CS3210 - Parallel Computing
AY25/26, Y3S1
Notes

Sim Ray En Ryan

December 3, 2025

Contents

1 Lecture 1: Introduction

1.1 Motivation for Parallel Computing

- **The Power Density Wall:** Around 2004, single-processor performance hit a physical limit due to heat generation and power consumption. Increasing clock frequency was no longer viable.
- **The Shift:** The industry shifted from increasing single-core clock speeds to placing multiple cores on a single chip (Multicore).
- **Goal:** To solve complex problems (e.g., climate modelling, simulations, AI) faster or to solve larger problems.

1.2 Flynn's Taxonomy

A classification of computer architectures based on the number of instruction streams and data streams:

- **SISD (Single Instruction, Single Data):** Traditional uniprocessor.
- **SIMD (Single Instruction, Multiple Data):** Vector processors, GPUs. Same instruction applied to different data elements simultaneously.
- **MISD (Multiple Instruction, Single Data):** Rare (e.g., systolic arrays).
- **MIMD (Multiple Instruction, Multiple Data):** Modern multicore processors, clusters. Independent processors executing different instructions on different data.

2 Lecture 2: Processes, Threads, and Synchronization

2.1 Program Parallelization Steps

1. **Decomposition:** Breaking the problem into tasks.
2. **Scheduling:** Assigning tasks to processes or threads.
3. **Mapping:** Assigning processes/threads to physical processing units (cores).

2.2 Processes vs. Threads

2.2.1 Processes

- An instance of a program in execution with its own address space (Text, Data, Heap, Stack).

- Managed by the OS. Identified by PID.
- **Pros:** Memory isolation (safety).
- **Cons:** High overhead for creation and context switching; difficult communication (Inter-Process Communication - IPC required).
- **Unix:** Created via ‘fork()‘.

2.2.2 Threads

- Independent control flows within a single process.
- Share the process's address space (Code, Data, Files) but have private registers and stacks.
- **Pros:** Fast creation and switching; efficient communication via shared memory.
- **Cons:** Lack of protection (one crashing thread can crash the process); requires synchronization.

2.3 User-Level vs. Kernel-Level Threads

- **User-Level:** Managed by a library in user space. OS is unaware. Fast switching but blocking I/O blocks the whole process.
- **Kernel-Level:** Managed by the OS. OS can schedule threads on different cores. Slower switching due to syscalls.
- **Mapping Models:** Many-to-One, One-to-One (common in Linux/Windows), Many-to-Many.

2.4 Synchronization

2.4.1 Race Conditions and Critical Sections

- **Race Condition:** Outcome depends on the non-deterministic ordering of thread execution.
- **Critical Section:** A segment of code accessing shared resources that must not be concurrently executed by more than one thread.
- **Mutual Exclusion:** Mechanism to ensure only one thread enters a critical section.

2.4.2 Mechanisms

- **Locks/Mutexes:** ‘acquire()‘ and ‘release()‘. Can be spinlocks (busy-wait) or blocking.

- **Semaphores:** Counters used for signaling and resource tracking.
- **Barriers:** Synchronization point where threads wait until all have arrived.

2.4.3 Deadlock

Occurs when four conditions hold simultaneously:

1. **Mutual Exclusion:** Resources cannot be shared.
2. **Hold and Wait:** Process holds a resource while waiting for another.
3. **No Preemption:** Resources cannot be forcibly taken.
4. **Circular Wait:** A cycle of dependencies exists.

3 Lecture 3: Architecture and Memory

3.1 Forms of Parallelism

- **Bit-level:** Increasing word size (e.g., 32-bit to 64-bit).
- **Instruction-level (ILP):** Pipelining (overlapping execution stages) and Superscalar (multiple execution units).
- **Thread-level (TLB):** Simultaneous Multithreading (SMT/Hyper-threading) and Multicore.

3.2 Memory Organization

3.2.1 Shared Memory Systems

- **UMA (Uniform Memory Access):** Access time to memory is consistent across all processors (e.g., SMP).
- **NUMA (Non-Uniform Memory Access):** Memory is physically distributed but logically shared. Accessing local memory is faster than remote memory.
- **COMA (Cache-Only Memory Access):** Memory acts as a large cache.

3.2.2 Distributed Memory Systems

- Each processor has private memory.
- Communication occurs via message passing over a network.
- Highly scalable but requires explicit data distribution and communication management.

4 Lecture 4: Parallel Programming Models I

4.1 Data vs. Task Parallelism

- **Data Parallelism:** Partitioning data among processors; each performs the same operation on different data chunks (e.g., array addition).
- **Task Parallelism:** Partitioning distinct functions or tasks among processors (e.g., one thread does UI, another does networking).

4.2 Foster's Design Methodology

1. **Partitioning:** Divide computation and data into small tasks (Domain vs. Functional decomposition).
2. **Communication:** Determine communication needs (Local vs. Global).
3. **Agglomeration:** Combine tasks to reduce overhead and improve granularity.
4. **Mapping:** Assign tasks to physical processors to balance load and minimize communication.

4.3 Programming Patterns

- **Master-Worker:** One master coordinates work distribution; workers execute and return results.
- **Fork-Join:** Main thread forks child threads for parallel work, then joins them.
- **SPMD (Single Program Multiple Data):** All threads run the same program code (maybe different paths) on different data.
- **Pipelining:** Data flows through a sequence of stages (stream parallelism).
- **Task Pool:** Threads retrieve tasks from a shared queue.

5 Lecture 5: Performance

5.1 Metrics

- **Speedup (S_p):** Ratio of sequential execution time to parallel execution time.

$$S_p = \frac{T_{seq}}{T_p}$$

- **Efficiency (E_p):** Fraction of time processors are being used effectively.

$$E_p = \frac{S_p}{p}$$

- **Cost:** $p \times T_p$.

5.2 Scalability Laws

5.2.1 Amdahl's Law (Fixed Workload)

Pessimistic view. Speedup is limited by the sequential fraction (f) of the code.

$$S_{max} \leq \frac{1}{f + \frac{1-f}{p}}$$

As $p \to \infty$, speedup converges to $1/f$.

5.2.2 Gustafson's Law (Scaled Workload)

Optimistic view. As problem size increases, the parallel portion grows while the sequential portion remains constant.

$$S_p = p - f(p-1)$$

5.3 Arithmetic Intensity

Ratio of computation (FLOPs) to communication (Bytes). High arithmetic intensity is required to mask memory latency and utilize compute power.

6 Lecture 6: GPGPU Programming

6.1 GPU Architecture

- Optimized for **throughput** rather than latency.
- Massive number of simple cores processing parallel data streams.
- **SIMT (Single Instruction Multiple Threads):** Threads execute in groups called **Warps** (typically 32 threads).

6.2 CUDA Programming Model

- **Kernel:** Function executing on the GPU, defined with '__global__'.
- **Hierarchy:**
  - **Grid:** Collection of Blocks.
  - **Block:** Collection of Threads (mapped to a Streaming Multiprocessor - SM). Threads in a block can synchronize and share memory.
  - **Thread:** Fundamental unit of execution.

- **Warp Divergence:** If threads in a warp take different paths (if-else), execution is serialized, reducing performance.

6.3 Memory Hierarchy

- **Registers:** Fastest, per-thread.
- **Shared Memory:** Fast, user-managed cache per block. Vital for optimization.
- **Global Memory:** Large, slow, accessible by all. Requires **Coalesced Access** (adjacent threads accessing adjacent memory) for performance.
- **Constant/Texture Memory:** Read-only, cached specialized memories.

7 Lecture 7: Cache Coherence and Memory Consistency

7.1 Cache Coherence

Ensures that changes in the values of shared operands propagate throughout the system in a timely fashion. Addresses the problem of multiple cached copies of the same data.

- **Properties:**
    1. **Program Order:** A processor sees its own writes.
    2. **Write Propagation:** Writes eventually become visible to others.
    3. **Transaction Serialization:** All processors see writes to the *same* location in the same order.
- **Protocols:** Snooping (bus-based) vs. Directory-based.
- **False Sharing:** Coherence traffic generated when different processors write to different variables that reside on the same cache line. Solution: Padding.

7.2 Memory Consistency

Defines the order in which memory operations (to *different* locations) appear to execute.

- **Sequential Consistency (SC):** Operations appear to execute in a strict global serial order consistent with program order. Easy to reason, hard to implement efficiently.
- **Relaxed Consistency (TSO, PSO, Weak):** Allows reordering of reads/writes (e.g., Write-after-Read) to hide latency. Requires explicit synchronization (fences/barriers) for correctness.

8 Lecture 8: Parallel Programming Models II

8.1 Shared vs. Distributed Memory Models

- **Shared Memory:** Implicit communication via variables. Focus on synchronization.
- **Distributed Memory:** Explicit communication via messages. No shared address space. Focus on data distribution.

8.2 Data Distribution Strategies

- **Blockwise:** Contiguous chunks assigned to processors. Good for spatial locality.
- **Cyclic:** Round-robin assignment. Good for load balancing.
- **Block-Cyclic:** Chunks assigned in round-robin.

8.3 Communication Semantics

- **Blocking:** Call waits until resources (buffer) can be reused. Safe but may idle.
- **Non-Blocking:** Returns immediately. Requires a separate 'wait' or 'test' to ensure completion. Allows overlapping computation and communication.
- **Buffered:** Data copied to system buffer. Decouples sender/receiver but adds copy overhead.
- **Synchronous:** Handshake required. Sender waits for receiver to start.
- **Asynchronous:** "Fire and forget". Sender proceeds immediately.

9 Lecture 9: MPI (Message Passing Interface)

9.1 Overview

Standard for distributed memory programming.

- **SPMD:** Usually runs the same executable on all nodes.
- **Communicator:** Defines a group of processes (e.g., 'MPI_COMM_WORLD').
- **Rank:** Unique ID of a process within a communicator.

9.2 Operations

- **Point-to-Point:** 'MPI_Send', 'MPI_Recv'.
- **Collective Communication:** Involves all processes in a communicator.

  - ‘MPI_Bcast‘: One-to-all broadcast.
  - ‘MPI_Scatter‘: One-to-all distribution of different data.
  - ‘MPI_Gather‘: All-to-one collection.
  - ‘MPI_Reduce‘: All-to-one reduction (sum, max, etc.).
  - ‘MPI_Alltoall‘: Total exchange (transpose).
- **Barrier:** Synchronization point (‘MPI_Barrier‘).

10 Lecture 10: Interconnection Networks

10.1 Topologies

How nodes are connected. Can be Direct (static) or Indirect (dynamic/switched).

- **Linear Array / Ring:** Simple, low degree. High diameter.
- **Mesh / Torus:** Good for grid-based problems. Torus wraps around edges.
- **Hypercube:** Low diameter ($\log N$), high degree ($\log N$). Good scalability.
- **Fat Tree:** Indirect. Bandwidth increases towards the root to prevent bottlenecks. Common in clusters.
- **Dragonfly:** Hierarchical (groups of routers). Low diameter, high path diversity.

10.2 Metrics

- **Diameter:** Max distance between any two nodes (Latency).
- **Degree:** Number of links per node (Hardware cost).
- **Bisection Width:** Min wires cut to split network into two equal halves (Bandwidth bottleneck).

10.3 Routing

Mechanism to determine path from source to destination.

- **Deterministic:** Path determined solely by source and destination (e.g., XY routing, E-Cube). Prone to congestion.
- **Adaptive:** Path depends on network traffic.
- **Deadlock freedom:** Routing algorithms must avoid cycles in resource dependencies.

11 Lecture 11: Energy-Efficient Computing

11.1 Motivation

- **High Power Consumption:** Modern computing infrastructure consumes vast amounts of power. Datacenters account for over 2% of the world's energy supply.
- **Grid Constraints:** High energy demand puts strain on national power grids (e.g., issues in Ireland).
- **Goals:**
  - **Usability:** Extend battery life and reduce heat in mobile/consumer devices.
  - **Cost and Impact:** Reduce electricity costs and environmental impact for datacenters and HPC systems.

11.2 Definitions: Energy vs. Power

11.2.1 Energy (E)

- **Unit:** Joules (J).
- Represents the capacity for doing work.
- Different operations have different energy costs. Generally, data movement (communication) is significantly more expensive than arithmetic operations.

11.2.2 Power (P)

- **Unit:** Watts (W) or Joules per second (J/s).
- Represents the rate at which energy is transferred or converted ($P = E/t$).
- High power leads to high operational costs and heat generation, which physically limits processor performance (Thermal Design Power).

11.3 Per-Processor Efficiency

11.3.1 Trends and Limits

- **End of Dennard Scaling:** Around 2005, it became impossible to continue increasing transistor density without significantly increasing power density.
- **Power Wall:** Processors hit a limit on how much heat could be dissipated (active cooling required). This led to the stagnation of clock frequencies and the rise of multicore architectures.

11.3.2 Power Formula

The dynamic power consumed by a processor is modeled as:

$$P_{total} = P_{dynamic} + P_{static}$$

$$P_{dynamic} \approx k \times V^2 \times f$$

Where:

- k: Constant depending on hardware/software complexity.
- V: Voltage. Has a squared relationship with power (most significant factor).
- f: Frequency. Linear relationship with power.

11.3.3 Optimization Techniques

- **DVFS (Dynamic Voltage and Frequency Scaling):** Processors dynamically adjust voltage and frequency based on workload. Lowering frequency allows for lower voltage, resulting in cubic power savings.
- **Heterogeneous Cores:** combining different types of cores to handle different workloads efficiently.
  - **ARM big.LITTLE:** Combines high-performance cores (Cortex-A15) with high-efficiency cores (Cortex-A7).
  - **Intel Hybrid Technology:** Uses P-cores (Performance) and E-cores (Efficiency).
  - **Apple Silicon:** Uses performance and efficiency cores with shared memory architecture.

11.4 Datacenter and HPC Efficiency

11.4.1 Metrics

- **Performance-per-Watt (GFLOPs/Watt):** Measures computation efficiency. Tracked by the **Green500** list.
- **Power Usage Effectiveness (PUE):** Measures infrastructure efficiency.

$$PUE = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}} = 1 + \frac{\text{Non-IT Facility Energy}}{\text{IT Equipment Energy}}$$

  - Ideal PUE is 1.0.
  - Accounts for overheads like cooling, lighting, and power distribution.
  - PUE has improved over time (e.g., Google datacenters moved from $\sim$1.2 to $\sim$1.09).

11.4.2 Hardware Accelerators

- Using specialized hardware improves GFLOPs/Watt.
- **Current Trend:** Integrated CPU+GPU superchips (e.g., NVIDIA Grace Hopper, AMD MI300A) share unified memory, reducing data transfer overheads over PCIe.

11.4.3 Cooling Techniques

Cooling is a major component of non-IT energy overhead.

- **Aisle Containment:** Separating Hot Aisles and Cold Aisles to prevent mixing of air, improving cooling efficiency.
- **Liquid Cooling:** Direct-to-chip liquid cooling or warm-water cooling (used in Singapore's NSCC) is more efficient than air cooling because water has higher heat capacity.