Scientific Computing
Important information

October 2, 2023

Skoltech

Skolkovo Institute of Science and Technology

The structure of the course

- **Lectures:** Nikolay Koshev
- **Classes & Labs:** Dmitry Yarotsky
- **Homework grading:** Daniil Rabinovich
- **Personal research project**
- **Final written exam**

The Final written exam

The exam ticket consists of:

- 5 tasks consisting of 2 theoretical questions each.
- The theoretical questions can be found at the end of each lecture.
- 7 problems.

The Personal research project

Personal Research Project (PRP): Scientific research on the topics of the course. The presentation of the project should contain:

1. Informal description of the goal of the research and motivation.
2. The clear mathematical statement of the problem.
3. The State-Of-The-Art techniques on the problem: literature.
4. Methods being used.
5. Results.
6. Discussion: conclusions, findings, comparison of used method with other methods.

PRP: Topics

Typical topics can be found on Canvas.
You are welcome to propose your own research satisfying the following conditions:

- Nonempty intersection with the course topics.
- It is not your bachelor's research.

The grading criteria

- **Class Participation:** 5%
- **Homework assignments:** 30%
- **Final written exam:** 35%
- **Personal research project:** 30%

The structure of the lectures

Lec. 1. Introduction to Scientific Computing. A bit of history, a bit of preliminary information. Introduction to HPC: basic terminology, principles, programming strategies. Introduction to parallel computing.

Lec. 2. Brief classification of the problems arising in Scientific Computing: Forward and Inverse, Well- and ill-posed, static and dynamic, etc. Brief consideration SLAE and functional spaces.

Lec. 3. Differential equations: differential operator, ordinary differential equations (ODE), three kinds of problems for differential equations (Cauchy, BV, Eigenvalue), PDEs of 1^{st} and 2^{nd} order.

Lec. 4. Integral transforms and integral equations: Fourier Transform, Convolution, Wavelets.

Lec. 5. Discretization and numerical methods: FDM, FEM, examples.

Lec. 6. Basics of optimization/mathematical programming: concept, problem classification, some of numerical approaches.

Scientific Computing
Lecture 1: Introduction

Nikolay Koshev, Dmitry Yarotsky, Maxim Fedorov

October 2, 2023

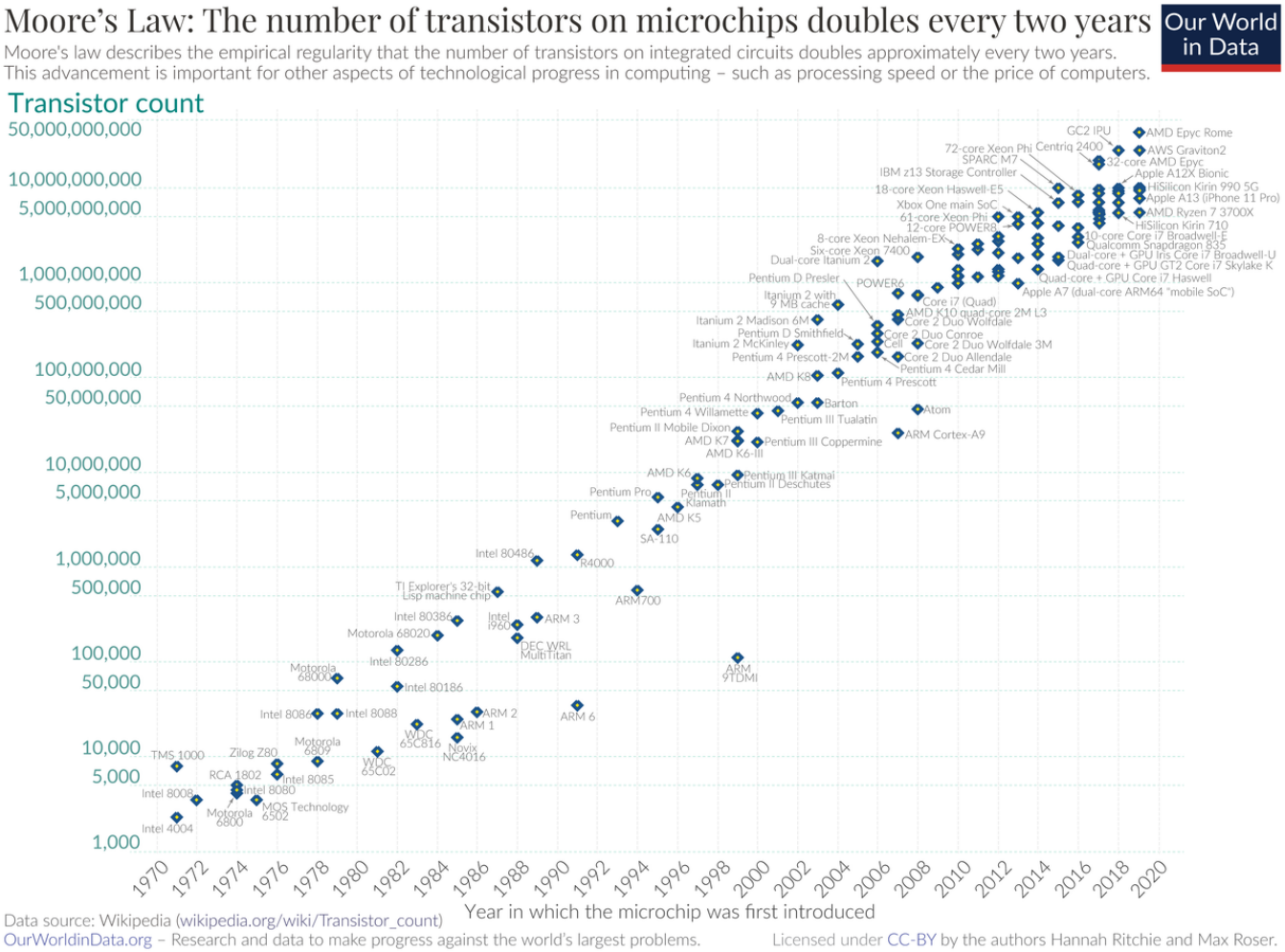
Tunnel Vision by Experts

- "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". **Ken Olson, president and founder of Digital Equipment Corporation, 1977**.
- "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, 1994**.

Slide source: Warfield et al.

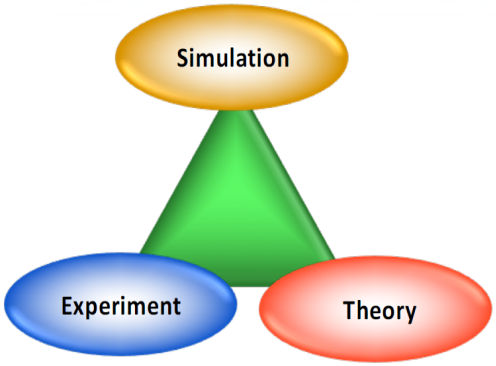
Moore's Law

2021: 50+ years of an exponential growth



Skoltech
Skolkovo Institute of Science and Technology

Simulation: The Third Pillar of Science



- **Traditional scientific and engineering method:**
  - Do **theory** of paper design;
  - Perform **experiments** or build system
- **Limitations**
  - Too difficult - build large wind tunnels;
  - Too expensive - build a throw-away passanger jet;
  - Too slow - wait for climate of galactic evolution;
  - Too dangerous - weapons, drug design, climate experimentation.

- **Computational science and engineering paradigm**:
  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 manually.

Traditional and modern methods

Traditional method: theory + experiment

1

Develop a theory or
Make a drawing

2

Conduct experiments
Or create a system

Restrictions – the experiments can be:

- Too difficult (e.g. to build large wind tunnels)
- Too expensive (e.g. to build a one-time passenger plane)
- Too slow (e.g. to wait for climatic or galactic changes)
- Too dangerous (e.g. weapons, drug development, climate experiments)

Modern method: modeling

1

Use computers to model and analyze phenomena: two main approaches

- Models based on known physical laws and effective numerical methods
- Models based on data:

Data → A model based on data

Failures history → A model that predicts failures

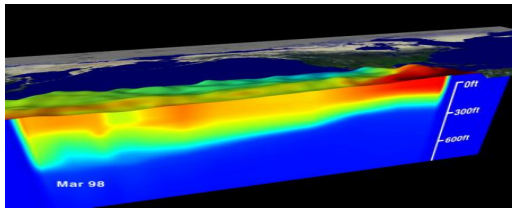
CompSci & Eng: Drivers for Change

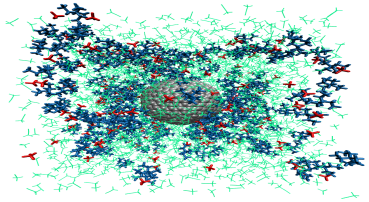
- Continued exponential increase in computational power simulation is becoming third pillar of science, complementing theory and experiment
  - Can simulate what theory and experiment can't do
- Continued exponential increase in experimental data techniques and technology in data analysis, visualization, analytics, networking, and collaboration tools are becoming essential in all data rich scientific applications
  - Moore's Law applies to sensors too
  - Need to analyze all that data

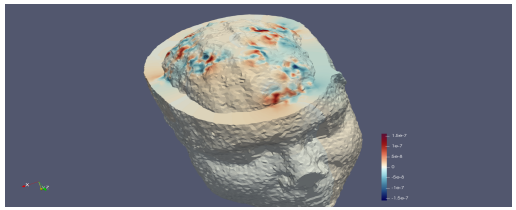
CompSci & Eng: Application areas

Application areas that benefit from usage of Computational Science and Engineering

- Computational Physics,
- Computational Fluid Dynamics,
- Bioinformatics,
- Modelling of Oil & Gas reservoirs,
- Weather forecasts,
- Vizualisation,
- Large Data analysis,
- Renewable Energy,
- Large Data and Smart Cities applications
- and many others.





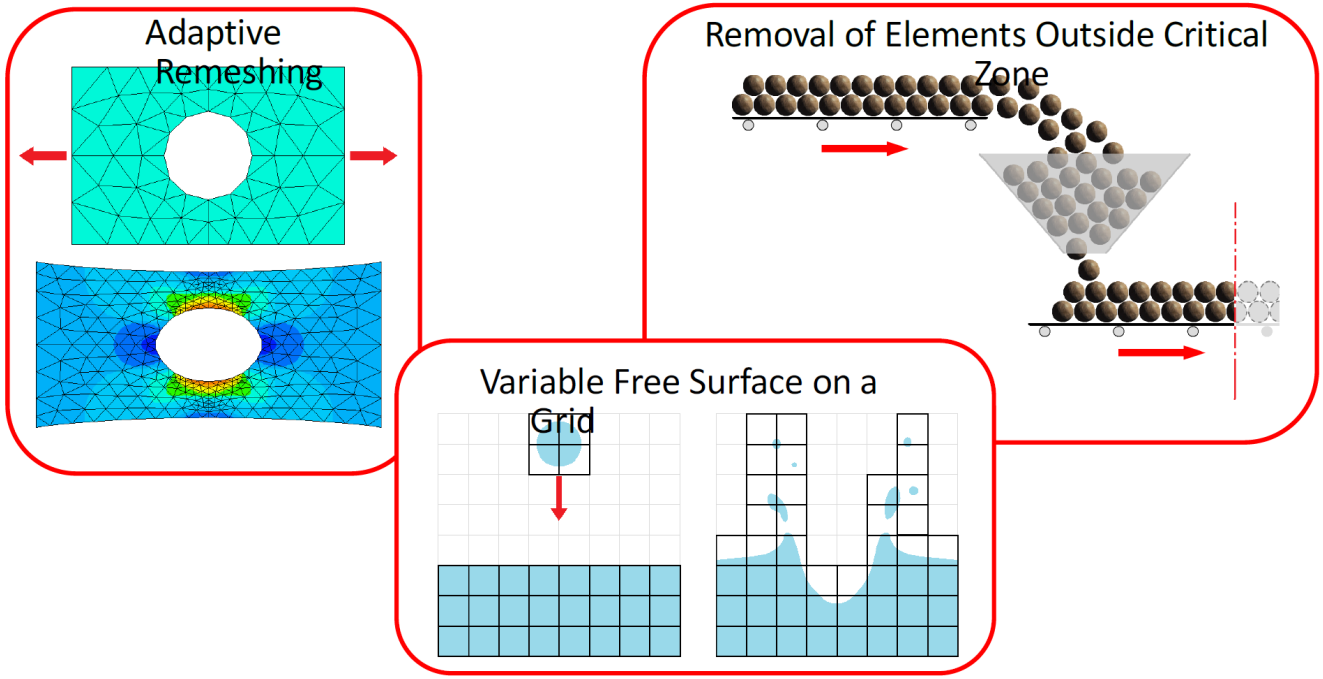


Skoltech
Skolkovo Institute of Science and Technology

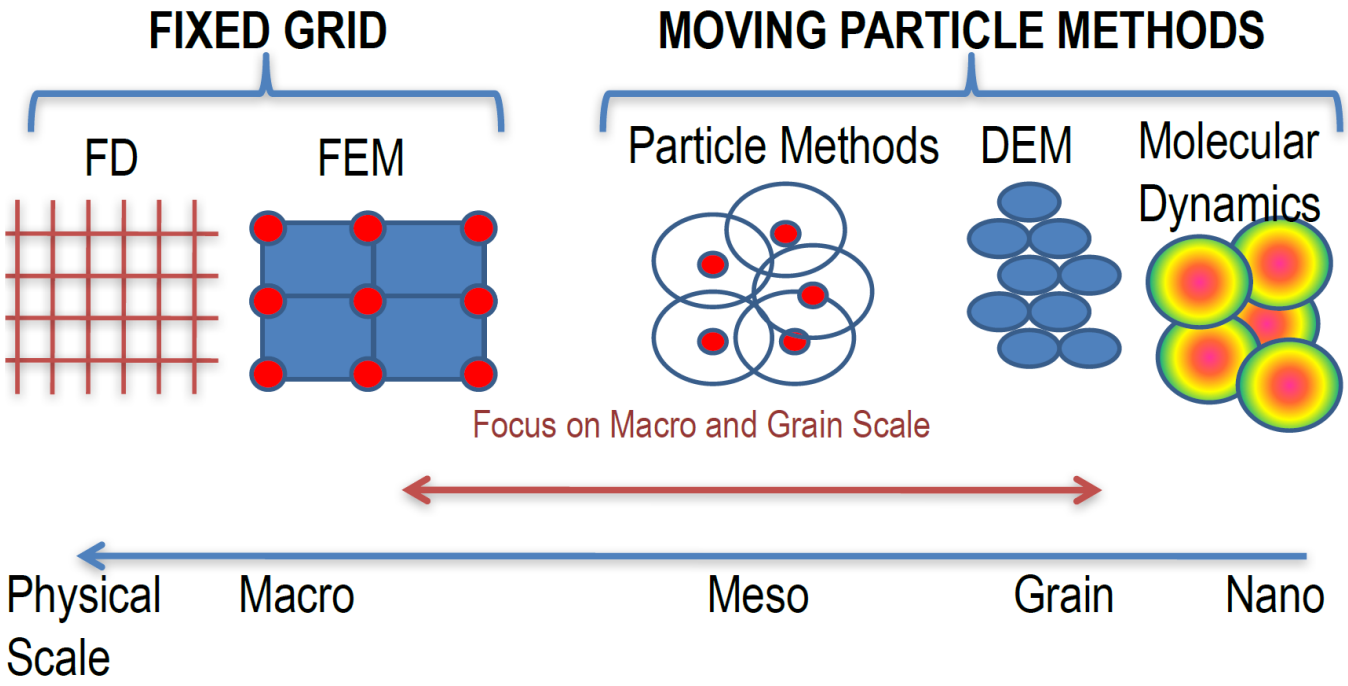
Challenges to Concurrency

- General challenges
- Synchronization
- Thread safety
- Load balance
- Spatial reasoning and task distribution
- Dynamic evolution of numerical tasks

Dynamic Evolution of the Numerical Task

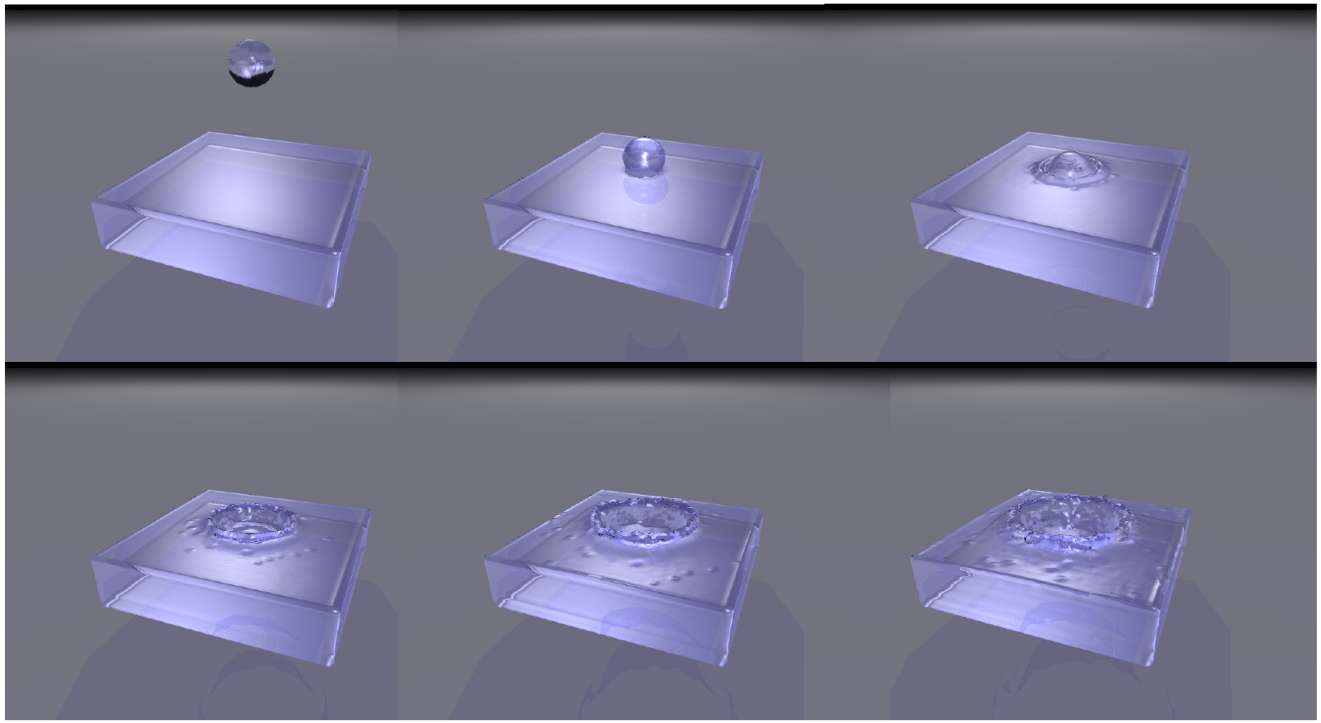


Computational methods Focus on Grain to Macro Scale Analysis



Testing and applications

Falling drop example: 200000 particles, 10000 time steps



Scientific Computing Lecture 1: Intro to HPC

Nikolay Koshev, Dmitry Yarotsky, Maxim Fedorov

October 2, 2023

NB: the further slides are based on introductory courses to HPC & Parallel Computing from Berkley ParaLab, University of Reims, Boston University (Doug Sondak), and Lawrence Livermore National Laboratory

Units of Measure

- High Performance Computing (HPC) units are:
  - Flop: floating point operation, usually double precision unless noted
  - Flop/s: floating point operations per second
  - Bytes: size of data (a double precision floating point number is 8)
- Typical sizes are millions, billions, trillions...

  Mega: Mflop/s $= 10^6$ flop/sec; Mbyte $= 10^6$ bytes (Mebibyte $= 2^{20}$ bytes, MiB)
  Giga: Gflop/s $= 10^9$ flop/sec; Gbyte $= 10^9$ bytes
  Tera: Tflop/s $= 10^{12}$ flop/sec; Tbyte $= 10^{12}$ bytes
  Peta: Pflop/s $= 10^{15}$ flop/sec; Pbyte $= 10^{15}$ bytes (Pebibyte $= 2^{50}$ bytes, PiB)
  Exa: Eflop/s $= 10^{18}$ flop/sec; Ebyte $= 10^{18}$ bytes
  Zetta: Zflop/s $= 10^{21}$ flop/sec; Zbyte $= 10^{21}$ bytes
  Yotta: Yflop/s $= 10^{24}$ flop/sec; Ybyte $= 10^{24}$ bytes
- Current fastest machine - 1102 Pflop/s (Frontier, Oak Bridge National Laboratory, USA). Up-to-date list at www.top500.org;
- Current fastest machine in Russia is "Chervonenkis", Yandex, Moscow (app. 30 Pflop/s) see http://top50.supercomputers.ru/?page=rating

The TOP500 Project

- Listing the 500 most powerful computers in the world
- Yardstick: Rmax of Linpack - Solve Ax=b, dense problem, matrix is random - Dominated by dense matrix-matrix multiply
- Update twice a year: - ISC'xy in June in Germany - SCxy in November in the U.S.
- All information available from the TOP500 web site at: www.top500.org

Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x ?
- Clock rate goes up by x because wires are shorter - actually less than x, because of power consumption
- Transistors per unit area goes up by x^2
- Die size also tends to increase (typically another factor of x)
- Raw computing power of the chip goes up by x^4! (In practice, typically, proportionally to x^3)
- parallelism: hidden parallelism such as ILP
- locality: caches
- So might the programs perform x^3 times faster, without changing them?

Parallelism nowadays

- These arguments are no longer theoretical
- All major processor vendors are producing multicore chips - Almost every machine is in fact is a parallel machine - To keep doubling performance, parallelism must double
- Which (commercial) applications can use this parallelism? - Do they have to be rewritten from scratch?
- Will all programmers have to be parallel programmers? - New software model needed - New compilers would hide complexity from most programmers - eventually - In the meantime, they would need to understand it
- Computer industry betting on this big change, but does not have all the answers

Memory is Not Keeping Pace

Technology trends against a constant or increasing memory per core

- Memory density is doubling every three years; processor logic is every two
- 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

Parallel Speedup

- Parallel speedup: how much faster does my code run in parallel compared to serial?
- Max value is P, the number of processors

$$\text{Parallel Speedup} = \frac{\text{Serial execution time}}{\text{parallel execution time}}$$

Parallel Efficiency

- Parallel efficiency: how much faster does my code run in parallel compared to linear speedup from serial?
- Max value is 1

$$\text{Parallel efficiency} = \frac{\text{Parallel speedup}}{P}$$

Principles of Parallel Computing

- Finding enough parallelism (Amdahl's Law, Gustafson's Law)
- Granularity - how big should each parallel task be
- Locality - moving data costs more than arithmetic
- Load balance - don't want 1000 processors to wait for one slow one
- Coordination and synchronization - sharing data safely
- Performance modelling/debugging/tuning

All of these things make parallel programming more challenging than sequential programming

"Automatic" Parallelism in Modern Machines

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

Limits to all of these - for very high performance, need user to identify, schedule and coordinate parallel tasks

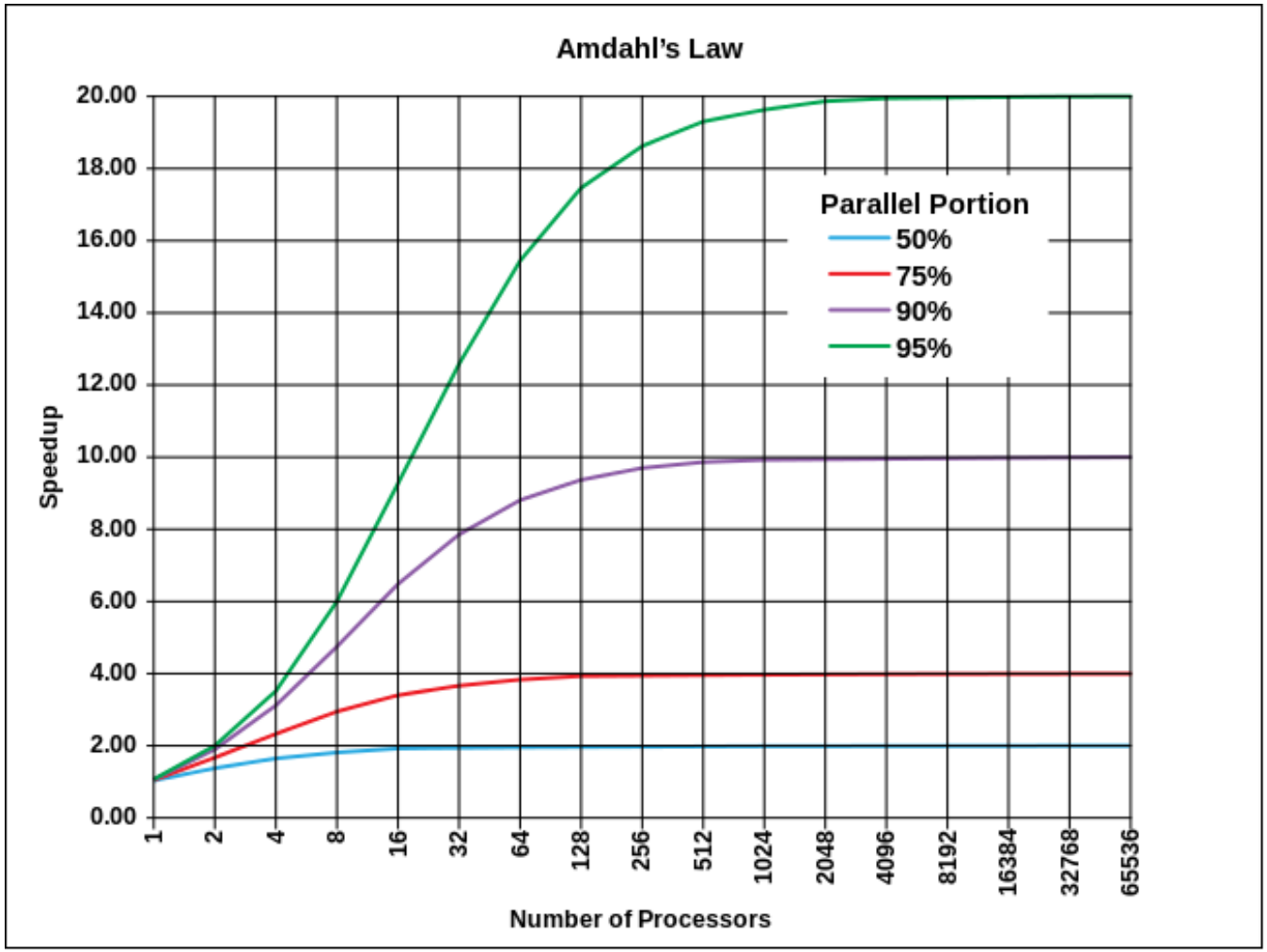
Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl's law - let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable
  - P = number of processors

$$\text{Speedup}(P) = \frac{\text{Time}(1)}{\text{Time(P)}} = \frac{1}{s + \frac{1-s}{P}} \quad (1)$$

- Even if the parallel part speeds up perfectly **performance is limited by the sequential part**

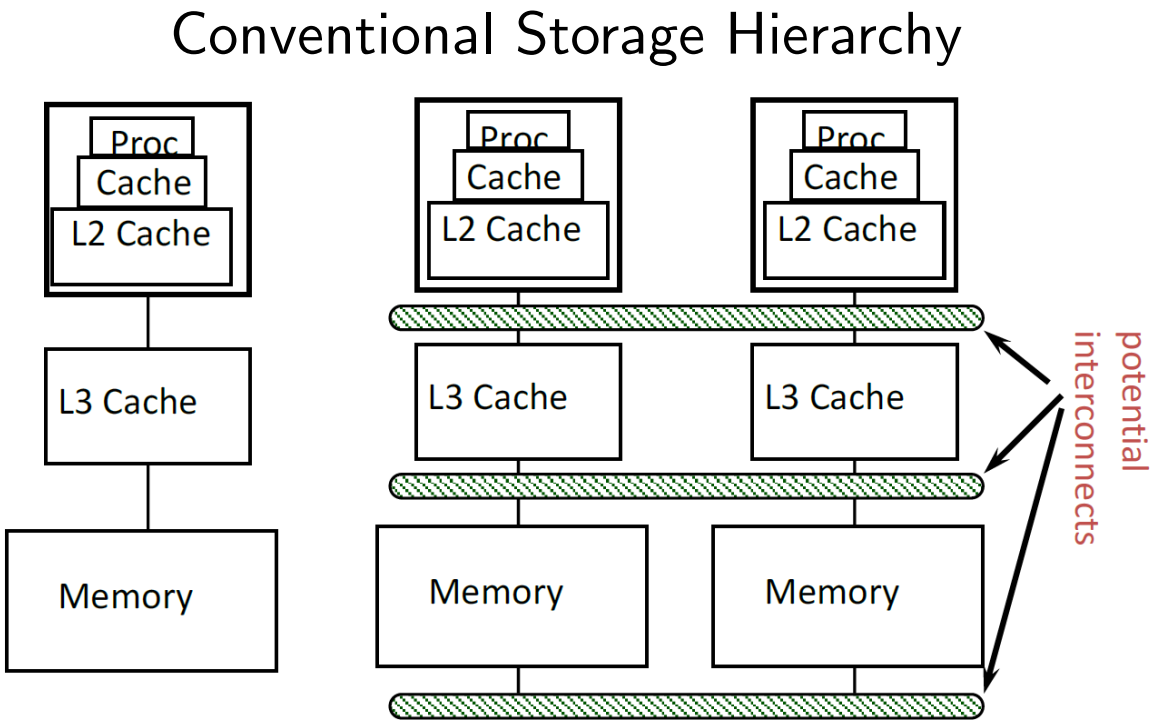
Amdahl's law illustration



Overhead of Parallelism

- 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 (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work

Locality and Parallelism



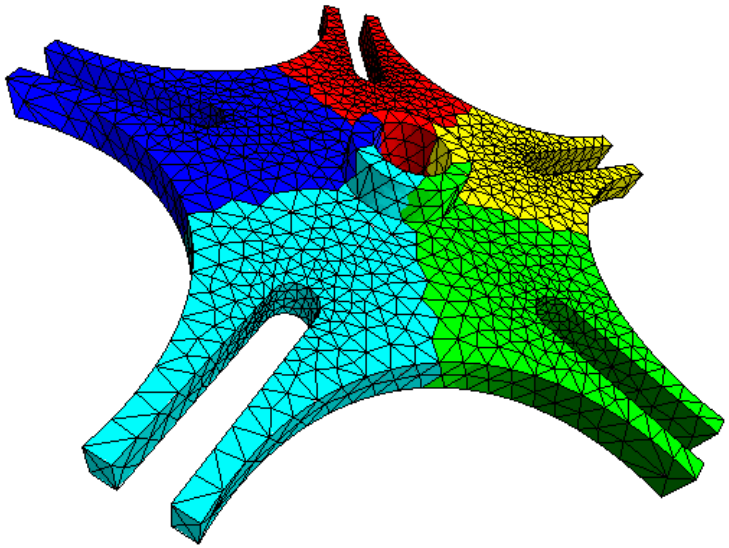
- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast on average
- Parallel processors, collectively, have large, fast cache
  - the slow accesses to "remote" data we call "communication"
- Algorithm should do most work on local data

Load Imbalance

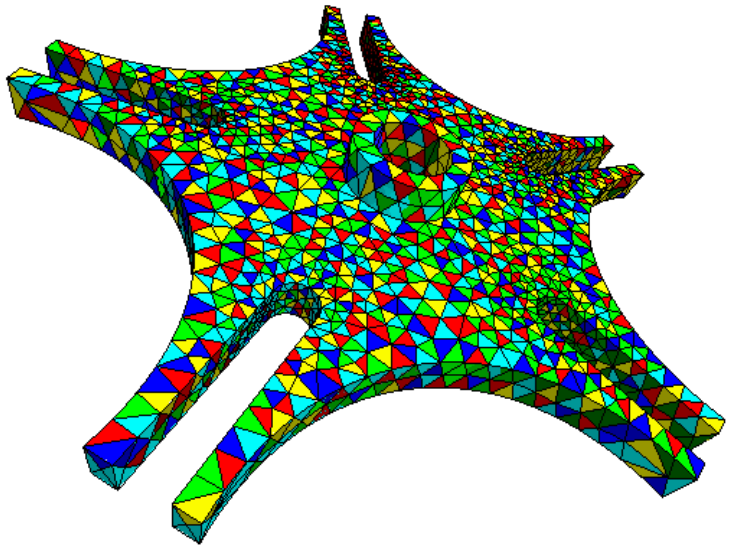
- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - adapting to "interesting parts of a domain"
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm 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"**

Spatial reasoning and task distribution

Domain Decomposition
(Predictive Load Balance)



Domain Distribution
(Events Based Load Balance)



Parallel Software Eventually - a view from Berkley (ParLab)

- 2 types of programmers and 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
- Expect students may want to work at either level
  - In the meantime, we all need to understand enough of the efficiency layer to use parallelism effectively

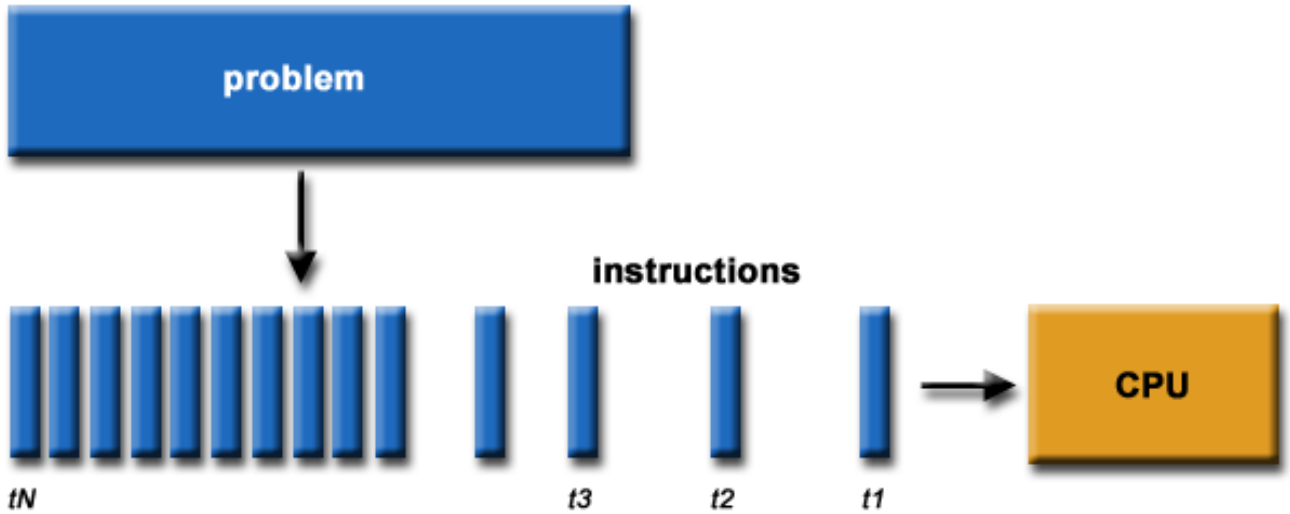
Parallel Computing: basic principles

Abstract

This presentation covers the basics of parallel computing. Beginning with a brief overview and some concepts and terminology associated with parallel computing, the topics of parallel memory architectures and programming models are then explored. These topics are followed by a discussion on a number of issues related to designing parallel programs. The last portion of the presentation is spent examining how to parallelize several different types of serial programs.

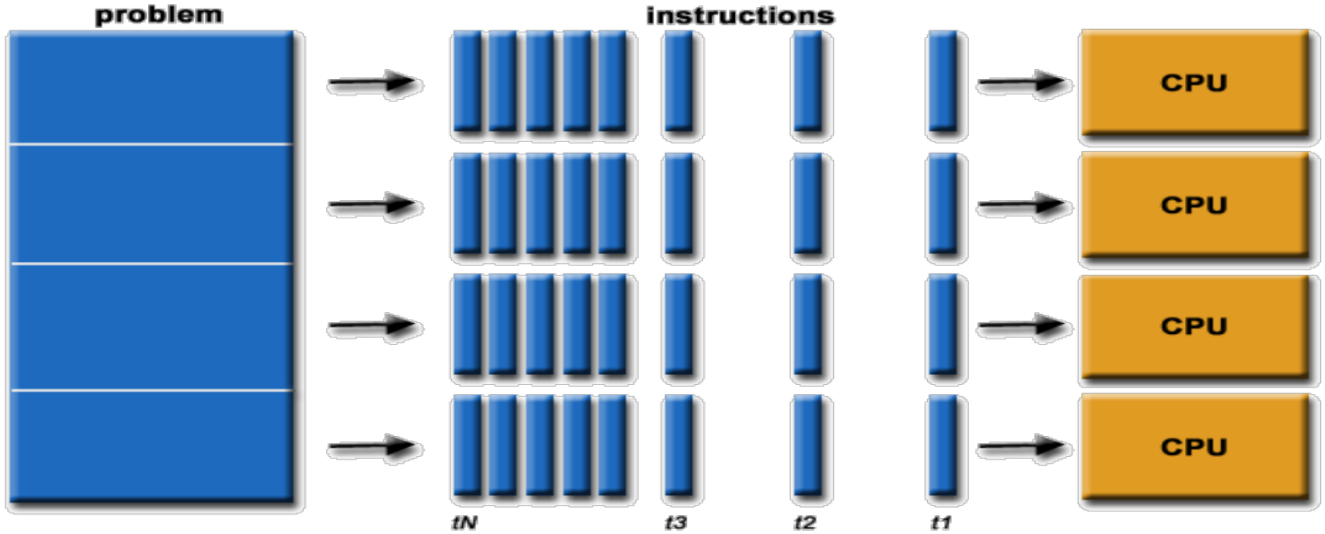
What is Parallel Computing? (1)

- Traditionally, software has been written for serial computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.



What is Parallel Computing? (2)

- In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs



Parallel Computing: Resources

The compute resources can include:

- A single computer with multiple processors;
- A single computer with (multiple) processor(s) and some specialized computer resources (GPU, FPGA etc)
- An arbitrary number of computers connected by a network;
- A combination of both.

Parallel Computing: The computational problem

The computational problem usually demonstrates characteristics such as the ability to be:

- Broken apart into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;
- Solved in less time with multiple compute resources than with a single compute resource.

Parallel Computing: what for? (1)

Parallel computing is an evolution of serial computing that attempts to emulate what has always been the state of affairs in the natural world: many complex, interrelated events happening at the same time, yet within a sequence.

Some examples:

- Planetary and galactic orbits
- Weather and ocean patterns
- Tectonic plate drift
- Rush hour traffic in Paris
- Automobile assembly line
- Daily operations within a business
- Building a shopping mall
- Ordering a hamburger at the drive through.

Parallel Computing: what for? (2)

Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:

- weather and climate
- chemical and nuclear reactions
- biological, human genome
- geological, seismic activity
- mechanical devices - from prosthetics to spacecraft
- electronic circuits
- manufacturing processes

Parallel Computing: what for? (3)

- Today, commercial applications are providing an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. Example applications include:
  - parallel databases, data mining
  - oil exploration
  - web search engines, web based business services
  - computer-aided diagnosis in medicine
  - management of national and multi-national corporations
  - advanced graphics and virtual reality, particularly in the entertainment industry
  - networked video and multi-media technologies
  - collaborative work environments
- Ultimately, parallel computing is an attempt to maximize the infinite but seemingly scarce commodity called time.

Why Parallel Computing? (1)

- This is a legitime question! Parallel computing is complex on any aspect!
- The primary reasons for using parallel computing:
  - Save time - wall clock time
  - Solve larger problems
  - Provide concurrency (do multiple things at the same time)

Why Parallel Computing? (2)

Other reasons might include:

- Taking advantage of non-local resources - using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
- Cost savings - using multiple "cheap" computing resources instead of paying for time on a supercomputer.
- Overcoming memory constraints - single computers have very finite memory resources. For large problems, using the memories of multiple computers may overcome this obstacle.

Limitations of Serial Computing

- **Limits to serial computing** - both physical and practical reasons pose significant constraints to simply building ever faster serial computers.
- **Transmission speeds** - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
- **Limits to miniaturization** - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- **Economic limitations** - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

The future

- during the past 10 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that parallelism is the future of computing.
- It will be multi-forms, mixing general purpose solutions (your PC...) and very speciliazed solutions as IBM Cells, ClearSpeed, GPGPU from NVidia...

Who and What? (1)

- Top500.org provides statistics on parallel computing users - the charts below are just a sample. Some things to note: - Sectors may overlap - for example, research may be classified research. Respondents have to choose between the two.
- "Not Specified" is by far the largest application - probably means multiple applications.

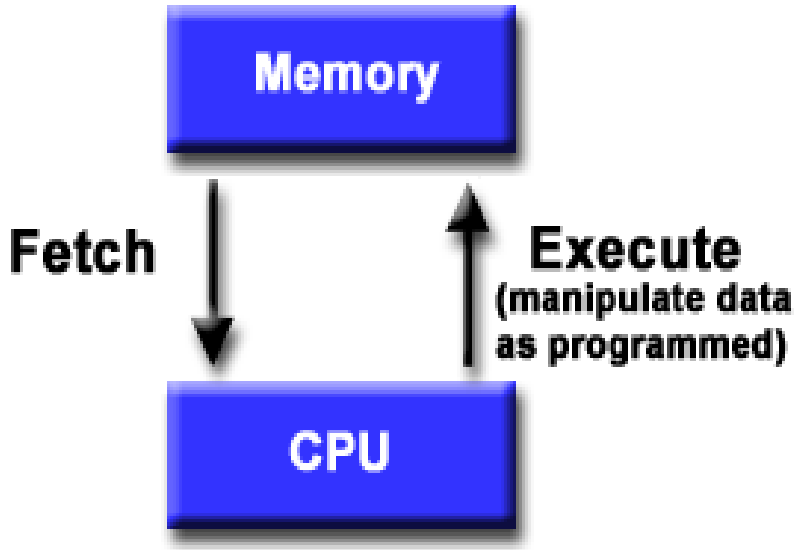
Concepts and Terminology

Von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. Named after the Hungarian mathematician John von Neumann.
- A von Neumann computer uses the storedprogram concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

Basic Design

- Basic design
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then **sequentially** performs them.



Main principles of writing a good code

- Optimization
  - Profile serial (1-processor) code
    - Tells where most time is consumed
  - Is there any "low fruit"?
    - Faster algorithm
    - Optimized library
    - Wasted operations
- Parallelization
  - Break problem up into chunks
  - Solve chunks simultaneously on different processors

Software

Compiler

- The compiler is your friend (usually)
- Optimizers are quite refined
  - Always try highest level
    - Usually -O3
    - Sometimes -fast, -Os,...
- Loads of flags, many for optimization
- Good news - many compilers will automatically parallelize for shared-memory systems
- Bad news - this usually doesn't work well

Libraries

- Solver is often a major consumer of CPU time
- Numerical Recipes is a good book, but many algorithms are not optimal
- Lapack is a good resource
- Libraries are often available that have been optimized for the local architecture
  - Disadvantage - not portable

Parellelization

Parallelization

Divide and conquer!

- divide operations among many processors
- perform operations simultaneously
- if serial run takes 10 hours and we hit the problem with 5000 processors, it should take about 7 seconds to complete, right?
  - not so easy, of course

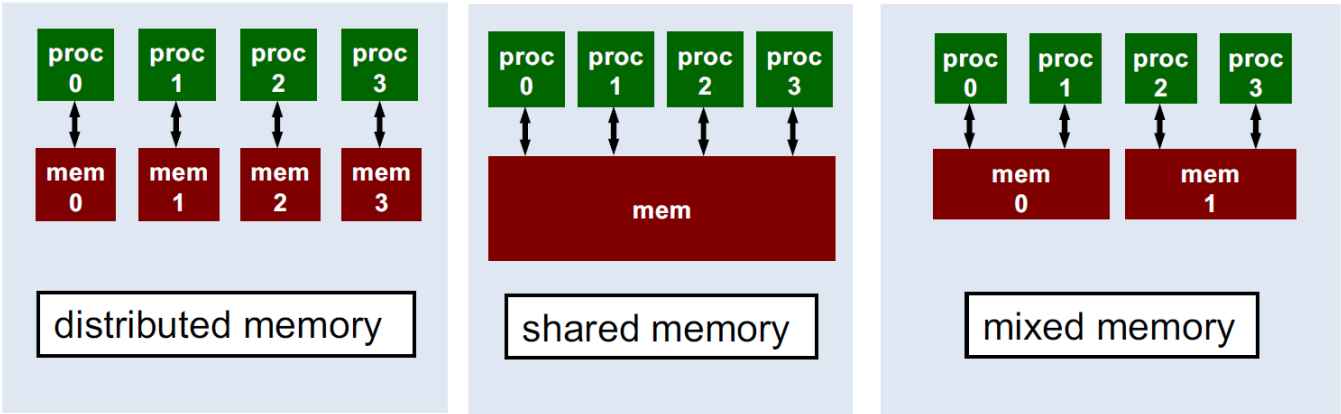
Parallelization (cont'd)

- problem - some calculations depend upon previous calculations
  - can't be performed simultaneously
  - sometimes tied to the physics of the problem, e.g., time evolution of a system
- want to maximize amount of parallel code
  - occasionally easy
  - usually requires some work

Parallelization (3)

- Method used for parallelization may depend on *hardware*
- Distributed memory
  - each processor has own address space
  - if one processor needs data from another processor, must be explicitly passed
- Shared memory
  - common address space
  - no message passing required

Parallelization (4)



Parallelization (5)

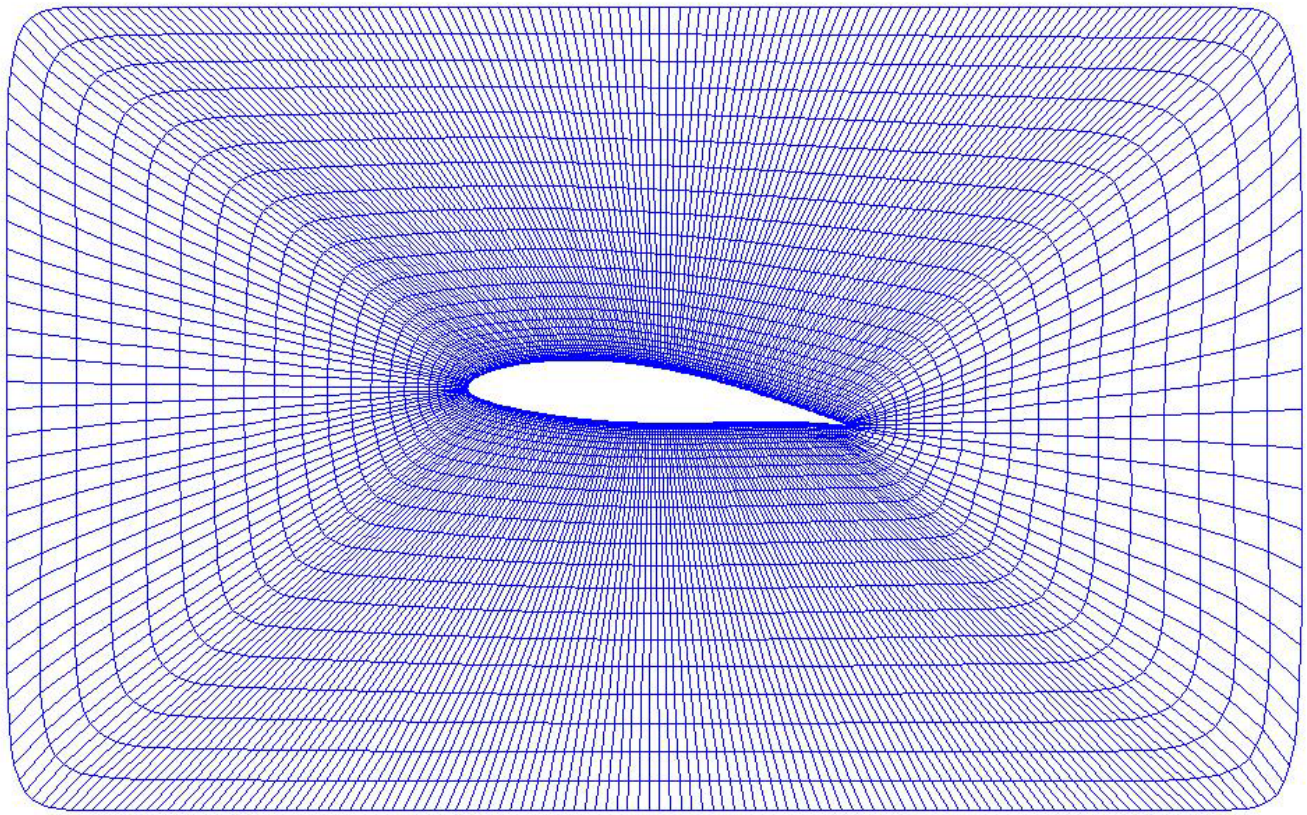
- Message Passing Interface (MPI)
  - for both distributed and shared memory
  - portable
  - freely downloadable
- Open Multi-Processing (OpenMP)
  - shared memory only
  - must be supported by compiler (most do)
  - usually easier than MPI
  - can be implemented incrementally

MPI

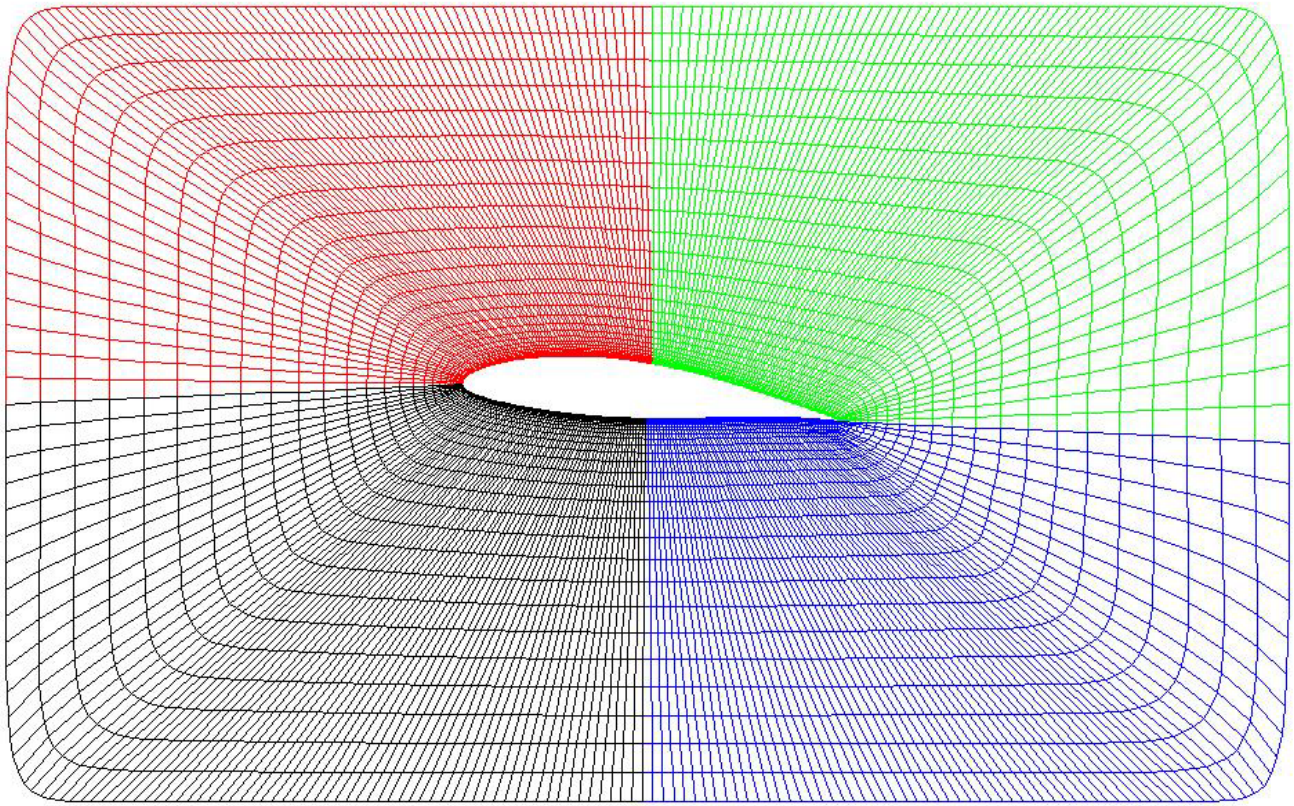
- Computational domain is typically decomposed into regions
  - One region assigned to each processor
- Separate copy of program runs on each processor

MPI

- Discretized domain to solve flow over airfoil
- System of coupled PDE's solved at each point



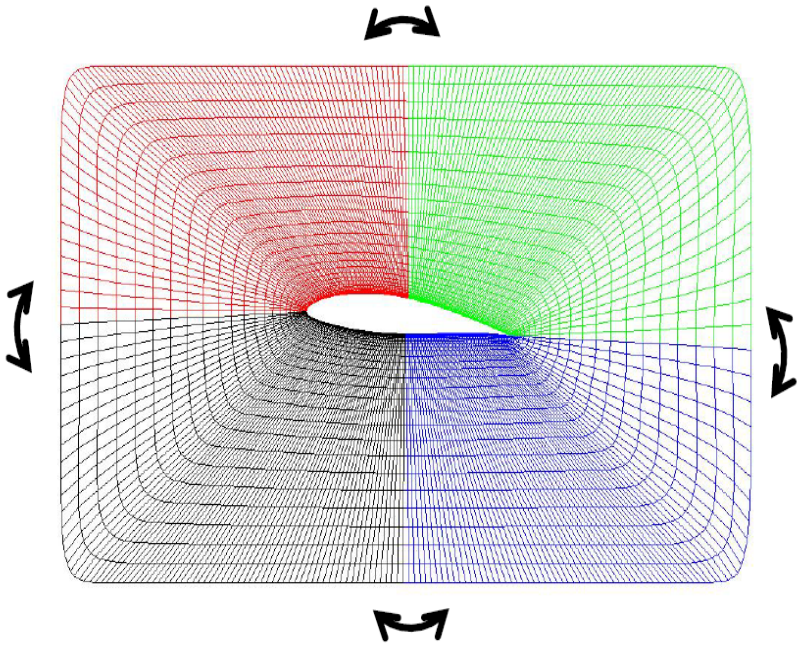
- Decomposed domain for 4 processors



MPI

- Since points depend on adjacent points, must transfer information after each iteration
- This is done with explicit calls in the source code

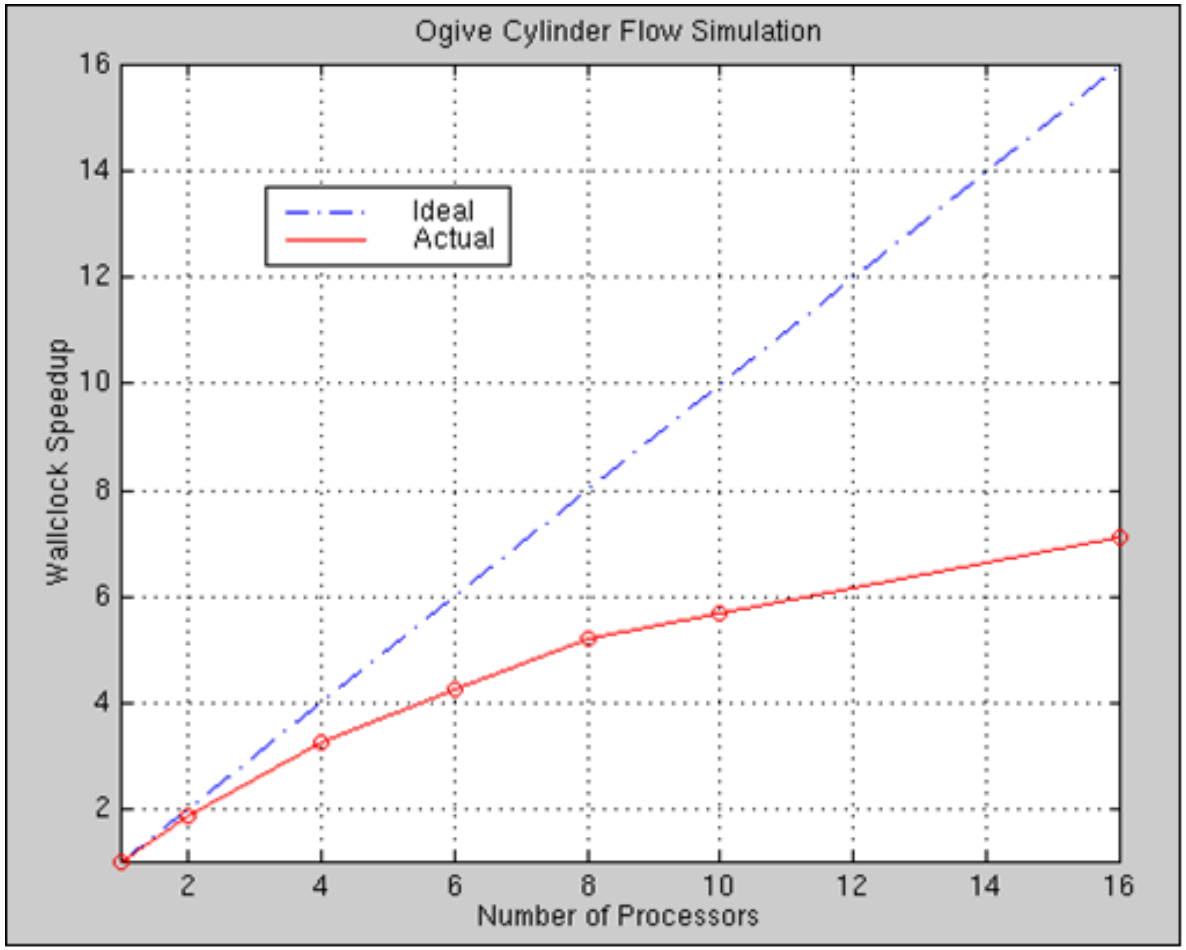
$$\frac{\partial \varphi_i}{\partial x} \approx \frac{\varphi_{i+1} - \varphi_{i-1}}{2\Delta x}$$



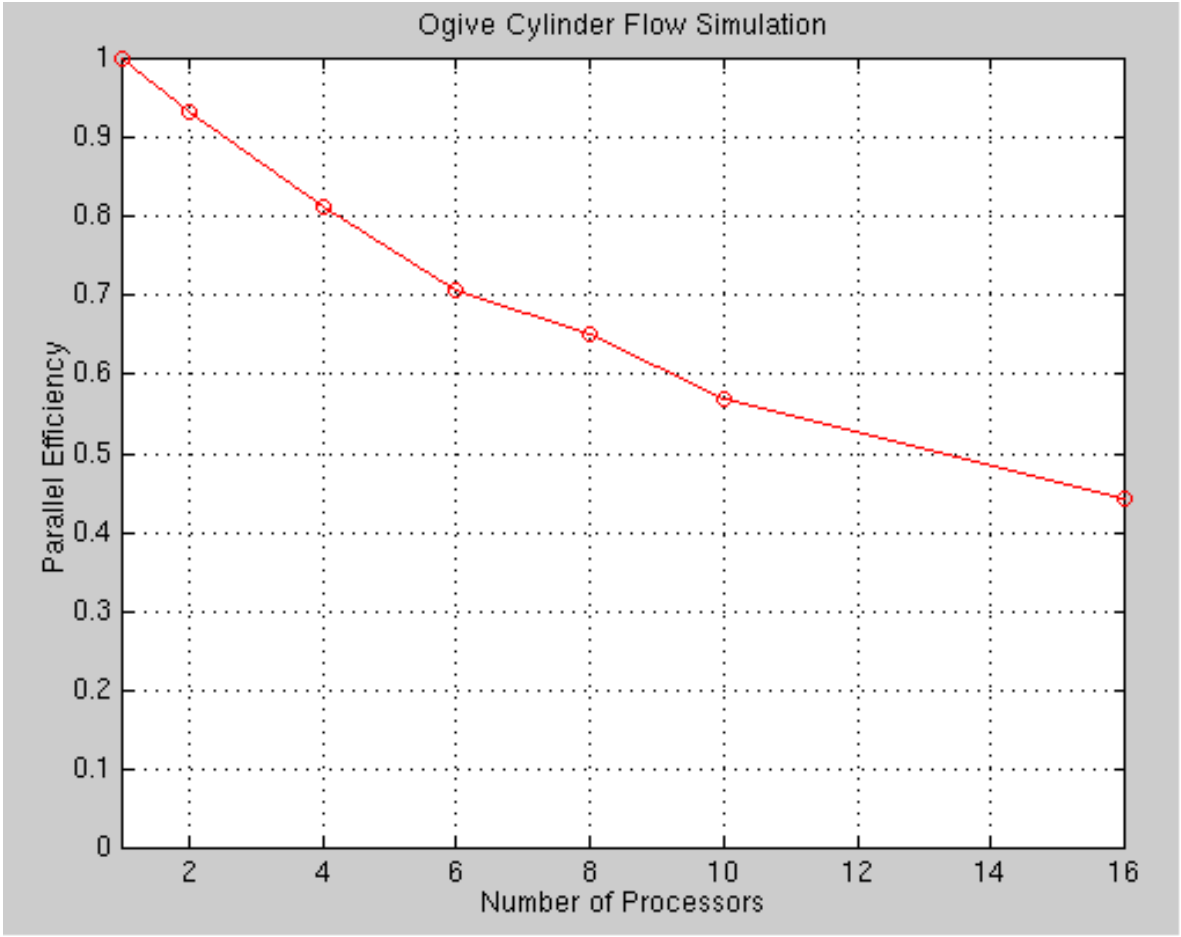
MPI

- Diminishing returns
  - Sending messages can get expensive
  - Want to maximize ratio of computation to communication
- Parallel speedup, parallel efficiency

Speedup



Parallel Efficiency



- Usually loop-level parallelization

1: **for** $i = 0$ to N **do**
   lots of stuff
2: **end for**

- An OpenMP directive is placed in the source code before the loop
  - Assigns subset of loop indices to each processor
  - No message passing since each processor can "see" the whole domain

Can't guarantee order of operations

```
for(i = 0; i < 7; i++)
    a[i] = 1;
for(i = 1; i < 7; i++)
    a[i] = 2*a[i-1];
```

Example of how to do it wrong!

← Parallelize this loop on 2 processors

i	a[i] (serial)	a[i] (parallel)
0	1	1
1	2	2
2	4	4
3	8	8
4	16	2
5	32	4
6	64	8

Proc. 0

Proc. 1

Questions? Comments?

Thank you for your attention!

Questions (Lecture 1)

- Traditional scientific and engineering approach includes only theory and experiment. Provide a reasoning on disadvantages/limitations of such approach. How the mathematical modelling might help to manage it?
- Formulate the Moore's Law. Provide a reasoning on its value for science, industry, and every-day life.
- List the computational approaches in terms of physical sizes scale (macro to nano). Explain the main idea meshed/meshfree approaches. Does the computational complexity of the problem depend on usage of concrete kind of mwthods with respect to the scale?
- List the HPC units; explain the meaning of each one.
- What will happen when the transistor size shrinks by some factor x? Explain in terms of clock rate, unit area, raw computer power and shrinkage of performance of the program.
- Explain the terms of parallel speedup and parallel efficiency. Might the parallel efficiency metric be equal to 1?
- Formulate the Amdahl's law and provide reasoning on it: which overheads lead to limitation of the speedup?
- List all overheads of Parallelism you know. Reason on the ways of overcoming some of them.
- List the limitations of serial computing. Explain each one in a few words.
- Explain what do the terms "load balance" and "load imbalance" mean? By what can be the imbalance caused? Provide examples.
- What kind of parallel computing resources do you know? List them and justify the answer: why the listed resources can be considered parallel?
- List at least five applications demanding parallel computations. Justify your answer.
- Describe the main idea of the von Neumann architecture.
- Which kinds of software we have to use in order to write a solver for a complicated problem? Which ways (in relation to these kinds of software) can be used for optimization of your program?
- Explain the meanings of shared and distributed memory. Explain the OpenMP and MPI paradigms. Compare them and analyse advantages and drawbacks/risks. Provide reasoning on when it is better to use one or another.