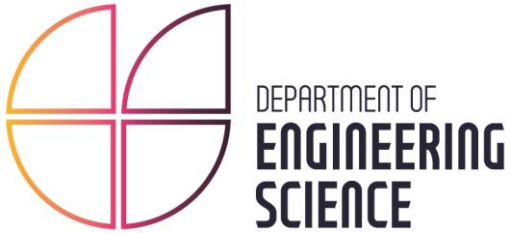


An Introduction to HPC and Scientific Computing

Wes Armour

Oxford e-Research Centre,
Department of Engineering Science

Aims and learning outcomes

The aims of this CWM are to introduce you to scientific computing and High Performance computing (HPC).

It's more important that you pick up the basics of computing and programming during the week, because these are the building blocks for everything else.

This CWM isn't designed to turn you into a world class HPC programmer, that that's years.

This CWM is designed to give you the skills to continue to learn in this area and for you to have the ability to write your own computer codes and tackle basic problems.

Assessment for this course will focus on the final two practical sessions in the latter half of the week. The aim of the assessment is for you to demonstrate that you've picked up the basics from this course.

The assessment will be light because I'm keen for you to focus on the content rather than worrying about the assessment.

In all I hope you will find this a fun and interesting week long introduction to HPC and Scientific Computing!

Locations and Timetable

Locations

Lectures will be in LR6
Practical sessions will be in the Linux Lab

Timetable

09:30 - 10:30 Morning lecture
10:30 - 11:00 break
11:00 - 12:30 Morning practical

12:30 - 13:30 lunch

13:30 - 14:30 Afternoon lecture
14:30 - 15:00 break
15:00 - 16:30 Afternoon practical

Lectures will be delivered by Wes Armour, Ian Bush, Karel Adamek.

Practical's supervised by Wes Armour, Ian Bush, Karel Adamek, Ania Brown and Jan Novotny.

On-line feedback form: http://bit.ly/OXUNICWM please, please, please do complete ☺

Lectures

Monday - Here we have three lectures to begin with and finish with a practical session, this is because we'll need to introduce you to several different topics before you can complete a meaningful practical.

Morning lecture:	Introduction to computer architectures.
Morning lecture:	Introduction to the C programming language.
Afternoon lecture:	Introduction to Linux, compilers and build systems.

Tuesday

Morning lecture:	Using repositories and good coding practices.
Afternoon lecture:	A deeper dive into C programming.

Wednesday afternoon

Afternoon lecture:	How to multi-task on CPUs using OpenMP.

Thursday

Morning lecture:	An introduction to the CUDA programming language.
Afternoon lecture:	Scientific Computing using the CUDA programming language part one.

Friday

Morning lecture:	Scientific Computing using the CUDA programming language part two.
Afternoon lecture: NVIDIA.	***Guest Lecture: Deep learning Demystified - Adam Grzywaczewski***

UNIVERSITY OF OXFORD

Oxford e-Research Centre

Practical Sessions

Monday - Here we have one practical in the afternoon.

Afternoon Practical:	Linux, compiling C code and using Make.

Tuesday

Morning Practical:	Practical examples of using repositories for your projects.
Afternoon Practical:	Practical examples using the C programming language.

Wednesday Afternoon

Afternoon Practical:	Practical examples of using OpenMP on CPUs.

Thursday

Morning Practical:	Practical examples of the CUDA programming language.
Afternoon Practical:	Advanced examples of CUDA programming part one.

Friday

Morning Practical:	Advanced examples of CUDA programming part two.
Afternoon Practical:	Finishing up.

A closer look at a CPU – Single processor

A single CPU (processor) is comprised of many cores (anything from one or two, all the way up to twenty or more).

All cores are connected by the L3 (or Last Level Cache - LLC) which is shared amongst them.

The size of the L3 cache varies from processor to processor, but typically on Intel's latest architecture, Skylake, this will be between 1 and 1.5 MB.

To extract the maximum performance from modern CPUs work must be shared amongst the cores. This is where OpenMP is helpful.

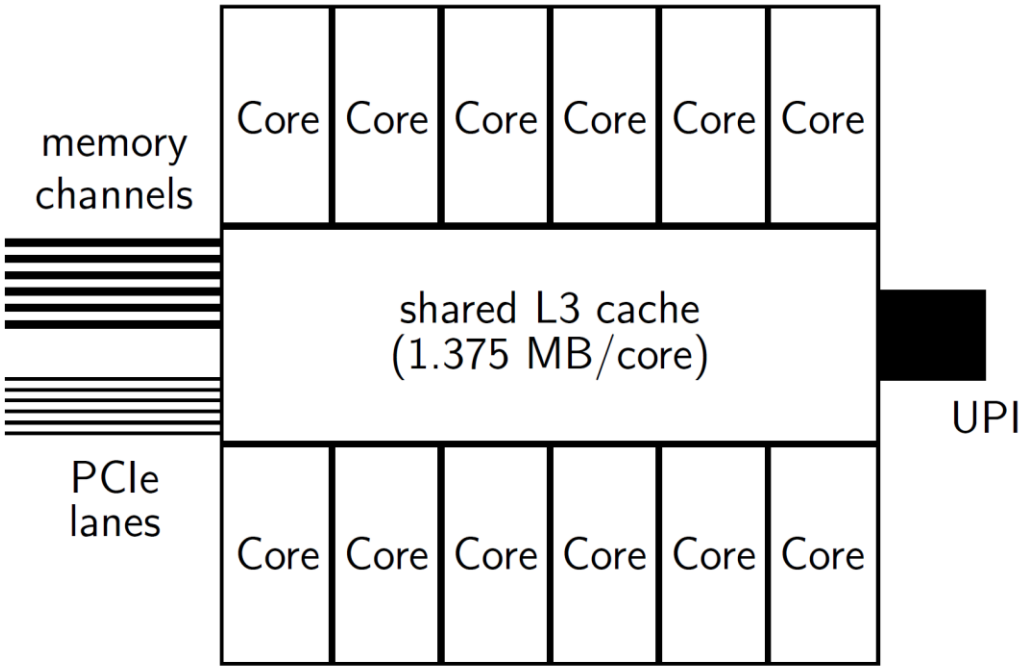


Image courtesy of Mike Giles

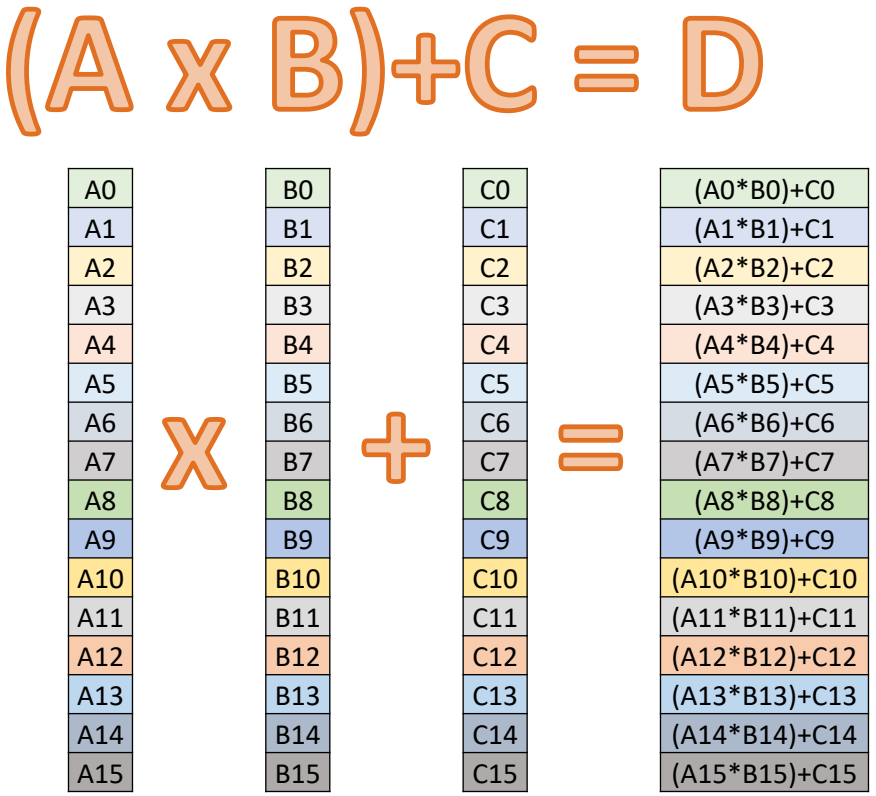
A closer look at a CPU – SIMD units

A core also has one (or two) *Advanced Vector eXtensions* (AVX) units.

These units are capable of executing a Single Instruction on Multiple Data (*SIMD*) elements at the same time (*in parallel*). A program that exploits these vector features is said to be *"vectorised"*

Skylake has AVX-512 vector units, meaning that the vectors are 512 bits long, so can store 16 single precision numbers in each. They can perform at most two instructions at once by issuing a *fused multiply add* (FMA) instruction (see right).

It's these units, along with the high core count that allow modern CPUs to perform very large and complex computations very quickly.



Memory Architecture

Modern computers can have several different types of memory all of which have different *bandwidths* (bandwidth is a measure of how quickly you can get a "unit", e.g. a GB of data from point A to point B).

They are arranged in a hierarchy. From lots of slower *RAM* down to very few, but very fast *registers* which are located closest to the CPU processing core.

The diagram on the right shows the different bandwidths for a modern Intel Skylake CPU. Note that all of the *caches* (smaller areas of fast memory) are located on the CPU.

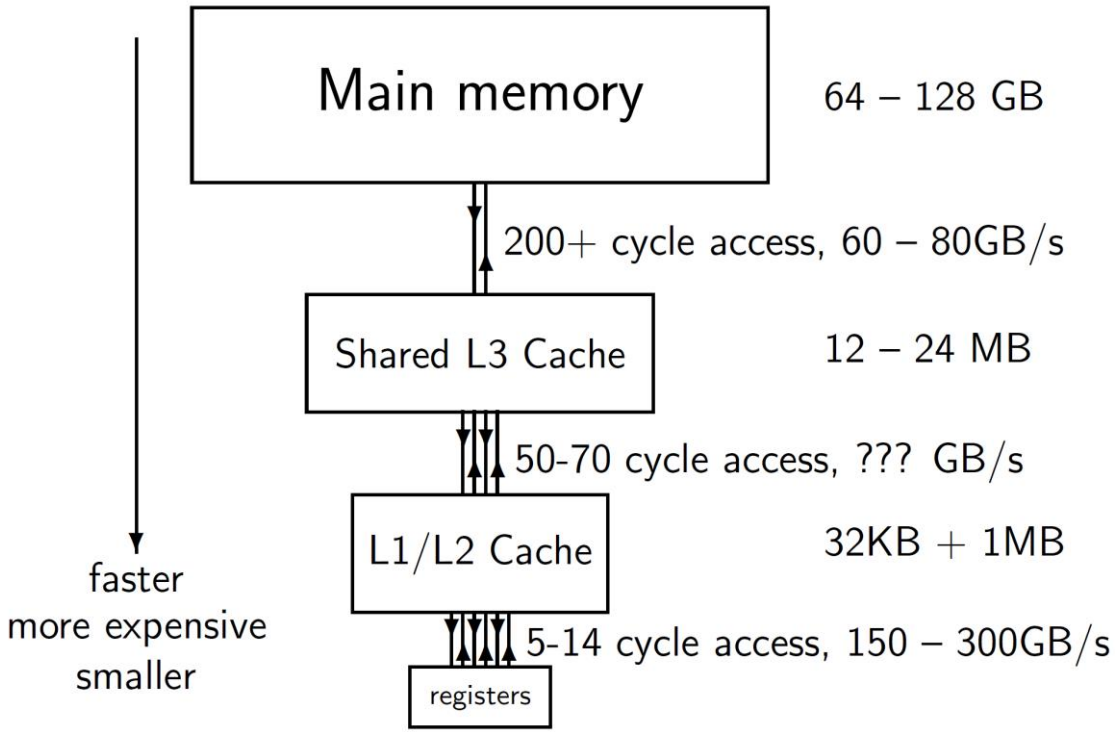


Image courtesy of Mike Giles

What's the point of a Cache?

As noted on our previous slide the area of memory where we can store the most data is the RAM, but this has lower bandwidth.

So if we need to move our data from RAM to processing cores many times our code would run slowly. This is where Cache helps. Caches exploit *data locality*.

- *temporal locality:* a data item just accessed is likely to be used again in the near future, so keep it in the cache.

- *spatial locality:* neighbouring data is also likely to be used soon, so load them into the cache at the same time using a 'wide' bus (like a multi-lane motorway).

It is these wide lanes that feed the AVX units with data.

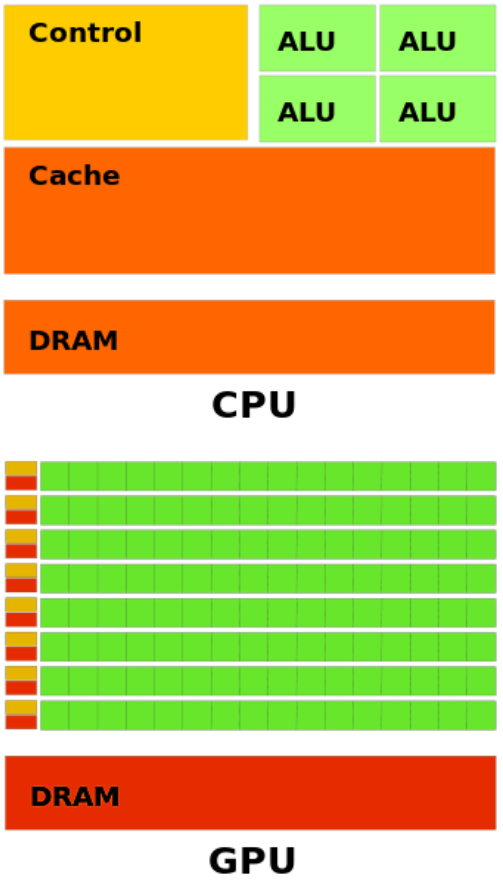
A closer look at a GPU - Design

GPUs have a different design to CPUs. Their complexity is greatly reduced (*no extensive data caching or command-and-control*). They dedicate most of there transistors to processing (right). This makes GPUs ideal for compute-intensive, highly parallel computation.

More specifically, GPUs excel at data-parallel computations. Just like the AVX (SIMD) units in CPUs.

GPUs also have access to a reasonable amount of *High Bandwidth Memory (HBM).* This can be as much as 32GB on flagship models. This memory has greater bandwidth than server RAM, but there is less of it.

However GPUs attach to a server via the PCIe bus, so this ultimately limits communication speed between a CPU and GPU.



C programming what did we learn?

In this lecture you have learnt about the basic building blocks of a C program.

You have learnt about standard libraries, expressions and statements.

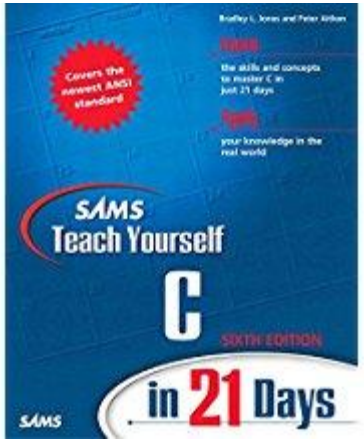
We have covered how data is stored on a computer and how it is represented in C.

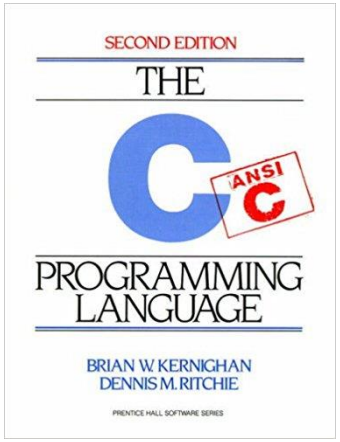
You have learnt about functions, operators, both logical and relational and program control.

Finally we covered the basics of input and output.

You should now be in a position to write your own C program.

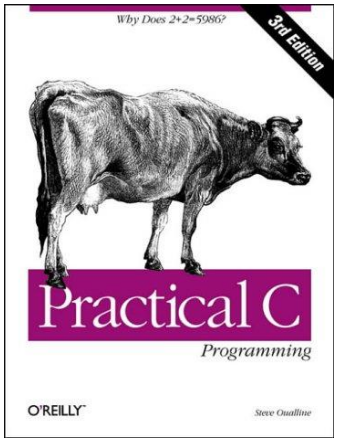
Further reading

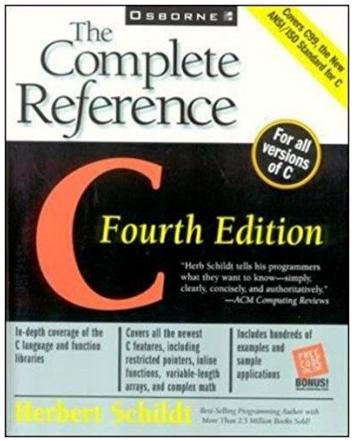




http://www.learn-c.org/

https://www.cprogramming.com/tutorial/c-tutorial.html





https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

UNIVERSITY OF OXFORD

Oxford e-Research Centre