# XJCO3221 Parallel Computation



Peter Jimack

University of Leeds



Lecture 20: Summary

# Previous lectures



This module has been structured to focus on one parallel architecture after another.

1. Shared memory CPU (with OpenMP); Lectures 2-7.
2. Distributed memory CPU (with MPI); Lectures 8-13.
3. General purpose GPU (with OpenCL); Lectures 14-19.





The practical elements of the module *(worked examples, worksheets and courseworks)* also followed this structure.

- Some *mentions* were out-of-order, *e.g.* OpenMP barriers mentioned in Lectures 11 and 17.

# Today's lecture



Want to understand **general parallel programming concepts** that transcend particular architectures.

- Architectures, parallel frameworks *etc.* will change in future.
- General parallel programming concepts will **not**.



In this final lecture we will summarise all of the material by **parallel concept** rather than by architecture, API *etc.*

- Easier to see the commonalities.



At the end I'll also spend a few minutes talking about the **Final assessment** for the module!

# Why parallel?

Lectures 1 and 4



**Parallel hardware** allows **simultaneous** computations.

- Necessary to improve performance as clock speeds limited by physical constraints.
- Subset of **concurrency**, which is 'in the same time frame' but could be *e.g.* time-sharing on a single core.



Want to attain good **scaling** - decrease in parallel computation time $t_p$ for increasing number of **processing units** *(threads, processes* etc.*)* $p$.

# Measuring parallel performance

Lecture 4



Two useful metrics for parallel performance are the **speed-up** $S$ and **efficiency** $E$

$$S = \frac{t_s}{t_p} = \frac{(\text{serial execution time})}{(\text{parallel execution time})} \qquad E = \frac{t_s}{pt_p}$$

Achieving $S = p$ (*i.e.* $E = 1$) usually regarded as ideal, but difficult to achieve due to various **overheads**.

- Synchronisation, load balancing, communication, extra calculations, ...
- **Super-linear scaling** $S > p$ possible (but rare) due to memory cache.

# Laws for maximum parallel performance

Lectures 4, 19



**Strong scaling** is when the problem size $n$ is fixed.

- Covered by **Amdahl's law**: $S \leq \frac{1}{f + \frac{1-f}{p}}$.
- $f \equiv$ fraction of code in serial.



**Weak scaling** allows $n$ to increase with $p$.

- Related to the **Gustafson-Barsis law**: $S \leq p + f(1 - p)$.



The **work-span model** provides another estimate for the maximum $S$ from **task-graphs** *[Lecture 19]*.

- $S \leq (\text{work})/(\text{span})$, with **work** and **span** determined from the task graph.

# Loop parallelism and data dependencies

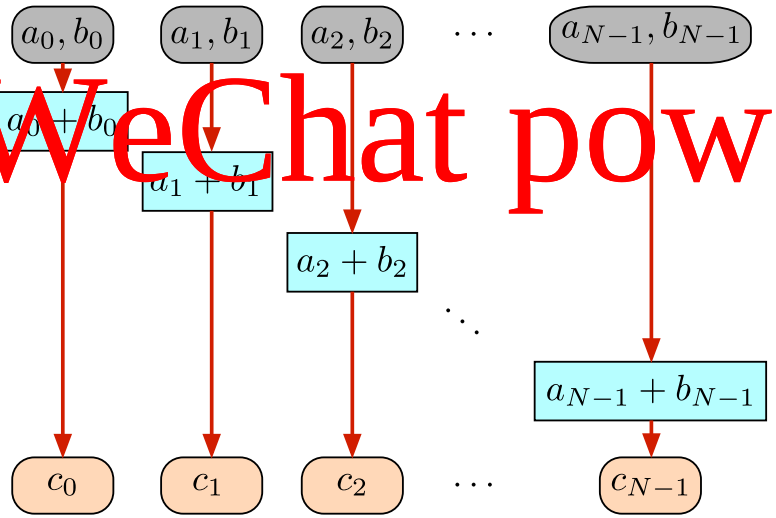
Lectures 3, 5, 9, 15



Initially looked at parallelising **loops**.

- If there are no **data dependencies**, is **data parallel** or a **map**.
- Often referred to as **embarrassingly parallel**.
- Standard example is **vector addition**





# Synchronisation

Lectures 7, 9, 11, 17



All but the simplest parallel problems require **synchronisation** between the processing units at one or more points.

- Often implemented as **barriers** or **blocking communication**.
- For instance, between levels of the binary tree in reduction.



Can lead to reduced **performance**.

- *e.g.* the extra operations required to achieve synchronisation.



Can also lead to **deadlock** *[Lectures 7, 9]*.

- When one or more processing units wait for a synchronisation even that never occurs.

# Load balancing and task parallelism

Lectures 13 and 19



Idle time is an example of poor **load balancing** *[Lecture 13]*.

- Want processing units to never be idle — good load balancing.
- **Serialisation** is the extreme case when only one processing unit is active at a time.

Can improve load balancing by using a **work pool**, where independent **tasks** are sent to processing units as soon as they become idle.

- This is an example of **dynamic** scheduling; can also be **static**.

Parallelising by **tasks** rather than loops is known as **task parallelism** *[Lecture 19]*.

- Increasingly supported by modern parallel frameworks.



In general, there will be **dependencies** between the tasks.

- Can represent as a **task graph**, with **nodes** representing tasks and **directed edges** denoted the dependencies.



- For tasks that take the same time, can define the **work** as the total number of tasks, and the **span** as the length of the critical path.

# Data reorganisation

Lecture 10



Parallel data reorganisation can be indexed by **read** locations ('gather'), or by **write** locations ('scatter').



In shared memory systems need to worry about **collisions**, an example of a **data race** *(see later)*.



In distributed memory systems, can exploit **collective communication methods** that are usually provided.

- One-to-many, many-to-one (also many-to-many).
- *e.g.* broadcasting, scattering and gathering.

# Parallel hardware

Lectures 2, 8, 14, 16



Modern HPC clusters are increasingly using all three architectures:

- **Nodes** with one or more **multi-core CPUs** plus one or more **GPGPU**s.
- Interconnected nodes makes a **distributed system**.



Most multi-core CPUs usually have **memory cache**, with issues of **cache coherency** and **false sharing** *[Lecture 2]*.



**Network connectivity** affects communication times, with **hypercube** often used *[Lecture 8]*.

GPU's most suited for **data parallel problems** and have multiple types of **memory** *[Lectures 14, 16]*.

# Data races / race conditions

Lectures 5, 6, 18



A **data race** potentially arises when two or more processing units read the same memory location, and **at least one writes to it**.

- Shared memory CPU, or global/local memory in a GPU.
- Can lead to **non-deterministic** behaviour.

Can control using **critical regions**.

- Exclusive access by a single processing unit.
- Simple critical regions can be implemented more efficiently (*i.e.* by compiler and hardware) as **atomics**.

# Lower level control

Lectures 7, 18



At a lower level, critical regions are controlled by **locks** or **mutexes**.



- **Multiple** locks can improve access to data structures.
- Improper use of multiple locks can result in **deadlock**.



At an even lower level, locks can be implemented using **atomic exchange** and **atomic compare-and-exchange** *[Lecture 18]*.

- **Lock-free data structures** are desirable whenever possible.

# Explicit communication

Lectures 9, 10, 12, 15, 19



If memory is distributed (in some sense), may need to use **explicit communication**.



- Can be **point-to-point** between processes.
- Also one-to-many *etc.*, *i.e.* **collective communication**.
- Between CPU and GPU, *i.e.* **host** and **device**.



Communication can be:

- **Blocking**: Returns once all resources safe to re-use.
- **Synchronous**: Does not complete until sender and receiver start their communication operations.

# Latency hiding

Lectures 12, 19



Can improve performance by **overlapping** communication with computation:

- Reduces the **communication overhead**.
- Known as **latency hiding**.



Often used with **domain partitioning** in HPC applications *[Lecture 12]*.



Can also overlap host-device communication with computation on a GPU *[Lecture 19]*.

- Can also perform calculations on host and device simultaneously.

# The end



This is the end of the material for XJCO3221 Parallel Computation.





I will now say a few words about the final assessment/exam for this module...

# The final assessment



This will be an online exam to be taken at 1600 (China time) on Tuesday 17th May 2022...

- It will be available via the "Final assessment" tab in Minerva from 1600 on 17th May
  - You will be able to download the exam paper as a PDF
  - You must upload your solutions as a single PDF file by 1800 on 17th May (unless you have been granted additional time)
- It is an "open book" exam but you will receive no credit for copying material directly from your lecture notes!
- Your overall module grade will be the sum of your scores on the 3 courseworks and your score (out of 50) on this final assessment
  - Hence the final assessment is worth 50% of the total marks

# The final assessment (continued)



- The paper will consist of two questions worth 25 marks each
  - So you should aim to spend about an hour on each
- To help you prepare I have provided past papers from the previous 4 years on Minerva
  - The 2020 and 2021 papers were both "open book" but the students were given more time to complete them – so the questions are longer than they will be this year
  - The 2018 and 2019 papers were both 2 hour exams however they were "closed book" – so some sections are pure "recall" questions
- See the announcement on Minerva for some further advice

# Your revision questions



- The final Zoom session will be held at 1730 on Tuesday 10th May
  - I will be available then to answer any questions that you may have on any aspect of the module
  - I advise you to attend so that you can hear other people's questions (and my responses)
  - There will also be a Zoom session at 1730 today (3rd May) if you already have any questions for me!
- I will also answer any questions that you post on any of the XJCO3221 Discussion Boards on Minerva prior to 1600 on 17th May
  - I will NOT answer any questions after the start of the final assessment!