CS 5220

Introduction and Performance Basics

David Bindel

2024-08-27

Logistics

Title: *Applied High-Performance and Parallel Computing*

Web: https://www.cs.cornell.edu/courses/cs5220/2024fa

When: TR 1:25-2:40

where: Gates G01

Who: David Bindel, Caroline Sun, Evan Vera

Enrollment

https://www.cs.cornell.edu/courseinfo/enrollment

FA24 Add/Drop Announcement

- CS limits pre-enrollment to CS MEng students.
- We almost surely will have enough space for all comers.
- Enroll if you want access to class resources.
- Enrolling as an auditor is OK.
- If you will not take the class, please formally drop!

Prerequisites

Basic logistical constraints:

- Class codes will be in C and C++
- Our focus is numerical codes

Fine if you're not a numerical C hacker!

- I want a diverse class
- Most students have *some* holes
- Come see us if you have concerns

Objectives: Performance sense

Reason about code performance

- Many factors: HW, SW, algorithms
- Want simple “good enough” models

Objectives: Learn about HPC

Learn about high-performance computing (HPC)

- Learn parallel concepts and vocabulary
- Experience parallel platforms (HW and SW)
- Read/judge HPC literature
- Apply model numerical HPC patterns
- Tune existing codes for modern HW

Objectives: Numerical SWE

Apply good software practices

- Basic tools: Unix, VC, compilers, profilers, …
- Modular C/C++ design
- Working from an existing code base
- Testing for correctness
- Testing for performance
- Teamwork

Lecture Plan: Basics

- Architecture
- Parallel and performance concepts
- Locality and parallelism

Lecture Plan: Technology

- C/C++ and Unix fundamentals
- OpenMP, MPI, CUDA and company
- Compilers and tools

Lecture Plan: Patterns

- Monte Carlo
- Dense and sparse linear algebra
- Partial differential equations
- Graph partitioning and load balance
- Fast transforms, fast multipole

Coursework: Lecture (10%)

- Lecture = theory + practical demos
    - 60 minutes lecture
    - 15 minutes mini-practicum
    - Bring questions for both!
- Notes posted in advance
- May be prep work for mini-practicum
- Course evaluations are also required!

Coursework: Homework (15%)

- Five individual assignments plus "HW0"
- Intent: Get everyone up to speed
- Assigned Tues, due one week later

Homework 0

- Posted on the class web page.
- Complete and submit by CMS by 9/3.

Coursework: Group projects (45%)

- Three projects done with partners (1–3)
- Analyze, tune, and parallelize a baseline code
- Scope is 2-3 weeks

Coursework: Final project (30%)

- Groups are encouraged!
- Bring your own topic or we will suggest
- Flexible, but *must* involve performance
- Main part of work in November–December

Palate Cleanser

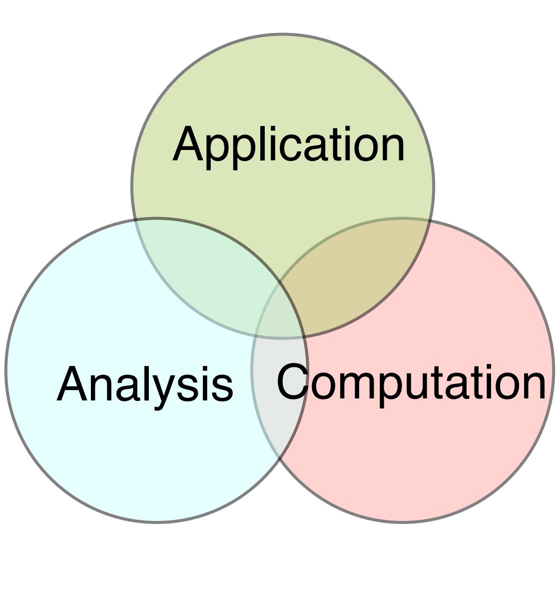
Hello, world!

Introduce yourself to a neighbor:

- Name
- Major / academic interests
- Something fun you have recently read or watched
- Hobbies

Jot down answers (part of HW0).

The Good Stuff


Application
Analysis
Computation

Applications Everywhere!

- Climate modeling
- CAD tools (computers, buildings, airplanes, …)
- Computational biology
- Computational finance
- Machine learning and statistical models
- Game physics and movie special effects
- Medical imaging
- …

- Need for *speed* and for *memory*
- Many processors working *simultaneously* on *same* problem
    - vs concurrency (about *logical structure* vs *performance*)
    - or distributed systems (coupled but distinct problems, clients and servers are often at different locations)

Why Parallel Computing?

Scientific computing went parallel long ago:

- Want an answer that is right enough, fast enough
- Either of those might imply a lot of work!
- … and we like to ask for more as machines get bigger
- … and we have a lot of data, too

Why Parallel Computing?

Today: Hard to get non-parallel hardware!

- How many cores are in your laptop?
- How many in NVidia's latest accelerator?
- Biggest single-node EC2 instance?

Organizational Basics

- *Cores* packaged together on *CPUs*
    - Cores have *instruction-level* parallelism (e.g. vector units)
- *Memory* of various types (memory hierarchy)
- *Accelerators* have similar pieces, organized differently
- CPUs and accelerators packaged together in *nodes*
- Nodes often connected in *racks*
- *Networks* (aka *interconnect* or *fabric*) connecting the pieces

How Fast Can We Go?

Speed records for Linpack benchmark

https://www.top500.org

Speed measured in flop/s (floating point ops / second):

- Giga (10^9) – a single core
- Tera (10^{12}) – a big machine
- Peta (10^{15}) – current top 10 machines
- Exa (10^{18}) – favorite of funding agencies

What do these machines look like?

An alternate benchmark: Graph 500

- Data-intensive graph processing benchmark
- Metric is traversed edges per second (TEPS)
- How do the top machines for Linpack and Graph 500 compare?

What do these machines look like?

What HW and How Fast?

- Some high-end machines look like high-end clusters
  - Except custom networks.
- Achievable performance is
  - $\ll$ peak performance
  - Application-dependent
- Hard to achieve peak on more modest platforms, too!

Parallel Performance in Practice

So how fast can I make my computation?

- Peak $>$ Linpack $>$ Gordon Bell $>$ Typical
- Measuring performance of real applications is hard
  - Even figure of merit may be unclear (flops, TEPS, ...?)
  - Typically a few bottlenecks slow things down
  - And figuring out why they slow down can be tricky!
- And we *really* care about time-to-solution
  - Sophisticated methods get answer in fewer flops
  - ... but may look bad in benchmarks (lower flop rates!)

See also David Bailey's comments:

- Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers (1991)
- Twelve Ways to Fool the Masses: Fast Forward to 2011 (2011)

Example: Reduction

How can we speed up summing an array of length n with $p \leq n$ processors?

- Theory: $n/p + O(\log(p))$ time with *reduction tree*
- Is this realistic?

Quantifying Parallel Performance

- Starting point: good *serial* performance
- Strong scaling: compare parallel to serial time on the same problem instance as a function of number of processors (p)

$$\text{Speedup} = \frac{\text{Serial time}}{\text{Parallel time}}$$
$$\text{Efficiency} = \frac{\text{Speedup}}{p}$$

Barriers

Ideally, speedup = p. Usually, speedup $< p$.

Barriers to perfect speedup:

- Serial work (Amdahl's law)
- Parallel overheads (communication, synchronization)

Amdahl's Law

$$
\begin{aligned}
p &= \text{number of processors} \\
s &= \text{fraction of work that is serial} \\
t_s &= \text{serial time} \\
t_p &= \text{parallel time} \geq s t_s + (1-s) t_s / p
\end{aligned}
$$

Amdahl's law:

$$
\text{Speedup} = \frac{t_s}{t_p} = \frac{1}{s + (1-s)/p} > \frac{1}{s}
$$

So 1% serial work $\implies$ max speedup < $100\times$, regardless of p.

A Little Experiment

Let's try a simple parallel attendance count:

- **Parallel computation:** Rightmost person in each row counts number in row.
- **Synchronization:** Raise your hand when you have a count
- **Communication:** When all hands are raised, each row representative adds their count to a tally and says the sum (going front to back).

(Somebody please time this.)

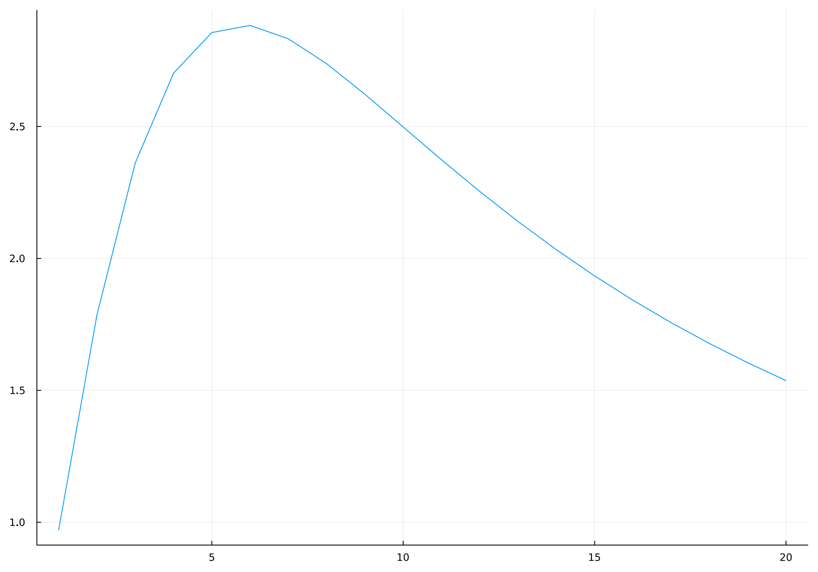
A Toy Analysis

Parameters:

$$
\begin{aligned}
n &= \text{number of students} \\
r &= \text{number of rows} \\
t_c &= \text{time to count one student} \\
t_t &= \text{time to say tally} \\
t_s &\approx n t_c \\
t_p &\approx n t_c / r + r t_t
\end{aligned}
$$

How much could I possibly speed up?

Modeling Speedup



(Parameters: $t_c = 0.3, t_t = 1, n = 111$.)

Modeling Speedup

Mostly-tight bound:

$$\text{speedup} < \frac{1}{2}\sqrt{\frac{nt_c}{t_t}}$$

Poor speed-up occurs because:

- The problem size n is small
- The communication cost is relatively large
- The serial computation cost is relatively large

Some of the usual suspects for parallel performance problems!

Weak scaling?

Things would look better if I allowed both n and r to grow — that would be a *weak* scaling study.

This probably does not make sense for a classroom setting…

Summary: Parallel Performance

Today:

- We're approaching machines with peak *exaflop* rates
- But codes rarely get peak performance
- Better comparison: tuned serial performance
- Common measures: *speedup* and *efficiency*
- Strong scaling: study speedup with increasing p
- Weak scaling: increase both p and n
- Serial overheads and communication costs kill speedup
- Simple analytical models help us understand scaling

And in case you arrived late

http://www.cs.cornell.edu/courses/cs5220/2024fa/

… and please enroll and submit HW0!