GPU Computing

Sam Sartor

March 9, 2017

Mines Linux Users Group

The GPU

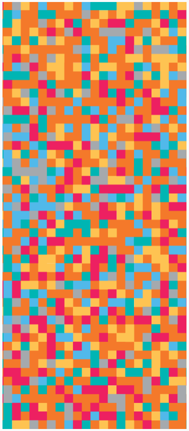
What is a GPU?

A Graphics Processing Unit (GPU) is a specialized chip primarily for accelerating graphical calculations.

GPUs generally derive their performance from their ability to do large numbers of identical arithmetic calculations in parallel.

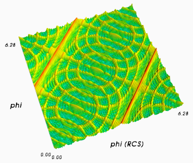
GPUs for Graphics

Screens have lot of pixels that need to be calculated very quickly. All of the required calculations are identical, just with different input numbers. And because pixels are independent the calculations are also trivial to parallelize. As a result, using the unnecessarily clever CPU would be wasteful and slow. A separate pixel-optimized chip can be used instead, leaving the CPU to do the important stuff.



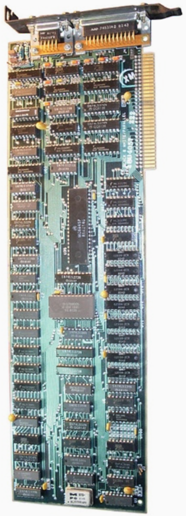
GPU Computing

Coloring pixels is not the only problem that involves a large number of similar, repetitive calculations. General-purpose GPUs can be used for countless other problems including machine learning, computer vision, signal processing, statistics, linear algebra, finance, and cryptography.



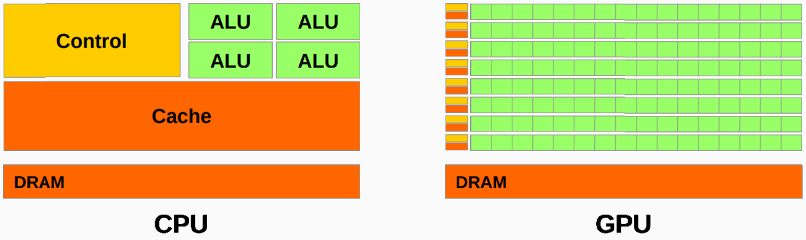
History

1970s - Highly specialized, used only for buffering video and drawing simple 2D rasters (sprites)
1980s - Common bitmap operations such as filling simple 2D shapes
1990s - 3D triangular graphics, common interfaces (OpenGL, Direct3D) developed
2000s - General purpose GPUs, capable of executing arbitrary instructions
2010s - Highly general, used as much for supercomputing as for graphics



How do GPUs Work?

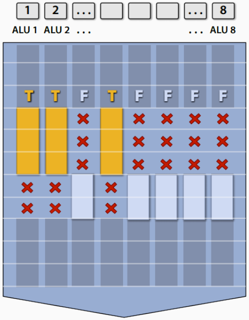
Architecture



GPUs excel at repetition. Instead of performing the same calculation many times in sequence, they step though sequences of instructions all at once using several cores. Each core does the same operation at the same time, but with different inputs.

Branching

Unlike CPUs, which jump back and forth through a program as conditions are met, a GPU will run every possible instruction in sequence, turning different cores on and off as branching occurs. In effect, GPUs are useful for parallel computations but not for multitasking.

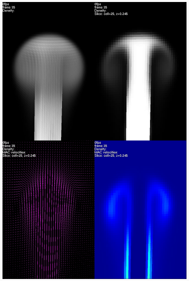


Computing At Home

OpenGL Shaders

Although shaders are used for pixel stuff, they are still fundamentally general purpose. Use vertex attributes, uniforms, and textures as input. Use the framebuffer for output.

OpenGL bindings exist for every language under the sun.



OpenGL Shaders - Pros & Cons

Pros

- Shaders have been around since like 2004
- Universally supported
- OpenGL allows for minimal setup

Cons

- Low level
- Not very general
- All data has to be stored in textures/images

CUDA

CUDA is a computing platform and API that provides truly general GPU computing. C/C++/Fortran code can be compiled ahead of time or at runtime and sent to the GPU along with arbitrary chunks of memory.

Libraries for controlling and communicating with CUDA programs exist for many languages including C/C++ (through the CUDA SDK) and Python (PyCUDA library).

CUDA - Pros & Cons

Pros

- Get to use real C/C++
- Pointers, recursion, etc.
- Copy arbitrary data between CPU and GPU
- Fast

Cons

- Only available on high-end Nvidia cards
- Low level
- Annoying to setup

OpenCL

OpenCL is a cross platform alternative to CUDA. It is similar in structure to OpenGL, but intended for general-purpose computation (not just 3D graphics).

Bindings exist for all languages. I even found a Brainfuck API.

OpenCL - Pros & Cons

Pros

- Cross platform
- Nice API
- Will use CPU instead of GPU if needed (works anywhere)

Cons

- Must use C-like OpenCL language
- No recursion, pointers, etc.
- Slightly slower than CUDA

ArrayFire

ArrayFire is an easy-to-use library of high-level functions with built-in implementations for CUDA, OpenCL, and the CPU. It is useful for linear algebra, statistics, trigonometry, signal processing, image processing, and more.

ArrayFire has first-party support for C++, Python, Go, Rust, Ruby, Lisp, Java, Fortran, D, R, C#, JavaScript, and Lua.

ArrayFire - Pros & Cons

Pros

- Trivial to use
- Cross platform
- Just pass arrays to functions

Cons

- Limited library of functions
- No way of defining your own

Torch

Torch is a popular Lua library for machine learning that seems to be used a lot. It has CPU, CUDA, and OpenCL backends available.

Torch - Pros & Cons

Pros

- Large community
- High level API
- Fast

Cons

- Lua

TensorFlow

TensorFlow is Google's library for moving big lists of numbers around, generally with machine learning in mind. As a result, Torch and Tensorflow are currently at war. It has a CPU implementation and a CUDA-based GPU implementation.

TensorFlow is primarily for Python, with C++ behind the scenes.

TensorFlow - Pros & Cons

Pros

- Python
- Good visualization tools
- Cool abstraction
- Best library for Recurrent Neural Networks

Cons

- Slightly slower than Torch (for now)
- Tricky to set up (CUDA)
- Needs a high-end Nvidia card to use the GPU

Copyright Notice

This presentation was from the **Mines Linux Users Group**. A mostly-complete archive of our presentations can be found online at `https://lug.mines.edu`.

Individual authors may have certain copyright or licensing restrictions on their presentations. Please be certain to contact the original author to obtain permission to reuse or distribute these slides.

Colorado School of Mines
Linux Users Group