Welcome to CS639!
Undergraduate Topics In Computing: Parallel and Throughput-Optimized Programming
Spring 2020, 2:30-3:45 Tue/Thu
Today’s lecture

• Introduce the content and motivation for this class
  • What is the technological context?
  • Why care about optimized parallel programming?
  • What methods and applications will we target?

• Discuss class logistics, expected work from students, prerequisites, grading and instructional support

• A few samples of throughput-conscious applications

• Get a first sample of the type of questions we will be addressing, and our design/evaluation philosophy
About the instructor

• Education
  • BSc CS ('00), BSc Math ('02) - Univ of Crete, Greece
  • PhD Computer Science ('07) - Stanford
  • PostDoc CS & Applied Math (07-10) - UCLA
  • At UW since early 2011

• Classes taught:
  • Undergrad : Computer Graphics,
    Intro to Numerical Methods (and now this class!)
  • Graduate : Advanced Computer Graphics,
    Physics-Based Modeling & Simulation
About the instructor

- Research Interests
  - Physics-Based Modeling
  - Digital Humans
  - Visual Simulation (natural phenomena)
  - Biomechanics
  - Visual effects
  - Fast math (in general)

- Current Company affiliations
  - Weta Digital, Disney Research|Studios
About the instructor

- Research Interests

(we’ll talk about some of these later)
Motivation for this class

• Modern computers have vastly higher performance potential

  • Modern desktop vs. 2010’s cluster
  • $30k Server today vs. 2012-era supercomputer
  • Drastically improved opportunities for high-performance computation

• Performance potential vs. guarantee

  • Programming such platforms is much more intricate (you need to be very conscious of platform quirks)
  • Accelerating legacy algorithms is highly nontrivial
  • Hardware and APIs becoming more specialized
Motivation for this class

• Should you care? (i.e. is this your problem to figure out?)

  • Everybody cares about performance and scalability (better productivity, improved capabilities, improved user experience)

  • If you are a developer of a performance-sensitive software library or application, chances are you should care (too much performance left on the table if you don’t)

  • If you are predominantly a user of optimized APIs and software libraries, you might not need to get hand dirty … … still a valuable skill to be able to understand what a modern platform should be able to do for you (even if somebody else implements it)
Motivating Applications

• What motivated *your instructor* to offer this class? (although somewhat “outside” my core research area …)

  • A distillation of lessons learned through nearly 20 years of development of scale/performance-sensitive applications

  • *(Selfish reason)* The skills that I wish new students working for me could easily obtain through a formal class

• In the new era of parallel computing *(post-Moore’s law)* performance advances require buy-in from application specialists vs. just better compilers & hardware *(this is shaping to be a near-consensus view …)*
How did this journey start ...?
How did this journey start ... ?
How did this journey start ... ?
How did this journey start ... ?
How did this journey start ... ?

Production Rig  Our Method
How did this journey start ...?
How did this journey start ...?

Turbulent Sediment Transport
How did this journey start ... ?
How did this journey start ... ?
Tech concatenation, or symbiotic interplay?

Conceptual Phenomenon

- Discretization
- Data structures
- Geometrical modeling
- Quantitative modeling (e.g. continuous PDE)
- Numerical algorithms
- Algorithmic accelerations
- Hardware optimization

Visual Content

Software engineering, programming models, computation delivery ...
Performance : Incremental benefit or critical feature?
Tech concatenation, or symbiotic interplay?

- Conceptual Phenomenon
- Discretization
- Data structures
- Algorithmic accelerations
- Visual Content
- Quantitative modeling (e.g. continuous PDE)
- Geometrical modeling
- Numerical algorithms
- Hardware optimization
The best serial algorithm does not always lead to the best parallel algorithm. While the initial implementation of a key module in fluid simulation used one algorithm, our best parallelization of the module uses an alternative algorithm. The alternative is 57% slower when serial, but has significantly more thread-level parallelization.

Some modules have high on-die and off-die bandwidth usage. Many of the streaming modules also perform relatively little computation per element per invocation or iteration. This results in high bandwidth usage. Five modules have on-die communication-to-computation ratios of at least one byte per ALU operation, and three have off-die ratios exceeding 0.5 bytes per ALU operation. Since CMPs have a large number of threads sharing both the on- and off-die bandwidth, the off-die bandwidth usage of some of the modules is so high, especially for PCG, that it is likely to limit parallel scalability on most current and near-future systems. On our simulated system, assuming a 3GHz clock, PCG uses an average of 64GB/s of main memory bandwidth for 64 threads. The average bandwidth usage for each of the applications is significantly lower than the peak bandwidth usage. However, the scaling of a worst-case module can...
Well-intended evaluation practices ...

“My serial implementation of algorithm X on machine Y ran in Z seconds. When I parallelized my code, I got a speedup of 15x on 16 cores ...”

... are sometimes abused like this:

“... when I ported my implementation to CUDA, this numerical solver ran 200 times faster than my original MATLAB code ...”

(frequent culprit: flawed understanding of how the computing platform works @ low level)
Watch for warning signs:

- Speedup across platforms grossly exceeding specification ratios
  - e.g. NVIDIA Titan RTX vs. Intel Cascade Lake Xeon

Are we pursuing the right efficiency?
Watch for warning signs:

- Speedup across platforms grossly exceeding specification ratios
  - e.g. NVIDIA Titan RTX vs. Intel Cascade Lake Xeon
- Relative (peak) specifications:
  - GPU has about 6x higher (peak) compute capacity
  - GPU has about 4x higher (peak) memory bandwidth
Watch for warning signs:

- Speedup across platforms grossly exceeding specification ratios
  - e.g. NVIDIA Titan RTX vs. Intel Cascade Lake Xeon
- Relative (peak) specifications:
  - GPU has about 6x higher (peak) compute capacity
  - GPU has about 4x higher (peak) memory bandwidth
- Significantly higher speedups likely indicate:
  - Different implementations on the 2 platforms
  - Baseline code was not optimal/parallel enough
Watch for warning signs:

- Speedup across platforms grossly exceeding specification ratios
  - e.g. NVIDIA Titan RTX vs. Intel Cascade Lake Xeon
- Relative (peak) specifications:
  - GPU has about 6x higher (peak) compute capacity
  - GPU has about 4x higher (peak) memory bandwidth
- Significantly higher speedups likely indicate:
  - Different implementations on the 2 platforms
  - Baseline code was not optimal/parallel enough
- “Standard” parallelization yields linear speedups on many cores
  - [Reasonable scenario] Implementation is CPU-bound
  - [Problematic scenario] Implementation is CPU-wasteful
A different perspective …

“… after optimizing my code, the runtime is about 5x slower than the best possible performance that I could expect from this machine …”

… i.e. 20% of maximum theoretical efficiency!

Challenge: How can we tell how fast the best implementation could have been? (without implementing it …)
Example: Solving the quadratic equation

\[ ax^2 + bx + c = 0 \]

What is the \textit{minimum} amount of time needed to solve this?

**Data access cost bound**

“We cannot solve this faster than the time needed to read \(a, b, c\) and write \(x\)”

“We cannot solve this faster than the time needed evaluate the polynomial, for given values of \(a, b, c\) and \(x\)” (i.e. 2 ADDs, 2 MULTs plus data access)

**Solution verification bound**

**Equivalent operation bound**

“We cannot solve this faster than the time it takes to compute a square root”
What about linear systems of equations?

Ax = b

"Textbook Efficiency" (for certain types of problems)

It is theoretically possible to compute the solution to a linear system (with certain properties) with a cost comparable to 10x the cost of verifying that a given value x is an actual solution

It is theoretically possible to compute the solution to a linear system at 10x the cost of computing the r=b-Ax and verifying that r=0

Are we pursuing the right efficiency?

RANT ALERT!
Scope of Class

• Narrower platform/API focus for this semester

• Single-chassis multiprocessors
  (but substantial similarity to GPU programming)

• Will not focus on distributed or highly heterogeneous programming (e.g. MPI)
Scope of Class

• Technical topics

  • Multithreaded programming; Synchronization; Using the OpenMP API

  • Instruction Level Parallelism; Vectorization and challenges; SIMD intrinsics

  • Memory hierarchy and its implications; Caches; Virtual Memory

  • Assessing efficiency, predicting parallel potential, and benchmarking performance

  • Understanding the role of compute and/or memory throughput as a limiting factor of performance

  • Optimizing data structures for target architecture; Memory allocation and management
Scope of Class

• Application focus

  • Sparse linear algebra; Matrix representations; Iterative solvers for sparse systems

  • Dense linear algebra; Matrix/Vector operations; Matrix Factorizations; Using the MKL library

  • Grid and stencil computations; Convolutions and their use in neural networks

  • Fourier transforms; Eigenvalue problems; PCA and Singular Value Decomposition

  • Optimization methods; Least-squares and approximation; Descent methods
Course logistics

• Location: Grainger Hall 2080

• 3 credits

• Class meets: TTh 2:30pm - 3:45pm

• Office hours (by instructor) - CS6387
  Mondays 11:00am - 11:45am
  Wednesdays 1:15pm - 2:00pm (starting Jan 29th)
  Fridays: 4:15pm - 5:00pm (starting Jan 24th)

• Friday slot will likely transform into an in-class review/help session within a couple weeks (location to be announced - in CS building)
Course information

• Piazza
  • Signup link: http://piazza.com/wisc/spring2020/cs639
  • Class link: piazza.com/wisc/spring2020/cs639/home
  • Email the instructor if you are have issues enrolling

• Email: sifakis@cs.wisc.edu
  • Please add “[CS639]” in the beginning of the subject line!

• Email policy: Feel free to email as frequently as you need. Typically you will receive a response within 24hrs. However, be prepared to wait until next office hours (worst case) to get a comprehensive answer. If urgent, ask for an appointment.
Evaluation

• Grading
  • 60% - Regular Programming Assignments
  • 15% - Optional Midterm - Friday March 6th, 7:15-9:15pm
  • 25% - Final Exam
  • Midterm grade will only count if greater than final exam. Otherwise the final exam will count for 40%

• There may be opportunities for bonus credit, for example:
  • Assistance with class infrastructure
  • Scribing lecture notes
  • Strong and helpful presence on Piazza
Prerequisites

Prerequisites not strictly enforced in this first offering, but ...

- You should be comfortable with programming in the C language
  - CS354, or equivalent, is strongly encouraged. Talk to the instructor if you’re not sure of your background/preparation in C programming.
  - You should be comfortable with debugging applications in C, using version control systems (e.g. Git, Mercurial, SVN), and simple build systems (e.g. Make/CMake). Some resources will be provided to help.
  - We will review relevant APIs (e.g. OpenMP) in class.
- Case studies and application topics will be mostly from scientific computing
  - Familiarity with basic linear algebra will be useful (but not essential)
Welcome to CS639!
Undergraduate Topics In Computing: Parallel and Throughput-Optimized Programming

Spring 2020, 2:30-3:45 Tue/Thu