CS 470
Spring 2021

Mike Lam, Professor

Parallel and Distributed Systems

Advanced System Elective

World's fastest supercomputer (2020)

- Fugaku (富岳)
  - RIKEN Center for Computational Science, Kobe, Japan
  - 7,630,848 cores total, custom Fujitsu A64FX CPUs (48 cores each)
  - 5 PB memory
  - RHEL w/ custom Fujitsu compilers
  - 537 Pflops peak Linpack performance
  - 30 MW power consumption

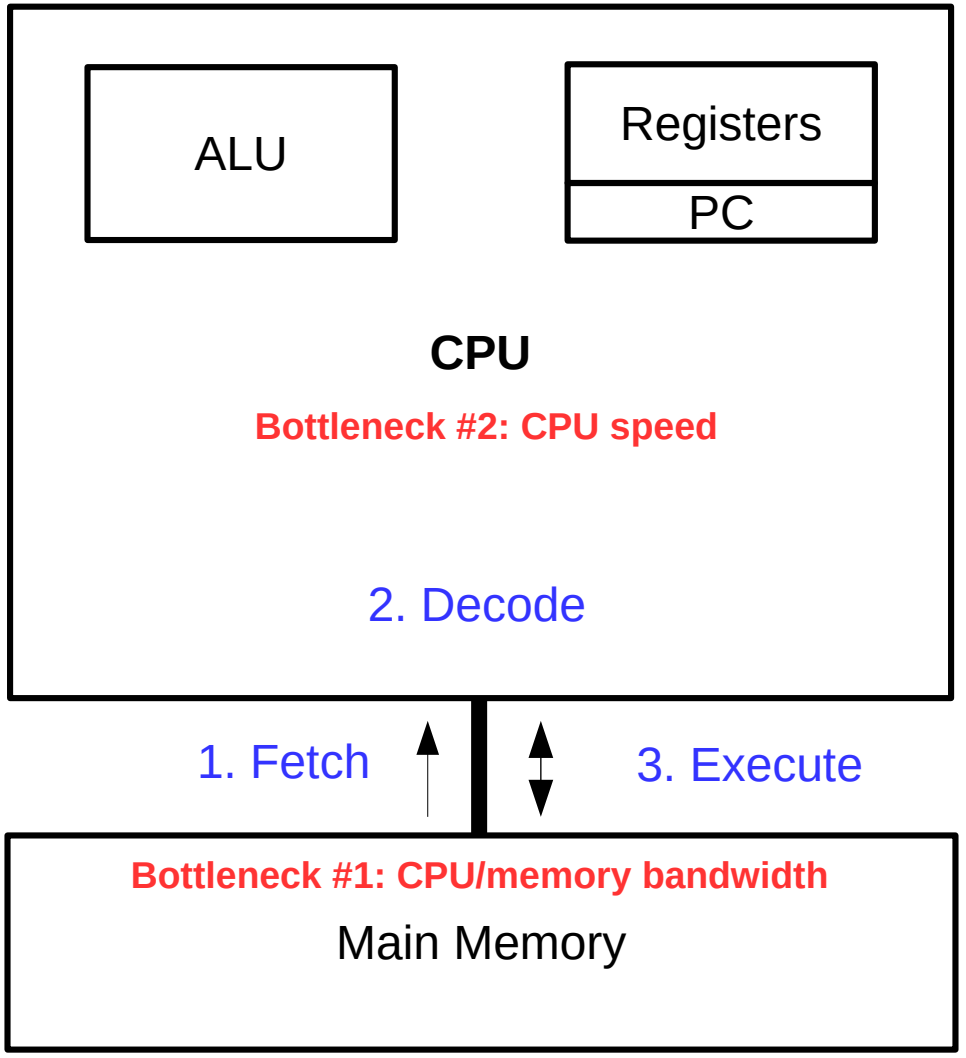
Sources:

- `top500.org`
- `ieee.org`

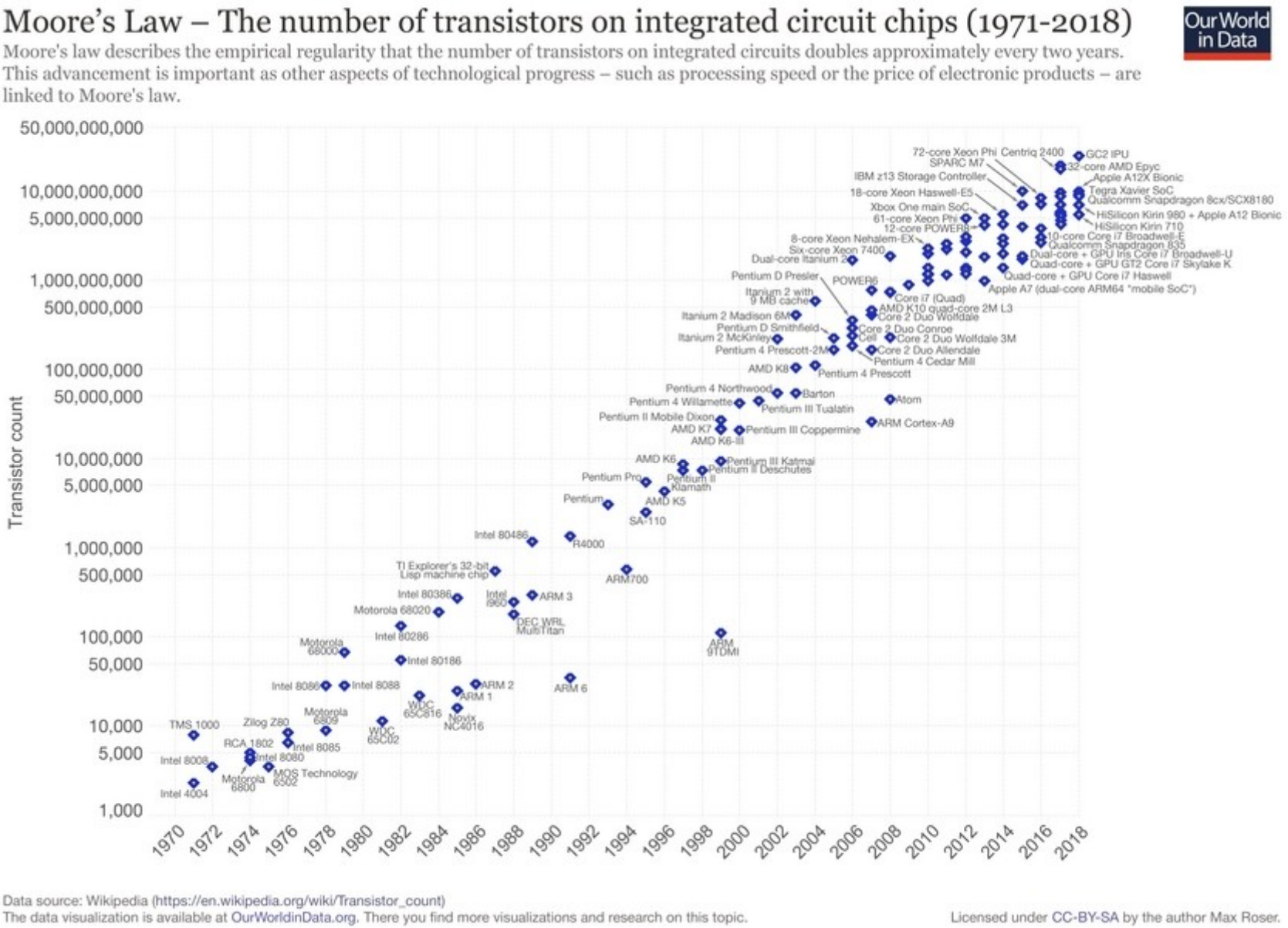
Motivation

- Why do we have (and why should we study) parallel and distributed systems?
- Let's go back to CS 261 …

von Neumann (CS 261)



Moore’s Law



MOSFET scaling (process nodes)

10 µm	– 1971
6 µm	– 1974
3 µm	– 1977
1.5 µm	– 1981
1 µm	– 1984
800 nm	– 1987
600 nm	– 1990
350 nm	– 1993
250 nm	– 1996
180 nm	– 1999
130 nm	– 2001
90 nm	– 2003
65 nm	– 2005
45 nm	– 2007
32 nm	– 2009
22 nm	– 2012
14 nm	– 2014
10 nm	– 2016
7 nm	– 2018
5 nm	– 2020

Future

3 nm	– *~2021*
2 nm	– *~2024*

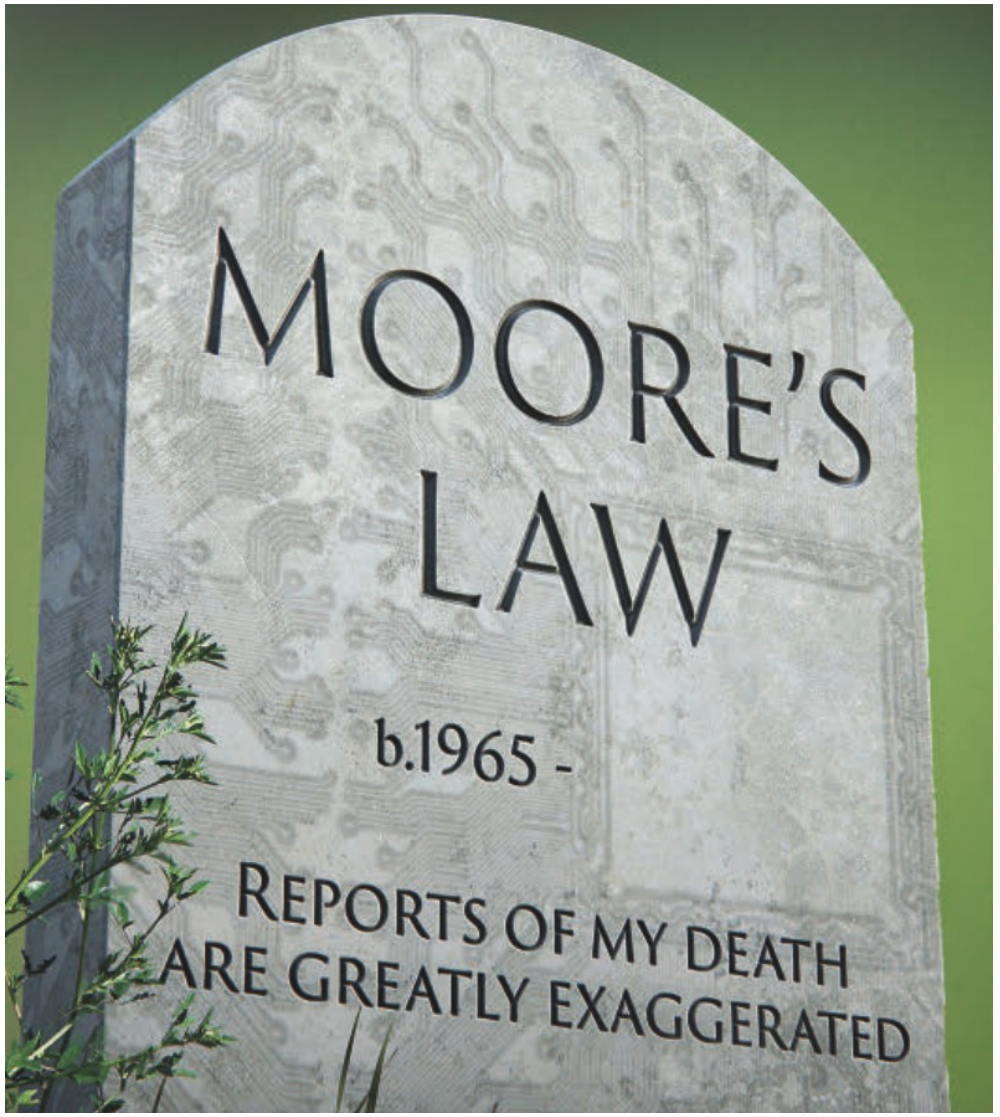
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Issue: CPU Physics

- More transistors → higher energy use
- Higher energy use → higher heat
- Higher heat → lower reliability (e.g., signal leakage)
- Manufacturing limitations
- Quantum effects at sub-nanometer resolution
- Related observation: Dennard scaling (i.e., power consumption per area remains constant) failed in 2000s

Will Moore's Law eventually fail?

Moore’s Law



Cover of the January 2017 edition of *Communications of the ACM*

Alternative to Moore's Law

- Scale out, not up
  - **More** processors rather than **faster** processors
  - Requires parallelism at higher levels than instruction-level parallelism (e.g., pipelining)

"Post-Moore's Law Era"

Alternative to Moore's Law

- "New" problem: writing parallel software
  - Running a program in parallel is not always easy
  - Sometimes the **problem** is not easily parallelizable
  - Sometimes communication overwhelms computation
  - But the stakes are too high to ignore parallelism!

Core issue: parallelization

- As humans, we usually think sequentially
  - *"Do this, then that"* w/ deterministic execution
- Parallel programming requires a different approach
  - *"Do this and that in any order (or at the same time)"*
  - Introduction of non-determinism
  - Requires sophisticated understanding of dependencies
- Sometimes, the best parallel solution is to discard the serial solution and revisit the problem

Example from IPP

- Compute n values and calculate their sum
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

How should we parallelize this?

What problems will we encounter?

Example from IPP

- Initial parallel solution:

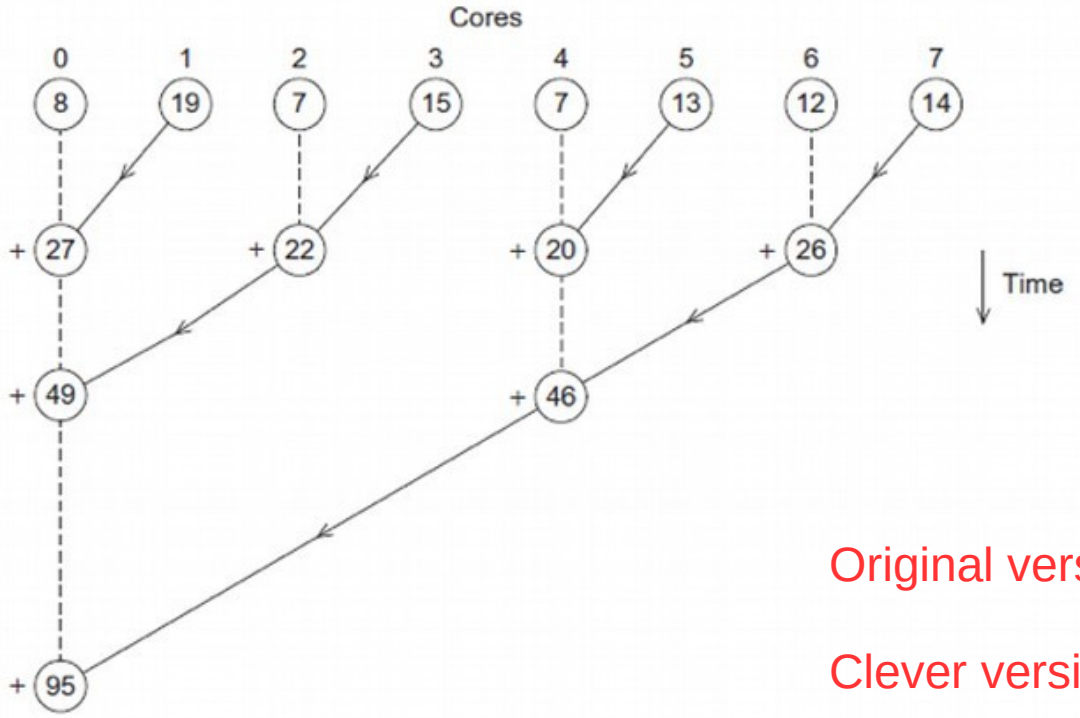
```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}

if (I'm the master core) {
   sum = my_x;
   for each core other than myself {
      receive value from core;
      sum += value;
   }
} else {
   send my_x to the master;
}
```

Insight: split up the compute work, then have the master core aggregate the results

Shared-mem alternative: use a mutex!

Example from IPP

- There's a better way to compute the final sum
  - Distribute the work; don't do all the additions serially
  - Fewer computations on the critical path (longest chain of work)



Original version: 7 messages and 7 additions

Clever version: 3 messages and 3 additions

Example from IPP

- Improvement is even greater w/ higher # of cores
- For 1000 cores:
  - Original version: 999 messages and 999 additions
  - Clever version: 10 messages and 10 additions

This is an asymptotic improvement!

(*why?*)

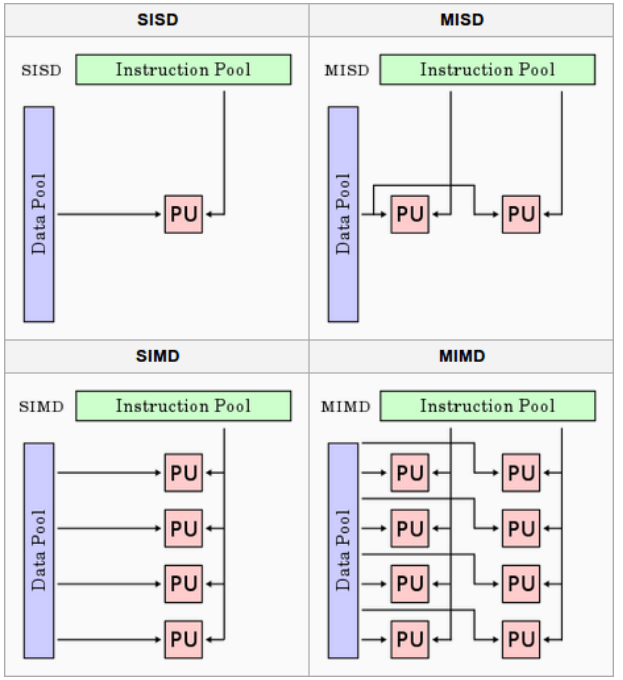
System architectures

- However, there's also a limit to how many cores we can put in a single computer
  - Energy consumption, heat emission, memory saturation
- Solution: more computers!
  - Communicate via network
  - This is called a distributed system
- There are so many kinds of systems
  - We need ways to concisely describe them

System architectures

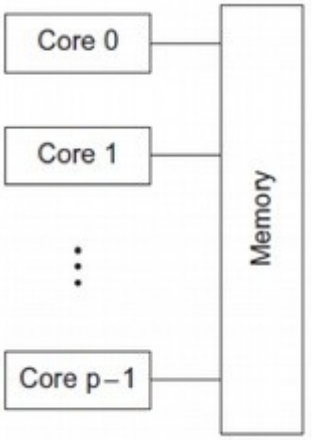
- Flynn's Taxonomy
  - Single Instruction, Single Data (SISD)
    - Traditional von Neumann
    - Increasingly insufficient!
  - Single Instruction, Multiple Data (SIMD)
    - Vector instructions (SSE/AVX)
    - GPUs and other accelerators
  - Multiple Instruction, Multiple Data (MIMD)
    - Single Program, Multiple Data (SPMD)
    - Shared memory and distributed memory
  - Single Instruction, Multiple Threads (SIMT)
    - New term gaining prominence in past few years
    - Alternative way of describing GPUs

Trend: higher number of slower, more energy-efficient processors

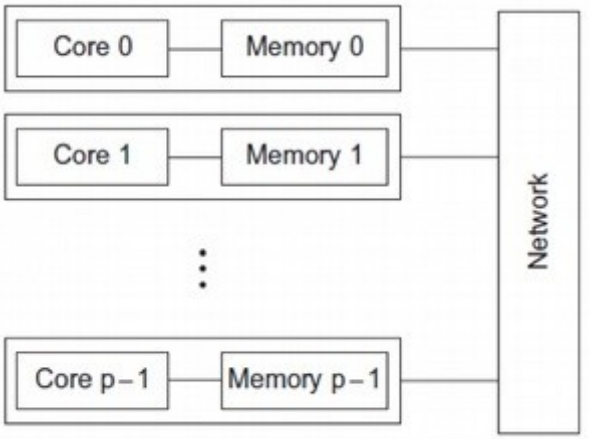


System architectures

- Shared memory
  - Idea: add more **CPUs**
  - Paradigm: threads
  - Technologies: Pthreads, OpenMP
  - Issue: synchronization



- Distributed memory
  - Idea: add more **computers**
  - Paradigm: message passing
  - Technologies: MPI, PGAS
  - Issue: data movement



Shared memory software

- Threading libraries
  - Low-level explicit multiprocessing programming
  - Independent threads of execution; shared variables
  - Synchronization mechanisms (locks, semaphores, conditions, barriers)
    - Prevents data races and enforces thread safety
  - Libraries: Pthreads, Java Threads, Boost Threads

- Language extensions
  - Write one program that is both serial and (implicitly) parallel
  - Use pragmas to annotate the program with parallelism guidelines
  - Threading and synchronization added automatically (usually by compiler)
  - Languages: OpenMP, OpenACC

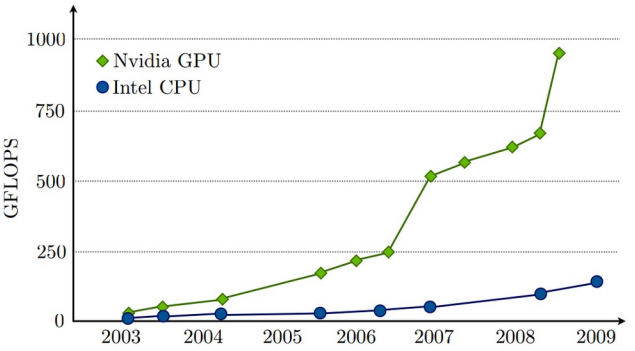
Distributed memory software

- Message-Passing Interface (MPI)
  - Low-level explicit message-passing programming
  - Point-to-point operations (Send / Receive)
  - Collective operations (Broadcast / Reduce)
    - Allow MPI implementations to optimize data movement
  - Libraries: OpenMPI, MPICH, MVAPICH
- Partitioned Global Address Space (PGAS)
  - Make distributed memory look and act “like” shared memory
  - Split address space among all processes
  - Message passing is added automatically (usually by compiler)
  - Languages: Chapel, X10, Fortress

Hybrid architectures

- Shared memory on the node
  - Hardware: many-core CPU and/or coprocessor (e.g., GPU)
  - Enables energy-efficient strong scaling
  - Technologies: OpenMP, CUDA, OpenACC, OpenCL

- Distributed memory between nodes
  - Hardware: interconnect and distributed FS
  - Enables weak scaling w/ efficient I/O
  - Technologies: Infiniband, Lustre, HDFS, MPI



History of parallelism

- Uniprogramming / batch (1950s)
  - Traditional von Neumann, no parallelism
- Multiprogramming / time sharing (1960s)
  - Increased utilization, lower response time
- Multiprocessing / shared memory (1970s)
  - Increased throughput, strong scaling
- Distributed computing / distributed memory (1980s)
  - Larger problems, weak scaling
- Hybrid computing / heterogeneous (2000s)
  - Energy-efficient strong/weak scaling

Shared memory summary

- Shared memory systems can be very efficient
  - Low overhead for thread creation/switching
  - Uniform memory access times (symmetric multiprocessing)
- They also have significant issues
  - Limited scaling (# of cores) due to interconnect costs
  - Requires explicit thread management and synchronization
  - Caching problems can be difficult to diagnose
- Core design tradeoff: synchronization granularity
  - Higher granularity: simpler but slower
  - Lower granularity: more complex but faster
  - Paradigm: synchronization is expensive

Distributed memory summary

- Distributed systems can scale massively
  - Hundreds or thousands of nodes, petabytes of memory
  - Millions of cores, petaflops of computation capacity
- They also have significant issues
  - Non-uniform memory access (NUMA) costs
  - Requires explicit data movement between nodes
  - More difficult debugging and optimization
- Core design tradeoff: data distribution
  - How to partition and arrange the data; is any of it duplicated?
  - Goal: minimize data movement
  - Paradigm: computation is “free” but communication is not

Our goals this semester

- Learn some parallel & distributed programming technologies
  - Pthreads, MPI, OpenMP, Chapel or Julia (maybe)
- Study parallel & distributed system architectures
  - Shared memory, distributed, hybrid, cloud
- Study general parallel computing approaches
  - Foster's methodology, message passing, task/data decomposition
- Analyze application performance
  - Speedup, weak/strong scaling, communication overhead
- Explore parallel & distributed issues
  - Synchronization, fault tolerance, consistency, security

Parallel & distributed systems

- Hardware architectures
- Software patterns & frameworks

First half of CS 470

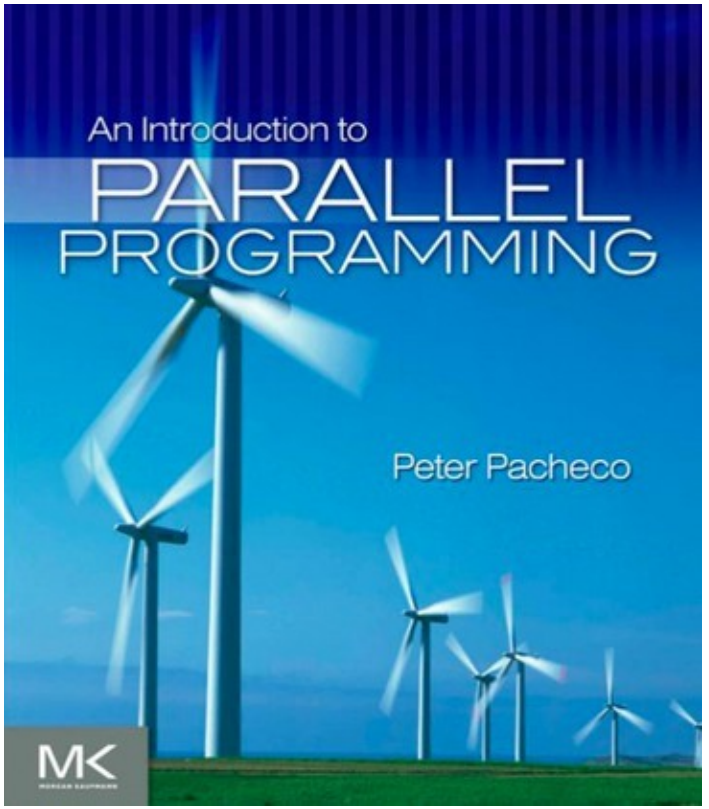
- Interconnects and naming
- Synchronization and consistency
- Fault tolerance
- Cloud computing
- Security
- Applications: Web & File Systems

Second half of CS 470

Course textbook

- **An Introduction to Parallel Programming**
  - Peter S. Pacheco

- Sources:
  - JMU Bookstore ($66)
  - Amazon ($45 used)
  - O'Reilly (free)
    - (link on syllabus)
  - ~~Library (on reserve)~~
    - (thanks, COVID...)



Course notes

- The course slides are quite comprehensive
  - Especially during the second half
  - Not all topics will be covered explicitly in class
  - You are responsible for reviewing the slides for any material not explicitly covered
  - Ask clarification questions on Slack

Course format

- Public files and calendar on website (bookmark it!)
- Private files and grades on Canvas
- Canvas quizzes (usually 1-2 per week)
  - Two attempts on reading quizzes, one attempt on review quizzes
- In-class labs (at most 1 per week) w/ Canvas submission
  - Groups of up to three (submit one copy with everyone's names)
- Regular projects (every 2-3 weeks) w/ Canvas submission
  - Groups of up to two
- Research project (entire semester)
  - Groups of up to four (three recommended)
- Take-home, open-book exams (midterm & final)

Standard projects

- Practice using parallel and distributed technologies
- Practice good software engineering and code analysis
- Submission: code + analysis / review / response
  - Code can be written individually or in teams of two
    - Benefits vs. costs of working in a team
  - Analysis must be included as comments at top
    - Requirements will vary by assignment
  - Graded code reviews after project submission
    - Review two other submissions; must be done individually
  - Response to assess the reviews you receive

Research

- In addition to standard projects, we will be doing a research project this semester

Discussion:

What is research?

What is research?

- *"Research is the process of **finding information**."*
- *"Research is when you use your own time to understand and **describe in words** a topic you did not know about before."*
- *"**Looking for credible information** pertaining to a specific topic."*
- *"**Utilization of academic, peer-reviewed publications** in order to better understand or solve a problem."*
- *"It's taking concepts or ideas and **collecting valuable information** pertaining to it, with some fact checking of course!"*

What is research?

- *"Working at the edge of knowledge in a field attempting to **push that frontier** a little further with your work."*
- *"Thorough investigation into a subject, with the end result of **finding new information**."*
- *"Research is building on the work of others on a topic of the researchers choice to posit new arguments and **find new discoveries** that might interest yourself or the general public."*
- *"**Learning new things** then doing those things then writing about those things."*

What is research?

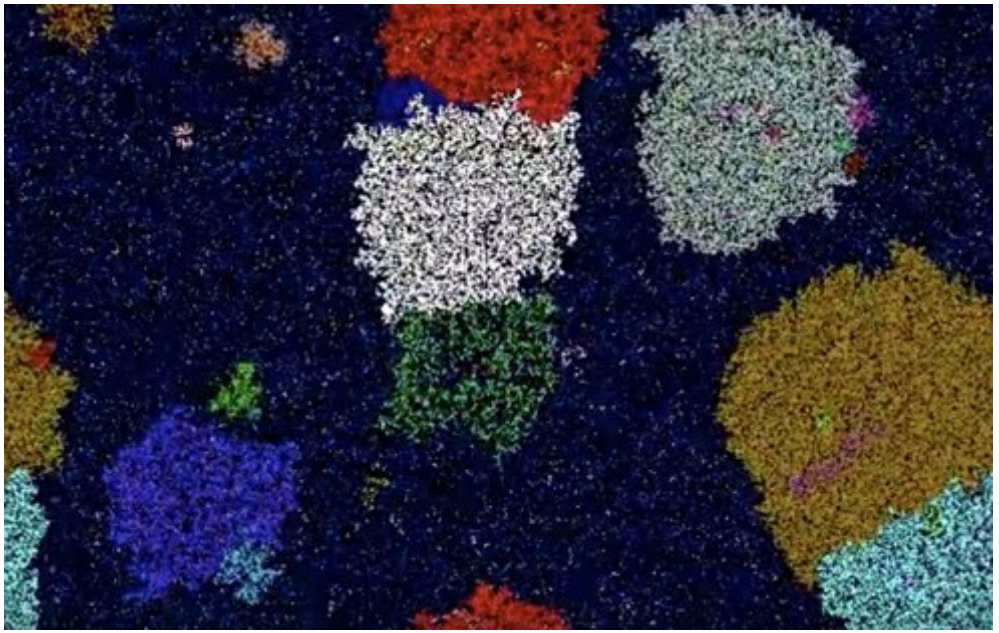
- *“Research is the process of systematically casting a fishing rod into the unknown and hoping that you reel in something worthwhile. Sometimes you catch nothing, sometimes you get something worthwhile, and sometimes you get something that looks worthless until it's published by somebody else three years later. But regardless you slowly begin to learn about the world on the other end of that hook.”*

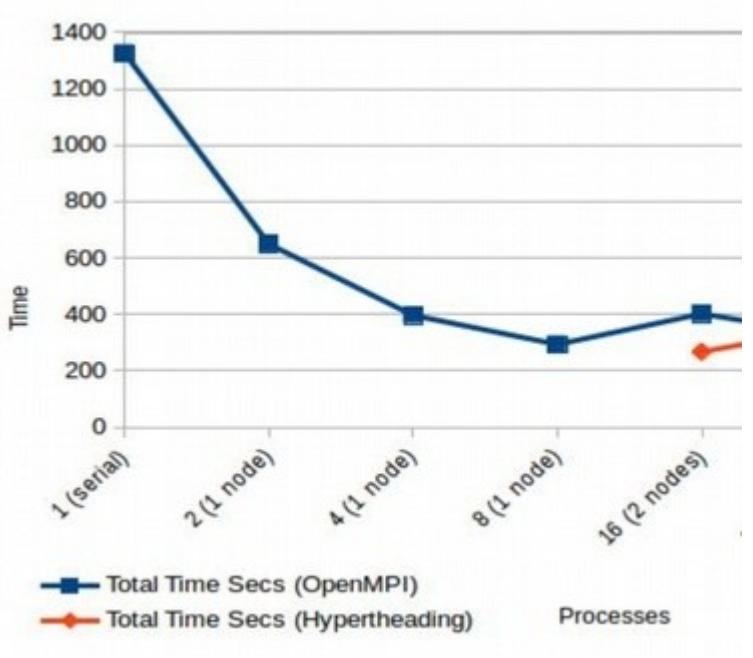
Research project

- Semester-long project
  - Teams of 2-4 people
  - Personalized topic; largely open-ended
  - Must involve parallel/distributed systems or software
  - Must include significant programming or analysis or large data sets
  - Preferably uses Pthreads, OpenMP, or MPI
  - Multiple submissions:
    - Overview, ideas, teams, proposal, mid-project, poster/video, final
    - Use LaTeX for proposal, mid-project report, and final report
  - Graded on progress and application of course concepts
  - Goal: **significant, open-ended** “capstone” project experience

Example Project

- Nanopond Simulation Parallelization





Example Project

- JMU Cluster Performance Variation

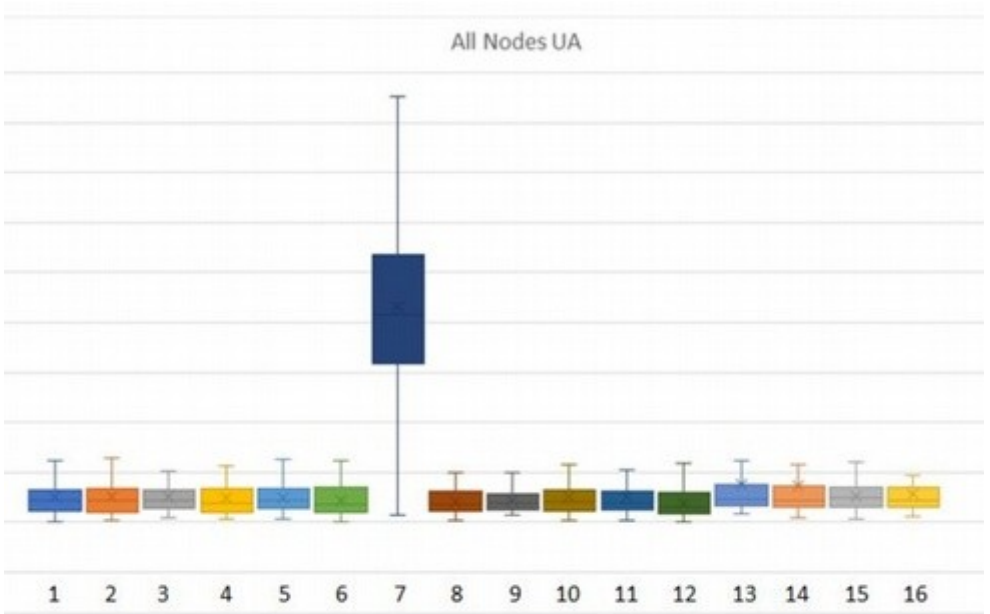


Figure 2: UA Results Across All Nodes

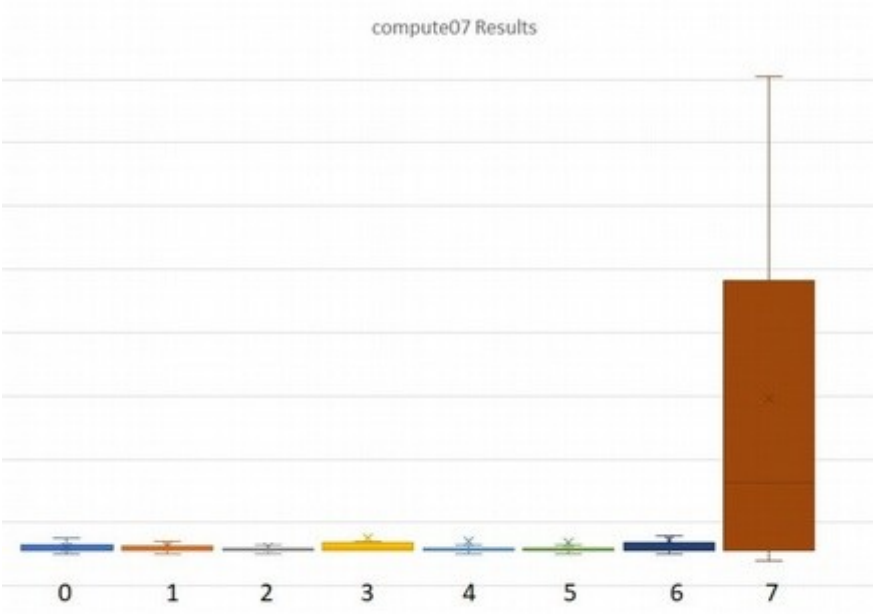
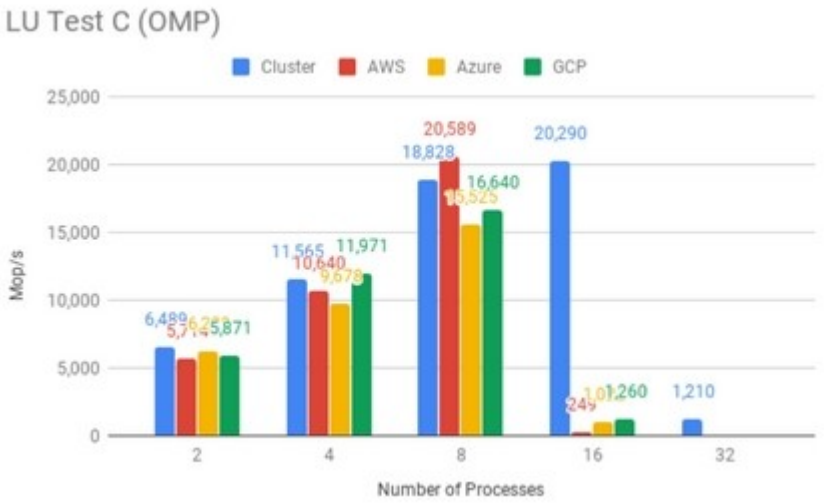
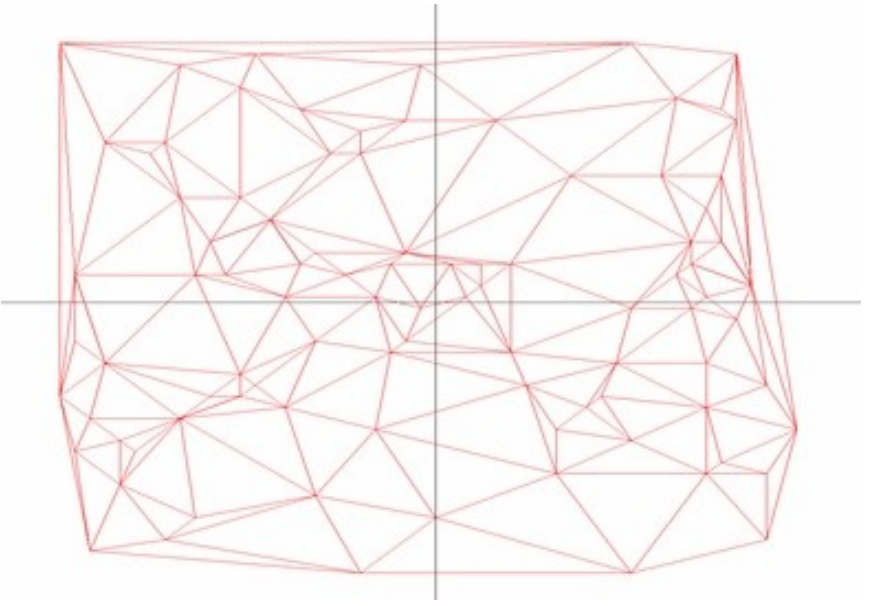


Figure 3: Core 7 is much slower

Example Projects

- Parallelizing Shamir’s Secret Sharing Algorithm
- Traveling Salesman Heuristic Scaling Analysis
- Cloud Benchmarking
- Floating-Point Variation in Multithreaded Code
- Parallel Triangulation
- Energy Measurement



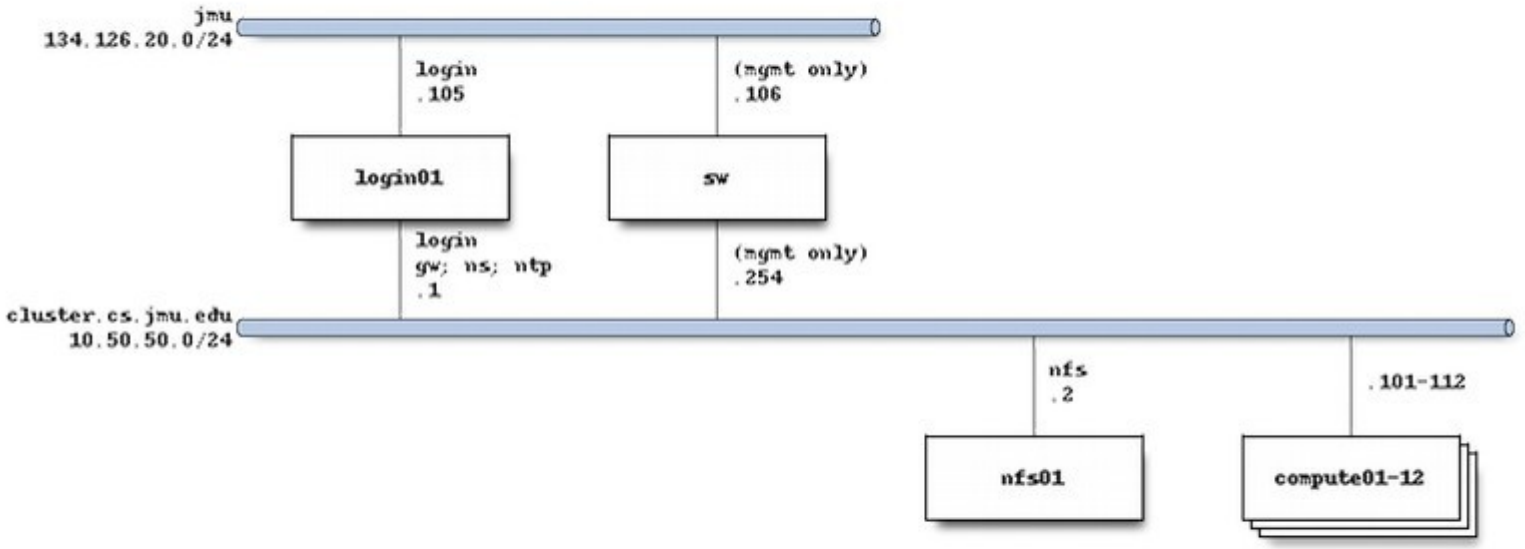


Research Project

- Most importantly: **DON’T PANIC!**
- Read about a lot of previous projects
- Find a topic you’re excited about
- Keep project teams to 2-3 members
- If you **start early**, schedule time weekly to meet and work with your group, leave time to deal with setbacks, and **communicate regularly** with me, your project *will* be successful.

Our distributed cluster

- **Compute nodes**: 16x Dell PowerEdge R430 w/ Xeon E5-2630v3 (8C, 2.4Ghz, HT) 32 GB
- **Login node**: Dell PowerEdge R430 w/ 2x Xeon E5-2630v3 (8C, 2.4Ghz, HT) 32 GB
- **File server**: Dell PowerEdge R730 w/ Xeon E5-2640v3 (8C, 2.6Ghz, HT) 32 GB
  - **Storage**: 8x 1.2TB 10K SAS HDD w/ RAID
- **Interconnect**: Dell N3024 Switch 24x1GbE, 2x10GbE SFP+ (212Gbps duplex)



Cluster access

- Detailed instructions online:
  `w3.cs.jmu.edu/lam2mo/cs470/cluster.html`
- Connect to login node via SSH
  - Hostname: `login.cluster.cs.jmu.edu`
  - User/password: *(your e-ID and password)*
- Recommended conveniences
  - Set up public/private key access from `stu`
  - Set up `.ssh/config` entries

Cluster access

- Things to play with:
  - "`squeue`" or "`watch squeue`" to see jobs
  - "`srun <command>`" to run an interactive job
    - Use "`-n <p>`" to launch *p* processes
    - Use "`-N <n>`" to request *n* nodes (defaults to *p*/8)
    - The given "`<command>`" will run in every process
  - "`salloc <command>`" to run an interactive MPI job
    - Use "`-n <p>`" to launch *p* MPI processes

```
srun hostname
srun -n 4 hostname
srun -n 16 hostname
srun -N 4 hostname
srun sleep 5
srun -N 2 sleep 5
```

```
module load mpi
salloc -n 1 mpirun /shared/cs470/mpi-hello/hello
salloc -n 2 mpirun /shared/cs470/mpi-hello/hello
salloc -n 4 mpirun /shared/cs470/mpi-hello/hello
salloc -n 8 mpirun /shared/cs470/mpi-hello/hello
salloc -n 16 mpirun /shared/cs470/mpi-hello/hello
(etc.)
```

What's the max *n*?

Have a great semester!

- Take course intro survey (free points!)
- Read IPP Ch.1 and 2.6 (reading quiz tomorrow)
- Research overview due Friday
  - Start thinking about research projects!
- Make sure you can access Slack
- Make sure you can SSH into `login.cluster.cs.jmu.edu`
  - Must be on JMU network (e.g., proxy jump through `stu`)
  - Email me before the next class if you encounter issues