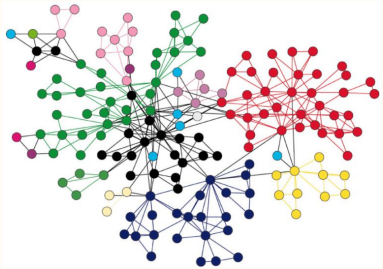
Parallel Processing

Lecture 1
February 5, 2025

What is this course about?

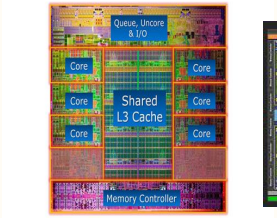


New Workload[1]

Analyze Resource Demands

Map To Existing Architectures

MP[2]

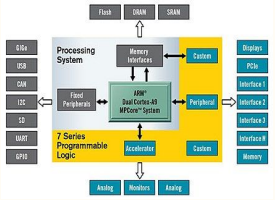


GPU[3]

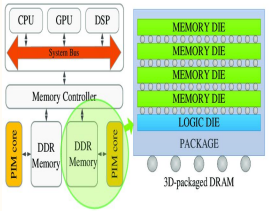
TPU[8]

Supercomputer[6]

Data Center[7]



FPGA[4]



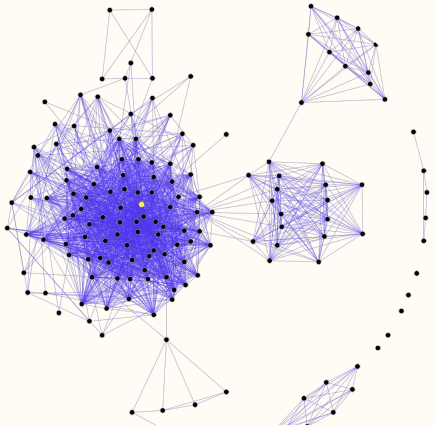
PIM[5]

[1] https://www.javatpoint.com/graph-algorithms
[2] Intel Core i7-3960X
[3] NVIDIA Turing GPU Architecture White Paper
[4] https://en.wikipedia.org/wiki/Field-programmable_gate_array
[5] https://www.researchgate.net/figure/Processing-in-Memory-PIM-concept_fig2_330116237
[6] https://en.wikipedia.org/wiki/Supercomputer
[7] https://www.datacenterknowledge.com/hyperscalers/a-glimpse-inside-google-s-data-centers
[8] https://www.anandtech.com/show/10340/googles-tensor-processing-unit-what-we-know

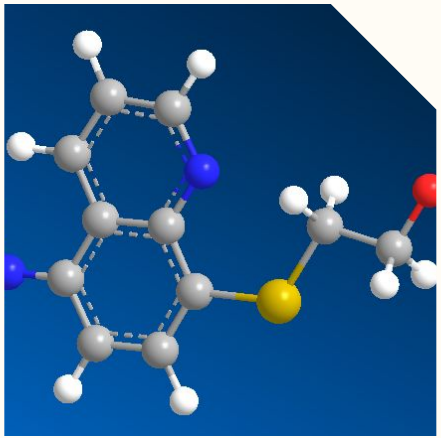
What algorithm characteristics impact its interaction with hardware?

- Degree / type of parallelism
- Communication frequency
- Computational intensity
- Memory intensity
- Branch frequency

Why Graph Algorithms? Graphs Are Everywhere



Social Network Graph[1]



Chemical Compound[2]

Protein Structure[3]

[1] https://commons.wikimedia.org/wiki/File:Social_Network_Diagram_%28large%29.svg
[2] https://commons.wikimedia.org/wiki/File:An_example_of_a_lead_compound.png
[3] https://commons.wikimedia.org/wiki/File:Spombe_Pop2p_protein_structure_rainbow.png

Different Types of Algorithms

Graph Analytics

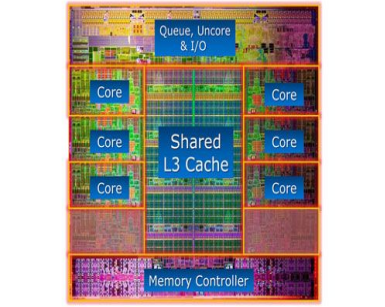
Determine strength / direction of relationships in graph (e.g., PageRank)

Graph Learning

Machine learning which learns from graphs to make predictions (e.g., unsupervised learning)

Graph Mining

Extraction of novel and useful knowledge from data (e.g., pattern matching)



What are the strengths and weaknesses of an architecture?

- Degree / type of parallelism
- Programmability
- Communication bandwidth and speed
- Memory system capacity, bandwidth, speed
- Compute resources
- Branch performance

Topics

Quantifying resource demands

Modeling performance

Concurrency and parallelism

Multiprogramming w/ processes

Message passing

Threads and synchronization

Graphics and vector processing

AI/ML processing

Cluster computing

Accelerators / PIM

What are the course objectives?

Learn about and learn how to program parallel systems

Learn to assess application resource demands

Learn about graph algorithms

Develop model for mapping applications to hardware

Experiment (including failing well)

Collaborate and learn together

Introductions

What should we call you?

Why interests you about this class?

What's the coolest CS concept you've learned so far?

What do you do for fun?

Anything else you want to share?

Logistics

- Meeting times
  - TuTh 9:55-11:10
- Help hours
  - when2meet poll
  - Likely in CS majors lab
  - Or by appointment
- Online help
  - http://cs338-parallelprocess.slack.com
- Course materials
  - https://cs.williams.edu/~kshaw/cs338
  - Glow course page

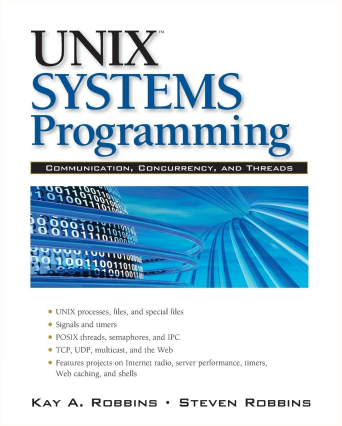
- Course activities
  - Class participation (5%)
  - Reading and short comprehension quizzes (5%)
    - 2 lowest dropped
  - Programming assignments (40%)
    - Including writeup / presentation
  - Midterm (30%)
    - Thursday, March 13
  - Final project (20%)
    - Map new algorithm OR
    - Map algorithm to new system

Collaboration!

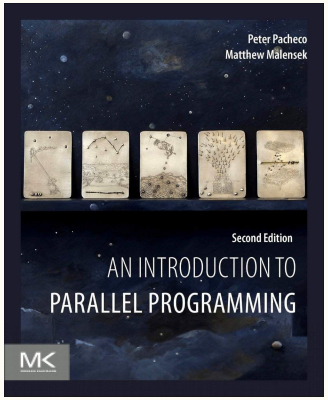
- This class is all about learning together and from one another.
- Assignments will be partner eligible
  - Don't create solutions with non-partners
- In class presentations of findings / insights
- Don't look for solutions online, but use online resources

Books

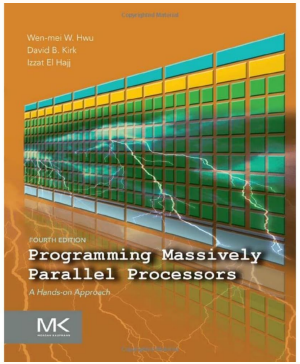
- Unix Systems Programming: Communication, Concurrency, and Threads (2nd Edition), by Kay Robbins and Steve Robbins



- An Introduction to Parallel Programming (2nd Edition), by Peter S. Pacheco and Matthew Malensek



- (Recommended) Programming Massively Parallel Processors: A Hands-on Approach (4th Edition), by David B. Kirk, Wen-mei W. Hwu, and Izzat El Hajj

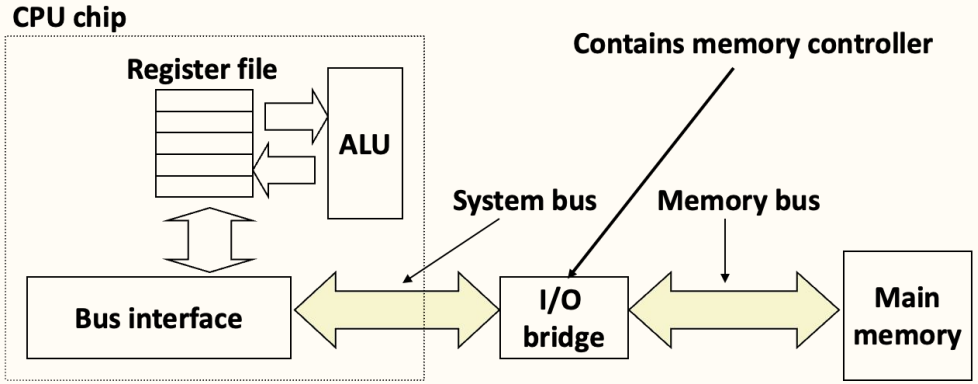


Let's Talk About Failing Well

https://www.youtube.com/watch?v=Gb9tjnJWu5g

Computer Architecture Review

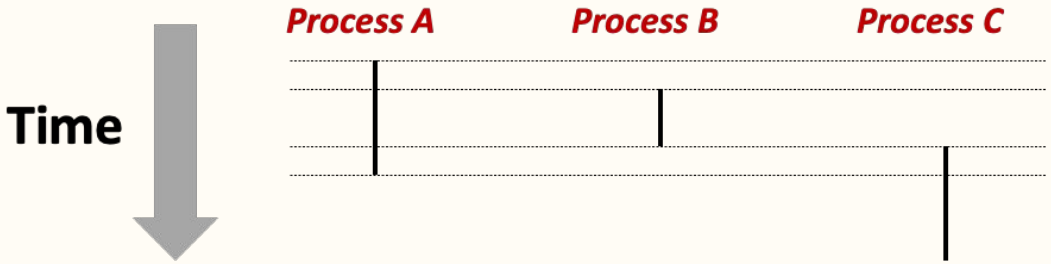
- CPU executes instructions
  - ALU
  - Memory
  - Control
- Memory hierarchy gives illusion of lots of fast memory
  - Caches
  - Main memory
  - Disk
- Time-share resources between different processes



- Performance enhancers
  - Exploiting memory locality
  - Branch prediction
  - Using less expensive compute ops

Concurrency

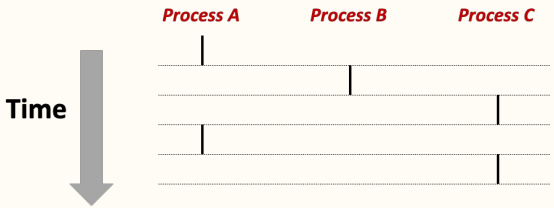
- Definition: sharing of resources or performing of tasks in progress at the same time
- Motivation
  - Multiple physical resources
  - Independent tasks
- Multiple locations
  - Hardware level
  - Application level
- Systems with concurrency
  - Operating System
  - Parallel Computer
  - Distributed System



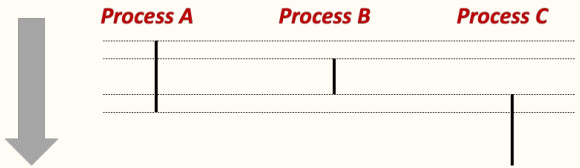
Parallelism

- Having multiple physical resources that can be used at the same time
- Typically, when we talk about parallel computing, the compute resources are working jointly to solve the same problem
- Reasons for it:
  - We have enough transistors to duplicate resources
  - We can't figure out ways to speed up sequential programs in HW anymore
  - Some applications have a lot of independent tasks that can run concurrently

Sequential



Parallel



AI Overview

Concurrency is the management of multiple tasks at once, while parallelism is the execution of multiple tasks simultaneously. Concurrency can create the illusion of parallelism, but parallelism is not always concurrent.

Explanation

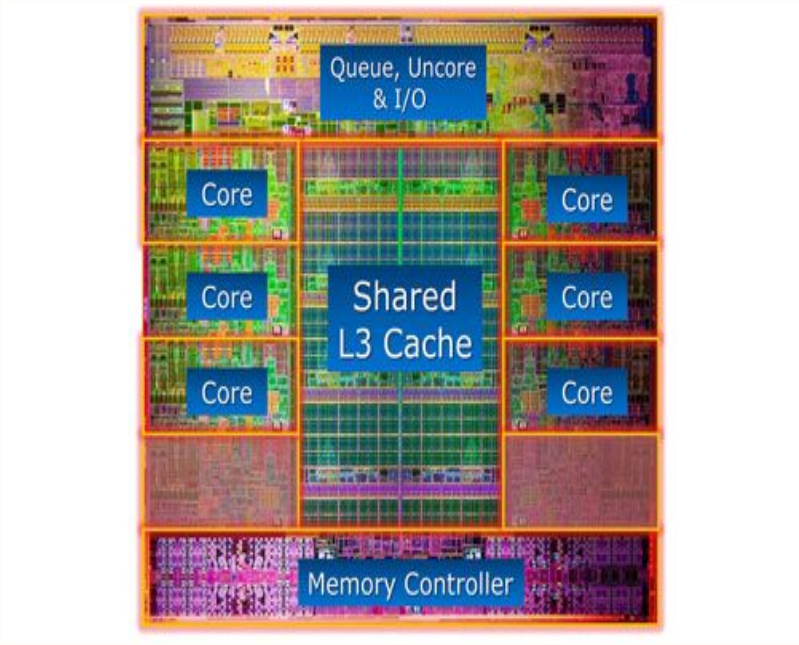
Concurrency

Involves managing multiple tasks at once, often on a single processor. Concurrency can make it seem like multiple tasks are running at the same time, but they are actually running interleaved.

Parallelism

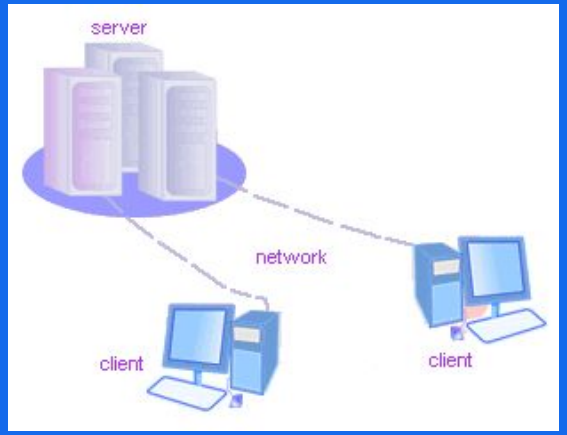
Involves executing multiple tasks at the same time on multiple processors. Parallelism can be achieved by breaking down a task into smaller subtasks that can be executed in parallel.

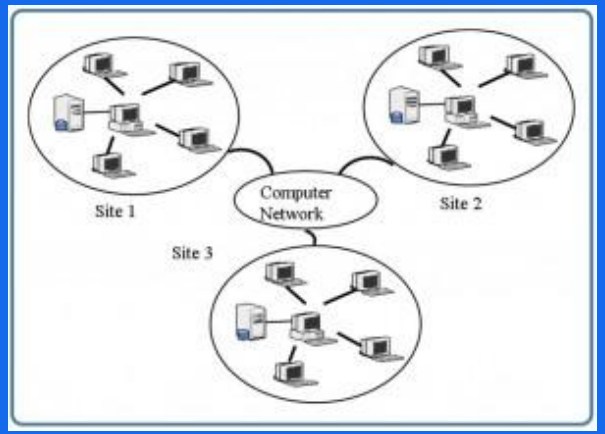
Modern Computer Chips Have Parallel Resources



Parallel Computing

- Lots of compute resources
- Close physical location
- Resources used to solve parts of single problem simultaneously
- Trends:
  - Multiple processing cores on single chips
  - Systems with multiple chips housed together
  - Specialized processors





Distributed Computing

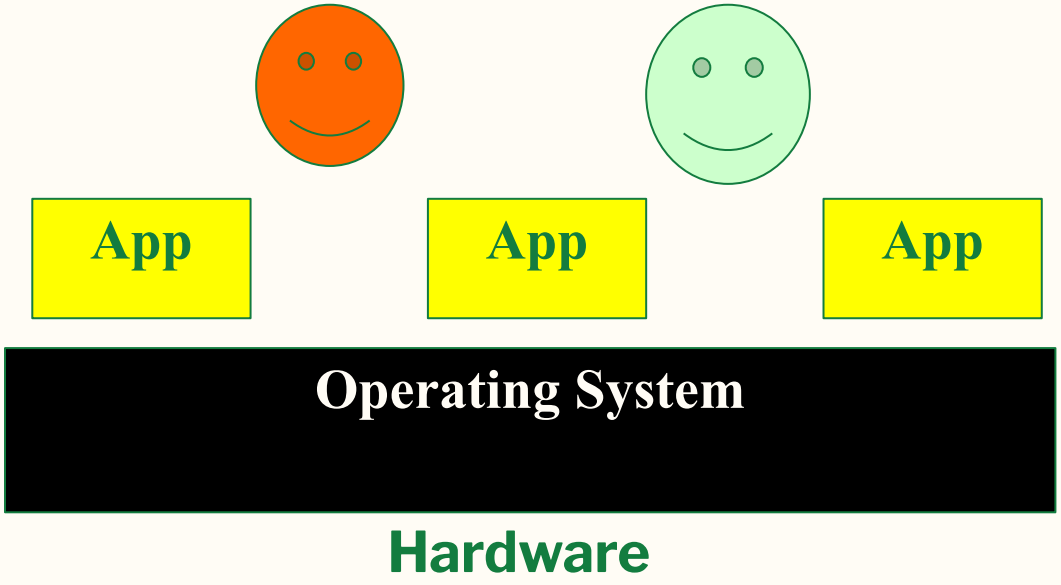
- Often commodity parts
- Loosely coupled computation
- Connected by internet
- Think “data centers”

Operating System *Process* Concept Enables Timesharing and Multiprogramming

- Timesharing
  - Resources are shared amongst multiple tasks in time
    - Creates illusion that tasks execute simultaneously even though only one CPU
- Multiprogramming
  - More than one task ready to execute
  - Switch between tasks when waiting for resource or input
    - Enables efficient use of resources, keeping them from being idle when in need

What is an Operating System?

- Layer of software that manages a computer's resources for its users and their applications



What does an OS do?

Referee

- *Resource allocation among users, apps*
- *Isolation of different users, apps*
- *Communication between users, apps*

Illusionist

- *Each app thinks it has entire machine to itself*
- *Infinite number of processors, infinite amount of memory, reliable storage, reliable network transport*

Glue

- *Common services facilitate sharing across apps*
- *Libraries, user interface widgets, etc.*

Example: File Systems

Referee

- *Prevent users from accessing each other's files without permission*
- *Even after a file is deleted and its space re-used*

Illusionist

- *Files can grow (nearly) arbitrarily large*
- *Files persist even when the machine crashes in the middle of a save*

Glue

- *Named directories, printf, …*

Unit of Work: *Process*

- Program is a **static** concept
  - Executable file contains sequences of instrs to perform tasks and initialize data
- *Process* is the corresponding **dynamic** concept
  - It is an executing program
  - It is the abstraction for protected execution provided by kernel. Asks OS for permission to do stuff.
- If you have many instances of the same program running, you have many processes running
- The OS allocates resources (CPU, files, memory, etc.) to processes

Process Abstraction

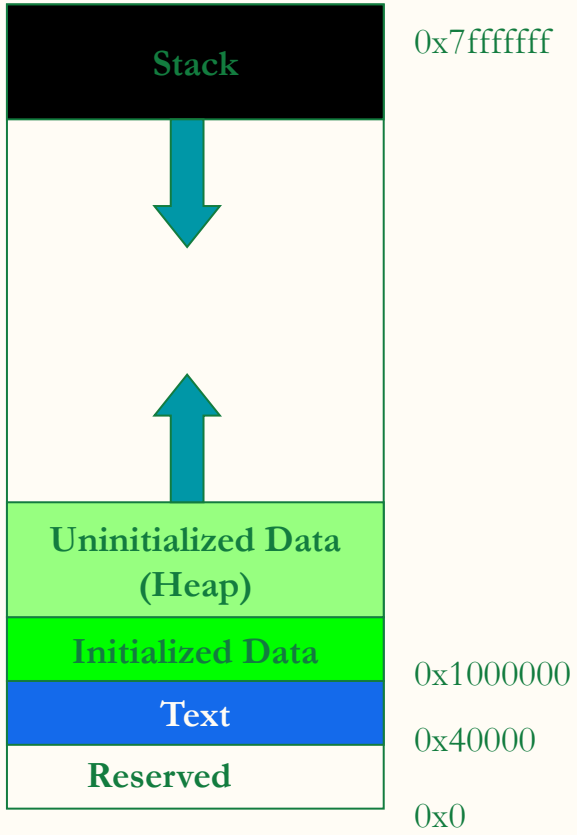
- Process: an *instance* of a program, running with limited rights
  - Thread: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - Address space: set of rights of a process
    - Memory that the process can access
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)
  - Process identifier (PID) to identify process
  - Has connections to ancestor and child processes

Tracking Processes

- OS keeps track of all processes
- Process Control Block (PCB) stores all info about process
  - where process stored in memory
  - where executable resides on disk
  - which user invoked it
  - what privileges process has
  - ...
- Processes can be standalone or interact with other processes through OS constructs
  - Filesystem, pipes, signals, shared memory, network

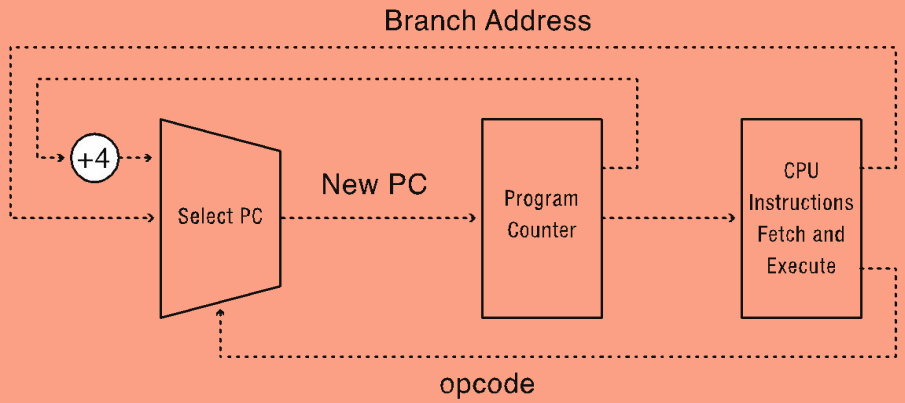
Process Address Space

- Each process has an address space
  - The logical memory it can access
- The address space is divided into segments:
  - Text
    - Instructions
  - Initialized Data
    - Globals
  - Uninitialized Data or Heap
    - `new` allocates space here
  - Stack
    - local variables are given space here



Process is a *Running* Program

- Starts at a given instruction (e.g., main)
- Executes on CPU
  - Current instruction address
  - Current register state



How Do the OS and Processes Interact?

- OS lets process run on hardware resources, but how does it control it and help it?
- Interrupts
  - Peripheral device can set a HW flag indicating it needs service
  - HW always checks if flag set
  - If flag set, OS will cause interrupt handler to run, taking CPU away from process
- Signals
  - SW notification of an event
  - OS may send process a signal based on interrupt
  - Process may generate signal (e.g., divide by zero) that it must then handle via signal handler
  - Inter-process communication between unrelated processes
- System calls
  - Process explicitly asks OS to do something on its behalf
    - e.g., read from console, write to disk