Introduction to Parallel Computing

Chris Kauffman

Last Updated:
Fri Jan 13 05:54:52 PM CST 2023

Logistics

Reading

Grama Ch 1

Goals

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

Registered or Not?

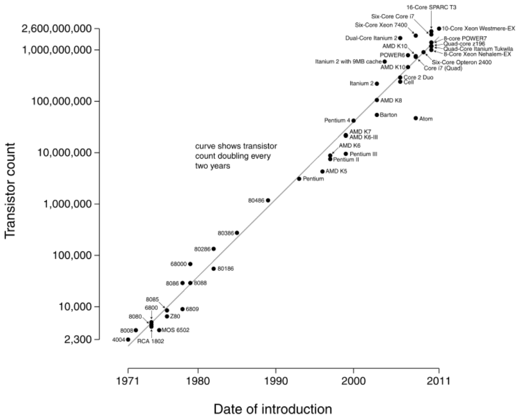
If you are **not registered** for the course but want to be...

- Come to the first week of Lecture so you don't fall behind
- Write on a piece of paper the following information
  1. Name, UMN Email address, Student ID Number
  2. Which Lecture and Lab section you want to register for
  3. 2-3 sentences about why you absolutely must take 2021 this semester, consequences if you do not
- Give me that sheet of paper
- Wait and hope: very limited space + waitlists for full labs

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**

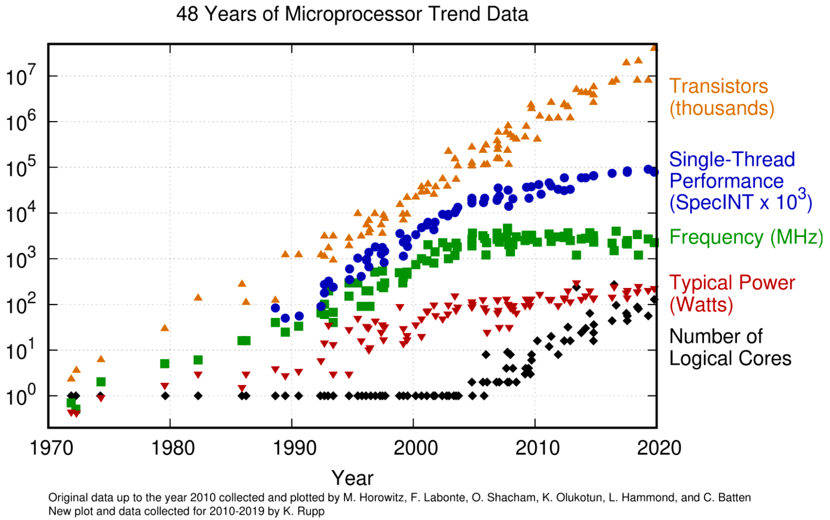
Microprocessor Transistor Counts 1971-2011 & Moore's Law



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
- The term "X-nanometer process" is not a true standard but the point is manufacturers continue increase Transistor Density

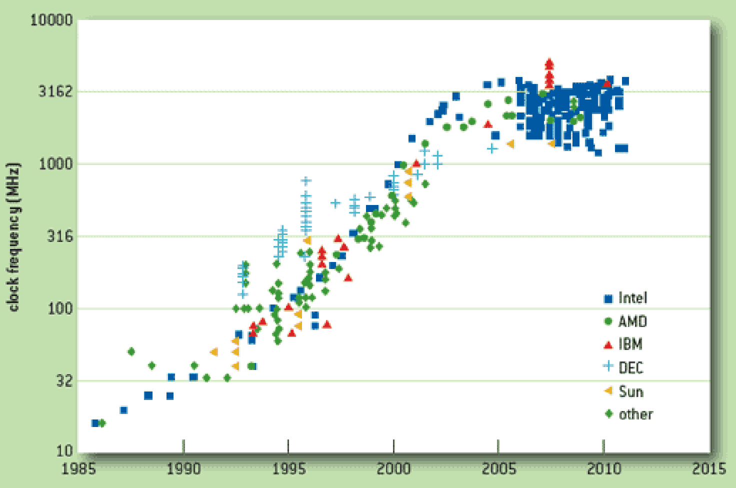
Moore's Law Alive and Well



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



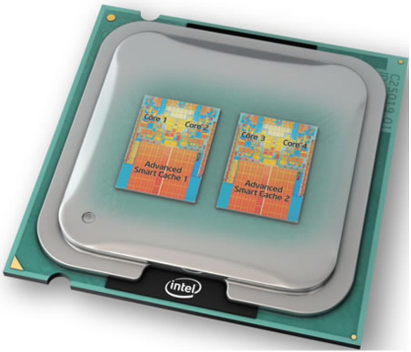
Source: Danowitz et al

- CPU speed isn't getting faster these days
- Fastest Dell Speed I found was 4.8 GhZ on "Turbo" (my laptop caps at 3.4 GhZ)

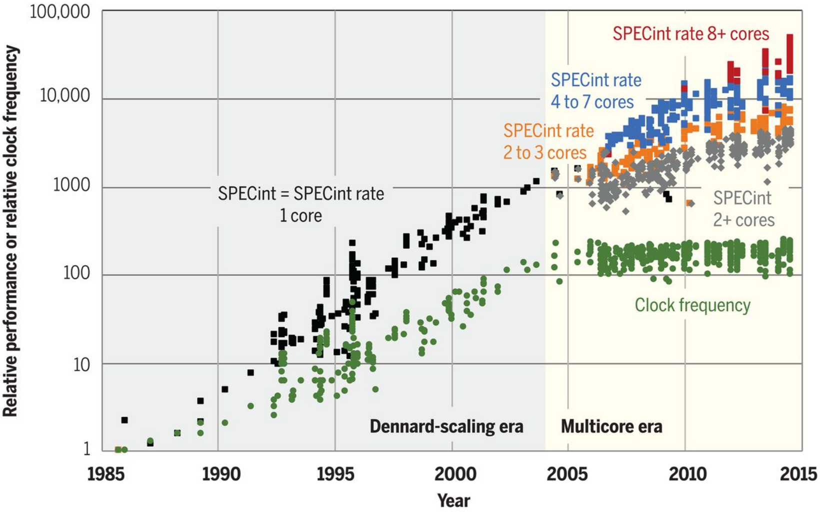
Today's Processors

Mini-Parallel Computer

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



Performance is Moving to Parallel



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

Multitasking

multitask (verb)

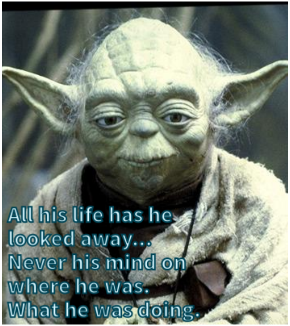
To use the restroom and brush your teeth at the same time.
I was late for work today so I had to multitask!

- Urban Dictionary Definition 2 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,
- VLIW: Very Long Instruction Word
- Multi-media CPU instructions
- Graphics and GPU instructions
- CPU Pipelines
- Memory subsystem
- I/O controller

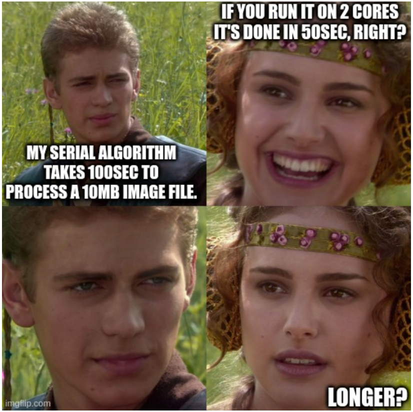
Some of these can be controlled via assembly language choices, others are implicit in the hardware and only indirectly controllable by a programmer.

- We will touch at times on **Vector Instructions** that to 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



Speedup and Sizeup Counter Examples

Speedup

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

Sizeup

- Two 2 ovens cannot handle a double-sized cake...
- 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 never quite a perfect 2X speedup. This has a formalization.

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
- Additionally, all parallel algorithms feature some communication overhead that is not present in serial versions
- Together these ensure near linear speedup will never become perfect speedup

When parallelizing serial algorithms, will pay careful attention to these factors.

Concurrency and Parallelism

StackOverflow Questions

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*

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

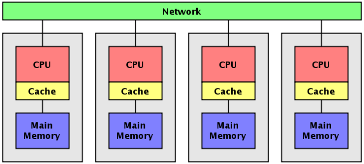
What We Will Discuss

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

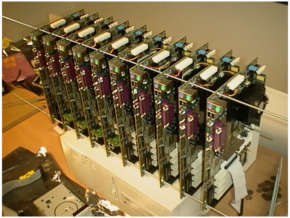
General Arrangement of Topics

Weeks 1-7

- Basics + Theory
- Distributed Memory



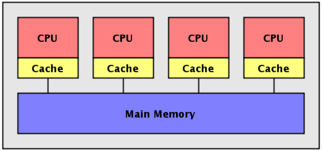
Source: Kaminsky/Parallel Java



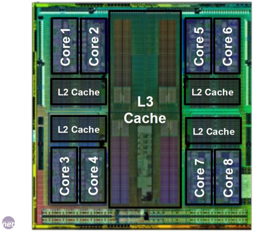
Source: ClusterComputer.com

Weeks 8-12

- Shared Memory
- GPGPU / CUDA Programming



Source: Kaminsky/Parallel Java



Source: Bit-Tech AMD FX-8350 review

What We Won't Discuss

- General Networking (CSCI 4211/5211)
- Server/Client Model for Web Programming (CSCI 4131)
- Deep Dive into Parallel Hardware (CSCI 5204)
- Graphics applications for GPUs (CSCI 4611 / 5607+8)
- Distributed/Grid Computing (CSCI 5105 / 5751)

Course Mechanics

Mull over the syllabus linked to our Canvas Page
Look for

- Contact info
- GTA
- Textbook
- Weights / Grading Scheme

Schedule of topics and lecture materials:
https://www-users.cse.umn.edu/~kauffman/5451/schedule.html

Assignments and Hardware

- Assignments are 50% of your grade
- Combination of writing and programming
- Students work Alone or in Pairs
- Programming mostly in **C in a Linux environment**
  - If you have never programmed in C before, begin studying it **intensively** or 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 an optional coding portion which indicates level of coding to expect
  - Consult me during office hours if in doubt
- Run Parallel codes on CSE Labs and MSI machines for eval
- 4 Assignments roughly on
  1. Basics + Theory
  2. MPI Coding + Analysis
  3. Shared Memory Coding via PThreads / OpenMP
  4. CUDA / GPGPU coding

Exams Small and Large

Mini-Exams: 3 scheduled, 30% of grade

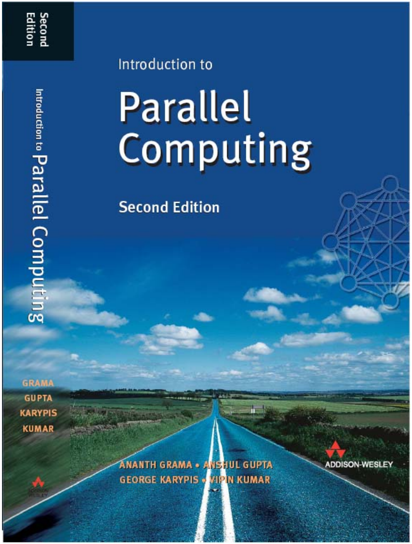
- Between a quiz and a midterm exam in length
- 30 minutes during lecture
- 1 page, front and back
- Topics will cover stuff from lecture, assignments, readings
- Will work practice problems in class preceding them

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, 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