Course Review

Goals

- Goals for scientific codes: right enough, fast enough

- Hopefully after this class you
  - Can reason about code performance
    - By understanding hardware, software, algorithms
  - Can recognize common HPC patterns
    - Dense LA, Sparse LA, n-body codes, structured/unstructured grids, FFT, etc.
  - Are aware of available software and tools for HPC

Hardware concepts

- Things that matter:
  - Instruction-level parallelism (ILP)
  - Memory hierarchy and cost of cache misses
  - Communication costs (latency, bandwidth, flop rate)
  - Synchronization overheads

Numerical concepts

- Thinking about high-performance numeric often involves:
  - Tiling and blocking algorithms; building atop the BLAS
  - Ideas of sparsity and locality
  - Graph partitioning and communication/computation ratios
  - Information propagation, bandwidth/latency tradeoffs
  - Big picture view of sparse and direct iterative solvers
  - Multilevel approaches

Parallel Environments

- MPI
  - Portable to many implementations
  - Giant legacy code base
  - Still evolving
- OpenMP
  - Parallelize C, Fortran codes with simple changes
    - But may need more invasive changes to go fast
- CUDA, OpenCL
  - Highly data-parallel kernels (e.g., for GPU)
- Other
  - GAS systems (e.g., UPC)
  - Shared-memory threads (e.g., pthreads, TBB)

Libraries and frameworks

- Dense LA: LAPACK, BLAS, ScaLAPACK, ...
- Sparse direct solvers: SuperLU, MUMPS, Elemental, Pardiso, ...
- FFTs: FFTW
- Graph partitioning: METIS, ParMETIS, hmetis, SCOTCH, Zoltan, ...
- Frameworks: PETSc, Trilinos
  - Gigantic, a pain to compile, but does a lot
  - Good starting place for ideas, library bindings

Related reading on SWE

- "Five recommended practices for computational scientists who write software" (Kelley, Hook, and Sanders in Computing in Science and Engineering, 9/09)
- "Barely sufficient software engineering: 10 practices to improve your CSE software" (Heroux and Willenbring)
- Best Practices for Scientific Computing (Wilson et al) (http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745)
  - Follow-up: Good Enough Practices for Scientific Computing (http://swcarpentry.github.io/good-enough-practices-in-scientific-computing/)

Where we are and where we're going

Today: Hardware

- Your cell phone is a multicore machine
- Shared memory programming hasn't disappeared
- Accelerators are everywhere
- Caches keep getting more important
- A modest course cluster has 100-1000 processors
- Getting a significant fraction of peak is hard (especially for real applications)

Today: Software

- Lots of code is still C/C++/Fortran
  - These are still evolving languages
- Increased emphasis on high-level languages (e.g., Python)
  - High performance in specific domains
  - Domain-specific specializations
  - JIT and on-the-fly optimization are commonplace
- High productivity matters along with high performance

Today: Applications

- Still lots of "traditional" HPC computations
    - Large-scale optimization
    - PDE solves
    - Engineering simulation
- Graph applications?
    - Different properties from PDEs
    - Similar applications
- Also lots of ML computations
    - Often more opportunities for parallelism
    - Often more data, less accuracy - I/O becomes the key
    - Lots of work on frameworks for these problems
    - Closer to traditional HPC over time...
- "Big data" and DB ideas
    - Lots of relatively modest computations over lots of data
    - Still rather different community from lots of HPC

Where we're heading

- “If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?” -Seymour Cray

- Done with scaling up frequency, pipeline length
- Current hardware: multicore and manycore (GPUs, Xeon Phis)
  - Often specialized parallelism
- Where current hardware lives
  - Often in clusters, maybe "in the cloud"
  - More embedded computing too
- Straight-line prediction: double core counts every 18 months
- Real question is still how we'll use these cores!
- Even-worse issues: deep memory, communication costs

Where we're heading

- Many dimensions of "performance"
  - Time to execute a program or routine
  - Energy to execute a program or routine
  - Total cost of ownership/computation
  - Time to write and debug programs
- Scientific computing has been driven by speed
- Other measures of performance also have influence

Where we're heading

- The first exascale machine as of June 2022
    - And still an open question of how to use it
- Cloud vendors still care more about high throughput
    - But accelerated cloud instances a viable path to some HPC
- Languages still advancing slowly

Your Feedback

- Topics you didn't like vs. topics you would want to see included?
- More labs/less labs?
  - Focus labs on fewer topics? Broader range of topics?
  - Homework instead of labs?
- Balance between hardware, software, algorithms?
- No labs/homework at all?

- Please feel free to send me your thoughts and feedback openly, anytime
  - You can wait until after exam grades are finalized if you feel more comfortable

Final Exam

- Exam slots are 1 hour long (may not take the whole hour, depending)

- Exams will be given in person or via zoom
  - An email about scheduling will be sent out during the last week of course instruction

Final Exam

- You will be given a sheet with 4 topics/questions. You must choose 3 to address.
    - I will give you up to 15 minutes to consider the topics and write down whatever notes you want
    - We will then go over the questions, and you will present your answers orally to me, making reference to your notes or using the blackboard if desired
    - I will very likely ask you follow-up questions
    - Questions will likely be very similar to practice questions, but I reserve the right to ask other questions not included there
    - Will cover material from lectures 1-12, exercises 1-11

Oral Exam Grading

- Final Exam grading: On each question, I will score you out of 3 points:
  - 0 points: not answered or incorrect answer
  - 1 point: attempted to address the question, had some idea about the topic
  - 2 points: addressed the question and demonstrated a good level of understanding
  - 3 points: demonstrated thorough, deep understanding of the topic and issues, clearly gave explanation and examples

- Total grade is out of 9 points
  - e.g., if you get (3,2,2), you get 7/9 or 77.7%
  - if you get (1,2,2), you get 5/9 or 55.5%
  - if you get (3,3,2), you get 8/9 or 88.8%

- For translation to final course grade, see Syllabus

Topics Overview

Lecture 1: Introduction

-Moore's Law, Why computers are parallel

  -Supercomputer's today

  -Parallel programming challenges

  -Performance measures - strong scaling vs. weak scaling

Lecture 2: Sequential Computer Architectures

- Single Processor Machines
  - Idealized and actual costs in modern processors
  - the memory hierarchy, caches
  - parallelism within a processor
    - pipelining
    - SIMD
  - Case study: matrix multiply.
    - Costs/computational intensity for naive version, blocked version
    - Communication lower bounds for matrix multiply
    - BLAS2 vs. BLAS3

Lecture 3: Parallel Architectures and Shared Memory Programming

- Parallel machines and programming models
  - shared memory
  - message passing/distributed memory
  - data parallel
  - Flynn's taxonomy
- OpenMP basics
  - fork-join parallelism
  - typical bugs and performance issues
- Memory consistency and cache-coherence

Lecture 4: Sources of Parallelism and Locality

- Discrete event systems
    - synchronous vs. asynchronous
- Particle systems
    - external vs. nearby vs. far-field forces and computational/parallelization approaches for each
- Lumped variables (ODEs)
- PDEs

Lecture 5/6: Performance Modeling/MPI

- -Distributed memory architectures
  - -properties of communication networks (diameter, bisection BW)
  - -network topologies
    - -linear, ring, mesh, torus, hypercube, tree, butterfly
- -Performance models
  - -PRAM
  - -Bulk synchronous parallel (BSP)
  - -LogP
  - -alpha,beta,gamma model
- -MPI
  - -Basic communication in MPI
  - -Collective communication in MPI

Lecture 7: Dense LA

-Communication lower bounds and derivation

-Parallel data layouts for matrices

-Parallel dense matrix-vector products

-Parallel matrix multiply

    -SUMMA algorithm

-Parallel LU factorization

    -best data layout

-Parallel QR factorization

    -tournament pivoting idea

Lecture 8: Sparse LA

-sparse matrix formats and basic SpMV

  -dependency graph, lower bounds

  -sequential optimizations

    -types of blocking (register, cache)

    -reordering

  -distributed memory optimizations

    -partitioning problem

-higher level kernels

  -sparse matrix multiply

  -matrix powers computations

-iterative solvers

  -pipelined Krylov subspace methods

  -communication-avoiding Krylov subspace methods

Lecture 9: Graph Partititioning

-Definition and connection with communication

-Partitioning with nodal coordinates

- -Planar separator theorem
- -Inertial partitioning
- -Random spheres

-Partitioning without nodal coordinates

- -Breadth-first search method
- -Spectral bisection

-Iterative Refinement

- -Kernighjan-Lin
- -Fiduccia-Mattheyses

-Multilevel acceleration

-Hypergraph models, benefit of hypergraph model

Lecture 10: Particle Methods

-Approximation of far-field forces idea

-Data structures - quad/oct trees

-Barnes-Hut algorithm

-Fast multipole methods (FMM)

-Differences between B-H and FMM (e.g., how approximation can be improved)

-Parallelization of Barnes-Hut and FMM

  -load balancing schemes

Lecture 11: Fast Fourier Transform

-Main divide and conquer idea behind FFT

-Sequential algorithm (Cooley and Tukey)

- -Lower bounds on communication

-1D FFT Parallelization approaches

- -Block versus Cyclic layout
- -Transpose approach
- -Cost of each in terms of alpha, beta, gamma model

-Higher dimension FFTs (3D case)

- -Parallelization approaches
  - -Packed slabs versus slabs versus pencils
    - -Tradeoffs in bandwidth and latency costs
    - -Opportunities for communication/computation overlapping

Lecture 12: ML and Data Analysis

-implicit versus explicit parallelization

-Supervised learning problems

- -neural network basics
  - -data versus model parallelization approaches
- -support vector machines
  - -parallelization challenges

-Unsupervised learning problems

- -nonnegative matrix factorization
  - -parallelization approach
- -spectral and markov clustering
  - -basic idea behind spectral clustering
  - -basic idea behind markov clustering (MCL)

Practice Exam Questions

Explain the different possibilities for distributing a matrix amongst processors. In the context of the distributed memory implementation of LU factorization, explain which distribution is best and why others are suboptimal.

- Source: Lecture 7 (slides 50-69)
- Key points:
  - Be able to explain the choices for distributing a 2D matrix
  - Be able to explain what LU factorization is
  - Be able to explain the distributed memory parallel algorithm for LU factorization (e.g., the blocked algorithm)
    - BLAS2 vs. BLAS3

What is bisection bandwidth? What is the bisection bandwidth for the following networks: Linear, 2D mesh, 2D torus, tree, and fattree? Is bisection bandwidth a useful metric to describe networks topologies?

- Source: Lecture 5, slides 14-20

Give examples of deadlock and race conditions as they might occur in a distributed memory parallel program based on message passing.

- Source: Lecture 3,5
    - Lecture 3: slide 19
    - Lecture 5: slide 53
    - Example: Exercises 6 (nonblocking bcast)
- Other examples possible from your experience in labs

Describe the following types of parallelism and give examples of each: task, data, divide and conquer, pipeline.

- Task: MIMD (Flynn's taxonomy, Lecture 3, slide 30)
- Data: SIMD/SIMT (Flynn's taxonomy, Lecture 3, slide 30; Lecture 2 slides 35-36)
- Divide and Conquer: e.g., reductions, FFT
- Pipeline (Lecture 2, slides 32-34; Lecture 8, slides 77-83)

Explain the basics of OpenMP. Give an example of how a data race can occur in shared memory parallel programming.

- Lecture 3, slides 42-92
- Data races
    - Lecture 3 (slides 10-12)
    - Exercises 3 (slides 16-23)

Give an overview of how algorithms like Barnes Hut and FMM speed up n-body calculations. What is the main difference between Barnes-Hut and FMM in terms of accuracy/computation tradeoffs? What other ways are BH and FMM similar and different?

- Source: Lecture 10
  - main idea: slide 2
  - differences: slide 22
- Other similarities: data structures, issues of load balance, data (re)distribution, similar computational structure (slide 43), similar parallelization schemes

- Key points:
  - Understand key point that allows speeding up of n-body computations
  - Give high-level description of BH and FMM algorithms
    - Data structures used, how approximations are done
  - Discuss differences and similarities

Given the DFT matrix, we wish to multiply it by a vector to transform the vector from the time domain into the frequency domain. Usually a matrix vector multiply takes O(n^2) steps. Describe how the FFT algorithm performs a matrix vector multiply in only O(nlogn) steps.

- Source: Lecture 11
- Divide and conquer approach: main idea, slide 19-22

State the lower bounds on the number of words moved between levels of the memory hierarchy and the number of messages for sequential dense matrix multiply. Now explain how to derive lower bounds for the parallel case. Give a count of the number of messages and number of words moved in the SUMMA algorithm.

- Sequential and parallel case: Slides 12-19 of lecture 7
- SUMMA algorithm: Slides 35-38 of lecture 7

- Potential follow-up questions
  - Explain idea behind 2.5D matrix multiply (slides 39-42 of lecture 7)

Explain how graph partitioning can be used to distribute a sparse matrix-vector computation. Give an example that shows that minimizing the graph cut is not equivalent to minimizing the total number of words communicated between processors. Explain how this can be remedied by a hypergraph model. Is using a hypergraph model always worth it in practice?

- Source: Lecture 9
    - Slides 3-4, Slide 6
    - Slides 80-89

Explain the difference between implicit and explicit parallelization. Give an example of an algorithm where implicit parallelization is a good approach, and one algorithm where it is not.

- Source: Lecture 12
- Definitions: slide 7
- Examples: from Lecture 12
  - Implicit: e.g., NMF
  - Explicit: e.g., SVMs
- Many examples also from other lectures (e.g., Lecture 8; see s-step Krylov subspace methods)

Explain the difference between strong scaling and weak scaling. Why are both measurements important?

- Source: Lecture 1, slide 23, 68-69, 75-79
- Key points:
  - How to measure strong scaling?
    - Explain Amdahl's law
  - How to measure weak scaling?
  - Which is generally easier to attain? Good strong scaling or good weak scaling?
  - Are there applications where strong scaling makes more sense? Where weak scaling makes more sense?

Define the graph partitioning problem. How would the approach you use differ when the nodes have associated coordinate information versus when they don't? Explain how "multilevel" approaches work in the context of graph partitioning.

- Source: Lecture 9
- Definition: slide 3
- Nodal coordinates: 13-28
- Without nodal coordinates: 30-59
- Multilevel approaches: 61-77

Explain what is meant by "collective communication" in MPI. Explain the following collectives: broadcast, scatter, gather, Alltoall, Reduce, Allreduce. Give brief explanation or pseudocode describing how you could implement a broadcast collective using just MPI_Send and MPI_Recv.

- Sources
  - Lecture 6, slides 25-37
  - Exercises 6

Which parallel benchmarks are used to rank supercomputers today? Explain why the HPCG benchmark gets such a small percentage of peak, even on the top computers, compared to the LINPACK benchmark. Can the computational intensity of iterative solvers like CG be improved? At what cost (e.g., potentially decreased accuracy)?

- Source: Lecture 1, slides 29-34
- Lecture 8, slides 11-14, 60-105

Thank you!