Online materials

- In the next 2 lectures, we will also be using online materials hosted here:

  https://training.sharcnet.ca/courses/course/view.php?id=171

  (You need to use your Alliance credentials to gain an access to the materials.)

INTRODUCTION TO GPUS

The appeal of GPGPU

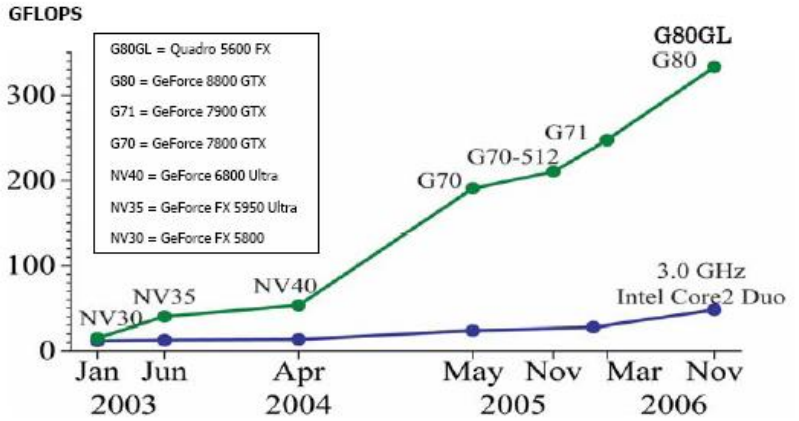
- GPGPU = General Purpose Graphical Processing Units

- "Supercomputing for the masses"
  - significant computational horsepower at an attractive price point
  - readily accessible hardware

- Scalability
  - programs can execute without modification on a run-of-the-mill PC with a $150 graphics card or a dedicated multi-card supercomputer worth thousands of dollars

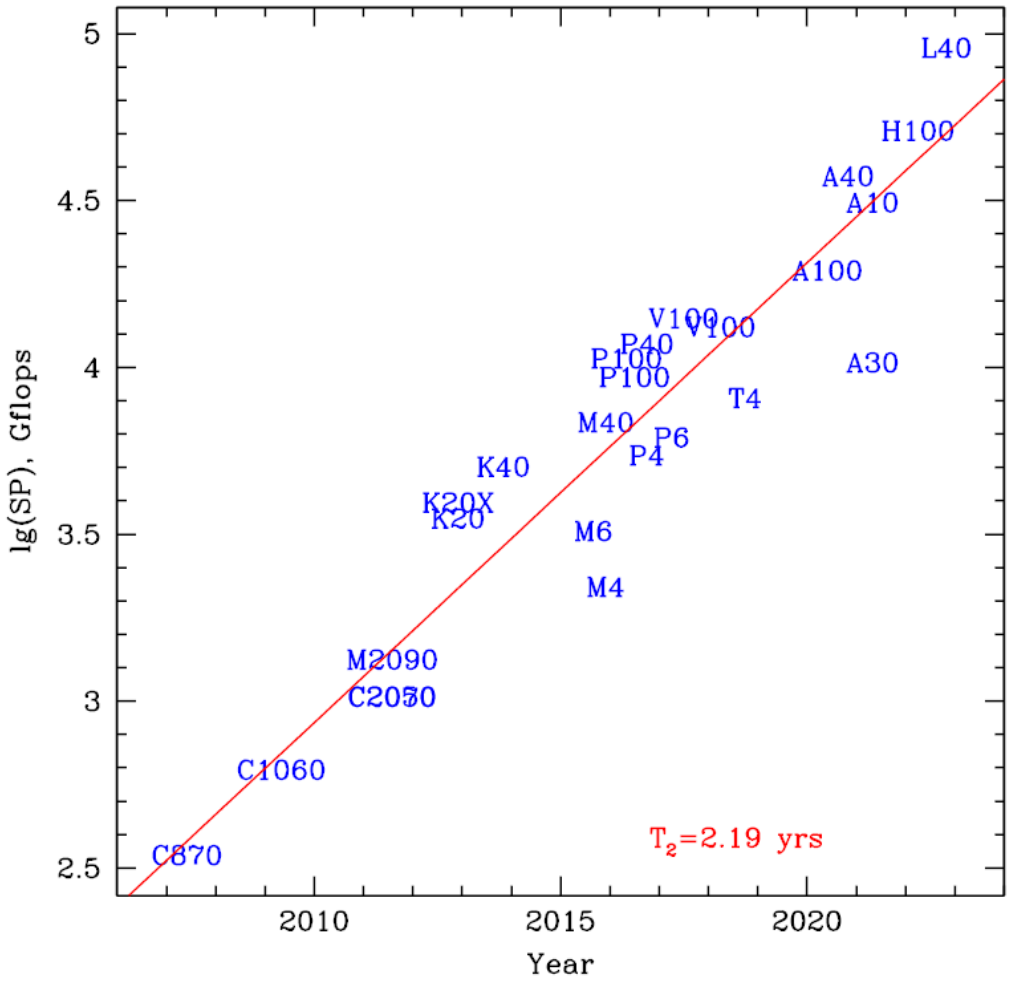
- Bright future – the computational capability of GPUs doubles each year
  - more thread processors, faster clocks, faster DRAM, …
  - "GPUs are getting faster, faster"

GPU computing timeline (early history)



- Before 2003 - Calculations on GPU, using graphics API
- 2003 - Brook “C with streams”
- 2005 - Steady increase in CPU clock speed comes to a halt, switch to multicore chips to compensate. At the same time, computational power of GPUs increases
- November, 2006 - CUDA released by NVIDIA
- November, 2006 - CTM (Close to Metal) from ATI
- December 2007 - Succeeded by AMD Stream SDK
- December, 2008 - Technical specification for OpenCL1.0 released
- April, 2009 - First OpenCL 1.0 GPU drivers released by NVIDIA
- August, 2009 - Mac OS X 10.6 Snow Leopard released, with OpenCL 1.0 included
- September 2009 - Public release of OpenCL by NVIDIA
- December 2009 - AMD release of ATI Stream SDK 2.0 with OpenCL support
- March 2010 - CUDA 3.0 released, incorporating OpenCL
- May 2011 - CUDA 4.0 released, better multi-GPU support
- Mid-2012 - CUDA 5.0
- Late-2012 - NVIDIA K20 Kepler cards

Historical performance for Tesla GPUs



How to get running on the GPU?

- Easiest case: the package you are using already has a GPU-accelerated version. No programming needed.
- Medium case: your program spends most of its time in library routines which have GPU accelerated versions. Use libraries that take advantage of GPU acceleration. Small programming effort required.
- Hard case: You cannot take advantage of the easier two possibilities, so you must convert some of your code to an explicit GPU code – OpenACC, CUDA or OpenCL.

GPU-enabled software

- A growing number of popular scientific software packages have now been accelerated for the GPU
- Using a GPU accelerated package requires no programming effort for the user
- Acceleration of Molecular Dynamics and AI software has been particularly successful, with all major packages offering the GPU acceleration option
- Many of these software packages use a combination of GPUs and CPUs in parallel. The resources used must be carefully chosen so that the program is running efficiently.
- Performance may depend on problem size.

GPU applications

- The GPU can be utilized in different capacities
- One is to use the GPU as a massively parallel coprocessor for number crunching applications
  - upload data and kernel to GPU
  - execute kernel
  - download results
  - CPU and GPU can execute asynchronously
- Some applications use the GPU for both data crunching and visualization
  - CUDA has bindings for OpenGL and Direct3D

Output of device diagnostic program - graham

```
$ module load cuda
$ nvcc device_diagnostic.cu
$ ./a.out
found 1 CUDA devices
  --- General Information for device 0 ---
Name:  Tesla P100-PCIE-12GB
Compute capability:  6.0
Clock rate:  1328500
Device copy overlap:  Enabled
Kernel execution timeout :  Disabled
  --- Memory Information for device 0 ---
Total global mem:  12786073600
Total constant Mem:  65536
Max mem pitch:  2147483647
Texture Alignment:  512
  --- MP Information for device 0 ---
Multiprocessor count:  56
Shared mem per mp:  49152
Registers per mp:  65536
Threads in warp:  32
Max threads per block:  1024
Max thread dimensions:  (1024, 1024, 64)
Max grid dimensions:  (2147483647, 65535, 65535)
```

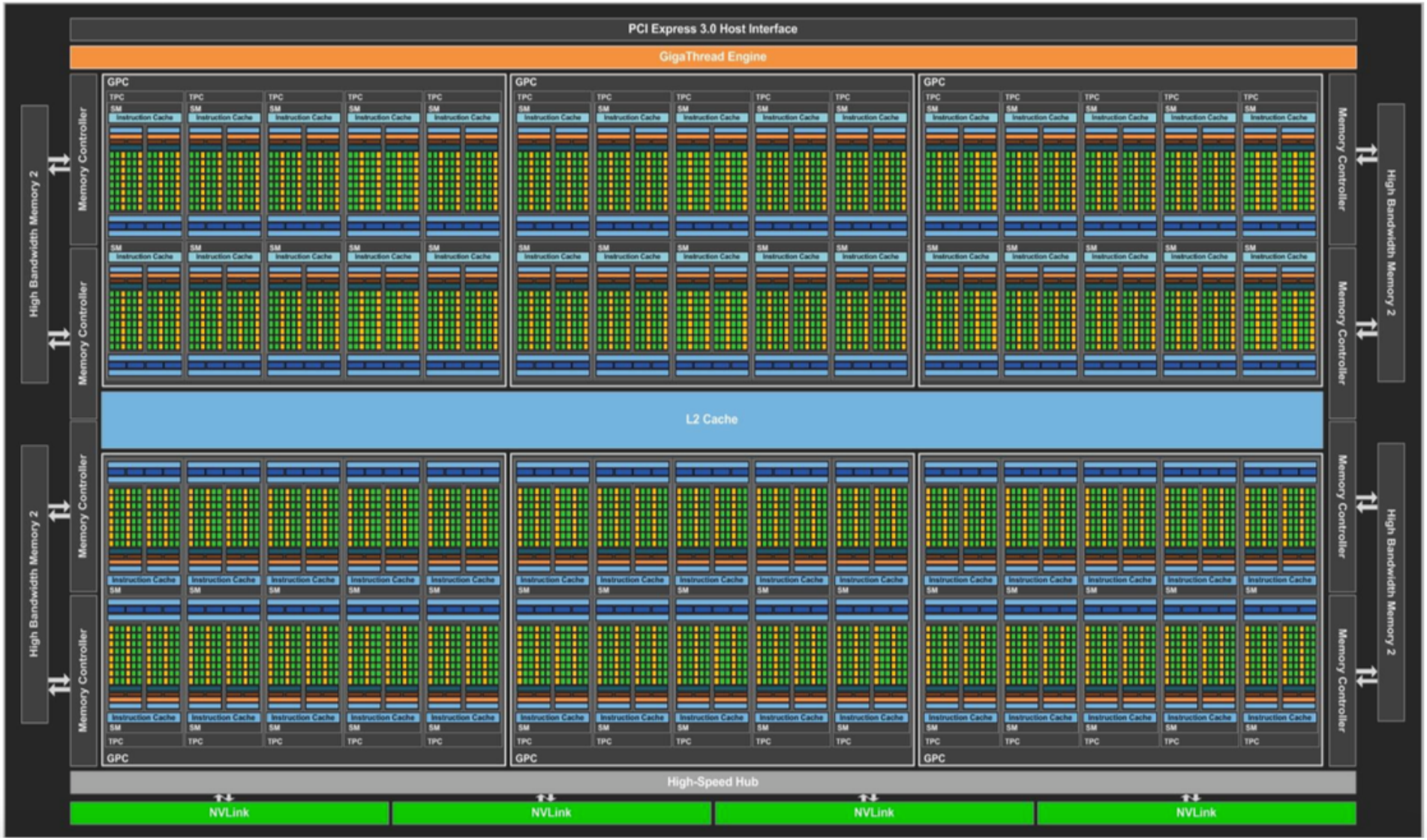
Which GPU to use?

Common now to have different generation GPUs within a cluster. Graham has Pascal P100, Volta V100, Turing T4 GPUs

To submit a job to gpu queue on graham

```
#!/bin/bash
…
#SBATCH --gres=gpu:1  # Use P100 (default)
… or …
#SBATCH --gres=gpu:v100:1  # Use V100
… or …
#SBATCH --gres=gpu:t4:1  # Use T4
```

GPU Hardware architecture - NVIDIA Pascal



GPU Hardware architecture - NVIDIA Pascal

Hardware basics

- The compute device is composed of a number of ***multiprocessors***, each of which contains a number of SIMD processors
  - Tesla P100 has 56 multiprocessors (each with 64 SP CUDA cores and 32 DP CUDA cores)

- A multiprocessor can execute K ***threads*** in parallel physically, where K is called the ***warp size***
  - ***thread*** = instance of kernel
  - warp size on current hardware is 32 threads

- Each multiprocessor contains a large number of 32-bit ***registers*** which are divided among the active threads