XJCO3221 Parallel Computation

Peter Jimack and David Head

University of Leeds

Lecture 1: Introduction

Today

- Materials available for this module.
- Assessments (3×coursework plus exam), with deadlines.
- How historical trends in computer architectures have lead to the current ubiquity of parallel machines.
- The three classes of parallel architecture we will look at.
- Overview of all 20 lectures.

Minerva resources

Lecture slides

- Will be released on Minerva in the week prior to the lecture.
- Where relevant there will also video demonstrations of example code.

Worksheets

- **Formative**, *i.e.* not assessed.
- 3 worksheets covering lectures 2-19.
- Worksheets will be available on Minerva as soon as you are able to start to tackle them.
- Specimen answers will appear on Minerva roughly 1 week after the corresponding lecture.

Announcements

- Will be made from time to time in order to clarify issues or answer general questions that are raised.

Other support

Labs

- You have a timetabled lab class each week.
- Please attend these to practice solving the formative and assessed assignments.
- You will be able to get help and advice at these sessions.

Assessment (summative)

50% of the final module mark comes from the **final exam**, and the remaining 50% from **courseworks**.

Coursework	**Weight**	**Release**	**Deadline**
1	15%	6th March	10am (China), 27th March
2	20%	27th March	10am (China), 17th April
3	15%	17th April	10am, Tuesday 9th May

Before attempting the courseworks you should familiarise yourself with the relevant material:

Coursework	**Up to and including**
1	Lecture 6
2	Lecture 11
3	Lecture 16

Language

For this module we will use C.

- We will cover three different parallel libraries/API's, and the only languages that cover **all** of them are C/C++.
- Since our codes are short, we will just use C, not C++.
- Will provide starting codes in C for each coursework.
- Coursework submissions must be in C.

If you have not programmed in C for a while you may like to revise *XJCO1711 Procedural Programming*.

We will mostly use **loops**, **conditionals**, **arrays** and **pointers**.

Books

For additional information on for parallel programming **in general**:

- **Parallel Programming**, *Wilkinson and Allen* (Pearson).
  - Old (2^{nd} ed. 2005), covers CPU architectures but not GPU.
  - Many examples, though some only schematic.
  - Quite old now but material is freely available on Internet.

- **Structured Parallel Programming**, *McCool, Robison and Reinders* (Morgan-Kauffman, 2012).
  - Modern, focuses on patterns of parallel algorithm design.
  - Few code examples, mainly for shared memory systems.
  - eBook available *via* UoL library.

Books for specific architectures will also be mentioned when introduced. **You do not need to buy <u>any</u> of these books**.

Why this module?

- Almost all[1] modern computers and devices fall into one of three classes of **parallel architecture**.
- Software must be **parallelised** to use these resources.
    - This is the job of the programmer, *i.e.* **you**.
- Popular APIs/frameworks are constantly changing.
    - No point focusing on any one as it may not last.
- Need to develop **portable** skills in **parallel algorithm design** that can be applied to current **and future** APIs, frameworks and architcetures.

[1]With very few exceptions, *e.g.* feature phones *etc.*

Objectives and learning outcomes

Objectives: This module will introduce the fundamental skills and knowledge required to develop parallel computer software.

Learning outcomes: On successful completion of this module a student will have demonstrated the ability to:

- Recall key concepts of parallel software and hardware.
- Apply parallel design paradigms to serial algorithms.
- Evaluate and select appropriate parallel solutions for real world problems.
- Generalise parallel concepts to future hardware and software developments.

Skills outcomes: Programming, design, performance measurement, evaluation.

Syllabus

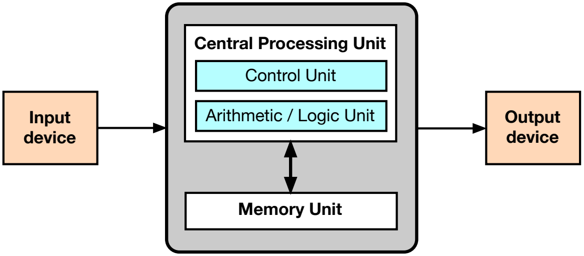
This module covers the following 3 topic areas:

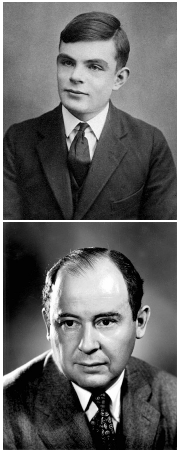
- **Parallel programming design patterns**: Work pools, data parallelism, synchronisation, locks, MapReduce and atomic instructions.
- **Parallel computation models**: shared memory parallelism (SMP), distributed memory parallelism and general purpose graphics processing unit (GPGPU).
- **Common frameworks**: OpenMP, Message passing interface (MPI) and OpenCL.

Background and motivation

Early computers followed the so-called **von Neumann architecture** (1946):

- Based on Turing's **universal machine** (1936).
- Fundamentally **sequential**, *i.e.* processes a **series** of instructions.

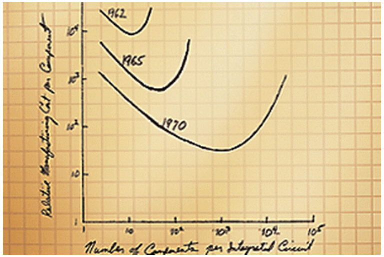



Turing (top), von Neumann (bottom) *(from Wikipedia)*

Moore's law

In 1965 Gordon Moore made the empirical observation that the number of transistors on a chip doubles every 18-24 months.

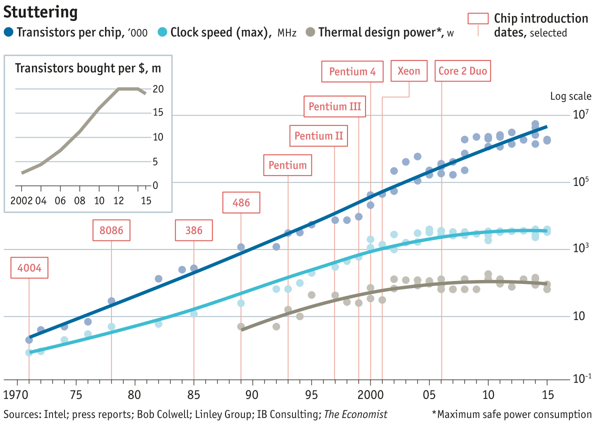
This is known as **Moore's law** and holds to this day.



http://www.startupinnovation.org/research/moores-law

- Component cost *versus* no. of components.
- Projected 1970 data.
- **Exponential increase** of the most cost-effective number of components.

Processor speeds also used to follow Moore's law, but stopped around 15 years ago at $\approx$ 3.3GHz (ignoring overclocking).



`http://www.economist.com/technology-quarterly/2016-03-12/after-moores-law`

Limitations on clock speed

Increased frequencies result in greater **leakage** and greater **power consumption**[1]:

$$P \propto C_{\mathrm{L}} V^2 f$$

- P is the processor's power consumption.
- C_{L} is a load capacitance.
- V is the supply voltage.
- f is the frequency.

However, $V \propto f$, so $\mathbf{P} \propto \mathbf{f^3}$.

- **Rapid increase** that exceeds 100W for $f \approx 3.3$GHz.
- Unsustainable even with sophisticated cooling technology.

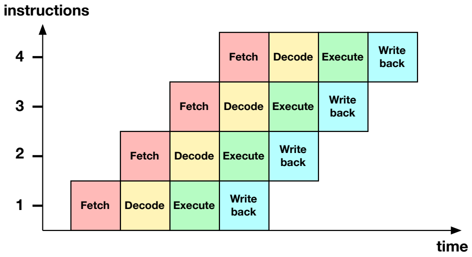
[1]You don't need to learn this equation [which was taken from *Parallel Programming*, 2nd ed., Rauber and Rünger (Springer, 2013)].

ILP: Instruction Level Parallelism

Chip designers have tried various **architectural improvements** to increase performance *(memory cache, speculative execution etc.)*

One is **pipelining**, where different stages of subsequent instructions are overlapped.

- Only one *fetch*, *decode etc.* at any given time.



This **instruction level parallelism** (ILP) is **limited** to around 10-20 instructions.

- We say it does not **scale**.

Multi-core CPUs (Lectures 2-7)

These architectural improvements did not require changes to code.

- Legacy sequential code **automatically** benefited.

Each improvement has **limitations** that have not been overcome.

Starting around 2005, chips for consumer machines have been **multi-core**, where each **core** has distinct control flows.

- For a few cores, can run applications simultaneously.

With new chips having many cores (6, 8, 12, 16, 24, 28, . . .), running one application per core is not feasible.

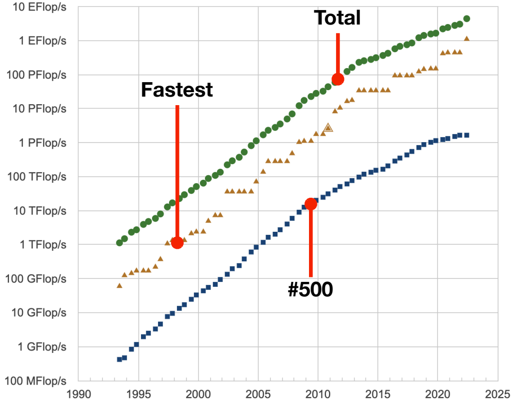
- **Single** applications need to use **multiple** cores.
- **Requires new program logic.**

Clusters / Supercomputers (Lectures 8-13)

Even before clock speeds plateaued, some applications used multiple **machines**.

- Scientific computing.
- Weather forecasting.
- ...

PFlops = **petaflops** = 10^{15} floating point operations per second;
EFlops = **exaflops** = 10^{18} floating point operations per second.



https://www.top500.org/statistics/perfdevel

GPGPU (Lectures 14-19)

In the mid-1990s, the rise of grahical applications (especially games) drove the development of **graphics accelerators**:

- Chips specialised to computations for 2D/3D graphics.

In 2006 Nvidia released the first graphics card capable of **general purpose calculations** using its CUDA architecture.

- GPGPU = General Purpose Graphics Processing Unit.
- Now supported by most manufacturers *via* OpenCL.

Suitable for other applications **including machine learning**.

- GPUs are part of the **deep learning** revolution.
- Now have dedicated **neural processing units** (NPUs).

Precedent from nature

Arguably the most complex system known is the **human brain**.

If regarded as a computer, it would be **massively parallel**:

- Synapse speeds are about 5 ms, so the ‘clock speed’ would be less than 1kHz.
- We have about 10^{11} neurons, each connected to 10^4 others.
- The current fastest supercomputer has $\approx 10^7$ cores.

http://scitechconnect.elsevier.com

Parallel *versus* concurrent

Two or more applications run **concurrently** if they both execute ‘in the same time frame.’

- *i.e.* a **multi-tasking OS**, where processes are swapped in and out without the user noticing.
- Possible on a single-core architecture.

Whereas **parallel** applications actually perform calculations **simultaneously** on a parallel architecture.

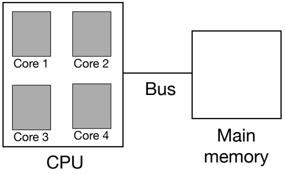
Parallelism implies concurrency, but **not** *vice versa*, *i.e.*

Parallel $\subset$ Concurrent

Shared *versus* distributed memory

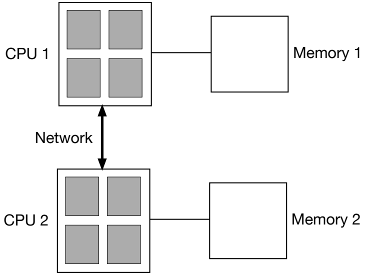
Shared memory

- All cores can 'see' whole of main memory.
- *e.g.* multi-core CPU.



Distributed memory

- Cores only see a fraction of the total memory.
- *e.g.* cluster, supercomputer.



Computation *versus* communication

Moving data to and from the cores also affects performance:

Fast: **Registers** for each processing unit.

- **Cache** memory to registers.
- **Main** memory to cache memory.
- Fast communication in **high-performance clusters** (*e.g.* InfiniBand, Gigabit Ethernet).
- **Local area** network communication (*e.g.* Ethernet).
- **File I/O**.

Slow: **Wide area** network communication (*e.g.* the internet).

Flynn's taxonomy[1]

Characterises parallel architectures by **data** and **control flows**.

Acronym	**Instruction/data streams**	**Examples**
SISD	Single Instruction, Single Data	Single-core CPU
SIMD	Single Instruction, Multiple Data	GPU *(also SIMT; c.f. Lecture 14)*
MIMD	Multiple Instruction, Multiple Data	Multi-core CPU; cluster/supercomputer
MISD	Multiple Instruction, Single Data	Specialist hardware only

[1]Flynn, *IEEE Transactions on Computers* **21**, 948 (1972).

Module overview

Lectures	**Content**
1	Introduction
2-7	**Shared memory parallelism** C with OpenMP Worksheet 1 and Coursework 1
8-13	**Distributed memory parallelism** MPI-C Worksheet 2 and Coursework 2
14-19	**General purpose GPU** OpenCL *(a C-based language)* Worksheet 3 and Coursework 3
20	Module review

Next lecture

Next lecture is the first of six on **shared memory parallelism**:

- Relevant to **multi-core architectures**, such as on modern laptops, desktops, tablets and phones.
- Overview typical hardware **architecture**, including **memory cache**.
- Look at some language and library support.
- How to install and run OpenMP programs.