University of Leeds

Lecture 14: Introduction to GPGPU p

Previous lectures

So far we have looked at CPU programming.

- are
- on
- Many **common** parallelism issues (sc synchronisation, binary tree reduction)

- Also some **unique** to each type (locks a shared memory; explicit communication for distributed memory).

Today's lecture

Today's lecture is the first of 6 on programming **GPUs** (Graphics Proc

- eneral
- GPU **devices** contain multiple **SIMD** units.
- Different **memory types**, some 'shar interpreted as 'distributed.'

- Programmable using a variety of C/C++-based languages, notably **OpenCL** and **CUDA**.

Development of GPUs[1]

Early **accelerators** were driven by graphical operating systems and high-end applications (defense, science and engineering *etc.*).

-
-

Consumer applications employing 3D domin .

- First person shooters in mid-90s (Doom, Q
- 3D graphics accelerators by Nvidia, ATI T
- Initially as external **graphics cards**.

[1] Sanders and Kandrot, *CUDA By Example* (Addison-Wesley, 2011).

Programmable GPUs

The first **programmable** graphics cards were Nvidia's GeForce3 series (2001).

- **e** vertex
-

Early **general purpose** applications 'dis

graphical.

1. Input data converted to pixel **colours**.
2. Pixel shaders performed calculations on this data.
3. Final 'colours' converted back to **numerical data**.

GPGPUs

In 2006 Nvidia released its first GPU with CUDA.

- General calculations *without* converting to/from colours.

Now h

-
-
- Vendors include Nvidia, AMD and Intel.

Originally designed for **data parallel** g

- Increasing use of GPUs for *e.g.* **machine learning**[1] and **cryptocurrencies**.

[1] Now also have **neural processing units** (NPUs) for machine learning.

Overview of GPU architectures

Design and terminology of GPU hardware differs between vendors.

- Nvidia *different to* AMD *different to* Intel *different to* ...

Typi

SIMD processors contain **SIMD functio** :

- Each SIMD core contains multiple
- Executes **the same** instruction on m

Hierarchy:

Threads $\in$ SIMD Cores $\in$ SIMD Processors $\in$ GPU

SIMD processor

A typical SIMD processor has:

- A thread **scheduler**.
- Multiple **SIMD function**
-

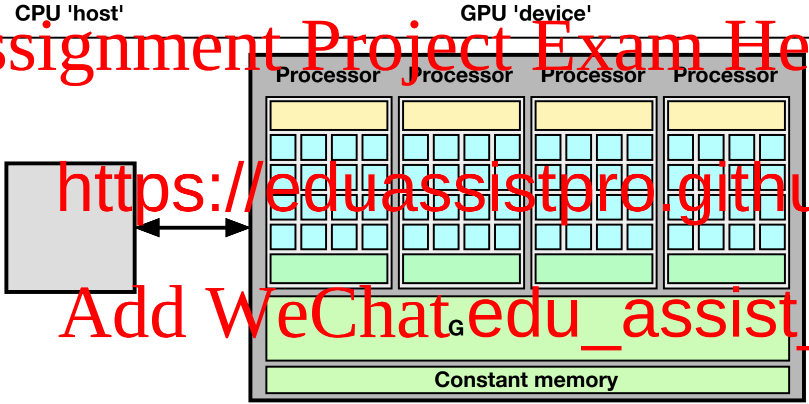
Not shown but usually present:

- Registers, special floating point units, ...

Note:

Thread scheduling is performed **in hardware**.

CPU with a single GPU



- The **data bus** between CPU and GPU is **very slow**.
- Faster for **integrated GPUs**.

SIMD *versus* SIMT

Nvidia refer to their architectures as **SIMT** rather than SIMD.

- <u>S</u>ingle <u>I</u>nstruction <u>M</u>ultiple **<u>T</u>hreads**.
-
- **simultaneously**.
- Therefore 'in between' SIMD and MIMD.

Will look at this more closely in Lecture 17, where we will see how it can be detrimental to performance.

Books

McCool *et al.* *[Lecture 1]* includes some OpenCL, but does not address GPUs specifically. Books for GPU programming include:

- *li,*

- **CUDA by example**, *Sanders and K* ey, 2011).

  - Slightly old, but a gentle introduction.
  - Only considers CUDA, whereas we will use OpenCL, but may still be useful.

You do not need any of these books for this module!

GPU programming languages 1. CUDA

The first language for GPGPU programming was Nvidia's **CUDA**.

-
-
-
- **Only works on CUDA-enabled device** .

As the first GPGPU language it has much docume

Therefore we will reference CUDA concepts and terminology quite frequently, often in footnotes.

GPU programming languages 2. OpenCL

Currently the main alternative to CUDA is **OpenCL** (2008).

- Stands for **Open Computing Language**.
- rs

-
- Can also run on CPUs, FPGAs (=F e Arrays), . . .

- C/C++ based.
- Similar programming model to CUDA.
- OpenCL 3.0 released Sept. 2020.

Directive based programming abstractions

OpenACC (2011):

- Open ACCelerator, originally intended for **accelerators**.
-
-

OpenMP:

- GPU support from version 4.0 onwards,
- Usual #pragma omp directives, wit

Both give **portable** code, but both require some understanding of the hierarchical nature of GPU hardware to produce reasonable performance.

Installing OpenCL

Already installed on cloud-hpc1.leeds.ac.uk (and most Macs).

Othe
archi

Nvidia: https://developer.nvidia.com

Intel: https://software.intel.com/en-

AMD: https://www.amd.com/en and search for OpenCL.

OpenCL header file

All OpenCL programs need to include a header file

Sinc
UNI
have t


```
#ifde _
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif
```


Note that the coursework will be marked on a system similar to `cloud-hpc1.leeds.ac.uk`, so it **must** run on that system.

Compiling and running

We use the CUDA nvcc compiler on cloud-hpc1.leeds.ac.uk:

```
nvcc -lOpenCL -o <executable> <source>.c
```

Note t

Exe

To execute on a GPU it will be necessary to use the batc (see next slide). However, it is also possible to run an O code on the login node's CPU by launching as any nor executable:


```
./<executable> [any command line arguments]
```

Running on GPU via batch jobs

The batch node of cloud-hpc1.leeds.ac.uk may be configured with a Tesla T4 GPU.

Henc

follo

-
- Create a job submission script as outlined below;
- Submit ot the batch queue using

Here is a typcal batch script to run “g

```
#!/bin/bash
#SBATCH --partition=gpu --gres=gpu:t4:1
./gpu-example
```

Compiling and running: Macs

Compiling:

Use the OpenCL framework:

```
gcc -Wal
```

-
- If you see deprecation **errors**, try
  gcc.

Executing:

Launch as any normal executable

```
./<executable> [any command line arguments]
```

Platforms, devices and contexts

Since OpenCL runs on many different devices by many different vendors, it can be quite laborious to initialise.

Need

Pla	) and
Device	Belongs to a platform; may b

Need to initialise:

Context	Coordinates inter **device** (*e.g.* a GPU). One per device.
Command queue	To request action by a device. Normally one per device, but can have more *[Lecture 19]*.

Initialisation code

Most code for this module will come with `helper.h` which contains two useful routines:

`simp`

-

`compileKernelFromFile()`

- Compiles an OpenCL **kernel** to be e
  Will cover this next lecture.

You don't need to understand how these routines work, but are welcome to take a look.

Using simpleOpenContext_GPU()

```
#include "helper.h"    // Also includes OpenCL.
int main() {
  // Get context and device for a GPU.
  cl_device_id device;
  cl

  // Op
  cl
  cl_command_queue queue = clCreateCommandQueue(
    context,device,0,&status);

  ...  // Use the GPU through 'queue'.

  // At end of program.
  clReleaseCommandQueue(queue);
  clReleaseContext(context);
}
```

'Hello world' in OpenCL

Code on Minerva: `displayDevices.c`

Since most GPU's cannot print in the normal sense, there is no simple 'Hello World' program.

Inste se
help

- Loops through **all** platforms and dev
- Lists all **OpenCL-compatible** de
- Also a list of extensions; *e.g.* `cl_ _` supports double precision floating point arithmetic.
- In the output, a **compute unit** is a SIMD processor or streaming multiprocessor.

Summary and next lecture

Today we have started looking at GPU programming:

- Overview of GPU architectures.
-

  using the functions:
  - `clGetPlatformIDs`
  - `clGetDeviceIDs`

Next time we will implement a “real” program in OpenCL: **vector addition**.