Introduction

Parallel Computing – COSC 3P93

Course Outline

- **Introduction**
- **Parallel Hardware**
- **Parallel Software**
- **Performance**
- **Parallel Programming Models**
- **Patterns**
- **SMP with Threads and OpenMP**
- **Distributed-Memory Programming with MPI**
- **Algorithms**
- **Tools**
- **Parallel Program Design**

Previous Class

- Introduction
  - Parallel computing
  - Parallel systems
- Importance
- Overview
  - Complexity
  - Task and data parallelism

Today Class

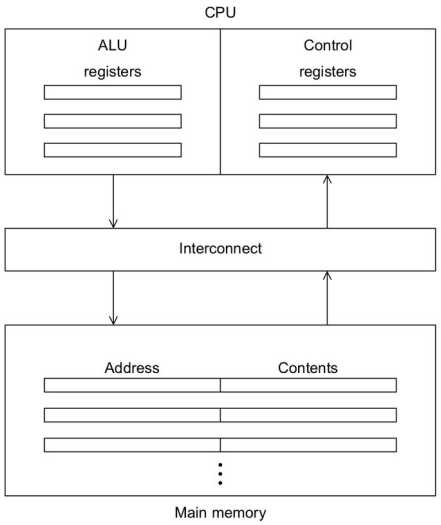
- Von Neumann Architecture
- Flynn's Taxonomy
  - SISD
  - SIMD
  - MISD
  - MIMD
- General terminology and assumptions

von Neumann Architecture

- Named after the Hungarian mathematician/genius John von Neumann
  - First authored the general requirements for an electronic computer in his 1945 papers
- Also known as "**stored-program computer**"
  - Both program instructions and data are kept in electronic memory
  - Differs from earlier computers which were programmed through "hard wiring"

von Neumann Architecture

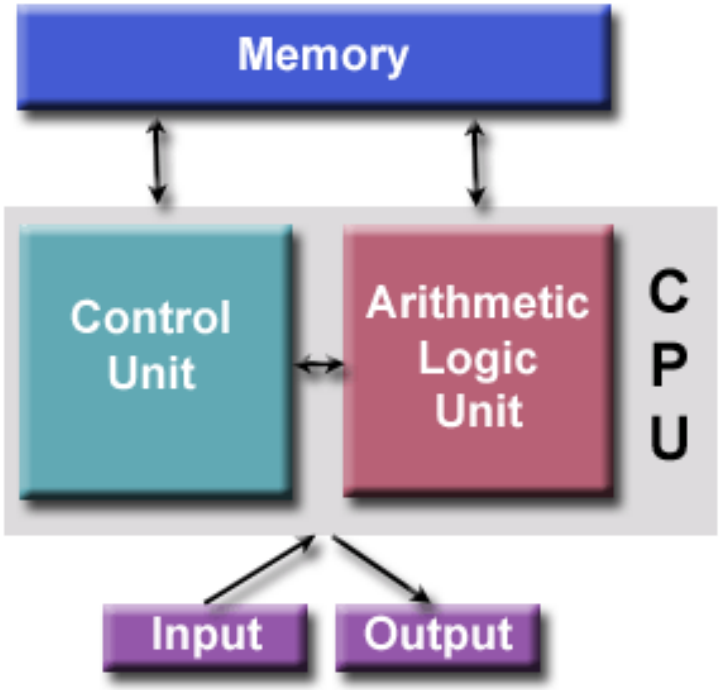
- "**Classic**" architecture
  - Main memory
  - Central processing unit (CPU) → processor, core
    - Control Unit and Arithmetic and Logic Unit (ALU)
    - Registers, Program Counter
  - Interconnection → CPU and memory
    - Bus → parallel wires
      - von Neumann bottleneck
- Execution
  - One instruction at a time
    - Fetching or reading from and writing or storing into memory



von Neumann Architecture

- Since then, virtually all computers have followed this basic design
  - **Four main components**
    - Memory
    - Control Unit
    - Arithmetic Logic Unit
    - Input/Output
  - **Read/write**, **random access memory** is used to store both program instructions and data
    - Program instructions are coded data which tell the computer to do something
    - Data is simply information to be used by the program
  - Control unit fetches instructions/data from memory, decodes the instructions and then sequentially coordinates operations to accomplish the programmed task.
  - Arithmetic Unit performs basic arithmetic operations
  - Input/Output is the interface to the human operator

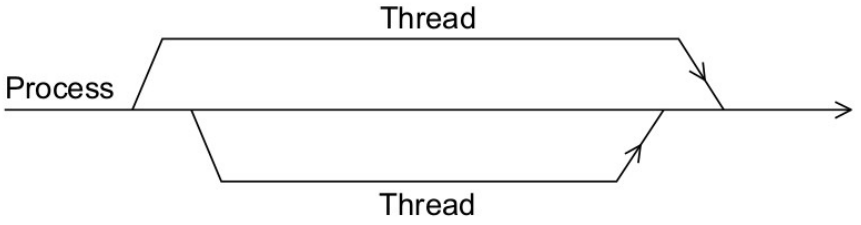
von Neumann Architecture



- Why is this architecture important?
  - Parallel computers still follow this basic design, just multiplied in units
  - The basic, fundamental architecture remains the same

Processes, Multitasking, and Threads

- **Process** → instance of a computer program
  - Block of memory
    - Executable code
    - Call stack → active functions
    - Heap → memory locations
    - Descriptors → I/O
    - Security data
    - Execution state
- **Multitasking**
  - Multiprogramming and time slicing (time sharing)
    - Maximizing CPU utilization (blocking calls) and fairness
- **Threading** → splitting programs in smaller parts → lighter-weight instruction sets
  - Thread → forking off the process or joining the process



Modifications to the von Neumann Model

- Addressing performance issues
  - von Neumann bottleneck
- **Caching** → principle of locality
  - Faster access → CPU cache (same or separate chip)
  - Limited size (smaller) → management
    - Keeping it consistent and fresh
      - Policies → LRU
  - Locality (spacial and temporal) → example

```
float z[1000];
. . .
sum = 0.0;
for (i = 0; i < 1000; i++)
    sum += z[i];
```

    - Access to a “wider” set
      - Blocks of data → cache blocks or cache lines
  - Levels: L1, L2
  - Fetching → Cache hit, cache miss
  - Inconsistency → write-through or write-back (dirty entries)

Cache Mapping

- Where to store cache lines/blocks
- A variety of strategies → spectrum
  - Fully associative → any location in cache
  - Direct mapped → unique location in cache for the line
  - In between these two → n-way set associative
    - N designate locations in cache
- Eviction
  - Replacement policies
    - LRU

Assignments of a 16-line Main Memory to a 4-line Cache

	Cache Location		
Memory Index	*Fully Assoc*	*Direct Mapped*	*2-way*
0	0, 1, 2, or 3	0	0 or 1
1	0, 1, 2, or 3	1	2 or 3
2	0, 1, 2, or 3	2	0 or 1
3	0, 1, 2, or 3	3	2 or 3
4	0, 1, 2, or 3	0	0 or 1
5	0, 1, 2, or 3	1	2 or 3
6	0, 1, 2, or 3	2	0 or 1
7	0, 1, 2, or 3	3	2 or 3
8	0, 1, 2, or 3	0	0 or 1
9	0, 1, 2, or 3	1	2 or 3
10	0, 1, 2, or 3	2	0 or 1
11	0, 1, 2, or 3	3	2 or 3
12	0, 1, 2, or 3	0	0 or 1
13	0, 1, 2, or 3	1	2 or 3
14	0, 1, 2, or 3	2	0 or 1
15	0, 1, 2, or 3	3	2 or 3

Caches and Programs

- Cache controlled by hardware
  - No direct control from user space
- **Indirect** control → spacial and temporal locality
  - Example
    - Which pair of loops (in C) runs faster?

```
double A [ MAX ][ MAX ], x [ MAX ], y [ MAX ];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for ( i = 0; i < MAX ; i ++)
    for ( j = 0; j < MAX ; j ++)
        y [ i ] += A [ i ][ j ]* x [ j ];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for ( j = 0; j < MAX ; j ++)
    for ( i = 0; i < MAX ; i ++)
        y [ i ] += A [ i ][ j ]* x [ j ];
```

    - C language stores 2D arrays in row-major order
      - 2D into a 1D → row 0 1st and then row 1

Virtual Memory

- Extending capacity of main memory
  - A single to large program
  - Multiple programs
- A must for multiprogramming
- Main memory works as a cache → locality
  - Programs (or part) must be in the main memory to run
- Secondary memory
  - Idle/inactive program parts store in swap space of 2nd mem
    - Storage unity → page
  - Virtual addressing → page table
    - Translation Lookaside Buffer → register for speeding up lookup
  - Page hit, page fault

Instruction-level Parallelism (ILP)

- Improvement of processor performance
  - Functional units executing instructions simultaneously
- Approaches
  - **Pipelining** → arranging units in stages
  - **Multiple issue** → instructions being simultaneously initiated
- Both used in modern CPUs

Pipelining

Time	Operation	Operand 1	Operand 2	Result
0	Fetch operands	9.87×10^4	6.54×10^3	
1	Compare exponents	9.87×10^4	6.54×10^3	
2	Shift one operand	9.87×10^4	0.654×10^4	
3	Add	9.87×10^4	0.654×10^4	10.524×10^4
4	Normalize result	9.87×10^4	0.654×10^4	1.0524×10^5
5	Round result	9.87×10^4	0.654×10^4	1.05×10^5
6	Store result	9.87×10^4	0.654×10^4	1.05×10^5

Time	Fetch	Compare	Shift	Add	Normalize	Round	Store
0	0						
1	1	0					
2	2	1	0				
3	3	2	1	0			
4	4	3	2	1	0		
5	5	4	3	2	1	0	
6	6	5	4	3	2	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
999	999	998	997	996	995	994	993
1000		999	998	997	996	995	994
1001			999	998	997	996	995
1002				999	998	997	996
1003					999	998	997
1004						999	998
1005							999

- **Hardware-level improvement**
- Analogy to factory assembly line
  - No idle function units
- Example
  - Floating point instructions (broken in 7 operations)
    - 1 nanosec each operation

```
float x [1000], y [1000], z [1000];
. . .
for ( i = 0; i < 1000; i ++)
    z [ i ] = x [ i ] + y [ i ];
```

      - Takes 7000 nanosec
  - With pipeline → 7000 ns to 1006 ns
- However → k stages will not give a k-fold improvement
  - Slowest function unit dictates speed
  - Delays involved → dependencies between units

Multiple Issue

- **Replication** of functional units → simultaneous execution
- Example
  - Two complete floating point adders → halve the time

```
for ( i = 0; i < 1000; i ++)
    z [ i ] = x [ i ] + y [ i ];
```

    - adder 1 – z[0], adder 2 – z[1], adder 1 – z[2], ...
- Scheduling of functional units
  - Compile time → **static** multiple issue
  - Run-time → **dynamic** multiple issue
    - Processor supporting dynamic MI → **superscalar**
- Benefit of Multiple issue
  - Instructions that can be executed simultaneously
- **Speculation** → guessing (most important techniques)
  - Example 1 → outcome of addition is positive
    - It processes 1st block, speculatively
  - Example 2 → no instruction dependency

```
z = x + y;
if (z > 0)
   w = x;
else
   w = y;
```

```
z = x + y;
w = *a_p;  /* a_p is a pointer */
```

Hardware Multithreading

- **Hard to achieve instruction-level parallelism**
  - Thus → multithreading
    - Thread-level Parallelism (TLP)
      - Coarser-grained than ILP
- Hardware **multithreading** → minimizing context switching
  - Fine-grained → switching after every instruction
    - Too many stalls of long threads
  - Coarse-grained → threads waiting for time-consuming operations
    - More costly switching → more delays
    - Idling processor
- **Simultaneous multithreading** (SMT)
  - Fine-grained variation
  - Exploiting superscalar CPUs
    - Multiple threads using multiple functional units at the same time

Multithreading

- Popular
- Widely supported
  - Hardware
  - OS
  - Languages/compilers
-
-
-
-

Java
- /basicthreads
- /executors
- java WordCounter folder house 8

C++
- C++/simpleExamples
- C++/hellos
- C++/image_processing

Some General Parallel Terminology

- Like everything else, parallel computing has its own **jargon**
  - Most of these will be discussed in more detail later
  - **Supercomputing / High Performance Computing**
    - Using the world's fastest and largest computers to solve large problems.
  - **Node**
    - A standalone "computer in a box"
      - Multiple CPUs/processors/cores, memory, NICs
      - Nodes are networked together to build a supercomputer

Some General Parallel Terminology

- Like everything else, parallel computing has its own **jargon**
  - **CPU / Socket / Processor / Core**
    - In the past → a CPU (Central Processing Unit) was a singular execution component for a computer
      - Multiple CPUs were incorporated into a node
      - Individual CPUs were subdivided into multiple cores
      - CPUs with multiple cores are sometimes called sockets
      - The result is a node with multiple CPUs
        - Each containing multiple cores

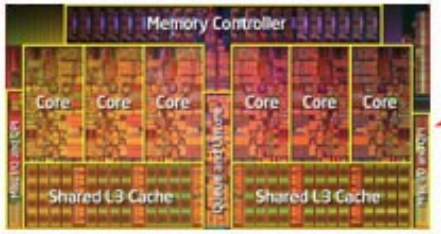
Some General Parallel Terminology

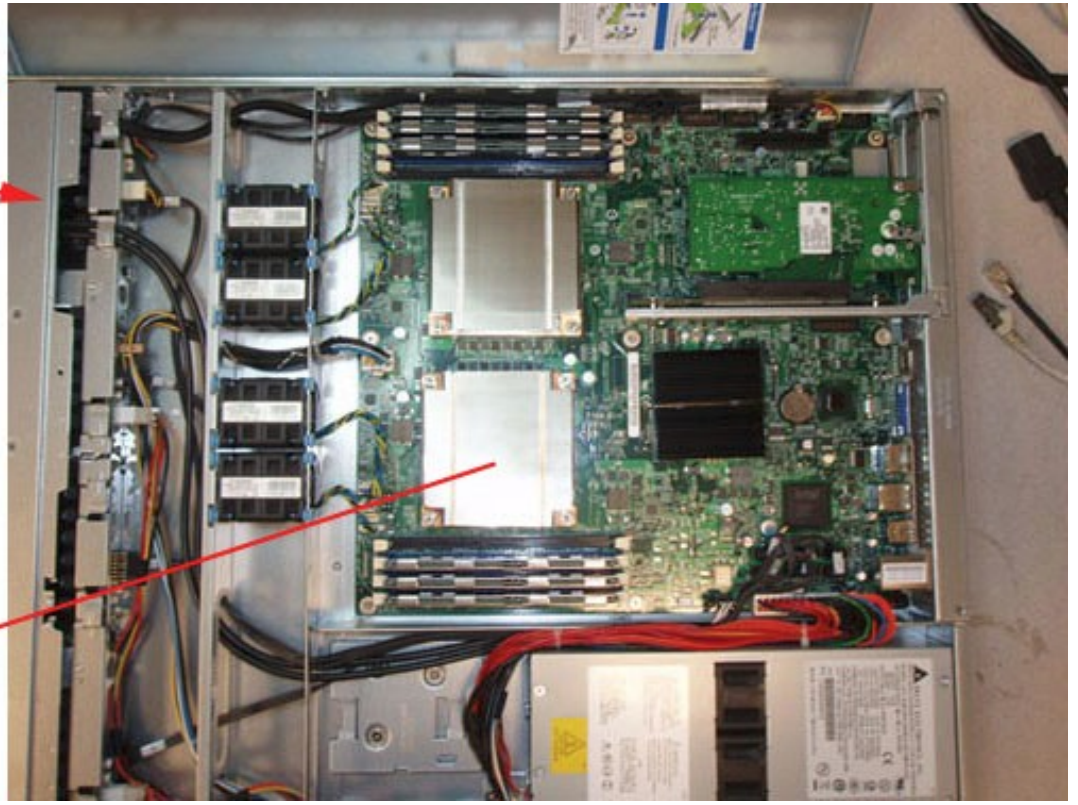
- **CPU / Socket / Processor / Core**

Supercomputer - each blue light is a node

Node - standalone Von Neumann computer

CPU / Processor / Socket - each has multiple cores / processors.





Some General Parallel Terminology

- **Task**
  - A logically discrete section of computational work
  - Typically a program or program-like set of instructions that is executed by a processor
  - A parallel program consists of multiple tasks running on multiple processors
- **Pipelining**
  - Breaking a task into steps performed by different processor units
    - With inputs streaming through
    - Much like an assembly line
    - A type of parallel computing

Some General Parallel Terminology

- **Shared Memory**
  - From a strictly hardware point of view
    - A computer architecture where all processors have direct (usually bus-based) access to common physical memory
  - In a programming sense
    - A model where parallel tasks all have the same "picture" of memory
    - They can directly address and access the same logical memory locations
      - Regardless of where the physical memory actually exists

Some General Parallel Terminology

- **Symmetric Multi-Processor (SMP)**
  - Shared memory hardware architecture
    - Multiple processors share a single address space and have equal access to all resources
- **Distributed Memory**
  - In hardware
    - Network-based memory access for physical memory that is not common
  - As a programming model
    - Tasks can only logically "see" local machine memory
    - They must use communications to access memory on other machines where other tasks are executing

Some General Parallel Terminology

- **Communications**
  - Parallel tasks typically need to exchange data
  - Several ways data can be exchanged
    - Through a shared memory bus or over a network
  - However
    - The actual event of data exchange is commonly referred to as communications regardless of the method employed

Some General Parallel Terminology

- **Synchronization**
  - Coordination of parallel tasks in real time
    - Very often associated with communications
  - Often implemented by establishing a synchronization point within an application
    - A task may not proceed further until another task(s) reaches the same or logically equivalent point
  - It usually involves waiting by at least one task
    - It can thus cause a parallel application's wall clock execution time to increase → add delay to the execution
- **Granularity**
  - In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
    - **Coarse**: relatively large amounts of computational work are done between communication events
    - **Fine**: relatively small amounts of computational work are done between communication events

Some General Parallel Terminology

- **Observed Speedup**
  - Observed speedup of a code which has been parallelized

$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$

  - One of the simplest and most widely used indicators for a parallel program's performance
- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work
  - Parallel overhead include
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel languages, libraries, operating system, etc.
    - Task termination time

Some General Parallel Terminology

- **Massively Parallel**
  - Refers to the hardware that comprises a given parallel system - having many processing elements
  - The meaning of "many" keeps increasing
    - Currently
      - The largest parallel computers are comprised of processing elements numbering in the hundreds of thousands to millions
- **Embarrassingly Parallel**
  - Solving many similar, but independent tasks simultaneously
  - Little to no need for coordination between the tasks
  - Speedup follows the number of processing entities

Some General Parallel Terminology

- **Scalability**
  - Parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more resources
  - Factors that contribute to scalability
    - Hardware - particularly memory-cpu bandwidths and network communication properties
    - Application algorithm
    - Parallel overhead related
    - Characteristics of your specific application

Class Recap

- Von Neumann Architecture
  - Modifications
    - Caching
    - Virtual memory
    - Multithreading
- General terminology and assumptions

Next Class

- Von Neumann Architecture
- Flynn's Taxonomy
  - SISD
  - SIMD
  - MISD
  - MIMD
- General terminology and assumptions