Introduction to Parallel Computing

Chris Kauffman

Last Updated:
Tue Jan 27 01:45:27 PM EST 2026

Logistics

Reading

Grama Ch 1

Goals

- Motivate: Parallel Programming
- Overview concepts a bit
- Discuss course mechanics

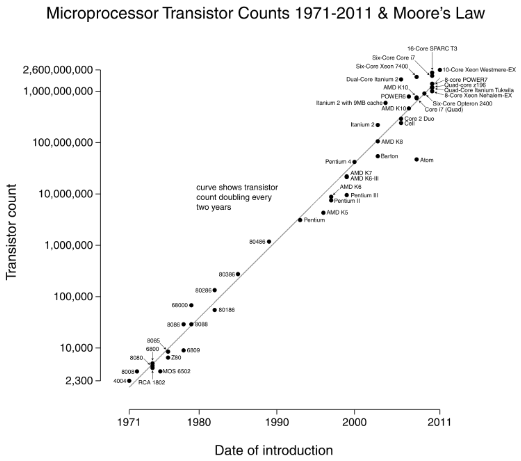
Registered or Not?

- I am afraid I cannot help anyone get registered for the course: lots of requests to do so and in fairness, students must resolve this with the CS advising team on their own
- No additional seats will be added to the course as we are understaffed as is

Sorry to those that want to get in but can't

Moore's Law

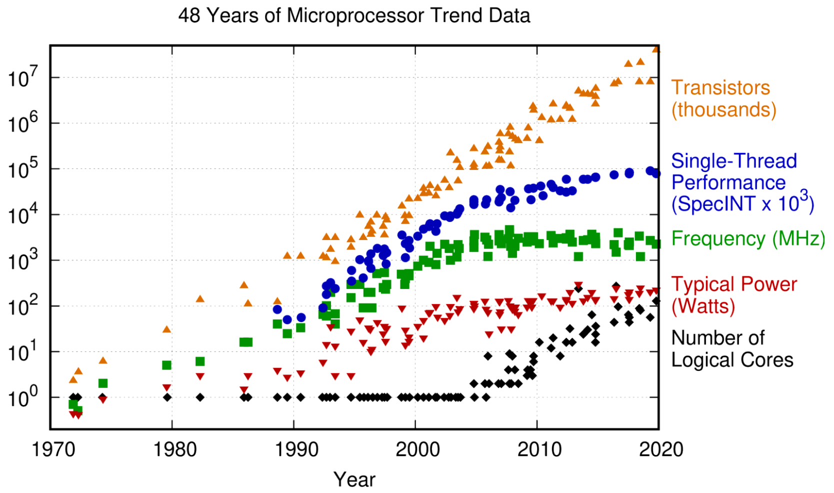
- Smaller transistors → closer together
- Smaller transistors can "flip" faster
- More + faster transistors on a chip → more speed
- **Processor speed doubles every 18 months**



Aside: How Small are Transistors?

- Recent Intel CPUs uses a 14 nanometer manufactoring process
- Distance between memory units in the processor is about 28 nanometers
- A hard sphere radius of a hydrogen Atom is about 0.11 nanometers
- About 255 **atoms** apart - weird things start happening at that scale, quantum things that are less than deterministic
- The term "X-nanometer process" is not a true standard but the point is manufacturers continue increasing Transistor Density BUT *they've just changed what those transistors are used for...*

Moore's Law Alive and Well

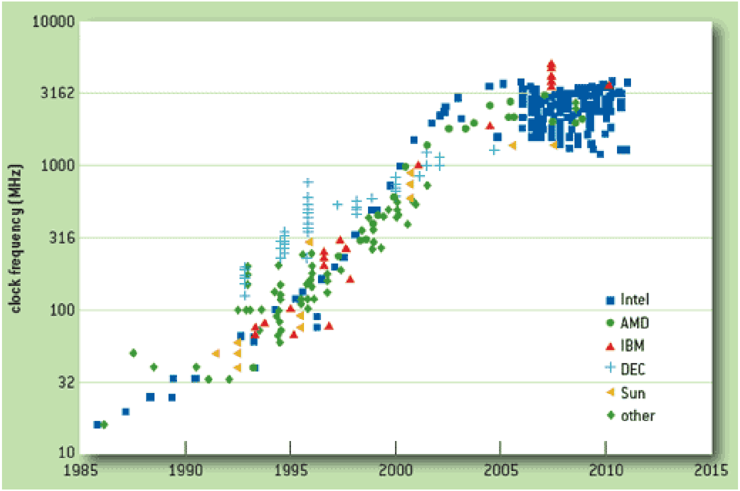


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Source: karlrupp / microprocessor-trend-data on Github

- Still get more transistors (yay!)
- Clock Frequency & Single CPU Speed is stalling (boo!)

Similar Plot on Processor Speed

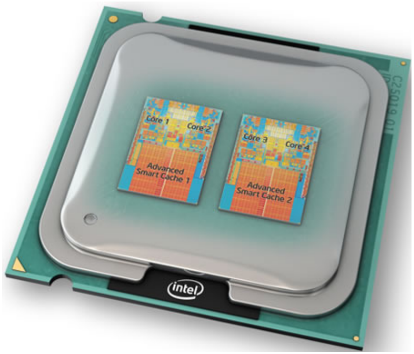


Source: Danowitz et al

- CPU speed isn't getting faster these days
- Companies like Dell don't even list clock frequency anymore, they list **number of cores** in their desktops/laptops

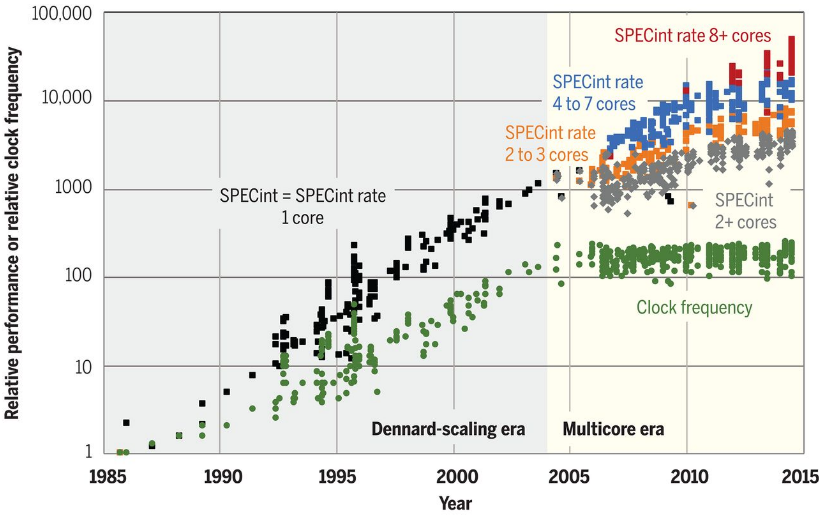
Today's Processors are mostly Parallel Computers

- 1 CPU comes with several "Cores" - independent identical processing units
- Cores have their own ALUs, registers, pipelines
- Cores share some resources like Cache and Comm. hardware, main memory
- Cores can run multiple programs simultaneously
- Cores can **cooperate** within on a single program to (possibly) make that program faster



Parallelism is now the norm in most hardware; coding for that parallel hardware has never been more essential

Performance is Moving to Parallel



Source: Theres plenty of room at the Top: What will drive computer performance after Moores law? by Leiserson et. al, Science 2020

Multitasking

multitask (verb)

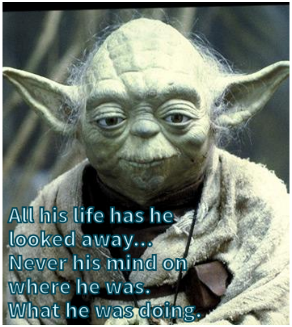
To use the restroom and brush your teeth at the same time.
I was late for work today so I had to multitask!

- Urban Dictionary Definition 1 by sasm

Questions

- Do humans multitask and if so how?
- What kinds of things can humans multitask?
- Are you a good multitasker?
- How do humans get a big job done faster?

Focus



Laptop use lowers student grades, experiment shows, The Canadian Press, 8-14-2013

The students in the first experiment who were asked to multitask averaged 11 per cent lower on their quiz.

The students in the second experiment who were surrounded by laptops scored 17 per cent lower.

Original Paper by Sana et al

Computers and Multitasking

- How do computers multitask?
- Can a single CPU computer multitask?
- What are the advantages/disadvantages of this?
- What might drive one to write a parallel program?

Parallelism Shows Up a **Lot** in Computing

Low-level/hardware parallelism

- CPU Instructions,
- CPU Pipelines
- VLIW: Very Long Instruction Word
- Multi-media CPU instructions
- Graphics and GPU instructions
- Memory subsystem and I/O controller

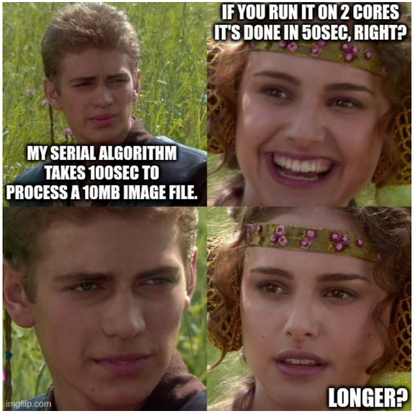
Some of these can be controlled via assembly language choices, others are implicit in the hardware and only indirectly controllable by a programmer.

- We will touch at times on **Vector Instructions** that perform the same calculation on several pieces of data
- We will focus on **Programmatic Parallelism**: writing code that is parallel at a higher level

Primary Motivation for Parallel Computing

Use multiple processors to...

- **Speedup**: solve the same size problem in less time
- **Sizeup**: solve a larger problem in the same time


MY SERIAL ALGORITHM TAKES 100SEC TO PROCESS A 10MB IMAGE FILE.
IF YOU RUN IT ON 2 CORES IT'S DONE IN 50SEC, RIGHT?
LONGER?
imgflip.com

Speedup and Sizeup Counter Examples

Speedup

- **Assertion:** Two people cannot bake a cake twice as fast as a single person... why?
- Any similar examples of a computing task that does not easily break entirely into 2 equal, independent chunks?

Sizeup

- **Assertion:** Two small ovens cannot handle a double-sized cake... why?
- Any equivalent examples of a computing tasks?

Exercise: Scan for Max Serial vs Parallel

- Have an array of large size N with numbers in it
- Want to find the maximum number

```
# Find max serially
# ON CPU 0
my_max = arr[0]
for i=1 to N-1:
  if my_max < arr[i]:
    my_max = arr[i]
done
return my_max
```

- If you were allowed both CPU 0 **and** CPU 1, could you make this go faster?
- State any assumptions you make about how CPU 0 & 1 interact / access data in `arr[]`.
- Try writing down your instructions for CPU 0 and 1 on a piece of paper.

Answers: Scan for Max

Assumption: both CPUs can access all of arr[]

```
# ON CPU 0
my_max = arr[0]
for i=1 to N/2-1:
  if my_max < arr[i]:
    my_max = arr[i]
done

other_max = receive number
            from CPU 1
if other_max > my_max:
  my_max = other_max

return my_max
```

```
# ON CPU 1
my_max = arr[N/2]
for i=N/2+1 to N-1:
  if my_max < arr[i]:
    my_max = arr[i]
done
send my_max to CPU 0
```

Final comparison steps cannot be parallelized - creates a limit (albeit mild) on how much can be done in parallel.

Amdahl's Law and the Cost of Coordination

Likely that Parallel Max version completes in around 50% of time of the Serial Max, but not quite 2X speedup.

Amdahl's Law

Speedup is limited by the portion of the program that can be parallelized and the degree to which that portion can be parallelized

- i.e. All algorithms will contain some serial portion that cannot be parallelized
- Ensures near linear speedup will never become perfect speedup

Additional Overhead

- All parallel algorithms feature some communication or coordination overhead that is not present in serial versions
- When parallelizing serial algorithms, often tension: splitting a task up induces communication requirements, must balance these carefully

Concurrency and Parallelism

StackOverflow Question

Concurrency vs Parallelism - What is the difference?

Accepted Answer (RichieHindle)

Concurrency is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. Eg. multitasking on a single-core machine.
Parallelism is when tasks literally run at the same time, eg. on a multicore processor.

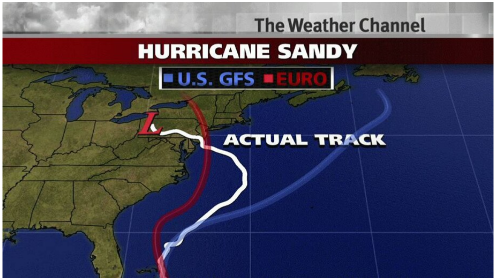
Implications

$Concurrent \not\Longrightarrow Parallel$: A concurrent program does not need to be run in parallel (but it often is).

$Parallel \Longrightarrow Concurrent$: A parallel program must have concurrent parts and had better deal with concurrency issues.

Speed to Completion Matters

By 2015 the GFS model had fallen behind the accuracy of other global weather models. This was most notable in the GFS model incorrectly predicting Hurricane Sandy turning out to sea until four days before landfall, while the European Centre for Medium-Range Weather Forecasts' model predicted landfall correctly at 7 days. Much of this was suggested to be due to limits in computational resources within the National Weather Service. In response, the NWS purchased new supercomputers, increasing processing power from 776 teraflops to 5.78 petaflops. In 2018, the processing power was increased again to 8.4 petaflops.
– Wikip: Global Forecast System



Source: How Hurricane Sandy sprung weather models into the mainstream, Washington Post, 19-Oct-2022

Getting correct predictions in time for action is important in many disciplines - parallel computing provides the mechanism for this.

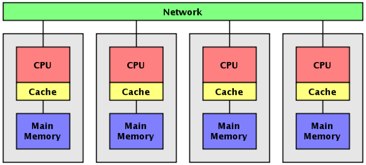
What We Will Discuss

- Low latency parallel computers
- General hardware/network architectures of parallel computers
- Distributed memory parallel computers
  - Message Passing Interface (MPI)
- Shared Memory parallel computers
  - POSIX Threads and OpenMP
- Co-processor Parallelism
  - GPGPU: General Purpose Graphics Processing Units
  - CUDA for NVIDIA GPU programming
- Analyzing Parallel Algorithms
- Parallelizing Classic Algorithms like Matrix Ops, Sorting, Stats
- *Scientific computing* angle

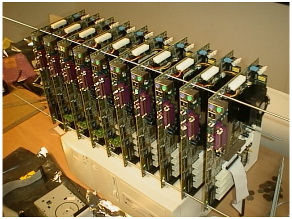
General Arrangement of Topics

Weeks 1-7

- Basics + Theory
- Distributed Memory



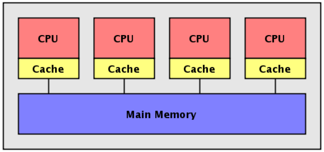
Source: Kaminsky/Parallel Java



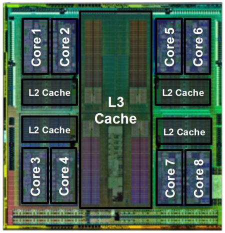
Source: ClusterComputer.com

Weeks 8-12

- Shared Memory
- GPGPU / CUDA Programming



Source: Kaminsky/Parallel Java



Source: Bit-Tech AMD FX-8350 review

What We Won't Discuss

- Server/Client Model for Web Programming (CMSC335)
- General Networking (CMSC414)
- Deep Dive into Hardware Parallelism (CMSC411)
- Graphics applications for GPUs (CMSC427)
- Database/Distributed/Grid Computing (CMSC424)

Course Mechanics

Mull over the syllabus linked to our Canvas Page

Look for

- Contact info for course staff
- Textbook
- Weights / Grading Scheme

Schedule of topics and lecture materials:
https://cs.umd.edu/~profk/416/schedule.html

Assignments and Hardware

- Assignments are 20% of your grade
- Combination of writing and programming
- **Free Collaboration** allowed, submit Alone or in Pairs
- Programming mostly in **C in a Linux environment**
  - If you have never programmed in C before, drop the course
  - If syntax like `malloc()` or `int *p = &s.field;` is unfamiliar, begin reviewing **intensively**
  - If unfamiliar with `make` and command line, **get familiar quick**
  - Assignment 1 will have a C coding portion which indicates expected proficiency level
  - Consult me during office hours if in doubt
- Run Parallel codes on Zaratan HPC Cluster
- 4 Assignments roughly on
  1. Basics + Theory + C Review
  2. MPI Coding + Analysis
  3. Performance Analysis
  4. Shared Memory/OpenMP + CUDA / GPGPU coding

Exams Small and Large

Mini-Exams: 4 scheduled, 60% of grade

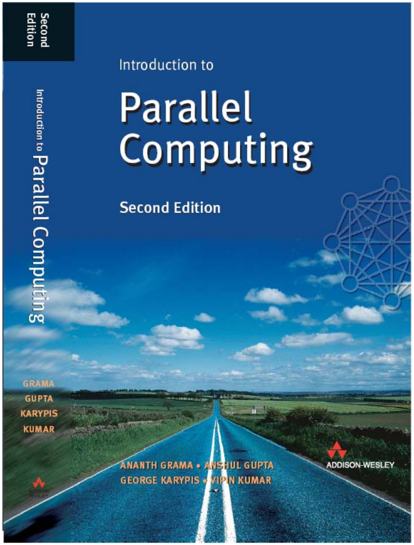
- Between a quiz and a midterm exam in length
- 30 minutes during lecture
- 1 page, front and back
- Topics will cover stuff from lecture, assignments, readings
- Will work practice problems in class preceding them
- General Pattern
  - Week N: Assignment Due on Topic X
  - Week N+1: Mini-Exam on Topic X

Final Exam: 2 hours, 20% of grade

Open Resource Exams

Unless otherwise specified, exams are **Open Resource**: use notes, compiler, code, slides, textbook. No googling, browsing, communication, AI, cheating

Reading For Next Time



Reading: Grama Ch 2

- **Focus on 2.3-5**, material pertaining to distributed memory
- We will return to shared memory arch later in the course
- Cache Coherence, PRAM models, False Sharing, Memory Bus are all shared memory topics
- Sections 2.1 and 2.2 optional, deeper architectures
- Sections 2.6 and 2.7 encouraged, deeper on networks