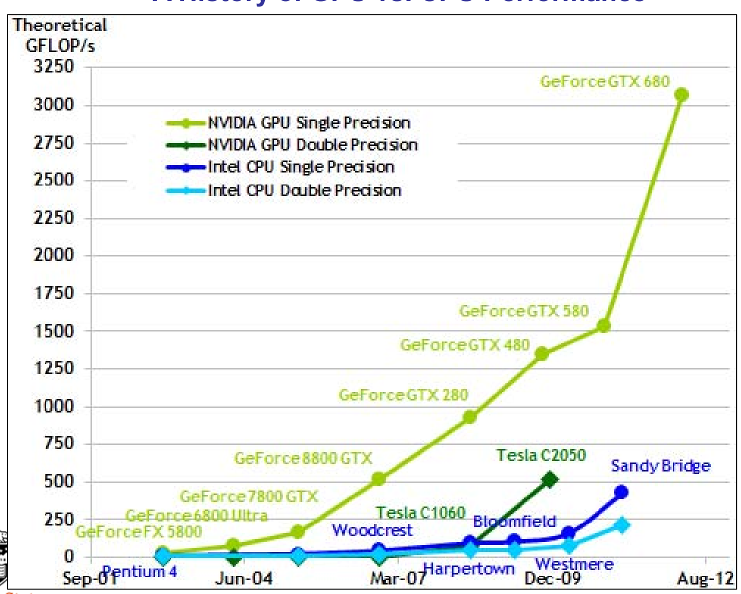
GPU 101

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

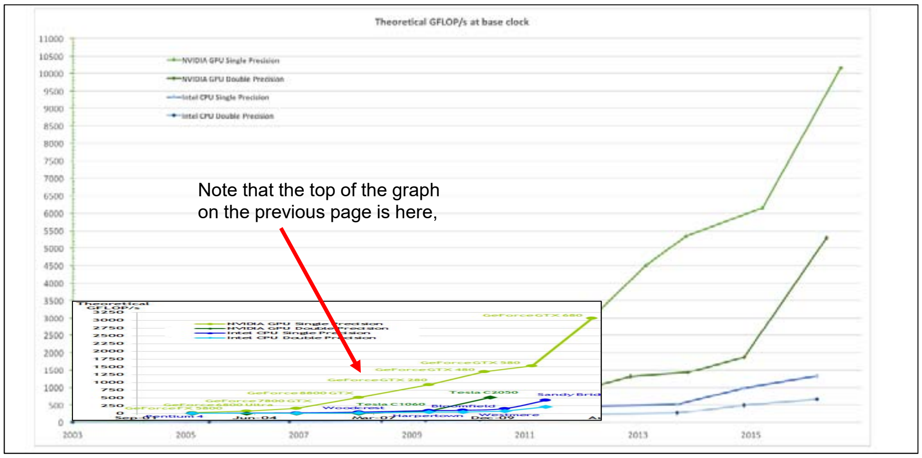
Why do we care about GPU Programming? A History of GPU vs. CPU Performance



NVIDIA

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

3

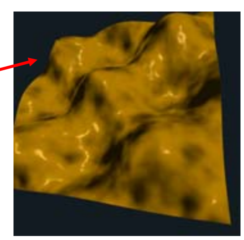


NVIDIA

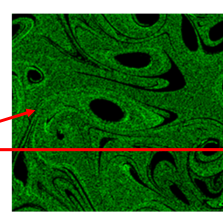
How Have You Been Able to Gain Access to GPU Power?

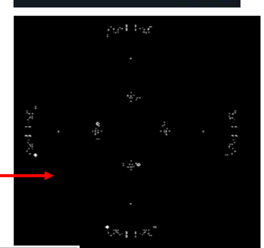
4

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)

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

5

Oregon State University
Computer Graphics

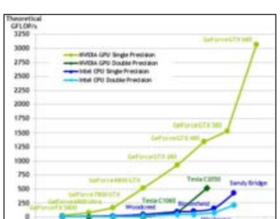
mjb – March 24, 2020

Why have GPUs Been Outpacing CPUs in Performance?

6

Due to the nature of graphics computations, GPU chips are customized to handle **streaming 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

mjb – March 24, 2020

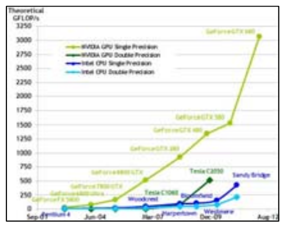
Why have GPUs Been Outpacing CPUs in Performance?

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.

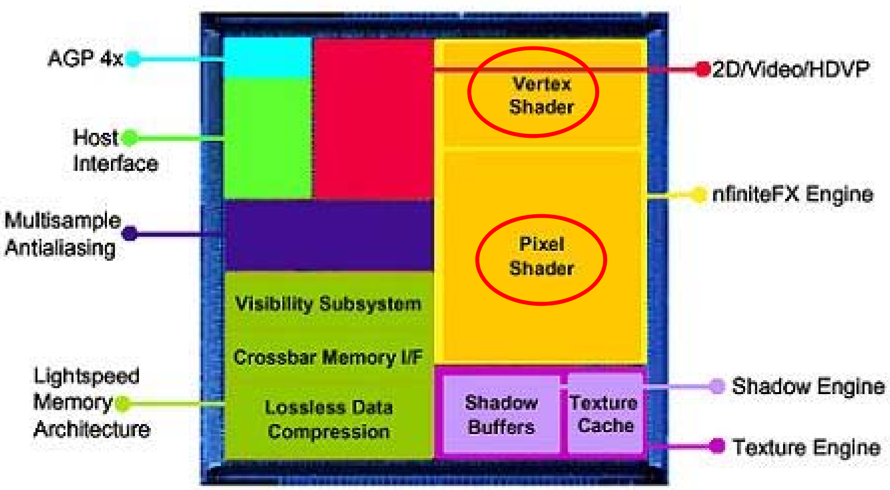
But, CPU chips can handle more general-purpose computing tasks.

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

It depends on what you are trying to do!

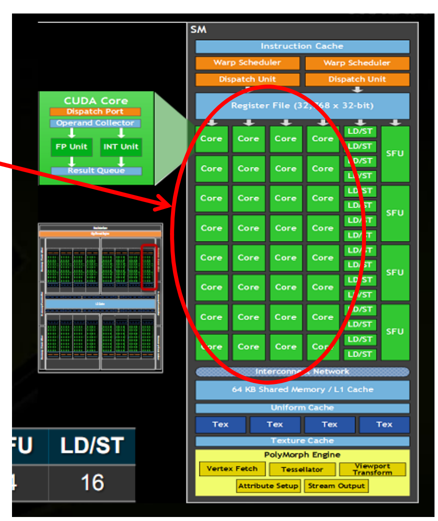


Originally, GPU Devices were very task-specific



Today's GPU Devices are much less task-specific

Consider the architecture of the NVIDIA Tesla V100's that we have in our *GDX System*



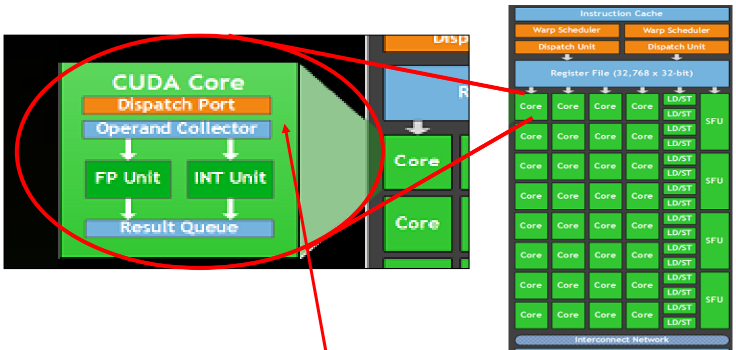
84 Streaming Multiprocessors (SMs) / chip

64 cores / SM

Wow! **5,396** cores / chip? Really?

Oregon State University
Computer Graphics

What is a "Core" in the GPU Sense?

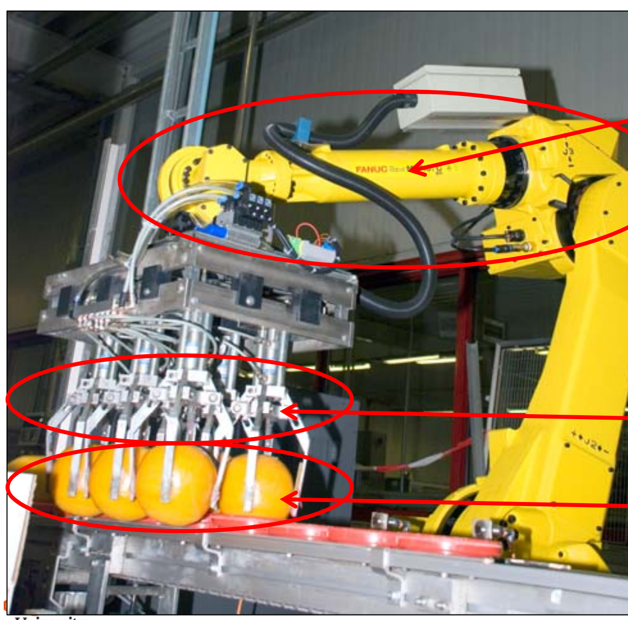


Look closely, and you'll see that NVIDIA really calls these "CUDA Cores"

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

Other vendors refer to these as "Lanes". You might also think of them as 64-way SIMD.

A Mechanical Equivalent…



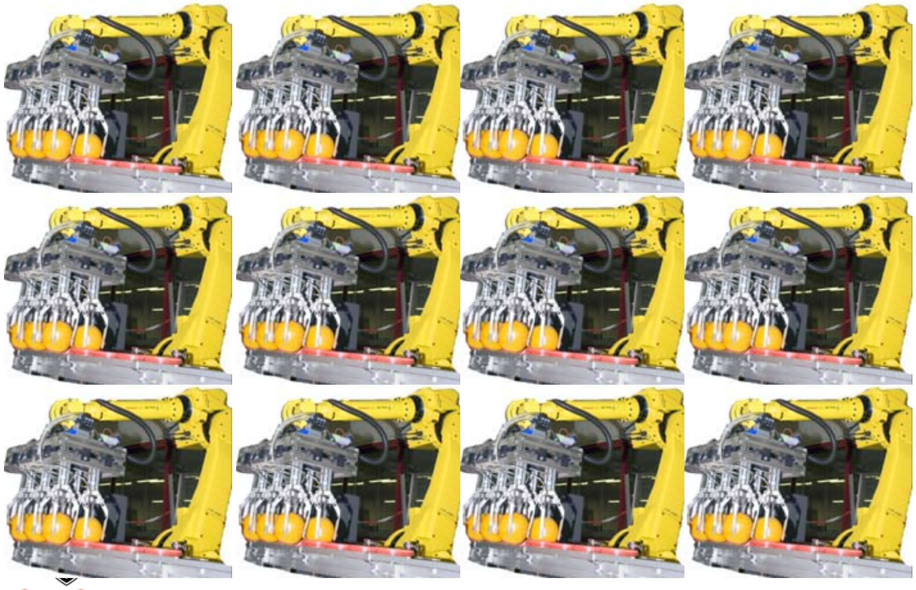
"Streaming Multiprocessor"

"CUDA Cores"

"Data"

http://news.cision.com

How Many Robots Do You See Here?



12? 72? Depends what you count as a "robot".

A Spec Sheet Example

Streaming Multiprocessors

CUDA Cores per SM

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOPS[1]	5	6.8	10.6	15.7
Peak FP64 TFLOPS[1]	1.7	.21	5.3	7.8
Peak Tensor TFLOPS[1]	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB	20480 KB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm²	601 mm²	610 mm²	815 mm²
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

NVIDIA

The Bottom Line is This

So, the Titan Xp has 30 processors per chip, each of which is optimized to do 128-way SIMD. This is an amazing achievement in computing power. But, it is obvious that it is difficult to *directly* compare a CPU with a GPU. They are optimized to do different things.

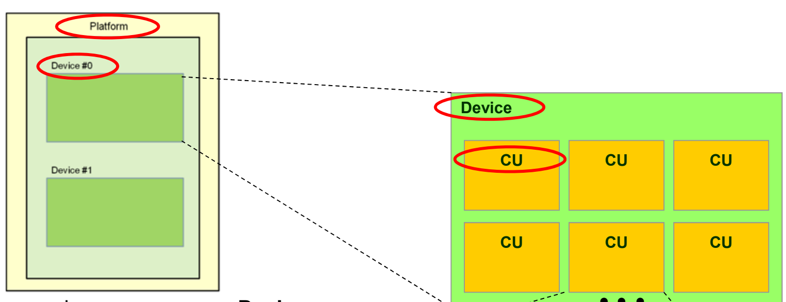
So, let's use the information about the architecture as a way to consider what CPUs should be good at and what GPUs should be good at

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**.

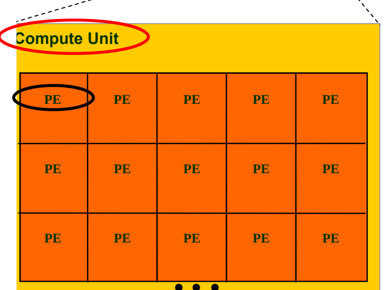
Oregon State University
Computer Graphics

Compute Units and Processing Elements are Arranged in Grids



A GPU Platform can have one or more **Devices**.

A 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 new V100 GPU has 84 Compute Units, each of which has 64 Processing Elements, for a grand total of 5,396 Processing Elements.

Thinking ahead to CUDA and OpenCL…

17

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 hoping to work on a small piece of the overall problem.

• OpenCL has 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 do.

• When a thread gets blocked somehow (a memory access, waiting for information from another thread, etc.), the processor switches to executing another thread to work on.

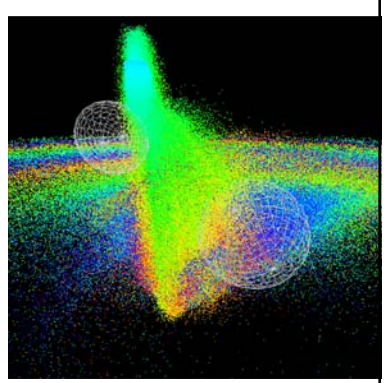
So, the Trick is to Break your Problem into Many, Many Small Pieces

18

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

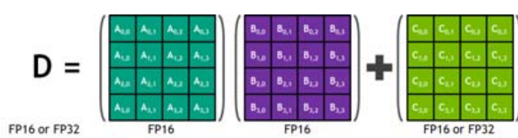
Something New – Tensor Cores
19

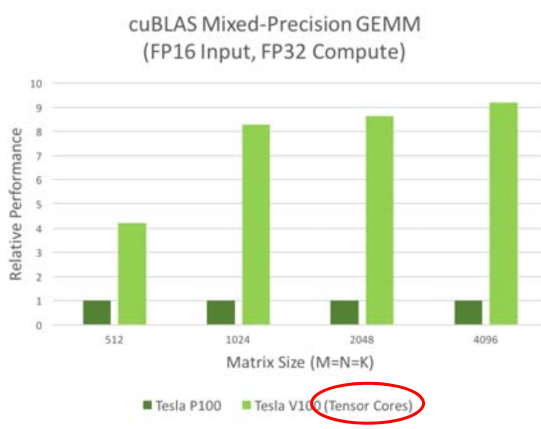
NVIDIA

Oregon State University
Computer Graphics

mjb – March 24, 2020

Tensor Cores Accelerate Fused-Multiply-Add Arithmetic
20





Oregon State University
Computer Graphics

mjb – March 24, 2020

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)

There are Two Approaches to Combining CPU and GPU Programs

1. Combine both the CPU and GPU code in the same file. The CPU compiler compiles its part of that file. The GPU compiles just its part of that file.

2. Have two separate programs: a .cpp and a .somethingelse that get compiled separately.

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. The line “if(a > b)” creates a vector of Boolean values giving the results of the if-statement for each thread. This becomes a “mask”.
2. Then, the GPU executes all parts of the divergence:
   Do This;
   Do That;
3. During that execution, anytime a value wants to be stored, the mask is consulted and the storage only happens if that thread’s location in the mask is the right value.

Oregon State University
Computer Graphics

- GPUs were originally designed for the streaming-ness of computer graphics
- Now, GPUs are also used for the streaming-ness of data-parallel computing
- GPUs are better for some things. CPUs are better for others.

Oregon State University
Computer Graphics

Bonus -- Looking at a GPU Spec Sheet

25

GPU	Kepler GK180	Maxwell GM200	Pascal GP100	Volta GV100
Compute Capability	3.5	5.2	6.0	7.0
Threads / Warp	32	32	32	32
Max Warps / SM	64	64	64	64
Max Threads / SM	2048	2048	2048	2048
Max Thread Blocks / SM	16	32	32	32
Max 32-bit Registers / SM	65536	65536	65536	65536
Max Registers / Block	65536	32768	65536	65536
Max Registers / Thread	255	255	255	255[1]
Max Thread Block Size	1024	1024	1024	1024
FP32 Cores / SM	192	128	64	64
Ratio of SM Registers to FP32 Cores	341	512	1024	1024
Shared Memory Size / SM	16 KB/32 KB/ 48 KB	96 KB	64 KB	Configurable up to 96 KB

Oregon State University
Computer Graphics

mjb – March 24, 2020

Bonus -- Looking at a GPU Spec Sheet

26

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOPS[1]	5	6.8	10.6	15.7
Peak FP64 TFLOPS[1]	1.7	.21	5.3	7.8
Peak Tensor TFLOPS[1]	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB	20480 KB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm²	601 mm²	610 mm²	815 mm²
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

Oregon State University
Computer Graphics

mjb – March 24, 2020