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

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.
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

What is a “Core” in the GPU Sense?

A Mechanical Equivalent...
Computer Graphics

How Many Robots Do You See Here?

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

A Spec Sheet Example

<table>
<thead>
<tr>
<th>Streaming Multiprocessors</th>
<th>CUDA Cores per SM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA</td>
<td></td>
</tr>
</tbody>
</table>

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**
  - General purpose programming
  - Multi-core under user control
  - Irregular data structures
  - Irregular flow control

- **GPU**
  - Data parallel programming
  - Little user control
  - Regular data structures
  - 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**.

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 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.

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.

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!
What is Fused Multiply-Add?
Many scientific and engineering computations take the form:
\[ D = A + (B \cdot C); \]
A “normal” multiply-add would likely handle this as:
\[ \text{tmp} = B \cdot C; \]
\[ D = A + \text{tmp}; \]
A “fused” multiply-add does it all at once, that is, when the low-order bits of \( B \cdot C \) are ready, they are immediately added into the low-order bits of \( A \) at the same time the higher-order bits of \( B \cdot C \) are being multiplied.

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

<table>
<thead>
<tr>
<th>789</th>
<th>( \times )</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>738</td>
<td></td>
<td></td>
</tr>
<tr>
<td>615</td>
<td></td>
<td></td>
</tr>
<tr>
<td>492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \rightarrow )</td>
<td>789</td>
<td></td>
</tr>
<tr>
<td>( \text{Can start adding the 9 the moment the 8 is produced!} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: “Normal” \( A + (B \cdot C) \neq \text{“FMA” } A + (B \cdot C) \)

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

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.

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

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

Removing the covers:

Removing the heat sink:

This transfers heat from the GPU Chip to the cooling fins.

Removing the fan assembly reveals the board:

This one contains 7.2 billion transistors! (Thank you, Moore’s Law)
Dismantling a Graphics Card

Underside of the board:

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

Bonus — Looking at a GPU Spec Sheet

<table>
<thead>
<tr>
<th>GPU</th>
<th>Kepler GK110</th>
<th>Maxwell SM200</th>
<th>Pascal GP100</th>
<th>Volta GV100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Capability</td>
<td>3.5</td>
<td>5.2</td>
<td>6.0</td>
<td>7.0</td>
</tr>
<tr>
<td>Threads/Streaming</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Max Warps/SM</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Max Threads/SM</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>Max Thread Blocks/SM</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Max 32-bit Registers/SM</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>Max Registers/Block</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>Max Registers/Thread</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Max Thread Block Size</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>FPS Core(s)/SM</td>
<td>182</td>
<td>128</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Ratio of SM Registers to FPS Core(s)</td>
<td>341</td>
<td>512</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Shared Memory Size/SM</td>
<td>18 KB/256 KB</td>
<td>48 KB</td>
<td>48 KB</td>
<td>Configurable up to 64 KB</td>
</tr>
</tbody>
</table>

TITAN RTX

TITAN RTX POWERs AI, MACHINE LEARNING, AND CREATIVE WORKFLOWS
NVIDIA TITAN RTX™ brings the power of GPU-accelerated artificial intelligence and machine learning directly to the desktop.