HELP US ALL STAY HEALTHY

SIX SIMPLE TIPS

Maintain 1.5 metres distance between yourself and others where possible

Cough and sneeze into your elbow or a tissue (not your hands)

Avoid shaking hands

Put used tissues in the bin

Wash hands with soap and warm water or use an alcohol-based hand sanitiser after you cough or sneeze

Do not touch your face

IF YOU ARE UNWELL AND WORRIED ABOUT COVID-19:

- Call the National Coronavirus Helpline: 1800 020 080
- Call your usual GP for advice
- Call the UWA Medical Centre for advice: 6488 2118

UWA FAQs:
uwa.edu.au/coronavirus

Report COVID-19 hazards and suspected/confirmed cases via RiskWare:
uwa.edu.au/riskware

THE UNIVERSITY OF WESTERN AUSTRALIA

DCS:507757521

THE UNIVERSITY OF WESTERN AUSTRALIA

High-Performance Computing

Lecture 2 Introduction to Parallel Programming and OpenMP

CITS5507

Zeyi Wen

Computer Science and Software Engineering

School of Maths, Physics and Computing

Acknowledgement: The lecture slides are adapted from many online sources.

Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

Why Parallel Programming

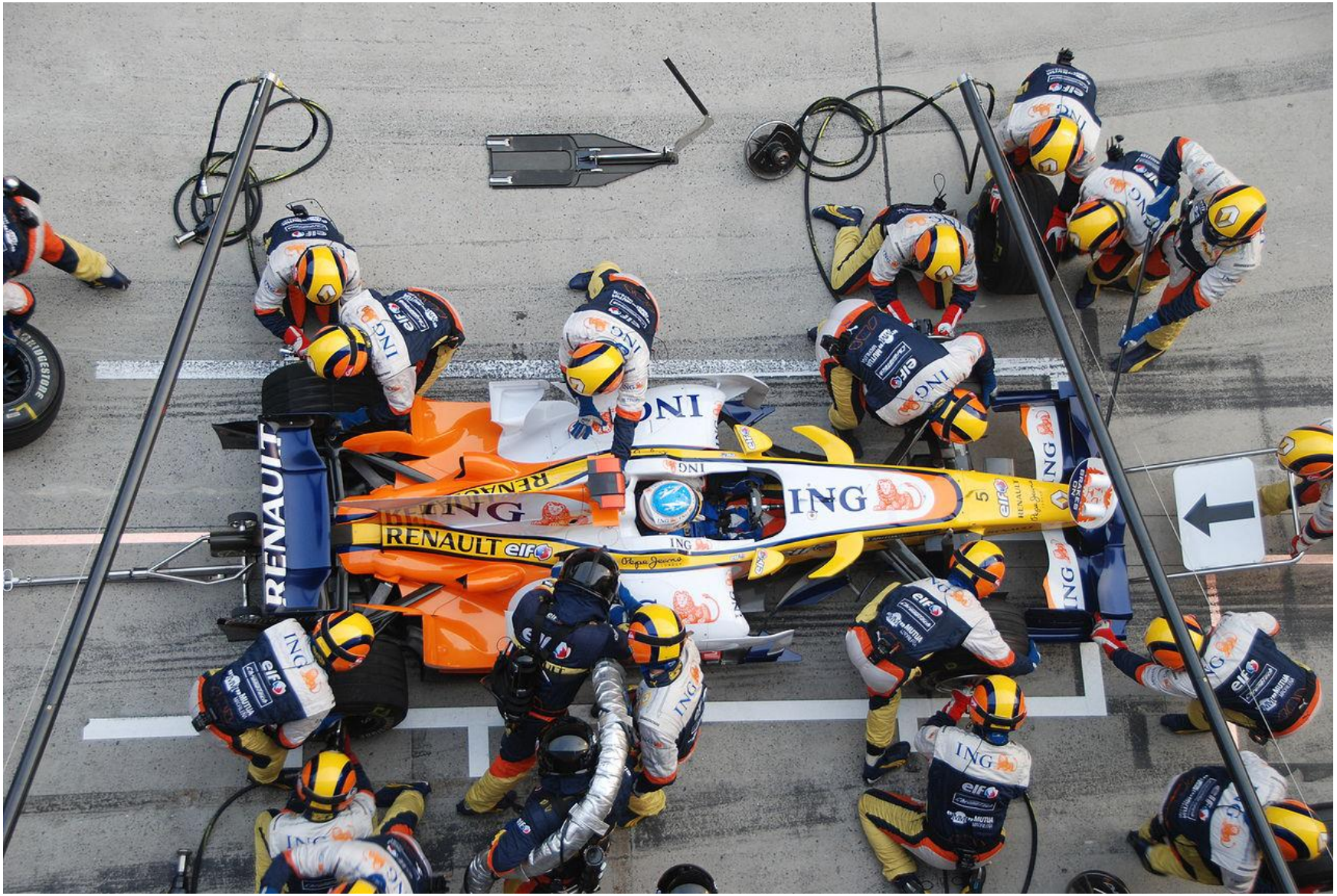
- **To solve larger problems**
  - many applications need significantly more memory than a regular PC can provide/handle
  - many computers, or “nodes” can be combined into a cluster

- **To solve problems faster**
  - despite of many advances in computer hardware, many applications are running slower and slower
    - ✓ databases having to handle more and more data
    - ✓ working on more accurate solutions

- **Make use of less powerful hardware**

Parallel Programming Analogy

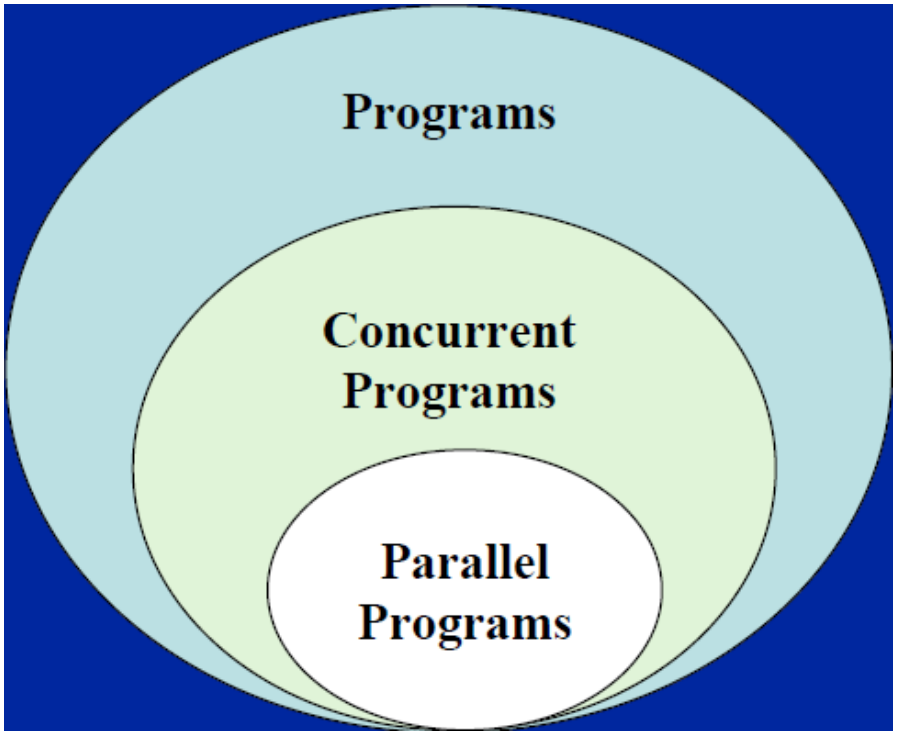
THE UNIVERSITY OF WESTERN AUSTRALIA



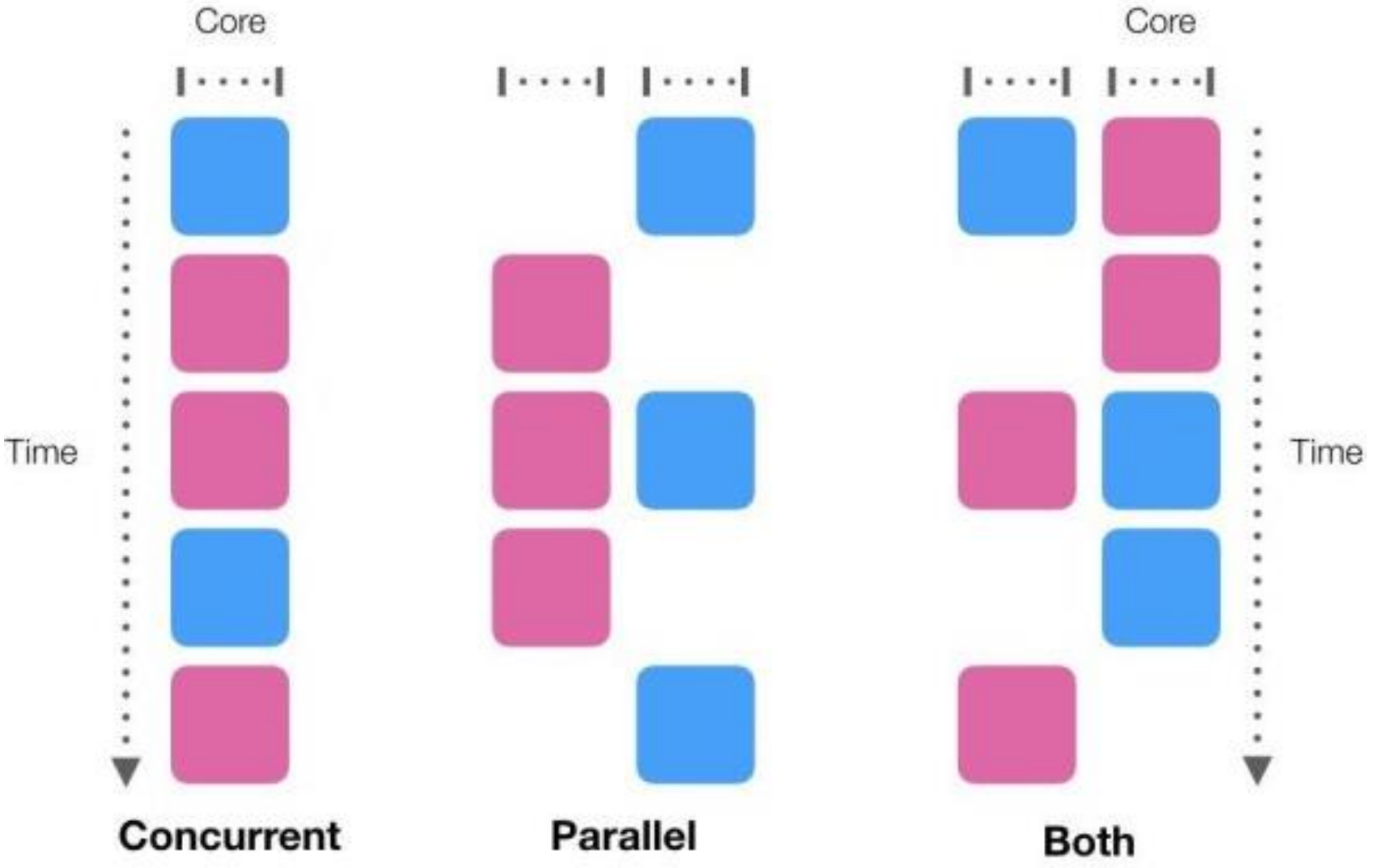
Concurrent and Parallel Programs

Two important definitions:

- Concurrency: A condition of a system in which multiple tasks are logically active at one time.
- Parallelism: A condition of a system in which multiple tasks are actually active at one time.

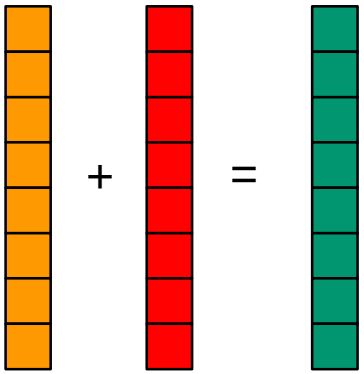


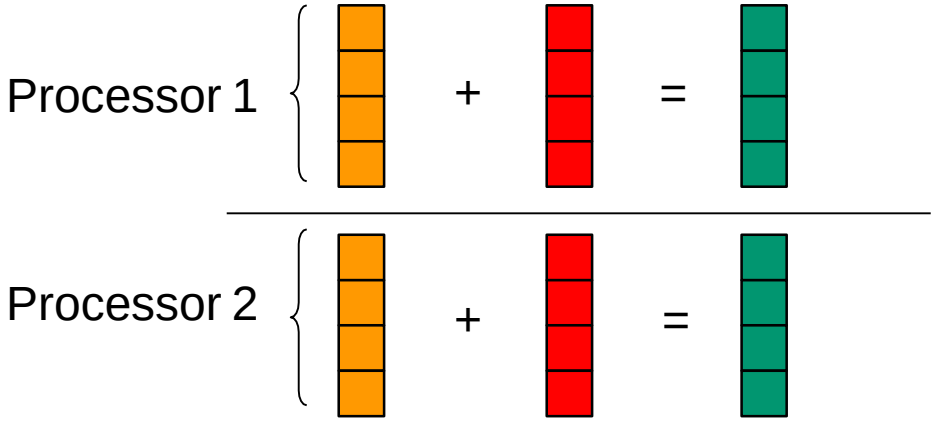
(Lecture 1) Concurrency v.s. Parallelism



Parallel Programming

- **Exploit concurrency**
  - Internet: client and server are independent, interacting applications
  - Searching an element: distribute the search database onto multiple processors
  - Adding two arrays of integers:





Parallel Programming (Continued)

- **Scalar product:**

$$s = \sum_{i=0}^{N-1} a[i] * b[i]$$

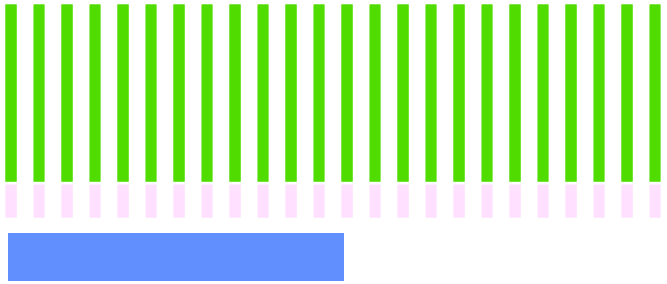
- **Parallel algorithm**

$$s = \sum_{i=0}^{N/2-1} a[i] * b[i] + \sum_{i=N/2}^{N-1} a[i] * b[i]$$

Parallel Programming Process

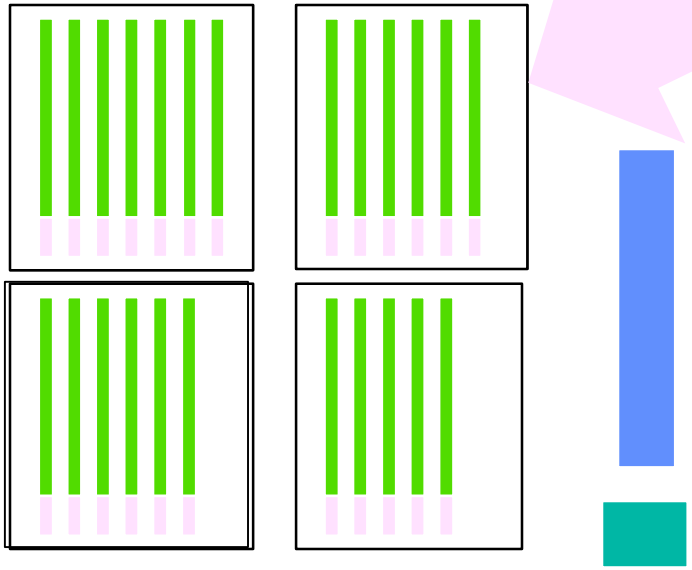
Original Problem

Find Concurrency



Tasks, shared and local data

Algorithm strategy



Units of execution + new shared data for extracted dependencies

Implementation strategy

```
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
{
TYPE *tmp, *func();
global_array Data(TYPE);
global_array Res(TYPE);
int Num = get_num_procs();
int id = get_proc_id();
if (id==0) setup_problem(N, Data);
for (int I= ID; I<N;I=I+Num){
tmp = func(I, Data);
Res.accumulate( tmp);
}
}
```

Job Scheduling

THE UNIVERSITY OF WESTERN AUSTRALIA

- Integral to parallel computing
  - ✓ assign tasks to cores
- Job scheduling also used in
  - ✓ batch jobs,
  - ✓ multiple users,
  - ✓ resource sharing,
  - ✓ system monitoring

Realistic Expectations

THE UNIVERSITY OF WESTERN AUSTRALIA

- Ex. – Your program takes 20 days to run
- 95% can be parallelized
- 5% cannot (serial)
- What is the fastest this code can run?
  - ✓ As many CPU's as you want!

1 day!

Amdahl's Law

- Hardware speed measured in FLOPS
  - ✓ FLOPS: Floating Point Operation Per Second

Disadvantages and Issues

THE UNIVERSITY OF WESTERN AUSTRALIA

- No free lunch - can't just "turn on" parallel
  - ✓ It's a lot more complex to implement
- Parallel programming requires work
  - ✓ Code modification – **always**
  - ✓ Algorithm modification – often
  - ✓ New sneaky **bugs** – you bet
- Speedup limited by many factors
- **Not everything benefits**
  - ✓ Many problems must be solved sequentially (e.g. protein folding)
  - ✓ Interactive stuff

Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

Computer Architectures

- As you consider parallel programming understanding the **underlying architecture** is important (review Lecture 1).

- Performance is affected by hardware configuration
  - ✓ Memory or CPU architecture
  - ✓ Numbers of cores/processor
  - ✓ Network speed and architecture

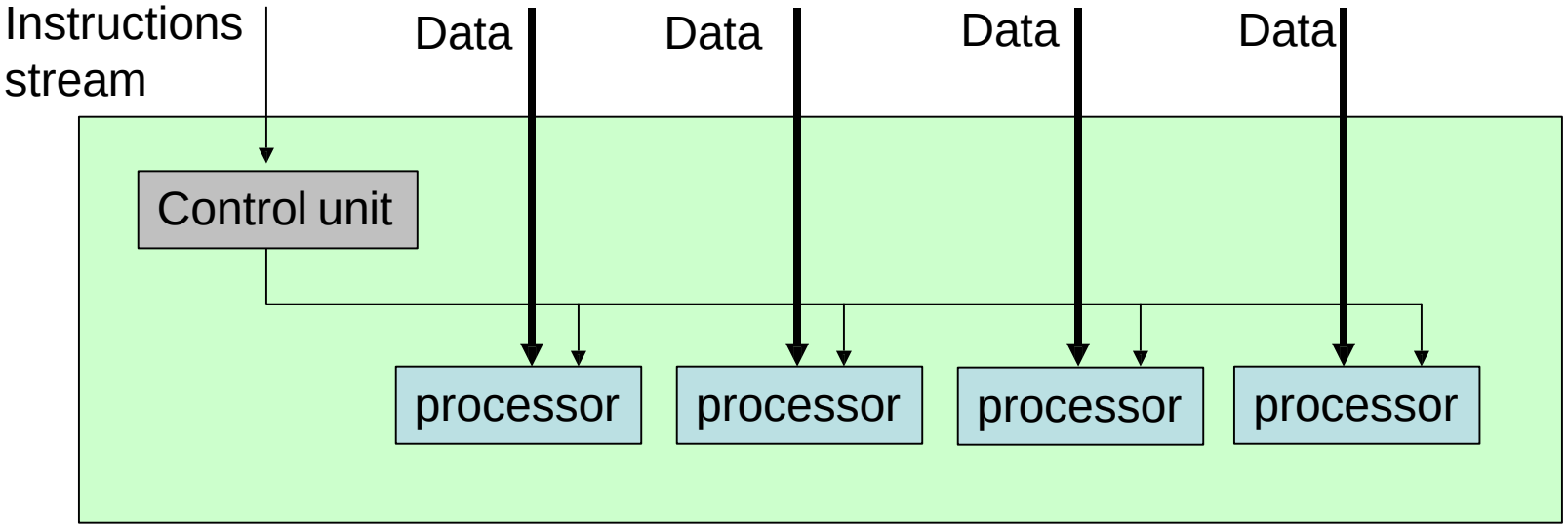
Classification of Parallel Architectures

Flynn's Taxonomy

- SISD: Single instruction single data
  - Classical von Neumann architecture
- SIMD: Single instruction multiple data
- MISD: Multiple instructions single data
  - Non existent, just listed for completeness
- MIMD: Multiple instructions multiple data
  - Most common and general parallel machine

Single Instruction Multiple Data (SIMD)

- Also known as array/vector-processors
- A single instruction stream is broadcasted to multiple processors, each having its own data stream
  - Still used in graphics cards (i.e. GPUs) today
  - CPUs can also support SIMD (e.g. AVX-512)



Multiple Instructions Multiple Data (MIMD)

THE UNIVERSITY OF WESTERN AUSTRALIA

- Each processor has its own instruction stream and input data
- Very general case
  - every other scenario can be mapped to MIMD
- Further breakdown of MIMD usually based on the memory organisation
  - Shared memory systems
  - Distributed memory systems

Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

GPU Architectures

THE UNIVERSITY OF WESTERN AUSTRALIA

- Like a multi-core CPU, but with thousands of cores
- Has its own memory to calculate with
- GPU advantages
  - ✓ higher computation power than CPUs
  - ✓ can be thousands of simultaneous calculations
  - ✓ relatively cheap in terms of FLOPS or cores per $
- Modern GPU frameworks
  - ✓ CUDA: Proprietary, easy to use, sponsored by NVIDIA and only runs on their cards
  - ✓ OpenCL: Open, a bit harder to use, runs on both AMD and NVIDIA GPUs; also can run on CPUs

CPU or GPU? Cluster or Multicore?

THE UNIVERSITY OF WESTERN AUSTRALIA

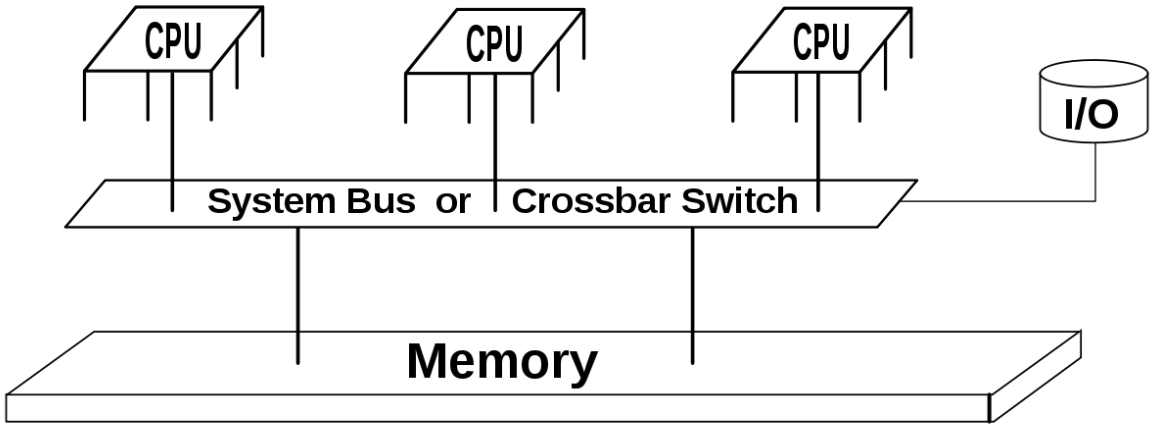
- CPU or GPU?
  - ✓ CPU: Easier to program for, has much more powerful individual cores
  - ✓ GPU: Trickier to program for, thousands of relatively weak cores

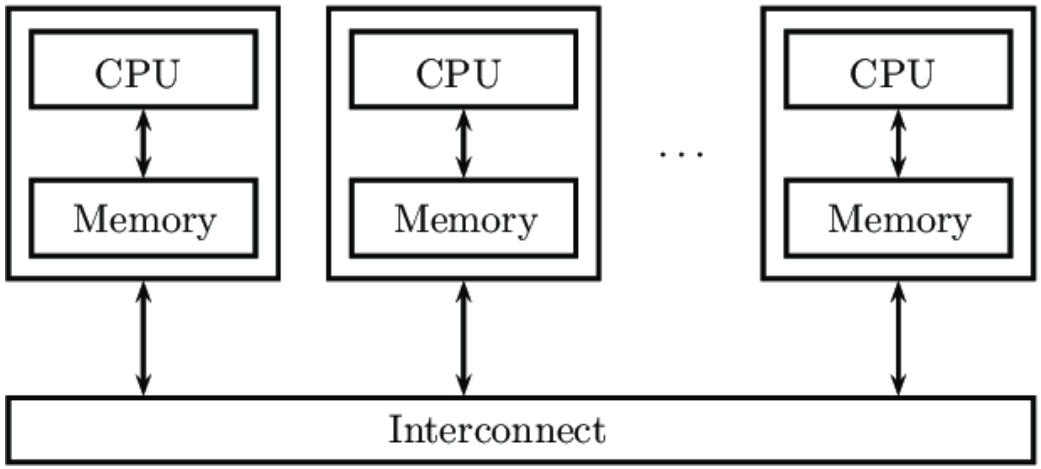
- Cluster or Multicore?
  - ✓ Multicore: All the cores are in a single computer, usually shared memory.
  - ✓ Cluster: Many computers linked together, each with individual memory.

Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

Shared and Distributed Memory Systems





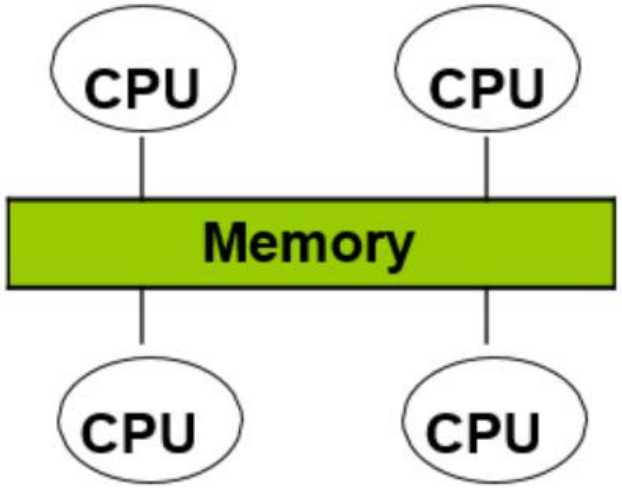
Shared Memory Systems

THE UNIVERSITY OF WESTERN AUSTRALIA

- All processes have access to the same address space
  - A computer/machine with more than one processor
- Data exchange between processes by writing/reading shared variables
  - Shared memory systems are easy to program
  - Popular programming interface: OpenMP
- Two versions of shared memory systems available today
  - Symmetric multiprocessors (SMP)
  - Non-uniform memory access (NUMA) architectures

Symmetric Multi-Processors (SMPs)

- All processors share the same physical main memory

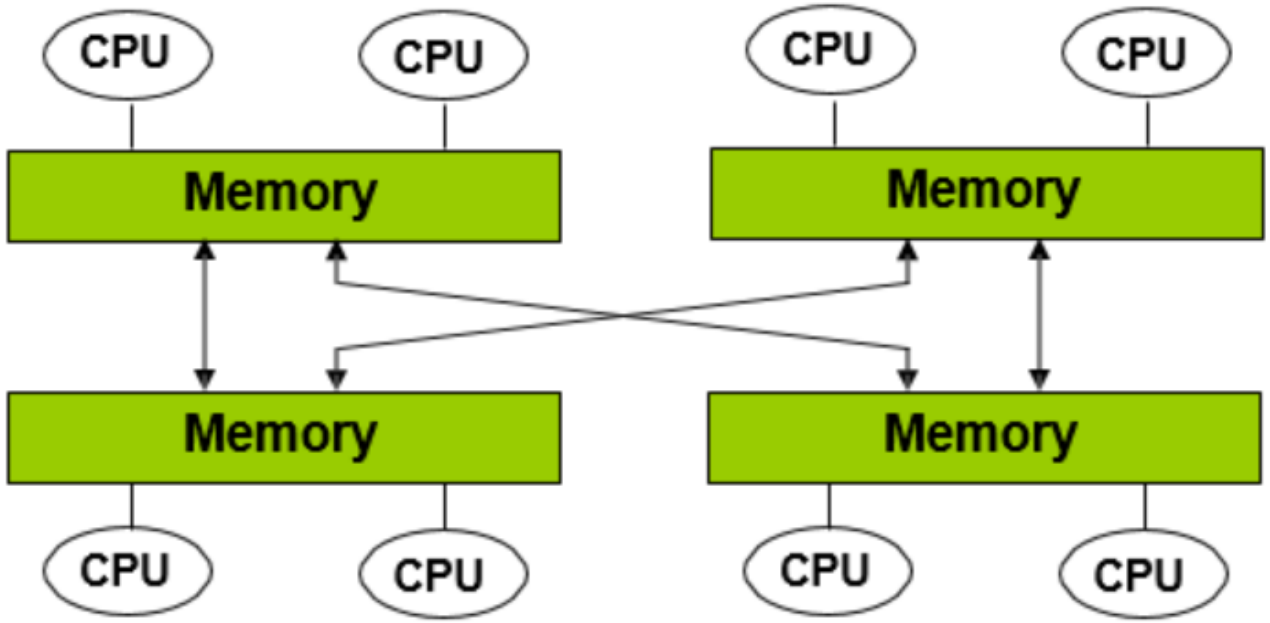


- Memory bandwidth per processor is a limiting factor for this type of architecture

- Typical size: 2-32 processors

NUMA Architectures

- Some memory is closer to a certain processor than other memory
  - The whole memory is still addressable from all processors
  - Depending on what data item a processor retrieves, the access time might vary strongly

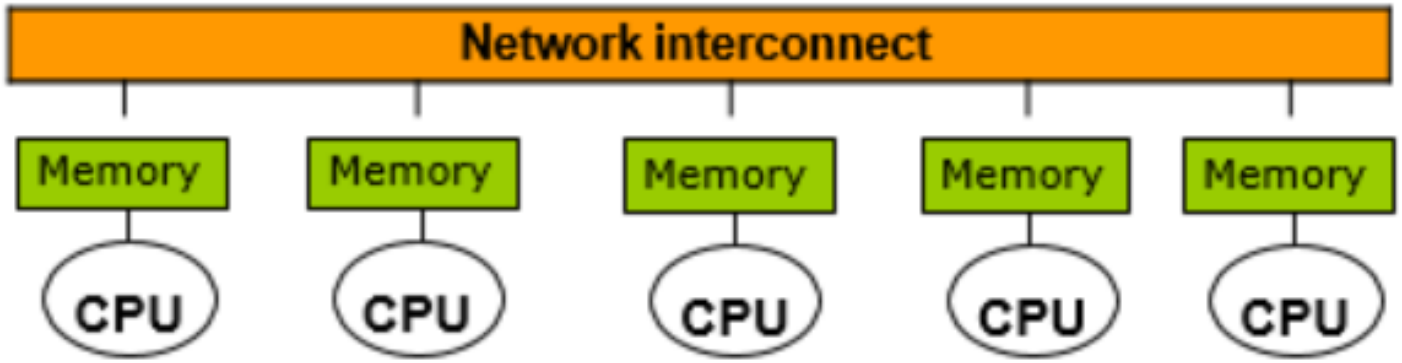


NUMA Architectures (Continued)

- Reduces the memory bottleneck compared to SMPs
- More difficult to program efficiently
  - E.g. first touch policy: data item will be located in the memory of the processor which uses a data item first
- To reduce effects of non-uniform memory access, caches are often used
  - ccNUMA: cache-coherent non-uniform memory access architectures
- Example: SGI Origin with 512 processors

Distributed Memory Machines

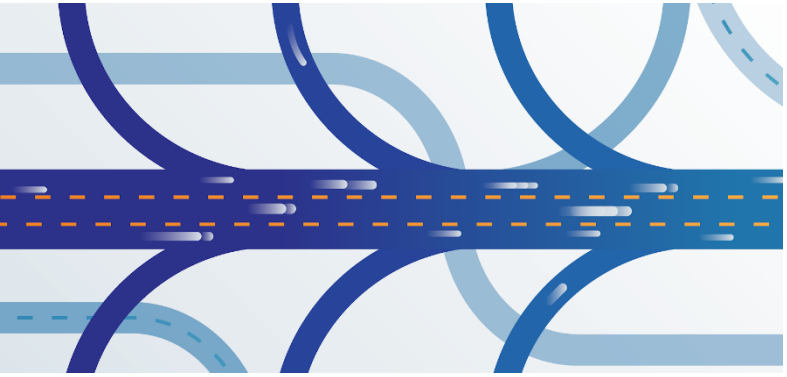
- Each processor has its own address space
- Communication between processes by explicit data exchange
  - Sockets (a term in computer network)
  - Message passing (this unit covers in the 2nd half)
  - Remote procedure call/remote method invocation

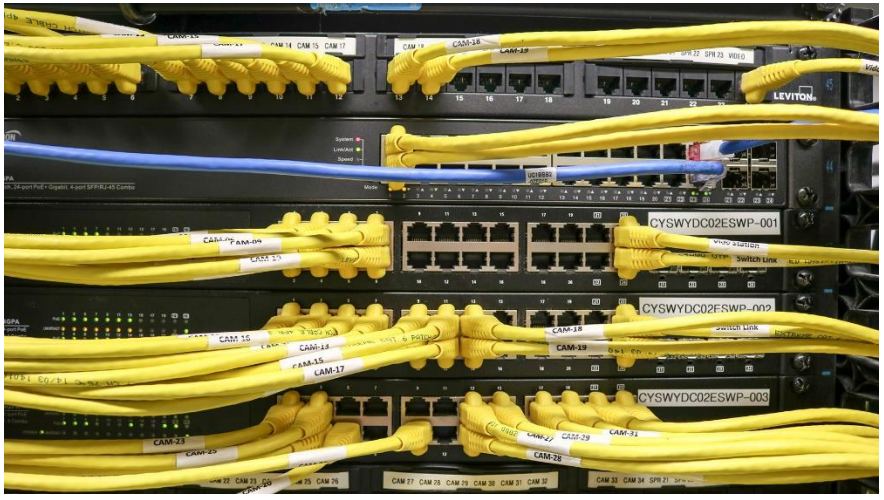


Distributed Memory Machines (Continued)

THE UNIVERSITY OF WESTERN AUSTRALIA

- Performance of a distributed memory machine strongly depends on the quality of the network interconnect and the topology of the network interconnect
  - Of-the-shelf technology: fast-Ethernet, gigabit-Ethernet
  - Specialised interconnects: Myrinet, **Infiniband**, Quadrics, 10G Ethernet…





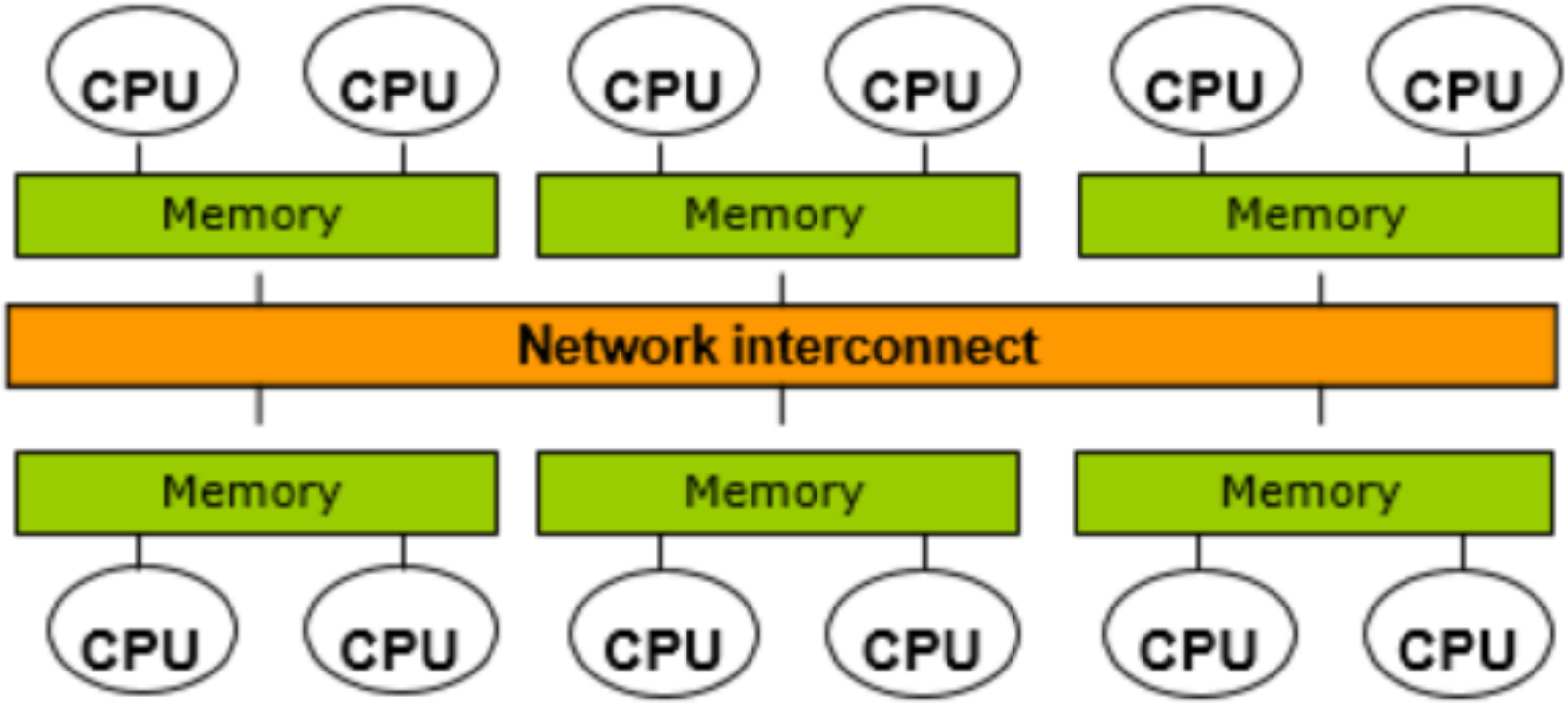
Distributed Memory Machines (Continued)

THE UNIVERSITY OF WESTERN AUSTRALIA

- Two classes of distributed memory machines:
  - Massively parallel processing systems (MPPs)
    - Tightly coupled environment
    - Single system image (specialised OS)
  - Clusters: of-the-shelf hardware/software components
    - Intel Pentium, AMD Opteron, etc.
    - Standard operating systems such as LINUX, Windows, BSD UNIX

Hybrid systems

- E.g. clusters of multi-processor nodes

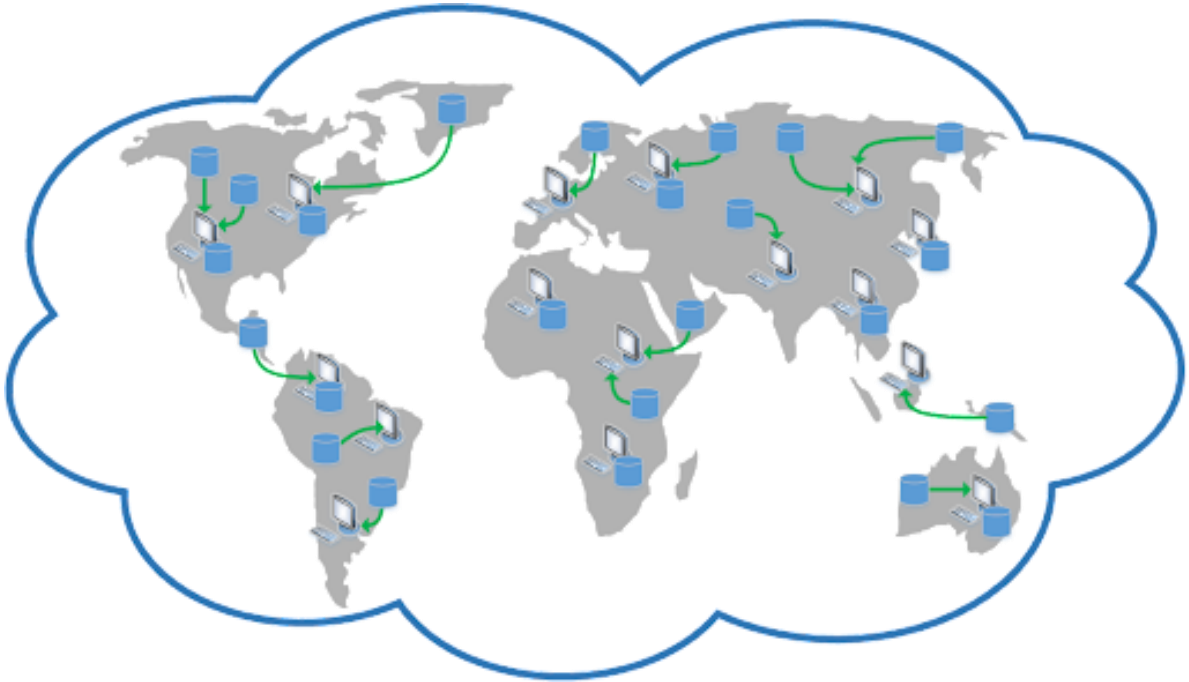


OpenMP and MPI

- **Message Passing Interface (MPI)–designed for distributed memory**
  - Multiple systems
  - Send/receive messages
- **OpenMP (Open Multi-Processing)–designed for shared memory**
  - Single system with multiple cores
  - One thread/core sharing memory
- **C, C++, and Fortran**
- **There are other options**
  - Interpreted languages with multithreading
    - Python, R, matlab (have OpenMP & MPI underneath)
  - CUDA, OpenACC (GPUs)
  - Pthreads, Intel Cilk Plus (multithreading)
  - OpenCL, Chapel, Co-array Fortran, Unified Parallel C (UPC)

Grid

- “Evaluation” of distributed memory machines and distributed computing
- Several (parallel) machines connected by wide-area links (typically the internet)
  - Machines are in different administrative domains



Outline

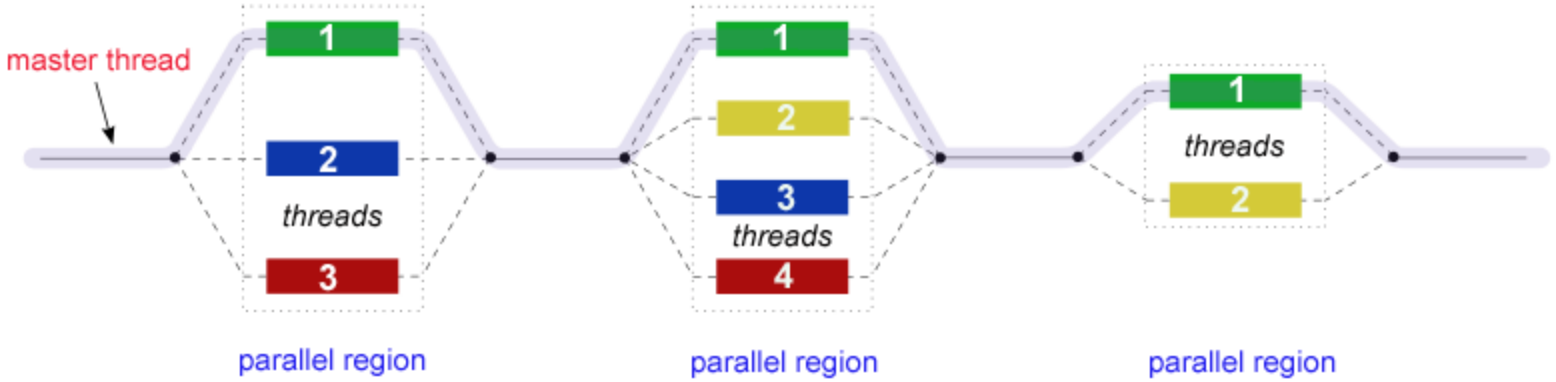
- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

OpenMP

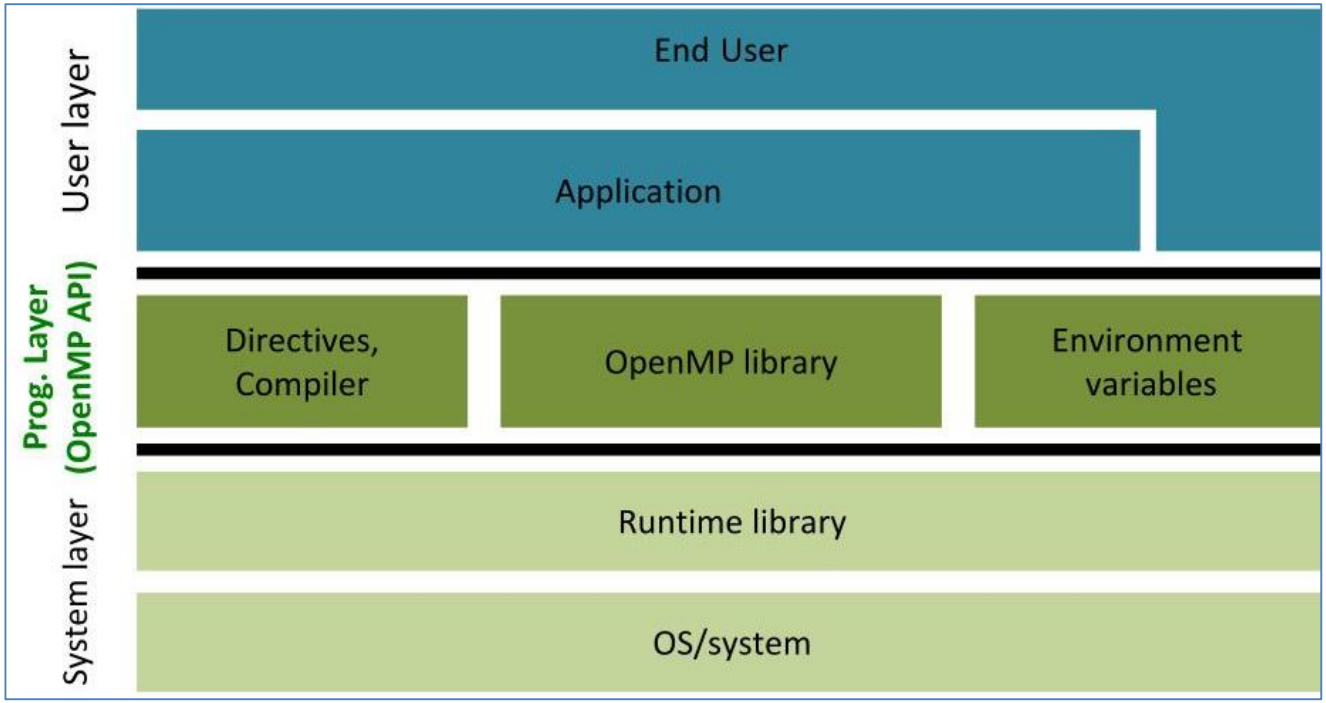
- **What is it?**
  - Open Multi-Processing
  - Completely independent from MPI
  - Multi-*threaded* parallelism
- **Standard since 1997**
  - Defined and endorsed by the major players
- **Fortran, C, C++**
- **Requires compiler to support OpenMP**
  - Nearly all do
- **For shared memory machines**
  - Limited by available memory
  - Some compilers support GPUs

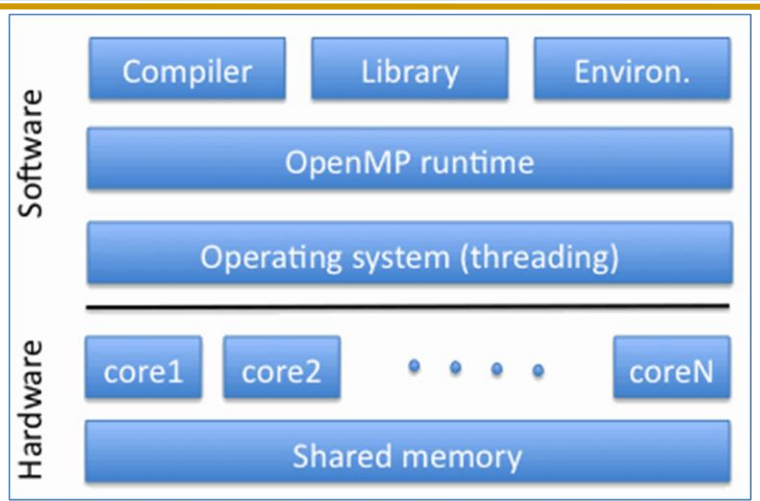
OpenMP (Continued)

- OpenMP is one of the most common parallel programming models in use today.
- It is relatively easy to use which makes a great language to start with when learning to write parallel programs.



OpenMP Solution Stack





Preprocessor Directives

THE UNIVERSITY OF WESTERN AUSTRALIA

- Preprocessor directives tell the compiler what to do
- Always start with #
- You've already seen one:

```
#include <stdio.h>
```

- OpenMP directives tell the compiler to add machine code for parallel execution of the following block

```
#pragma omp parallel
```

- "Run this next set of instructions in parallel"

Some OpenMP Subroutines

```
int omp_get_max_threads()
```

- Returns max possible (generally set by OMP_NUM_THREADS)

```
int omp_get_num_threads()
```

- Returns number of threads in current team

```
int omp_get_thread_num()
```

- Returns thread id of calling thread
- Between 0 and omp_get_num_threads-1

(Lecture 1) Process and Thread

THE UNIVERSITY OF WESTERN AUSTRALIA

- A process can be considered as an independent execution environment in a computer system.
- There are usually many processes in a system at any time, each with its own memory space.
- Each process executes a sequence of instructions (the machine language program).
- Threads are also independent execution environments, but with a shared memory space (or address space).

Process vs. Thread

THE UNIVERSITY OF WESTERN AUSTRALIA

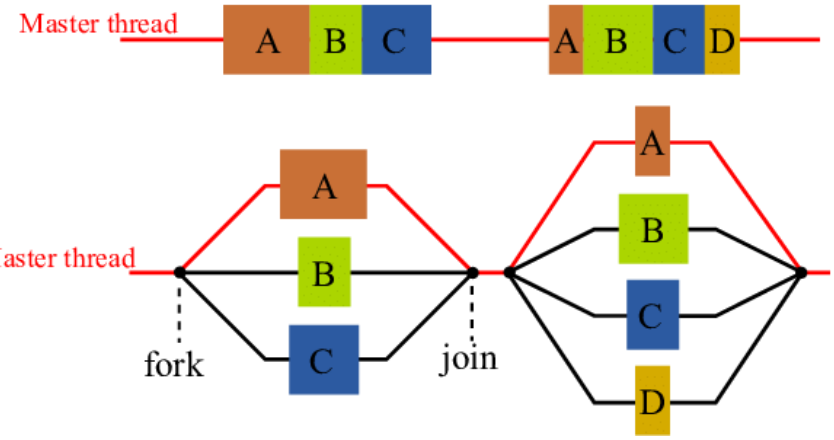
- MPI = Process, OpenMP = Thread
- Program starts with a single process
- Processes have their own (private) memory space
- A process can create one or more threads
- Threads created by a process share its memory space
  - ✓ Read and write to same memory addresses
  - ✓ Share same process ids and file descriptors
- Each thread has a unique instruction counter and stack pointer
  - ✓ A thread can have private storage on the stack

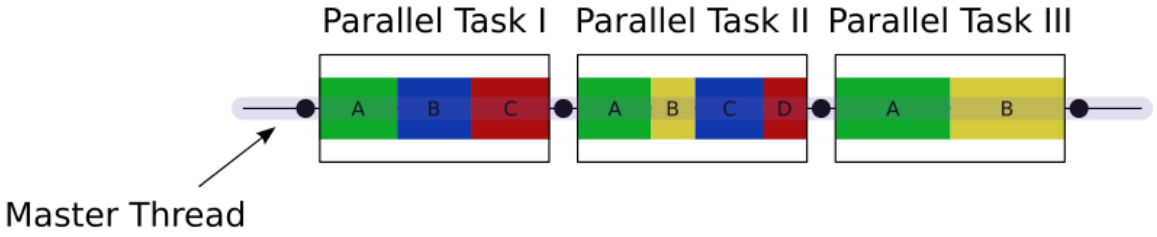
Hyperthreading

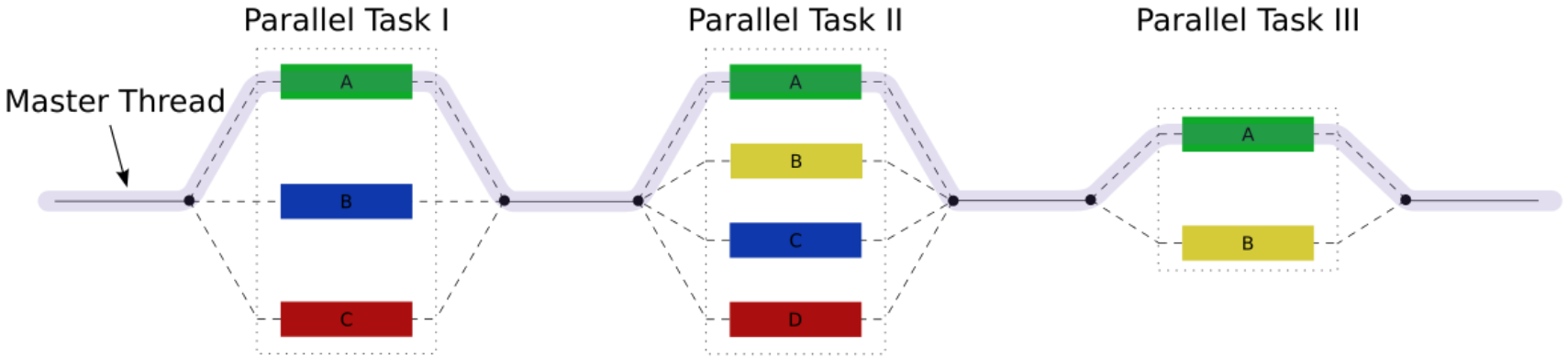
- Hyperthreading is an Intel technology that treats each physical core as two logical cores.
- Two threads are executed at the same time (logically) on the same core.
- Hyperthreading schedules two threads to every core.
- The purpose of hyperthreading is to improve the throughput (processing more per unit time).

OpenMP Fork-Join Model

- Automatically distributes work
- Fork-Join Model







Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

OpenMP: Hello World Example

THE UNIVERSITY OF WESTERN AUSTRALIA

```
#include <omp.h>  //<-- necessary header file for OpenMP API
#include <stdio.h>

int main(int argc, char *argv[]){

  printf("OpenMP running with %d threads\n", omp_get_max_threads());

#pragma omp parallel
  {
    //Code here will be executed by all threads
    printf("Hello World from thread %d\n", omp_get_thread_num());
  }

  return 0;
}
```

Running OpenMP Hello World

```
[vagrant@kaya2]$ module load gcc
[vagrant@kaya2]$ gcc -fopenmp hello_world_omp.c -o hello_world_omp
```

Compiler flag to enable OpenMP
(-fopenmp for gcc)

Environment variable defining max threads

```
[vagrant@kaya2]$ export OMP_NUM_THREADS=4
[vagrant@kaya2]$ ./hello_world_omp
OpenMP running with 4 threads
 Hello World from thread 1
 Hello World from thread 0
 Hello World from thread 2
 Hello World from thread 3
```

- OMP_NUM_THREADS defines run time number of threads can be set in code as well using: omp_set_num_threads()
- OpenMP may try to use all available CPUs if not set (on cluster–always set it!)

Threads Run Independently

```
#pragma omp parallel
  {
    //Code here will be executed by all threads
    printf("Hello World from thread %d\n", omp_get_thread_num());
  }
```

- There is only one thread until the parallel directive is encountered.
- Thread 0 is usually the master thread (that spawns the other threads).
- The parallel region is enclosed in curly brackets.
- There is an implied barrier at the end of the parallel region.

What is a Barrier?

THE UNIVERSITY OF WESTERN AUSTRALIA

- A barrier is a place in the process where all threads must reach before further processing occurs.
- Barriers are sometime implicit.
- Barriers are expensive in terms of run time performance. A typical barrier may take hundreds of clock cycles to ensure that all threads have reached the barrier.
- It is better to remove barriers, but this is fraught with danger.

A Variation of OpenMP Hello World

THE UNIVERSITY OF WESTERN AUSTRALIA

```
#include <omp.h>  //<-- necessary header file for OpenMP API
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]){

  printf("OpenMP running with %d threads\n", omp_get_max_threads());

#pragma omp parallel
  {
    //Code here will be executed by all threads
    if(omp_get_thread_num()==3) sleep(1);
    printf("Hello World from thread %d\n", omp_get_thread_num());
  }

  return 0;
}
```

Outline

- Parallel Programming
  - ✓ Why, What and How
- Parallel Architectures
  - ✓ SISD, SIMD, MIMD
  - ✓ CPU v.s. GPU, Cluster v.s. Multicores
- Shared and Distributed Memory Systems
  - ✓ OpenMP and MPI
- Introduction to OpenMP
  - ✓ What is OpenMP
  - ✓ Hello World Example
  - ✓ Loop Example

The Loop Worksharing Construct

THE UNIVERSITY OF WESTERN AUSTRALIA

- The loop worksharing construct splits up loop iterations among the threads in a team.

```
#pragma omp parallel
  {
    #pragma omp for
        for(i=0; i<N; i++){
           do_something(i);
        }
  }
```

```
#pragma omp parallel
  {
    #pragma omp for
        for(i=0; i<N; i++){
           c[i] = a[i]+b[i];
        }
  }
```

Sequential code

```
for(i=0; i<N; i++){
   c[i] = a[i]+b[i];
}
```

References

- Readings
  - Hyperthreading
  - A Hands-on Introduction to OpenMP

Copyright Notice

Material used in this recording may have been reproduced and communicated to you by or on behalf of **The University of Western Australia** in accordance with section 113P of the *Copyright Act 1968*.

Unless stated otherwise, all teaching and learning materials provided to you by the University are protected under the Copyright Act and is for your personal use only. This material must not be shared or distributed without the permission of the University and the copyright owner/s.