CS 5220: Introduction

David Bindel
2017-08-22
http://www.cs.cornell.edu/courses/cs5220/2017fa/

Time: TR 8:40–9:55
Location: Gates G01
Instructor: David Bindel (bindel@cs)
TA: Eric Hans Lee (erichanslee@cs)
Enrollment

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

• Many CS classes (including 5220) limit pre-enrollment to ensure majors and MEng students can get in.
• 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!
The Computational Science & Engineering Picture

Application

Analysis

Computation
These tools are used in more places than you might think:

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

Parallel computing shows up in all of these.
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

- Today: Hard to get a non-parallel computer!
  - Totient nodes (2015): 12-core compute nodes
  - Totient accelerators (2015): 60-core Xeon Phi 5110P
  - My laptop (late 2013): Dual core i5 + built in graphics

- Cluster access ≈ internet connection + credit card
Roughly three parts:

1. **Basics:** architecture, parallel concepts, locality and parallelism in scientific codes
2. **Technology:** OpenMP, MPI, CUDA/OpenCL, cloud systems, compilers and tools
3. **Patterns:** Monte Carlo, dense and sparse linear algebra and PDEs, graph partitioning and load balancing, fast multipole, fast transforms
Objectives

• Reason about code performance
  • Many factors: HW, SW, algorithms
  • Want simple “good enough” models

• 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

• Apply good software practices
Basic logistical constraints:

- Default class codes will be in 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
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
Coursework: Small group assignments (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
Homework 0

- Posted on the class web page.
- Complete and submit by CMS by 8/29.
Questions?
How Fast Can We Go?

Speed records for the Linpack benchmark:

http://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 (5 in US)
- Exa ($10^{18}$) – favorite of funding agencies
Current Record: China’s Sunway TaihuLight

- 93 petaflop/s (125 petaflop/s peak)
- 15 MW (LAPACK) – relatively energy efficient
  - Does not include custom chilled-water cooling unit
- Based on SW26010 manycore RISC processors
  - Management processing element (CPE) = 64-bit RISC core
  - Computer processing element (CPE) = 8 × 8 core mesh
  - Custom interconnect
  - Sunway Raise OS (Linux)
  - Custom compilers (Sunway OpenACC)
Performance on TaihuLight (Dongarra, June 2016)

- Theoretical peak: 125.4 petaflop/s
- Linpack: 93 petaflop/s (74% peak)
- Three SC16 Gordon Bell finalists
  - Explicit PDE solves: 30–40 petaflop/s (25–30%)
  - Implicit solver: 1.5 petaflop/s (1%)
  - Numbers taken from June 2016, may have improved
  - Even with improvements: peak is not indicative!
Commodity nodes, custom interconnect:

- Nodes consist of Xeon E5-2692 + Xeon Phi accelerators
- Intel compilers + Intel math kernel libraries
- MPICH2 MPI with customized channel
- Kylin Linux
- TH Express-2
Alternate Benchmark: Graph 500

Graph processing benchmark (data-intensive)

- Metric: traversed edges per second (TEPS)
- K computer (Japan) tops the list (38.6 teraTEPS)
- Sunway TaihuLight is second (23.8 teraTEPS)
- Tianhe-2 is at 8 (2.1 teraTEPS)
• 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!
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)
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 ($p$)

\[
\text{Speedup} = \frac{\text{Serial time}}{\text{Parallel time}}
\]

\[
\text{Efficiency} = \frac{\text{Speedup}}{p}
\]

• Ideally, speedup = $p$. Usually, speedup < $p$.
• Barriers to perfect speedup
  • Serial work (Amdahl’s law)
  • Parallel overheads (communication, synchronization)
Amdahl’s Law

Parallel scaling study where some serial code remains:

\[ p = \text{number of processors} \]
\[ s = \text{fraction of work that is serial} \]
\[ t_s = \text{serial time} \]
\[ t_p = \text{parallel time} \geq s t_s + (1 - s) t_s / p \]

Amdahl’s law:

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

So 1% serial work \( \rightarrow \) max speedup < 100\( \times \), regardless of \( p \).
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:

\[ n = \text{number of students} \]
\[ r = \text{number of rows} \]
\[ t_c = \text{time to count one student} \]
\[ t_t = \text{time to say tally} \]
\[ t_s \approx nt_c \]
\[ t_p \approx nt_c/r + rt_t \]

How much could I possibly speed up?
Modeling Speedup

(Parameters: $n = 80$, $t_c = 0.3$, $t_t = 1$.)
Modeling Speedup

The bound

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

is usually tight.

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!

Things would look better if I allowed both \( n \) and \( r \) to grow — that would be a \textit{weak} scaling study.
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/cs5220/2017fa/

... and please enroll and submit HW0!