Scientific Computing Introduction to HPC

NB: the 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

August 26, 2020

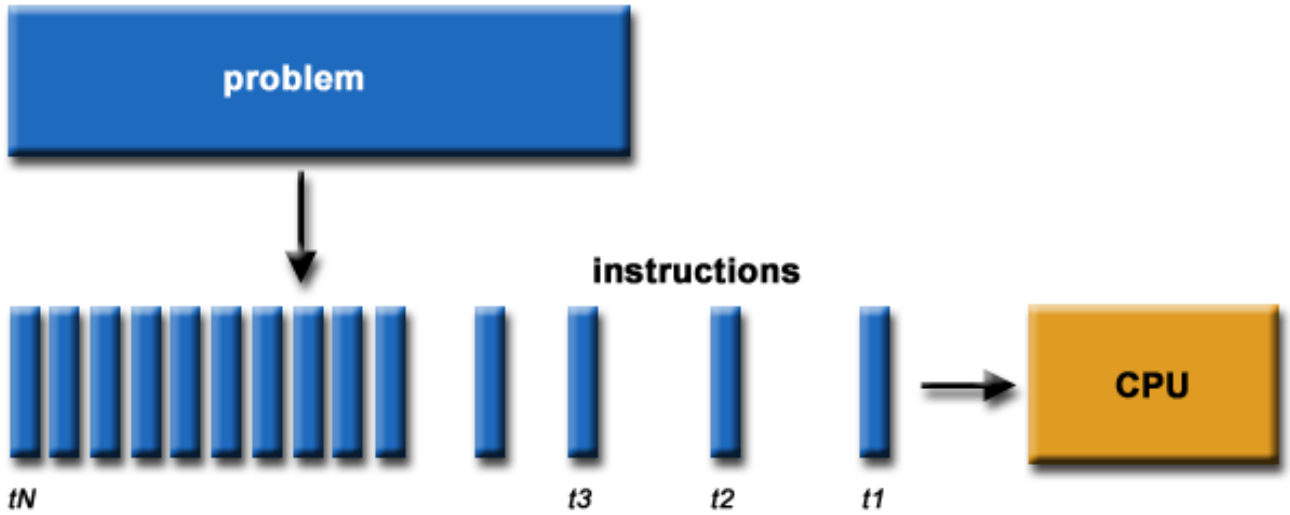
Skoltech

Skolkovo Institute of Science and Technology

Serial computing

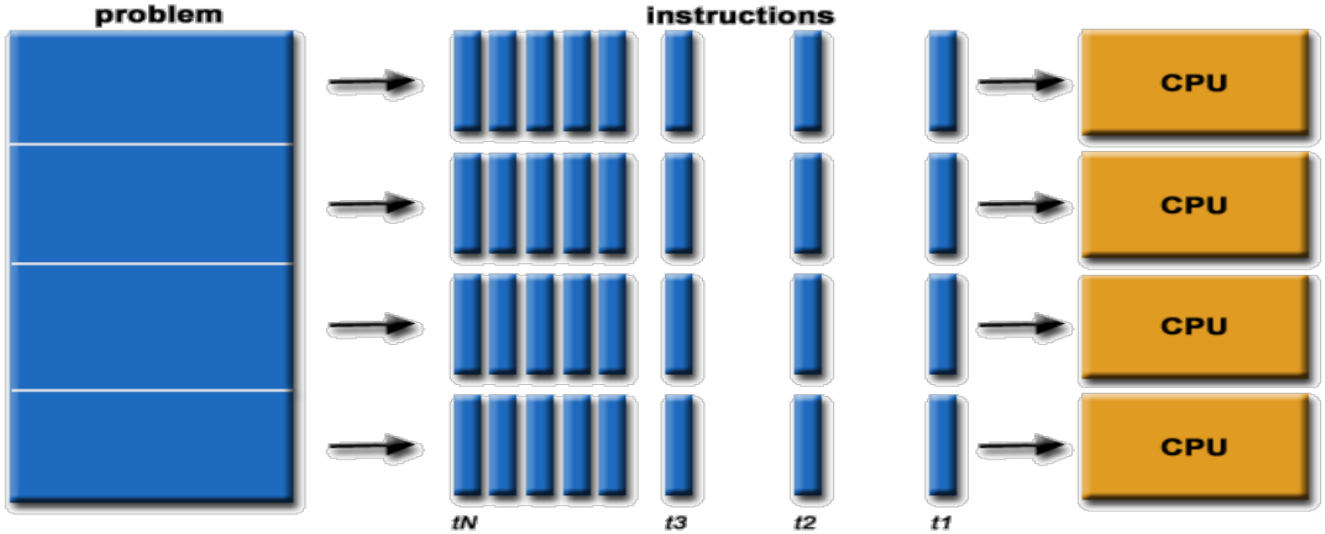
Traditionally, software has been written for serial computation:

- 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.



Parallel computing

- 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: source

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: motivation

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: industry

- 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?

- 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?

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

Both physical and practical reasons pose significant constraints to simply building ever faster serial computers.

- **Transmission speeds**
- **Limits to miniaturization**
- **Economic limitations**

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...

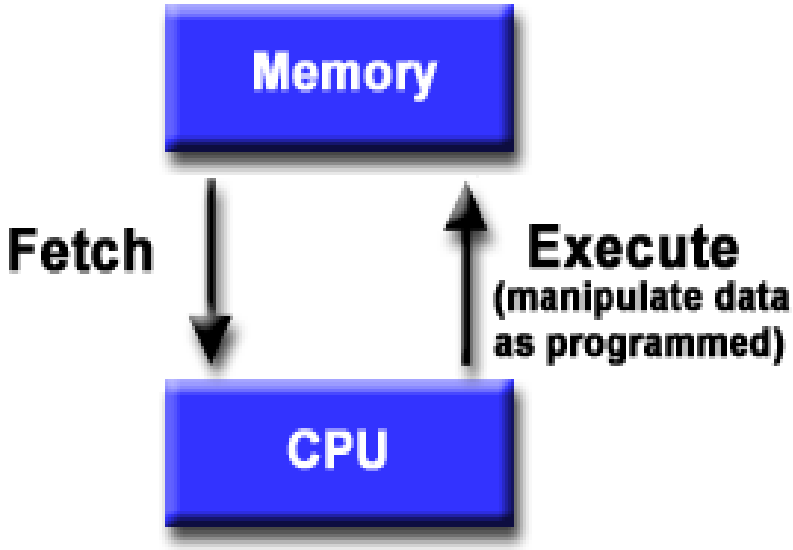
Concepts and Terminology

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)

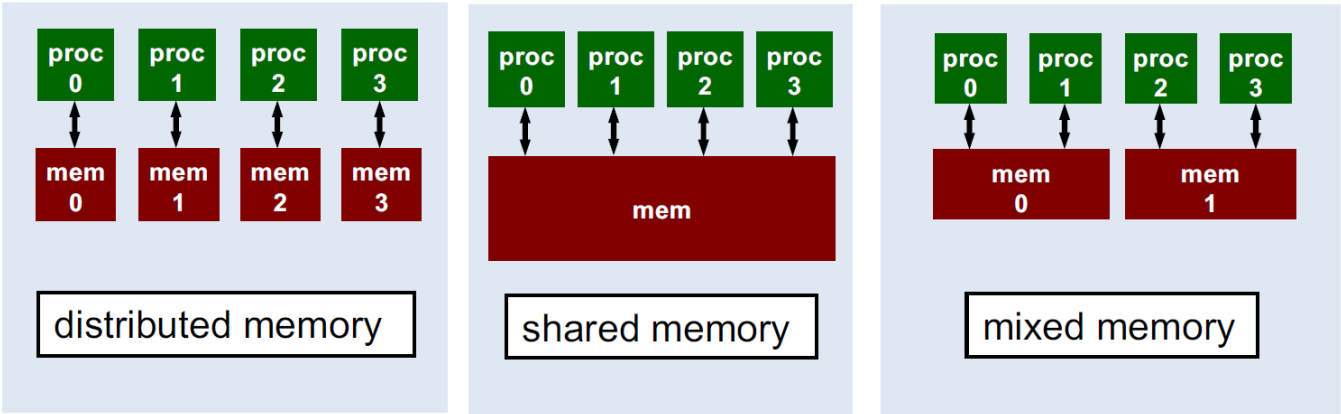
- problem - some calculations depend upon previous calculations
  - can't be performed simultaneously
  - sometimes tied to the physics of the problem, e.g., time evolution of a system
- want to maximize amount of parallel code
  - occasionally easy
  - usually requires some work

Parallelization: memory design

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
- Mixed design

Parallelization: memory design



Parallelization: two approaches

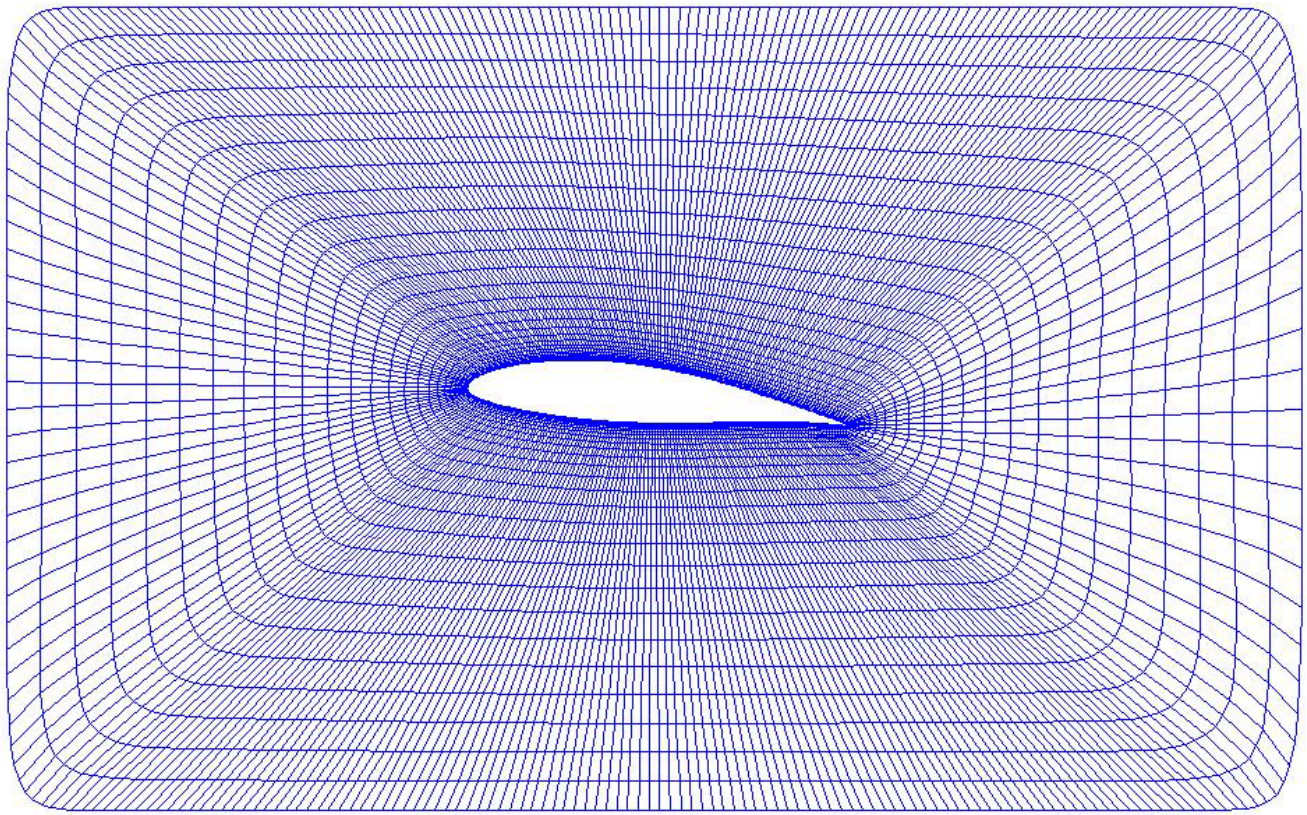
- Message Passing Interface (MPI)
  - for both distributed and shared memory
  - portable
  - freely downloadable
- Open Multi-Processing (OpenMP)
  - shared memory only
  - must be supported by compiler (most do)
  - usually easier than MPI
  - can be implemented incrementally

MPI

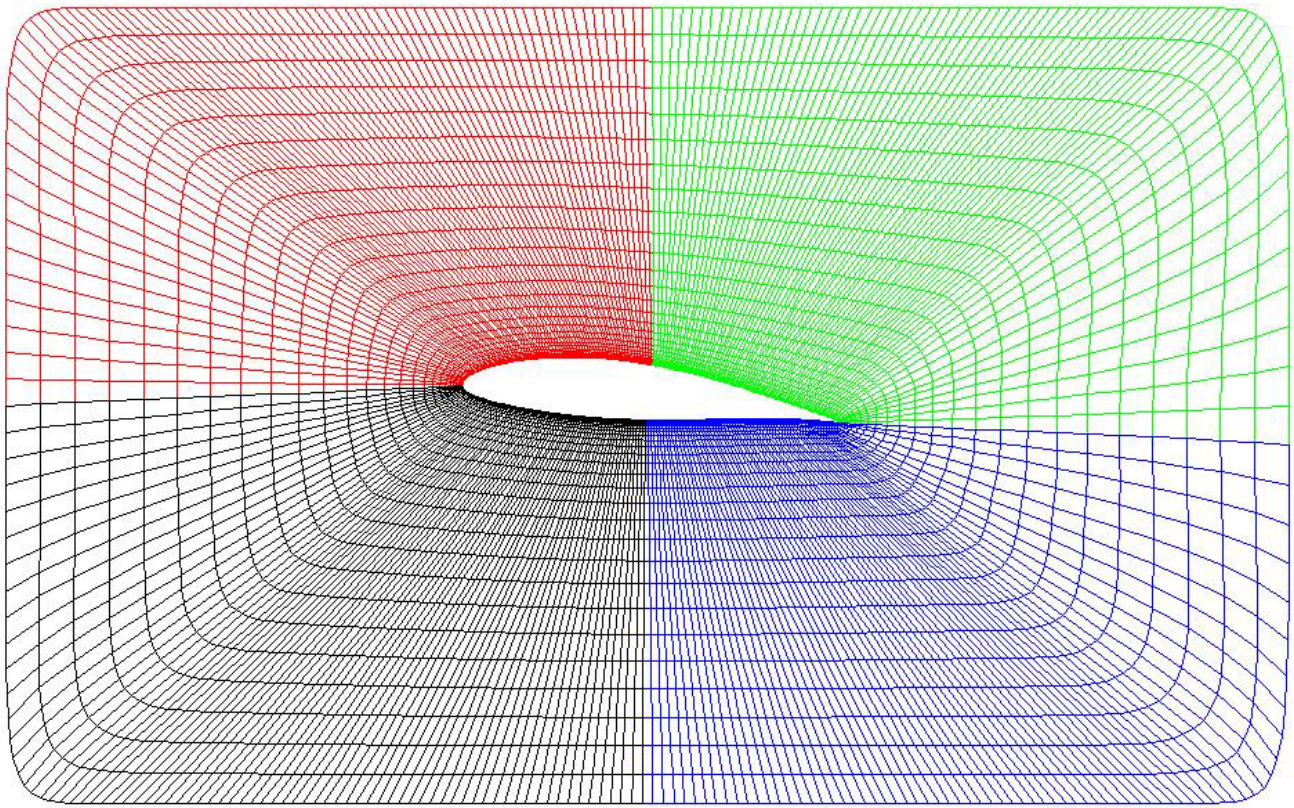
- Computational domain is typically decomposed into regions
  - One region assigned to each processor
- Separate copy of program runs on each processor

MPI

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



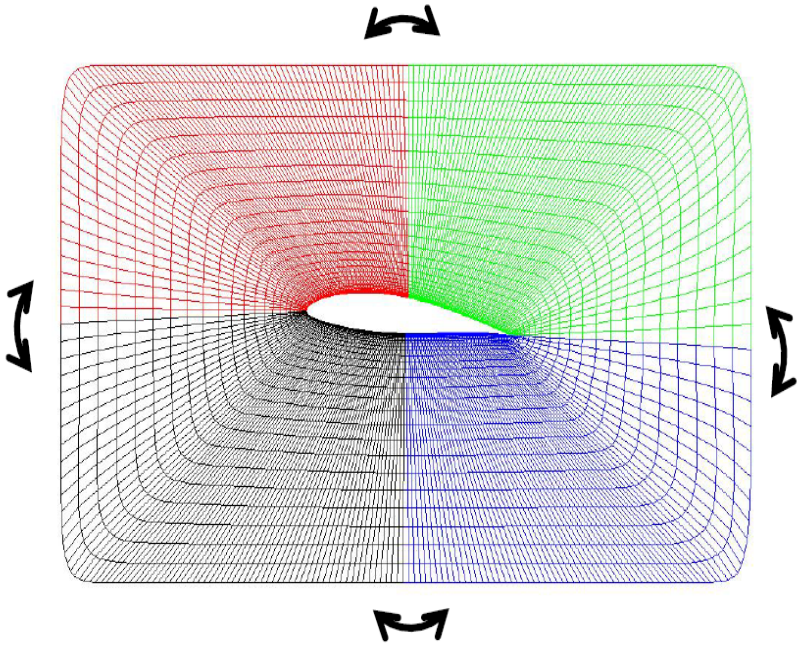
- Decomposed domain for 4 processors



MPI

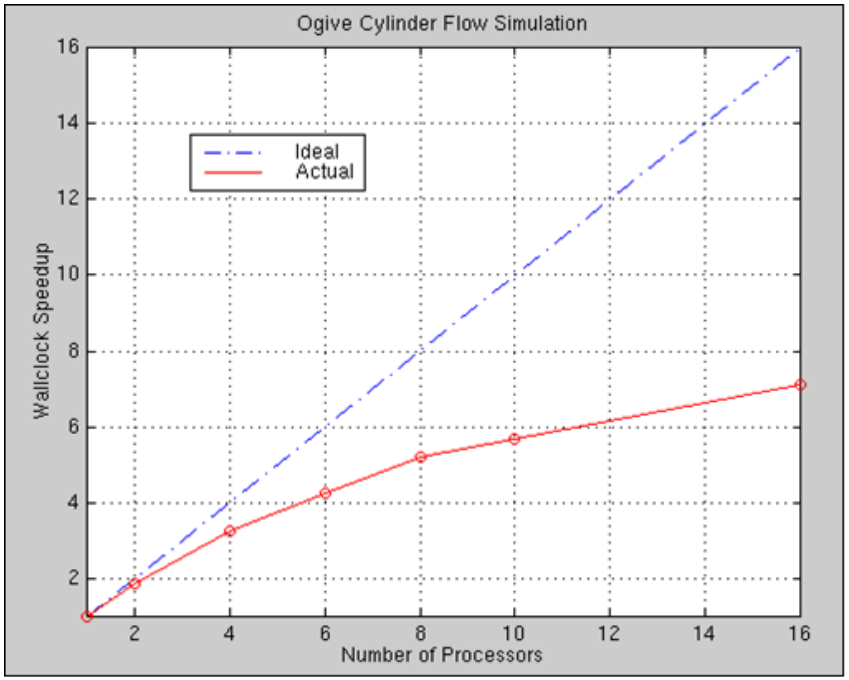
- Since points depend on adjacent points, must transfer information after each iteration
- This is done with explicit calls in the source code

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

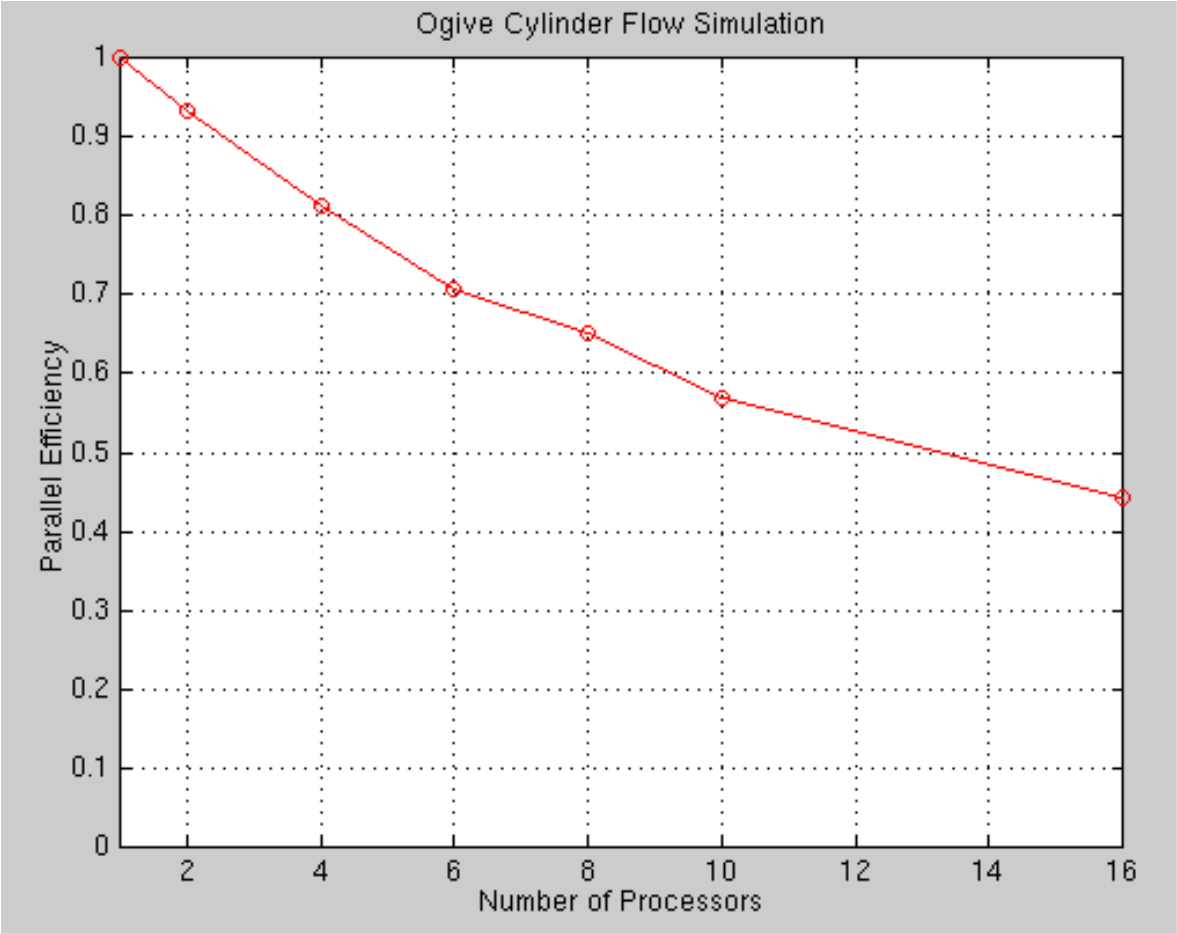


MPI

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



Parallel Efficiency



Skoltech
Skolkovo Institute of Science and Technology

OpenMP

- 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!