## GPUs

And there are many others. For example AMD have their *Radeon Open Compute platform* (ROCm) infrastructure

## GPUs

And there are many others. For example AMD have their *Radeon Open Compute platform* (ROCm) infrastructure

They have a language *Heterogeneous-Compute Interface for Portability* (HIP) that is very similar to CUDA and runs on AMD and NVIDIA hardware

## GPUs

And there are many others. For example AMD have their *Radeon Open Compute platform* (ROCm) infrastructure

They have a language *Heterogeneous-Compute Interface for Portability* (HIP) that is very similar to CUDA and runs on AMD and NVIDIA hardware

In fact, they have a CUDA to HIP translator to aid porting code

## GPUs

And Intel have *OneAPI*, with *Data Parallel C++*, (DPC++) also intended to be multi-architectural

## GPUs

And Intel have *OneAPI*, with *Data Parallel C++*, (DPC++) also intended to be multi-architectural

They also have CUDA code migration tools

## GPUs

And Intel have *OneAPI*, with *Data Parallel C++*, (DPC++) also intended to be multi-architectural

They also have CUDA code migration tools

But it is clear each of CUDA, HIP and OneAPI are “best used” with the hardware of their respective developers

## GPUs

In development is *WebGPU*, a JavaScript API for graphics and compute, providing a uniform Web interface to whatever is running underneath

## GPUs

In development is *WebGPU*, a JavaScript API for graphics and compute, providing a uniform Web interface to whatever is running underneath

**Exercise** Read about these

## GPUs

Microsoft have their own versions of everything, of course

## GPUs

Microsoft have their own versions of everything, of course

Their *DirectCompute* is not a million miles from CUDA, but is based on their DirectX suite

## GPUs

Microsoft have their own versions of everything, of course

Their *DirectCompute* is not a million miles from CUDA, but is based on their DirectX suite

It runs on NVIDIA and AMD cards

## GPUs

Microsoft have their own versions of everything, of course

Their *DirectCompute* is not a million miles from CUDA, but is based on their DirectX suite

It runs on NVIDIA and AMD cards

But the portability to other operating systems is an open question

## GPUs

Microsoft have their own versions of everything, of course

Their *DirectCompute* is not a million miles from CUDA, but is based on their DirectX suite

It runs on NVIDIA and AMD cards

But the portability to other operating systems is an open question

They also have C++ Accelerated Massive Parallelism (C++ AMP), an annotated version of C++ that is reputedly much easier to write code for

## GPUs

Microsoft have their own versions of everything, of course

Their *DirectCompute* is not a million miles from CUDA, but is based on their DirectX suite

It runs on NVIDIA and AMD cards

But the portability to other operating systems is an open question

They also have C++ Accelerated Massive Parallelism (C++ AMP), an annotated version of C++ that is reputedly much easier to write code for

This is more like an OpenMP for GPUs

## GPUs

### OpenACC

In fact, there is also OpenACC, which is essentially OpenMP for GPUs

## GPUs
### OpenACC

In fact, there is also OpenACC, which is essentially OpenMP for GPUs

`pragma` annotations indicate code can be run on a GPU

## GPUs
### OpenACC

In fact, there is also OpenACC, which is essentially OpenMP for GPUs

`pragma` annotations indicate code can be run on a GPU

```
#pragma acc parallel loop
  for (int i = 0; i < n; ++i) {
    z[i] = x[i] + y[i];
}
```

## GPUs

### OpenACC

In fact, there is also OpenACC, which is essentially OpenMP for GPUs

`pragma` annotations indicate code can be run on a GPU

```
#pragma acc parallel loop
  for (int i = 0; i < n; ++i) {
    z[i] = x[i] + y[i];
}
```

runs the loop on the GPU. The programmer does not have to think about copying data back and forth or writing and calling kernels

## GPUs

### OpenACC

In fact, there is also OpenACC, which is essentially OpenMP for GPUs

`pragma` annotations indicate code can be run on a GPU

```
#pragma acc parallel loop
  for (int i = 0; i < n; ++i) {
    z[i] = x[i] + y[i];
}
```

runs the loop on the GPU. The programmer does not have to think about copying data back and forth or writing and calling kernels

**Exercise** Is that a good or a bad thing?

# GPUs

## OpenACC

Similar to other systems, simply ignoring the `pragma` and running on the CPU will produce equivalent results

## GPUs

### OpenACC

Similar to other systems, simply ignoring the `pragma` and running on the CPU will produce equivalent results

OpenACC does for accelerators (co-processors) what OpenMP does for multi-core

## GPUs
### OpenACC

Similar to other systems, simply ignoring the `pragma` and running on the CPU will produce equivalent results

OpenACC does for accelerators (co-processors) what OpenMP does for multi-core

OpenMP and OpenACC pragmas can sit side-by-side in the same code

## GPUs

### OpenACC

Similar to other systems, simply ignoring the `pragma` and running on the CPU will produce equivalent results

OpenACC does for accelerators (co-processors) what OpenMP does for multi-core

OpenMP and OpenACC pragmas can sit side-by-side in the same code

In fact, OpenACC is supposed to merge with OpenMP at some point, but progress seems slow

## GPUs

### OpenACC

Similar to other systems, simply ignoring the `pragma` and running on the CPU will produce equivalent results

OpenACC does for accelerators (co-processors) what OpenMP does for multi-core

OpenMP and OpenACC pragmas can sit side-by-side in the same code

In fact, OpenACC is supposed to merge with OpenMP at some point, but progress seems slow

A freely available version of OpenACC for NVIDIA GPUs is available and GCC also supports it (but only on Nvidia and AMD): this may help OpenACC to become more popular

## GPUs

GPUs have a great future ahead of them as they are excellent at certain kinds of problem, when programmed by really good programmers

## GPUs

GPUs have a great future ahead of them as they are excellent at certain kinds of problem, when programmed by really good programmers

There are CUDA bindings for Python and Java (of course), so you don’t have to use C

## GPUs

GPUs have a great future ahead of them as they are excellent at certain kinds of problem, when programmed by really good programmers

There are CUDA bindings for Python and Java (of course), so you don’t have to use C

Another item to note is that GPUs use (relatively) very little energy for the amount of processing they deliver

## GPUs

GPUs have a great future ahead of them as they are excellent at certain kinds of problem, when programmed by really good programmers

There are CUDA bindings for Python and Java (of course), so you don’t have to use C

Another item to note is that GPUs use (relatively) very little energy for the amount of processing they deliver

In a world where supercomputer centres spend more on electricity than they do on the computers themselves, the operations per watt that GPUs provide turns out to be very attractive

## GPUs

GPUs have a great future ahead of them as they are excellent at certain kinds of problem, when programmed by really good programmers

There are CUDA bindings for Python and Java (of course), so you don’t have to use C

Another item to note is that GPUs use (relatively) very little energy for the amount of processing they deliver

In a world where supercomputer centres spend more on electricity than they do on the computers themselves, the operations per watt that GPUs provide turns out to be very attractive

See the current Top 500

## GPUs

If we were starting from scratch, we probably wouldn’t design a GPU in the way it is

## GPUs

If we were starting from scratch, we probably wouldn’t design a GPU in the way it is

Just like the original CPU was based on existing integrated circuits that engineers noticed could be made programmable, the GPU is based on graphics co-processors that engineers noticed could be made programmable

## GPUs

If we were starting from scratch, we probably wouldn't design a GPU in the way it is

Just like the original CPU was based on existing integrated circuits that engineers noticed could be made programmable, the GPU is based on graphics co-processors that engineers noticed could be made programmable

So the accidents of history brought us to where we are today

## GPUs

As previously mentioned, we are currently seeing multicore processors merging with GPUs

## GPUs

As previously mentioned, we are currently seeing multicore processors merging with GPUs

This is repeating the historical precedents of coprocessors merging with main processors

## GPUs

As previously mentioned, we are currently seeing multicore processors merging with GPUs

This is repeating the historical precedents of coprocessors merging with main processors

One processor that has had a lot of attention recently is the Apple M1

## Coprocessors

### Apple M1

The Apple M1 is a 4CPU+4CPU+GPU+NPU+memory ARM architecture system on a chip (SoC), using 16 billion transistors

## Coprocessors

### Apple M1

The Apple M1 is a 4CPU+4CPU+GPU+NPU+memory ARM architecture system on a chip (SoC), using 16 billion transistors

- 4 fast CPU cores
- 4 energy efficient CPU cores
- 8 GPU cores (24,576 threads)
- up to 16GB memory in a *unified memory* architecture
- 16 core neural processing unit (NPU) (11 trillion ops/sec)
- a digital signal processor (DSP)
- an image processing unit (ISP)
- a video encoder/decoder

## Coprocessors

### Apple M1

The Apple M1 is a 4CPU+4CPU+GPU+NPU+memory ARM architecture system on a chip (SoC), using 16 billion transistors

- 4 fast CPU cores
- 4 energy efficient CPU cores
- 8 GPU cores (24,576 threads)
- up to 16GB memory in a *unified memory* architecture
- 16 core neural processing unit (NPU) (11 trillion ops/sec)
- a digital signal processor (DSP)
- an image processing unit (ISP)
- a video encoder/decoder

This takes coprocessing to new levels!

## Coprocessors

### Apple M1

The various units share memory in the unified memory architecture

## Coprocessors

### Apple M1

The various units share memory in the unified memory architecture

This is 16GB memory, on the same chip

## Coprocessors

### Apple M1

The various units share memory in the unified memory architecture

This is 16GB memory, on the same chip

Note that on-chip memory is fast(er), but not expandable

## Coprocessors

### Apple M1

The various units share memory in the unified memory architecture

This is 16GB memory, on the same chip

Note that on-chip memory is fast(er), but not expandable

**Advanced Exercise** Read about the memory consistency features used by the M1 to support compatability with x86 code

## Coprocessors

Incidentally, Intel has its *Gaussian and Neural Accelerator* (GNA) integrated into the main CPU chip

## Coprocessors

Incidentally, Intel has its *Gaussian and Neural Accelerator* (GNA) integrated into the main CPU chip

Initially for support of speech recognition, it could probably be used for more general deep learning

## ARM Mali

At the low-power end of the scale, ARM have their Mali core

## ARM Mali

At the low-power end of the scale, ARM have their Mali core

“Core” in the sense of a chunk of silicon design that can be incorporated into other system chips

## ARM Mali

At the low-power end of the scale, ARM have their Mali core

“Core” in the sense of a chunk of silicon design that can be incorporated into other system chips

With current generations having up to 32 processing cores, this is a GPU you will find in your phone

## ARM Mali

At the low-power end of the scale, ARM have their Mali core

“Core” in the sense of a chunk of silicon design that can be incorporated into other system chips

With current generations having up to 32 processing cores, this is a GPU you will find in your phone

It supports OpenCL and 64-bit floating point

## ARM Mali

At the low-power end of the scale, ARM have their Mali core

“Core” in the sense of a chunk of silicon design that can be incorporated into other system chips

With current generations having up to 32 processing cores, this is a GPU you will find in your phone

It supports OpenCL and 64-bit floating point

As well as doing some graphics. . .

## The End

End of Lectures

Future sessions will be problems classes, and going through past papers