sup> It was presented as a "single-chip [processor](#) with integrated [transform](#), [lighting](#), [triangle setup/clipping](#), and rendering engines".<sup>[4]</sup> Rival [ATI Technologies](#) coined the term "**visual processing unit**" or **VPU** with the release of the [Radeon 9700](#) in 2002.<sup>[5]</sup>

# GPU History

COMP222

Wikipedia

The term "GPU" was coined by [Sony](#) in reference to the 32-bit [Sony GPU](#) (designed by [Toshiba](#)) in the [PlayStation](#) video game console, released in 1994.<sup>[2]</sup>

In the PC world, notable failed first tries for low-cost 3D graphics chips were the [S3 VIRGE](#), [ATI Rage](#), and [Matrox Mystique](#). These chips were essentially previous-generation 2D accelerators with 3D features bolted on. Many were even [pin-compatible](#) with the earlier-generation chips for ease of implementation and minimal cost. Initially, performance 3D graphics were possible only with discrete boards dedicated to accelerating 3D functions (and lacking 2D GUI acceleration entirely) such as the [PowerVR](#) and the [3dfx Voodoo](#). However, as manufacturing technology continued to progress, video, 2D GUI acceleration and 3D functionality were all integrated into one chip. [Rendition's](#) *Verite* chipsets were among the first to do this well enough to be worthy of note. In 1997, Rendition went a step further by collaborating with [Hercules](#) and Fujitsu on a "Thriller Conspiracy" project which combined a Fujitsu FXG-1 Pinolite geometry processor with a Vérité V2200 core to create a graphics card with a full T&L engine years before Nvidia's [GeForce 256](#). This card, designed to reduce the load placed upon the system's CPU, never made it to market.<sup>[citation needed]</sup>

[OpenGL](#) appeared in the early '90s as a professional graphics API, but originally suffered from performance issues which allowed the [Glide API](#) to step in and become a dominant force on the PC in the late '90s.<sup>[41]</sup> However, these issues were quickly on the wayside. Software implementations of OpenGL were common during this time, although the influence of OpenGL eventually led to the time, a parity emerged between features offered in hardware and those offered in OpenGL. [DirectX](#) became popular among [Windows](#) in the 90s. Unlike OpenGL, Microsoft insisted on providing strict one-to-one support of hardware. The approach made DirectX less popular initially, since many GPUs provided their own specific features, which existing OpenGL applications were already able to benefit from. DirectX was a generation behind. (See: [Comparison of OpenGL and Direct3D](#).)

# GPU History

COMP222

Wikipedia

## 2010 to present [\[ edit \]](#)

In 2010, Nvidia began a partnership with [Audi](#) to power their cars' dashboards. These [Tegra](#) GPUs were powering the cars' dashboard, offering increased functionality to cars' navigation and entertainment systems.<sup>[49]</sup> Advancements in GPU technology in cars has helped push [self-driving technology](#).<sup>[50]</sup> AMD's [Radeon HD 6000 Series](#) cards were released in 2010 and in 2011, AMD released their 6000M Series discrete GPUs to be used in mobile devices.<sup>[51]</sup> The Kepler line of graphics cards by Nvidia came out in 2012 and were used in the Nvidia's 600 and 700 series cards. A feature in this [new GPU microarchitecture](#) included GPU boost, a technology adjusts the clock-speed of a video card to increase or decrease it according to its power draw.<sup>[52]</sup> The [Kepler microarchitecture](#) was manufactured on the 28 nm process.

## GPU companies [\[ edit \]](#)

Nvidia, AMD, Intel

Many companies have produced GPUs under a number of brand names. In 2009, [Intel](#), [Nvidia](#) and [AMD/ATI](#) were the market share leaders, with 49.4%, 27.8% and 20.6% market share respectively. However, those numbers include Intel's integrated graphics solutions as GPUs. Not counting those, [Nvidia](#) and [AMD](#) control nearly 100% of the market as of 2018. Their respective market shares are 66% and 33%.<sup>[61]</sup> In addition, [S3 Graphics](#)<sup>[62]</sup> and [Matrox](#)<sup>[63]</sup> produce GPUs. [Modern smartphones](#) also use mostly [Adreno](#) GPUs from [Qualcomm](#), [PowerVR](#) GPUs from [Imagination Technologies](#) and [Mali](#) GPUs from [ARM](#).

## Computational functions [\[ edit \]](#)

3D graphics rendering and decoding

Modern GPUs use most of their [transistors](#) to do calculations related to [3D computer graphics](#). In addition to the 3D hardware, today's GPUs include basic 2D acceleration and [framebuffer](#) capabilities (usually with a VGA compatibility mode). Newer cards such as AMD/ATI HD5000-HD7000 even lack 2D acceleration; it has to be emulated by 3D hardware. GPUs were initially used to accelerate the memory-intensive work of [texture mapping](#) and [rendering](#) polygons, later adding units to accelerate [geometric](#) calculations such as the [rotation](#) and [translation](#) of [vertices](#) into different [coordinate systems](#). Recent developments in GPUs include support for [programmable shaders](#) which can manipulate vertices and textures with many of the same operations supported by [CPUs](#), [oversampling](#) and [interpolation](#) techniques to reduce [aliasing](#), and very high-precision [color spaces](#). Because most of these computations involve [matrix](#) and [vector](#) operations, engineers and scientists have increasingly studied the use of GPUs for non-graphical calculations; they are especially suited to other [embarrassingly parallel](#) problems.

With the emergence of deep learning, the importance of GPUs has increased. In research done by Indigo, it was found that while training deep learning neural networks, [GPUs can be 250 times faster than CPUs](#). The explosive growth of Deep Learning in recent years has been attributed to the emergence of general purpose GPUs.<sup>[64]</sup> There has been some level of competition in this area with [ASICs](#), most prominently the [Tensor Processing Unit](#) (TPU) made by Google. However, ASICs require changes to existing code and GPUs are still very popular.

# GPU History

## P&H Ch 9

### A brief history of GPU evolution

Fifteen years ago, there was no such thing as a GPU. Graphics on a PC were performed by a video graphics array (VGA) controller. A VGA controller was simply a memory controller and display generator connected to some DRAM. In the 1990s, semiconductor technology advanced sufficiently that more functions could be added to the VGA controller. By 1997, VGA controllers were beginning to incorporate some *three-dimensional* (3D) acceleration functions, including hardware for triangle setup and rasterization (dicing triangles into individual pixels) and texture mapping and shading (applying "decals" or patterns to pixels and blending colors).

In 2000, the single chip graphics processor incorporated almost every detail of the traditional high-end workstation graphics pipeline and, therefore, deserved a new name beyond VGA controller. The term GPU was coined to denote that the graphics device had become a processor.

Over time, GPUs became more programmable, as programmable processors replaced fixed function dedicated logic while maintaining the basic 3D graphics pipeline organization. In addition, computations became more precise over time, progressing from indexed arithmetic, to integer and fixed point, to single precision floating-point, and recently to double precision floating-point. GPUs have become massively parallel programmable processors with hundreds of cores and thousands of threads.

Recently, processor instructions and memory hardware were added to support general purpose programming languages, and a programming environment was created to allow GPUs to be programmed using familiar languages, including C and C++. This innovation makes a GPU a fully general-purpose, programmable, manycore processor, albeit still with some special benefits and limitations.

### GPU graphics trends

GPUs and their associated drivers implement the OpenGL and DirectX models of graphics processing. OpenGL is an open standard for 3D graphics programming available for most computers. DirectX is a series of Microsoft multimedia programming interfaces, including Direct3D for 3D graphics. Since these *application programming interfaces* (APIs) have well-defined behavior, it is possible to build effective hardware acceleration of the graphics processing functions defined by the APIs. This is one of the reasons (in addition to increasing device density) why new GPUs are being developed every 12 to 18 months that double the performance of the previous generation on existing applications.

## GPU forms [\[ edit \]](#)

### Terminology [\[ edit \]](#)

In personal computers, there are two main forms of GPUs. Each has many synonyms:<sup>[65]</sup>

- *Dedicated graphics card* - also called *discrete*.
- *Integrated graphics* - also called: *shared graphics solutions*, *integrated graphics processors* (IGP), or *unified memory architecture* (UMA).

### Video decoding processes that can be accelerated [\[ edit \]](#)

The video decoding processes that can be accelerated by today's modern GPU hardware are:

- Motion compensation (mocomp)
- Inverse discrete cosine transform (iDCT)
  - Inverse telecine 3:2 and 2:2 pull-down correction
- Inverse modified discrete cosine transform (iMDCT)
- In-loop deblocking filter
- Intra-frame prediction
- Inverse quantization (IQ)
- Variable-length decoding (VLD), more commonly known as slice-level acceleration
- Spatial-temporal deinterlacing and automatic interlace/progressive source detection
- Bitstream processing (Context-adaptive variable-length coding/Context-adaptive binary arithmetic coding) and perfect pixel positioning.

The above operations also have applications in video editing, encoding and transcoding

# GPU Functions

## Usage specific GPU [\[ edit \]](#)

Most GPUs are designed for a specific usage, real-time 3D graphics or other mass calculations:

1. Gaming
  - [GeForce GTX, RTX](#)
  - [Nvidia Titan](#)
  - [Radeon HD, R5, R7, R9, RX, Vega and Navi series](#)
2. Cloud Gaming
  - [Nvidia Grid](#)
  - [AMD Radeon Sky](#)
3. Workstation (Video editing, encoding, decoding, transcoding and rendering (digital content creation), 3D animation and rendering (CGI), videogame development and 3D texture creation, product development/3D CAD, structural analysis, simulations, (calculations...))
  - [Nvidia Quadro](#)
  - [AMD FirePro](#)
  - [AMD Radeon Pro](#)
  - [AMD Radeon VII](#)
4. Cloud Workstation
  - [Nvidia Tesla](#)
  - [AMD FireStream](#)
5. Artificial Intelligence training and Cloud
  - [Nvidia Tesla](#)
  - [AMD Radeon Instinct](#)
6. Automated/Driverless car
  - Nvidia [Drive PX](#)

# GPU's



**Intel Graphics Technology (GT)** is the collective name for a series of integrated graphics processors (IGPs) produced by Intel that are manufactured on the same package or die as the central processing unit (CPU). It was first introduced in 2010 as **Intel HD Graphics**.

## Graphics Processing Unit



A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display

device. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing. Their highly parallel structure makes them more efficient than general-purpose central processing units (CPUs) for algorithms that process large blocks of data in parallel. In a personal computer, a GPU can be present on a video card or embedded on the motherboard. In certain CPUs, they are embedded on the CPU die.

# CPU vs GPU

## Why don't CPUs have as much parallelism as a GPU?



**Jeff Drobman**, Lecturer at California State University, Northridge (2016-present)

Answered just now

one way to look at it: CPU's operate mostly on single data (SISD or MISD), and often relegate parallel operations on vector data to the GPU's. while some CPU's do have extensions to support vectors (SIMD, AVX), when paired with GPU's they don't need the SIMD extensions because the GPU's will do that work, and more effectively with their very large number of EU's and threads.

# CPU vs GPU: ILP/DLP

---

## ❖ CPU

- ❑ ILP limited (typ  $\leq 4$  per core,  $\leq 24$  cores)
- ❑ DLP small in AVX-128/256/512
- ❑ Cores: larger so fewer
- ❑ Threads limited to 2-4/core
- ❑ **Integer** operations (mostly)

## ❖ GPU

- ❑ ILP very high (typ  $> 1000$ )
- ❑ DLP very high (1000's of EU's)
- ❑ Cores: smaller so many more
- ❑ Threads in the 1000's
- ❑ **FP** operations

# Discrete vs Integrated GPU

**Quora**



**Mohd Zaid**, computer enthusiast, techno freak nd android lover♥

Answered Aug 9, 2016

Originally Answered: Why Intel HD graphics is unfit for gaming?

The difference is of speciality.

The intel HD graphics are not very good for gaming because they are integrated graphics as compared to the dedicated graphic cards such as Nvidia's

The Integrated (or shared graphics) can never be as good as the dedicated one.

## Integrated vs. Dedicated graphics

As its name suggests, a **dedicated** graphics card — often also called discrete graphics — is a piece of specialist hardware dedicated solely to managing the graphics performance in a computer.

It consists of a graphics processing unit (GPU), which functions similarly to the main processor (CPU) in the computer, and its own dedicated RAM.

# Discrete vs Integrated GPU

**Quora**



**Mohd Zaid**, computer enthusiast, techno freak nd android lover♥

Answered Aug 9, 2016

Originally Answered: Why Intel HD graphics is unfit for gaming?

**In shared — or integrated —** graphics systems, these components are built into the same chip as the CPU. The memory assigned to graphics is shared with the main system memory. This means if your PC has 4GB of RAM and 1GB shared graphics memory, only 3GB of that memory will be available to general computing tasks.

A dedicated graphics card is generally more powerful than a shared graphics system. It's also larger, uses more power and generates more heat.

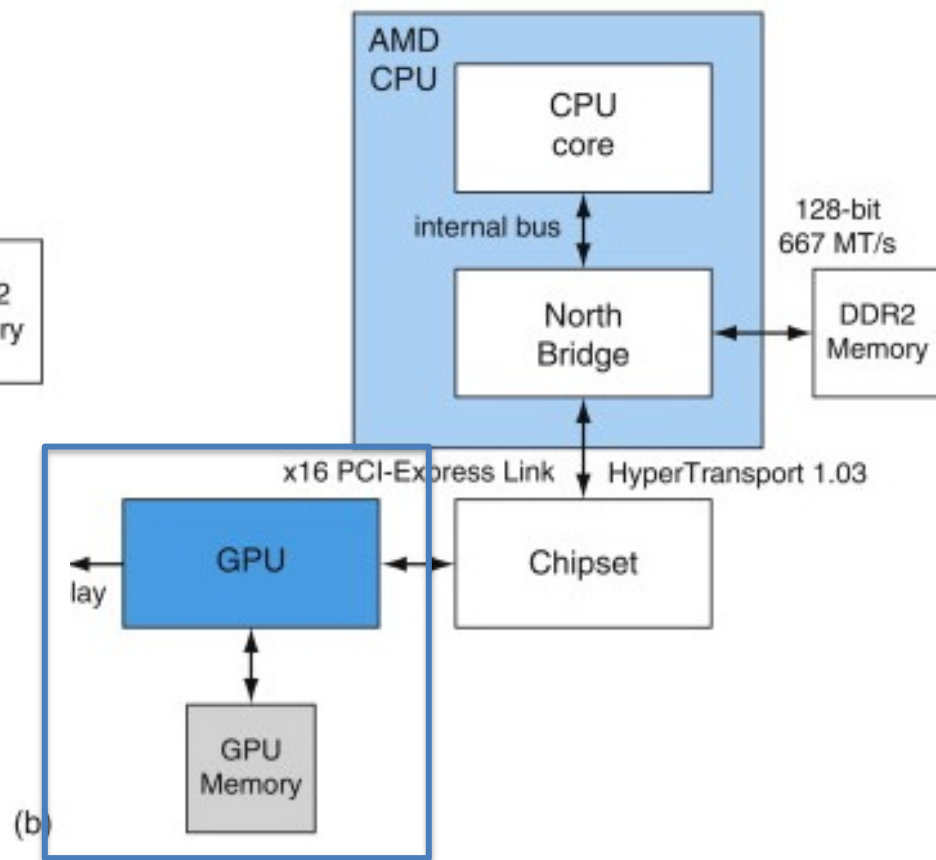
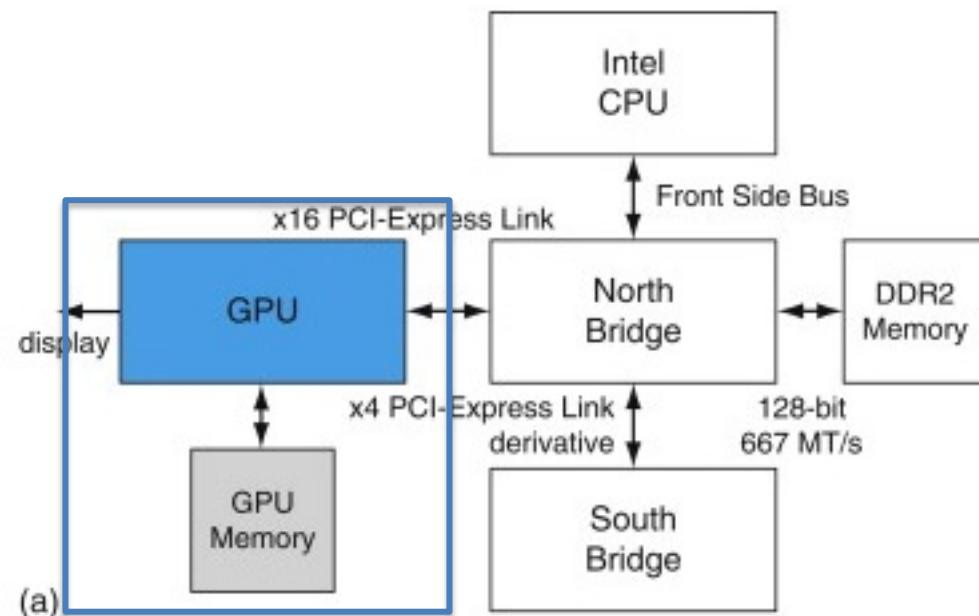
Some computers offer both shared and dedicated graphics, providing the choice between the best graphical performance or longer battery life. In these systems, you can either make the choice yourself, or the computer will decide what's best on the fly.

# APU = CPU + GPU

Hennessy & Patterson

Intel

AMD

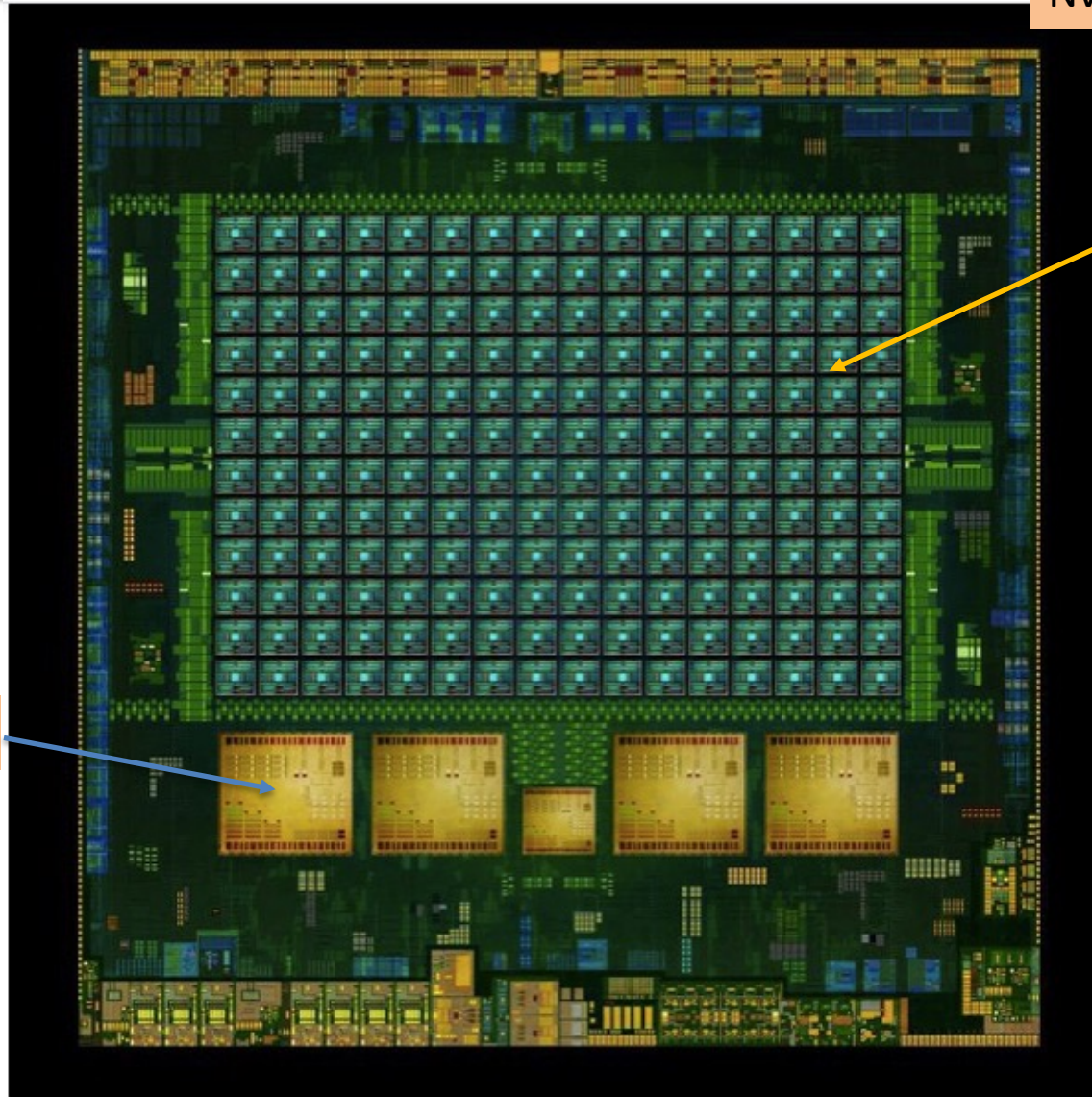


# SoC = CPU + GPU

Nvidia

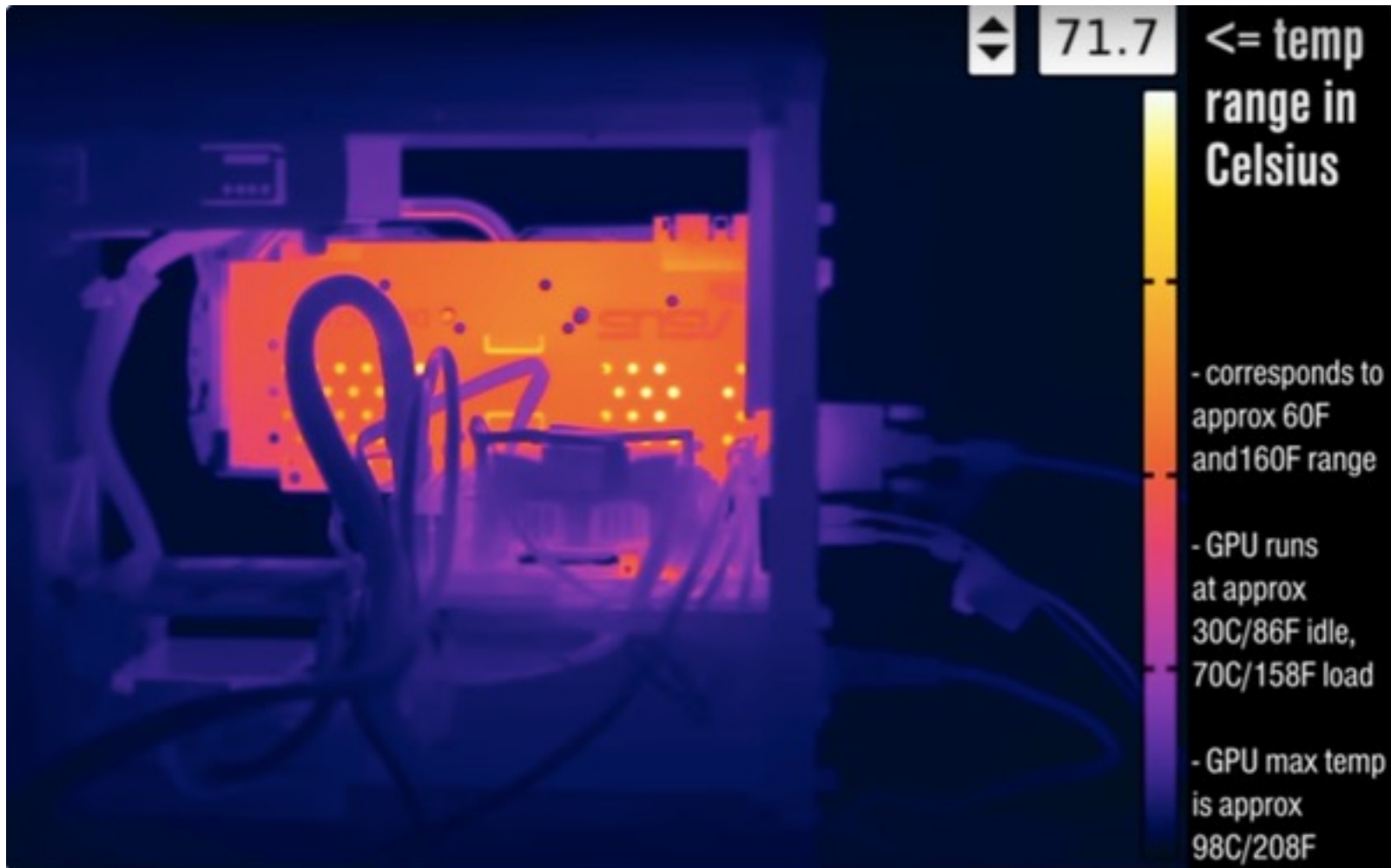
GPU cores

CPU cores



*This SOC has four CPU cores (ARM Cortex) and 192 GPU cores (Kepler).*

# GPU < CPU Clock Freq



This is an image made with a thermal imaging camera.

# GPU



**Huseyin Tugrul Buyukisik**, B.S. Numerical Analysis, Physics Engineer  
(2012)

A GPU is made of multiple "streaming multiprocessors".

A streaming multiprocessor is made of multiple "warp schedulers"

Each warp scheduler run commands on multiple SIMD/SIMT units.

There are many SIMD/SIMT units per streaming multiprocessor.

Each SIMD/SIMT unit is made of many ALU (arithmetic logic units) with their floating-point units. There are also dedicated compute units to help SIMD/SIMTs in computing. These are special function units, tensor cores, raytracing cores, texture z-order fetching but not all GPUs have same types, For example, older Nvidia GPUs have only special function units. Old Amd GPUs have different number of floating-point units for 64-bit.



**Kiryl Persianov**



Studied Ontology (philosophy) & Artificial Intelligence · Updated 7mo

**Are the highly-marketed Tensor cores from Nvidia, Tensor processing Units (TPU) from Google, and other Deep Learning and Machine Learning processors just simple matrix-multiplication accelerators?**

It is just simple data matrix-multiplication accelerators. All the rest is commercial propaganda.

Unlike other computational devices that treat scalar or vectors as primitives, Google's Tensor Process Unit (TPU) ASIC treats matrices as primitives. The TPU is designed to perform matrix multiplication at a massive scale.

# GPU



Kiryl Persianov



Studied Ontology (philosophy) & Artificial Intelligence · Updated 7mo

## Are the highly-marketed Tensor cores from Nvidia, Tensor processing Units (TPU) from Google, and other Deep Learning and Machine Learning processors just simple matrix-multiplication accelerators?

A Graphical Processing Unit (GPU) enables you to run high-definition graphics on your computer. GPU has hundreds of cores aligned in a particular way forming a single hardware unit. It has thousands of concurrent hardware threads, utilized for data-parallel and computationally intensive portions of an algorithm. Data-parallel algorithms are well suited for such devices because the hardware can be classified as SIMT (Single Instruction Multiple Threads). GPUs outperform CPUs in terms of GFLOPS.

# GPU



**Kiryl Persianov**

Studied Ontology (philosophy) & Artificial Intelligence ·

The TPU and NPU go under a Narrow/Weak AI/ML/DL accelerator class of specialized hardware accelerator or computer system designed to accelerate special AI/ML applications, including artificial neural networks and machine vision.

ASIC

Big-Tech companies such as Google, Amazon, Apple, Facebook, AMD and Samsung are all designing their own AI ASICs.

Typical applications include algorithms for training and inference in computing devices, as self-driving cars, machine vision, NLP, robotics, internet of things, and other data-intensive or sensor-driven tasks. They are often manycore designs and generally focus on low-precision arithmetic, novel dataflow architectures or in-memory computing capability, with a typical NAI integrated circuit chip contains billions of MOSFET transistors.

Focus on training and inference of deep neural networks, Tensorflow uses a symbolic math library based on dataflow and differentiable programming

# GPU's

Comparing say the speed of the interface between a soldered down GPU to soldered VRAM chips, to a socketed CPU to socketed DRAM:

CPU->DRAM, DDR4-2133 to 2667, 2.667 GT/s, 64 bits \* 2 = 128 bits

GPU->VRAM, GDDR6 (10-18 Gbps), 18 GT/s, 256 to 384 bits

A common CPU to DRAM interfaces today is DDR4, with speeds of 2133 to 2667 MHz.

See the big speed and width difference? A typical CPU will have 2 channels of 64 bits at DDR4 speeds, which is analogous to each bit having 2.667 Gigatransfers/second.

A typical high-end GPU will have 256 - 384 bit bus to the VRAM at GDDR6 speeds, which can run at up to 18 Gbps, analogous to 18 Gigatransfers/second.

To achieve that kind of speed and width out from the GPU to its VRAM, the interface has to be soldered. For various signal integrity and board routing

# GPU's

## CPU Cache vs. GPU VRAM

You might ask, well, doesn't the CPU suffer with the DRAM interface being so slow? Yes, but it is very well masked by the CPU cache (L1, L2, LLC) in the CPU such that in an optimal situation with typical CPU workloads, the caches can maintain a > 90% cache hit rate, i.e. most of the accesses by the CPU to DRAM can be satisfied by the cache.

A typical GPU workload cannot be cached in the same way, it just needs the fast access to its VRAM. (you could argue that the VRAM is a big cache for the GPU and the alternative is to access the CPU DRAM).

So if a GPU has to be soldered down to a board with its VRAM, by that point, it's going to be a pretty big board to be socketed like a CPU. Technically speaking, it's not practical to build the GPU on a socket and make it run at the speeds it does.

Still, it still ends up that GPUs are sold separate, and that's called a video card! The complete package of the GPU, VRAM, and the power supply for both of them, are all packaged into a board and that is sold as a video card, which you can buy.

Video card

# Nvidia A100 SM

Joe Zbiciak replied to your comment on an answer to: "The difference of number of cores between CPUs and GPUs is huge, then why CPU's performance is far better than GPUs?"

If I understood correctly, that diagram represents one streaming multiprocessor (,SM) core, and apparently the A100 has 108 of them. One SM has:

- 64 FP32 cores
- 32 FP64 cores
- 64 INT32 cores
- 4 Tensor cores
- 65,536 32-bit registers
- 192K L1 cache / shared RAM

"Cores" or "EU's?"

The SM is partitioned ways internally, sharing common L1 instruction and L1 data caches. Each partition apparently has its own L0 instruction cache.

That's separate from the texture processing cores (TPCs), of which there's 54.

So it's either 1 core or 132 cores, depending on how you want to look at it.

# CPU L0 D-Cache

**Joe Zbiciak replied to your comment on an answer to: "The difference of number of cores between CPUs and GPUs is huge, then why CPU's performance is far better than GPUs?"**

L0 cache is its own level in the hierarchy, above the L1. L0 caches tend to be tiny compared to the L1.

The first processor I encountered with an L0 cache was an x86 clone that didn't make it to market. It had an L0 data cache that was a mere 256 bytes. It had a really short latency and acted essentially as an extended register file. Since x86 was a memory-register architecture, it made sense there.

We explored the possibility of an L0 cache in DSP designs while I worked there. Usually they were more complexity than their worth.

Where we saw potential value on the data side of our DSPs was in reducing load-to-use latency to more like 2 cycles. That typically helps pointer-chasers though, and those have poor locality, and so that's what nixed it there. The L0 cache would have used discrete registers rather than an SRAM.

We couldn't use it to save power on the data side, because we still needed to launch a tag lookup in the L1D in parallel to avoid hurting its load-to-use latency.

# CPU L0 I-Cache

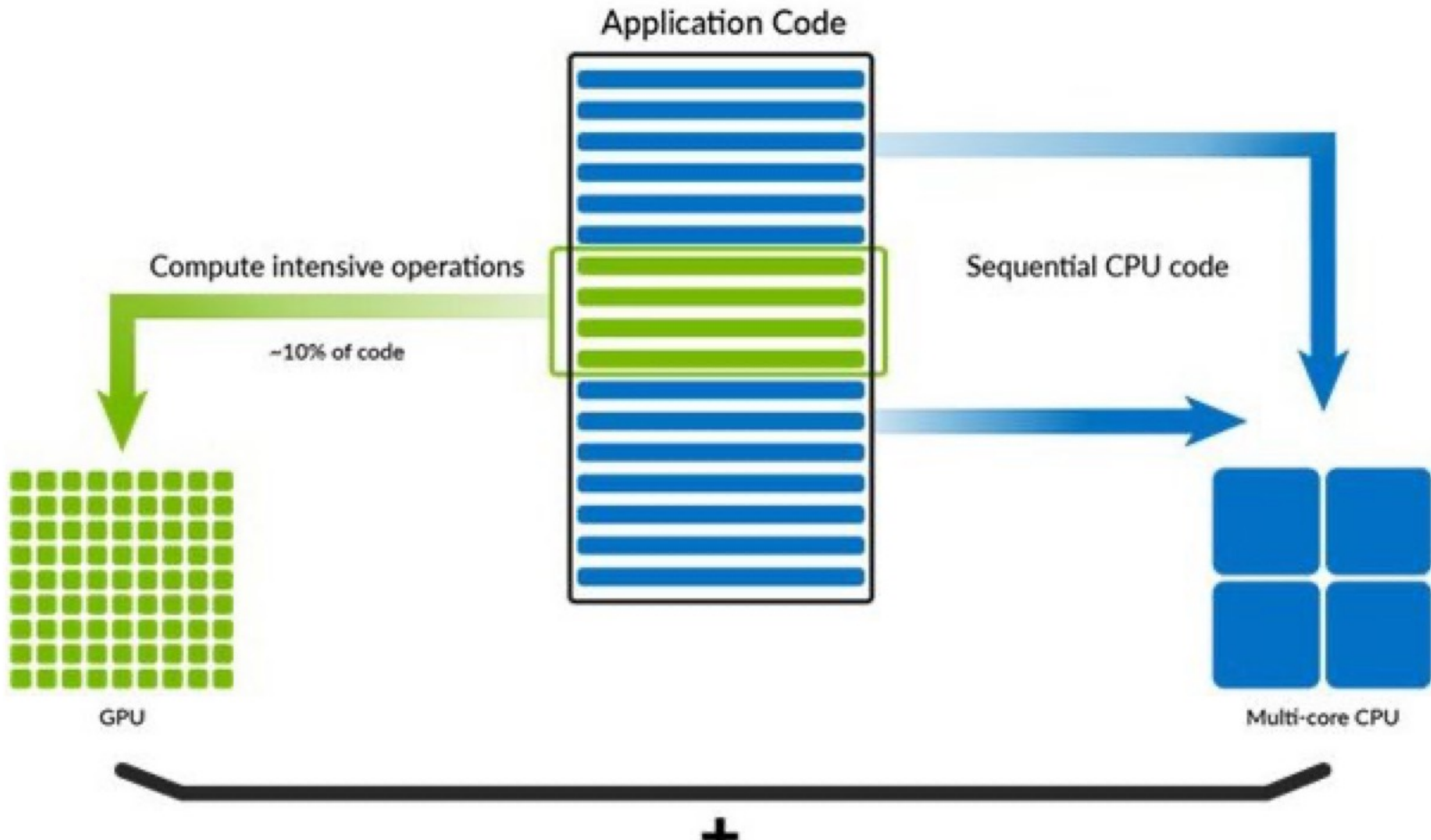
**Joe Zbiciak replied to your comment on an answer to: "The difference of number of cores between CPUs and GPUs is huge, then why CPU's performance is far better than GPUs?"**

The SPLOOP buffer in some of the later DSPs I worked on could be considered a form of L0 instruction cache, although it was fairly specialized and did more than cache things. It actually constructed prologs and epilogs for software pipelined loops!

Loop buffers are a big energy efficiency win on the instruction side.

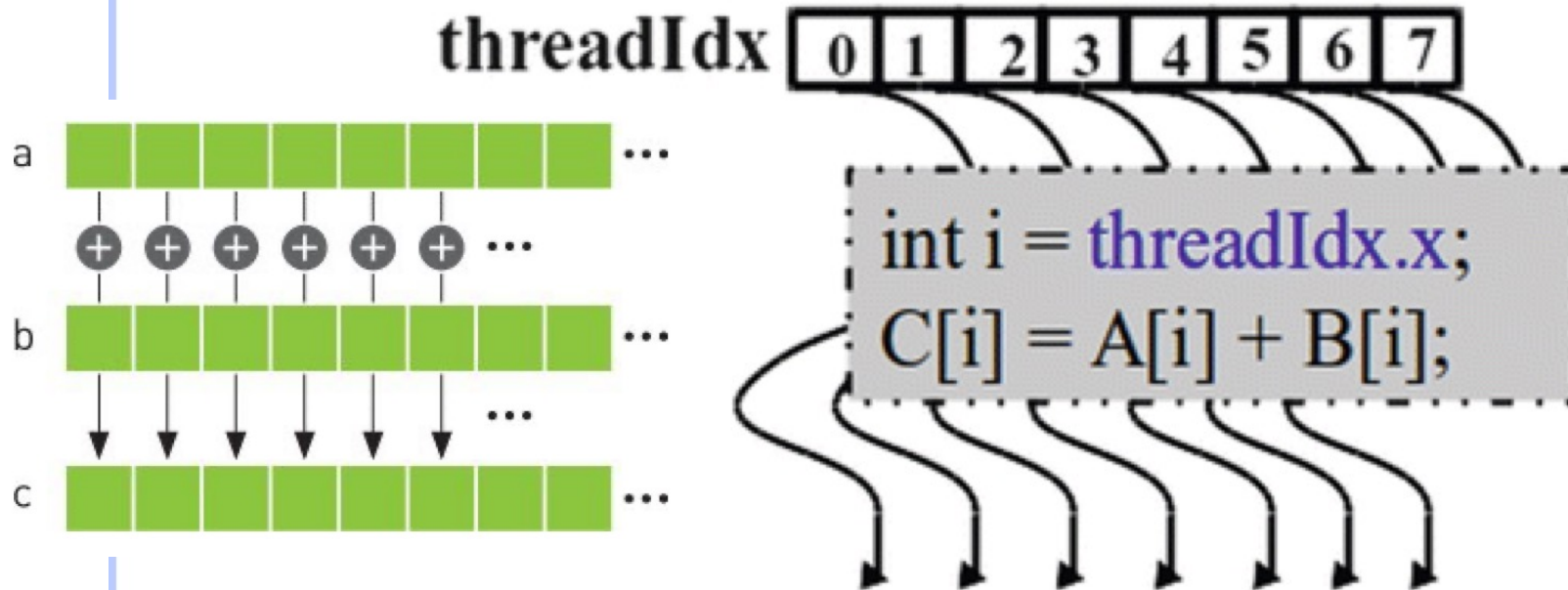
I don't know how big the L0 instruction cache is in the NVidia Ampere architecture. My guess is that it's big enough to cover a typical loop, times the number of threads actively making progress.

## How GPU Acceleration Works



# GPU

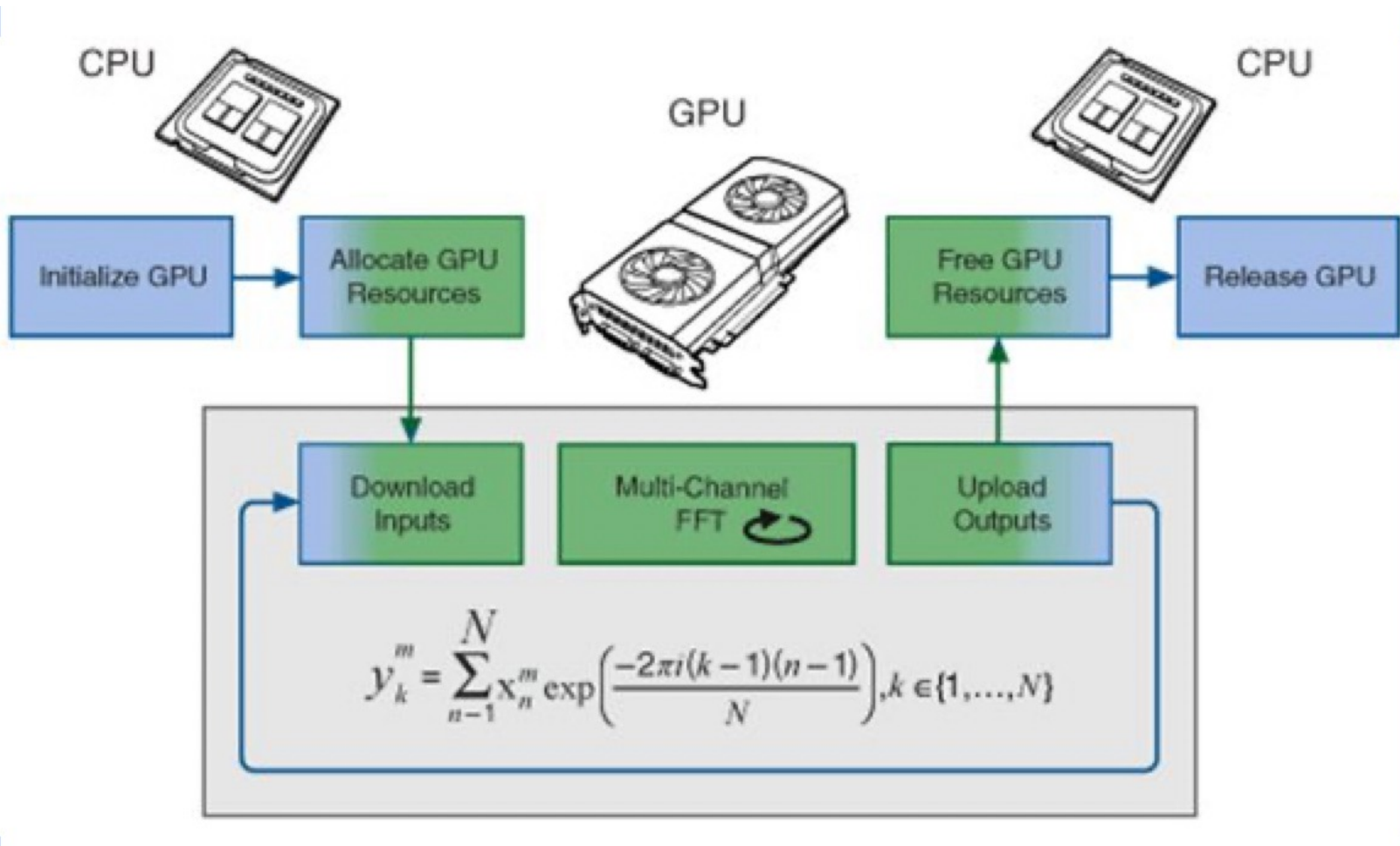
Each CUDA pipeline knows what to do through its unique thread-id



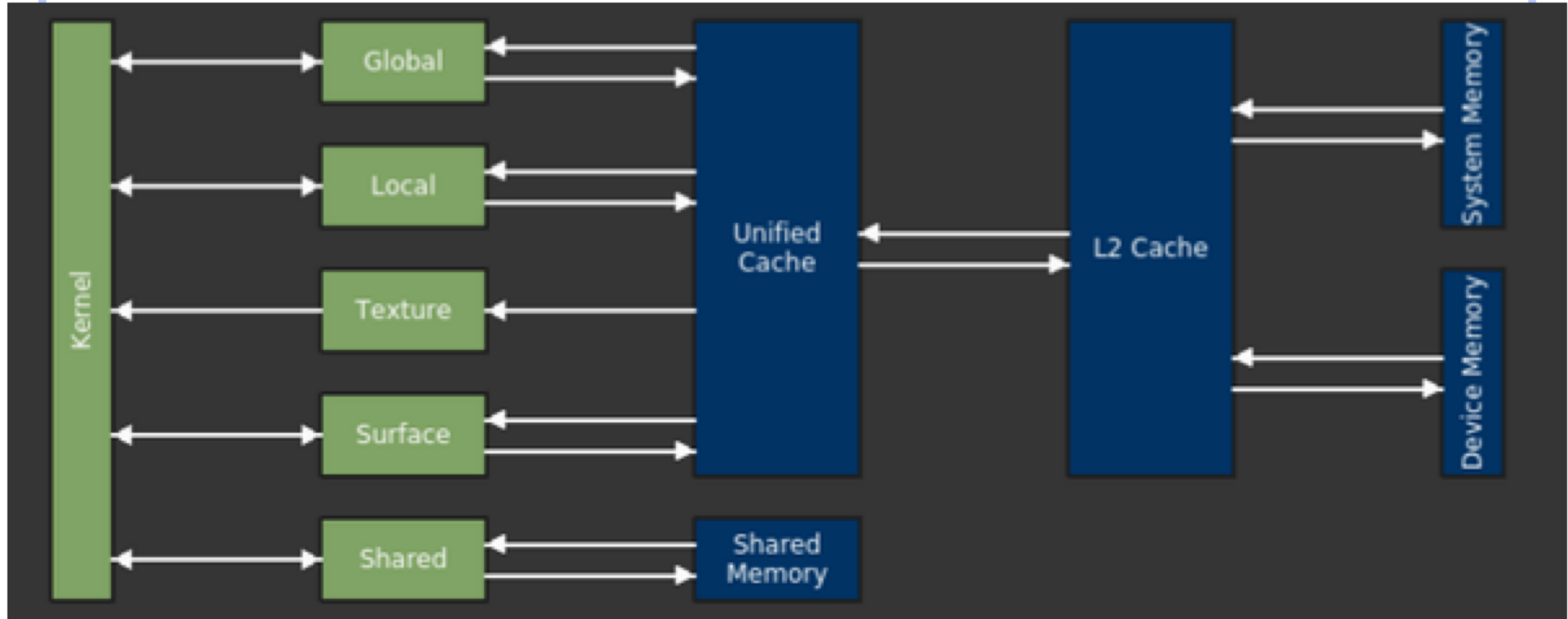
then they all finish a work much quicker than a serial code.

- Highly **vectorized** data
- Large **Thread** count

# GPU

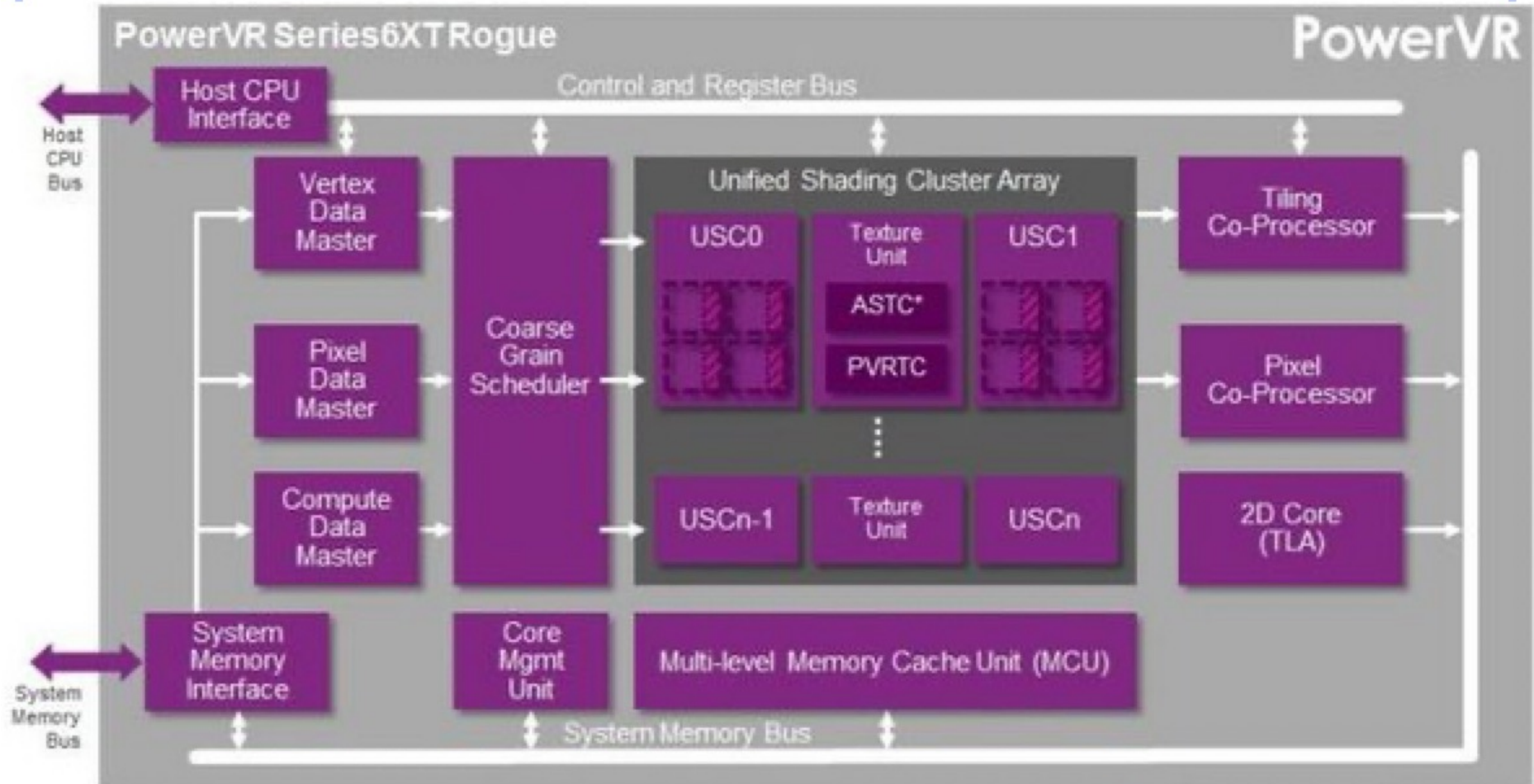


# GPU





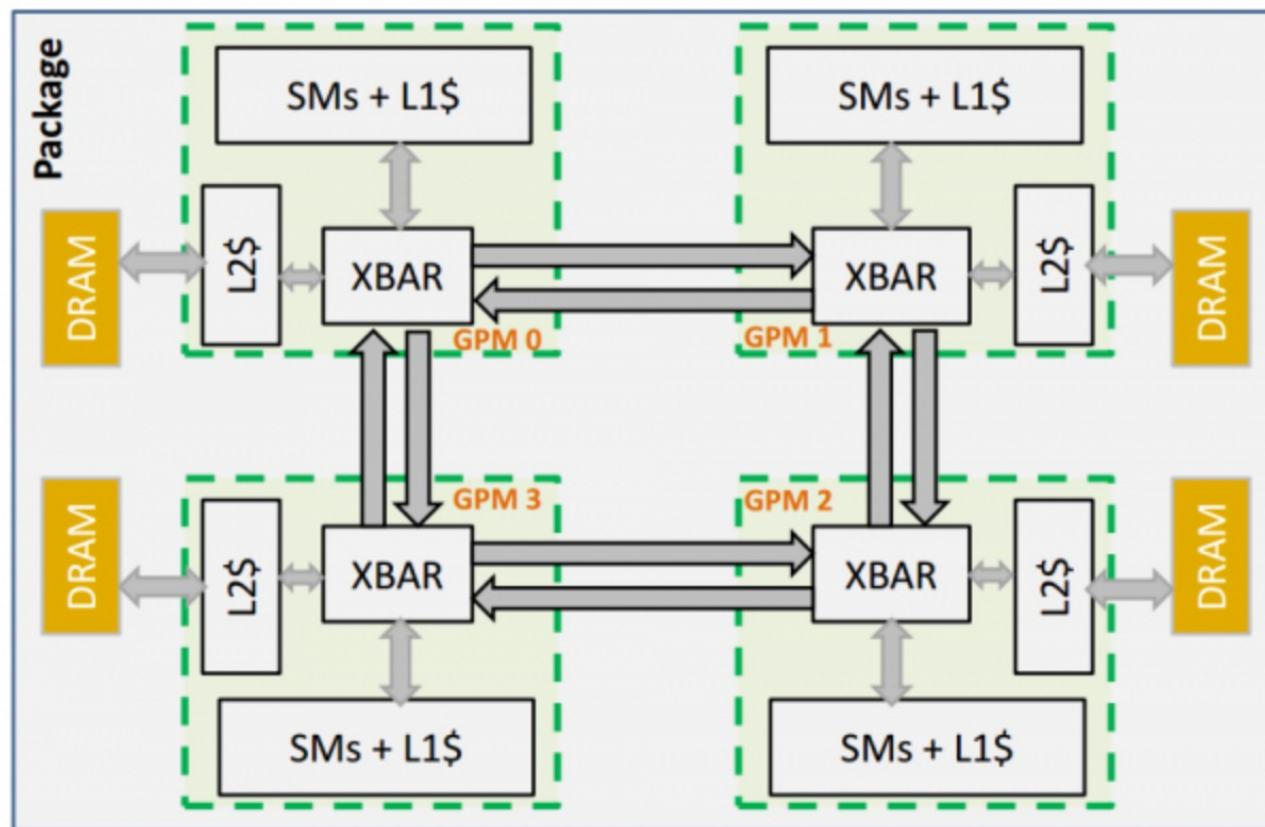
**DR JEFF**  
SOFTWARE  
INDIE APP DEVELOPER  
© Jeff Drobman  
2016-2023



# GPU Chipllets

## The Long Road to GPU Chipllets

The concept of splitting GPUs into discrete blocks and aggregating them together on-package predates common usage of the word “chipllet”, even though that’s what we’d call this approach today. Nvidia [performed a study](#) on the subject several years ago.



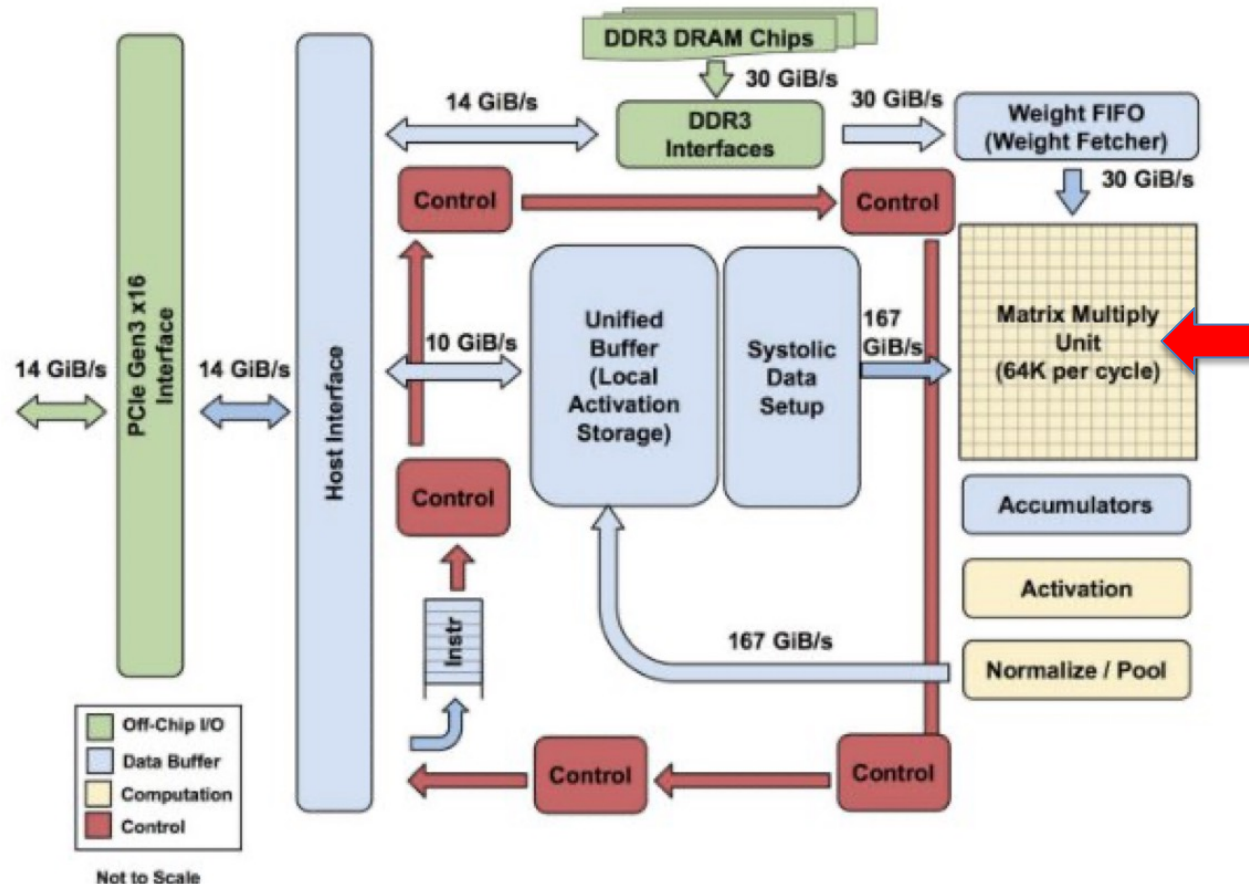
**Figure 3: Basic MCM-GPU architecture comprising four GPU modules (GPMs).**

# TPU



Kiryl Persianov

Here's a diagram of Google's TPU:



At its core, you find something that inspired by the heart and not the brain. It's called a "Systolic Array" described in 1982 in "[Why Systolic Architectures](#)":

# GPU Packaging



**Luc Boulesteix**

Updated April 11, 2020 · Upvoted by Thomas Martin, [Computer Hardware Enthusiast](#)



I wonder why this wasn't mentioned yet, but you used to be able to get NVIDIA chips in apple products.

A couple years back, around 2008-2009, NVIDIA sold chips that had a critical flaw, where the GPU would basically detach itself from the package/substrate. This was later coined as "bumpgate".

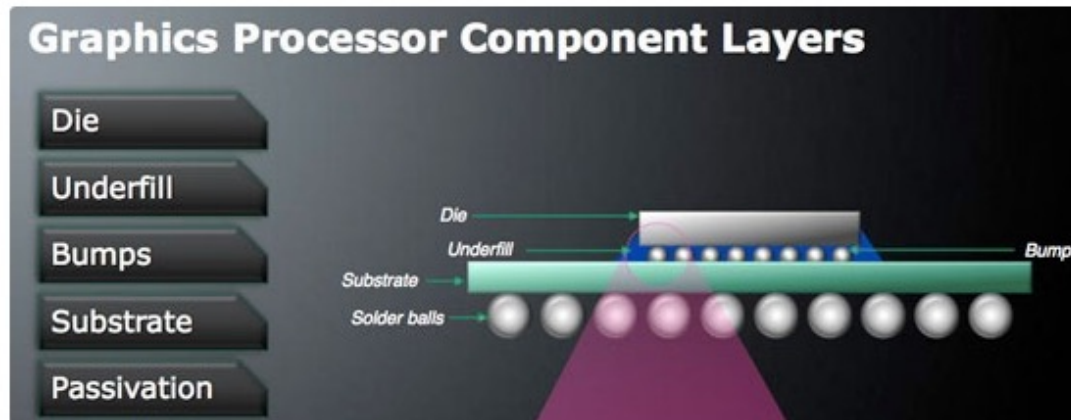


Image from Anandtech

Basically, NVIDIA changed the material used to connect the GPU die and substrate, which had better properties overall, but had the downside of having lower conductivity, meaning you need more of them to supply adequate power to the chip. NVIDIA didn't see this coming, meaning these connections, called bumps, had a tendency to expand more than expected under load, due to thermal expansion. This is obviously not good on its own, but NVIDIA also used a very stiff underfill under their chips, which didn't have appropriate deformation to absorb this bump expansion. This meant that these critical connections had a tendency to simply break in half, severing electrical

# CPU → GPU



**Georgios Deligiannoudis** · [Follow](#)

BEng in Computing, Edinburgh Napier University (Expected 2024) · Updated 2y



Related **Why do some games use your CPU far more than your GPU?**

A nice point to consider would be to think of what each part does.

How rendering works in a game is that the CPU prepares each frame, then the GPU does additional processing on it.

Your CPU will mostly take care of physics, the positions and behaviour of different entities (such as an enemy in a game) and hand all of that data over to the GPU. The GPU will then do the heavier tasks of rendering and post-processing with it all. Here's some examples from The Crew to illustrate.



# CPU → GPU

Below is the same frame but after the GPU has done all of its work.



Note the fact that lighting actually exists now, the skybox isn't missing, foliage quality is higher, the different materials of each object are taken into account, light sources such as the car's rear lights are acknowledged, textures are properly loaded, tessellation is used to give textures a three-dimensional look, water appears in the image and has its own interactions with light including reflections, and so on.

# CPU → GPU

To put it plainly, games that require less work from the GPU to produce a frame will inevitably mean that the CPU has to prepare more frames for the GPU to complete more often, as the GPU doesn't need to do as much. Thus, CPU usage will be higher, and the game's performance will depend on the CPU more. This is also the reason why gaming on a lower resolution or graphics settings increases how much performance will depend on your CPU.

Other games that are \*immensely\* CPU intensive could be Turn-Based Strategy titles such as Sid Meier's Civilization, where the actions of every nation excluding the player's have to be calculated almost entirely by the CPU. In such cases there are complex and advanced decision making (and other) algorithms that need to be executed which rely solely on your CPU and as such directly depend on it for performance.

# CPU → GPU

A 5GHz CPU can compute the locations of a complex wireframe vertex array very quickly, while a GPU has tools and storage specifically designed to render textures and lighting effects to bring the wireframe to life.



It is a constant volley between the CPU and GPU, exchanging real-time data and flipping at back and forth to each other. CPU creates vertex array—flips it to GPU. GPU renders frame—notifies CPU when finished. CPU updates position and creates new vertex array—and so on.

# CPU → GPU

A CPU core can only handle one or two threads per clock cycle, while a GPU core can also handle one thread per clock cycle. But when you link hundreds of them or thousands of them together, each managing a small section of the task, the overall job of rendering a screen of graphics is extremely quick. Rendering an image might require hundreds of steps or stages in all. But when each step only takes a few hundred nanoseconds to compute, the the 8-millisecond (8-million-nanosecond) frame time required for 120fps is very achievable.

It just occurred to me that I left out the most interesting part. 3D rendering uses a technique called DBSC. The **double-buffer swap chain** uses two frame buffers. One frame buffer functions as a canvas so that the renderer can "paint" its picture. When the picture is done, its image gets swapped to the monitor. The first frame buffer is erased and the renderer goes to work painting the next picture. The buffers are swapped. One always displays and image on the screen while the other is being processed.

# Section



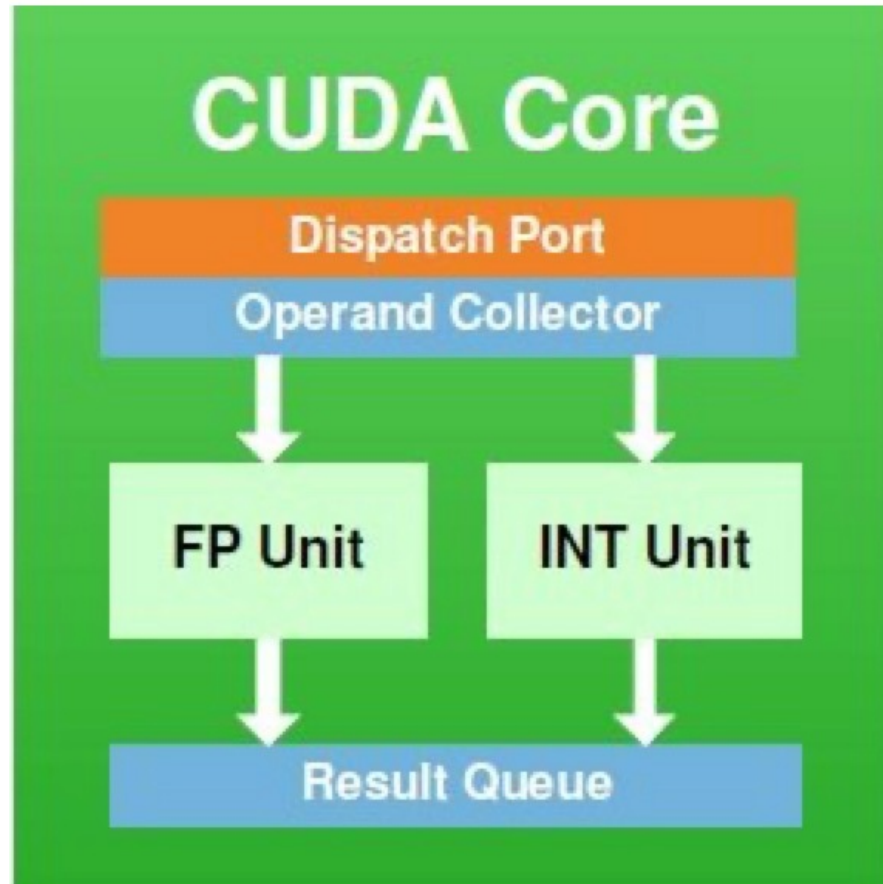
GPU  
CUDA

# GPU Coding w/CUDA



**Huseyin Tugrul Buyukisik** · [Follow](#)

B.S. in Numerical Analysis, Physics Engineer



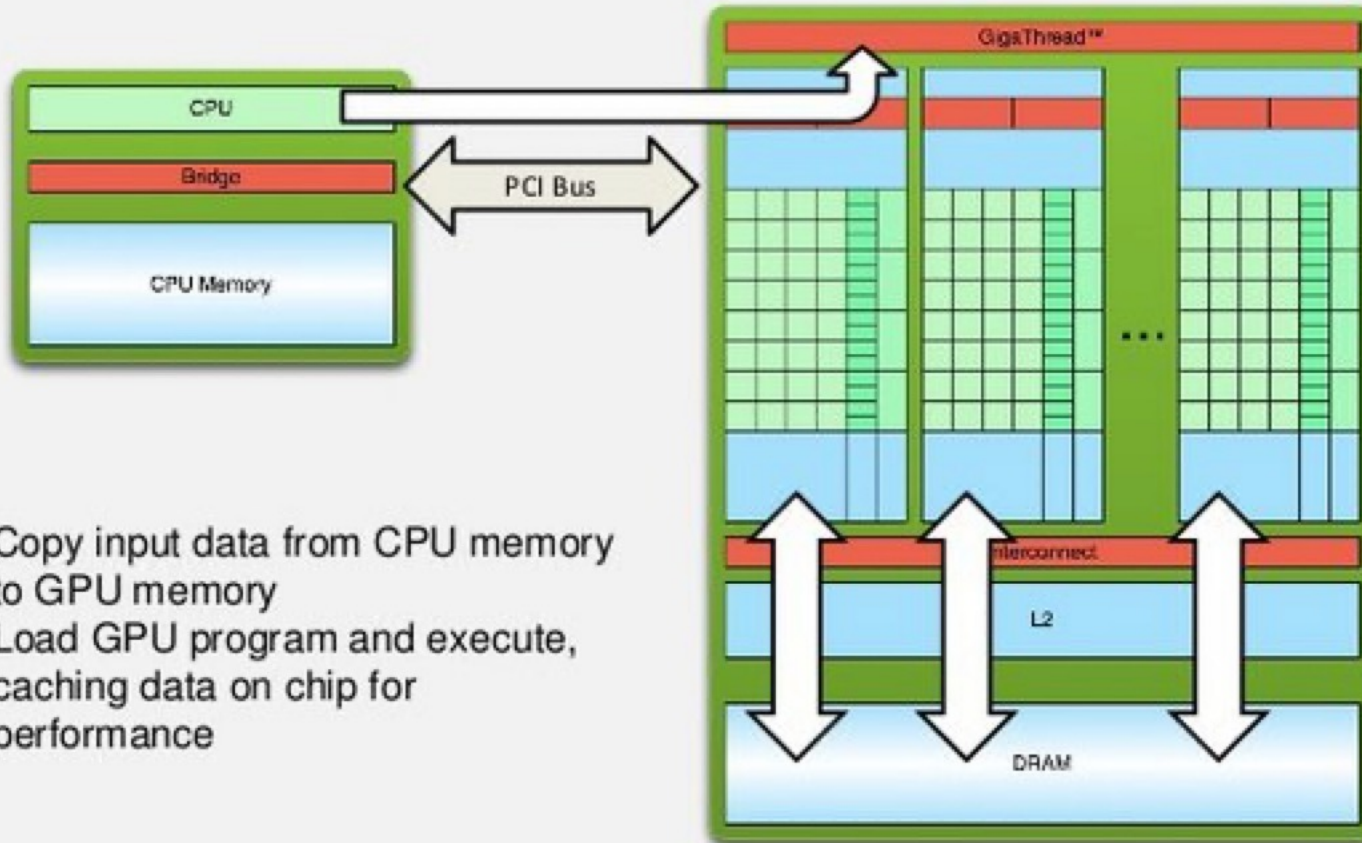
# GPU Coding w/CUDA



Huseyin Tugrul Buyukisik · Follow

B.S. in Numerical Analysis, Physics Engineer

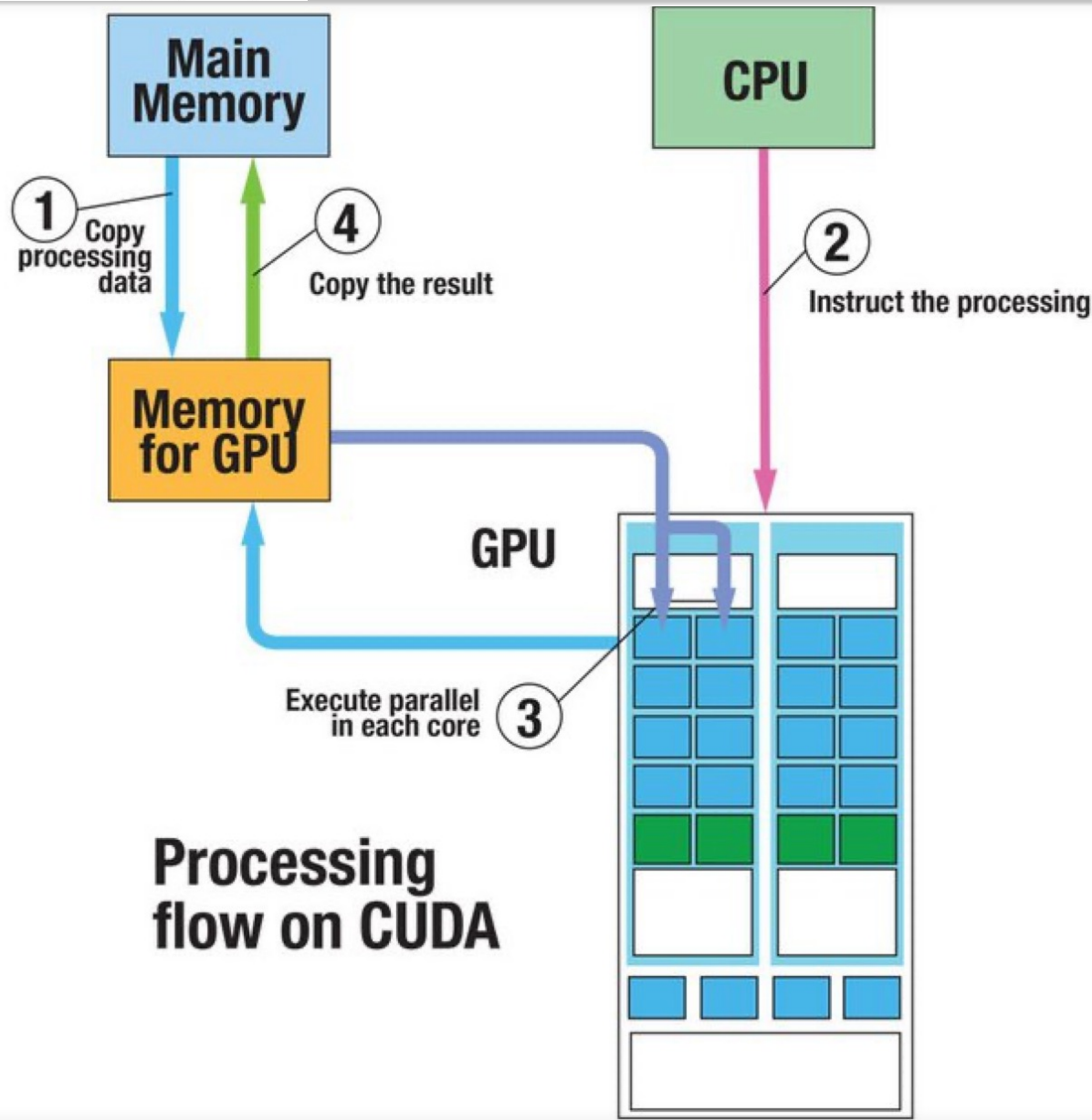
## Simple Processing Flow



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance



# GPU Coding w/CUDA

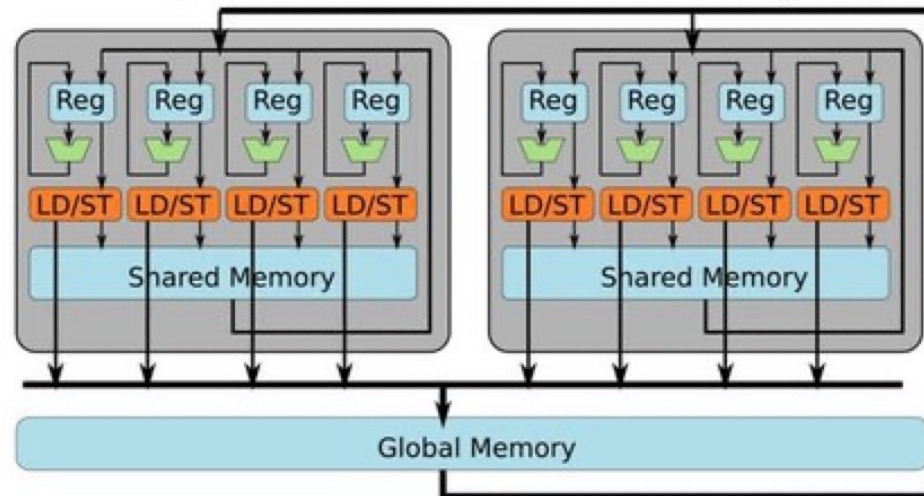




# GPU Coding w/CUDA

## CUDA Programmer's View of GPUs

A GPU contains multiple SIMD Units. All of them can access global memory.



With this coding style, write on a GTX480, run it on a RTX2080ti easily without changing code, in CUDA. Also for OpenCL, you can write code on a development machine that has AMD only, later can run it on an Nvidia machine easily.

A GPU is made of multiple "streaming multiprocessors".

SIMD/SIMT

A streaming multiprocessor is made of multiple "warp schedulers"

Each warp scheduler run commands on multiple SIMD/SIMT units.

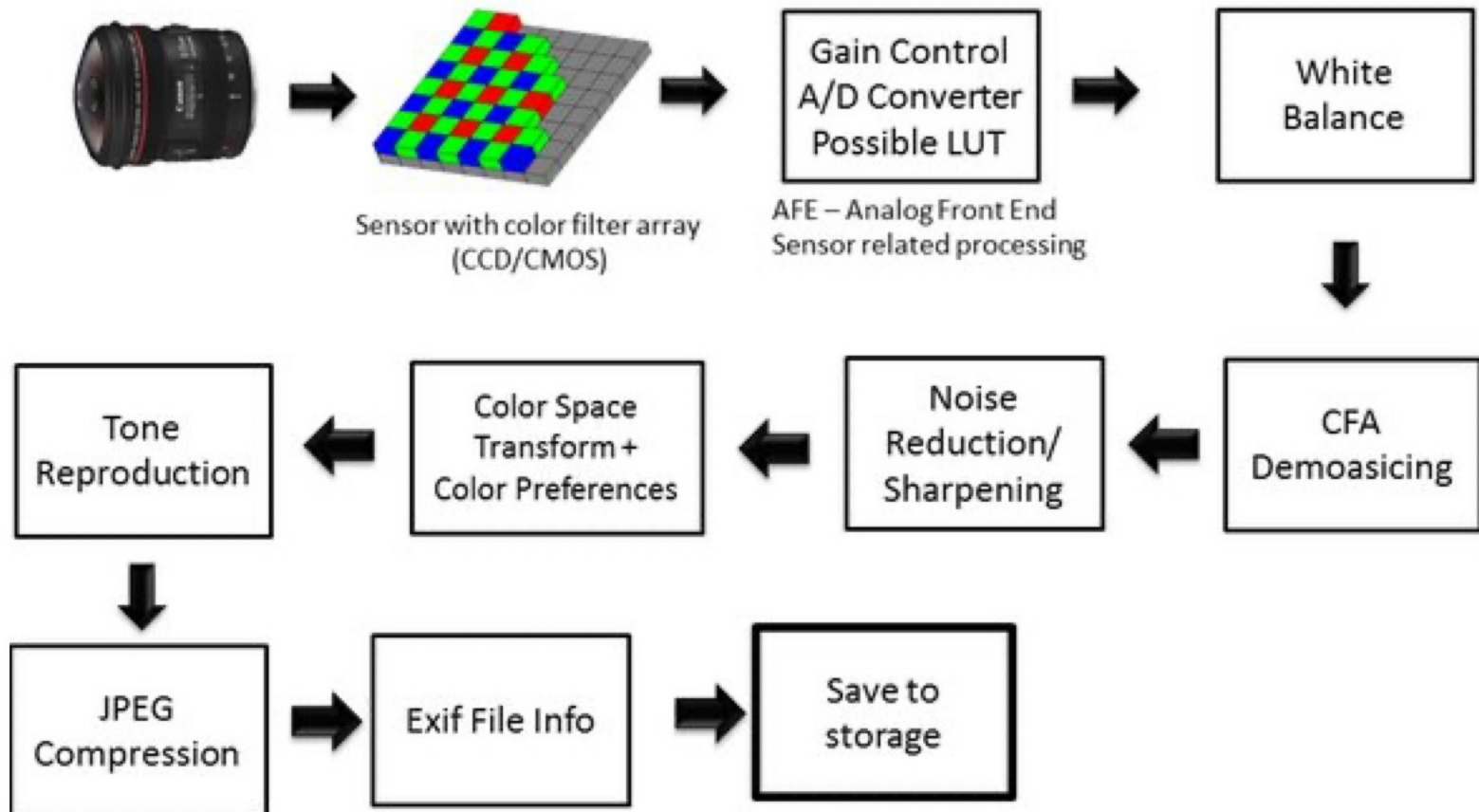
There are many SIMD/SIMT units per streaming multiprocessor.



# GPU Coding w/CUDA

- Example: image processing pipeline

## Pipeline for sRGB (JPEG)



# GPU Coding w/CUDA



**Karthikeyan Natarajan** · [Follow](#)

Works at NVIDIA · 3y



You can't run all of your python code in GPU. You have to write some parallel python code to run in CUDA GPU or use libraries which support CUDA GPU.

Your best bet for rewriting custom code is Numba. [GPU Accelerated Computing with Python](#) ↗

If it is for deep learning, use tensorflow, or pytorch or keras. Make sure to follow install instructions for CUDA GPU. (I would suggest to use nvidia docker images).

If it is for machine learning and ETL, use [RAPIDS](#) ↗.

# GPU Coding w/CUDA



**Huseyin Tugrul Buyukisik** · [Follow](#)

B.S. in Numerical Analysis, Physics Engineer (Graduated 2012) · Updated 1y



You have some options:

- 1- write a module in C++ (CUDA) and use its bindings in Python
- 2- use somebody else's work (who has done option 1)
- 3- write CUDA program in another language with some input/output. then call it from python commandline (using subprocess, etc...) and giving it i/o within python environment.
- 4-(not suggested) write a python interpreter that uses CUDA natively



# GPU Coding w/CUDA

- Workitem-level parallelization
  - Each workitem (CUDA thread, OpenCL workitem) computes its own independent data. It's actually data parallelism.
  - For embarrassingly parallel workloads
    - Example: vector-vector addition  $c=a+b$

1	Serial:	$c[i]$	=	$a[i]$	+	$b[i]$
2	Parallel:	$c[\text{threadId}]$	=	$a[\text{threadId}]$	+	$b[\text{threadId}]$



# GPU Coding w/CUDA

## Thread level

- Thread-block level parallelization
  - Each block of workitems work on a shared resources to complete an independent bigger work. Since each block can work independently from others, its also task parallelism.
  - For parallel workloads where some parts need to be synchronized at certain intervals
    - Example: reduction

```
1 Serial: sum = a[0] + a[1] + a[2] + a[3] + ... + a[N-1]
2 Parallel: for(unsigned int leap=n/2; n>0; n>=>1)
3             {
4                 if(threadId<n)
5                     sum[threadId]+=sum[threadId+leap];
6                 barrier(); // or __syncthreads()
7             }
8 summation = sum[0]
```

# GPU Coding w/CUDA

For example, **Numba** for Python is option-2. It is simple to use:

```
1 from numba import cuda, float32
2
3 # Controls threads per block and shared memory usage.
4 # The computation will be done on blocks of TPBxTPB elements.
5 TPB = 16
6
7 @cuda.jit
8 def fast_matmul(A, B, C):
9     # Define an array in the shared memory
10    # The size and type of the arrays must be known at compile
    time
11    sA = cuda.shared.array(shape=(TPB, TPB), dtype=float32)
12    sB = cuda.shared.array(shape=(TPB, TPB), dtype=float32)
13
14    x, y = cuda.grid(2)
15
16    tx = cuda.threadIdx.x
17    ty = cuda.threadIdx.y
18    bpg = cuda.gridDim.x    # blocks per grid
19
20    if x >= C.shape[0] and y >= C.shape[1]:
21        # Quit if (x, y) is outside of valid C boundary
22        return
```

# GPU Coding w/CUDA

```
24     # Each thread computes one element in the result matrix.
25     # The dot product is chunked into dot products of TPB-long
    vectors.
26     tmp = 0.
27     for i in range(bpg):
28         # Preload data into shared memory
29         sA[tx, ty] = A[x, ty + i * TPB]
30         sB[tx, ty] = B[tx + i * TPB, y]
31
32         # Wait until all threads finish preloading
33         cuda.syncthreads()
34
35         # Computes partial product on the shared memory
36         for j in range(TPB):
37             tmp += sA[tx, j] * sB[j, ty]
38
39         # Wait until all threads finish computing
40         cuda.syncthreads()
```

# GPU Coding w/CUDA

You write a "kernel" code (and compile for GPU). Send it to GPU. Also send data to GPU. Then call compiled kernel code on GPU with data attached to it. Then wait until it completes. After that, you get results from GPU.

Something like this:

```
1 MY_API void kernelIncrement(int * data)
2 {
3     int workItemId = threadIdx.x+blockIdx.x*blockDim.x;
4     data[workItemId]++;
5 }
6 cudaMemcpy(gpuData, hostData, n, cudaMemcpyHostToDevice);
7 kernelIncrement<<<128,128>>>(gpuData);
8 cudaMemcpy(hostData, gpuData, n, cudaMemcpyDeviceToHost);
```

Kernel here is a function that runs on each streaming pipeline of GPU (64–192 of such pipelines make a SM unit "streaming multiprocessor"). If its about graphics acceleration, kernel is called as "shader". If it is computing it is "\_\_kernel" or "kernel". These are just names of functions exposed to developers through APIs like OpenGL, OpenCL and CUDA. Each so-called "GPGPU pipeline" of GPU runs this same code.

# GPU Coding w/CUDA

COMP222



Huseyin Tugrul Buyukisik · [Follow](#)

B.S. in Numerical Analysis, Physics Engineer

Hardware is composed of SIMD/SIMT units but APIs expose them to developers like multi-core CPU so that its easier to write programs for developers. Normally on a CPU, you'd need to write vectorized code ( vectorized multiplications, vectorized branching, vectorized nested branching...) which makes it hard to tune for each new architecture. But APIs like Opencl and CUDA has online-compiling or jit-compiling capabilities that get your "non-SIMD, scalar-simple" code and apply it onto SIMD/SIMT units as if they were independent cores of a CPU (but actually they are pipelines sharing many things with other pipelines). Nvidia's Volta microarchitecture has made more progress on making these SIMD/SIMT pipelines even more independent of each other on hardware level.

# GPU Software



**Mats Petersson**, Worked with AMD Hardware Virtualization

Answered Fri



You install a GPU driver, which probably contains some selection of APIs for graphics: OpenGL, Vulkan, DirectX and compute: OpenCL and Cuda.

These API's are a way to build sets of operations for the GPU to perform - it could be as simple as a list of triangles and a colour information, that will draw a cube (and by rotating the view of that, we can make the cube spin).

Or a whole frame for a 3D Game, with many different objects, each with their individual texture-mapping, bump mapping and lighting, and a "background".

The drawing is done by a sequence of "drawcalls", one for each "item" in the frame - just one for the spinning cube, with different parameters to indicate what angle the cube should be at. In a game, you'd have one call for the monster walking towards you, one for each of the bullets you are firing off, one for each section of wall, the obstacles you are hiding behind or having to navigate around, etc.

The information on what to draw will be fed to the GPU by storing the data and draw call itself in the GPU memory or in main (aka host or CPU) memory, and informing the GPU "Next, do this".

# GPU Software

Modern GPUs allow "pixel shaders" and "fragment shaders", which is little snippets of programs that run for each pixel being drawn or each "fragment" (typically a triangle), and make modifications to that pixel or fragment. This allows the programmer a freedom to do almost anything - as long as it doesn't take too long! :)

In among the draw calls will be "markers" or "signals", which the GPU will signal back to the CPU that "Ok, I got to this point" - this allows the driver to tell the application that "Yup, done all the work to point X". This allows the application to "wait" for the GPU to finish stuff, but also to feed more work in before the GPU is finished.

For "compute" type tasks, there's a compiler and a language. Short (or long) bits of code gets translated to machine code for the GPU and loaded into GPU memory, with, and placed in the same type of memory as the draw-calls, with the data [or a reference to where it can be found] - for example two large matrices of numbers to do some calculations on. The GPU will then run that code supplied, and signal back when the work is done (similar to graphics). This uses the same resources of the GPU as the fragment shading - the only difference is that the result is not drawn onto the screen.

This is a VERY brief form of it. The driver package for a modern GPU contains tens of megabytes of code, and are very, very complicated. It takes months or years to understand how they work.

# GPU's

## How does a GPU-enabled computer know to use the GPU while a game is being played?



**Jeff Drobman** · just now

I am an expert witness for technology

how does any computer know which cores of any kind to use? Cores are assigned by an OS when multitasking like in Windows or Linux or Mac OS. For games, I presume each game has a "driver" process responsible for core and thread assignment. Threads in CPU's are also handled automatically in hardware via MT/SMT. GPU's are "SIMD" style vector processors, and data must be in parallel as vectors.

# Section

---

## GPU P&H Ch 6, 9:

# Parallel Processing: GPU

P&H Ch 6

COMP 122: Computer  
Architecture and  
Assembly Language  
Spring 2020

## 6.6 Introduction to graphics processing units

(This section concentrates on using GPUs for computing. To see how GPU computing combines with the traditional role of graphics acceleration, see COD Appendix C (Graphics and Computing GPUs).)

Here are some of the key characteristics as to how GPUs vary from CPUs:

- GPUs are accelerators that supplement a CPU, so they do not need to be able to perform all the tasks of a CPU. This role allows them to dedicate all their resources to graphics. It's fine for GPUs to perform some tasks poorly or not at all, given that in a system with both a CPU and a GPU, the CPU can do them if needed.
- The GPU problems sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes.

These differences led to different styles of architecture:

- Perhaps the biggest difference is that GPUs do not rely on multilevel caches to overcome the long latency to memory, as do CPUs. Instead, GPUs rely on hardware multithreading (COD Section 6.4 (Hardware multithreading)) to hide the latency to memory. That is, between the time of a memory request and the time that data arrives, the GPU executes hundreds or thousands of threads that are independent of that request.
- The GPU memory is thus oriented toward bandwidth rather than latency. There are even special graphics DRAM chips for GPUs that are wider and have higher bandwidth than DRAM chips for CPUs. In addition, GPU memories have traditionally had smaller main memories than conventional microprocessors. In 2013, GPUs typically have 4 to 6 GiB or less, while CPUs have 32 to 256 GiB. Finally, keep in mind that for general-purpose computation, you must include the time to transfer the data between CPU memory and GPU memory, since the GPU is a coprocessor.
- Given the reliance on many threads to deliver good memory bandwidth, GPUs can accommodate many parallel processors (MIMD) as well as many threads. Hence, each GPU processor is more highly multithreaded than a typical CPU, plus they have more processors.

# GPU Parallel Processing

P&H Ch 6

## Hardware/Software Interface

Although GPUs were designed for a narrower set of applications, some programmers wondered if they could specify their applications in a form that would let them tap the high potential performance of GPUs. After tiring of trying to specify their problems using the graphics APIs and languages, they developed C-inspired programming languages to allow them to write programs directly for the GPUs. An example is NVIDIA's CUDA (Compute Unified Device Architecture), which enables the programmer to write C programs to execute on GPUs, albeit with some restrictions. COD Appendix C (Graphics and Computing GPUs) gives examples of CUDA code. (OpenCL is a multi- company initiative to develop a portable programming language that provides many of the benefits of CUDA.)

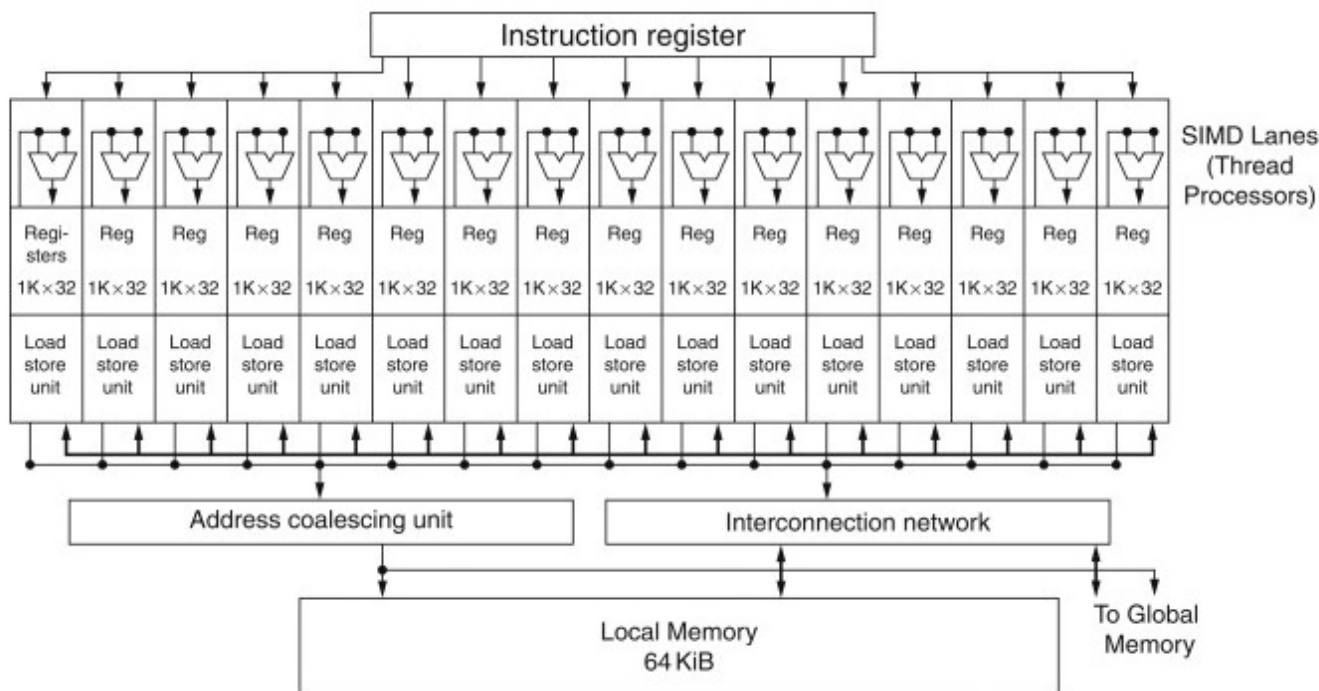
NVIDIA decided that the unifying theme of all these forms of parallelism is the CUDA Thread. Using this lowest level of parallelism as the programming primitive, the compiler and the hardware can gang thousands of CUDA threads together to utilize the various styles of parallelism within a GPU: multithreading, MIMD, SIMD, and instruction-level parallelism. These threads are blocked together and executed in groups of 32 at a time. A multithreaded processor inside a GPU executes these blocks of threads, and a GPU consists of 8 to 32 of these multithreaded processors.

# SIMD

## P&H Ch 6

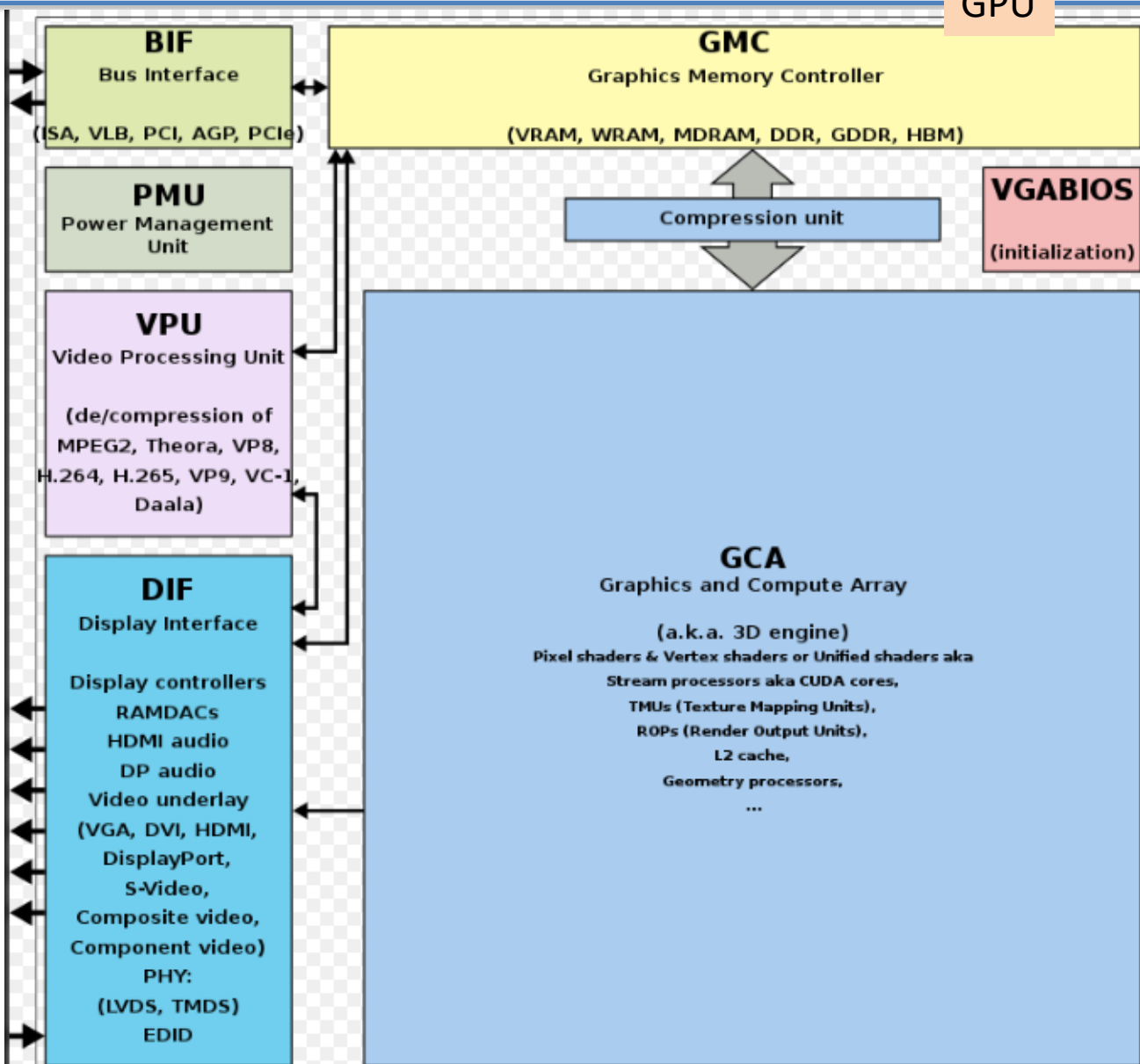
Figure 6.6.1: Simplified block diagram of the datapath of a multithreaded SIMD Processor (COD Figure 6.9).

It has 16 SIMD lanes. The SIMD Thread Scheduler has many independent SIMD threads that it chooses from to run on this processor.



# GPU Architecture

GPU



# Parallel Processing

Wikipedia

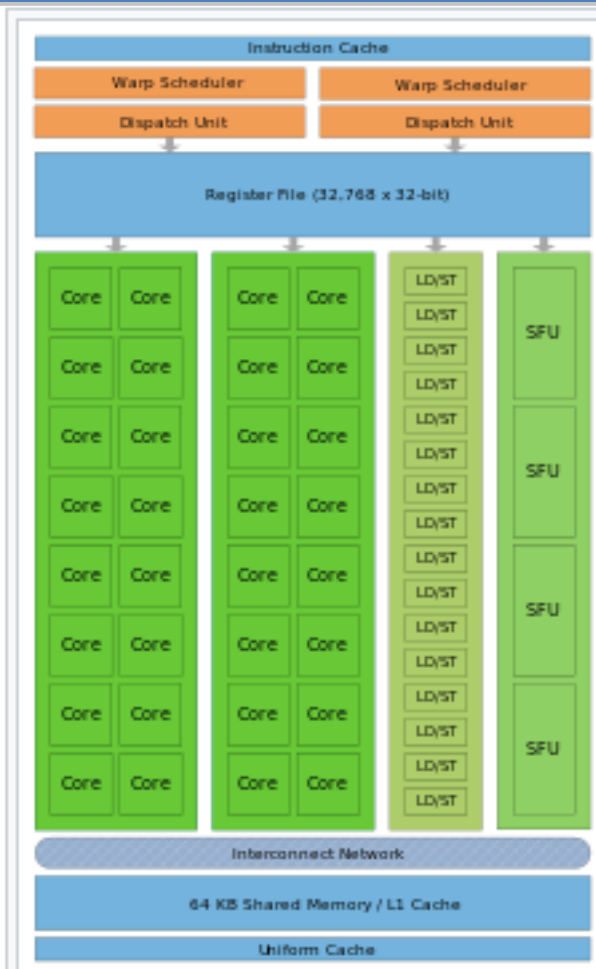


Fig. 1. NVIDIA Fermi architecture  
Convention in figures: orange - scheduling and dispatch; green - execution; light blue - registers and caches.

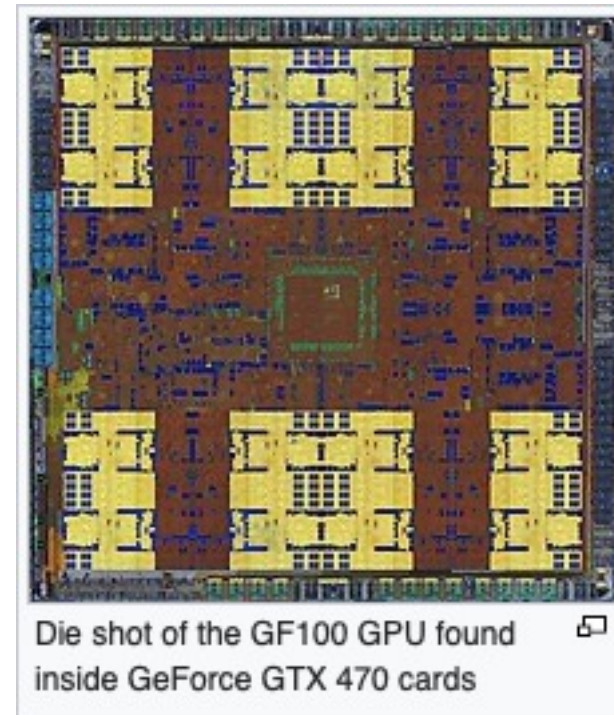


Figure 6.6.4: Quick guide to GPU terms (COD Figure 6.12).

Type	More descriptive name	Closest old term outside of GPUs	Official CUDA/ NVIDIA GPU term	Book definition
Program abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel.
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via Local Memory.
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register.
Machine object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it contains just SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask.
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes.
Processing hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors.
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors.
	SIMD Thread Scheduler	Thread scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution.
	SIMD Lane	Vector lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask.
Memory hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU.
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors.
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full thread block (body of vectorized loop).

Figure 6.6.3: Similarities and differences between multicore with Multimedia SIMD extensions and recent GPUs (COD Figure 6.11).

Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Largest cache size	8 MiB	0.75 MiB
Size of memory address	64-bit	64-bit
Size of main memory	8 GiB to 256 GiB	4 GiB to 6 GiB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Cache coherent	Yes	No

## Why CUDA and GPU computing?

This uniform and scalable array of processors invites a new model of programming for the GPU. The large amount of floating-point processing power in the GPU processor array is very attractive for solving nongraphics problems. Given the large degree of parallelism and the range of scalability of the processor array for graphics applications, the programming model for more general computing must express the massive parallelism directly, but allow for scalable execution.

*GPU computing* is the term coined for using the GPU for computing via a parallel programming language and API, without using the traditional graphics API and graphics pipeline model. This is in contrast to the earlier *General Purpose computation on GPU (GPGPU)* approach, which involves programming the GPU using a graphics API and graphics pipeline to perform nongraphics tasks.

**GPU computing:** Using a GPU for computing via a parallel programming language and API.

**GPGPU:** Using a GPU for general-purpose computation via a traditional graphics API and graphics pipeline.

*Compute Unified Device Architecture (CUDA)* is a scalable parallel programming model and software platform for the GPU and other parallel processors that allows the programmer to bypass the graphics API and graphics interfaces of the GPU and simply program in C or C++. The CUDA programming model has an SPMD (single-program multiple data) software style, in which a programmer writes a program for one thread that is instanced and executed by many threads in parallel on the multiple processors of the GPU. In fact, CUDA also provides a facility for programming multiple CPU cores as well, so CUDA is an environment for writing parallel programs for the entire heterogeneous computer system.

**CUDA:** A scalable parallel programming model and language based on C/C++. It is a parallel programming platform for GPUs and multicore CPUs.

## GPU unifies graphics and computing

With the addition of CUDA and GPU computing to the capabilities of the GPU, it is now possible to use the GPU as both a graphics processor and a computing processor at the same time, and to combine these uses in visual computing applications. The underlying processor architecture of the GPU is exposed in two ways: first, as implementing the programmable graphics APIs, and second, as a massively parallel processor array programmable in C/C++ with CUDA.

Although the underlying processors of the GPU are unified, it is not necessary that all of the SPMD thread programs are the same. The GPU can run graphics shader programs for the graphics aspect of the GPU, processing geometry, vertices, and pixels, and also run thread programs in CUDA.

The GPU is truly a versatile multiprocessor architecture, supporting a variety of processing tasks. GPUs are excellent at graphics and visual computing as they were specifically designed for these applications. GPUs are also excellent at many general-purpose throughput applications that are "first cousins" of graphics, in that they perform a lot of parallel work, as well as having a lot of regular problem structure. In general, they are a good match to data-parallel problems (see COD Chapter 6 (Parallel Processor from Client to Cloud)), particularly large problems, but less so for less regular, smaller problems.

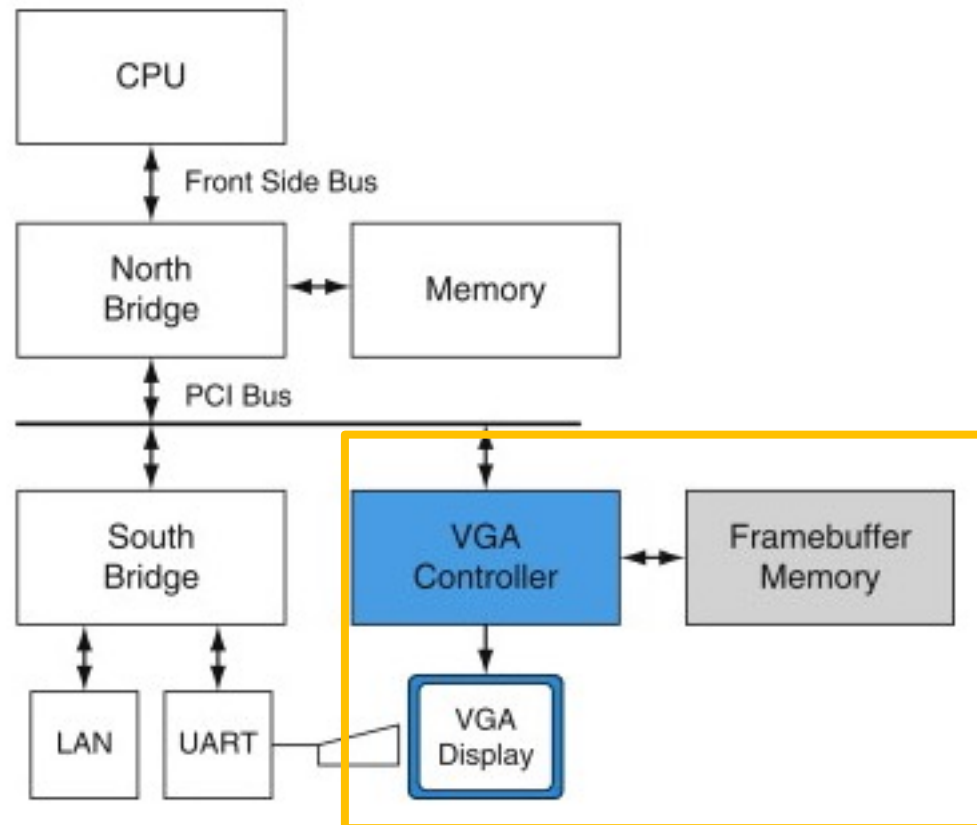
## GPU visual computing applications

Visual computing includes the traditional types of graphics applications plus many new applications. The original purview of a GPU was "anything with pixels," but it now includes many problems without pixels but with regular computation and/or data structure. GPUs are effective at 2D and 3D graphics, since that is the purpose for which they are designed. Failure to deliver this application performance would be fatal. 2D and 3D graphics use the GPU in its "graphics mode," accessing the processing power of the GPU through the graphics APIs, OpenGL™, and DirectX™. Games are built on the 3D graphics processing capability.

Beyond 2D and 3D graphics, image processing and video are important applications for GPUs. These can be implemented using the graphics APIs or as computational programs, using CUDA to program the GPU in computing mode. Using CUDA, image processing is simply another data-parallel array program. To the extent that the data access is regular and there is good locality, the program will be efficient. In practice, image processing is a very good application for GPUs. Video processing, especially encode and decode (compression and decompression according to some standard algorithms), is quite efficient.

Figure 9.2.1: Historical PC (COD Figure C.2.1).

VGA controller drives graphics display from framebuffer memory.

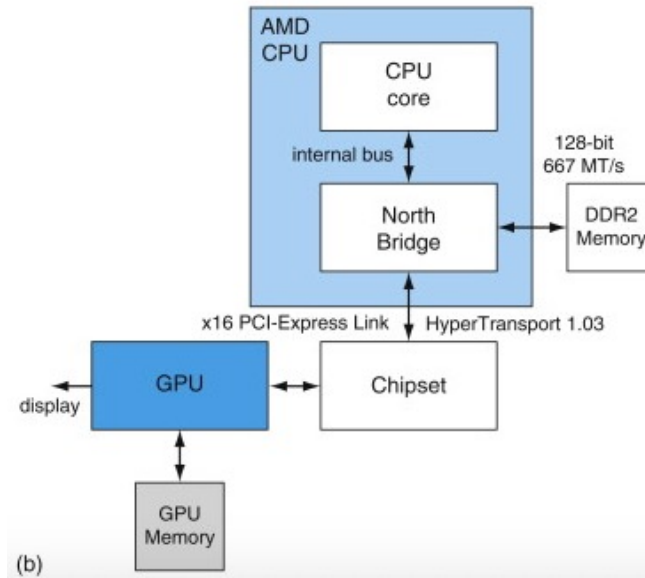
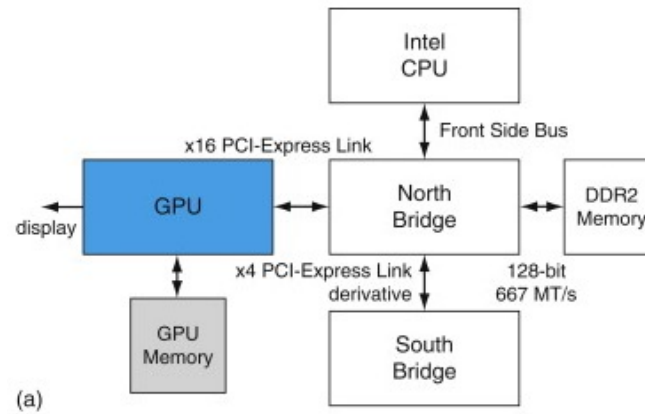


# GPU

## P&H Ch 9

Figure 9.2.2: Contemporary PCs with Intel and AMD CPUs (COD Figure C.2.2).

See COD Chapter 6 (Parallel Processor from Client to Cloud) for an explanation of the components and interconnects in this figure.



**PCI-Express (PCIe):** A standard system I/O interconnect that uses point-to-point links. Links have a configurable number of lanes and bandwidth.

A low-cost variation on these systems, a *unified memory architecture (UMA)* system, uses only CPU system memory, omitting GPU memory from the system. These systems have relatively low performance GPUs, since their achieved performance is limited by the available system memory bandwidth and increased latency of memory access, whereas dedicated GPU memory provides high bandwidth and low latency.

**Unified memory architecture (UMA):** A system architecture in which the CPU and GPU share a common system memory.

A high performance system variation uses multiple attached GPUs, typically two to four working in parallel, with their displays daisy-chained. An example is the NVIDIA SLI (scalable link interconnect) multi-GPU system, designed for high performance gaming and workstations.

The next system category integrates the GPU with the north bridge (Intel) or chipset (AMD) with and without dedicated graphics memory.

## Graphics logical pipeline

The graphics logical pipeline is described in COD Section C.3 (Programming GPUs). The figure below illustrates the major processing stages, and highlights the important programmable stages (vertex, geometry, and pixel shader stages).

Figure 9.2.3: Graphics logical pipeline (COD Figure C.2.3).

Programmable graphics shader stages are blue, and fixed-function blocks are white.

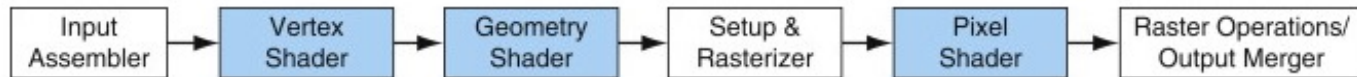


Figure 9.2.4: Logical pipeline mapped to physical processors (COD Figure C.2.4).

The programmable shader stages execute on the array of unified processors, and the logical graphics pipeline dataflow recirculates through the processors.

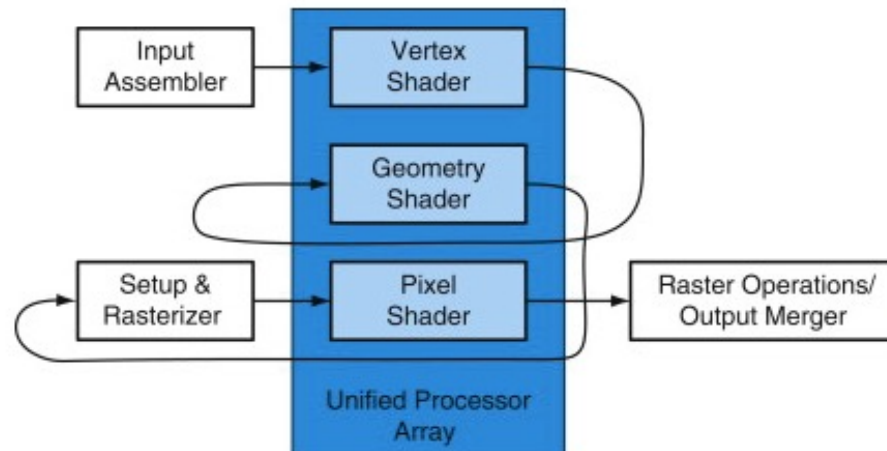
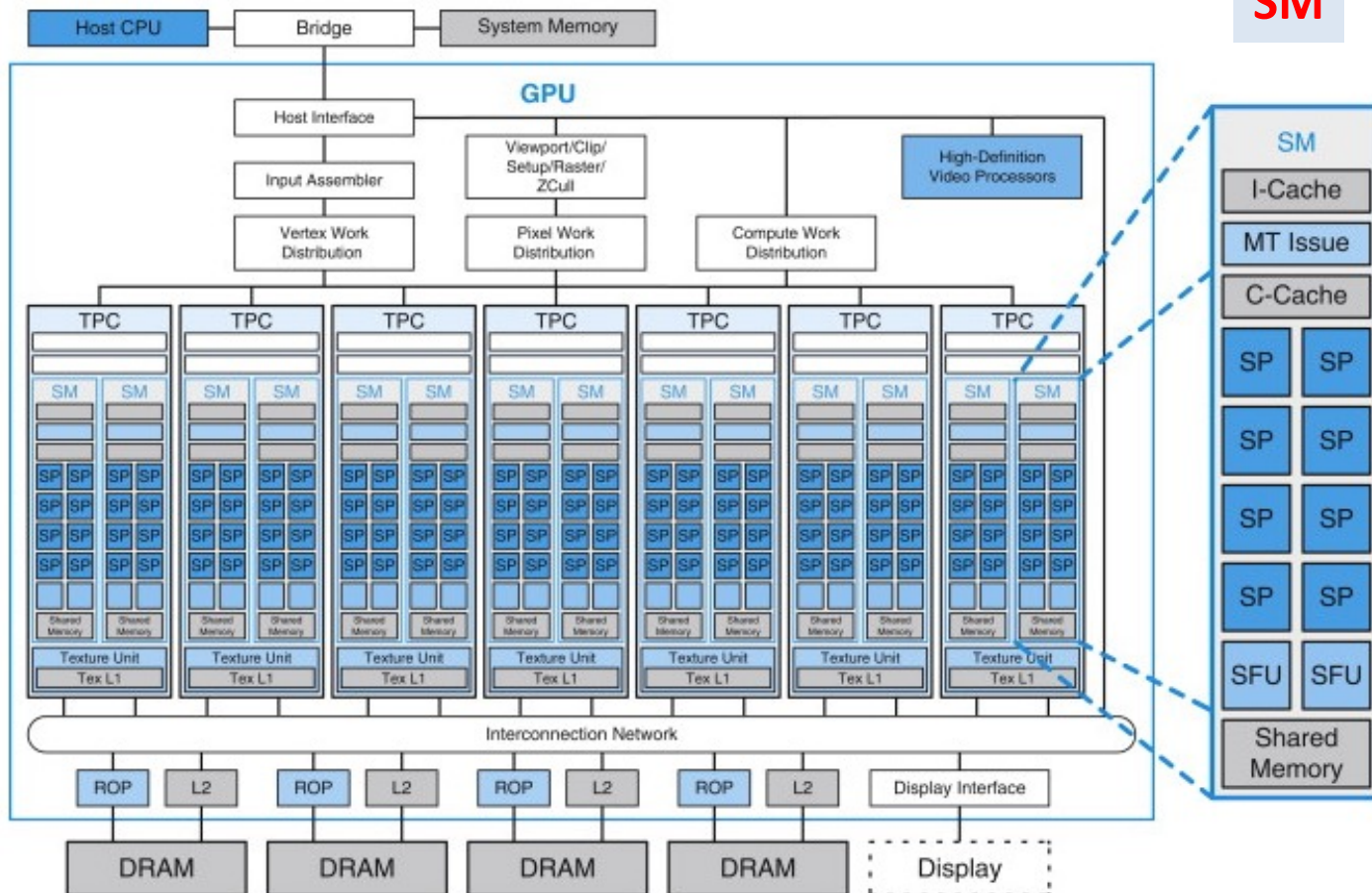


Figure 9.2.5: Basic unified GPU architecture (COD Figure C.2.5).

Example GPU with 112 streaming processor (SP) cores organized in 14 streaming multiprocessors (SMs); the cores are highly multithreaded. It has the basic Tesla architecture of an NVIDIA GeForce 8800. The processors connect with four 64-bit-wide DRAM partitions via an interconnection network. Each SM has eight SP cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory.



## 9.3 Programming GPUs

(Original section<sup>1</sup>)

Programming multiprocessor GPUs is qualitatively different than programming other multiprocessors like multicore CPUs. GPUs provide two to three orders of magnitude more thread and data parallelism than CPUs, scaling to hundreds of processor cores and tens of thousands of concurrent threads. GPUs continue to increase their parallelism, doubling it about every 12 to 18 months, enabled by *Moore's law [1965]* of increasing integrated circuit density and by improving architectural efficiency. To span the wide price and performance range of different market segments, different GPU products implement widely varying numbers of processors and threads. Yet users expect games, graphics, imaging, and computing applications to work on any GPU, regardless of how many parallel threads it executes or how many parallel processor cores it has, and they expect more expensive GPUs (with more threads and cores) to run applications faster. As a result, GPU programming models and application programs are designed to scale transparently to a wide range of parallelism.

The driving force behind the large number of parallel threads and cores in a GPU is real-time graphics performance—the need to render complex 3D scenes with high resolution at interactive frame rates, at least 60 frames per second. Correspondingly, the scalable programming models of graphics shading languages such as Cg (C for graphics) and HLSL (high-level shading language) are designed to exploit large degrees of parallelism via many independent parallel threads and to scale to any number of processor cores. The CUDA scalable parallel programming model similarly enables general parallel computing applications to leverage large numbers of parallel threads and scale to any number of parallel processor cores, transparently to the application.

In these scalable programming models, the programmer writes code for a single thread, and the GPU runs myriad thread instances in parallel. Programs thus scale transparently over a wide range of hardware parallelism. This simple paradigm arose from graphics APIs and shading languages that describe how to shade one vertex or one pixel. It has remained an effective paradigm as GPUs have rapidly increased their parallelism and performance since the late 1990s.

## Programming real-time graphics

APIs have played an important role in the rapid, successful development of GPUs and processors. There are two primary standard graphics APIs: *OpenGL* and *Direct3D*, one of the Microsoft DirectX multimedia programming interfaces. OpenGL, an open standard, was originally proposed and defined by Silicon Graphics Incorporated. The ongoing development and extension of the OpenGL standard [Segal and Akeley, 2006], [Kessenich, 2006] is managed by Khronos, an industry consortium. Direct3D [Blythe, 2006], a de facto standard, is defined and evolved forward by Microsoft and partners. OpenGL and Direct3D are similarly structured, and continue to evolve rapidly with GPU hardware advances. They define a logical graphics processing pipeline that is mapped onto the GPU hardware and processors, along with programming models and languages for the programmable pipeline stages.

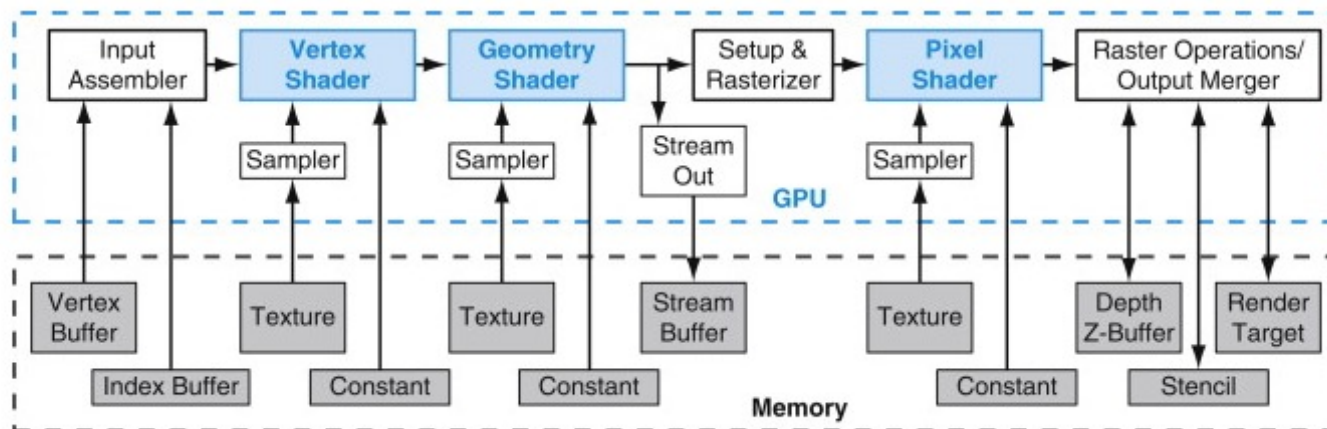
**OpenGL:** An open-standard graphics API.

**Direct3D:** A graphics API defined by Microsoft and partners.

**Texture:** A 1D, 2D, or 3D array that supports sampled and filtered lookups with interpolated coordinates.

Figure 9.3.1: Direct3D 10 graphics pipeline (COD Figure C.3.1).

Each logical pipeline stage maps to GPU hardware or to a GPU processor. Programmable shader stages are blue, fixed-function blocks are white, and memory objects are gray. Each stage processes a vertex, geometric primitive, or pixel in a streaming dataflow fashion.



## The CUDA paradigm

CUDA is a minimal extension of the C and C++ programming languages. The programmer writes a serial program that calls parallel *kernels*, which may be simple functions or full programs. A kernel executes in parallel across a set of parallel threads. The programmer organizes these threads into a hierarchy of thread blocks and grids of thread blocks. A *thread block* is a set of concurrent threads that can cooperate among themselves through barrier synchronization and through shared access to a memory space private to the block. A *grid* is a set of thread blocks that may each be executed independently and thus may execute in parallel.

**Kernel:** A program or function for one thread, designed to be executed by many threads.

**Thread block:** A set of concurrent threads that execute the same thread program and may cooperate to compute a result.

**Grid:** A set of thread blocks that execute the same kernel program.

When invoking a kernel, the programmer specifies the number of threads per block and the number of blocks comprising the grid. Each thread is given a unique *thread ID* number **threadIdx** within its thread block, numbered **0, 1, 2, ..., blockDim-1**, and each thread block is given a unique *block ID* number **blockIdx** within its grid. CUDA supports thread blocks containing up to 512 threads. For convenience, thread blocks and grids may have 1, 2, or 3 dimensions, accessed via **.x**, **.y**, and **.z** index fields.

## Programming parallel computing applications

CUDA, Brook, and CAL are programming interfaces for GPUs that are focused on data parallel computation rather than on graphics. CAL (Compute Abstraction Layer) is a low-level assembler language interface for AMD GPUs. Brook is a streaming language adapted for GPUs by Buck et al. [2004]. CUDA, developed by NVIDIA [2007], is an extension to the C and C++ languages for scalable parallel programming of manycore GPUs and multicore CPUs. The CUDA programming model is described below, adapted from an article by Nickolls et al. [2008].

With the new model the GPU excels in data parallel and throughput computing, executing high performance computing applications as well as graphics applications.

### Data parallel problem decomposition

To map large computing problems effectively to a highly parallel processing architecture, the programmer or compiler decomposes the problem into many small problems that can be solved in parallel. For example, the programmer partitions a large result data array into blocks and further partitions each block into elements, such that the result blocks can be computed independently in parallel, and the elements within each block are computed in parallel. The figure below shows a decomposition of a result data array into a  $3 \times 2$  grid of blocks, where each block is further decomposed into a  $5 \times 3$  array of elements. The two-level parallel decomposition maps naturally to the GPU architecture: parallel multiprocessors compute result blocks, and parallel threads compute result elements.

Figure 9.3.3: Decomposing result data into a grid of blocks of elements to be computed in parallel (COD Figure C.3.3).

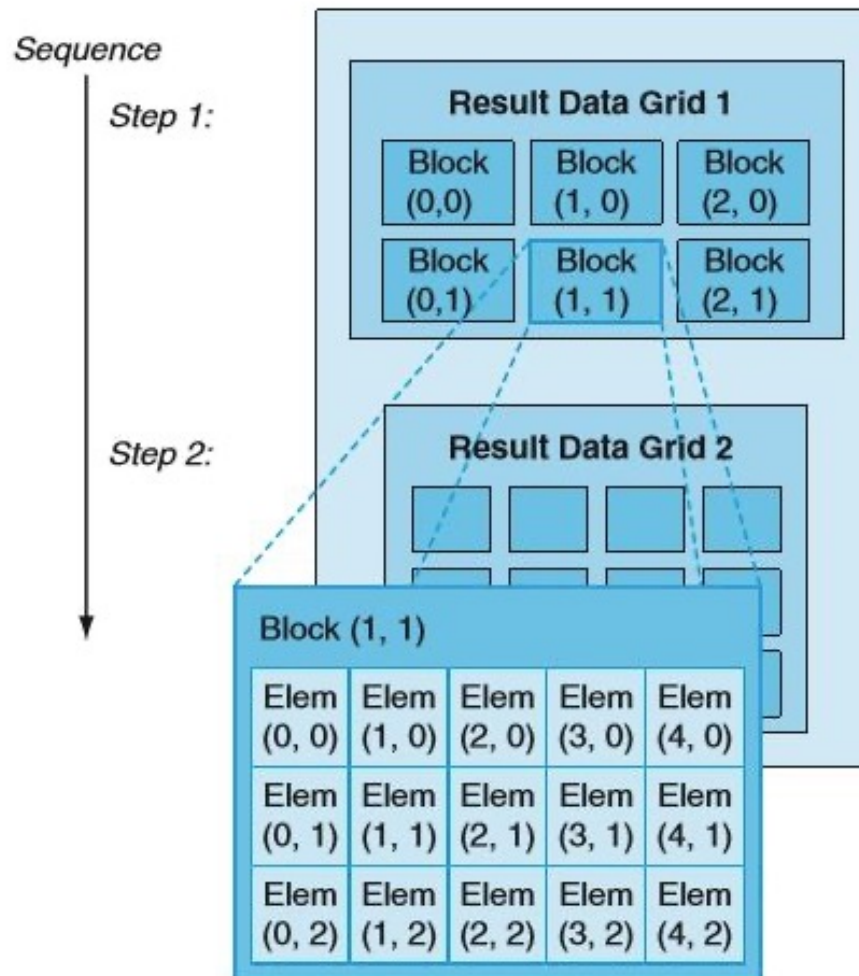


Figure 9.3.4: Sequential code (top) in C versus parallel code (bottom) in CUDA for SAXPY (see COD Chapter 6 (Parallel Processor from Client to Cloud)) (COD Figure C.3.4).

CUDA parallel threads replace the C serial loop—each thread computes the same result as one loop iteration. The parallel code computes  $n$  results with  $n$  threads organized in blocks of 256 threads.

***Computing  $y = ax + y$  with a serial loop:***

```
void saxpy_serial(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i<n; ++i)
        y[i] = alpha*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

***Computing  $y = ax + y$  in parallel using CUDA:***

```
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    if( i<n ) y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

**Atomic memory operation:** A memory read, modify, write operation sequence that completes without any intervening access.

Threads may access data from multiple memory spaces during their execution. Each thread has a private *local memory*. CUDA uses local memory for thread-private variables that do not fit in the thread's registers, as well as for stack frames and register spilling. Each thread block has a *shared memory*, visible to all threads of the block, which has the same lifetime as the block. Finally, all threads have access to the same *global memory*. Programs declare variables in shared and global memory with the `__shared__` and `__device__` type qualifiers. On a Tesla architecture GPU, these memory spaces correspond to physically separate memories: per-block shared memory is a low-latency on-chip RAM, while global memory resides in the fast DRAM on the graphics board.

**Local memory:** Per-thread local memory private to the thread.

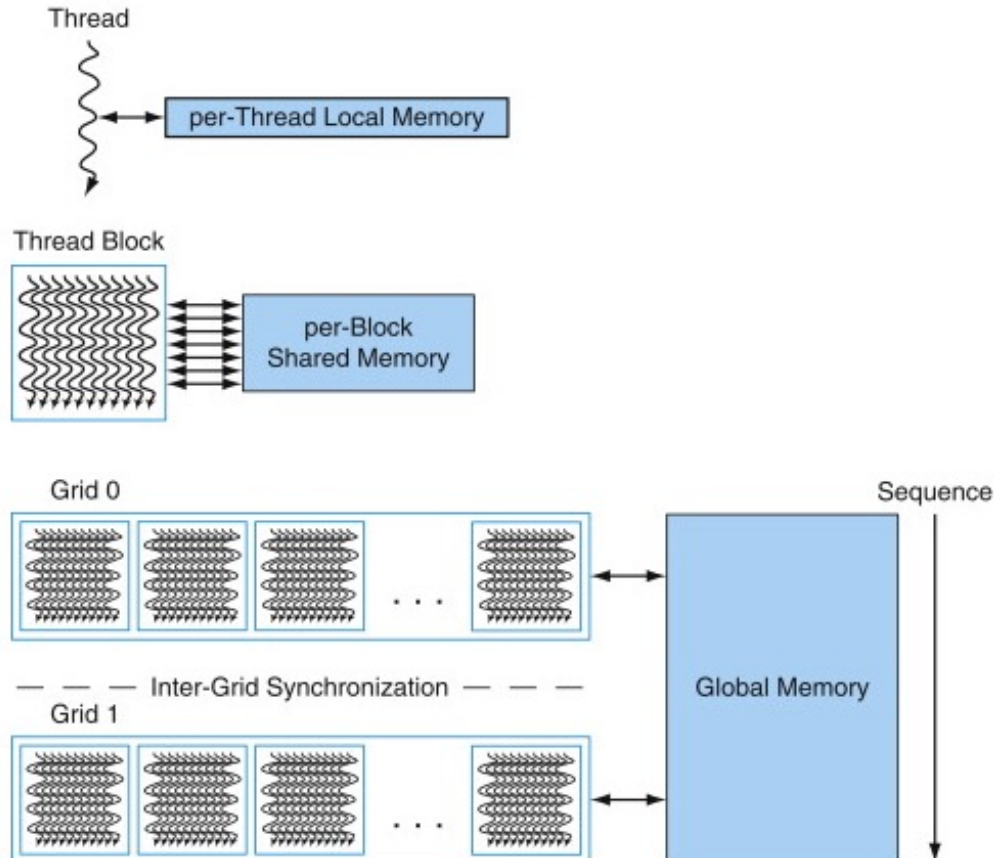
**Shared memory:** Per-block memory shared by all threads of the block.

**Global memory:** Per-application memory shared by all threads.

Shared memory is expected to be a low-latency memory near each processor, much like an L1 cache. It can therefore provide high-performance communication and data sharing among the threads of a thread block. Since it has the same lifetime as its corresponding thread block, kernel code will typically initialize data in shared variables, compute using shared variables, and copy shared memory results to global memory. Thread blocks of sequentially dependent grids communicate via global memory, using it to read input and write results.

Figure 9.3.5: Nested granularity levels—thread, thread block, and grid—have corresponding memory sharing levels—local, shared, and global (COD Figure C.3.5).

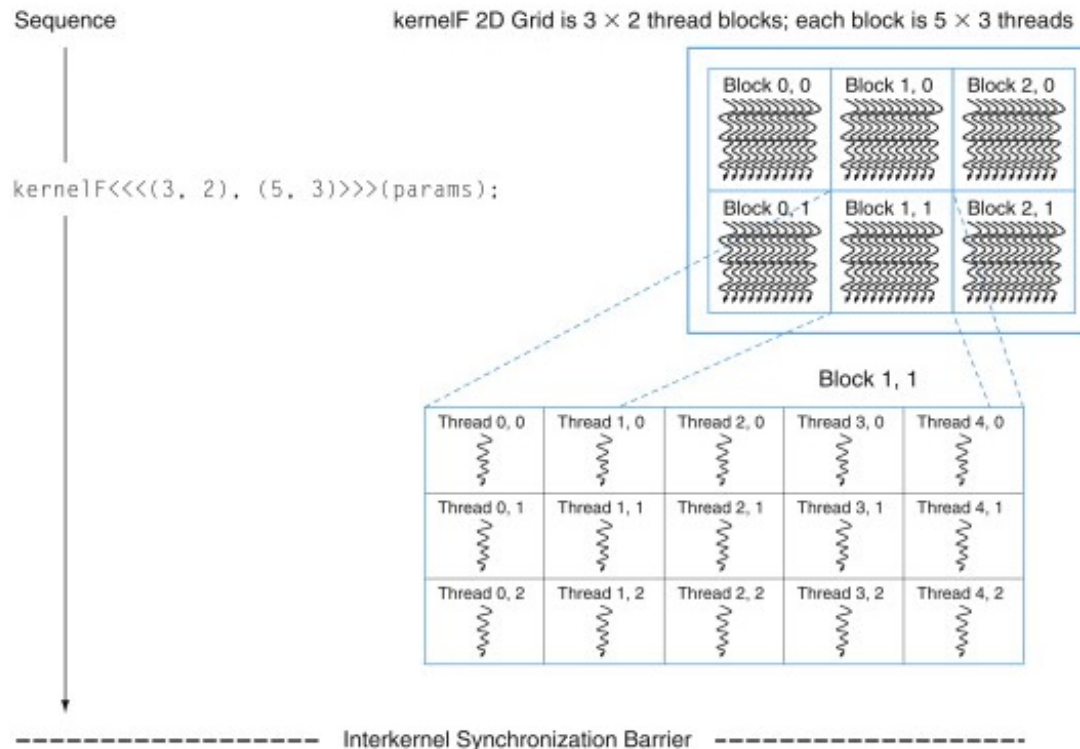
Per-thread local memory is private to the thread. Per-block shared memory is shared by all threads of the block. Per-application global memory is shared by all threads.



**Single-program multiple data (SPMD):** A style of parallel programming model in which all threads execute the same program. SPMD threads typically coordinate with barrier synchronization.

Figure 9.3.6: Sequence of kernel F (COD Figure C.3.6).

Sequence of kernel **F** instantiated on a 2D grid of 2D thread blocks, an interkernel synchronization barrier, followed by kernel **G** on a 1D grid of 1D thread blocks.



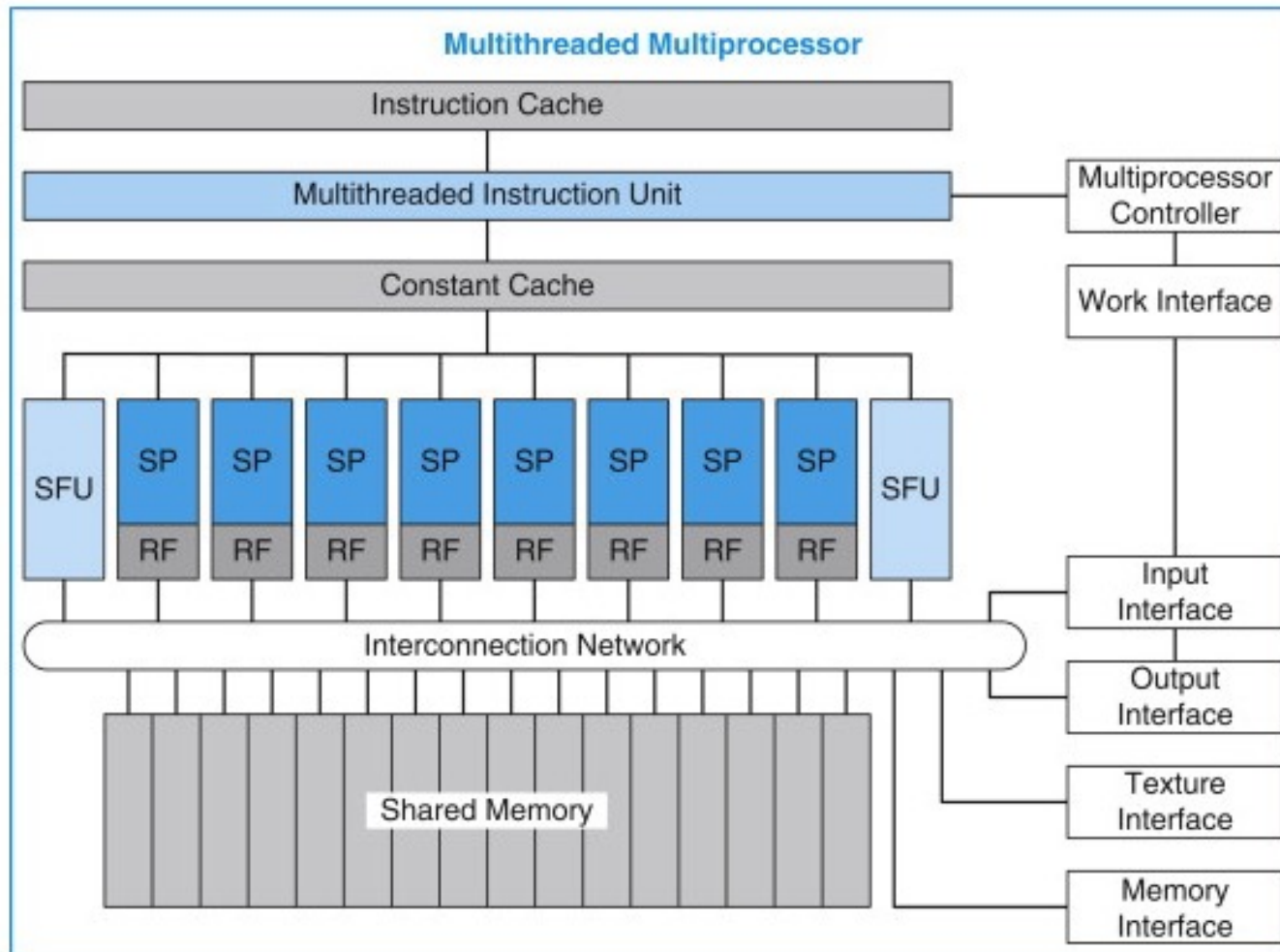
## Massive multithreading

GPU processors are highly multithreaded to achieve several goals:

- Cover the latency of memory loads and texture fetches from DRAM
- Support fine-grained parallel graphics shader programming models
- Support fine-grained parallel computing programming models
- Virtualize the physical processors as threads and thread blocks to provide transparent scalability
- Simplify the parallel programming model to writing a serial program for one thread

Figure 9.4.1: Multithreaded multiprocessor with eight scalar processor (SP) cores (COD Figure C.4.1).

The eight SP cores each have a large multithreaded register file (RF) and share an instruction cache, multithreaded instruction issue unit, constant cache, two special function units (SFUs), interconnection network, and a multibank shared memory.



## Single-instruction multiple-thread (SIMT)

To manage and execute hundreds of threads running several different programs efficiently, the multiprocessor employs a *single-instruction multiple-thread (SIMT)* architecture. It creates, manages, schedules, and executes concurrent threads in groups of parallel threads called *warps*. The term *warp* originates from weaving, the first parallel thread technology. The photograph in the figure below shows a warp of parallel threads emerging from a loom. This example multiprocessor uses a SIMT warp size of 32 threads, executing four threads in each of the eight SP cores over four clocks. The Tesla SM multiprocessor described in COD Section C.7 (Real stuff: The NVIDIA GeForce 8800) also uses a warp size of 32 parallel threads, executing four threads per SP core for efficiency on plentiful pixel threads and computing threads. Thread blocks consist of one or more warps.

**Single-instruction multiple-thread (SIMT):** A processor architecture that applies one instruction to multiple independent threads in parallel.

**Warp:** The set of parallel threads that execute the same instruction together in a SIMT architecture.

Figure 9.4.2: SIMT multithreaded warp scheduling (COD Figure C.4.2).

The scheduler selects a ready warp and issues an instruction synchronously to the parallel threads composing the warp. Because warps are independent, the scheduler may select a different warp each time.

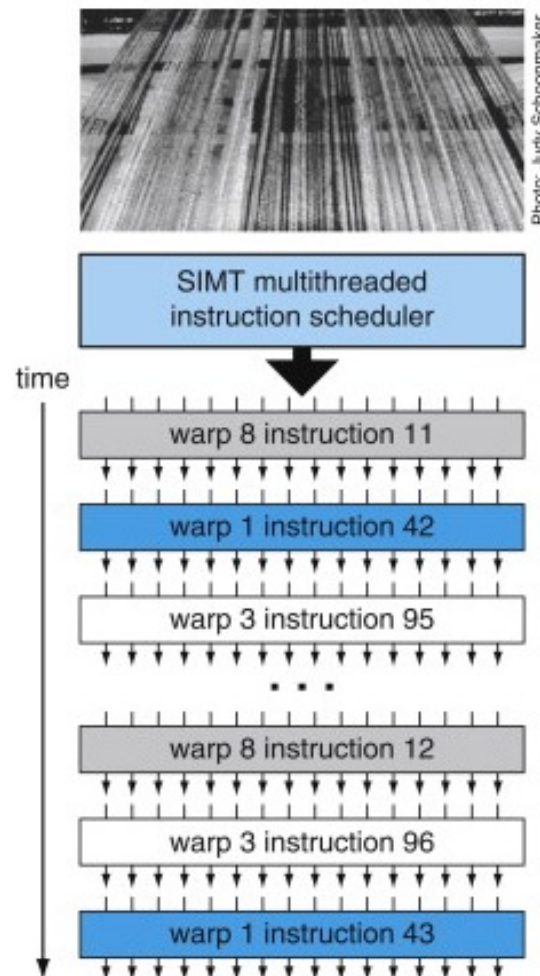


Figure 9.4.3: Basic PTX GPU thread instructions (COD Figure C.4.3).

Basic PTX GPU Thread Instructions

Group	Instruction	Example	Meaning	Comments
Arithmetic	arithmetic .type = .s32, .u32, .f32, .s64, .u64, .f64			
	add.type	add.f32 d, a, b	$d = a + b;$	
	sub.type	sub.f32 d, a, b	$d = a - b;$	
	mul.type	mul.f32 d, a, b	$d = a * b;$	
	mad.type	mad.f32 d, a, b, c	$d = a * b + c;$	multiply-add
	div.type	div.f32 d, a, b	$d = a / b;$	multiple microinstructions
	rem.type	rem.u32 d, a, b	$d = a \% b;$	integer remainder
	abs.type	abs.f32 d, a	$d =  a ;$	
	neg.type	neg.f32 d, a	$d = 0 - a;$	
	min.type	min.f32 d, a, b	$d = (a < b) ? a : b;$	floating selects non-NaN
	max.type	max.f32 d, a, b	$d = (a > b) ? a : b;$	floating selects non-NaN
	setp.cmp.type	setp.lt.f32 p, a, b	$p = (a < b);$	compare and set predicate
	numeric .cmp = eq, ne, lt, le, gt, ge; unordered cmp = equ, neu, ltu, leu, gtu, geu, num, nan			
	mov.type	mov.b32 d, a	$d = a;$	move
	selp.type	selp.f32 d, a, b, p	$d = p ? a : b;$	select with predicate
	cvt.dtype.atype	cvt.f32.s32 d, a	$d = \text{convert}(a);$	convert atype to dtype
Special Function	special .type = .f32 (some .f64)			
	rcp.type	rcp.f32 d, a	$d = 1/a;$	reciprocal
	sqr.type	sqr.f32 d, a	$d = \text{sqrt}(a);$	square root
	rsqr.type	rsqr.f32 d, a	$d = 1/\text{sqrt}(a);$	reciprocal square root
	sin.type	sin.f32 d, a	$d = \sin(a);$	sine
	cos.type	cos.f32 d, a	$d = \cos(a);$	cosine
	lg2.type	lg2.f32 d, a	$d = \log(a)/\log(2)$	binary logarithm
	ex2.type	ex2.f32 d, a	$d = 2 ** a;$	binary exponential

Logical	logic.type = .pred, .b32, .b64			
	and.type	and.b32 d, a, b	d = a & b;	
	or.type	or.b32 d, a, b	d = a   b;	
	xor.type	xor.b32 d, a, b	d = a ^ b;	
	not.type	not.b32 d, a, b	d = ~a;	one's complement
	cnot.type	cnot.b32 d, a, b	d = (a==0)? 1:0;	C logical not
	shl.type	shl.b32 d, a, b	d = a << b;	shift left
	shr.type	shr.s32 d, a, b	d = a >> b;	shift right
Memory Access	memory.space = .global, .shared, .local, .const; .type = .b8, .u8, .s8, .b16, .b32, .b64			
	ld.space.type	ld.global.b32 d, [a+off]	d = *(a+off);	load from memory space
	st.space.type	st.shared.b32 [d+off], a	*(d+off) = a;	store to memory space
	tex.nd.dtype.btype	tex.2d.v4.f32.f32 d, a, b	d = tex2d(a, b);	texture lookup
	atom.spc.op.type	atom.global.add.u32 d,[a], b	atomic { d = *a;	atomic read-modify-write operation
		atom.global.cas.b32 d,[a], b, c	*a = op(*a, b); }	
atom.op = and, or, xor, add, min, max, exch, cas; .spc = .global; .type = .b32				
Control Flow	branch	@p bra target	if (p) goto target;	conditional branch
	call	call (ret), func, (params)	ret = func(params);	call function
	ret	ret	return;	return from function call
	bar.sync	bar.sync d	wait for threads	barrier synchronization
	exit	exit	exit;	terminate thread execution

Figure 9.6.1: Special function approximation statistics (COD Figure C.6.1).

For the NVIDIA GeForce 8800 special function unit (SFU).

Function	Input interval	Accuracy (good bits)	ULP* error	% exactly rounded	Monotonic
$1/x$	$[1, 2)$	24.02	0.98	87	Yes
$1/\sqrt{x}$	$[1, 4)$	23.40	1.52	78	Yes
$2^x$	$[0, 1)$	22.51	1.41	74	Yes
$\log_2 x$	$[1, 2)$	22.57	N/A**	N/A	Yes
sin/cos	$[0, \pi/2)$	22.47	N/A	N/A	No

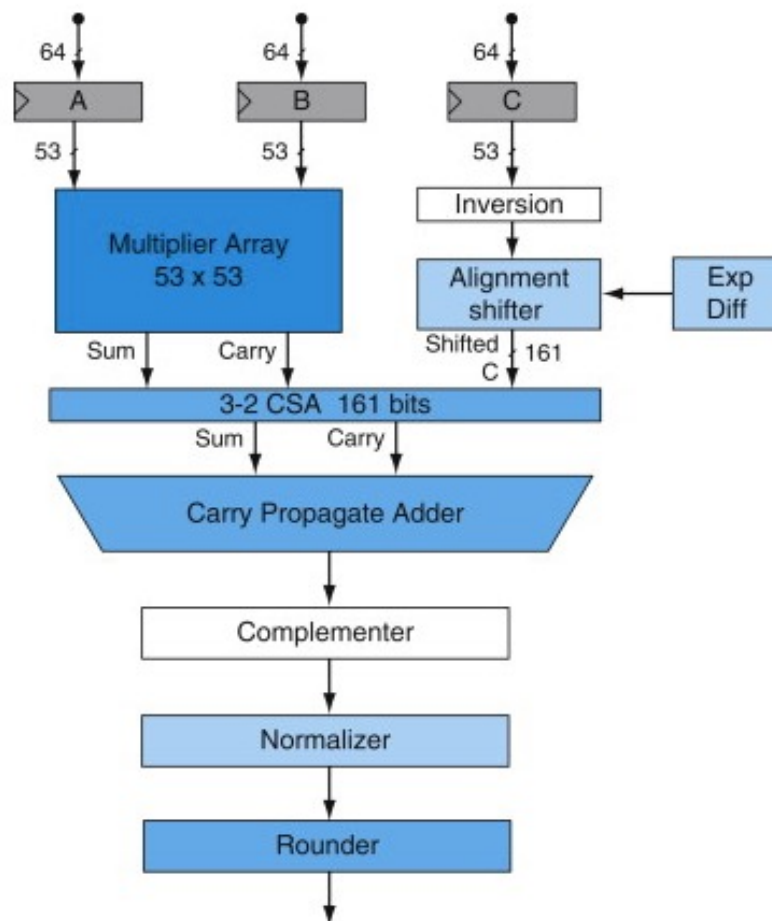
\*ULP: unit in the last place.\*\*N/A: not applicable.

# GPU

## P&H Ch 9

Figure 9.6.2: Double precision fused-multiply-add (FMA) unit (COD Figure C.6.2).

Hardware to implement floating-point  $A \times B + C$  for double precision.



# GPU Memory

## P&H Ch 9

### MMU

Modern GPUs are capable of translating virtual addresses to physical addresses. On the GeForce 8800, all processing units generate memory addresses in a 40-bit virtual address space. For computing, load and store thread instructions use 32-bit byte addresses, which are extended to a 40-bit virtual address by adding a 40-bit offset. A memory management unit performs virtual to physical address translation; hardware reads the page tables from local memory to respond to misses on behalf of a hierarchy of translation lookaside buffers spread out among the processors and rendering engines. In addition to physical page bits, GPU page table entries specify the compression algorithm for each page. Page sizes range from 4 to 128 kilobytes.

### ROP

As shown in COD Figure C.2.5 (Basic unified GPU architecture), NVIDIA Tesla architecture GPUs comprise a scalable streaming processor array (SPA), which performs all of the GPU's programmable calculations, and a scalable memory system, which comprises external DRAM control and fixed function *Raster Operation Processors* (ROPs) that perform color and depth framebuffer operations directly on memory. Each ROP unit is paired with a specific memory partition. ROP partitions are fed from the SMs via an interconnection network. Each ROP is responsible for depth and stencil tests and updates, as well as color blending. The ROP and memory controllers cooperate to implement lossless color and depth compression (up to 8:1) to reduce external bandwidth demand. ROP units also perform atomic operations on memory.

# Section

---

## GPU Products

# Apple

# Apple M1

 **M1**

**8-core  
GPU**

**Up to 8 cores**

**128 execution units**

**Up to 24,576 concurrent threads**

**2.6 teraflops**

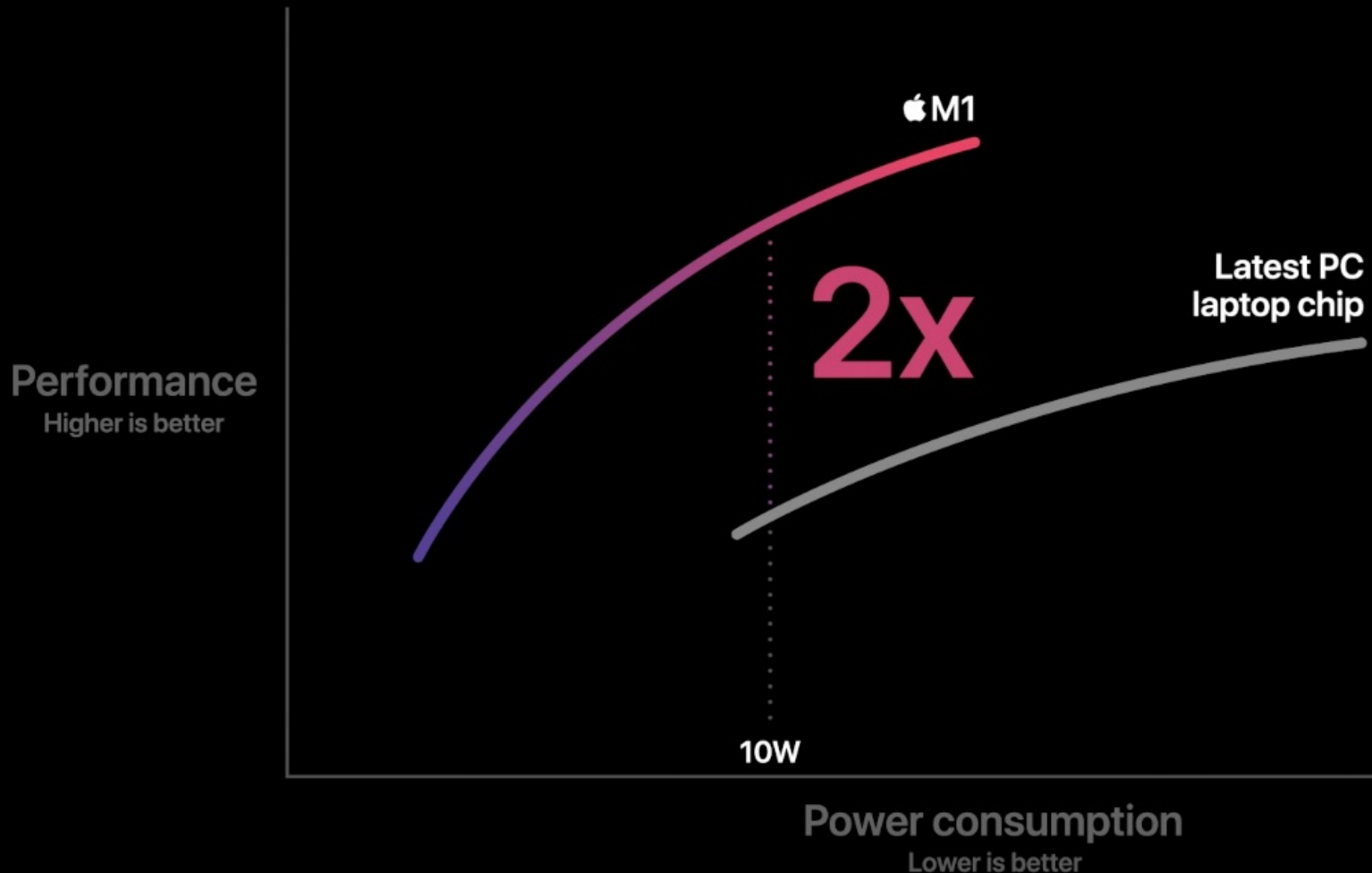
**82 gigatexels/second**

**41 gigapixels/second**

# Apple M1

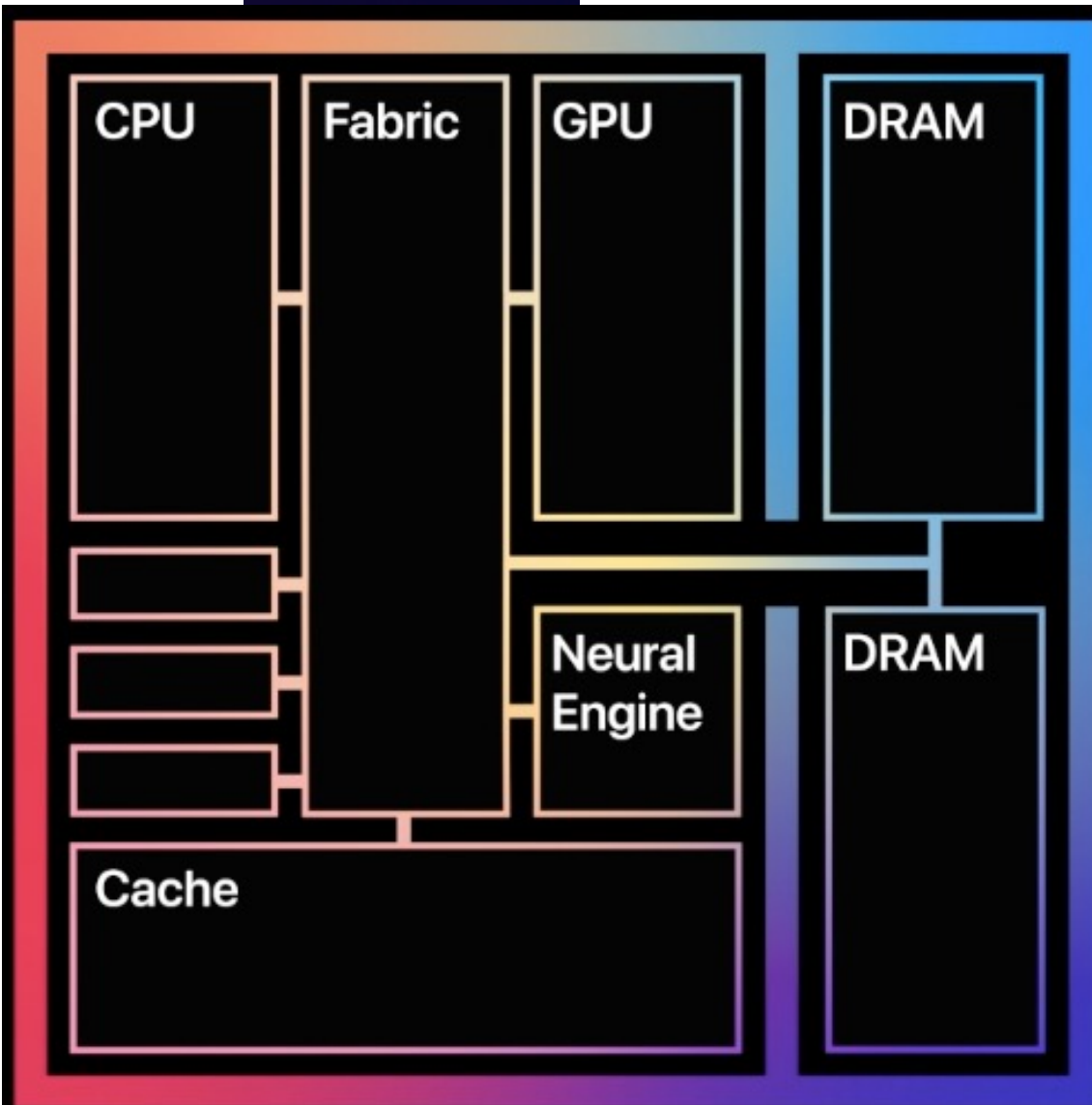
November 10, 2020

## GPU Performance vs. Power



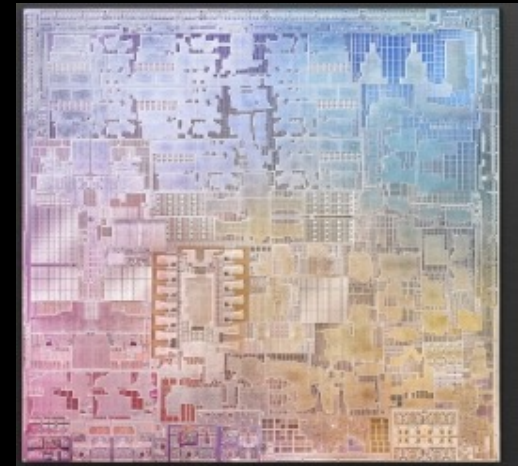
November 10, 2020

# Apple M1

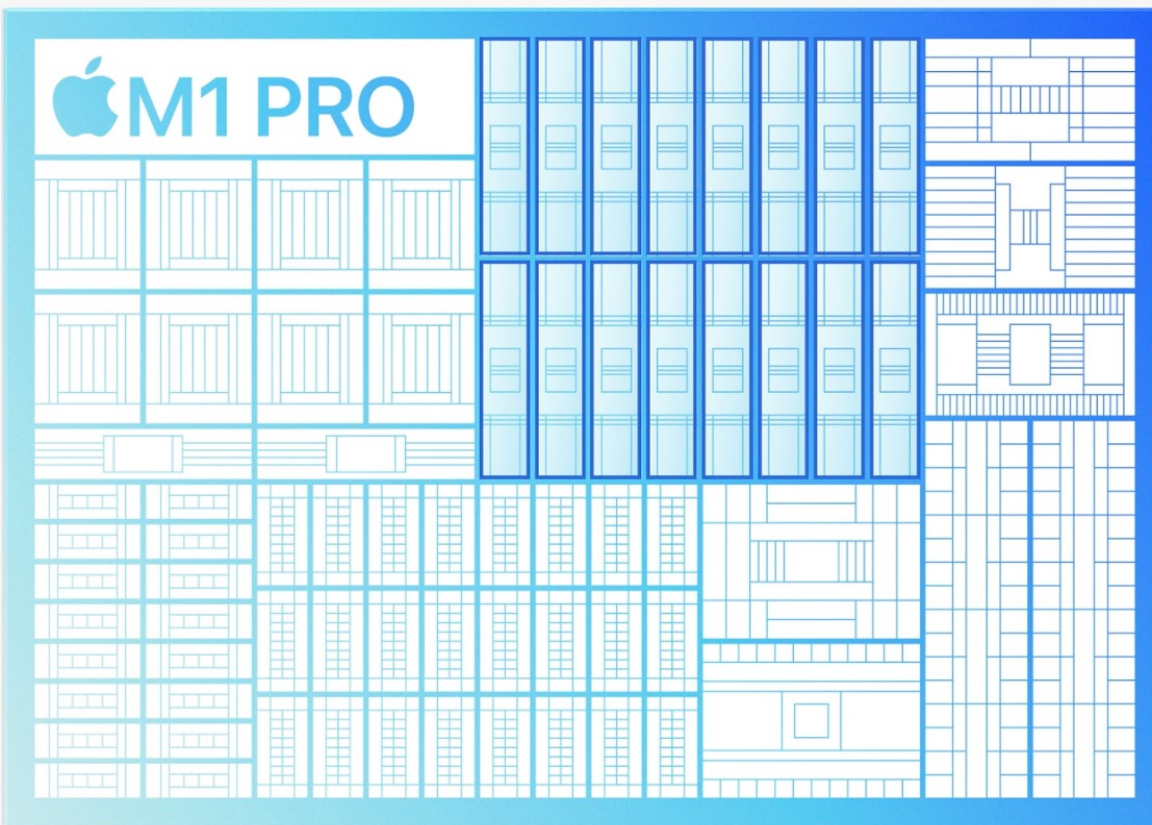


## Unified memory architecture

High bandwidth, low latency  
Apple-designed package  
Accessible to entire SoC



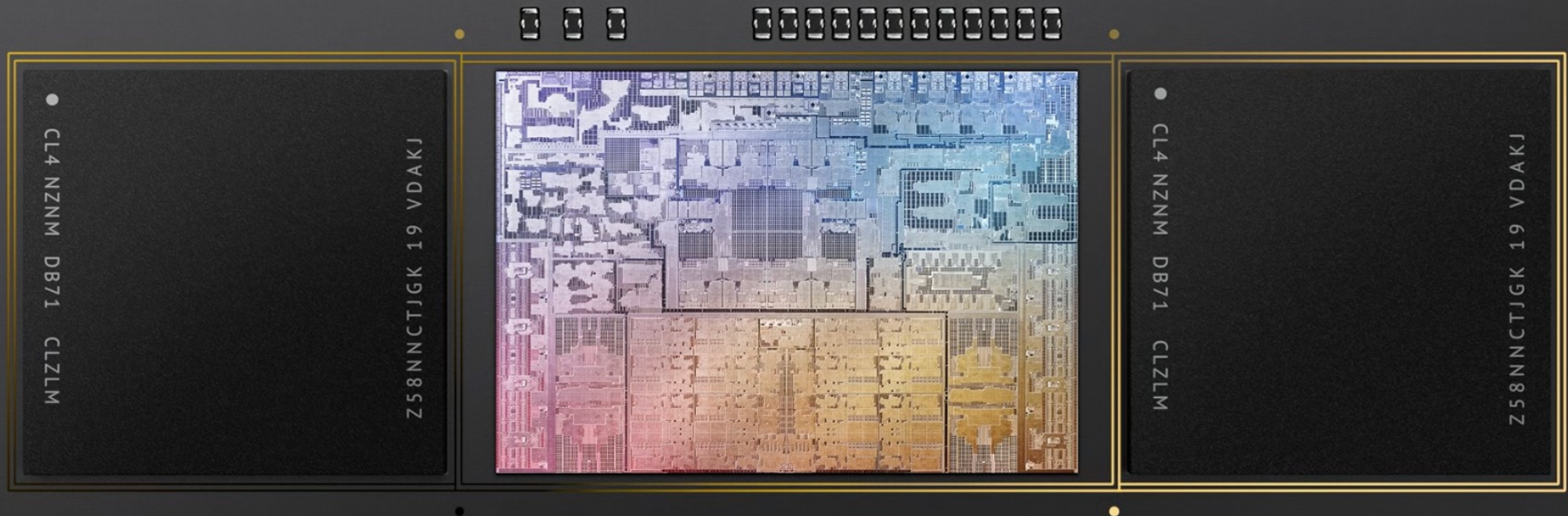
# Apple M1 Pro



## 16-core GPU

2048 execution units  
Up to 49,512 concurrent threads  
5.2 teraflops  
164 gigatexels/second  
82 gigapixels/second

# Apple M1 Pro



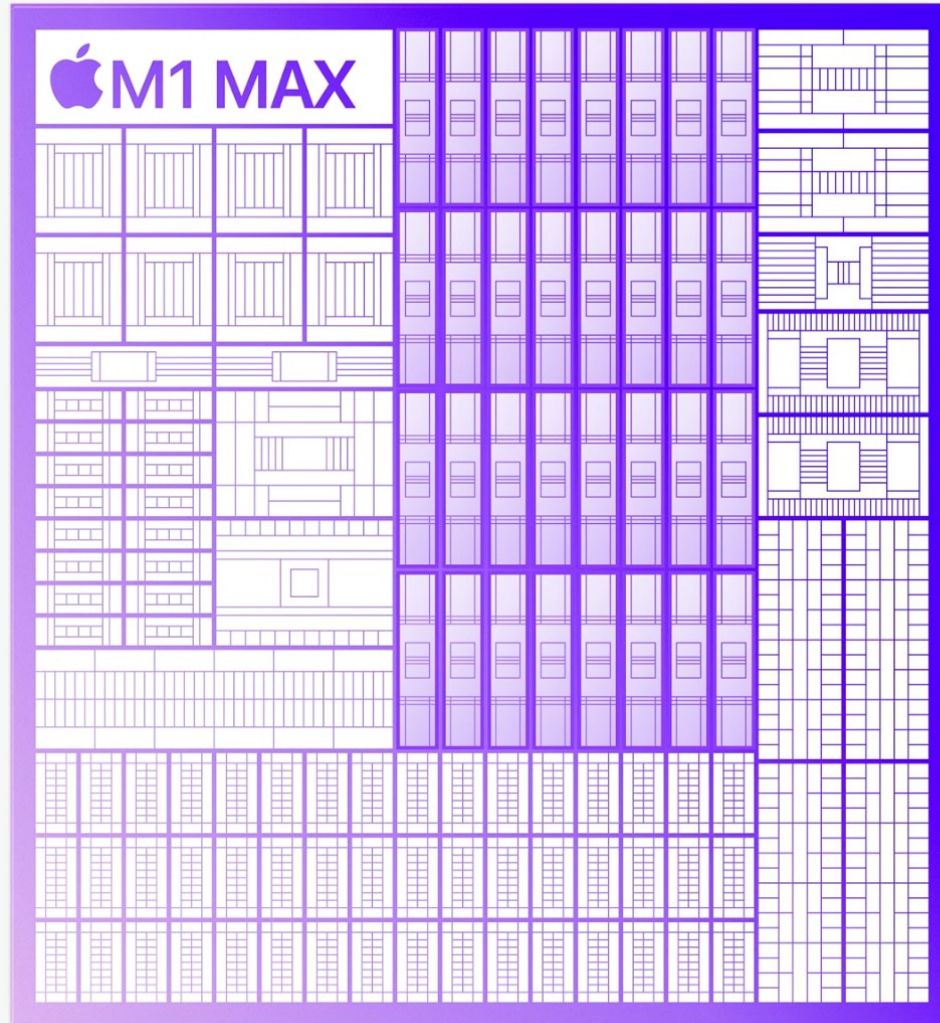
## 32GB unified memory

High bandwidth, low latency

256-bit LPDDR5 interface

Apple-designed custom package

# Apple M1 Max



## 32-core GPU

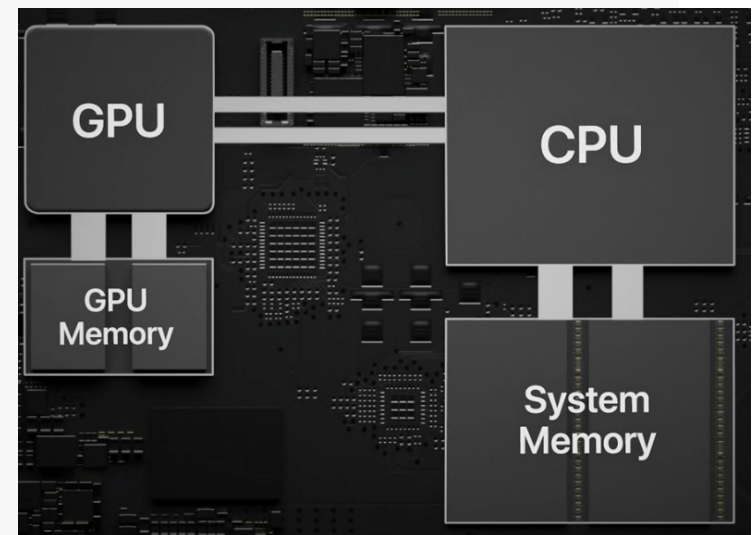
4096 execution units

Up to 98,304 concurrent threads

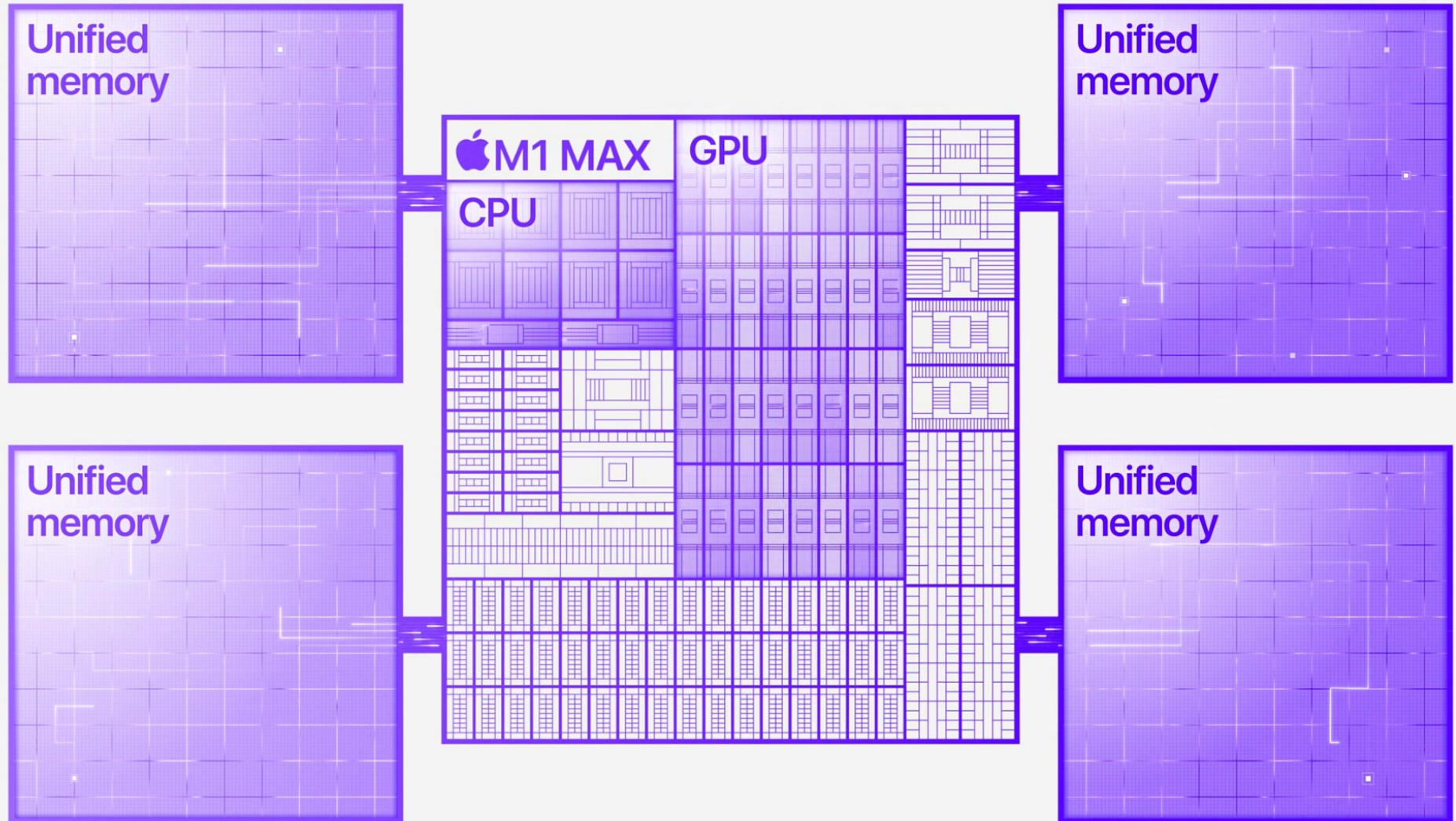
10.4 teraflops

327 gigatexels/second

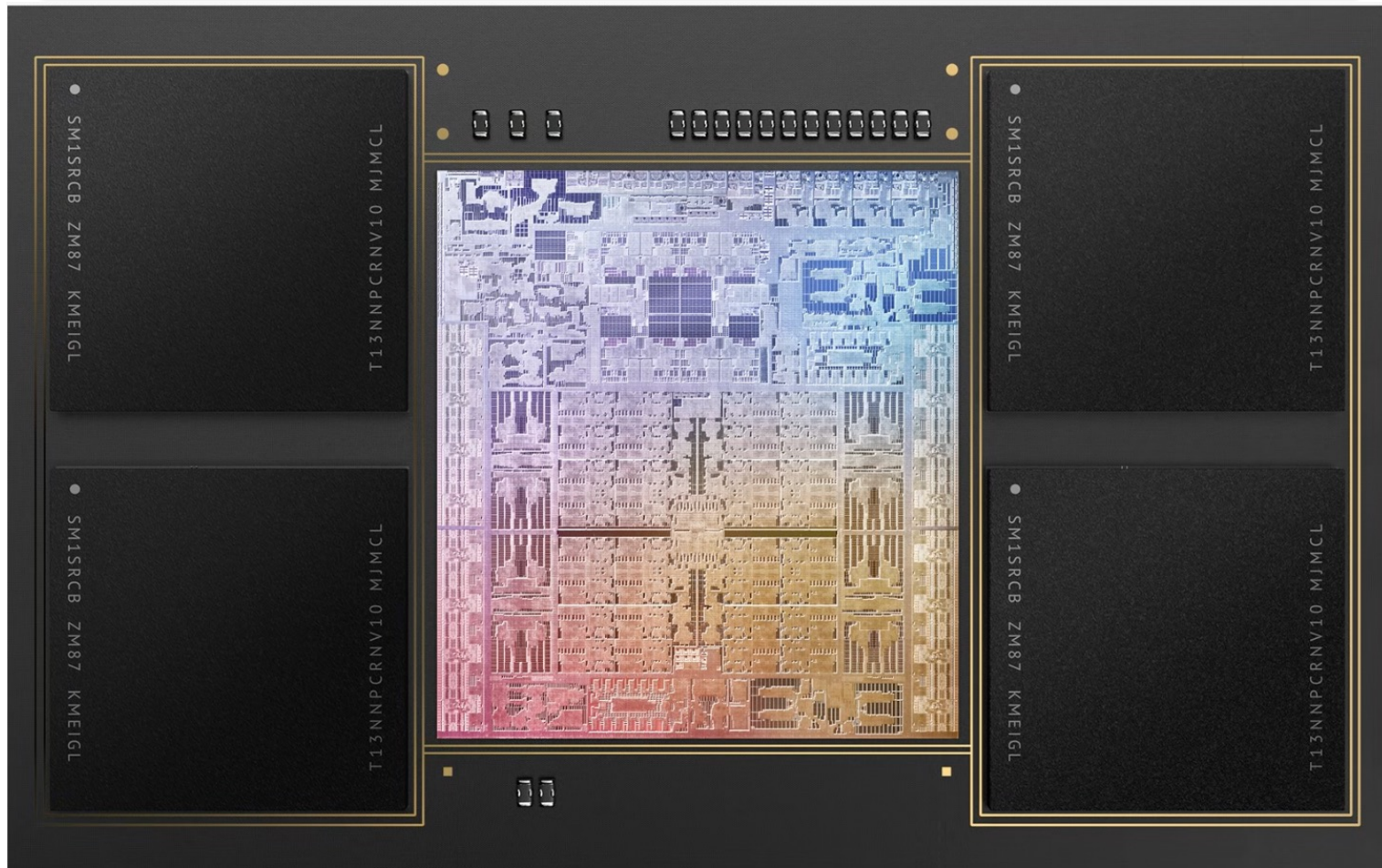
164 gigapixels/second



# Apple M1 Max



# Apple M1 Max



## 64GB unified memory

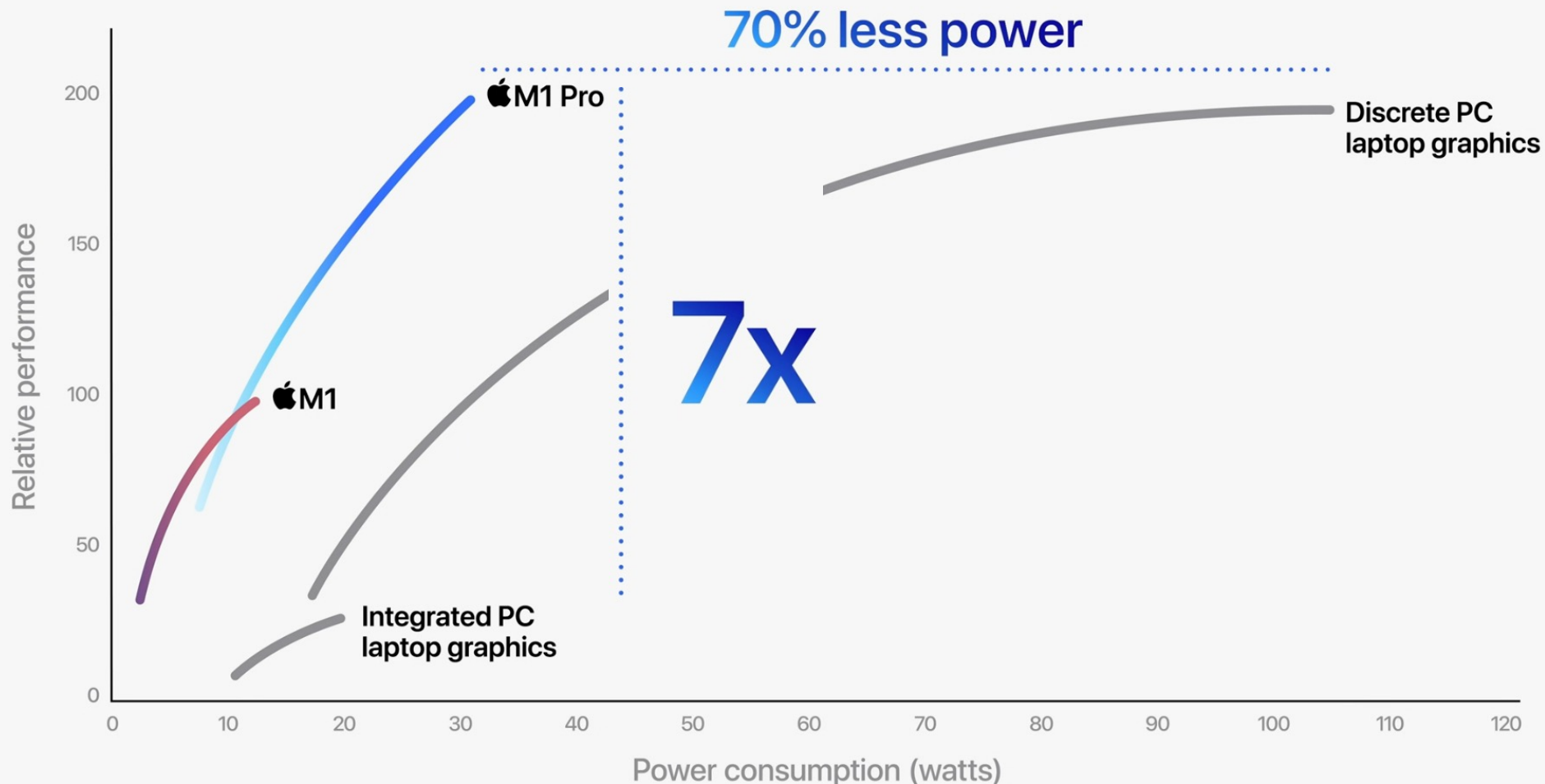
High bandwidth, low latency

512-bit LPDDR5 interface

Apple-designed custom package

# Apple M1 Max

## GPU performance vs. power



# GPU Wars: Apple v Nvidia



**Mahaveer Verma**, System Design Engineer at NVIDIA (2017-present)

Answered August 8, 2017



Because Apple has some of its own algorithms for optimizing graphics output, to run which they needed to bypass Nvidia drivers and interact directly with the hardware and thus, were asking for a more direct exposure to Nvidia's hardware (compared to all other software that runs on top of Nvidia drivers and never bypassing it) which is against Nvidia's policy to not 'bend' its implementation for a customer. I'm no legal expert and I have no clue how much of an exposure is generally considered acceptable by Nvidia, but Apple's optimization algorithms required a level more than that and so Nvidia and Apple had to part ways after Maxwell I suppose.

Apple then 'moved' to using AMD.

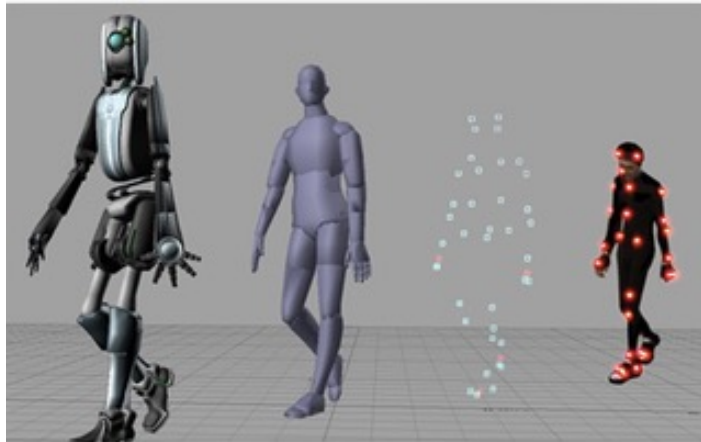
So, from what I've heard being in the company, it was a conflict of interest. Nothing else.

# Section

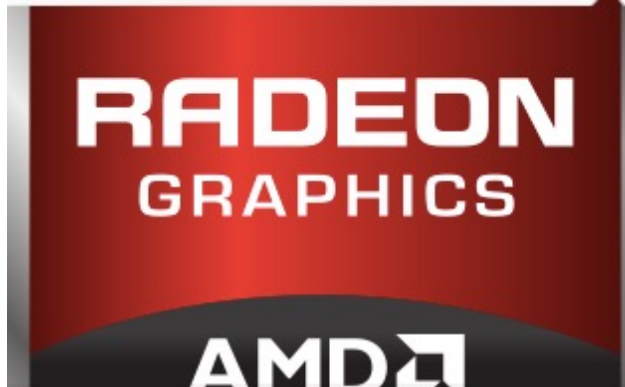
## GPU Products

# AMD

# Graphics



**3D rendering** is the 3D computer graphics process of converting 3D models into 2D images on a computer. 3D renders may include photorealistic effects or non-photorealistic styles.



The **Radeon HD 7000 series**, codenamed "Southern Islands", is a family of GPUs developed by AMD, and manufactured on TSMC's 28 nm process. The primary competitor of Southern Islands, Nvidia's GeForce 600 Series, also shipped during Q1 2012, largely due to the immaturity of the

AMD vs Nvidia



VGA section on the motherboard in IBM PS/55



Tseng Labs ET4000/W32p



S3 Graphics VIRGE

# Graphics Cards

See gpu vs graphics card



XFX - AMD Radeon  
RX 580 GTS Black  
Edition 8GB GDDR5  
\$189.99  
Best Buy  
4.6K+ viewed



MSI GeForce GTX  
1050 Ti Directx 12  
GTX 1050 Ti 4GT LP  
\$157.38  
Newegg.com  
Free shipping



MSI NVIDIA GeForce  
RTX 2060 SUPER  
VENTUS GP OC  
\$429.99  
Newegg.com  
Free shipping



NVIDIA Tesla M10  
Graphic Card - 4  
Gpus - 32 GB  
\$2,450.00  
Newegg.com



MSI NVIDIA GeForce  
RTX 2060 SUPER  
VENTUS GP OC  
\$449.99  
Newegg.com  
Free shipping



MSI GeForce GTX  
1050 Ti Directx 12  
GTX 1050 Ti 4GT LP  
\$239.99  
Newegg.com  
Free shipping



NVIDIA GeForce GT  
320 1GB DDR3 PCI  
Express Pcie  
\$81.97  
PCLiquidations.com  
Refurbished



NVIDIA GeForce GT  
320 1GB DDR3 PCI  
Express Pcie  
\$81.97  
PCLiquidations.com  
Refurbished



MSI GeForce GTX  
1650 GAMING X 4G  
Graphics Card, PC  
\$256.56  
Newegg.com  
Free shipping



MSI Radeon R9 390  
Directx 12 R9 390  
GAMING 8G 8GB  
\$465.00  
Newegg.com  
★★★★★ 36



MSI GeForce GTX  
1660 Directx 12  
GTX 1660 VENTUS  
\$219.95  
Newegg.com  
Free shipping



MSI GeForce GTX  
1050 Ti GAMING X  
4G Graphics Card  
\$169.45  
B&H Photo-Vid...  
650+ viewed



MSI GeForce GTX  
1050 Ti Directx 12  
GTX 1050 Ti 4GT LP  
\$188.00  
Newegg.com  
Free shipping



MSI GeForce GTX  
1650 Directx 12  
GTX 1650 4GT LP  
\$159.99  
Newegg.com  
★★★★★ 1



NVIDIA's **TurboCache** technology is a method of allowing video cards more available framebuffer memory by using both onboard video memory and main system memory. Main memory is accessed using the high-bandwidth PCI-Express bus.



**HyperMemory** was a brand for ATI's method of using the motherboard's main system RAM as part of or all of the video card's framebuffer memory on their line of Radeon video cards and motherboard chipsets. It relies on new fast data transfer mechanisms within PCI Express.

The **AMD Accelerated Processing Unit (APU)**, formerly known as **Fusion**, is the marketing term for a series of 64-bit microprocessors from Advanced Micro Devices (AMD), designed to act as a central processing unit (CPU) and graphics processing unit (GPU) on a single die. A

## AMD *External* (PCI)



AMD

PRO

CPU

## Business Processors



### AMD Ryzen™ PRO Processors and Ryzen™ PRO Processors with Radeon™ Vega Graphics

For power users and mainstream users in the workplace

- From 4 to 12 cores
- Up to 24 processing threads
- Some models include Radeon™ Vega graphics

### AMD Athlon™ PRO Processors with Radeon™ Vega Graphics

For entry-level users in the workplace



# AMD

GPU

## AMD Embedded Graphics Processors



### AMD Welcomes The Next Generation Power-Efficient Embedded GPUs

The AMD Embedded Radeon™ E9170 Series

## Products



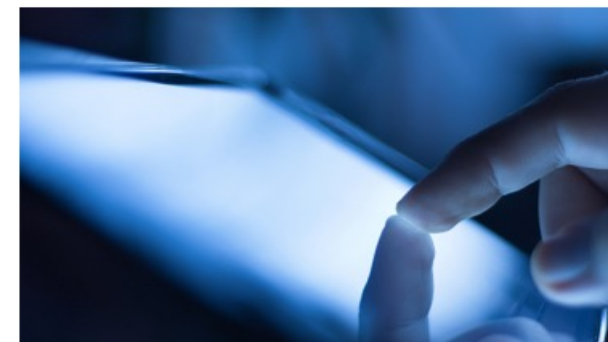
### Ultra-High Performance Embedded GPUs

These are AMD's highest performing embedded discrete GPUs. These incredibly powerful processors are well-suited for [high-end casino and arcade gaming machines](#), [high-end medical imaging devices](#), and [high-end aerospace applications](#).



### High-Performance Embedded GPUs

These GPUs provide the right balance of performance, power and price to meet the needs of most customers. They are well-suited for [casino and arcade gaming machines](#), [medical imaging devices](#), and [digital signage installations](#).



### Power-Efficient Embedded GPUs

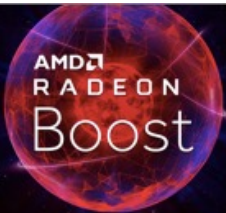
These GPUs provide excellent processing performance at reduced levels of power consumption, making them well-suited for [mobile signage](#), [retail and kiosks](#), [factory human-machine interface systems](#), [heads-up aerospace displays](#), and [thin client computers](#).



# AMD



GPU



## AMD Radeon™ Boost

Turbocharge your game

[LEARN MORE](#)

## AMD Radeon™ Anti-Lag

Delivering even faster click-to-response times

# Technologies for Gaming

## AMD Enhanced Sync Technology

Now available for games based on DirectX® 9, 11, 12 and Vulkan® APIs as well as on all GCN-based GPUs, GCN-based GPU combinations, and/or AMD Eyefinity Technology display combinations.

[LEARN MORE](#)



AMD Virtual Super Resolution

AMD TressFX Hair

The Vulkan® API

64 CORES    48 CORES    32 CORES    24 CORES    16 CORES    12 CORES

**8 CORES**

up to  
**128** threads

up to  
**2.6GHz** base

up to  
**3.4GHz** boost

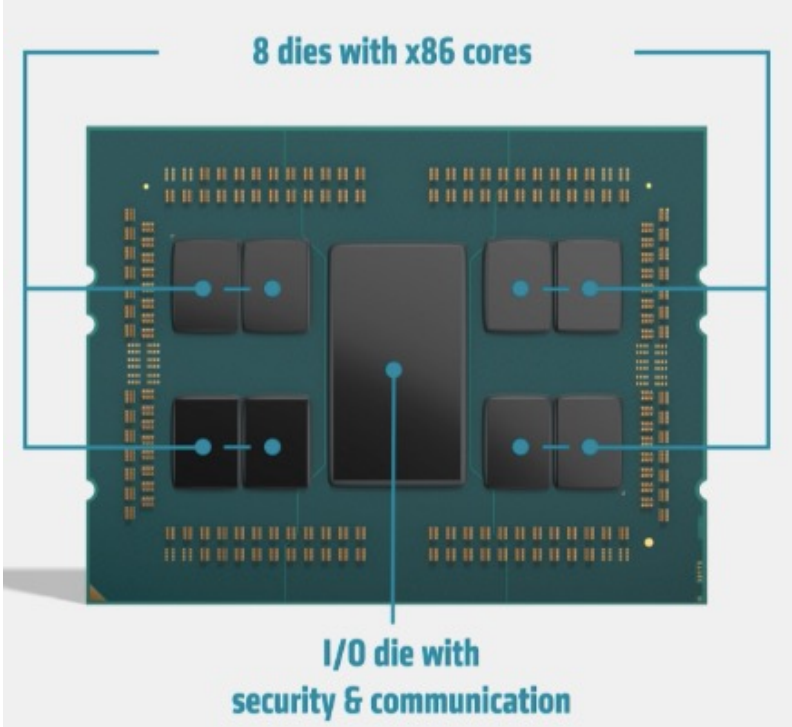
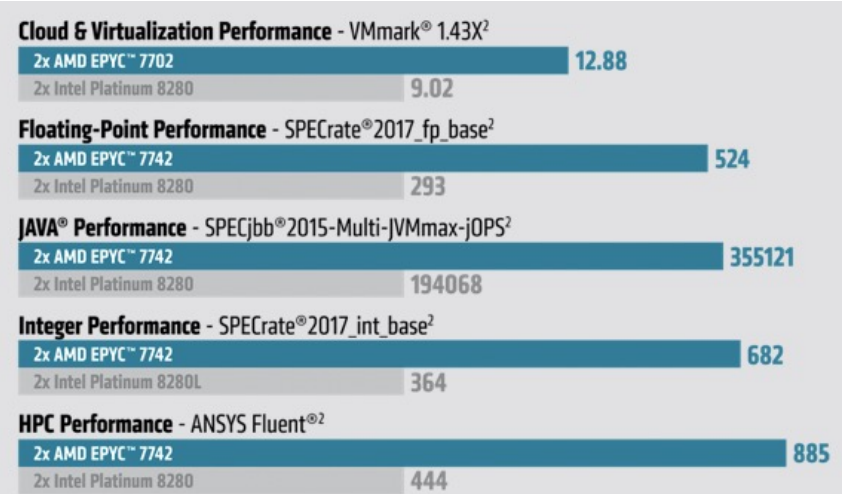
**256MB** cache

**X86 compatibility**



**2P Intel vs 1P EPYC comparison<sup>7</sup>**

Model	2x6262V	1x7702P
Cores	48	64
Memory Capacity	2TB	4TB
Max Memory Frequency	2400MHz	3200MHz <sup>6</sup>
I/O Lanes	96 PCIe <sup>®</sup> 3.0	128 PCIe <sup>®</sup> 4.0 <sup>6</sup>
TDP	270Watts	200Watts
SPECrate <sup>®</sup> 2017_int_base	242	319
'Per Socket software' licensing cost	x2	x1
List Price	5800USD	4425USD



# AMD GPU's: Radeon Pro 5K

Key capabilities and features of AMD Radeon Pro 5000 series GPUs include:

- **Exceptional compute performance** – Up to 7.6 TFLOPS of single-precision (FP32) floating point performance.
- **GDDR6 memory** – Up to 16GB of GDDR6 memory with 384 GB/s bandwidth provides ultra-fast transfer speeds to power data-intensive pro applications.
- **AMD RDNA architecture** – AMD RDNA architecture was designed from the ground up for superior performance and power efficiency. It is built on industry-leading 7nm FinFET process technology, providing up to 1.5X higher performance per watt compared to the previous-generation graphics architecture<sup>1</sup>.

AMD Radeon™ Pro 5000 series GPUs	Compute Units	Stream Processors	FP32 TFLOPS	GDDR6 Memory
AMD Radeon™ Pro 5700 XT	40	2560	Up to 7.6	16GB
AMD Radeon™ Pro 5700	36	2304	Up to 6.2	8GB
AMD Radeon™ Pro 5500 XT	24	1536	Up to 5.3	8GB
AMD Radeon™ Pro 5300	20	1280	Up to 4.2	4GB

7.6 Tflops

# AMD RDNA3 GPU

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>



# AMD RDNA3 GPU

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>

**WORLD'S FIRST CHIPLET GAMING GPU**

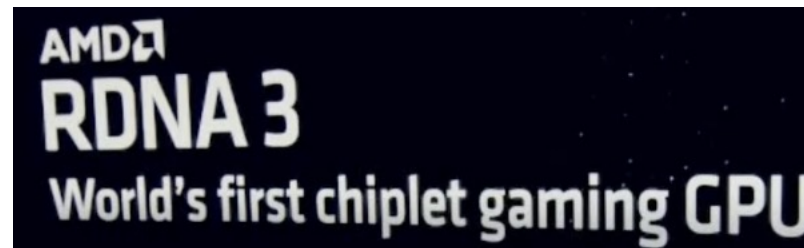
**INTERCONNECT BANDWIDTH  
UP TO 5.3 TB/s**

(k)

# AMD RDNA3 GPU

11-3-22

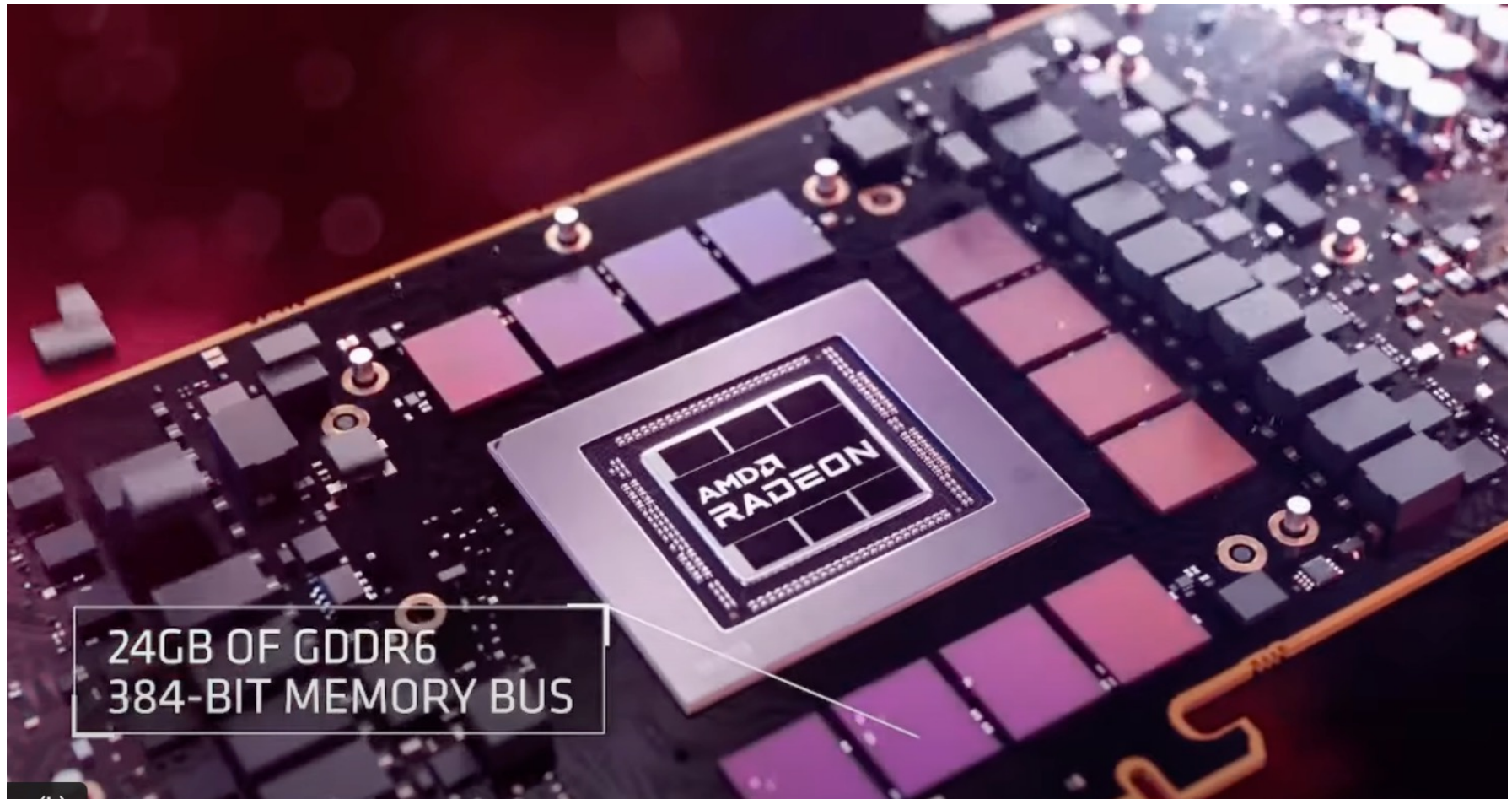
<https://www.youtube.com/watch?v=hhwd6UgGVk4>



# AMD RDNA3 GPU

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>

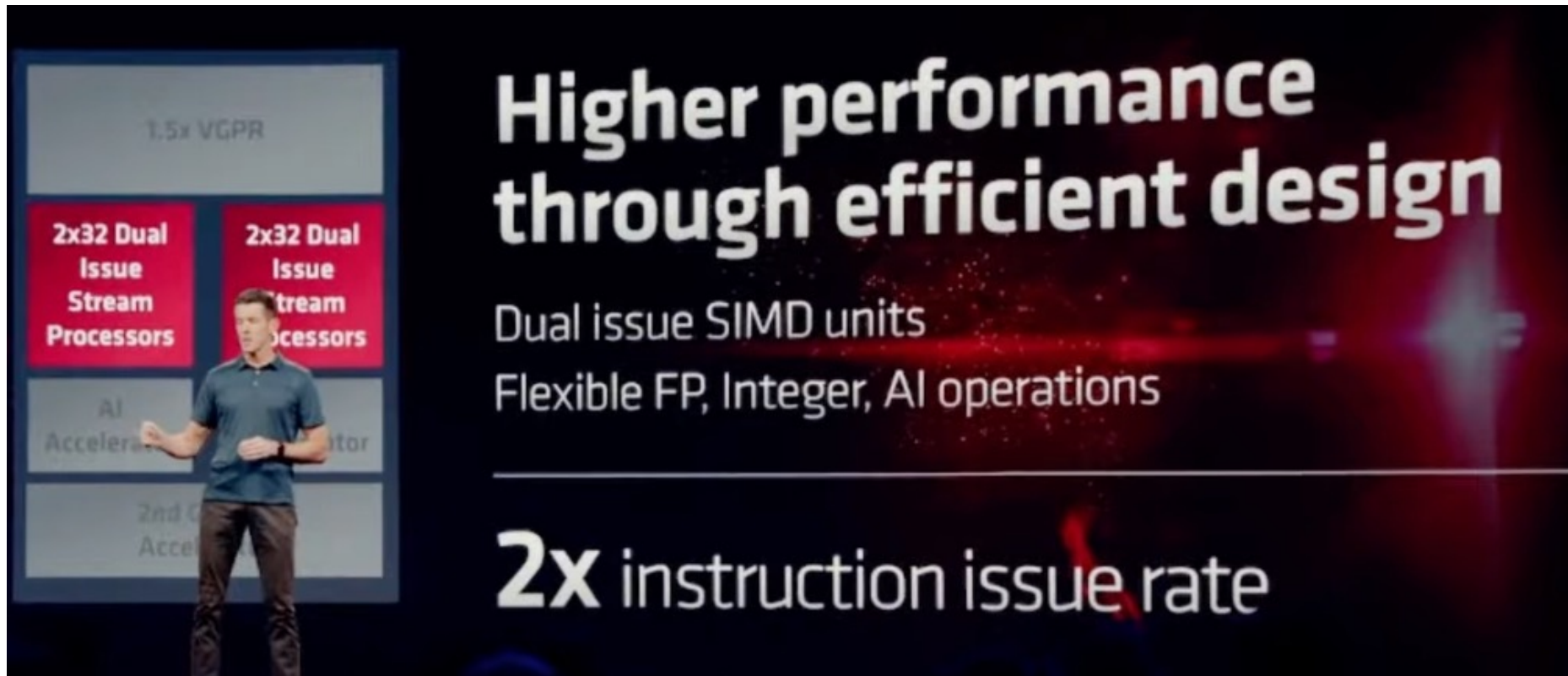


# AMD RDNA3 GPU

11-3-22



<https://www.youtube.com/watch?v=hhwd6UgGVk4>



A presentation slide for the AMD RDNA3 GPU. On the left, a man in a blue polo shirt and dark pants stands next to a large screen. The screen displays a diagram of the GPU architecture with the following text: '1.5x VGPR' at the top, '2x32 Dual Issue Stream Processors' in two red boxes, 'AI Accelerator' in a grey box, and '2nd Gen Accelerator' at the bottom. The main text on the right reads 'Higher performance through efficient design' in large white letters, followed by 'Dual issue SIMD units' and 'Flexible FP, Integer, AI operations' in smaller white letters. At the bottom, it says '2x instruction issue rate' in large white letters.

1.5x VGPR

2x32 Dual Issue Stream Processors

2x32 Dual Issue Stream Processors

AI Accelerator

2nd Gen Accelerator

## Higher performance through efficient design

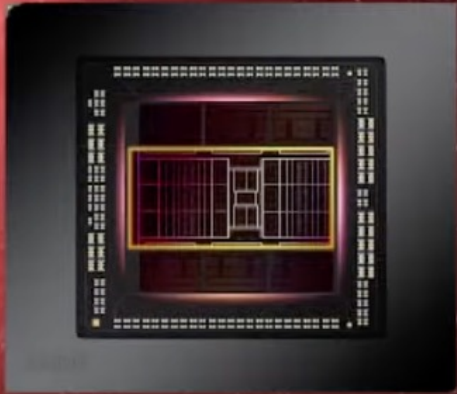
Dual issue SIMD units  
Flexible FP, Integer, AI operations

## 2x instruction issue rate

# AMD RDNA3 GPU

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>



**New Graphics Compute Die**

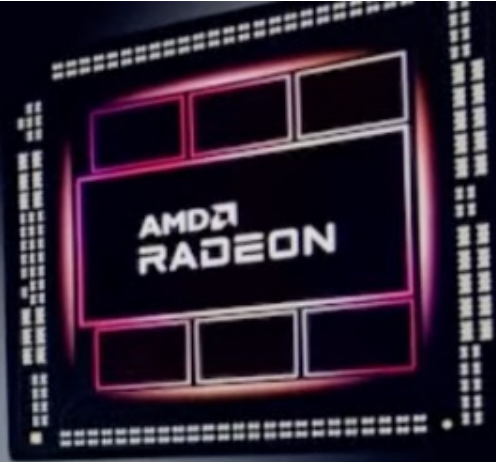
- Unified AMD RDNA™ 3 Compute Units
- New Display Engine
- New Dual Media Engine

**+54%**  
vs. prior generation

**World's most advanced gaming graphics**

**AMD RDNA 3**

<b>61</b> TFLOPS	<b>5.3TB/s</b> World's Fastest Chiplet Interconnect	<b>24GB</b> GDDR6	<b>58B</b> Transistors
---------------------	---	----------------------	---------------------------



# AMD RDNA3 GPU

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>

## Decoupled clocks

**2.3 GHz** shader clock speed

**2.5 GHz** front-end clock speed

Up to 25% power savings

+15% frequency

## Dedicated AI accelerators

2 per CU

New AI instructions

Improved AI throughput

up to

**2.7x** more performance

# AMD Radeon

11-3-22

<https://www.youtube.com/watch?v=hhwd6UgGVk4>



AMD Radeon™  
**RX 7900 XTX**

**96** CUs | **2.3** GHz | **24** GB | **2.1** | **AV1** | **355** W  
AMD RDNA™ 3 | Game Clock | 384-bit GDDR6 | DisplayPort™ | Encode & Decode | Total Board Power

The advertisement features a dark background with vibrant red and blue energy-like patterns. On the right side, a portion of the AMD Radeon RX 7900 XTX graphics card is visible, showing its dual-fan design and the 'RADEON' branding. In the center, a small image of a man in a black and white soccer jersey stands next to the specifications. The overall aesthetic is high-tech and dynamic.



# AMD Z1

# AMD Introduces Ryzen™ Z1 Series Processors, Expanding the "Zen 4" Lineup into Handheld Game Consoles

Zen 4 CPU + RDNA 3

**AMD Ryzen Z1 and AMD Ryzen Z1 Extreme processors bring ultimate portability and battery life to handheld PC gaming consoles**

SANTA CLARA, Calif., April 25, 2023 (GLOBE NEWSWIRE) -- Today, **AMD** (NASDAQ: AMD) introduced the new Ryzen Z1 Series processors, the ultimate high-performance processor for handheld PC gaming consoles<sup>1</sup>. The Ryzen Z1 Series features two high performance processors, the Ryzen Z1 and Ryzen Z1 Extreme, both offering industry-leading gaming experiences, uncompromising battery life, and featuring AMD RDNA™ 3 architecture-based graphics. AMD is partnering with Asus to launch the first Ryzen Z1 Series device with the Asus ROG Ally, a premium handheld PC console, featuring up to a Ryzen Z1 Extreme processor.

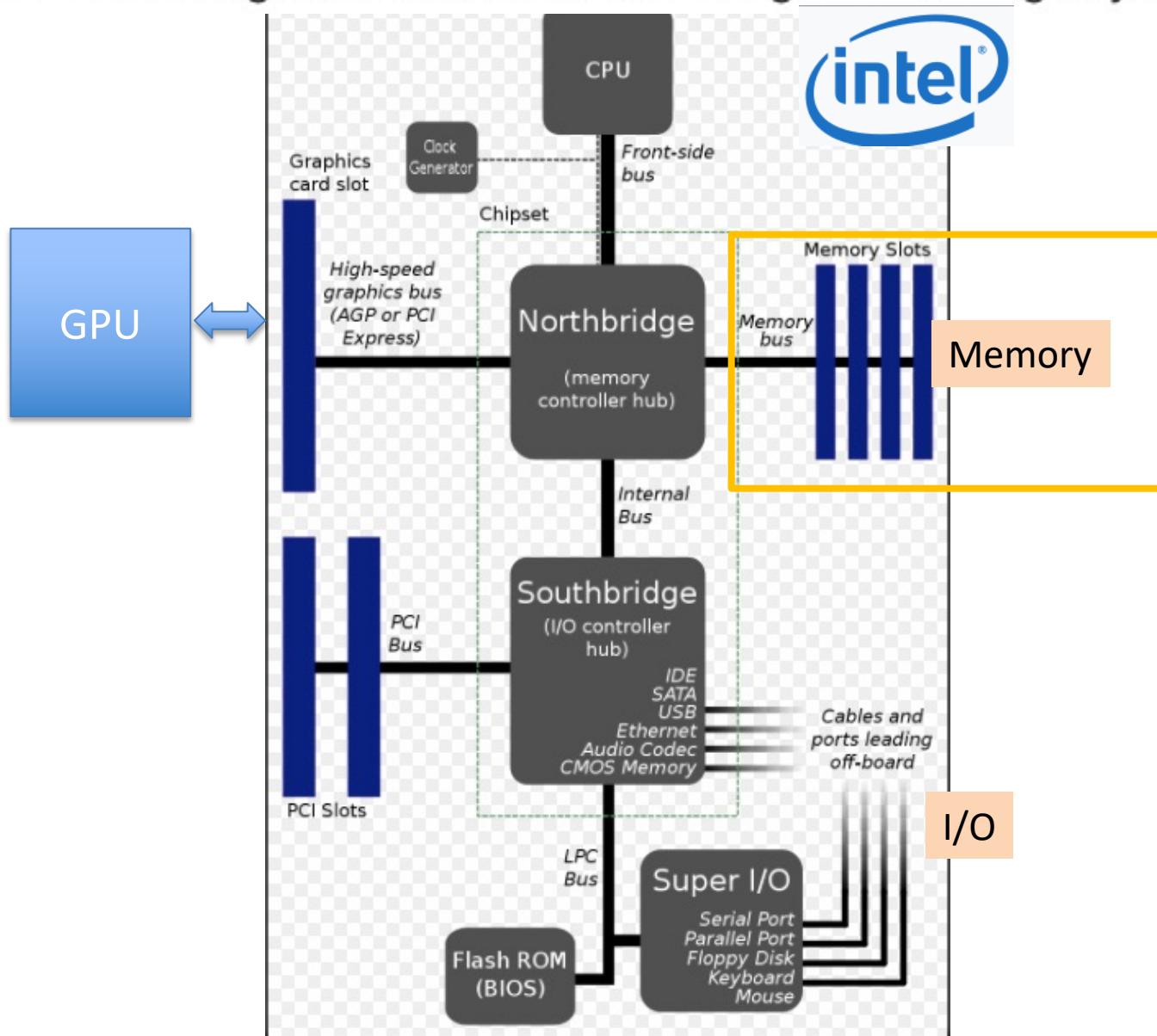
# Section

## GPU Products

# Intel

# New Intel Style IG Interface

The position of an integrated GPU in a northbridge/southbridge system layout



# Intel Xeon Max

**What the Intel Xeon Max CPU Delivers:** The Xeon Max CPU offers up to 56 performance cores constructed of four tiles and connected using Intel's embedded multi-die interconnect bridge (EMIB) technology, in a 350-watt envelope. Xeon Max CPUs contain 64GB of high bandwidth in-package memory, as well as PCI Express 5.0 and CXL1.1 I/O. Xeon Max CPUs will provide more than 1GB of high bandwidth memory (HBM) capacity per core, enough to fit most common HPC workloads. The Max Series CPU provides up to 4.8x better performance compared to competition on real-world HPC workloads.<sup>1</sup>

- AMX extensions boost AI performance and deliver 8x peak throughput over AVX-512 for INT8 with INT32 accumulation operations.<sup>2</sup>

# Intel Max GPU

The Xeon Max CPU is the first and only x86-based processor with high bandwidth memory, accelerating many HPC workloads without the need for code changes. The Max Series GPU is Intel's highest density processor, packing over 100 billion transistors into a 47-tile package with up to 128 gigabytes (GB) of high bandwidth memory. The oneAPI open software ecosystem provides a single programming environment for both new processors. Intel's 2023 oneAPI and AI tools will deliver capabilities to enable the Intel Max Series products' advanced features.

# Intel Max GPU

**What the Intel Max Series GPU Delivers:** Max Series GPUs deliver up to 128 X<sup>e</sup>-HPC cores, the new foundational architecture targeted at the most demanding computing workloads. Additionally, the Max Series GPU features:

- 408MB of L2 cache – the highest in the industry – and 64MB of L1 cache to increase throughput and performance.
- The only HPC/AI GPU with native ray tracing acceleration, designed to speed scientific visualization and animation.
- Workload benchmarks:
  - Finance: 2.4x performance gain over NVIDIA's A100 on Riskfuel credit option pricing.
  - Physics: 1.5x improvement over A100 for NekRS virtual reactor simulations.

# Intel Max GPU

Max Series GPUs will be available in several form factors to address different customer needs:

- Max Series 1100 GPU: A 300-watt double-wide PCIe card with 56 X<sup>e</sup> cores and 48GB of HBM2e memory. Multiple cards can be connected via Intel Xe Link bridges.
- Max Series 1350 GPU: A 450-watt OAM module with 112 X<sup>e</sup> cores and 96GB of HBM.
- Max Series 1550 GPU: Intel's maximum performance 600-watt OAM module with 128 X<sup>e</sup> cores and 128GB of HBM.

# Intel Xeon Max CPU + GPU

In January, we launched our strongest offerings for high performance computing (HPC) and AI ever with the 4th Gen Intel® Xeon® Scalable processors, Intel® Xeon® CPU Max Series and Intel® Data Center GPU Max Series. We also introduced the Intel® Data Center GPU Flex Series last year – a flagship product for media streaming, cloud gaming and AI inference – and the Habana® Gaudi®2 deep learning processor for training.

Co-designed with leading cloud service providers, enterprise and supercomputing customers, these products showcase key technical innovations, including the integration of high-bandwidth memory with x86 CPUs and advanced chiplet architectures. Intel's full data center and AI hardware portfolio, including our Xeon and Habana products, have been developed to help our customers solve the world's most difficult problems and train the largest AI models.

Accelerated computing and GPUs are among the fastest-growing segments of the computing market and central to Intel's long-term success. We are seeing great customer support and we continue to

# Intel XPU

The Intel Data Center Max Series GPU, code-named Rialto Bridge, is the successor to the Max Series GPU and is intended to arrive in 2024 with improved performance and a seamless path to upgrade. Intel is then planning to release the next major architecture innovation to enable the future of HPC. The company's upcoming XPU, code-named Falcon Shores, will combine X<sup>e</sup> and x86 cores on a single package. This groundbreaking new architecture will also have the flexibility to integrate new IPs from Intel and customers, manufactured using our IDM 2.0 model.

# Intel Supers

You have probably heard about Argonne National Laboratory, which will be deploying more than 60,000 Max Series GPUs and 20,000 Max Series CPUs to power the [Aurora supercomputer](#) this year. Aurora is expected to become the world's first supercomputer with 2 exaflops of peak performance. Deployment is going well, with Intel collaborating closely on [testing and development](#). Argonne expects the system to be accessible to early researchers by the third quarter of 2023.

Lawrence Livermore National Laboratories (LLNL) and Sandia National Laboratories are installing thousands of nodes of 4th Gen Intel Xeons in their CTS-2 systems – the supercomputing workhorse of the Department of Energy (DOE). LLNL's Intel Xeon-powered predecessor, JADE, recently contributed to the [breakthrough in fusion energy](#), helping to design the optimal package for laser induction.

Los Alamos National Laboratory (LANL), another DOE research center, is installing more than 10,000 Max Series CPUs for its [Crossroads supercomputer](#), which will power [national security](#) and [wildfire](#) research.

# Section

---

## GPU Products

# Nvidia

# Parallel Processing

Wikipedia

## Overview [\[ edit \]](#)

**Fermi Graphic Processing Units (GPUs)** feature 3.0 billion transistors and a schematic is sketched in Fig. 1.

- Streaming Multiprocessor (SM): composed of 32 **CUDA** cores (see Streaming Multiprocessor and CUDA core sections).
- GigaThread global scheduler: distributes thread blocks to SM thread schedulers and manages the context switches between threads during execution (see Warp Scheduling section).
- Host interface: connects the GPU to the CPU via a PCI-Express v2 bus (peak transfer rate of 8GB/s).
- DRAM: supported up to 6GB of GDDR5 DRAM memory thanks to the 64-bit addressing capability (see Memory Architecture section).
- Clock frequency: 1.5 GHz (not released by NVIDIA, but estimated by Insight 64).
- Peak performance: 1.5 TFlops.
- Global memory clock: 2 GHz.
- DRAM **bandwidth**: 192GB/s.

# Parallel Processing: GPU

Wikipedia

## Streaming multiprocessor [\[ edit \]](#)

Each SM features 32 single-precision CUDA cores, 16 load/store units, four Special Function Units (SFUs), a 64KB block of high speed on-chip memory (see L1+Shared Memory subsection) and an interface to the L2 cache (see L2 Cache subsection).

## Load/Store Units [\[ edit \]](#)

Allow source and destination addresses to be calculated for 16 threads per clock. Load and store the data from/to [cache](#) or [DRAM](#).

## Special Functions Units (SFUs) [\[ edit \]](#)

Execute transcendental instructions such as sin, cosine, reciprocal, and square root. Each SFU executes one instruction per thread, per clock; a warp executes over eight clocks. The SFU pipeline is decoupled from the dispatch unit, allowing the dispatch unit to issue to other execution units while the SFU is occupied.

## CUDA core [\[ edit \]](#)

*Integer Arithmetic Logic Unit (ALU):* Supports full 32-bit precision for all instructions, consistent with standard programming language requirements. It is also optimized to efficiently support 64-bit and extended precision operations.

## Floating Point Unit (FPU) [\[ edit \]](#)

Implements the new IEEE 754-2008 floating-point standard, providing the [fused multiply-add](#) (FMA) instruction for both single and double precision arithmetic. Up to 16 double precision fused multiply-add operations can be performed per SM, per clock.



Fig. 1.  
Conve  
sched  
execut  
caches



Die sh  
inside

# GPU Performance

Wikipedia

## Performance [\[ edit \]](#)

The theoretical [single-precision](#) processing power of a Fermi GPU in [GFLOPS](#) is computed as 2 (operations per FMA instruction per CUDA core per cycle) × number of CUDA cores × shader clock speed (in GHz). Note that the previous generation [Tesla](#) could dual-issue MAD+MUL to CUDA cores and SFUs in parallel, but Fermi lost this ability as it can only issue 32 instructions per cycle per SM which keeps just its 32 CUDA cores fully utilized.<sup>[2]</sup> Therefore, it is not possible to leverage the SFUs to reach more than 2 operations per CUDA core per cycle.

The theoretical double-precision processing power of a Fermi GPU is 1/2 of the single precision performance on GF100/110. However, in practice this double-precision power is only available on professional [Quadro](#) and [Tesla](#) cards, while consumer [GeForce](#) cards are capped to 1/8.<sup>[3]</sup>

# Nvidia GPU

## P&H Ch 6

### An introduction to the NVIDIA GPU architecture

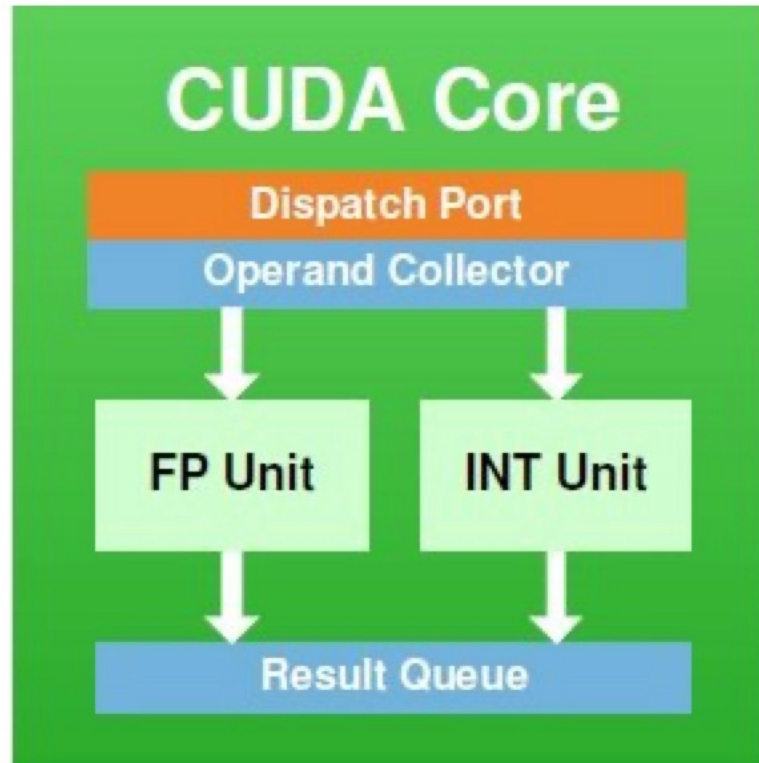
We use NVIDIA systems as our example as they are representative of GPU architectures. Specifically, we follow the terminology of the CUDA parallel programming language and use the Fermi architecture as the example.

Like vector architectures, GPUs work well only with data-level parallel problems. Both styles have gather-scatter data transfers, and GPU processors have even more registers than do vector processors. Unlike most vector architectures, GPUs also rely on hardware multithreading within a single multi-threaded SIMD processor to hide memory latency (see COD Section 6.4 (Hardware multithreading)).

A multithreaded SIMD processor is similar to a Vector Processor, but the former has many parallel functional units instead of just a few that are deeply pipelined, as does the latter.

As mentioned above, a GPU contains a collection of multithreaded SIMD processors; that is, a GPU is a MIMD composed of multithreaded SIMD processors. For example, NVIDIA has four implementations of the Fermi architecture at different price points with 7, 11, 14, or 15 multithreaded SIMD processors. To provide transparent scalability across models of GPUs with differing number of multithreaded SIMD processors, the Thread Block Scheduler hardware assigns blocks of threads to multithreaded SIMD processors. The figure below shows a simplified block diagram of a multithreaded SIMD processor.

# Nvidia Cuda



# Nvidia Cuda



This chip is roughly what the Ada Lovelace (AD102) RTX 4090 chip will look like. The full die is 16,384 CUDA cores arranged in 128 SM units. No GPU will ever be made that uses all 16,384 compute cores. A chip this size is almost 100% guaranteed to have failure points. Nvidia tests every chip and then

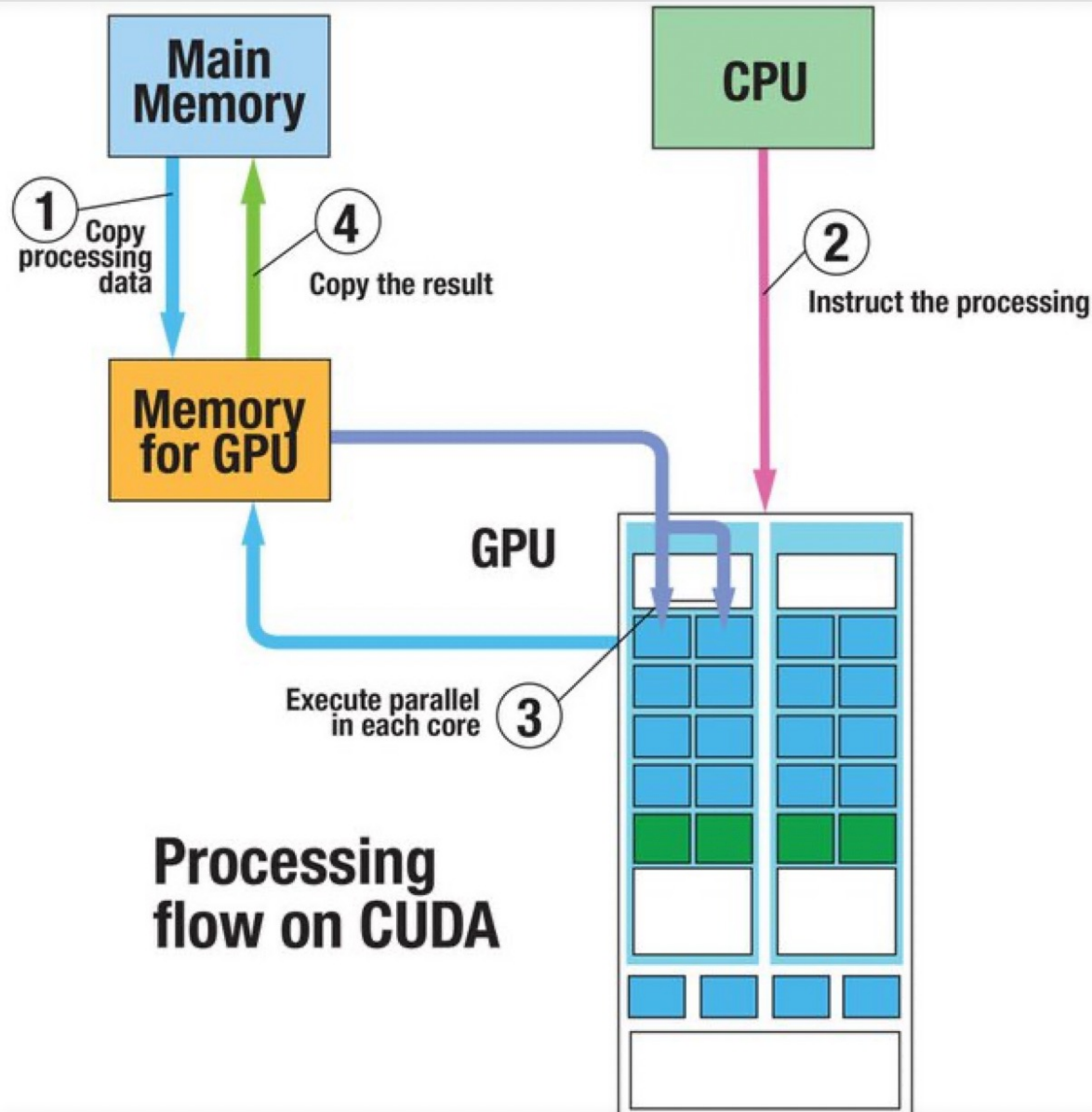
# Nvidia Cuda



This chip is roughly what the Ada Lovelace (AD102) RTX 4090 chip will look like. The full die is 16,384 CUDA cores arranged in 128 SM units. No GPU will ever be made that uses all 16,384 compute cores. A chip this size is almost 100% guaranteed to have failure points. Nvidia tests every chip and then isolates dead SM units. They can further test each SM for performance and then isolate the SM units that do not perform well.

If sixteen dead and underperforming SM units are removed from the chip, there are still 112 SM units remaining. That is still 14,336 CUDA cores left active on the chip. This is a reasonable expectation as to the final production spec of the RTX 4090.

# Nvidia Cuda



# Nvidia Cuda

You write a "kernel" code(and compile for GPU). Send it to GPU. Also send data to GPU. Then call compiled kernel code on GPU with data attached to it. Then wait until it completes. After that, you get results from GPU.

Something like this:

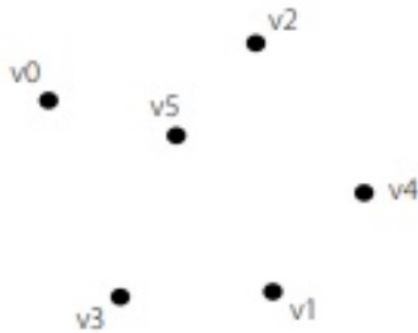
```
1 MY_API void kernelIncrement(int * data)
2 {
3     int workItemId = threadIdx.x+blockIdx.x*blockDim.x;
4     data[workItemId]++;
5 }
6 cudaMemcpy(gpuData, hostData, n, cudaMemcpyHostToDevice);
7 kernelIncrement<<<128,128>>>(gpuData);
8 cudaMemcpy(hostData, gpuData, n, cudaMemcpyDeviceToHost);
```

Kernel here is a function that runs on each streaming pipeline of GPU (64–192 of such pipelines make a SM unit "streaming multiprocessor"). If its about graphics acceleration, kernel is called as "shader". If it is computing it is "\_\_kernel" or "kernel". These are just names of functions exposed to developers through APIs like OpenGL, OpenCL and CUDA. Each so-called "GPGPU pipeline" of GPU runs this same code.

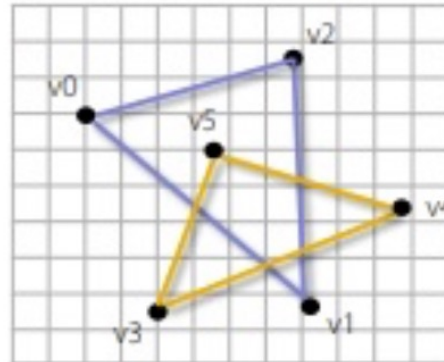
# Nvidia GPU



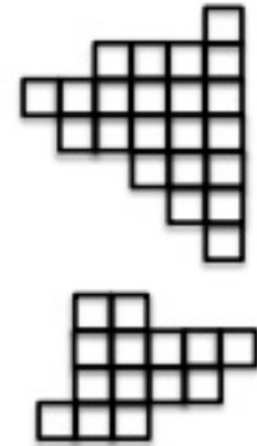
OpenGenus IQ: Computing Expertise & Legacy — Basic Graphics Processing Unit (GPU).



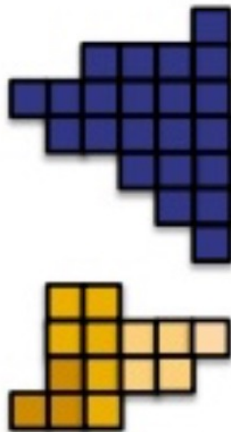
**Vertices**



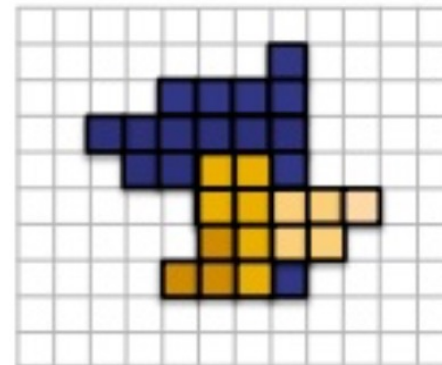
**Primitives**



**Fragments**



**Fragments (shaded)**

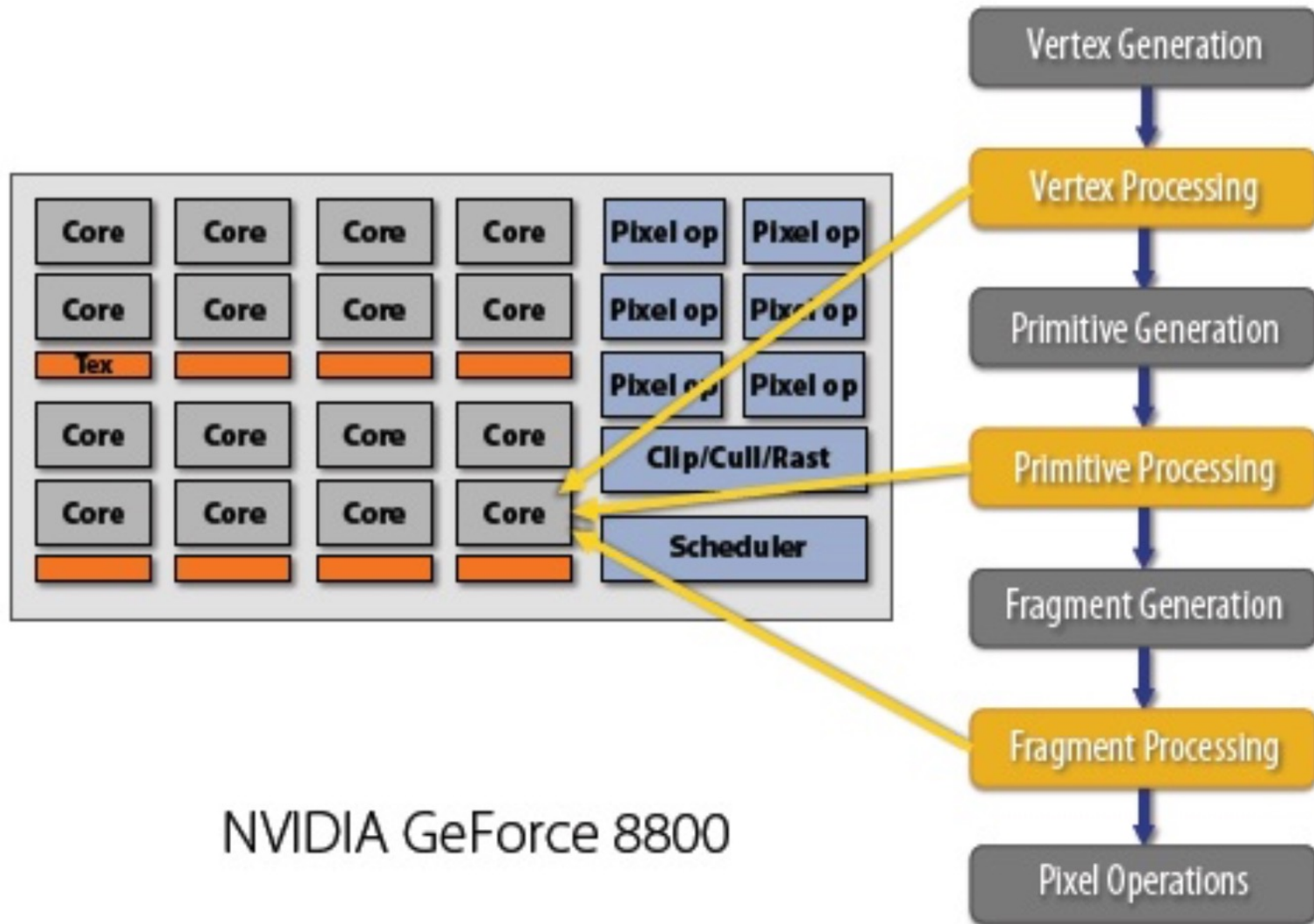


**Pixels**

# Nvidia GPU



OpenGenus IQ: Computing Expertise & Legacy — Basic Graphics Processing Unit (GPU).



# Nvidia

## GPU

### Nvidia Corporation



Headquarters at Santa Clara in 2018

<b>Type</b>	Public
<b>Traded as</b>	NASDAQ: NVDA <a href="#">🔗</a> NASDAQ-100 component S&P 100 component S&P 500 component
<b>ISIN</b>	US67066G1040 <a href="#">🔗</a>
<b>Industry</b>	Semiconductors Video games Consumer electronics Computer hardware
<b>Predecessor</b>	3dfx Interactive <a href="#">🔗</a>
<b>Founded</b>	April 5, 1993; 27 years ago
<b>Founders</b>	Jensen Huang Curtis Priem Chris Malachowsky
<b>Headquarters</b>	Santa Clara, California, U.S.

<b>Key people</b>	Jensen Huang (president & CEO) Colette M. Kress (CFO)
<b>Products</b>	Graphics processing units (GPU) Central processing units (CPU) Chipsets Drivers
<b>Revenue</b>	▲ US\$11.716 billion (2018) <sup>[1]</sup>
<b>Operating income</b>	▲ US\$3.804 billion (2018) <sup>[1]</sup>
<b>Net income</b>	▲ US\$4.141 billion (2018) <sup>[1]</sup>
<b>Total assets</b>	▲ US\$13.292 billion (2018) <sup>[1]</sup>
<b>Total equity</b>	▲ US\$9.342 billion (2018) <sup>[1]</sup>
<b>Number of employees</b>	13,227 (January 2019) <sup>[1]</sup>
<b>Subsidiaries</b>	NVIDIA Advanced Rendering Center, Mellanox Technologies
<b>Website</b>	<a href="http://www.nvidia.com">www.nvidia.com</a> <a href="#">🔗</a> <a href="http://developer.nvidia.com">developer.nvidia.com</a> <a href="#">🔗</a> <a href="http://www.geforce.com">www.geforce.com</a> <a href="#">🔗</a>

## Products

DGX Systems

DRIVE AGX

GeForce RTX 20-Series

NVIDIA Virtual GPU

Jetson

Quadro

SHIELD TV

Tesla

➤ #1 chip company in market cap

# Nvidia

## GPU

**Nvidia Corporation** ([/ɛnˈvɪdiə/ en-VID-ee-ə](#)),<sup>[note 1]</sup> is an American [multinational](#) technology company [incorporated in Delaware](#) and based in [Santa Clara, California](#).<sup>[2]</sup> It designs [graphics processing units](#) (GPUs) for the gaming and professional markets, as well as [system on a chip](#) units (SoCs) for the [mobile computing](#) and automotive market. Its primary GPU product line, labeled "[GeForce](#)", is in direct competition with [Advanced Micro Devices](#)' (AMD) "[Radeon](#)" products. Nvidia expanded its presence in the gaming industry with its handheld [Shield Portable](#), [Shield Tablet](#), and [Shield Android TV](#).

Since 2014,<sup>[citation needed]</sup> Nvidia has diversified its business focusing on four markets: gaming, professional visualization, data centers, and auto. Nvidia is also now focused on [artificial intelligence](#).<sup>[3]</sup>

In addition to GPU manufacturing, Nvidia provides [parallel processing](#) capabilities to researchers and scientists that allow them to efficiently run high-performance applications. They are deployed in [supercomputing](#) sites around the world.<sup>[4][5]</sup> More recently, it has moved into the [mobile computing](#) market, where it produces [Tegra](#) mobile processors for smartphones and tablets as well as vehicle navigation and entertainment systems.<sup>[6][7][8]</sup> In addition to [AMD](#), its competitors include [Intel](#), [Qualcomm](#), and [Arm](#) (e.g., because of [Denver](#), while Nvidia also licenses Arm's designs).

# Nvidia Products

GPU

## Product families [\[ edit \]](#)

Nvidia's family includes primarily graphics, wireless communication, PC processors and automotive hardware/software. Some families are listed below:

- [GeForce](#), consumer-oriented graphics processing products
- [Quadro](#), computer-aided design and digital content creation workstation graphics processing products
- [NVS](#), multi-display business graphics solution
- [Tegra](#), a [system on a chip](#) series for mobile devices
- [Tesla](#), dedicated general purpose GPU for high-end image generation applications in professional and scientific fields
- [nForce](#), a motherboard chipset created by Nvidia for Intel (Celeron, Pentium and Core 2) and AMD (Athlon and Duron) microprocessors
- Nvidia Grid, a set of hardware and services by Nvidia for graphics virtualization
- Nvidia Shield, a range of gaming hardware including the [Shield Portable](#), [Shield Tablet](#) and, most recently, the [Shield Android TV](#)
- Nvidia Drive automotive solutions, a range of hardware and software products for assisting car drivers. The [Drive PX-series](#) is a h platform aimed at autonomous driving through deep learning,<sup>[70]</sup> while Driveworks is an operating system for driverless cars.<sup>[71]</sup>

# Nvidia Products

## GPU

On May 6, 2016, Nvidia unveiled the first [GeForce 10 series](#) GPUs, the GTX 1080 and 1070, based on the company's new [Pascal microarchitecture](#). Nvidia claimed that both models outperformed its [Maxwell](#)-based Titan X model; the models incorporate [GDDR5X](#) and GDDR5 memory respectively, and use a 16 nm manufacturing process. The architecture also supports a new hardware feature known as simultaneous multi-projection (SMP), which is designed to improve the quality of multi-monitor and [virtual reality](#) rendering.<sup>[35][36][37]</sup> Laptops that include these GPUs and are sufficiently thin – as of late 2017, under 0.8 inches (20 mm) – have been designated as meeting Nvidia's "Max-Q" design standard.<sup>[38]</sup>

Nvidia officially released the NVIDIA TITAN V on December 7, 2017.<sup>[42][43]</sup>

Nvidia officially released the Nvidia Quadro GV100 on March 27, 2018.<sup>[44]</sup>

Nvidia officially released RTX 2080GPUs September 27, 2018.

In 2018, [Google](#) announced that Nvidia's Tesla P4 graphic cards would be integrated into Google Cloud service's artificial intelligence.<sup>[45]</sup>

On March 11, 2019, Nvidia announced a deal to buy [Mellanox Technologies](#) for \$6.9 billion<sup>[46]</sup> to substantially expand its footprint in the high-performance computing market.

In May 2019, Nvidia announced new RTX Studio laptops. The creators say that the new laptop is going to be seven times faster than a top-end MacBook Pro with a Core i9 and AMD's Radeon Pro Vega 20 graphics in apps like Maya and RedCine-X Pro.<sup>[47]</sup>

In August 2019, Nvidia announced [Minecraft](#) RTX, an official Nvidia-developed patch for the game adding real-time DXR raytracing exclusively to the Windows 10 version of the game. The whole game is, in Nvidia's words, "refit" with path tracing, which dramatically affects the way light, reflections and shadows work inside the engine.<sup>[48]</sup>

In May 2020, Nvidia's top scientists developed an [open-source ventilator](#) in order to address the shortage resulting from the global [coronavirus pandemic](#).<sup>[49]</sup>

On May 14, 2020, Nvidia officially announced their Ampere GPU microarchitecture and the Nvidia A100 GPU accelerator.<sup>[50][51]</sup>

## GAMING



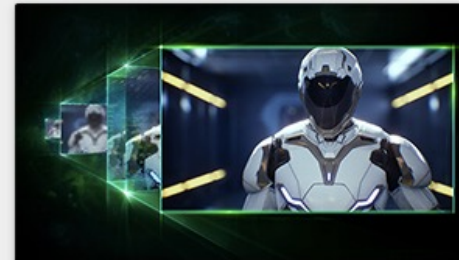
### Embark On Your Journey

Buy GeForce RTX™, Get Death Stranding.



### GeForce Laptops

Thin and powerful GeForce laptops are designed to level-up gaming performance and accelerate creativity. Worker by day. Warrior by night.



### DLSS 2.0

Max FPS. Max Quality. Powered by AI.



### Frames Win Games

144+ FPS Gaming.

## AI NEWS



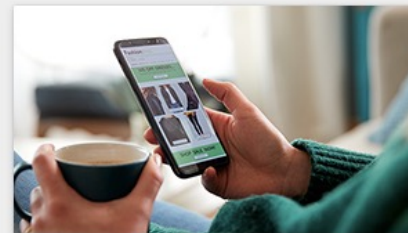
### The Ultimate Quadro Companion

Access advanced productivity tools and take your Quadro® graphics card to the next level with NVIDIA® Quadro Experience™.



### NVIDIA Untethers Enterprise AR and VR Streaming with CloudXR

NVIDIA CloudXR™ delivers wireless VR and AR streaming of graphics-rich content across 5G and Wi-Fi networks.



### NVIDIA Merlin Recommender Application Framework

NVIDIA makes recommender systems more accessible to businesses with new end-to-end pipeline for ingesting, training, and deploying GPU-accelerated recommender systems.



### GPU-Accelerated Apache Spark

The RAPIDS Accelerator for Apache Spark helps data teams increase the performance of their end-to-end analytics and machine learning pipelines without code changes, substantially lowering infrastructure costs.

# Nvidia Fermi

COMP222

## Fermi (microarchitecture)

Wikipedia

From Wikipedia, the free encyclopedia

**Fermi** is the codename for a [graphics processing unit](#) (GPU) [microarchitecture](#) developed by [Nvidia](#), first released to retail in April 2010, as the successor to the [Tesla](#) microarchitecture. It was the primary microarchitecture used in the [GeForce 400 series](#) and [GeForce 500 series](#). It was followed by [Kepler](#), and used alongside Kepler in the [GeForce 600 series](#), [GeForce 700 series](#), and [GeForce 800 series](#), in the latter two only in [mobile](#) GPUs. In the workstation market, Fermi found use in the [Quadro](#) x000 series, Quadro NVS models, as well as in [Nvidia Tesla](#) computing modules. All desktop Fermi GPUs were manufactured in 40 nm, mobile Fermi GPUs in 40 nm and 28 nm. Fermi is the oldest microarchitecture from NVIDIA that received support for the Microsoft's rendering API Direct3D 12 `feature_level 11`.

The architecture is named after [Enrico Fermi](#), an Italian physicist.

### Nvidia Fermi

<b>Release date</b>	April 2010
<b>Fabrication process</b>	40 nm and 28 nm
<b>History</b>	
<b>Predecessor</b>	<a href="#">Tesla 2.0</a>
<b>Successor</b>	<a href="#">Kepler</a>

# Nvidia New Chips

April 2022

## NVIDIA INVESTOR DAY

### NEW PRODUCTS ANNOUNCED:

- Hopper: silicon architecture
- H100: first datacenter built with Hopper
- Grace: CPU superchip

**EYE FOR AN  
AI**

BA-BA-  
BUYBACK

TES-JA

CURVE YOUR  
ENTHUSIASM

MARKET  
INTERNALS

NVIDIA REVEALS NEW PRODUCTS AT INVESTOR DAY

NVIDIA TEASES "FASTEST AI SUPERCOMPUTER"

# Nvidia New Chips

March 2022



**Brett Bergan** · [Follow](#)

Building PC's for 25 years · Updated Mar 29

**Why is it predicted that Moore's law will/is no longer be accurate?  
Can't we just make bigger processors?**

Nvidia just announced it new whopper of a GPU with **80 billion transistors** fabbed at 4nm. This up from 54 billion transistors on the Ampere A100.

One square inch is 645mm<sup>2</sup>

This new (Grace) Hopper GPU will be a massive 900mm<sup>2</sup> and potentially use 600W but capable of a staggering 48 FP32 TFLOPS. Just to be clear that is about three RTX 2080 Ti's.

Performance per watt will be about double that of a RTX 2080 Ti.

# Section

---

## GPU Products

# ARM

## Mali (GPU)

From Wikipedia, the free encyclopedia

The **Mali** series of [graphics processing units](#) (GPUs) and multimedia processors are [semiconductor intellectual property cores](#) produced by [ARM Holdings](#) for licensing in various [ASIC](#) designs by ARM partners.

Mali GPUs were developed by [Falanx Microsystems A/S](#), which was a [spin-off](#) of a research project from the [Norwegian University of Science and Technology](#).<sup>[1]</sup> [Arm Holdings](#) acquired Falanx Microsystems A/S on June 23, 2006 and renamed the company to [Arm Norway](#).<sup>[2]</sup>

### Technical details [\[ edit \]](#)

Like other embedded IP cores for 3D rendering [acceleration](#), the Mali GPU does not include [display controllers](#) driving monitors, in contrast to common desktop [video cards](#). Instead, the Mali ARM core is a pure 3D engine that renders graphics into memory and passes the rendered image over to another core to handle display.

ARM does, however, license display controller SIP cores independently of the Mali 3D accelerator SIP block, e.g. Mali DP500, DP550 and DP650.<sup>[3]</sup>

ARM also supplies tools to help in authoring [OpenGL ES shaders](#) named *Mali GPU Shader Development Studio* and *Mali GPU User Interface Engine*.

Display controllers such as the ARM HDLCD display controller are available separately.<sup>[4]</sup>

# Mali GPU Timeline

GPU

## Variants [\[ edit \]](#)

The Mali core grew out of the cores previously produced by Falanx and currently constitute:

Model ↕	Micro- archi- tecture ↕	Type ↕	Launch date ↕	Shader core count ↕	Fab (nm) ↕	Die size (mm <sup>2</sup> ) ↕	Core clock rate (MHz) ↕	L2 cache size ↕
<a href="#">Mali-55/110</a> ↗	?	Fixed function pipeline <sup>[5]</sup>	2005 <a href="#">↗</a> <sup>[permanent dead link]</sup>	1	?	?	?	N/A
<a href="#">Mali-200</a> ↗	Utgard <sup>[6]</sup>	Programmable pipeline <sup>[7]</sup>	2007 <sup>[8]</sup>	1	?	?	?	N/A
<a href="#">Mali-300</a> ↗			?	1	40 28	?	500	8 KiB
<a href="#">Mali-400 MP</a> ↗			2008	1–4	40 28	?	200–600	8-256 KiB
<a href="#">Mali-450 MP</a> ↗			2012	1–8	40 28	?	300–750	8-512 KiB
<a href="#">Mali-470 MP</a> ↗			2015	1–4	40 28	?	250–650	8–256 KiB
<a href="#">Mali-T604</a> <a href="#">↗</a> <sup>[9]</sup>	Midgard		?	1–4	32 28	?	533	

# Mali GPU Timeline

GPU								
Mali-G71 <a href="#">↗</a>	Bifrost 2 <sup>nd</sup> gen	Unified shader model + Unified memory + scalar, clause-based ISA	Q2 2016	1-32	16 14 10	?	546-1037	128-2048 KiB
Mali-G52 <a href="#">↗</a>			Q1 2018	1-4 (2 or 3 EU per core)	7 16	?	850	
Mali-G72 <a href="#">↗</a>			Q2 2017	1-32	16 12 10	1.36 mm <sup>2</sup> per shader core at 10 nm <sup>[29]</sup>	572-800	128-2048 KiB
Mali-G76 <a href="#">↗</a>			Q2 2018	4-20	12 8 7	?	600-800	512-4096 KiB
Mali-G57 <a href="#">↗</a>	Valhall 1 <sup>st</sup> gen	Superscalar engine + Unified memory + simplified scalar ISA	Q2 2019	1-6	7	?	?	64-512 KiB
Mali-G77 <a href="#">↗</a>			Q2 2019	7-16	7	?	850	512-4096 KiB
Model	Micro-architecture	Type	Launch date	Shader core count	Fab (nm)	Die size (mm <sup>2</sup> )	Core clock rate (MHz)	Max L2 cache size