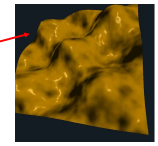
GPU 101

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

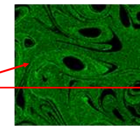
How Have You Been Able to Gain Access to GPU Power?

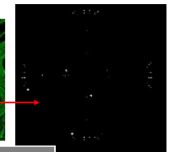
There have been three ways:

1. Write a graphics display program (≥ 1985)



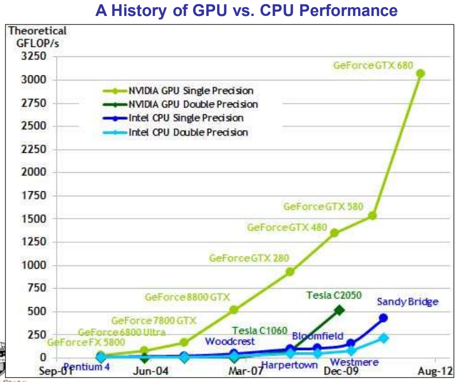
2. Write an application that looks like a graphics display program, but uses the fragment shader to do some per-node computation (≥ 2002)





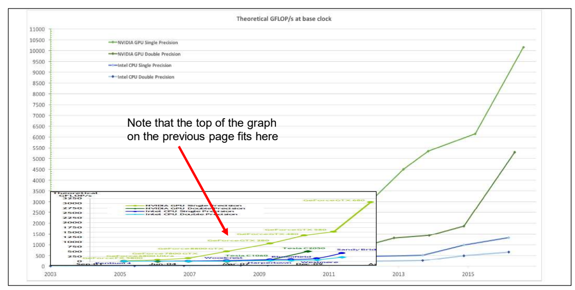
3. Write in OpenCL or CUDA, which looks like C++ (≥ 2006)

Why do we care about GPU Programming? A History of GPU vs. CPU Performance



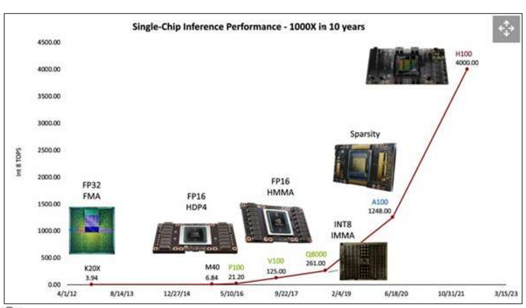
NVIDIA

Why do we care about GPU Programming? A History of GPU vs. CPU Performance



NVIDIA

Or, in AI Inference Performance



NVIDIA

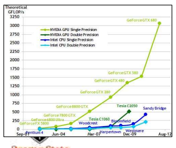
The "Core-Score". How can this be?

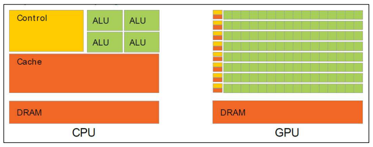
Why have GPUs Been Outpacing CPUs in Performance?

7

Due to the nature of graphics computations, GPU chips are customized to stream **regular data**. General CPU chips must be able to handle **irregular data**.

Another reason is that GPU chips do not need the significant amount of **cache** space that occupies much of the real estate on general-purpose CPU chips. The GPU die real estate can then be re-targeted to hold more cores and thus to produce more processing power.





NVIDIA

Oregon State University
Computer Graphics

Why have GPUs Been Outpacing CPUs in Performance?

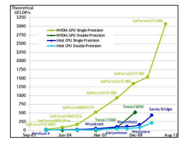
8

Another reason is that general CPU chips contain on-chip logic to do **branch prediction** and **out-of-order execution.** This, too, takes up chip die space.

But CPU chips can handle more general-purpose computing tasks.

So, which is better, a CPU or a GPU?

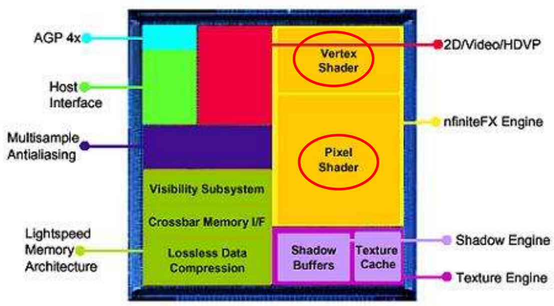
It depends on what you are trying to do!



Oregon State University
Computer Graphics

Originally, Parts of GPU Chips were very Task-specific

9



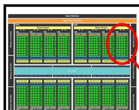
Oregon State University
Computer Graphics

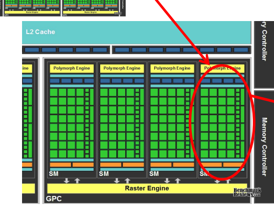
Today's GPU Devices are not Task-specific – They Can Be *Dynamically* Re-purposed for any GPU Function

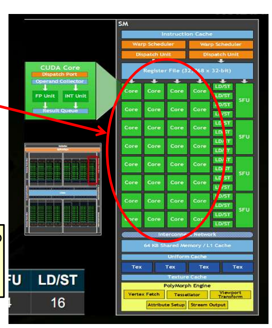
10

Consider the architecture of the NVIDIA 4090

11







128 Streaming Multiprocessors (SMs) / chip

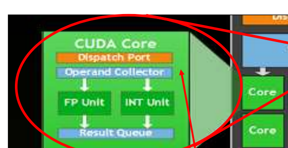
128 cores / SM

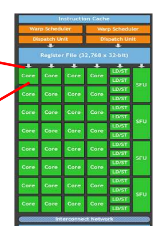
Wow! **16,384** cores / chip? Really?

Oregon State University
Computer Graphics

What is a "Core" in the GPU Sense?

12





Look closely, and you'll see that NVIDIA really calls these "CUDA Cores"

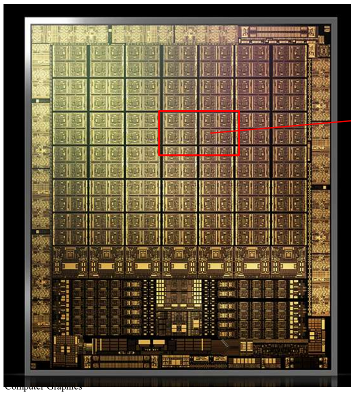
Look even more closely and you'll see that these CUDA Cores have no control logic – they are **pure compute units, i.e. ALUs**. (The surrounding SM has all the control logic.)

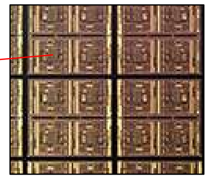
Other vendors refer to these as "Lanes". You can also think of them as 128-way SIMD.

Oregon State University
Computer Graphics

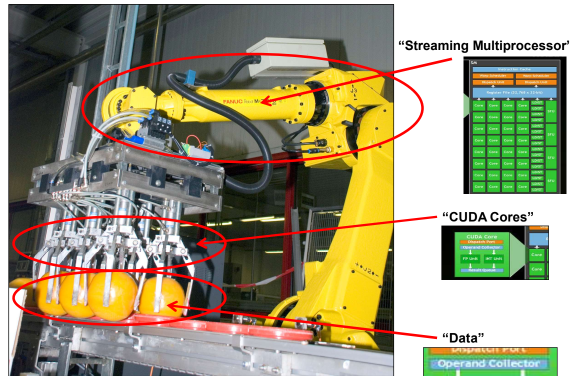
NVIDIA's Ampere Chip

NVIDIA





A Mechanical Equivalent…



http://news.cision.com

Graphics Card	RTX 5090	RTX 4090	RTX 3090	RTX 2080 Ti
Architecture	GB202	AD102	GA102	TU102
Process Technology	TSMC 4N	TSMC 4N	Samsung 8N	TSMC 12FFN
Transistors (Billion)	92.2	76.3	28.3	18.6
Die size (mm^2)	750	608.4	628.4	754
SMs / CUs / Xe-Cores	170	128	82	68
GPU Shaders (ALUs)	21760	16384	10496	4352
Tensor / AI Cores	680	512	328	544
Ray Tracing Cores	170	128	82	68
Boost Clock (MHz)	2407	2520	1695	1545
VRAM Speed (Gbps)	28	21	19.5	14
VRAM (GB)	32	24	24	11
VRAM Bus Width	512	384	384	352
L2 / Infinity Cache	96	72	6	5.5
Render Output Units	176	176	112	88
Texture Mapping Units	680	512	328	272
TFLOPS FP32 (Boost)	104.8	82.6	35.6	13.4
TFLOPS FP16 (FP4/ FP8 TFLOPS)	838 (3352)	661 (1321)	285	108

Tom's Hardware

The Bottom Line is This

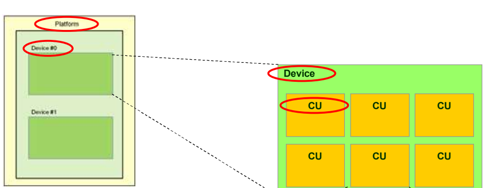
It is difficult to *directly* compare a CPU with a GPU. They are optimized to do different things.

So, let's use the information about the architecture as a way to consider what CPUs should be good at and what GPUs should be good at

CPU	**GPU**
General purpose programming	Data parallel programming
Multi-core under user control	Little user control
Irregular data structures	Regular data structures
Irregular flow control	Regular Flow Control

BTW,
The general term in the OpenCL world for an SM is a **Compute Unit**.
The general term in the OpenCL world for a CUDA Core is a **Processing Element**.

Compute Units and Processing Elements are Arranged in Grids

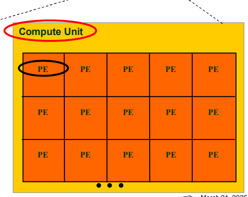


A GPU Platform can have one or more **Devices** (a Device is a GPU chip).

Each GPU **Device** is organized as a grid of **Compute Units.**

Each Compute Unit is organized as a grid of **Processing Elements**.

So, in NVIDIA terms, their 5090 GPU has 170 Compute Units, each of which has 128 Processing Elements, for a grand total of 21,760 Processing Elements.



Thinking ahead to CUDA and OpenCL…

How can GPUs execute General C Code Efficiently?

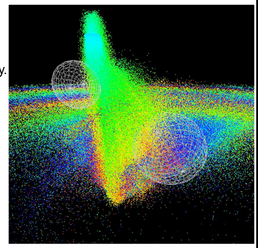
- Ask them to do what they do best. Unless you have a very intense ***Data Parallel*** application, don't even *think* about using GPUs for computing.
- GPU programs expect you to not just have a few threads, but to have ***thousands*** of them!
- Each thread executes the same program (called the *kernel*), but operates on a different small piece of the overall data
- Thus, you have many, many threads, all waking up at about the same time, all executing the same kernel program, all expecting to work on a small piece of the overall problem.
- CUDA and OpenCL have built-in functions so that each thread can figure out which thread number it is, and thus can figure out what part of the overall job it's supposed to work on.
- When a set of threads gets blocked somehow (a memory access, waiting for information from another thread, etc.), the processor switches to executing another set of threads to work on.

So, the Trick is to Break your Problem into Many, Many Small Pieces

Particle Systems are a great example.

1. Have one thread per *each particle*.
2. Put all of the initial parameters into an array in GPU memory.
3. Tell each thread what the current **Time** is.
4. Each thread then computes its particle's position, color, etc. and writes it into arrays in GPU memory.
5. The CPU program then initiates OpenGL drawing of the information in those arrays.

Note: once setup, the data never leaves GPU memory!

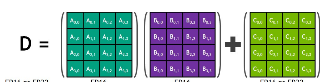


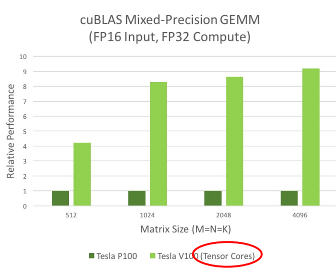
Ben Weiss

Something New – Tensor Cores

NVIDIA

Tensor Cores Accelerate Fused-Multiply-Add Arithmetic





What is Fused Multiply-Add?

Many scientific and engineering computations take the form:
D = A + (B*C);

A "normal" multiply-add would likely handle this as:
tmp = B*C;
D = A + tmp;

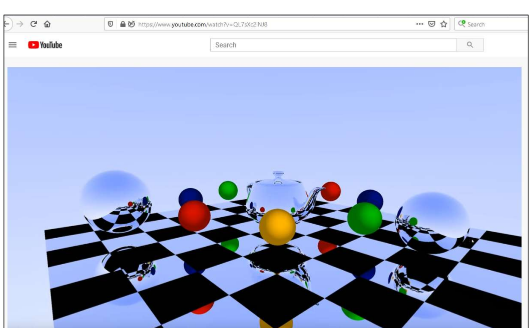
A "fused" multiply-add does it all at once, that is, when the low-order bits of B*C are ready, they are immediately added into the low-order bits of A at the same time the higher-order bits of B*C are being multiplied.

Consider a Base 10 example: **789 + (123*456)**

```
      123
    x 456
      738
     615
    492
    + 789   Can start adding the 9 the moment the 8 is produced!
   56,877
```

Note: "Normal" A+(B*C) ≠ "FMA" A+(B*C)

Something Even Newer – Ray-Trace Cores



https://www.youtube.com/watch?v=QL7sXc2iNJ8

NVIDIA

There are Two Approaches to Combining Your CPU and GPU Programs

1. Combine both the CPU and GPU code in the same code file. Somehow mark what part is CPU code and what part is GPU code. The CPU compiler compiles just its part of that file. The GPU compiler compiles just its part of that file.
2. Have two separate programs: a .cpp and a .somethingelse that get compiled separately by a CPU compiler and a GPU compiler.

Advantages of Each

1. The CPU and GPU sections of the code know about each others' intents. Also, they can share common structs, #define's, etc.
2. It's potentially cleaner to look at each section by itself. Also, the GPU code can be easily used in combination with other CPU programs.

Who are we Talking About Here?

1 = NVIDIA's CUDA

2 = Khronos's OpenCL

We will talk about each of these separately – stay tuned!

Looking ahead:
If threads all execute the same program, what happens on flow divergence?

```
if( a > b )
        Do This;
else
        Do That;
```

1. On a GPU, the line "if(a > b)" creates a vector of 0/1 Boolean values giving the results of the if-statement for each thread. This becomes a "bitmask".
2. Then, the GPU executes both parts of the divergence:
   Do This;
   Do That;
3. During that execution, anytime a value wants to be stored, the bitmask is consulted, and the storage only happens if that thread's location in the bitmask is a 1.

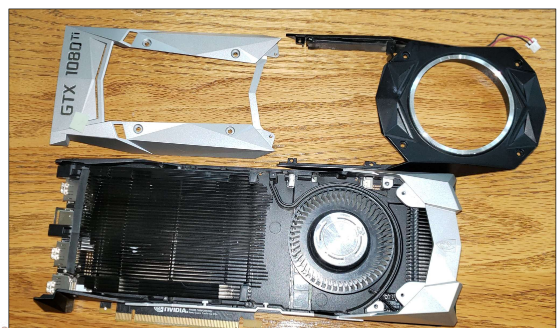
- GPUs were originally designed for the streaming-ness of computer graphics.
- That same streaming-ness can also be applied to data-parallel computing.
- GPUs are better for some things. CPUs are better for others.

Dismantling a Graphics Card

This is an Nvidia 1080 ti card – one that died on us. It willed its body to education.

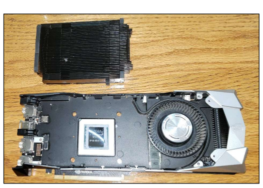
Dismantling a Graphics Card

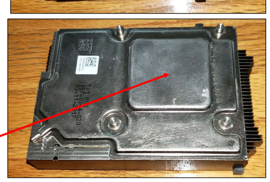
Removing the covers:



Dismantling a Graphics Card

Removing the heat sink:

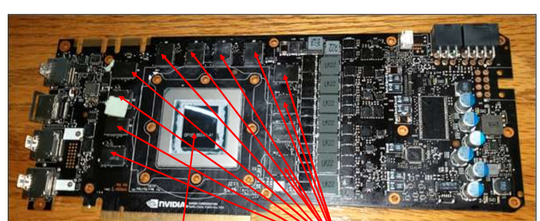




This transfers heat from the GPU Chip to the cooling fins

Dismantling a Graphics Card

Removing the fan assembly reveals the board:



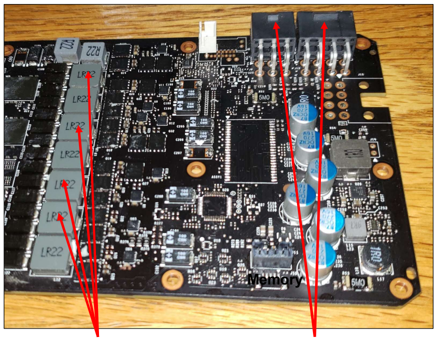
GPU Chip

Memory

Dismantling a Graphics Card

31

Power half of the board:



Power distribution

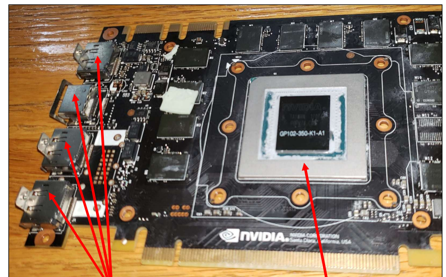
Power input

Oregon State University Computer Graphics

Dismantling a Graphics Card

32

Graphics half of the board:



Video out

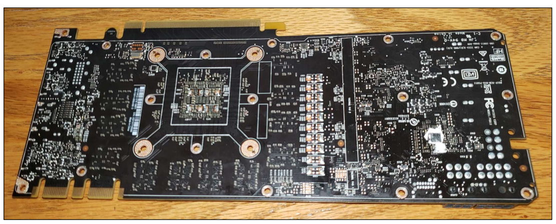
GPU Chip
This one contains 7.2 billion transistors!
The newer cards contain 70+ billion transistors.
(Thank you, Moore's Law)

Oregon State University Computer Graphics

Dismantling a Graphics Card

33

Underside of the board:

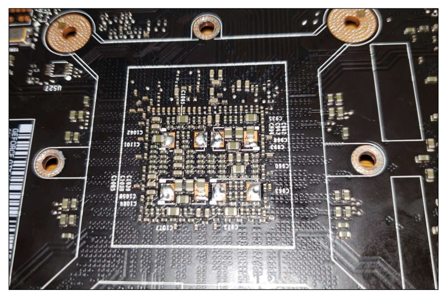


Oregon State University Computer Graphics

Dismantling a Graphics Card

34

Underside of where the GPU chip attaches:



Here is a fun video of someone explaining the different parts of this same card:
https://www.youtube.com/watch?v=dSCNf9DIBGE

Oregon State University Computer Graphics