SOFT354 - Parallel computation and distributed systems

Date: 29-09-16

Why parallel?

- Processor clock speeds aren't getting faster:
- But there's a lot more data around
- Science is also pushing the boundaries of what computers are required to do
- working with big data effectiently
- Advance simulations require lots of computing power
- As CPU clock speeds have not increased significantly in the past years but data has
  - parallelism is the solution

Parallelism

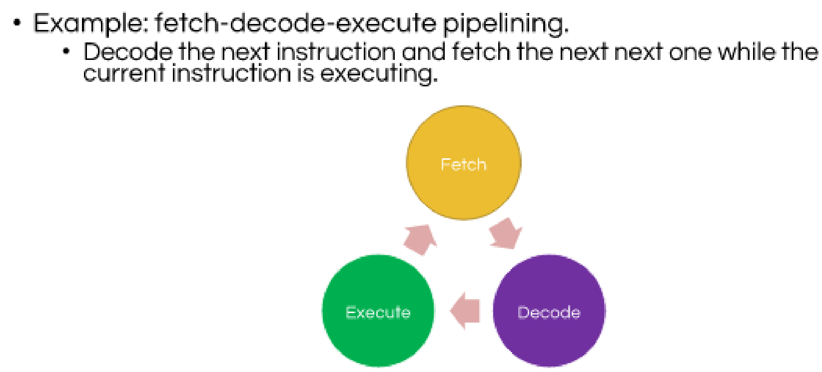
If you can't do something any faster, do more than one thing at the same time!

Levels of parallelism

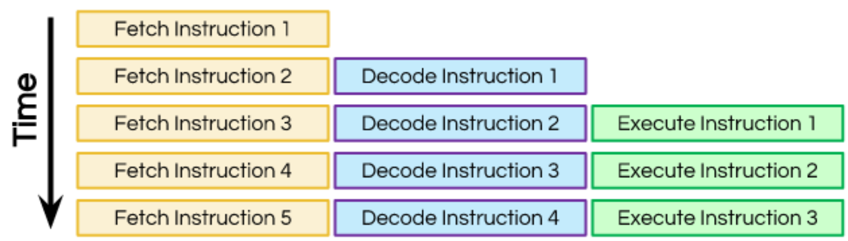
- Computers can be parallel at different scales:
  - Within a core (instruction-level parallelism)
  - By having **multiple cores** in a CPU
  - by having multiple CPUS

Instruction level parallelism

- The performance of a single processor core can be increased by soing same



thing in parallel *Example*



Multiple cores

- Morden CPUs have several largely independent processing cores
- All cores have access to the same memory, but can be running different programs or threads
- **GPUs** have loads of cores
- less flexible than CPU cores

Multiple processors

- Distribute a problem amongst multiple computers, connected via a network

What's a GPU

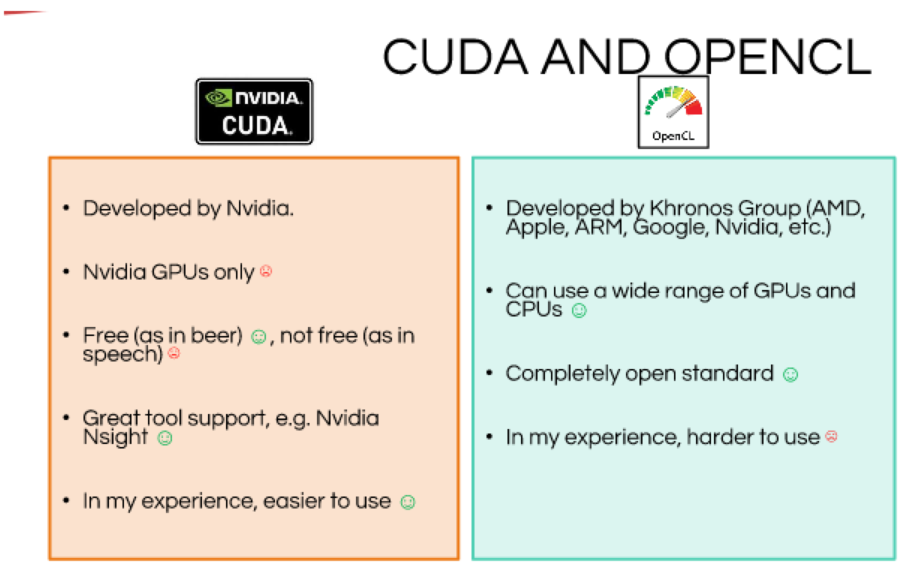
- A Graphical Processing Unit
  - Often used for computations that have **nothing to do with graphics**
- More accurate name: General purpose computing on graphical processing units

- **Key insight**: the transformation operation is the same for each vertex
- So why not design a processor that can perform the same operation on many different values at the same time
- All cores are run the same code- simpiler circuitry

General purpose GPUS

- Programmable cores
- Rather than performing a fixed funtion
  - you could write arbitrary code for them like you would a regular CPU

CUDA and OpenCL



Basic matrices and vector math

- To add two vectors, add their individual components:

$$\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1+3\\2+4\end{bmatrix} = \begin{bmatrix}4\\6\end{bmatrix}$$

- Same with matrices:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} + \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}1+5 & 2+6\\3+7 & 4+8\end{bmatrix} = \begin{bmatrix}6 & 8\\10 & 12\end{bmatrix}$$

- To multiply a vector or matrix by a number, multiply each component:

$$5\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}5\times 1 & 5\times 2\\5\times 3 & 5\times 4\end{bmatrix} = \begin{bmatrix}5 & 10\\15 & 20\end{bmatrix}$$

- To compute the dot product* of two vectors, multiply individual components and add them up:

$$\begin{bmatrix}1\\2\end{bmatrix} \cdot \begin{bmatrix}3\\4\end{bmatrix} = 1\times 3 + 2\times 4 = 3 + 8 = 11$$

CUDA compute capablity

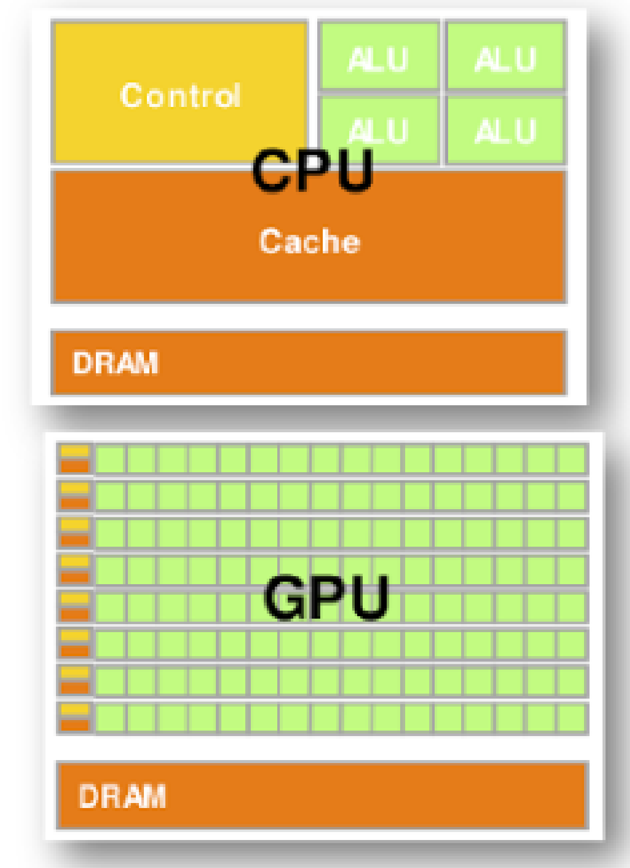
- The compute capability of a device describes its architecture, e.g.
  - Number of SMs and registers
  - Sizes of memories
  - Features & capabilities

Compute Capability	Selected Features (see CUDA C Programming Guide for complete list)	Tesla models
1.0	Fundamental CUDA support	870
1.3	Double precision, improved memory accesses, atomics	10-series
2.0	Caches, 3D grids, surfaces, ECC, P2P, concurrent kernels/copies, function pointers, recursion	20-series
3.0	Dynamic parallelism	K-series

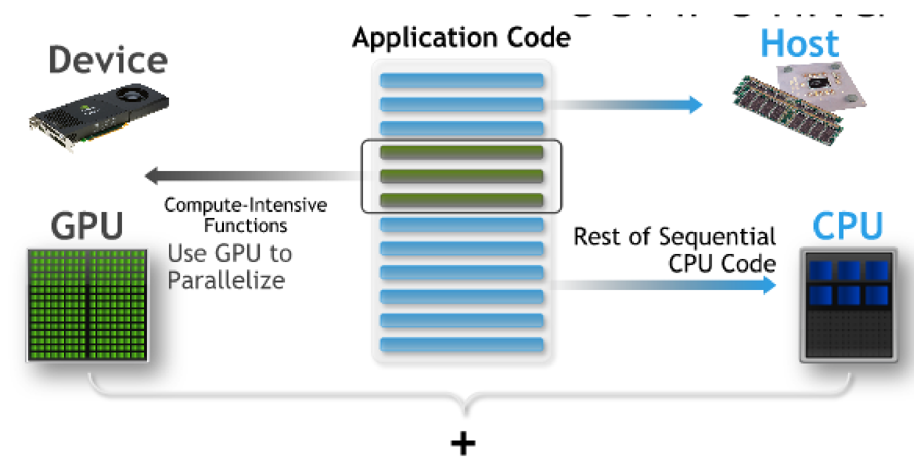
- We will concentrate on Fermi devices
  - Compute Capability >= 2.0

CUDA

- A GPU is organised differently to a CPU
- Each CPU core has its own control logic
- each thread that's running in parallel can be at a different place in the program
- CUDA cores are organised into streaming Multiprocessors (SMs)
  - *E.g.* in compute capability 2, there are 32 cores per SM
- **All cores within an SM execute the same instruction at the same time**



Heterogeneous computing CPU code alongside GPU code, working together



Simple processing flow

1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

Programing tipbits

- **global** keyword before function gets run on the GPU
- calling a device function ˆ mykernel<<1, 1>> how many functions you want to run on the GPU

To do

- C programming brush uo
- linear algebra