AMS 250

Introduction to
High Performance Computing

Nic Brummell
brummell@soe.ucsc.edu
(831) 459 2122
Applied Math & Statistics (AMS)
University of California, Santa Cruz
Maybe we want to model something complicated …
Global simulations: Differential rotation/Dynamo

Brun, Brown, Browning, Clune, Elliot, Miesch, Toomre (University of Colorado)

Largest global simulations in the world – spherical harmonic degree $l \sim O(1000)$

Problem: resolves from largest scale down $\Rightarrow$ diffuses smallest scale $\sim$ supergranules
One simulation costs 60,000 supercomputing hours.

= 2500 days

= 83 months

= 7 years!

*How come we can write scientific papers where we need to do a FEW simulations?*

*Are we that old?*

*NO!* (don’t be cheeky)

*PARALLEL COMPUTERS* => 60,000 supercomputing hours can be done in *1 wall clock hour* on 60,000 processors

(or *120 wall clock hours = 5 days* on 500 processors, etc)
Computing power continues to increase (yay!), but …

1. Why are all computers parallel computers now?
2. What problems can we do on such machines?
3. Why is it so hard to work with them?

This class will discuss these questions

This course should give you enough insight to be able to proceed with work in the current environment.
Computing power steadily increases!
Computing power steadily increases!

- “I think there is a world market for maybe five computers.”
  - Thomas Watson, chairman of IBM, 1943.
- “There is no reason for any individual to have a computer in their home”
- “640K [of memory] ought to be enough for anybody.”
  - Bill Gates, chairman of Microsoft, 1981.
- “On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it.”
  - Ken Kennedy, CRPC Directory, 1994
Computing power steadily increases!

**Moore’s Law:**

Gordon E. Moore  
Intel co-founder  
Then director of R&D at Fairchild Semiconductor  
“Electronics Magazine”, 35th anniversary issue, 1965

What is going to happen over the next 10 years?

“The complexity for minimum component costs has increased at a rate of *roughly a factor of two per year*. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years."

〜 “Number of transistors that can be placed cheaply on an integrated circuit board will double every two years”

〜 *computing power doubles every two years*

= *exponential rate of increase!*

Originally a forecast; now a goal!
Moore’s Law

Microprocessor Transistor Counts 1971-2011 & Moore’s Law

Curved line on graph showing transistor count doubling every two years.
Jack Dongarra (Univ of Tenn), Erich Strohmaier and Horst Simon (LBL)

- Ranks and details top 500 fastest supercomputer systems in the world
- HPL (High Performance Linpack) benchmark

[Graph showing performance of supercomputers over time]
The 38th TOP500 List as of November 2011

Performance Development

- IBM Roadrunner
- Fujitsu K Computer
- NEC Earth Simulator
- IBM BlueGene/L
- IBM ASCI White LLNL
- Intel ASCI Red Sandia
- Fujitsu NWT NAL
- Notebook

Time (1994 to 2012)

Performance (Pflop/s, Tflop/s, Gflop/s, Mflop/s)
Fastest!
### Top 10 positions of the 44th TOP500 on November, 2014

<table>
<thead>
<tr>
<th>Rank</th>
<th>Rmax (Rpeak)</th>
<th>Name</th>
<th>Operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.863 54.902</td>
<td>Tianhe-2  NUDT  Xeon E5-2692 + Xeon Phi 31S1P, TH Express-2</td>
<td>Linux (Kylin)</td>
</tr>
<tr>
<td>2</td>
<td>17.590 27.113</td>
<td>Titan   Cray  XK7  Opteron 6274 + Tesla K20X, Cray Gemini Interconnect</td>
<td>Linux (CLE, SLES based)</td>
</tr>
<tr>
<td>3</td>
<td>17.173 20.133</td>
<td>Sequoia Blue Gene/Q PowerPC A2, Custom</td>
<td>Linux (RHEL and CNK)</td>
</tr>
<tr>
<td>4</td>
<td>10.510 11.280</td>
<td>K computer RIKEN SPARC64 VIIIx, Tofu</td>
<td>Linux</td>
</tr>
<tr>
<td>5</td>
<td>8.586 10.086</td>
<td>Mira    Blue Gene/Q PowerPC A2, Custom</td>
<td>Linux (RHEL and CNK)</td>
</tr>
<tr>
<td>6</td>
<td>6.271 7.779</td>
<td>Piz Daint Cray XC30 Xeon E5-2670 + Tesla K20X, Aries</td>
<td>Linux (CLE)</td>
</tr>
<tr>
<td>7</td>
<td>5.168 8.520</td>
<td>Stampede PowerEdge C8220 Xeon E5-2680 + Xeon Phi, Infiniband</td>
<td>Linux</td>
</tr>
<tr>
<td>8</td>
<td>5.008 5.872</td>
<td>JUQUEEN Blue Gene/Q PowerPC A2, Custom</td>
<td>Linux (RHEL and CNK)</td>
</tr>
<tr>
<td>9</td>
<td>4.293 5.033</td>
<td>Vulcan  Blue Gene/Q PowerPC A2, Custom</td>
<td>Linux (RHEL and CNK)</td>
</tr>
<tr>
<td>10</td>
<td>3.577 6.132</td>
<td>Cray CS Xeon E5-2660v2 10C and Nvidia K40, Infiniband</td>
<td>Linux</td>
</tr>
</tbody>
</table>
Technology over the years: 1942 ENIAC 1

100 kops
Technology over the years: 1947 Whirlwind 3
Technology over the years: 1951 Univac

250 kops
Technology over the years: IBM 1950s

IBM 650

IBM 701

IBM 704
Technology over the years: 1964 IBM 360
Technology over the years: 1965 CDC 6600

3 Mflops
Technology over the years: 1976 Cray-1

250 Mflops
Technology over the years: 1978 VAX 11-780
Technology over the years: 1982 Cray X-MP

941 Mflops
Technology over the years: 1985 TM CM-1

1 Gflops
Technology over the years: 1990 NEC SX2/3

23 Gflops
Technology over the years: 1993 Cray T3D

1993 also: TM CM-5 ~ 65 Gflops, Intel Paragon ~ 143 Gflops
Technology over the years: Early 2000’s ASCI

ASCI White

ASCI Red
Earth Simulator

Fastest in world until 2004

Holistic simulations of global earth climate and oceans down to 10km resolution

5120 processors

35 Tflops
Technology over the years: 2007 IBM Blue Gene/L

Fastest machine in the world 2007
~ 213,000 cpus
596 Tflops peak, 478 sustained
Technology over the years: 2008 Roadrunner

IBM: 1.1 Pflops
Technology over the years: 2009 Cray XT5

Jaguar Cray XT5: 1.7 PFlops
Technology over the years: 2011 K-computer

Fujitsu

Japanese: “Kei” = quadrillion (Pflops) … 8 Pflops
Technology over the years: 2012 Sequoia

IBM Blue Gene: 1.5 million cores, 16 PFlops
Technology over the years: 2012 Titan

Cray XK7: 18 PFlops
Technology over the years: NOW! Tianhe-2

Tianhe: Chinese “Milky Way”

Tianhe-2: 34 Pflops!!

NUDT: National University of Defense Technologies
#1 since June 2013
(Tianhe-1 #1 in 2010)
Processors
How powerful?
How powerful?

Computing power steadily increases!

However …

1. Exponentially improving hardware does not mean exponentially improving software to go with it!

   **Wirth’s Law:** “Software gets slower faster than hardware gets faster”
   
   Software must get larger and more complex => slower!

2. Moore’s Law can’t go on forever. Even Moore acknowledged this!

   “In terms of size [of transistors] you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far—but that's as far out as we've ever been able to see. We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions”

   Technological singularity?? Eventually transistors will become size of an atom.
   Krauss & Starkman: 600 years.

3. **BUT new technology come along and replace integrated circuits?** (R. Kurzweil: “The Law of Accelerating Returns”)
Moore’s 2nd law (Rock’s Law)!

Costs go up exponentially too!
Manufacturing issues for high density chips limit increases

Power density limits serial performance too:
Scaling clock speed as usual will not work
Lower clock speed to lower power density
A shift in emphasis

- Chip density is continuing to increase ~2x every 2 years
- BUT clock speed is not
- Number of processor cores may double instead
- Power is under control, no longer growing
A shift in emphasis
Memory is not keeping pace

Memory density is doubling every three years; processor logic (computation) is every two years.

Storage costs (dollars/Mbyte) are dropping gradually compared to logic costs.

Question: Can you double concurrency without doubling memory?

Strong scaling: fixed problem size, increase number of processors

Weak scaling: grow problem size proportionally to number of processors.
The future?

All major processor vendors are producing *multicore* chips

- Every machine will soon be a parallel machine
- To keep doubling performance, parallelism must double
- Number of cores per chip can double every two years (re-interpretation of Moore’s Law!)
- Clock speed will NOT increase (and maybe even decrease for power density issues)

Which (commercial) applications can use this parallelism?

- Need to deal with systems with a high degree of concurrency – millions of threads?
- Need to deal with inter- and intra- chip parallelism
- Do they have to be rewritten from scratch?

Will all programmers have to be parallel programmers?

- New software model needed
- Try to hide complexity from most programmers – eventually
- In the meantime, need to understand it

Computer industry betting on this big change, but does not have all the answers!
What does the technology enable?
What does the technology enable?

OK, so we’ve got some large computers …

What can we do?

Why do we need them?

- **Continued exponential increase in computational power**

  → simulation is becoming third pillar of science, complementing theory and experiment

- **Continued exponential increase in experimental data**

  → techniques and technology in data analysis, visualization, analytics, networking, and collaboration tools are becoming essential in all data rich scientific applications
Third pillar of science

- **Traditional scientific and engineering method:**
  1. Do *theory* or paper design
  2. Perform *experiments* or build system

- **Limitations:**
  - Too difficult—build large wind tunnels
  - Too expensive—build a throw-away passenger jet
  - Too slow—wait for climate or galactic evolution
  - Too dangerous—weapons, drug design, climate experimentation

- **Computational science and engineering (CSE) paradigm:**
  3. Use computers to simulate and analyze the phenomenon
  - Based on known physical laws and efficient numerical methods
  - Analyze simulation results with computational tools and methods beyond what is possible experimentally
“An important development in sciences is occurring at the intersection of computer science and the sciences that has the potential to have a profound impact on science. It is a leap from the application of computing … to the integration of computer science concepts, tools, and theorems into the very fabric of science.” - Science 2020 Report, March 2006
Data-driven science

- Scientific data sets are growing exponentially
  - Ability to generate data is exceeding our ability to store and analyze
  - Simulation systems and some observational devices grow in capability with Moore’s Law

- Petabyte (PB) data sets will soon be common:
  - Climate modeling: estimates of the next IPCC data is in 10s of petabytes
  - Genome: JGI (LBL) alone will have .5 petabyte of data this year and double each year
  - Particle physics: LHC is projected to produce 16 petabytes of data per year
  - Astrophysics: LSST and others will produce 5 petabytes/year (via 3.2 Gigapixel camera)

- Create scientific communities with “Science Gateways” to data: a new problem – “Big Data”; “Data Science”
Particularly challenging problems

- **Science**
  - Weather prediction (speed); Global climate modeling (long time)
  - Biology: genomics; protein folding; drug design
  - Astrophysical modeling (cosmology=molecular dynamics!)
  - Computational Chemistry
  - Computational Material Sciences and Nanosciences

- **Engineering**
  - Semiconductor design
  - Earthquake and structural modeling
  - Computation fluid dynamics (aircraft design)
  - Combustion (engine design)
  - Crash simulation

- **Business**
  - Financial and economic modeling
  - Transaction processing, web services and search engines

- **Defense**
  - Nuclear weapons -- test by simulations
  - Cryptography (not so much anymore!)
Particularly challenging problems

- **“Graph Science”**
  - “Non-physics” problems
  - New(ish) data-driven, data-intensive “science”

- **Kernel areas**
  - Concurrent search
  - Optimisation
  - Edge-orientated

- **Applications**
  - Cybersecurity
  - Medical informatics
  - Social networks
  - Graph500.org
Example: Climate Modelling

- Problem is to compute:
  \[ f(\text{latitude}, \text{longitude}, \text{elevation}, \text{time}) \rightarrow \text{“weather”} = \]
  \[ (\text{temperature, pressure, humidity, wind velocity}) \]

- Approach:
  - *Discretize* the domain - a measurement point every 10 km (0.1 deg)?
    (Current models are coarser than this ~ 50km: 0.5 deg)
  - Devise an algorithm to predict weather at time \( t+dt \) given \( t \)

- Importance:
  - Predict major events, e.g., El Nino, hurricanes
  - Evaluate global warming scenarios


Hurricane model video
Example: Climate Modelling

- **State of the art models** require integration of atmosphere, ocean, clouds, sea-ice, land models, plus possibly carbon cycle, geochemistry and more
- **One piece** is modeling the fluid flow in the atmosphere
  - Solve Navier-Stokes (fluid) equations
    - Takes roughly 100 Flops per grid point with 1 minute timestep
- **Computational requirements:**
  - **Speed:** 
    - $\text{# points} = \frac{\text{Area/resln} \times \#\text{height\_levels}}{4 \pi (6000 \text{km})^2 / (10 \text{km} \times 10 \text{km}) \times 1000} \approx 5 \times 10^9$ points \(\Rightarrow\) $5 \times 10^{11}$ Flops/timestep (min)
  - To match real-time, need $5 \times 10^{11}$ flops in 60 seconds $\approx 8$ Gflop/s
  - Weather prediction (7 days in 24 hours) $\Rightarrow$ 56 Gflop/s
  - Climate prediction (50 years in 30 days) $\Rightarrow$ 4.8 Tflop/s
  - To use in policy negotiations (50 years in 12 hours) $\Rightarrow$ 288 Tflop/s
  - **Data:** $5 \times 10^9 \times 5 \text{ years} \times 8 \text{ bytes} \times 60\text{mins} \approx 10 \text{ Tbytes per sim hour}$ $\approx 5 \text{ Exabytes per climate prediction!}$
- To double the grid resolution, computation is $8x$ to $16x$ !!
Example: Climate Modelling

Effect of resolution:

Ref: P. Duffy et al, LLNL
Example: Climate Modelling

Effect of resolution: SST in Atlantic

Ref: NOAA GFDL
BUT … writing parallel code is hard! 😞
BUT … writing parallel code is hard! 😞

Intrinsic difficulties of parallel computing:

- Finding enough parallelism (Amdahl’s Law)
- Granularity – how to chop up a task into pieces – size matters!
- Locality – moving data about is expensive (slow)
- Load balance – don’t want 1000 processors to wait for one slow one
- Coordination and synchronization – sharing data safely
- Performance modeling/debugging/tuning

None of these things are issues in sequential programming!
Can it all be done automatically?

HOLY GRAIL!

But some already exists in the hierarchy:

- **Bit level parallelism:**
  - within floating point operations, etc.

- **Instruction level parallelism (ILP):**
  - multiple instructions execute per clock cycle

- **Memory system parallelism:**
  - overlap of memory operations with computation

- **<< BIG GAP at the programming layer! >>**

- **OS parallelism:**
  - multiple jobs run in parallel on commodity SMPs

These are modest parallelisations compared to the *task level parallelisation* by the user.
Finding enough parallelism

Suppose only part of an application seems parallelisable.

**Amdahl’s law**

Let $s$ be the fraction of work done sequentially.

$(1-s)$ is fraction parallelisable.

$P =$ number of processors.

$$\text{Speedup}(P) = \frac{\text{Time}(1)}{\text{Time}(P)}$$

$$< \frac{1}{s + (1-s)/P} \quad \text{(max parallel efficiency on parallel fraction)}$$

$$< \frac{1}{s} \quad \text{as } P \text{ tends to infinity}$$

Top500 list: currently fastest machine has $P \sim 3.1M$; 2$^{nd}$ fastest has $\sim 560K$.

Amdahl: Even if the parallel part speeds up perfectly performance is limited by the sequential part.

*(Things can be MUCH more complicated than this simplistic scenario: More later; just gives you the idea)*
Granularity

Parallelism introduces extra overheads

• Given enough parallel work, this is the biggest barrier to getting desired speedup

• Parallelism overheads include:
  – cost of starting a thread or process
  – cost of communicating shared data
  – cost of synchronizing
  – extra (redundant) computation

• Each of these can be in the range of milliseconds (equivalent to millions of flops on a Gflop machine)

• Tradeoff: Want to chop problem into many pieces to increase parallelism, but pieces cannot be so small that work is swamped by overheads. Granularity issue.
Far away, large memories are slow; Near, small memories are fast
Slow accesses to “remote” data we call “communication”
Storage hierarchies are large and quick only on average
Collectively, parallel machine have large, fast cache – but, right place, right time? Synced?
=> Algorithm should do most work on local data = LOCALITY issue
Gap between processor speeds and memory speeds is ever widening despite work on memory hierarchies, bus controllers, etc. Must minimise communication!
Load imbalance

• Load imbalance is the time that some processors in the system are idle due to
  – insufficient parallelism (during that phase)
  – unequal size tasks

• Examples of the latter
  – adapting to “interesting parts of a domain” — adaptive mesh refinement AMR
  – tree-structured computations
  – fundamentally unstructured problems

• Algorithm needs to balance load
  – Sometimes can determine work load, divide up evenly, before starting
    ⇒ “Static Load Balancing”
  – Sometimes work load changes dynamically, need to rebalance dynamically
    ⇒ “Dynamic Load Balancing,” eg work-stealing (greedy algorithms)
Computing power continues to increase (Moore’s Law)

Good, because enables us to do some very hard problems 😊

Paradigm shift #1 (production costs): All computers are now parallel

Paradigm shift #2 (power consumption): Chips are all becoming multicore

Multiple levels of concurrency, some automatic, main level ... NOT.

=> parallel computing is hard!

- Finding parallelism (Amdahl’s Law)
- Granularity
- Locality
- Load Balancing
- Coordination and synchronisation
- Debugging
Who does what?

Do all scientists have to become low level programmers?

2 types of programmers for 2 layers of software?

- **Efficiency Layer** (10% of programmers)
  - Expert programmers build Libraries implementing kernels, “Frameworks”, OS, ....
  - Highest fraction of peak performance possible

- **Productivity Layer** (90% of programmers)
  - Domain experts / Non-expert programmers productively build parallel applications by composing frameworks & libraries
  - Hide as many details of machine, parallelism as possible
  - Willing to sacrifice some performance for productive programming

Students may want to work at either level but this course assumes mainly latter? (BUT aim to understand enough of the efficiency layer to use parallelism effectively)
What is the point of this course?
What is the point of this course?

Learn how to harness all this computing power!

In particular, learn how to design and use code for 

**MASSIVELY PARALLEL MACHINES**

Elusive mixture of theory and practice …

**Highly interdisciplinary!** These days need:

- **Engineering skills**: architecture, hardware, network knowledge
- **Mathematical skills**: applied math methods, numerical analysis
- **Computer science skills**: algorithm design, software engineering, compilers, operating systems, ftp, …
- **Scientific skills**: design of problem, setup of models/equations, analysis of results
- **Artistic skills**: visualisation, movies, …
The syllabus

PART A: CONCEPTS

- Intro to Parallel Computing
  - Parallel machine models
  - Parallel programming models

- Designing Parallel algorithms
  - Partitioning, communication, agglomeration, mapping
  - Performance analysis

PART B: TOOLS

- Environment: Unix; QSUB; svn, github?
- Programming: Fortran?
- Programming: MPI, OpenMP, …
- Debugging, performance analysis
- Analysis ?(IDL?, Python, VAPOR)

PART C: SPECIALTY ITEMS

- GPU, Intel Phi?
- Map-Reduce, Hadoop, …

PART D: CASE STUDIES?

- Spectral Methods (HPS); Finite-Volume; N-Body?
The syllabus

PART A: CONCEPTS

- Intro to Parallel Computing
  - Parallel machine models
  - Parallel programming models
- Designing Parallel algorithms
  - Partitioning, communication, agglomeration, mapping
  - Performance analysis

PART B: TOOLS

- Environment: Unix; QSUB; svn, github
- Programming: Fortran
- Programming: MPI, OpenMP, …
- Debugging, performance analysis
- Analysis ?(IDL?, Python, VAPOR)

PART C: SPECIALTY ITEMS

- GPU, Intel Phi
- Map-Reduce, Hadoop, …

PART D: CASE STUDIES?

- Spectral Methods (HPS); Finite-Volume; N-Body

EASY!

But very seldom all collected together in one place for you!

Ask the specialists!
Expectations

Course is **BRAND NEW** = an EXPERIMENT!

Materials are “in preparation” (sorry, guinea pigs!)

*Expectations of the student:*

- Learn something useful!
- There will be exercises and tasks
- But very seldom “right” and “wrong” answers
- Often it is possible that I won’t know the answer!
- The expectation is only that the student try and make use of what they’ve learned.
Learning outcomes

A pretty in-depth understanding of:

- When parallel computing is useful
- Parallel computing hardware options
- Parallel programming models (software) and tools and experience using some of them
- Some important parallel applications and the algorithms
- Debugging, performance analysis and tuning
- Exposure to some open research questions
Course details

Website: http://courses.soe.ucsc.edu/courses/ams250/Spr15

Grading: some homeworks, some programming assignments, no exams!

Computer accounts:

Grape – grape.soe.ucsc.edu: our main "toy" resource 😊

(see next slide)

Books, resources: I will discuss as we go along, but …

• Part 1 based on Ian Foster’s (ANL) excellent analysis book “Designing and Building Parallel Programs” available for free on the web:

• Part 2: MPI, OpenMP

  ➢ lots of tutorials on web
  ➢ some useful reference books too, but really probably not worth buying anything
Computer accounts:

Grape – grape.soe.ucsc.edu: our main “toy” resource 😊

- Register immediately at
  - https://accounts.soe.ucsc.edu/accounts/register
  - Account sponsor = me!

- Information on grape (and other useful stuff) at
  - https://ams.soe.ucsc.edu/resources/computing
Current class skills?

Who has experience already with:

- ✓ Unix environment? (Unix commands, ssh, svn, …)
- ✓ Scripting languages? (Shell, python, …)
- ✓ Programming language? (Fortran, C, …)
- ✓ Using parallel environments on supercomputers? (i.e. running codes)
- ✓ Parallel programming (i.e. actually writing code)?
Top 500: Pick a computer and describe it. Specify at least

- Name
- Location
- No. of nodes, No. of processors per node \(\Rightarrow\) Total no. of processors
- Clock speed of chips
- Flops/processor and Total flops
- Memory per processor and Total memory
- Architecture type (SIMD, MIMD, …)
- Interconnect type
- Use?
- Anything special?

Idea: Get you used to the vocabulary of machine architectures

Create PDF and email to me: brummell@soe.ucsc.edu
Breakfast: parallel version

**TASK PARALLELISM!**
- Get up
- Go to bathroom
- Go downstairs
- Put kettle on
  - Start thinking about what I have to do today

**Drinking task**
- Grab breakfast dishes
- Put tea bag in cup
- Put bread in toaster
- Pour cereal/milk in bowl
- Start eating
  - Kettle boils
  - Pour tea; steep
  - Finish cereal
  - Push toaster on (n slices)
  - Toaster finishes
  - Eat toast
  - Tea ready
  - Drink tea
  - Tea ready
  - Drink tea

**Eating task**
- Finish cereal
- Push toaster on (n slices)
- Toaster finishes
- Eat toast

**Thinking task**
- Check calendar
  - Check weather for bike commute
- Start checking email
- Check news headlines
- Check FB
- Check calendar

**DATA PARALLEL!**
- Kettle boils
- Pour tea; steep
- Finish cereal
  - Push toaster on (n slices)
  - Toaster finishes
  - Eat toast

**FORK AND JOIN MODEL**
- Finish, wash dishes, get ready to leave

**SHARED MEMORY**
Tell me about something in your life experience that involves concurrency/parallelism!
HW 1: Recap

(i) Register for account on BSOE machines

(ii) Describe your Top500 machine

(iii) Tell me about something in your life experience that involves concurrency/parallelism!
Do we need to do primers for

Unix?

vi?

Fortran?
Spacer