MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

159.341 Programming Languages, Algorithms & Concurrency

Concurrent Programming
Parallel Hardware

Daniel Playne
d.p.playne@massey.ac.nz

What is Concurrent Programming?

Concurrent programming is the task of writing computer programs that use concurrent execution to solve a problem.

The tricky part is actually making them work as intended to give a correct solution to the problem.

Concurrent programming often requires the use of synchronisation primitives or other communication methods to ensure safe access to shared memory/resources and to avoid deadlock.

Reminder - Concurrency vs Parallelism

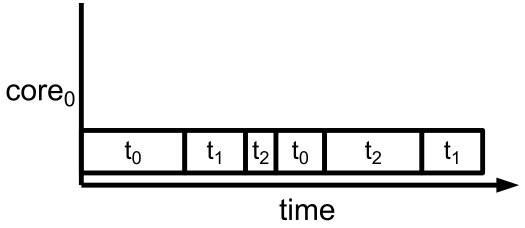
In the literature you will see the terms *concurrent computing* and *parallel computing* used to mean slightly different things by different authors.

Concurrency and *parallelism* are very closely related and are sometimes used synonymously, however they are two different concepts.

Reminder - Concurrency vs Parallelism

Concurrency - multiple tasks are *in progress* at the same time.

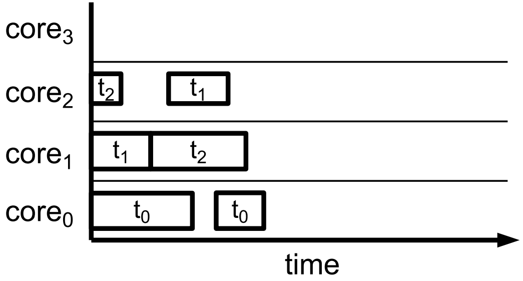
A single-core machine can still provide concurrency by switching back and forth between different threads even though only one instruction is executed at once.



Reminder - Concurrency vs Parallelism

Parallelism - multiple instructions are *being executed* at the same time.

Parallelism requires multiple processing units that are able to execute more than one instruction at the same time (within the same clock cycle).



What is Parallel Programming?

Unlike ***concurrent*** programming, which may be used simply for convenience, ***parallel*** programming often comes with an expectation of performance improvement.

One of the challenges of parallel programming over sequential programming is that the programs you write, the programming languages/APIs/libraries you use, and even the algorithms themselves are more dependant on the hardware that will be used to run the program.

Parallel Computing Works!

One of the challenges in the parallel computing area is disagreement on exactly what parallel computing is. In many ways the more practical question is whether parallel machines can be used to solve real problems.

"Solving real problems with real software on real hardware."

G. Fox. Parallel Computing Works! Morgan Kaufmann

Concurrent Programming

For this reason our exploration of concurrent/parallel programming will start by looking at types of parallel hardware.

- ***Hardware Classification***
- Types of Parallelism
- Algorithm Design
- Paradigm
- Program

Parallel Processing Hardware

There are a number of different general types of parallel processing architecture.

Understanding the architecture you are working with (at least at a high-level) can be useful when designing parallel algorithms and developing parallel programs.

Different parallel architectures may require completely different approaches, algorithms and languages/language features.

Memory Architecture

One way to classify a parallel computer is by the memory architecture it employs.

The memory architecture defines how the memory is structured and accessed by different processors/cores.

The two basic architectures are ***shared*** and ***distributed*** memory machines.

Shared Memory

In a ***shared memory*** architecture, all processors have access to the entire memory as a single, global address space.

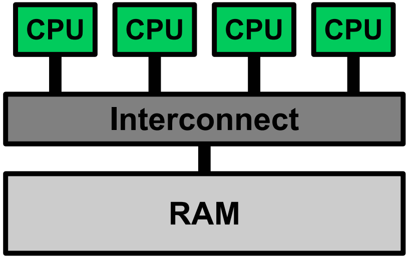
The exact designs of the machines differ widely but share the same characteristics:

- All processors have access to the same memory resources
- Changes made to a memory location by one processor are visible to all other processors.

Shared Memory - UMA

Shared memory architectures are often further categorised as either ***UMA*** and ***NUMA*** machines.

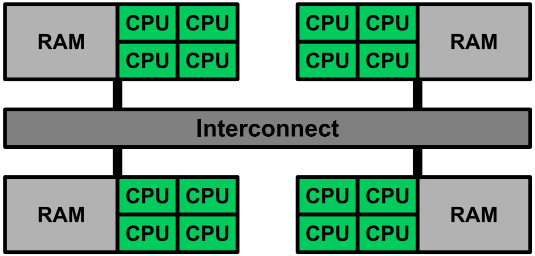
In a ***Uniform Memory Access (UMA)*** machine, all processors have equal access (and equal access times) to all memory locations.



Shared Memory - NUMA

In a ***Non-Uniform Memory Access (NUMA)*** machine, each processor still has direct access to every memory location but the access times will vary between processors and memory locations.

Memory accesses that have to go through the interconnect will be slower than those that don't.



Shared Memory

Advantages:

- Convenient global address space for memory locations.
- Data sharing is fast and simple.

Disadvantages:

- Hard to scale, increasing the number of processors geometrically increasing the traffic on the interconnect.
- Programmer is responsible for synchronisation and correctly protected access to memory.

Distributed Memory

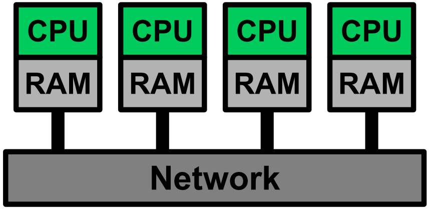
In a ***distributed memory*** architecture, each processor has its own local memory and cannot directly access the memory of another.

Memory locations in one area of local memory does not map to other processor's memory (no global address space).

Accessing data from another processor's memory requires the two processors to communicate with each other across a network - usually defined explicitly by the programmer.

Distributed Memory

There are a lot of ways to build the network connection between machines but it can be as simple as an ethernet connection.



Distributed Memory

Advantages:

- Easy to scale with the number of processors.
- Each processor has fast access to its local memory without interference from other processors.
- Can be built using commodity equipment.

Disadvantages:

- Programming is responsible for data communication between processors.
- Non-uniform memory access - data residing on a remote node is significantly slower to access.

Flynn’s Taxonomy

Flynn’s Taxonomy is a classification often used to describe computing architectures.

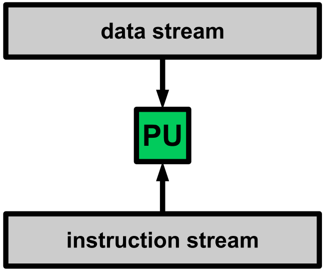
This taxonomy classifies computing architectures based on how they treat the ***instruction stream*** and the ***data stream***.

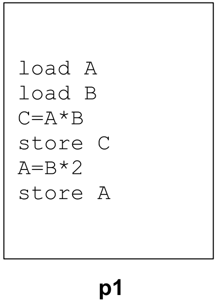
SISD single instruction single data	**SIMD** single instruction multiple data
MISD multiple instruction single data	**MIMD** single instruction multiple data

Flynn's Taxonomy - SISD

Single instruction single data (SISD) describe a single processing unit executes a single stream of instructions and acts on a single stream of data.

During 1 clock cycle, 1 instruction is executed that acts on 1 data stream.





```
load A
load B
C=A*B
store C
A=B*2
store A
```

Flynn’s Taxonomy - SISD

SISD machines describe the classical von Neumann architecture and are the oldest type of computer.

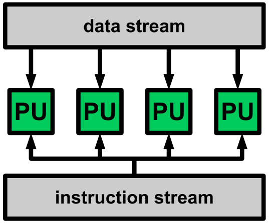
There are countless examples of architectures with this design:

- Old generation mainframes
- Workstations
- Single processor/core PCs
- etc.

Flynn’s Taxonomy - SIMD

Single instruction multiple data (SIMD) machines have multiple processing elements simultaneously performing the same sequence of instructions on multiple data items.

During 1 clock cycle, 1 instruction is executed on items from multiple data streams.



```
load A[1]
load B[1]
C[1]=A[1]*B[1]
store C[1]
```

p1

```
load A[2]
load B[2]
C[2]=A[2]*B[2]
store C[2]
```

p2

```
load A[n]
load B[n]
C[n]=A[n]*B[n]
store C[n]
```

pn

Flynn's Taxonomy - SIMD

SIMD machines are a type parallel computer as there are multiple processing units executing instructions at the same time.

This type of architecture is well-suited to regular problems (e.g. computer graphics and image processing).

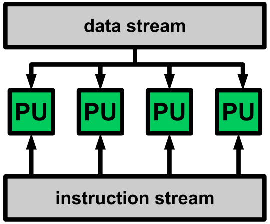
Examples include:

- Thinking Machines CM-2
- CPU vector units
- Graphical Processing Units

Flynn's Taxonomy - MISD

Multiple-instruction single-data (MISD) architectures contain multiple processing units that execute different instructions on the same data stream.

During 1 clock cycle, multiple instructions are acting on 1 data stream.



```
load A[1]
C[1]=A[1]*1
store C[1]
```

p1

```
load A[1]
D=A[1]*2
B[i]=D+8
```

p2

```
load A[1]
M[n]=A[1]
M[j]=m[n]+2
```

pn

Flynn’s Taxonomy - MISD

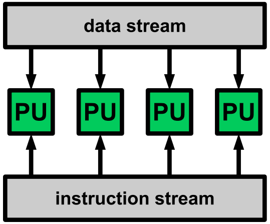
MISD is an unusual architecture as there are a very limited set of problems that are suited to this method of computing.

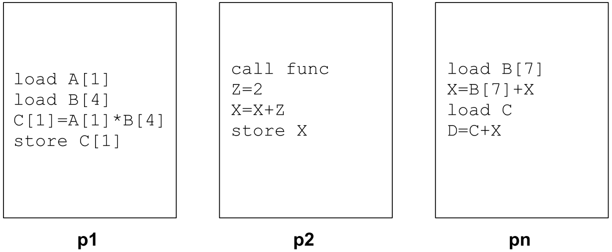
Few (if any) examples of this type of parallel computer have ever been built.

Flynn's Taxonomy - MIMD

Multiple-instruction multiple-data (MIMD) is another common architecture where multiple processing elements execute multiple instruction streams on multiple data streams.

During 1 clock cycle, multiple instructions are acting on multiple data streams.





```
load A[1]
load B[4]
C[1]=A[1]*B[4]
store C[1]
```

p1

```
call func
Z=2
X=X+Z
store X
```

p2

```
load B[7]
X=B[7]+X
load C
D=C+X
```

pn

Flynn's Taxonomy - MIMD

MIMD are the most common type of parallel computer with most computers now falling into this category.

Examples include:

- Multi-processor/core PCs
- Computer clusters
- Most modern Supercomputers

Note: MIMD architectures may contain SIMD components.

Hardware Classification

Knowing the type of processing architecture is important when designing both parallel algorithms and parallel programming languages / programming models.

Different types of machines may have different restrictions on how a program operates and have very different costs associated with communication and synchronisation between different processing cores.

Summary

- Hardware Classification
- Memory Architecture
  - ***Shared memory***
  - ***Distributed memory***
- Flynn's Taxonomy
  - ***SISD*** - Single Instruction Single Data
  - ***SIMD*** - Single Instruction Multiple Data
  - ***MISD*** - Multiple Instruction Single Data
  - ***MIMD*** - Multiple Instruction Multiple Data