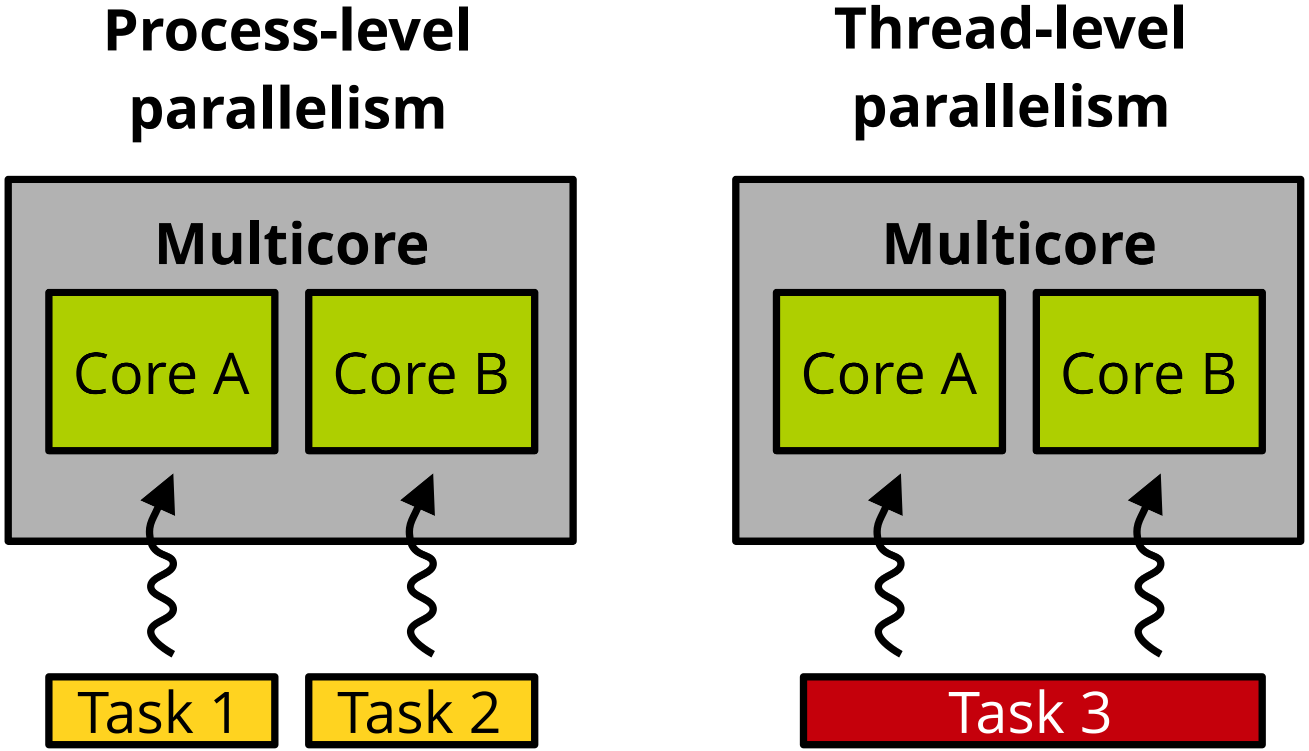
The University of Manchester

COMP35112 Chip Multiprocessors

Introduction 2

Pierre Olivier

How to Use Multiple Cores?



Process-level parallelism

Thread-level parallelism

Multicore

Core A

Core B

Task 1

Task 2

Multicore

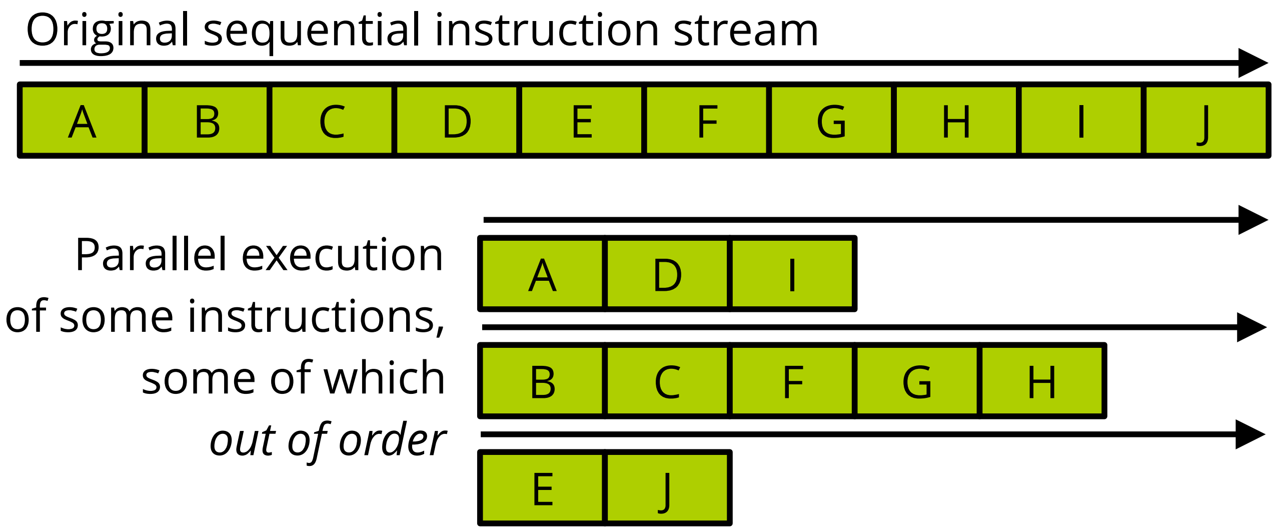
Core A

Core B

Task 3

Instruction- vs. Thread-Level Parallelism

- **Instruction Level Parallelism** (ILP)
  - Compiler/hardware **automatically** parralelises a sequential stream of instructions → **limited**



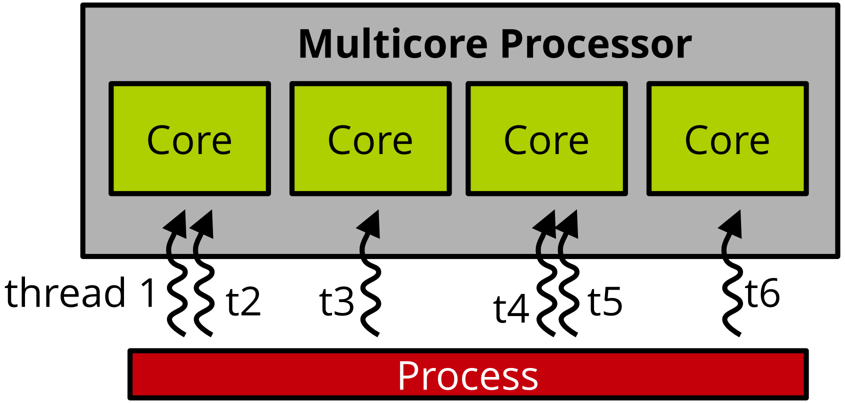
Original sequential instruction stream

Parallel execution of some instructions, some of which *out of order*

However, the overall result must be exactly the same as that produced when the whole sequence was executed in order!

Instruction- vs. Thread-Level Parallelism

- **Thread-Level Parallelism**
  - The programmer divides the program into (long) sequences of instructions ran in parallel



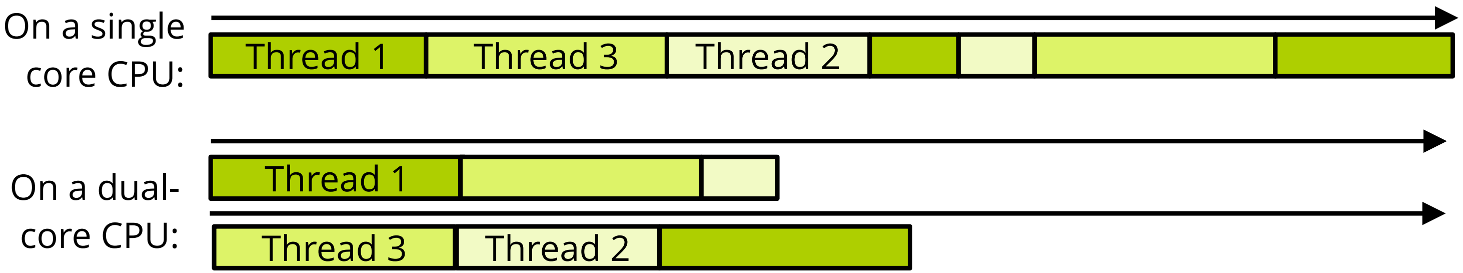
Many threads may execute in parallel and/or in any order. The overall result must be 'the same' (deterministic) whatever the details of execution!

Thread Level Parallelism

- We'll program with **threads** in the labs
- We will divide programs into concurrent sections executing as threads on different cores
- Main issue: **data sharing between threads**
  - *What happens if a thread reads a variable currently being written by another thread?*
  - Brings the need for **synchronisation** (keyword `synchronised` in Java)

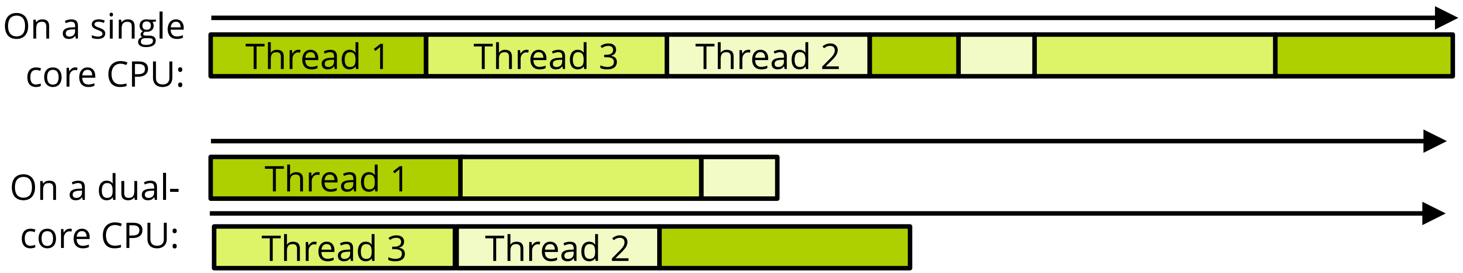
Thread Level Parallelism

- Set of threads belonging to the same program can run on a single core, total program's execution time is sum of each thread's exec. time
- On multicores threads can run in parallel, ideally execution time is `sequential execution time / number of parallel threads`



Thread Level Parallelism

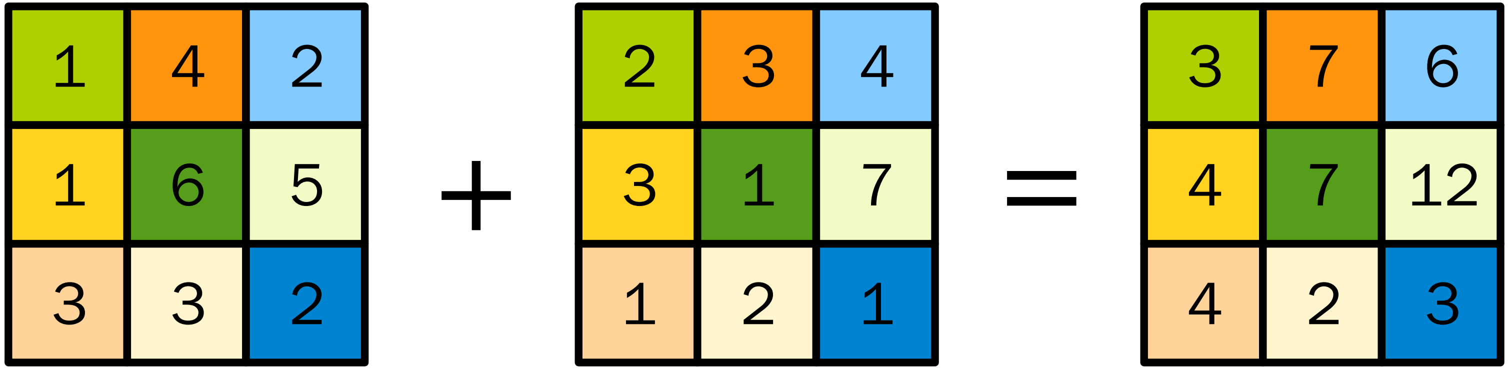
- Set of threads belonging to the same program can run on a single core, total program's execution time is sum of each thread's exec. time
- On multicores threads can run in parallel, ideally execution time is `sequential execution time / number of parallel threads`



- ILP is limited but TLP is "general purpose" and can be used to generate large amount of parallelism
  - At the cost of programmer's effort + program must be suitable

Data Parallelism

- Exploit structured parallelism contained in specific programs
- Data parallelism is usually associated with computation on a multi-dimensional array
- **Many array computations perform the same or very similar computation on all elements**



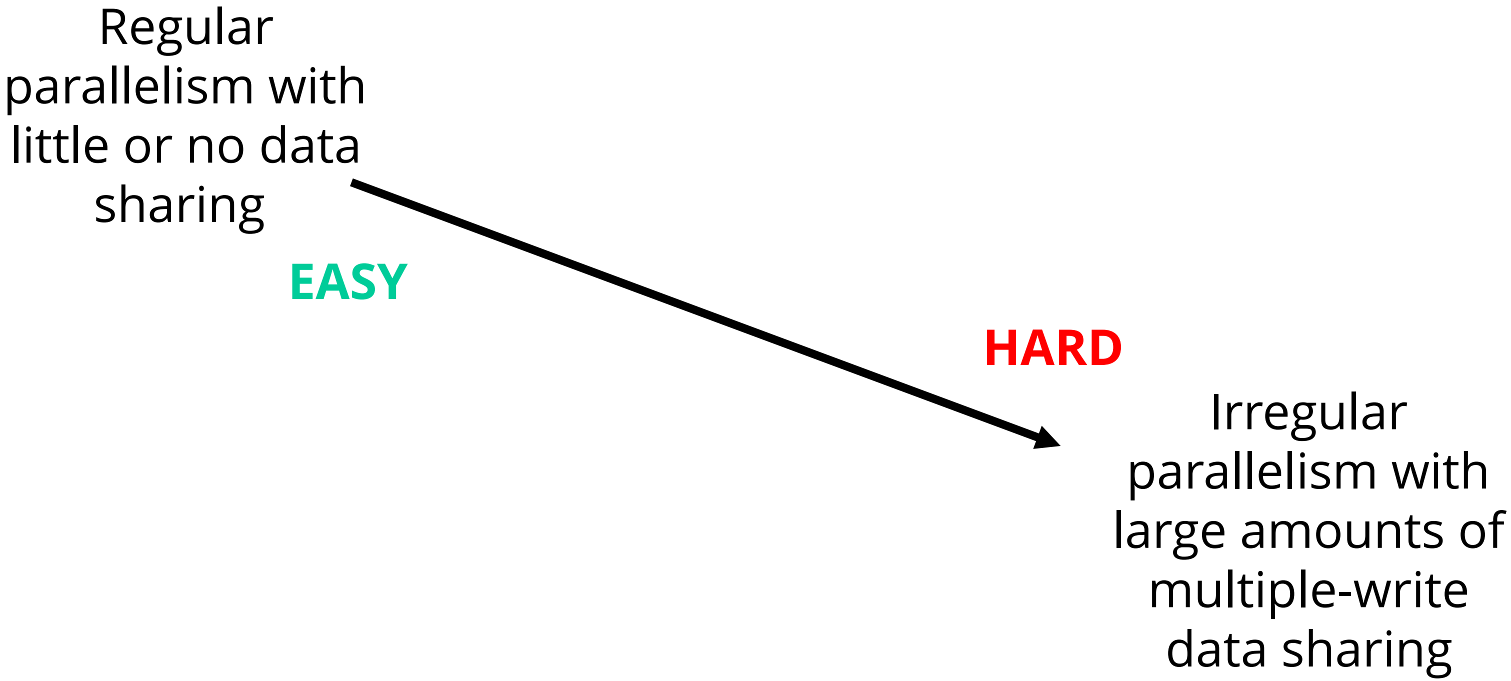
$$\begin{bmatrix} 1 & 4 & 2 \\ 1 & 6 & 5 \\ 3 & 3 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 7 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 7 & 6 \\ 4 & 7 & 12 \\ 4 & 2 & 3 \end{bmatrix}$$

Data Parallelism Examples

- **General**
  - Matrix multiply (used heavily in CNNs, for example)
  - Fourier transform
- **Graphics**
  - Anti-aliasing
  - Texture mapping
  - Illumination and shading
- **Differential Equations**
  - Weather/climate forecasting
  - Engineering simulation (and “Physics” in Games)
  - Financial modelling

Complexity of Parallelism

- Parallel programming is generally considered to be difficult, but depends a lot on the program structure



Regular parallelism with little or no data sharing

EASY

HARD

Irregular parallelism with large amounts of multiple-write data sharing

Chip Multiprocessor Considerations

- **How should we build the hardware?**
    - How are cores connected?
    - How are they connected to memory?
    - Should they reflect particular parallel programming patterns (e.g. data parallelism)?
    - Simple vs. complex cores?
    - General vs. Special Purpose (e.g. graphics processors)?

Chip Multiprocessor Considerations

- **How should we build the hardware?**
  - How are cores connected?
  - How are they connected to memory?
  - Should they reflect particular parallel programming patterns (e.g. data parallelism)?
  - Simple vs. complex cores?
  - General vs. Special Purpose (e.g. graphics processors)?
- **How should we program them?**
  - Extended ‘conventional’ languages?
  - Domain specific languages?
  - Totally new approaches?

Overview of Lectures

- Thread-based programming, thread synchronisation
- Cache coherency in homogeneous shared memory multiprocessors
- Hardware support for thread synchronisation
- Operating system support for threads, concurrency within the kernel
- Alternative programming views
- Speculation and transactional memory
- Heterogeneous processors/cores/programs
- Radical approaches (e.g. dataflow programming)

Summary

- Single core performance has been plateauing but we can still pack more transistors on a single chip
- Put multiple compute units on a single integrated circuit: **chip multiprocessors**
- Has important implications
  - Hardware: what CPUs to use, how are they connected, do they shared memory?
  - Software: how to program these things?
- Interesting read: http://www.gotw.ca/publications/concurrency-ddj.htm