Advanced Topics in Numerical Analysis: High Performance Computing

Cross-listed as MATH-GA.2012-001 and CSCI-GA 2945.001

Georg Stadler Dhairya Malhotra
Courant Institute, NYU
stadler@cims.nyu.edu dm4340@nyu.edu

Spring 2019, Monday, 5:10–7:00PM, WWH #1302

Jan 28, 2019

Outline

Organization issues

- Time and location: Mondays 5:10–7:00PM, WWH 1302
- Course webpage: https://nyu-hpc19.github.io/

Organization issues

- Time and location: Mondays 5:10–7:00PM, WWH 1302
- Course webpage: https://nyu-hpc19.github.io/
- We started a Slack group for communication in this class. Let us know if you want to be added and haven't been invited yet.
- Email: If you email us (stadler@cims.nyu.edu and dm4340@nyu.edu) about anything related to this course, please put [hpc19] in the email's subject line. However, we prefer if you message us on Slack.

Organization issues

- Time and location: Mondays 5:10–7:00PM, WWH 1302
- Course webpage: https://nyu-hpc19.github.io/
- We started a Slack group for communication in this class. Let us know if you want to be added and haven't been invited yet.
- Email: If you email us (stadler@cims.nyu.edu and dm4340@nyu.edu) about anything related to this course, please put [hpc19] in the email's subject line. However, we prefer if you message us on Slack.
- Office hours: TBD (always better to message on Slack). My office number is #1111 and Dhairya's office number is #1008.
- We have different backgrounds (Maths/CS/Economy/Physics(?)). You might know or not know some things already. Either way, please ask during class! Feedback and comments of any kind are welcome.

Organization issues

Prerequisites:

- Some basic experience in serial programming in either C, C++ or FORTRAN (this is not primarily a programming course!)
- Basic command line experience (shell, ssh, . . .)
- Basic knowledge/interest in numerical algorithms.

Organization issues

Prerequisites:

- Some basic experience in serial programming in either C, C++ or FORTRAN (this is not primarily a programming course!)
- Basic command line experience (shell, ssh, . . .)
- Basic knowledge/interest in numerical algorithms.

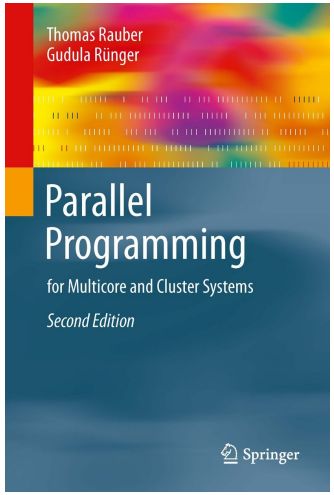
Class topics (intended):

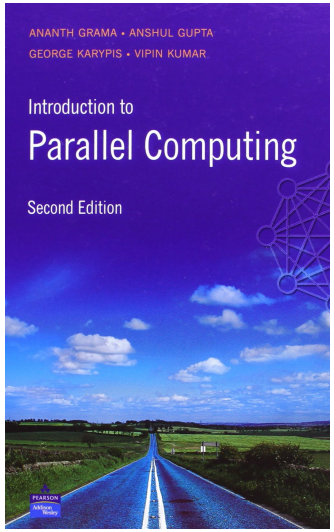
- Architecture and algorithms: hardware basics, memory hierarchy, programming models, networks
- Programming: OpenMP, CUDA, MPI, performance optimization
- Algorithms: Jacobi, Gauss-Seidel, multigrid, sorting, . . . + parallel versions thereof.
- Tools: Make, debugging, valgrind, git, paraview, job schedulers, . . .

Organization issues

Recommended textbooks/literature:

- We will mostly use online resources and to which we will post links on the public course website. If you insist on books, you can look at:
- T. Rauber and G. Rünger: *Parallel Programming for Multicore and Cluster Systems*, Springer, 2nd edition 2013. *Available online on NYU Campus.*
- A. Grama, A. Gupta, G. Karypis and V. Kumar: *Introduction to Parallel Computing*, Pearson, 2003.
- To refresh C programming language: Z. Shaw: *Learn C the hard way*, 2016.





Organization issues

Required work:

- Practical homework assignments every 2 weeks (50% of grade).
- Final project (50% of grade):
  - by yourself or in teams of 2
  - 10min project presentations at the end of the class, and handing in a short report
  - project can be related to your research project, but it needs to bring in some of the topics of the course!
  - please start thinking about your projects (we want to finalize project topics by mid March)
  - pitch your project to us during the next weeks (we'll also have examples for projects)

Even if you are only auditing this class, we encourage you to "get your hands dirty" (work on homework assignments, try out code, work on a project)!

Organization issues

Get the most out of this class!

- We'll "force" you to do some work (homework, final project)
- Will additionally try to open doors that allow you to explore and learn more things—it's up to yo uto get the most out of the class.

Organization issues

Get the most out of this class!

- We'll "force" you to do some work (homework, final project)
- Will additionally try to open doors that allow you to explore and learn more things—it's up to yo uto get the most out of the class.
- Ideally, your final project helps for your research, industry or academic job applications
- This is an interactive hands-on class. Please contribute and engage (in class, on Slack etc)
- Stop us and ask!

Outline

Units and performance measures

- flop: floating point operation, usually double precision (add or multiply; fused multiply-add)
- flop/s: floating point operation per second
- bytes: size of data (double floating point is 8 bytes), each byte has 8 bits

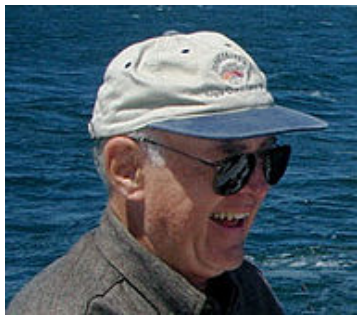
Things are getting large...

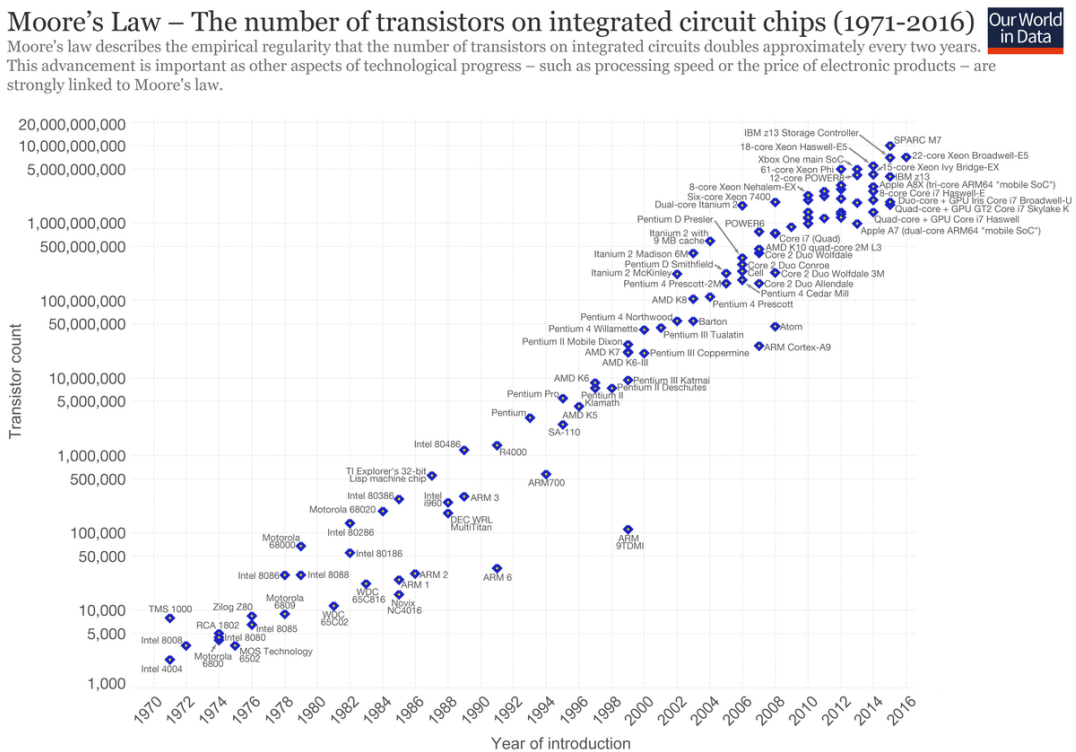
Mega:	Mflop/s=10^6 flop/sec	Mbyte=$2^{20} \approx 10^6$ bytes
Giga:	Gflop/s=10^9 flop/sec	Gbyte=$2^{30} \approx 10^9$ bytes
Tera:	Tflop/s=10^{12} flop/sec	Tbyte=$2^{40} \approx 10^{12}$ bytes
Peta:	Pflop/s=10^{15} flop/sec	Pbyte=$2^{50} \approx 10^{15}$ bytes
Exa:	Eflop/s=10^{18} flop/sec	Ebyte=$2^{60} \approx 10^{18}$ bytes
Zetta:	...	
Yotta:	...	

Currently fastest machine: Summit at Oak Ridge NL: ~201Pflop/s (sustained 143Pflop/s), 2.4 million cores, 2.8 Pbyte memory (we'll talk about the fastest machines later)

Moore's law

"The transistor density of semiconductor chips will double roughly every 18 months"
Gordon Moore (co-founder of Intel), 1965



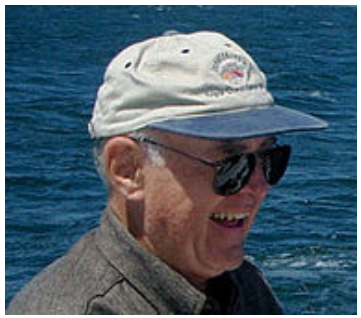


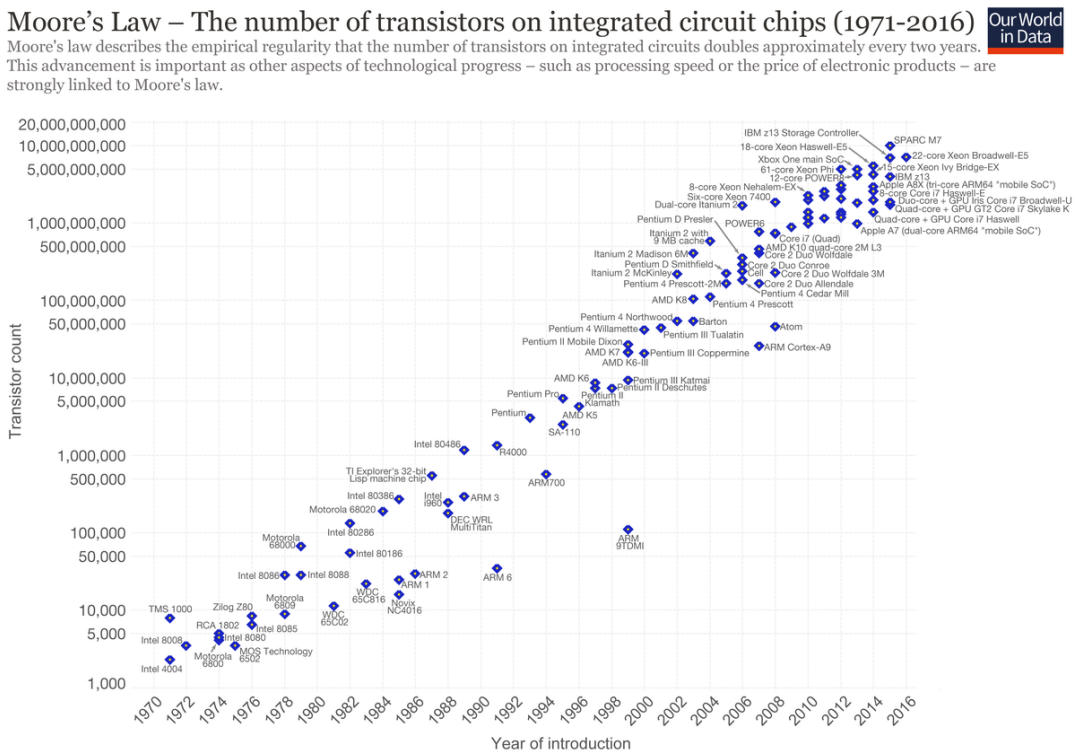
Source: Wikipedia

Transisters were getting smaller and faster (shorter wires).

Moore's law

"The transistor density of semiconductor chips will double roughly every 18 months"
Gordon Moore (co-founder of Intel), 1965





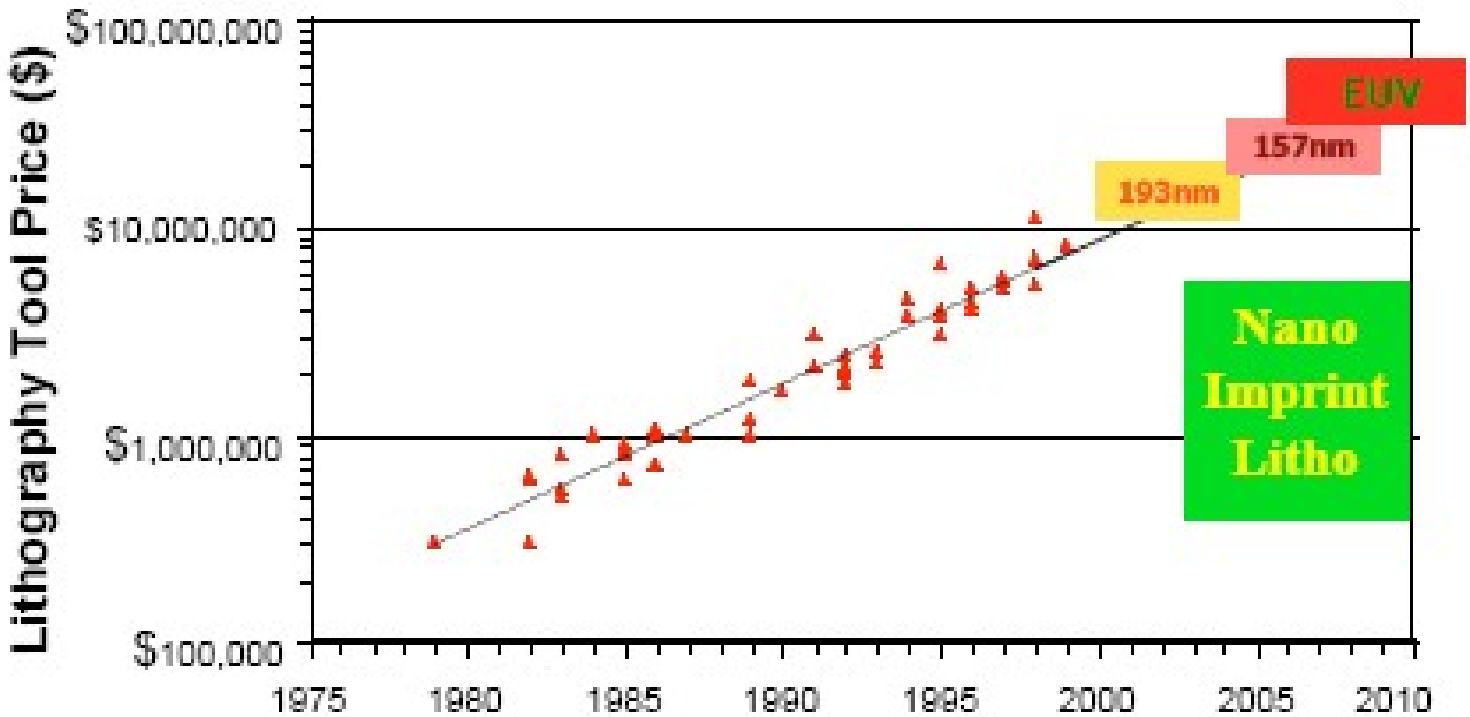
Source: Wikipedia

Transisters were getting smaller and faster (shorter wires).

Thus: Just need to wait, and your code will run faster.

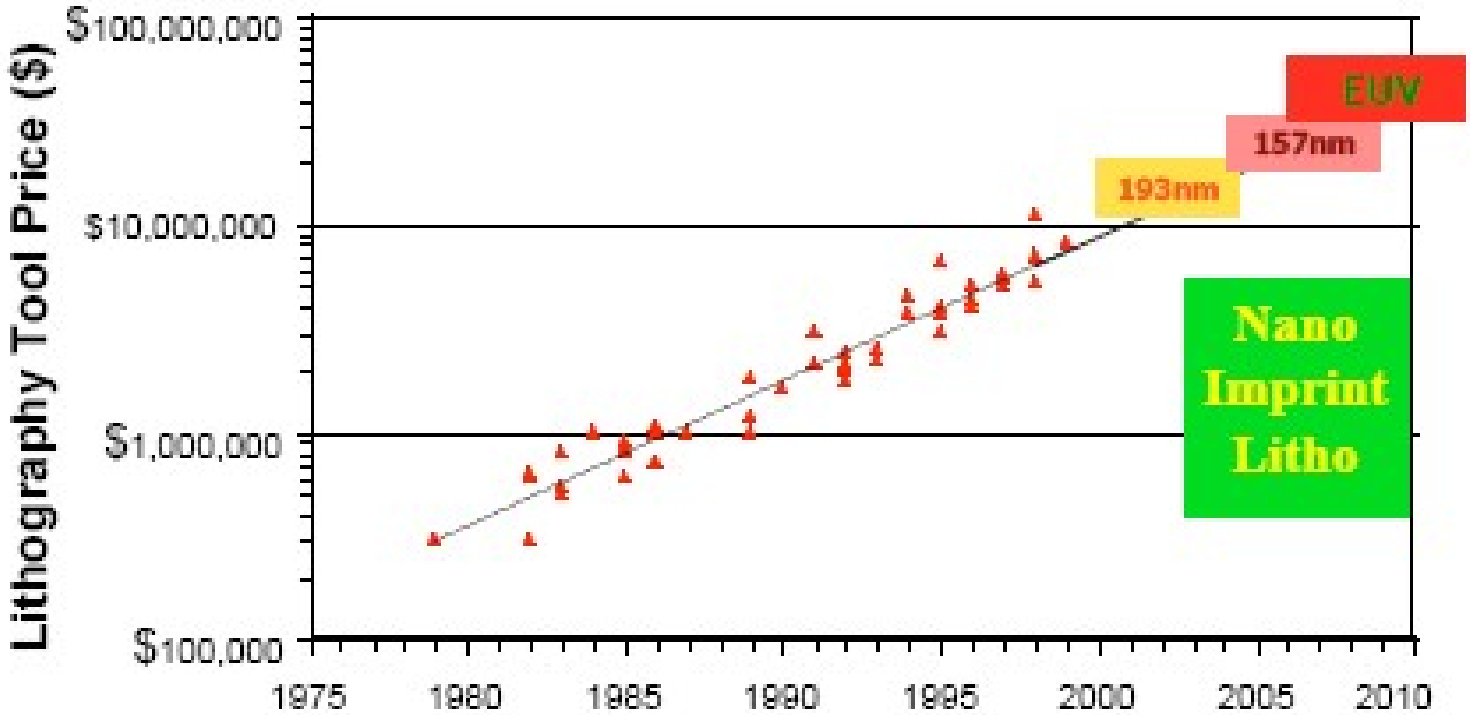
Limits of transistor scaling

- exponential growth of cost for tools for chip manufacturing (Moore's 2nd law); Currently AMD uses 7nm channel length produced by TSMC, and Intel uses 14++nm length



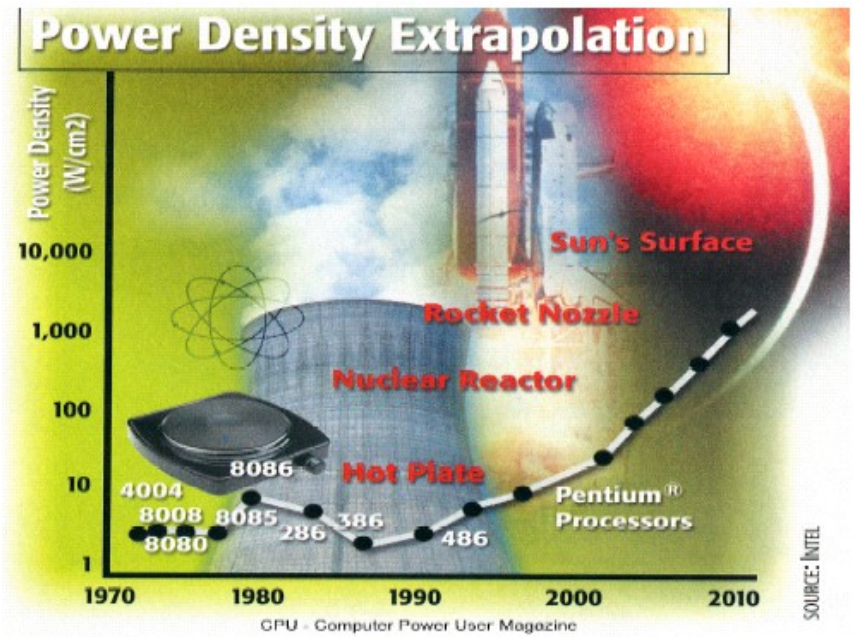
Limits of transistor scaling

- exponential growth of cost for tools for chip manufacturing (Moore's 2nd law); Currently AMD uses 7nm channel length produced by TSMC, and Intel uses 14++nm length
- reliability of production (high yield; number of Intel's Xeon Phi, "60+" cores)



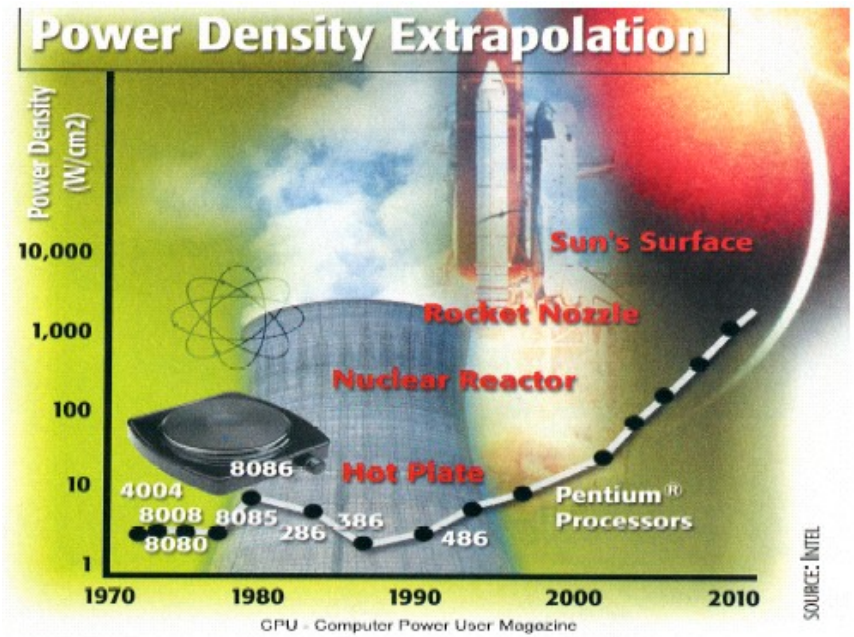
Limits of transistor scaling

- exponential growth of cost for tools for chip manufacturing (Moore's 2nd law); Currently AMD uses 7nm channel length produced by TSMC, and Intel uses 14++nm length
- reliability of production (high yield; number of Intel's Xeon Phi, "60+" cores)
- power density is cubic in frequency (leakage due to quantum mechanical effects becomes a problem)



Limits of transistor scaling

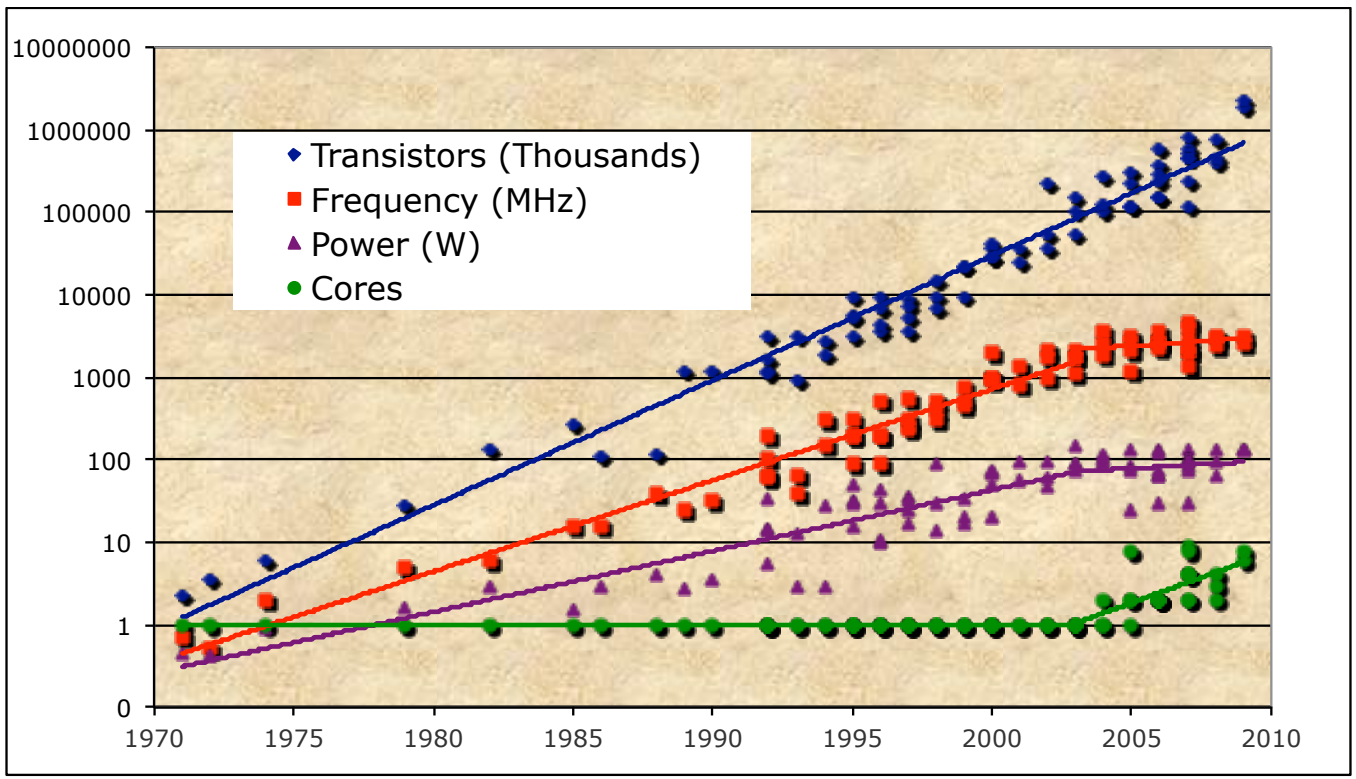
- exponential growth of cost for tools for chip manufacturing (Moore's 2nd law); Currently AMD uses 7nm channel length produced by TSMC, and Intel uses 14++nm length
- reliability of production (high yield; number of Intel's Xeon Phi, "60+" cores)
- power density is cubic in frequency (leakage due to quantum mechanical effects becomes a problem)



Thus: more transistors, but not faster serial processors

Moore's law today

- Frequency/clock speed stopped growing in ~ 2004
- Number of cores per CPU
- Moore's law still holds
- Energy use $\sim$bounded



Source: CS Department, UC Berkely.

Parallel computing $\subset$ high-performance computing

$$\boxed{\text{performance} = \text{single core \& multicore performance}}$$

- All major vendors produce multicore chips.
- How well can applications and algorithms exploit parallelism?
- Do we need to think differently about algorithms?
- How do we divide problems into smaller chunks?

Parallel computing $\subset$ high-performance computing

performance = single core & multicore performance

- All major vendors produce multicore chips.
- How well can applications and algorithms exploit parallelism?
- Do we need to think differently about algorithms?
- How do we divide problems into smaller chunks?

How about automatic parallelization?

- compilers help with other levels or parallelism (we will talk about those)
- automatic parallelization across different CPU cores has mostly been a failure (OK, some tools/libraries are in development)
- need to think parallel; parallelizing a sequential code can be difficult or impossible

Parallel computing $\subset$ high-performance computing

Beyond floating point operations per second

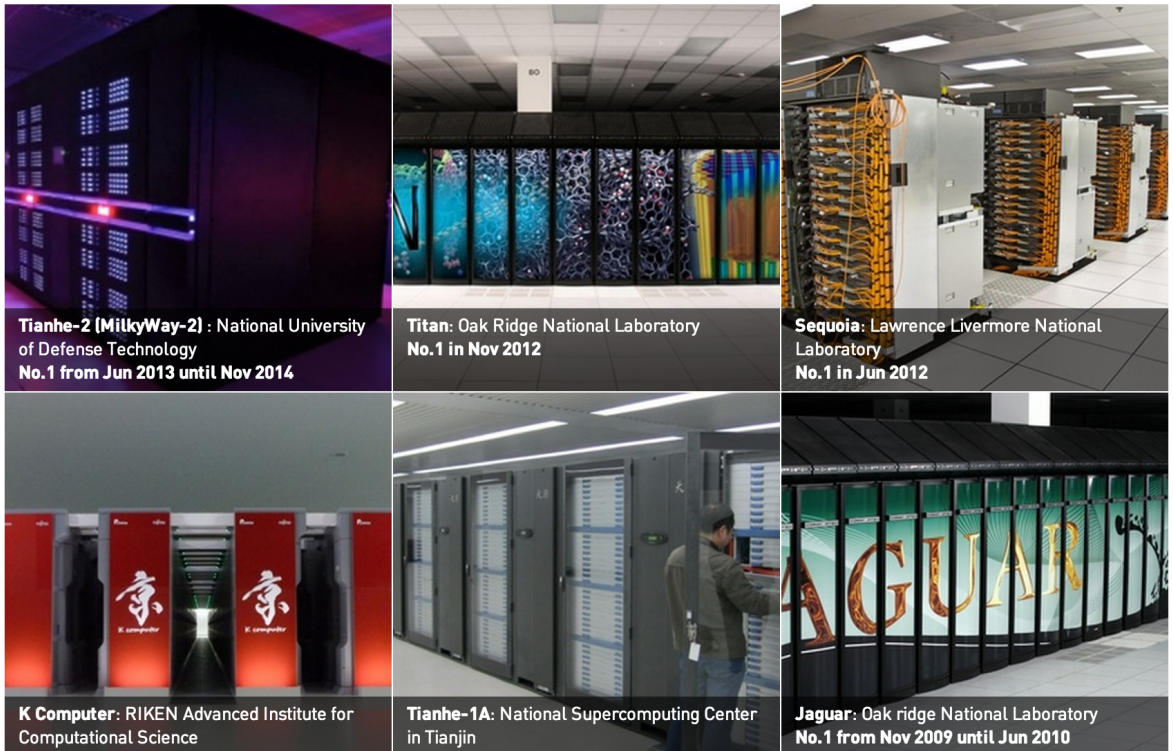
Memory density and time to access memory:

- The memory density only doubles every $\sim$3 years (Moore's law at a slower rate)
- Memory-bound algorithms: For many mathematical algorithms, loading/writing data from/to the memory is the bottleneck
- Memory hierarchies have been established (L1/L2/L3-cache), to allow faster read and write of memory
- Will be the topic of one of the next classes

The Top 500 List: http://www.top500.org/

Listing of most powerful supercomputers in the world

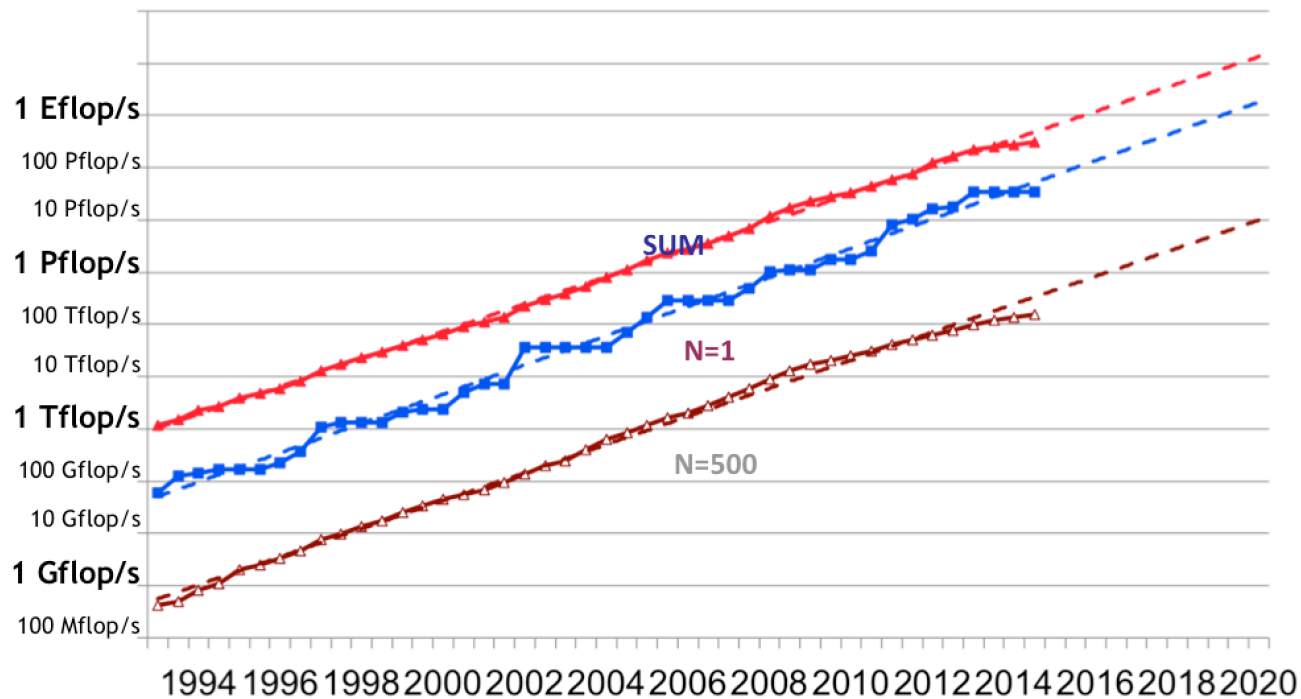
- updated twice a year at the European and US Supercomputing Conferences
- Reports theoretical flop/s (*RPEAK*), and "sustained" flop/s (*RMAX*) on a benchmark problem (dense matrix-vector multiplication)
- HPCG list: Uses a different benchmark based on sparse, memory-bounded operations.



The Top 500 List: http://www.top500.org/

Development over time (observe Moore's law)

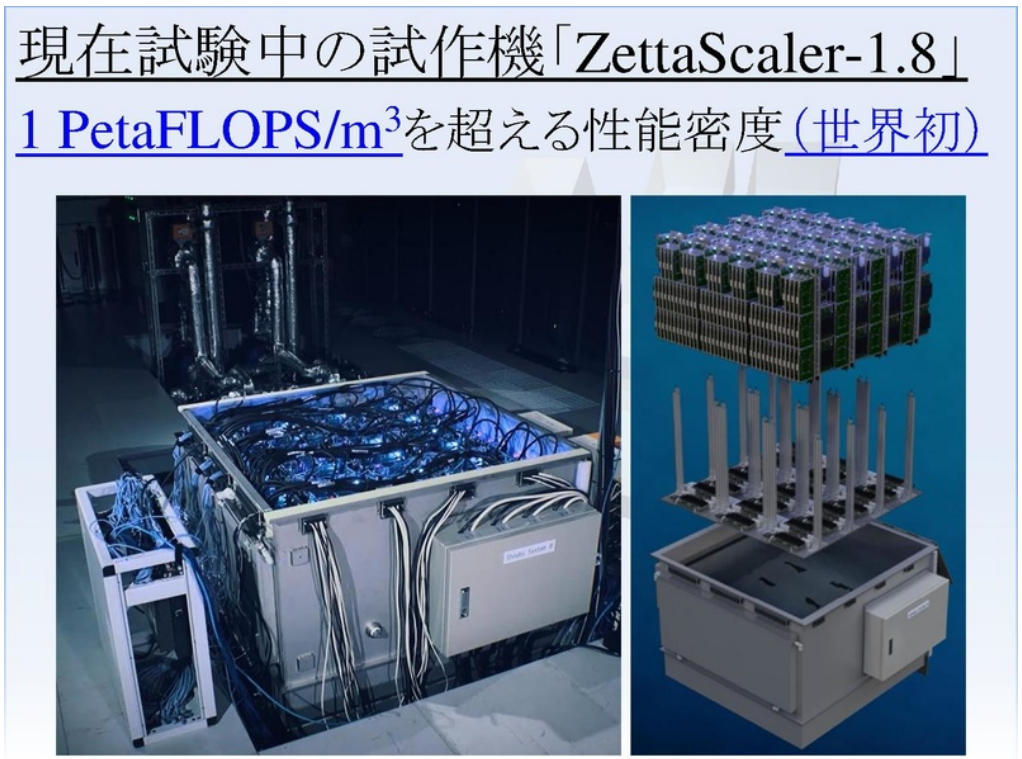
- #1 machine on Top500
- #500 machine on Top 500
- Sum over all machines on Top 500 list



Green 500: https://www.top500.org/green500/

Listing of most energy-efficient computers in the world

- Reports floating point operatios per Watt of energy used
- Results in a rather different order of the top machines
- The top machine uses low-energy processors that are submersed in mineral oil for efficient cooling:



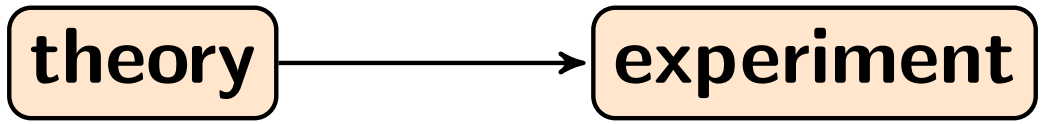
Do we really need larger and faster?

- Increased computational power allows simulations beyond what experiments or theory (paper & pencil) can do.
- Allows to go beyond "simulations", for instance by solving
  - inverse problems: infer unknown phenomena that cannot observed/measured from observations (e.g., earth structure; weather prediction)
  - control/design/decision problems: influence/control systems, simulate different scenaria
- Exponential growth in computing power brings along enhances in sensors, i.e., we have much more data that needs to be analyzed to make it useful ("finding the signal in the noise") $\Rightarrow$ big data requires big (and smart) computing

Do we really need larger and faster?

http://blog.tamest.org/computational-science-the-third-pillar-of-science/

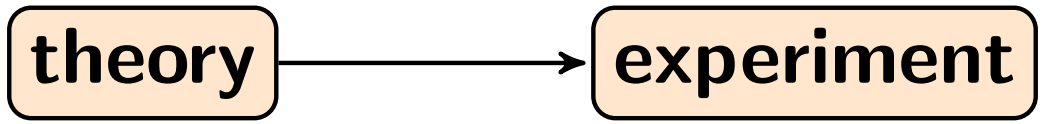
Traditional scientific method:



Do we really need larger and faster?

http://blog.tamest.org/computational-science-the-third-pillar-of-science/

Traditional scientific method:



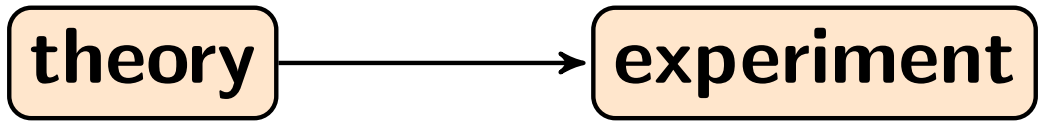
Limitations:

- difficult (wind tunnels)
- expensive (build experimental passenger jets; crash tests)
- slow (climate change)
- dangerous (weapons; drug design)

Do we really need larger and faster?

http://blog.tamest.org/computational-science-the-third-pillar-of-science/

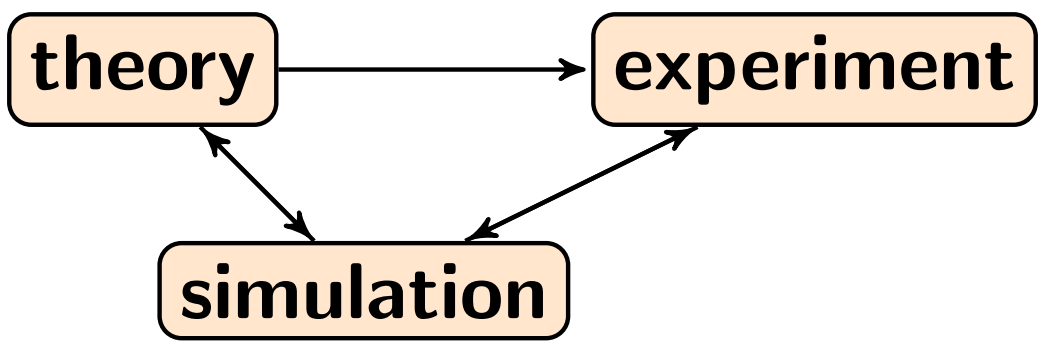
Traditional scientific method:



Limitations:

- difficult (wind tunnels)
- expensive (build experimental passenger jets; crash tests)
- slow (climate change)
- dangerous (weapons; drug design)

Simulation has become the third pillar of Science:



Particularly challenging computations

- Science
  - weather prediction, climate modeling
  - genomics, drug design
  - astrophysical simulation (supernova/black holes,...)
  - human brain modeling
    https://www.humanbrainproject.eu/
  - earthquake modeling and global imaging
  - global mantle convection modeling

Particularly challenging computations

- Science
  - weather prediction, climate modeling
  - genomics, drug design
  - astrophysical simulation (supernova/black holes,...)
  - human brain modeling
    https://www.humanbrainproject.eu/
  - earthquake modeling and global imaging
  - global mantle convection modeling
- Engineering
  - earthquake engineering
  - semiconductor design
  - computational fluid dynamics for airplane design
  - crash test simulations
  - weapons testing (US nuclear weapon tests stopped in 1992)

Particularly challenging computations

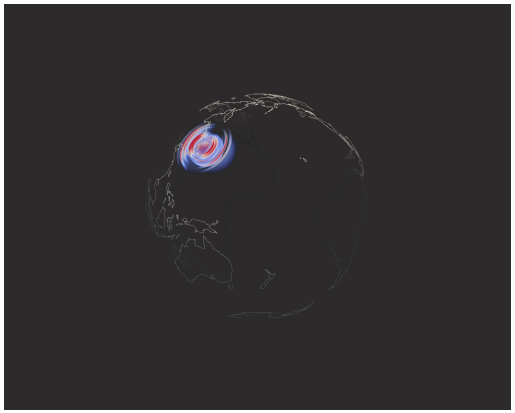
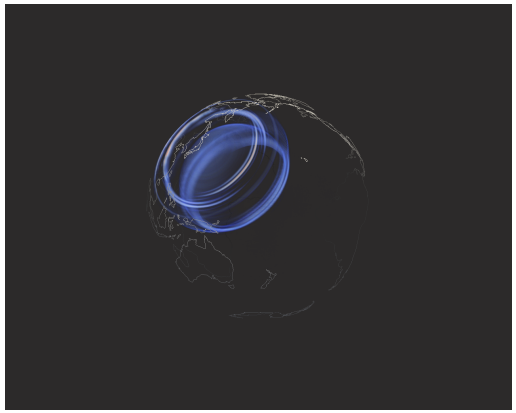
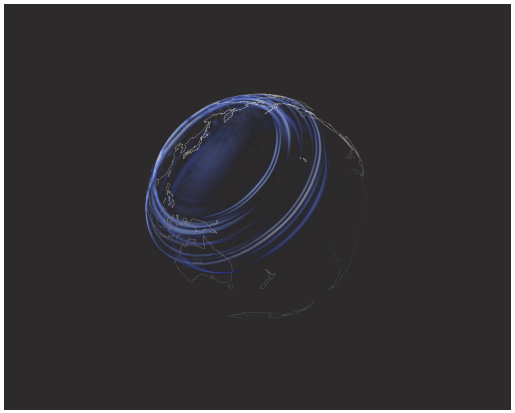
- Machine learning and AI
  - training of neural networks
  - deep learning
  - applications in self-driving cars, any kind of (un)supervised learning, etc

Particularly challenging computations

- Machine learning and AI
  - training of neural networks
  - deep learning
  - applications in self-driving cars, any kind of (un)supervised learning, etc
- Business
  - financial modeling; speed trading
  - transaction processing

Example 1: Seismic wave propagation

http://vimeo.com/16807465

Simulation of waves through the earth at a frequency of f Hertz

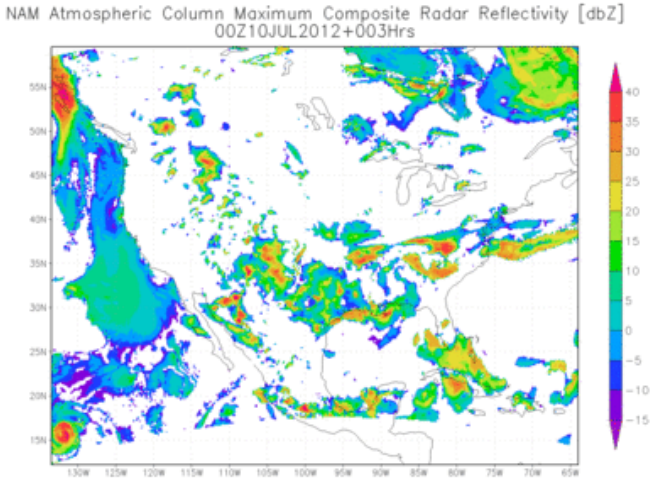
$$\sim 1600^3 p^3 f^3 \text{gridpoints},$$

where p is the number of points per wavelength (e.g., 6), and 1600 is computed from the earth radius (about 6700km) and the average wave speed 5-10km/sec).

f	#gridpints	per timestep
1/8Hz	1.7×10^9	∼400Gflop
1/44Hz	1.4×10^{10}	∼3Tflop
1/2Hz	1.1×10^{11}	∼24Tflop
1Hz	8.7×10^{12}	∼280Tflop

Example 2: Weather/Climate modeling

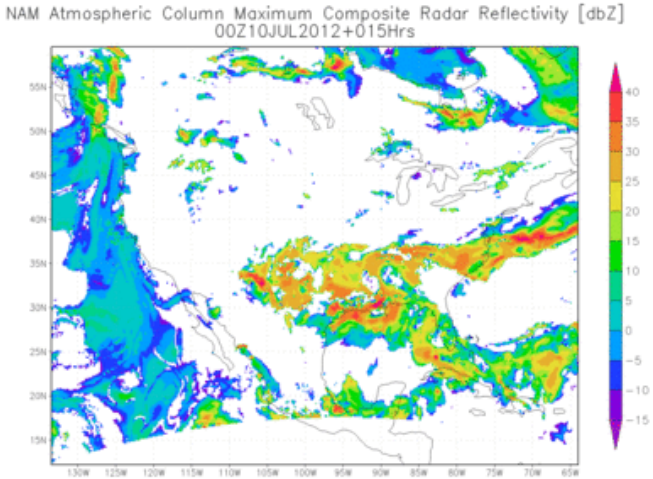
Target: Compute/predict weather or future climate.
Approach: Discretize governing PDEs (atmosphere, ocean, land ice, land models...)
Uses: Predict major events (Sandy, Katrina) and evaluate global warming scenarios



Simulated radar reflectivity (Source: NOAA).

Example 2: Weather/Climate modeling

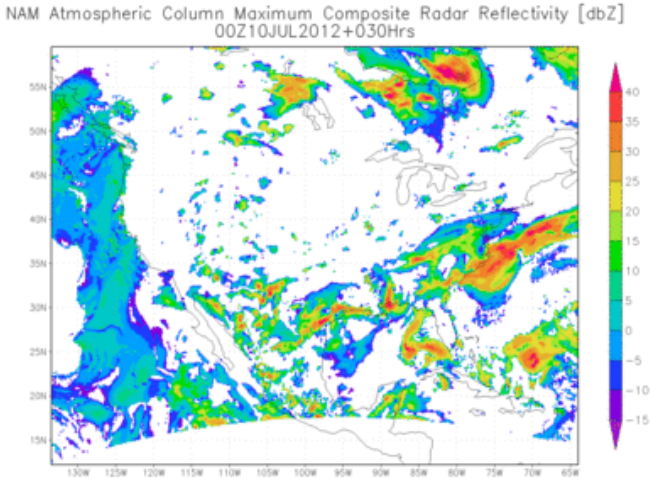
Target: Compute/predict weather or future climate.
Approach: Discretize governing PDEs (atmosphere, ocean, land ice, land models...)
Uses: Predict major events (Sandy, Katrina) and evaluate global warming scenarios



Simulated radar reflectivity (Source: NOAA).

Example 2: Weather/Climate modeling

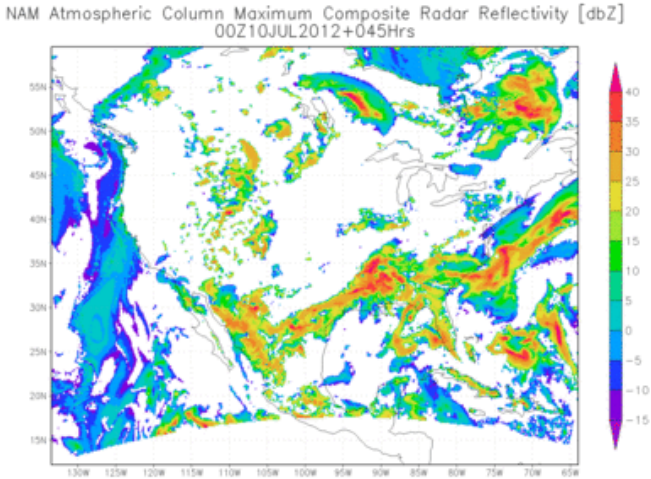
Target: Compute/predict weather or future climate.
Approach: Discretize governing PDEs (atmosphere, ocean, land ice, land models...)
Uses: Predict major events (Sandy, Katrina) and evaluate global warming scenarios



Simulated radar reflectivity (Source: NOAA).

Example 2: Weather/Climate modeling

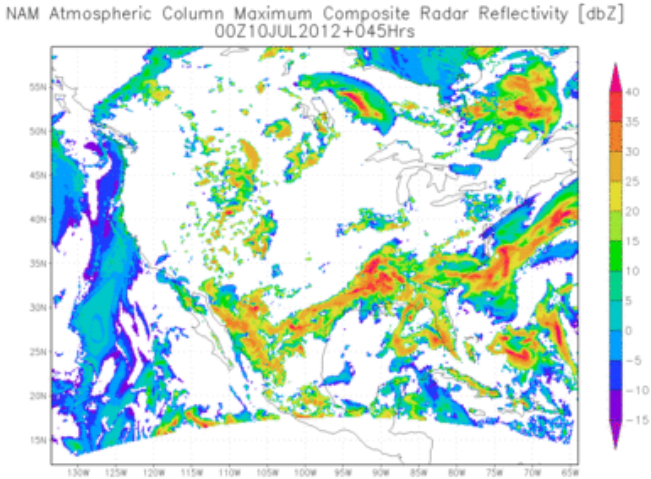
Target: Compute/predict weather or future climate.
Approach: Discretize governing PDEs (atmosphere, ocean, land ice, land models...)
Uses: Predict major events (Sandy, Katrina) and evaluate global warming scenarios



Simulated radar reflectivity (Source: NOAA).

Example 2: Weather/Climate modeling

Target: Compute/predict weather or future climate.
Approach: Discretize governing PDEs (atmosphere, ocean, land ice, land models...)
Uses: Predict major events (Sandy, Katrina) and evaluate global warming scenarios



Simulated radar reflectivity (Source: NOAA).

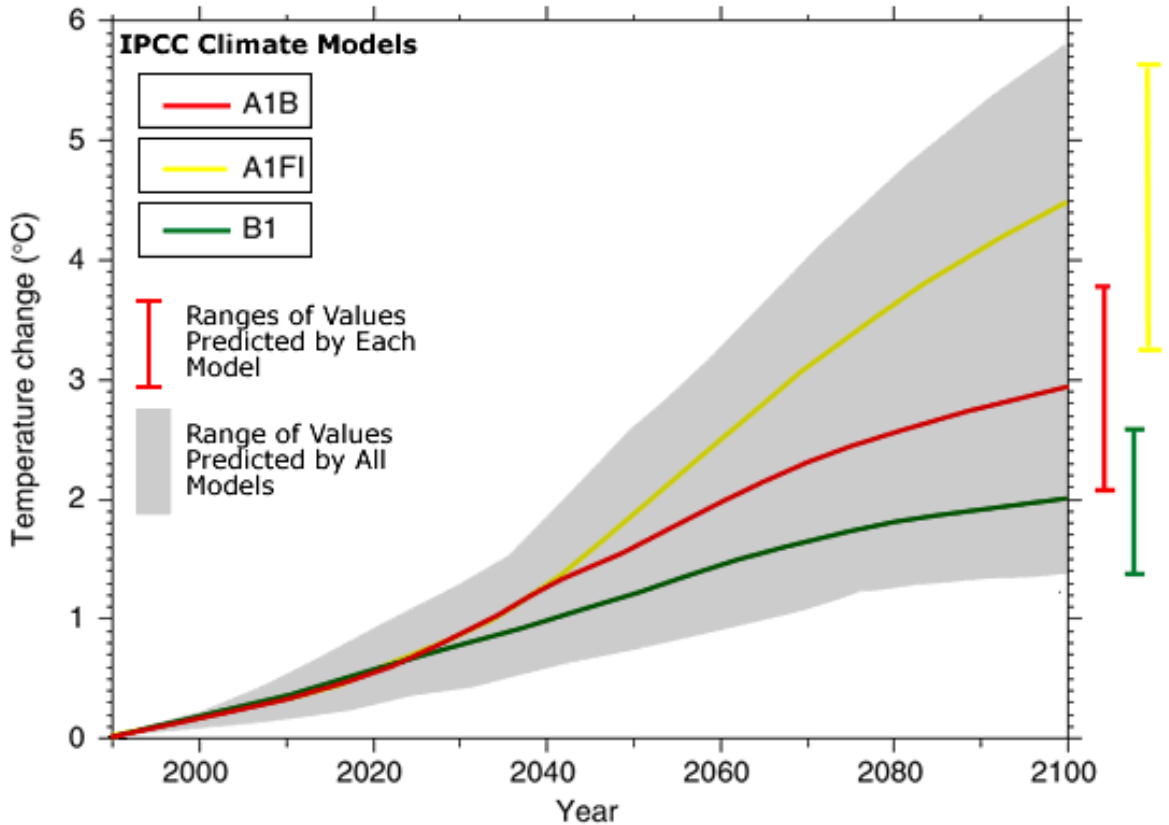
Weather prediction modeling (atmosphere only) requirements:

- For "real time" simulation: 8Gflop/s
- for weather prediction (7 days in 24h simulation): 56Gflop/s
- for climate prediction (50 years in 12 days) 288Tflop/s

Current weather models are often inaccurate due to coarse resolution (e.g., clouds not resolved).

Example 2: Weather/Climate modeling

Climate predictions based on different CO_2 scenaria (adapted from IPCC, 2001)



Climate models require coupled simulations of climate system (atmosphere+chemistry, land ice, oceans, sea ice) over long time scales (100, 200 years). Very challenging computations.

Outline

Demos, and preview

Can be downloaded as lecture1 from:
https://github.com/NYU-HPC19/