CS 470
Spring 2017

Mike Lam, Professor

Parallel and Distributed Systems

Advanced System Elective

Intro example

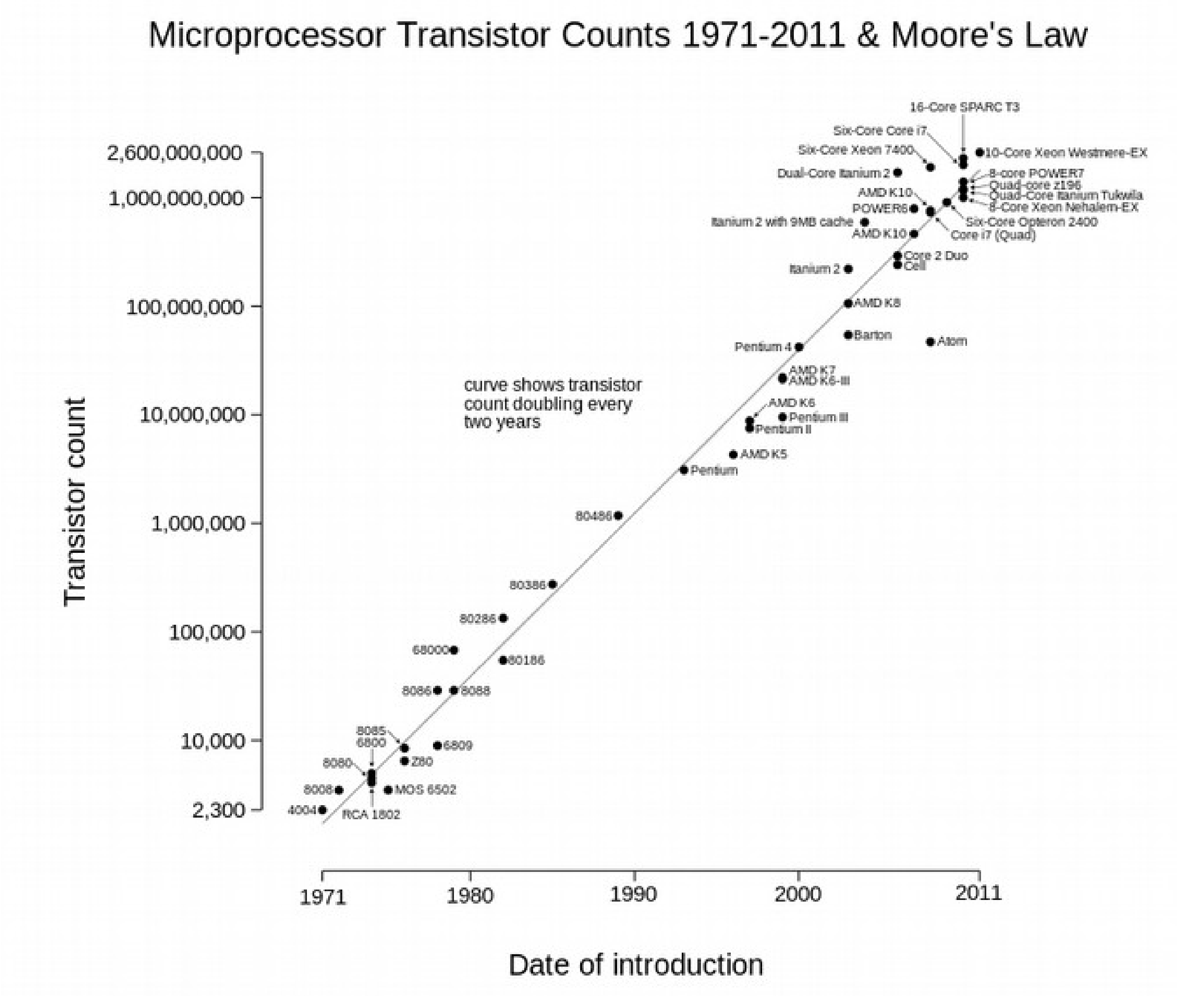
- Six sections of CS 159 this semester
  - Total of 180 students
- Suppose there is an exam with 15 questions
  - Suppose we have three graders
  - How do we finish the grading as quickly as possible?

Video

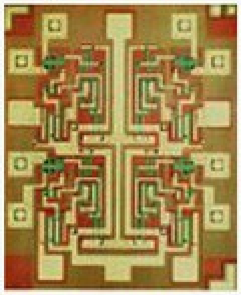
- #HPCMatters
  - `https://www.youtube.com/watch?v=9m0gZ2Gft4Q`

- A world without supercomputers
  - `https://www.youtube.com/watch?v=w3aI4sEUJ_Y`

Moore’s Law


Microprocessor Transistor Counts 1971-2011 & Moore's Law

Semiconductor manufacturing processes



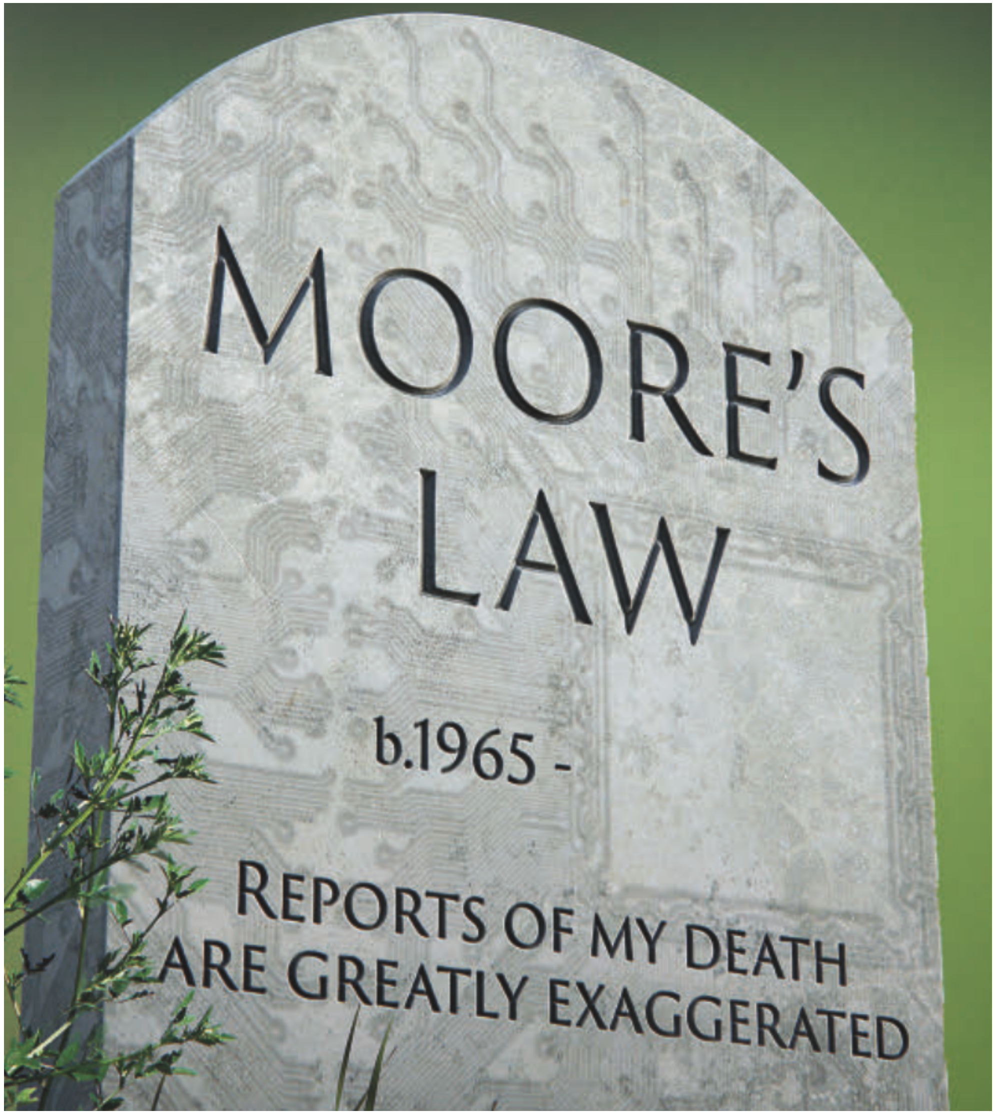
10 µm – 1971
6 µm – 1974
3 µm – 1977
1.5 µm – 1982
1 µm – 1985
800 nm – 1989
600 nm – 1994
350 nm – 1995
250 nm – 1997
180 nm – 1999
130 nm – 2001
90 nm – 2004
65 nm – 2006
45 nm – 2008
32 nm – 2010
22 nm – 2012
14 nm – 2014
10 nm – 2017
7 nm – *~2018*
5 nm – *~2020*

Issue: CPU Physics

- More transistors → higher energy use
- Higher energy use → higher heat
- Higher heat → lower reliability

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

History of Parallelism

- Uniprogramming / batch (1950s) - **CS 261**
  - One process at a time w/ complete control of CPU
- Multiprogramming / time sharing (1960s) - CS 261, **CS 450**
  - Multiple processes taking turns on a single CPU
  - Increased utilization, lower response time
- Multiprocessing (1970s) - **CS 361**, CS 450, **CS 470**
  - Multiple processes share multiple CPUs or cores
  - Increased throughput, increased parallelism
- Distributed processing (1980s) - CS 361, **CS 470**
  - Multiple processes share multiple computers
  - Capable of massive scaling

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

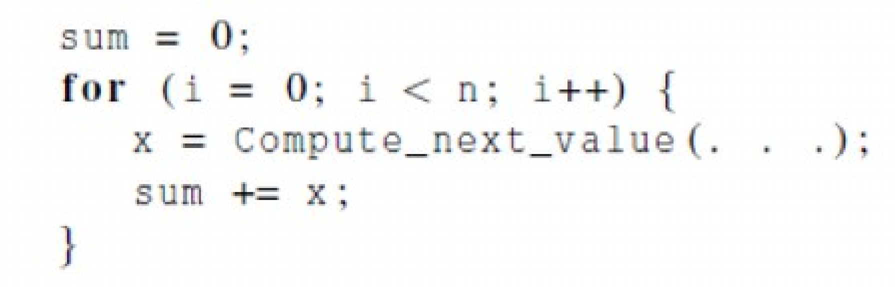
- As humans, we usually think sequentially
  - *"Do this, then that"* w/ deterministic execution

- Parallel programming requires a different approach
  - *"Do this and that in parallel (but how?)"*
  - Introduction of non-determinism

- 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:

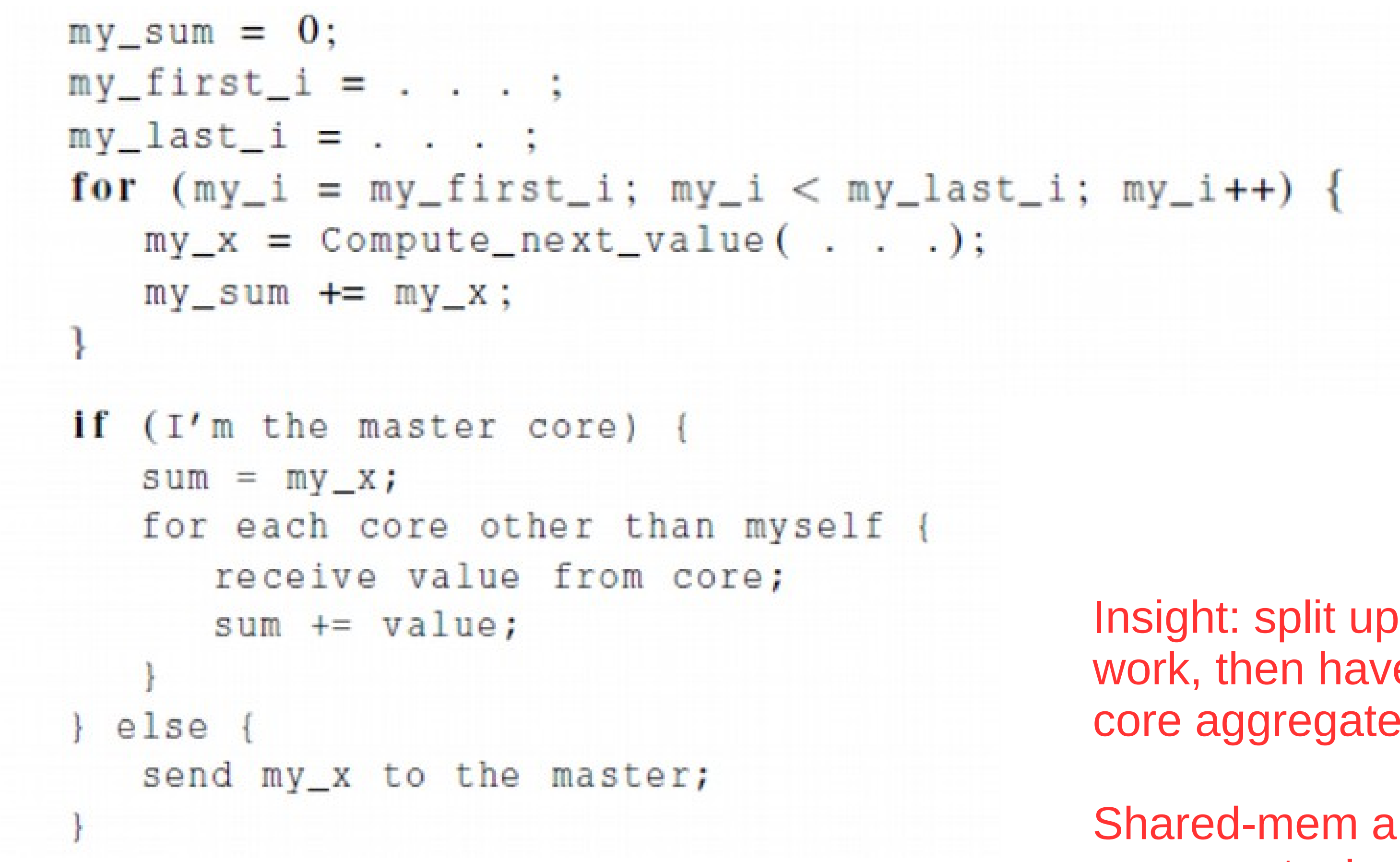


```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

How should we parallelize this?
What problems will we encounter?

Example from IPP

- Initial parallel solution:



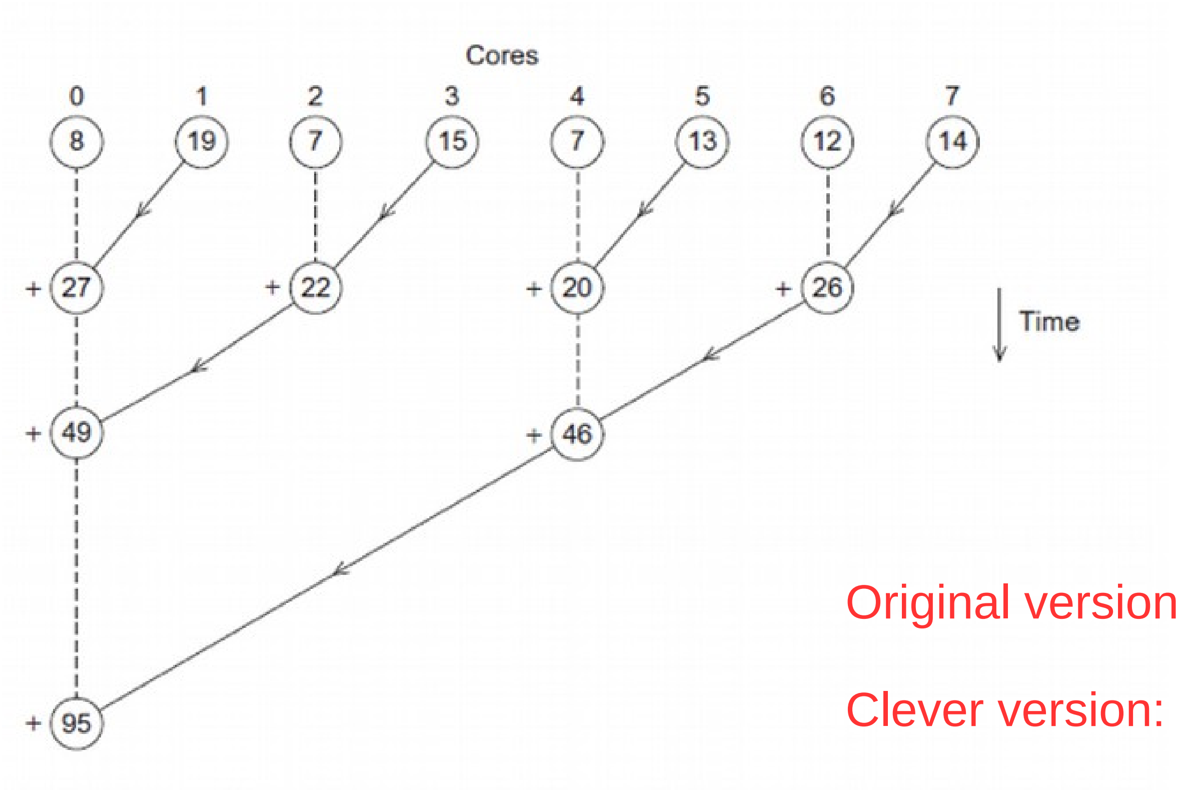
```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}

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*?)

Our goals this semester

- Learn some parallel & distributed programming technologies
  - Pthreads, MPI, OpenMP, Chapel
- 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

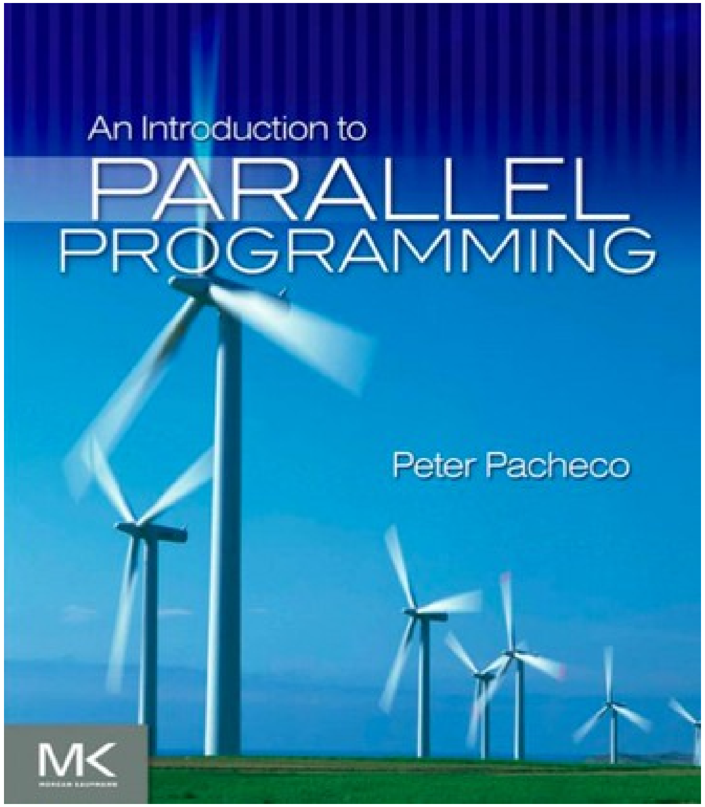
Course format

- Course calendar on website (bookmark it!)
- Resource links on website
- Private files and grades on Canvas
- Canvas quizzes (1-2 per week)
- In-class exercises (1 per week)
  - Canvas submission
- Standard projects (every 2-3 weeks)
  - Piazza Q&A w/ Canvas submission
- Elective project (entire semester)
- In-class exams (midterm & final)

Course textbook

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

- Sources:
  - JMU Bookstore ($65)
  - Amazon ($48)
  - Safari (free, limited sessions)
  - Library (on reserve)



Standard projects

- Practice using parallel and distributed technologies
- Practice introspective software development
- Submission: code + reflection + review
  - Code can be written in teams of two
    - Benefits vs. costs of working in a team
  - Reflection must be individual
  - Graded code reviews after project submission

Elective project

- Semester-long capstone project
  - Teams of 2-4 people
  - Individualized topic (but talk to me early!)
  - Must involve parallel/distributed systems or software
  - Must include significant programming or analysis
  - Preferably uses Pthreads, OpenMP, or MPI
  - Multiple submissions:
    - Proposal, mid-project, poster, final deliverable
  - Graded on progress and application of course concepts
  - Goal: open-ended project experience

My interests

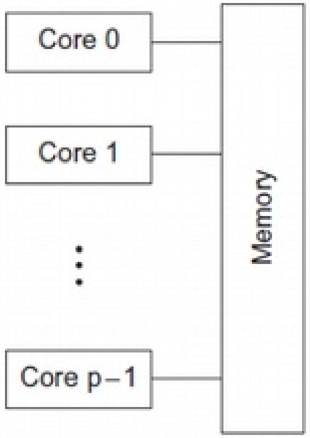
- High-performance computing (*supercomputing*)
- Program instrumentation and analysis
- Floating-point behavior

On the website:

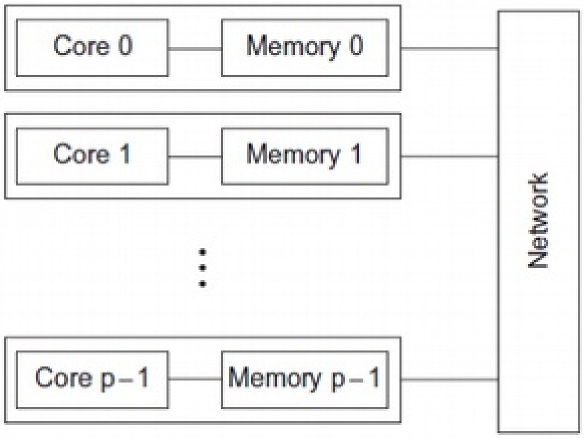
- **HPC internships** at Lawrence Livermore, Los Alamos, and Oak Ridge national labs
- **Student volunteer scholarships** for Supercomputing ‘17 in Denver, CO

Parallel Systems

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



- Distributed
  - Idea: add more **computers**
  - Paradigm: message passing
  - Technologies: MPI, SLURM



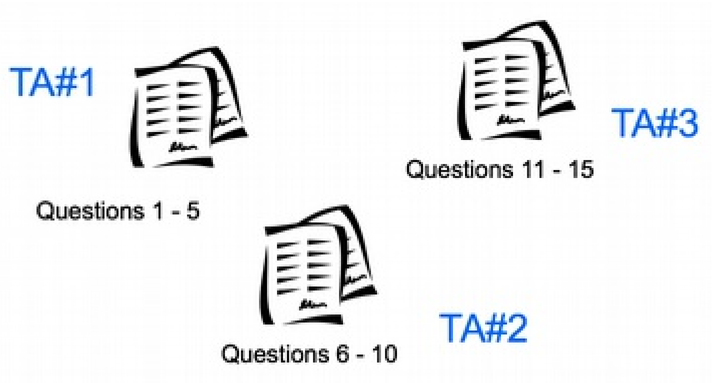
Parallelism

- Task parallelism
  - Partition **tasks** among processes
  - Pass data between processes

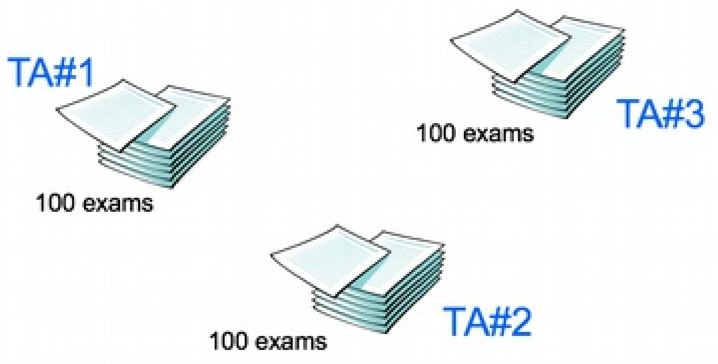
- Data parallelism
  - Partition **data** among processes
  - Each process performs all tasks

Parallelism

- Task parallelism
  - Partition **tasks** among processes
  - Pass data between processes



- Data parallelism
  - Partition **data** among processes
  - Each process performs all tasks



Parallelism example

- Six sections of CS 159 this semester
  - Total of 180 students
- Suppose there is an exam with 15 questions
  - Suppose we have three graders
  - How do we split up the work?
- Two approaches
  - Task parallel: each grader grades 5 questions on all 180 exams
  - Data parallel: each grader grades all questions on 60 exams
  - Latter is better for a distributed system (less communication)

Have a great semester!

- Take course intro survey (free points!)
- Read IPP Ch.1
- Reading quiz tomorrow (shorter than usual)
- Wed & Fri: mini-lecture and exercise
- Start thinking about project groups
- Make sure you can access Piazza
- Make sure you can SSH into `login.cluster.cs.jmu.edu`
  - Must be on JMU network (i.e., go through `stu`)
  - Email me before class on Wednesday if you encounter issues
- Plan on attending Feb 8 speaker series talk