Parallel Computing

Christoph Federrath

Material: https://computing.llnl.gov/tutorials/parallel_comp/

Image credit: M. S. Povich

Why parallel computing?

Main reason for *Parallel Computing* is that we can

SOLVE LARGER and MORE COMPLEX PROBLEMS

Auto Assembly

Jet Construction

Drive-thru Lunch

Galaxy Formation

Planetary Movments

Climate Change

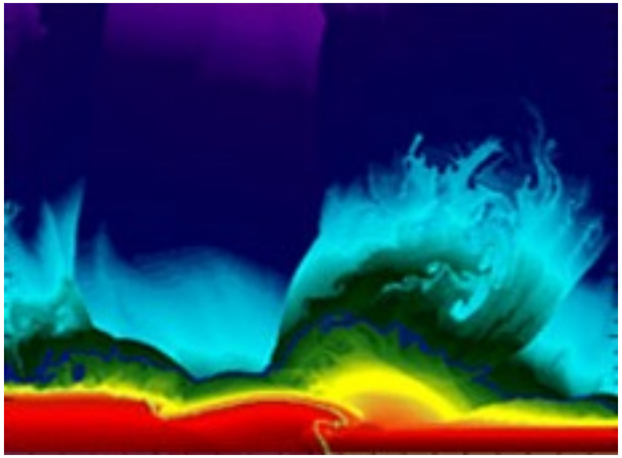
Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena

Parallel computing – applications

Use of parallel computing

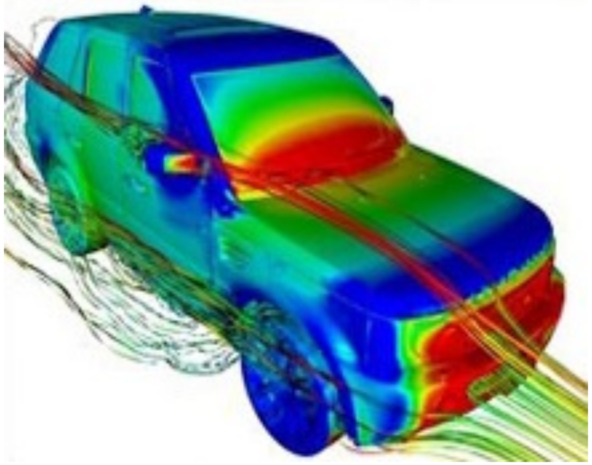
Science and Engineering

- Atmosphere, Earth, Environment
- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Bioscience, Biotechnology, Genetics
- Chemistry, Molecular Sciences
- Geology, Seismology
- Mechanical Engineering - from prosthetics to spacecraft
- Electrical Engineering, Circuit Design, Microelectronics
- Computer Science, Mathematics
- Defense, Weapons



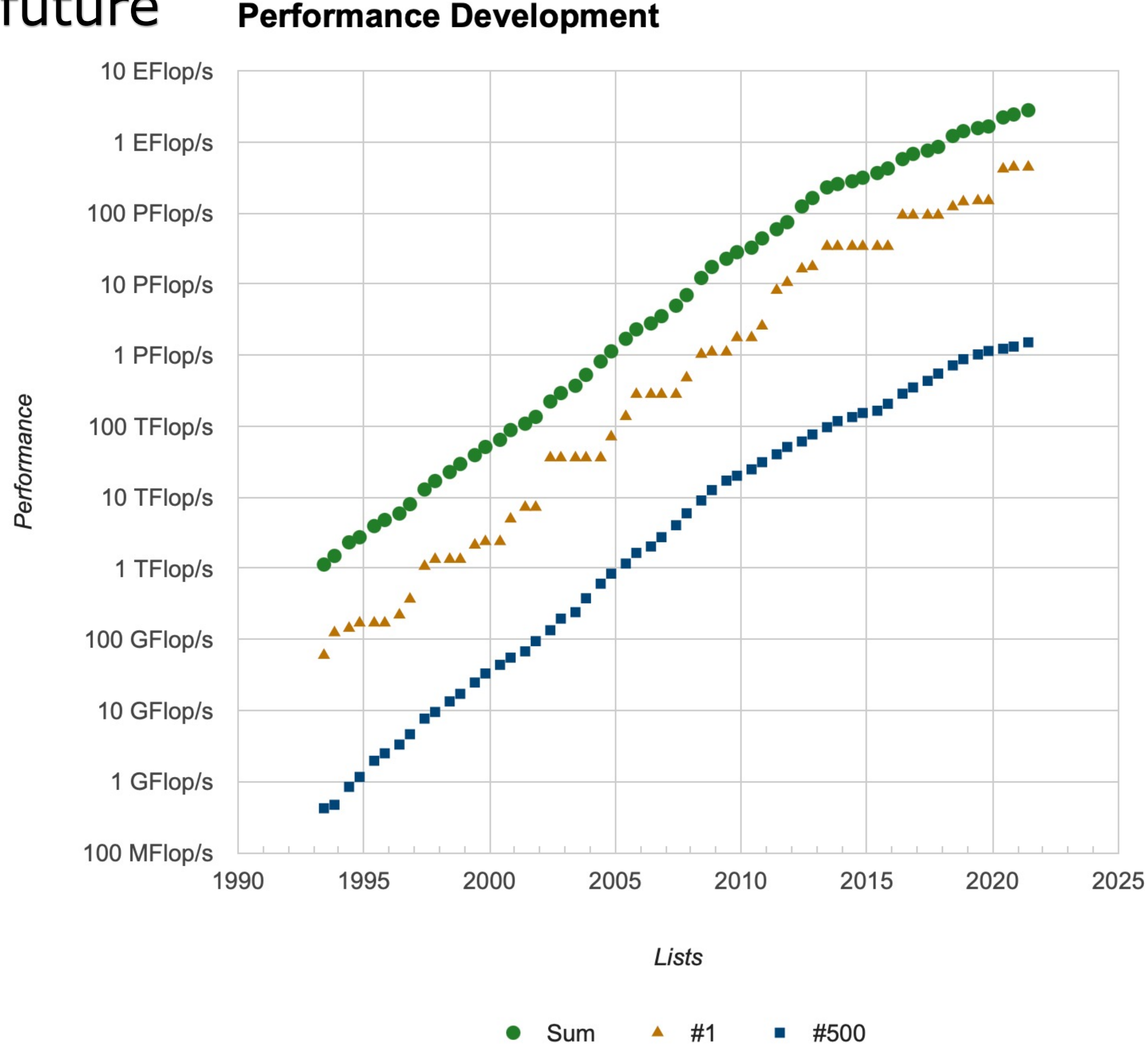
Industrial and Commercial

- "Big Data", databases, data mining
- Web search engines, web based business services
- Medical imaging and diagnosis
- Advanced graphics and virtual reality
- Networked video and multi-media technologies
- Collaborative work environments



Parallel computing – top computers worldwide

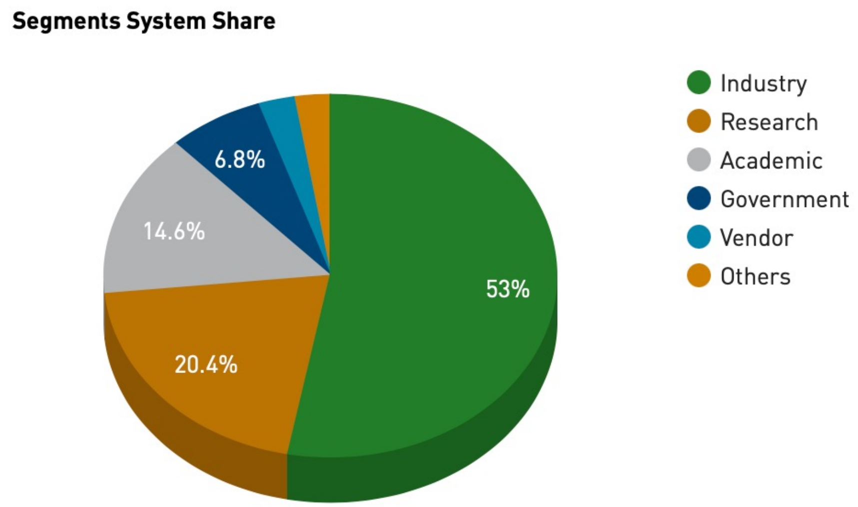
Parallel computing is the future

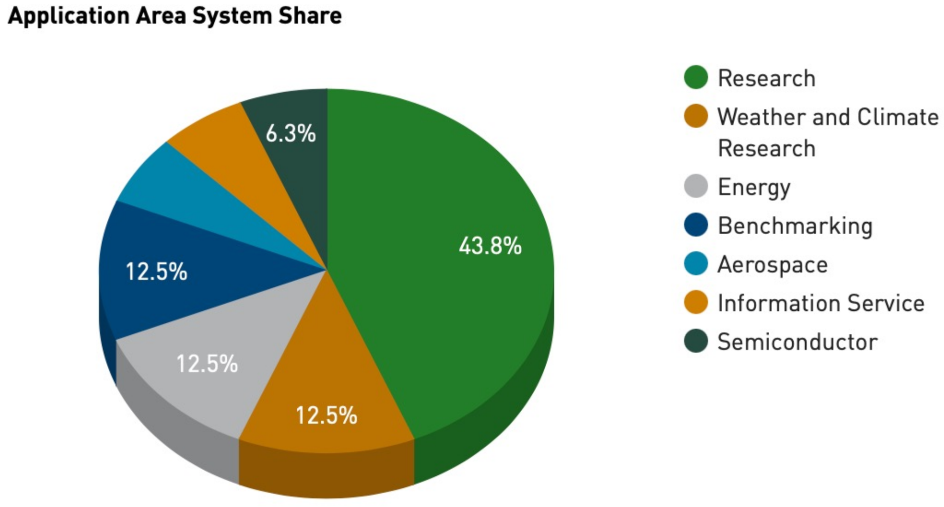


See current Top 500 list:
https://www.top500.org/lists/top500/list/2021/06/?page=1

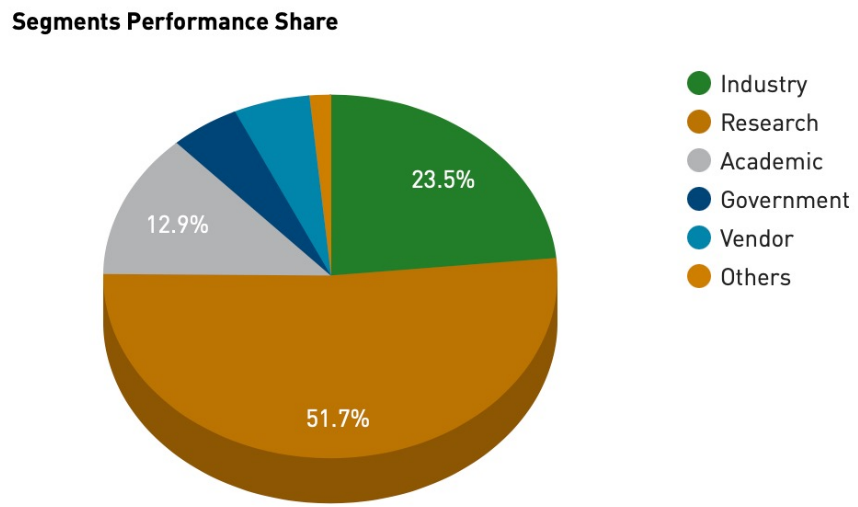
Top Australian Supercomputer:
Gadi (#44 in the world)

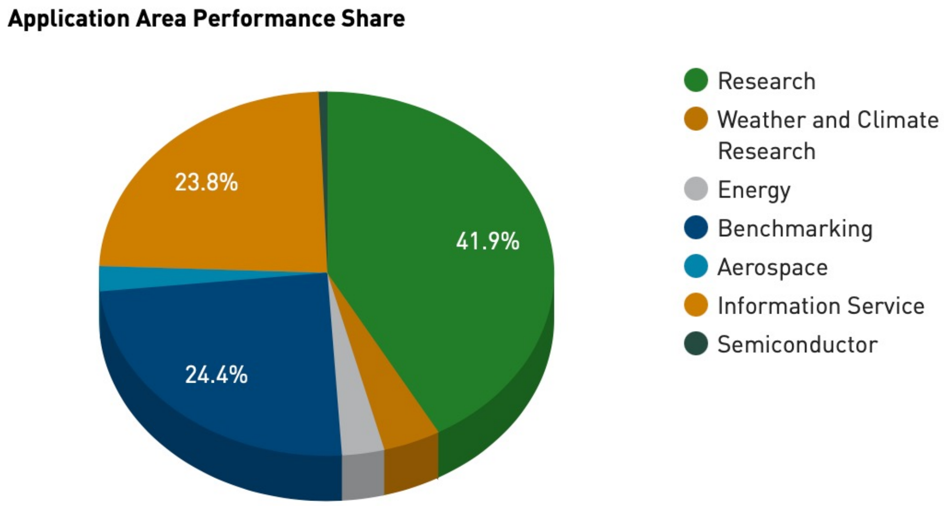
Parallel computing – application areas





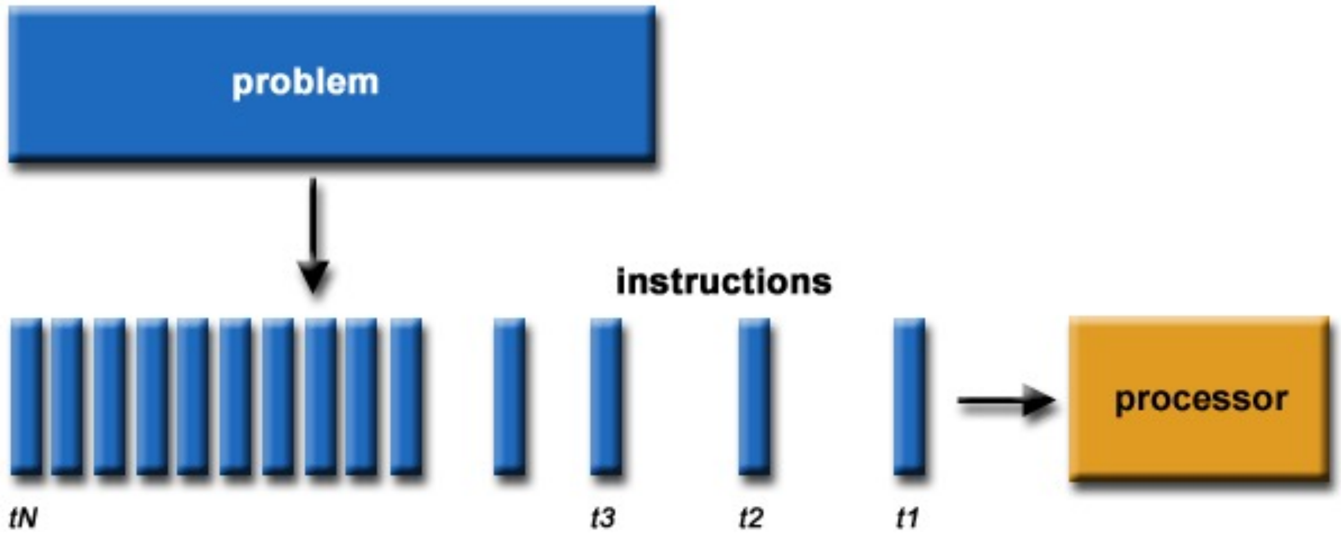
Source: https://www.top500.org/





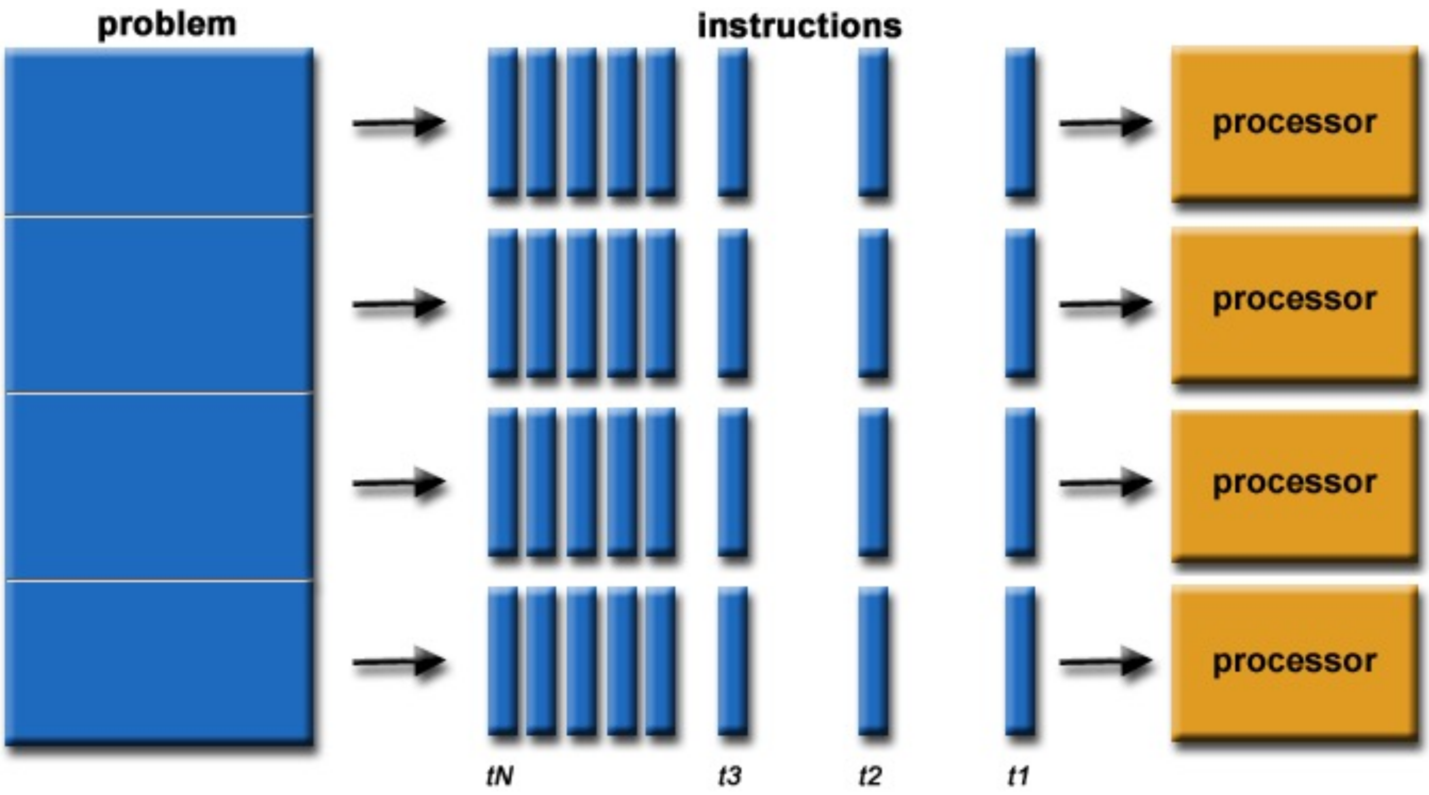
Parallel computing – basic concepts

Solving a problem in serial (single processor)



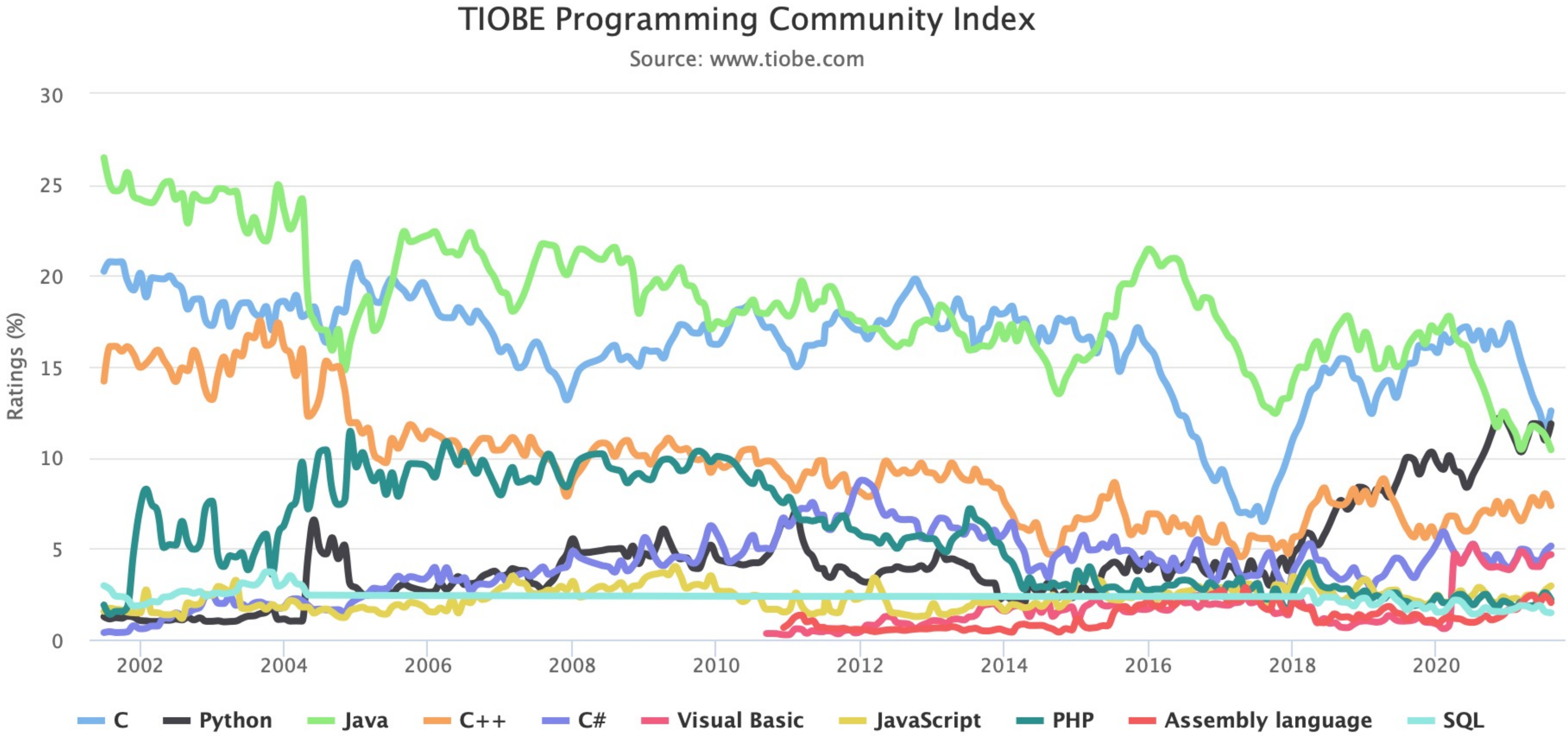
Parallel computing – basic concepts

Parallel version for solving the same problem



Parallel computing – basic concepts

Before diving into the details of parallelization, let's have a look at the performance of Python versus C/C++ programs.



Before diving into the details of parallelization, let's have a look at the performance of Python versus C/C++ programs.

Example: **summation of numbers**

- Write a small python program that sums up all integers from 1 to n and writes the sum to stdout.
- Use the argparse package to take an optional argument '-n' to read n from the command line (if –n is not specified, let the program use n = 5e6 by default).
- First, use a for-loop to sum up the numbers.
- Time the part of the code that does the summation. This means let the code write how much time (in seconds) it took to execute the summation. Suggest to use the timeit package.
- Now use the numpy function numpy.sum() and time it again.

Parallel computing – basic concepts

Before diving into the details of parallelization, let's have a look at the performance of Python versus C/C++ programs.

Example: **summation of numbers**

- Now let's write a small C program that sums up the numbers.

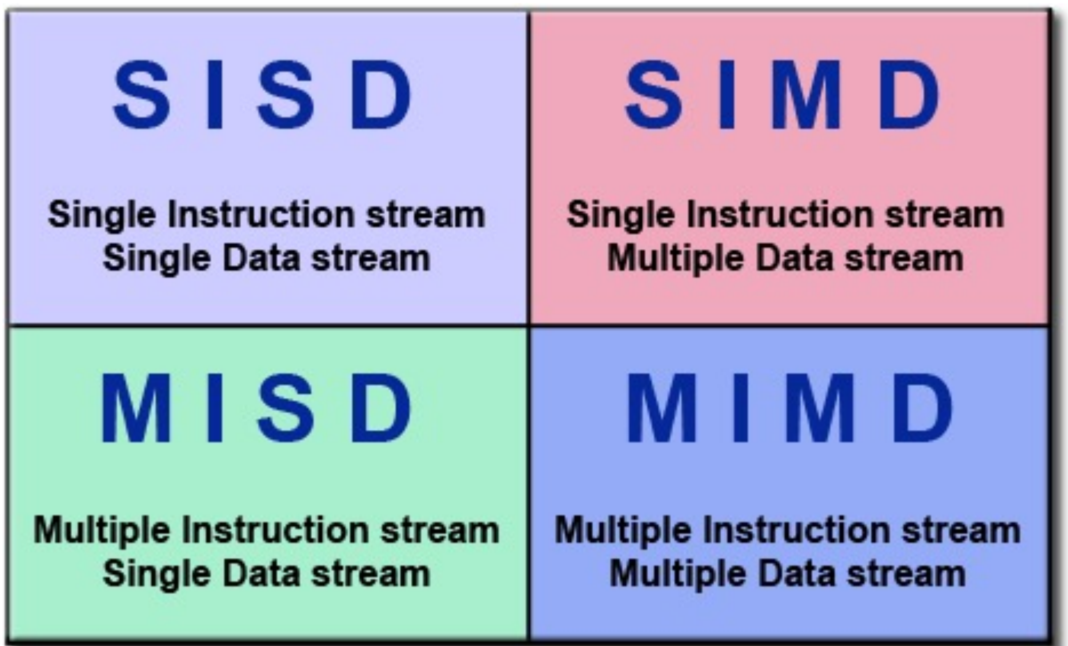
- …

- We can use a python wrapper program to do the timing of the C code (beware of overheads) or time it directly in the C code.

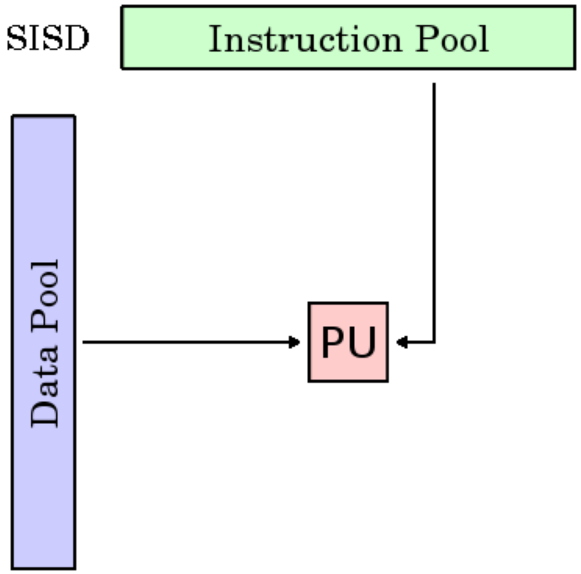
- Play with compile optimization options such as –O3.

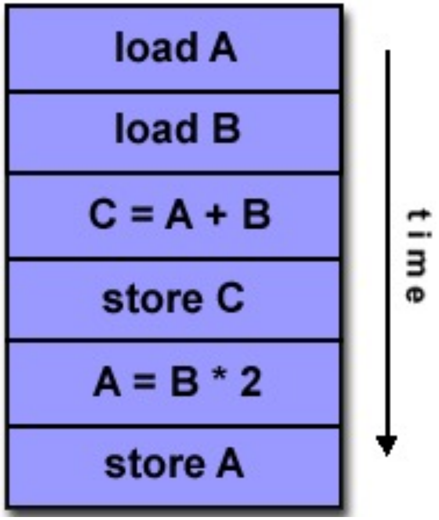
Parallel computing – basic concepts

4 main computer/architecture/operating classifications

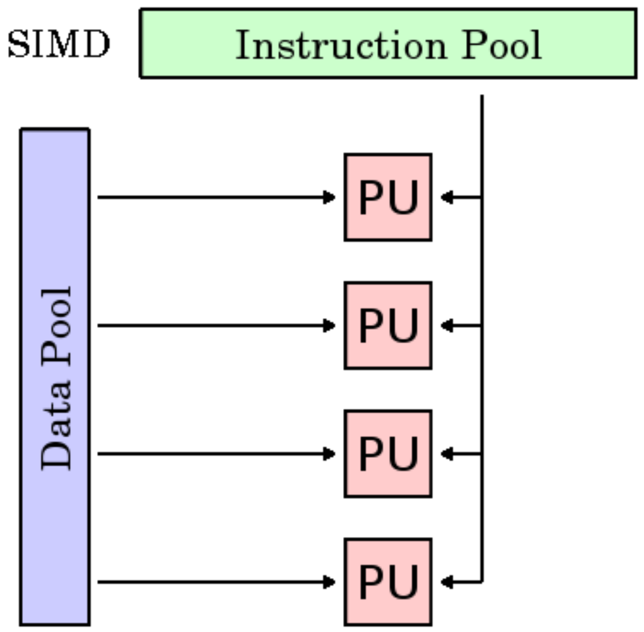


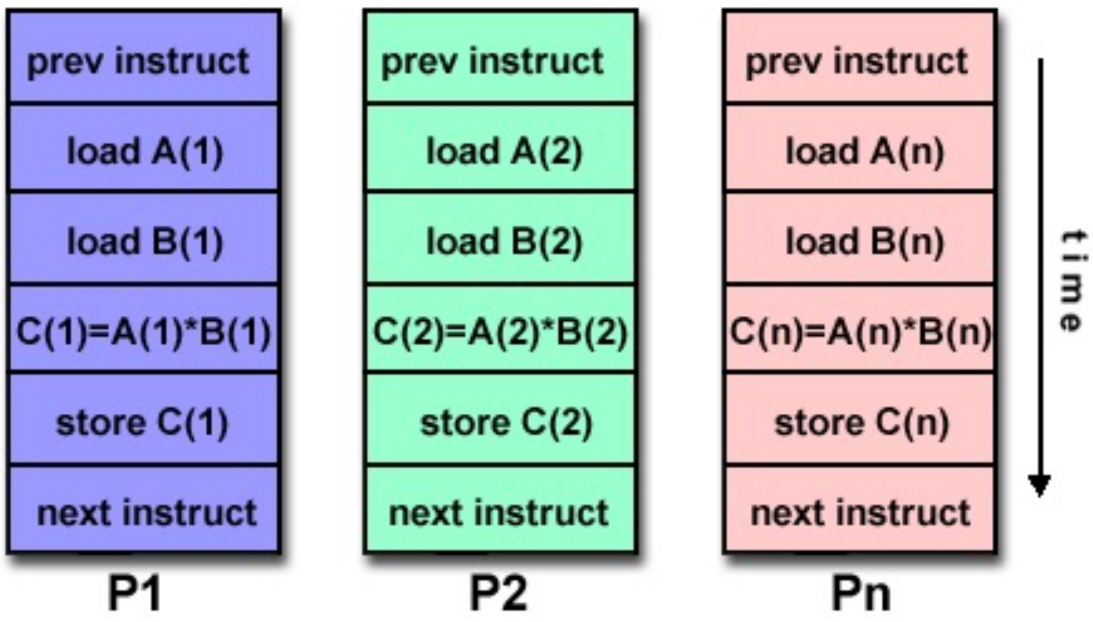
Single Instruction – Single Data (SISD)



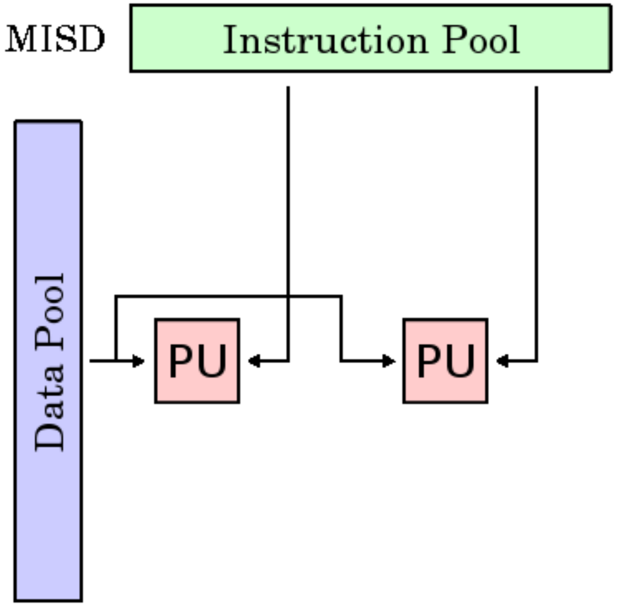


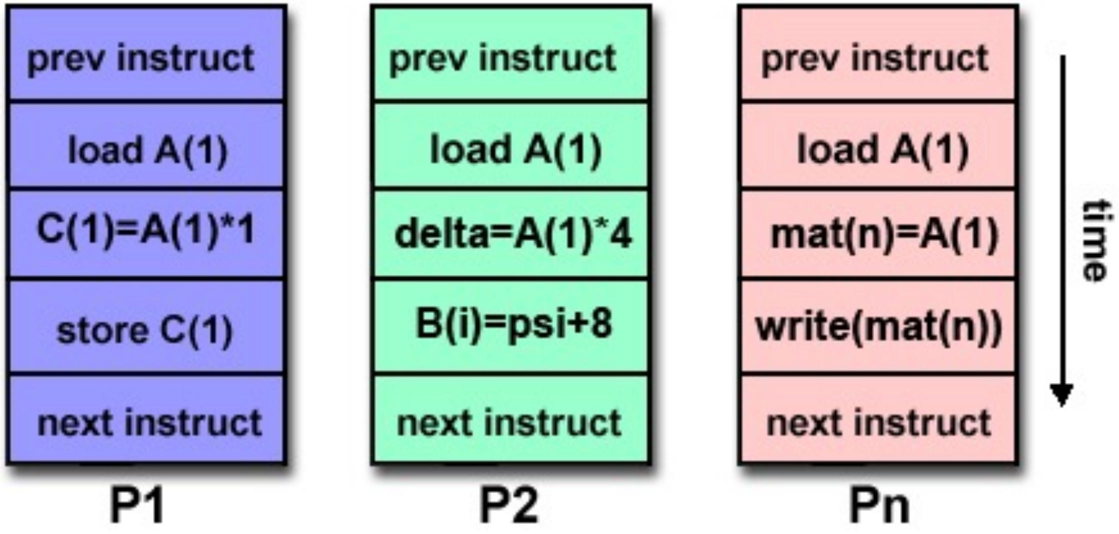
Single Instruction – Multiple Data (SIMD)





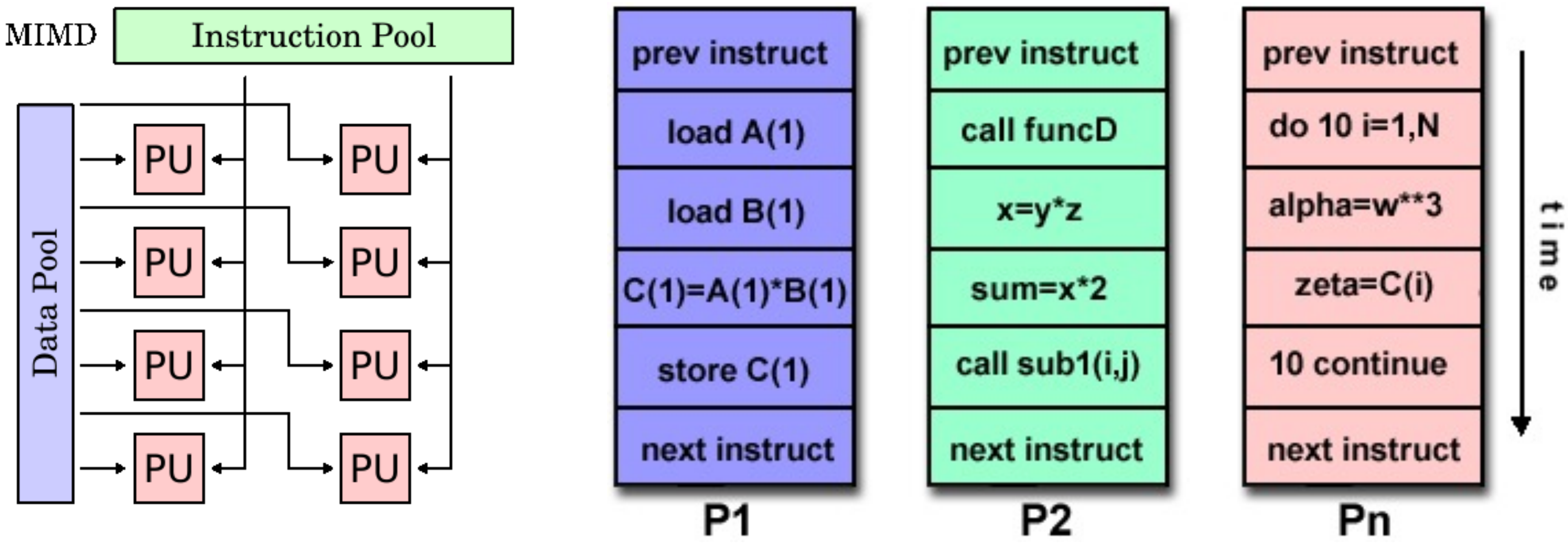
Multiple Instruction – Single Data (MISD)





Parallel computing – basic concepts

Multiple Instruction – Multiple Data (MIMD)

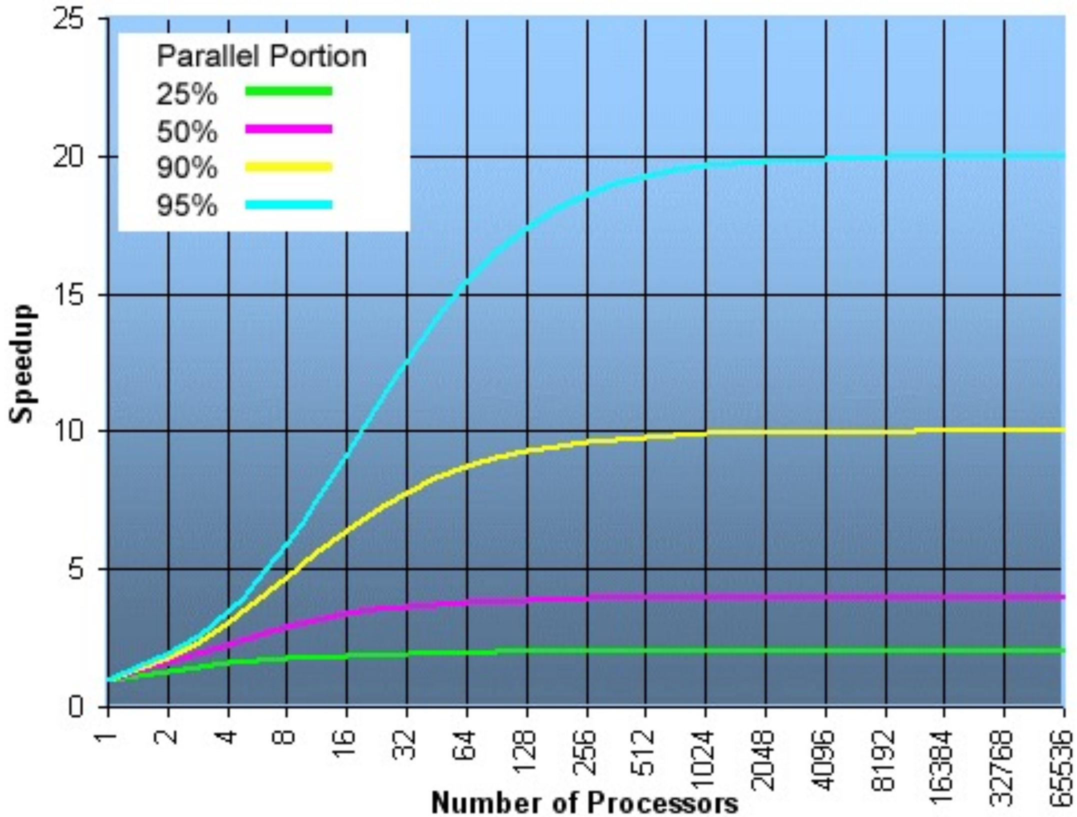


Parallel computing – scaling

Amdahl's Law:

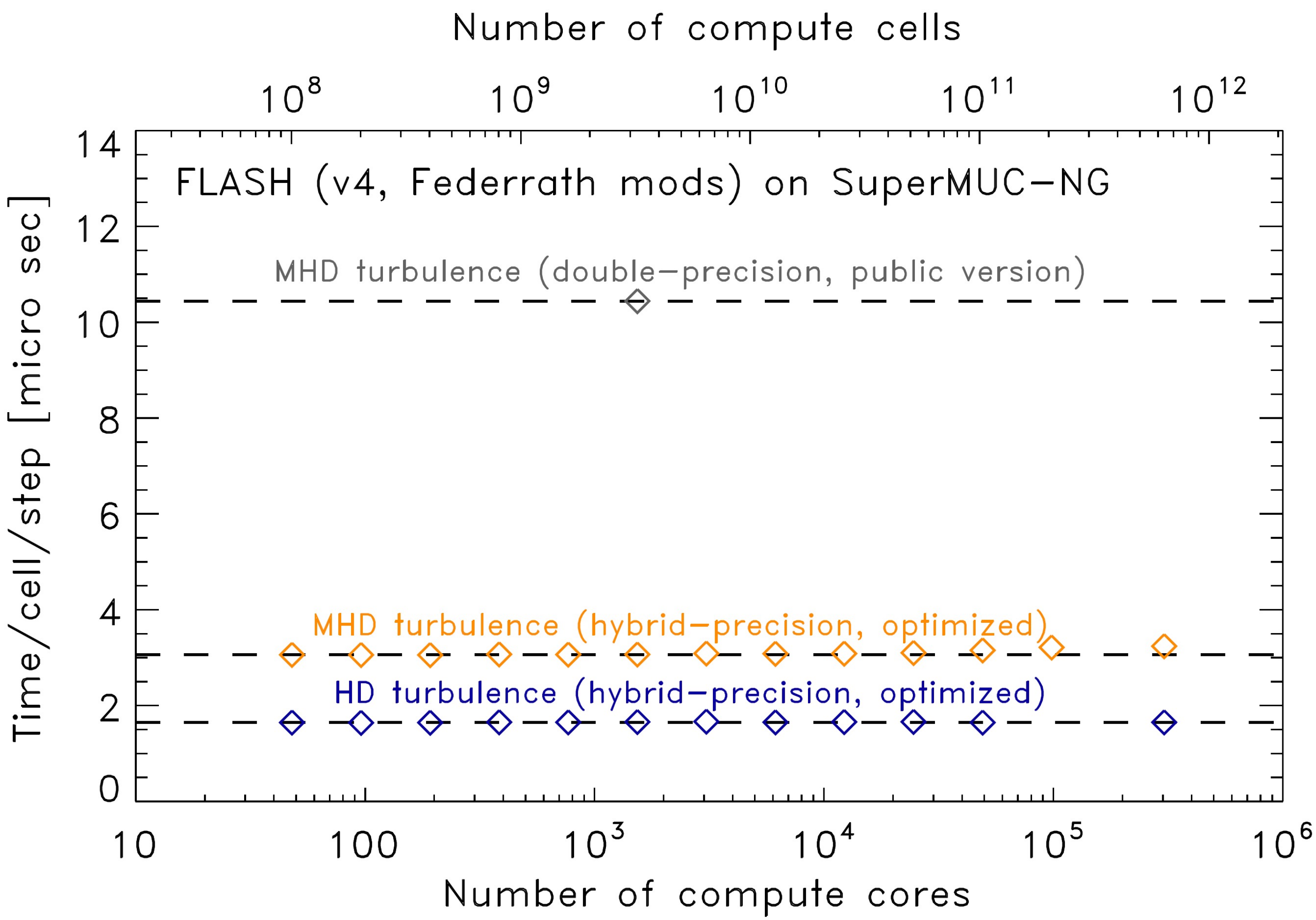
$$\text{speedup} = \frac{1}{P/N + S}$$

P: Parallel fraction of the code
N: Number of processors/cores
S: Serial fraction of the code



However: STRONG SCALING versus WEAK SCALING

FLASH code scaling for HD and MHD turbulence

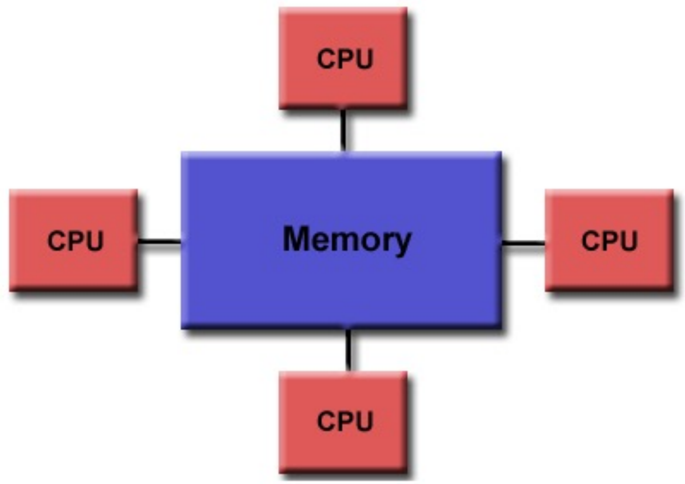


However: STRONG SCALING versus WEAK SCALING

Parallel computing – memory architectures

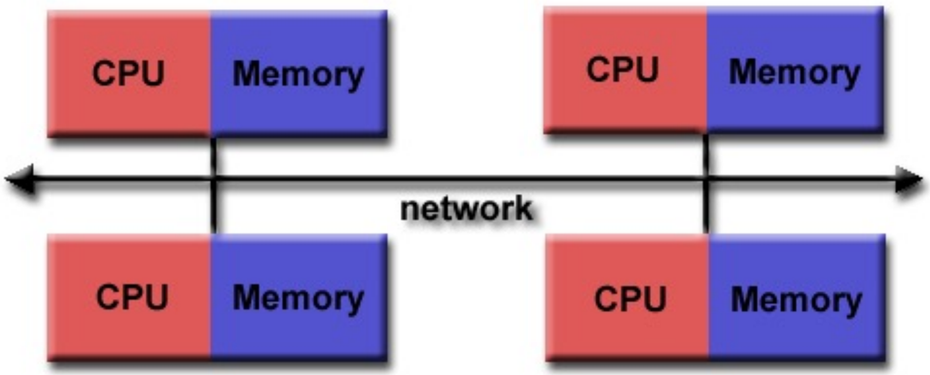
The two main parallel memory architectures

Shared memory
(e.g., OpenMP)



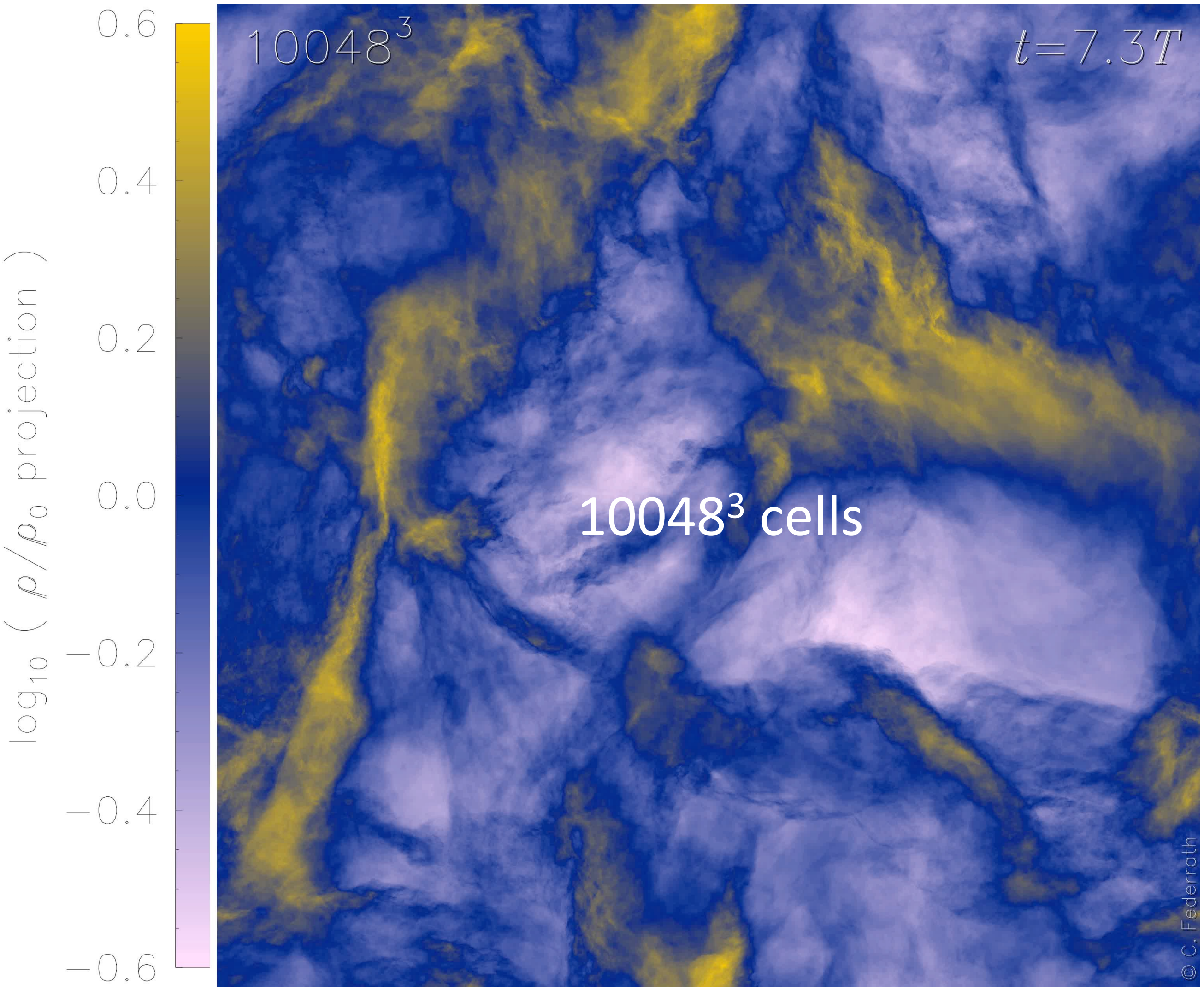
- User-friendly programming perspective to memory
- Lack of scalability between memory and CPUs
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory

Distributed memory
(e.g., MPI)



- Number of processors and size of memory increase proportionately
- Each processor can rapidly access its own memory without interference
- Cost effectiveness: can use commodity, off-the-shelf processors (and networking)
- Programmer responsible for data communication between processors
- Non-uniform memory access times

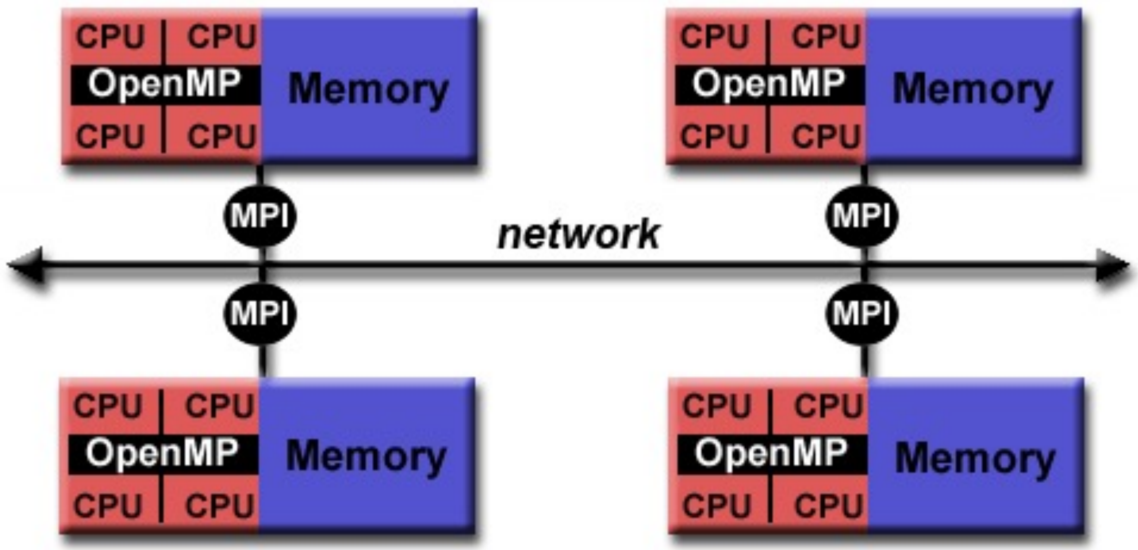
Parallel computing – domain decomposition



Estimate the amount of memory and number of CPUs required

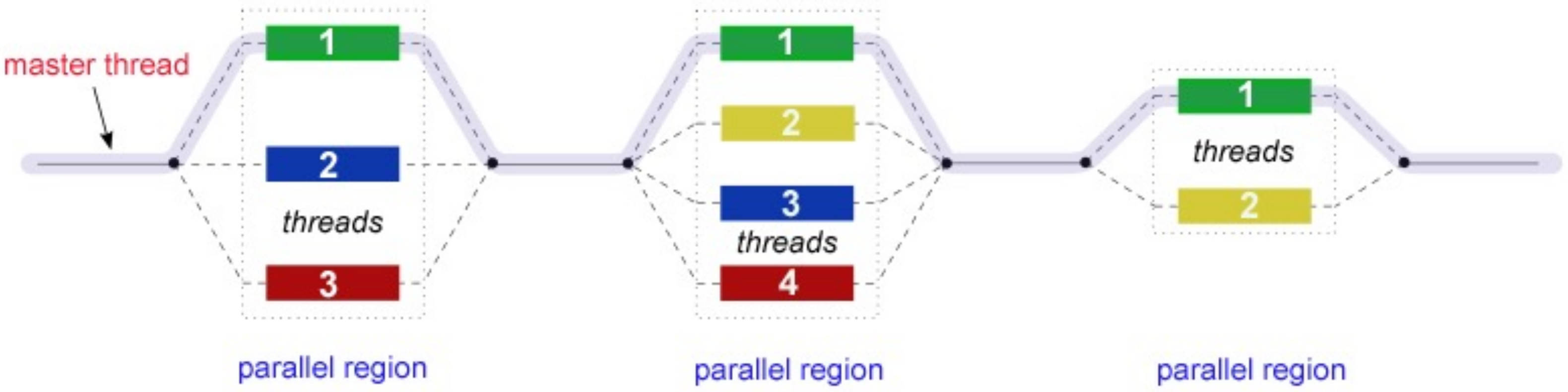
Parallel computing – memory architectures

Hybrid schemes (MPI+OpenMP)



Parallel computing – OpenMP example

OpenMP parallelization (shared-memory + threads)



Fork - Join Model

Now basic parallel coding example with OpenMP…

Automatic vs. Manual Parallelization

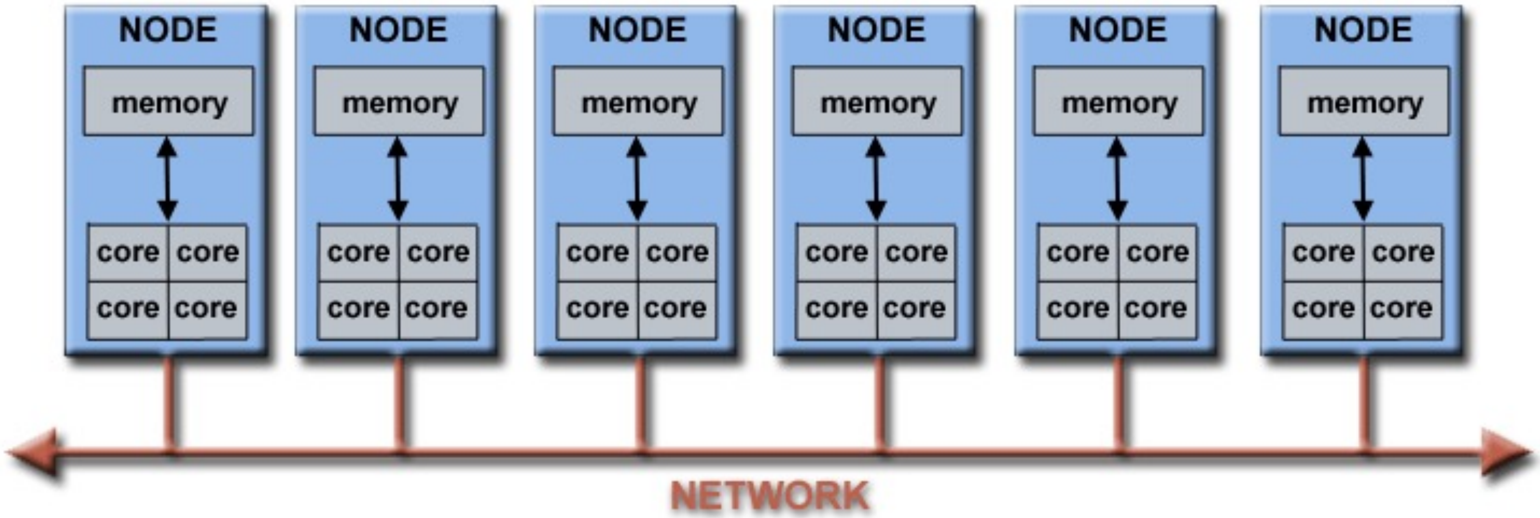
If you are beginning with an existing serial code and have time or budget constraints, then automatic parallelization may be the answer (e.g., OpenMP).

However, there are several important caveats that apply to automatic parallelization:

- Wrong results may be produced
- Performance may actually degrade
- Much less flexible than manual parallelization
- Limited to a subset (mostly loops) of code
- May actually not parallelize code if the compiler analysis suggests there are inhibitors or the code is too complex

Parallel computing – MPI

How to parallelize beyond a single node or single computer?



Message Passing Interface (MPI)

(distributed-memory parallelization)

Message Passing Interface (MPI)

All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

Reasons for using MPI

Standardization - MPI is the only message passing library that can be considered a standard. It is supported on virtually all HPC platforms.

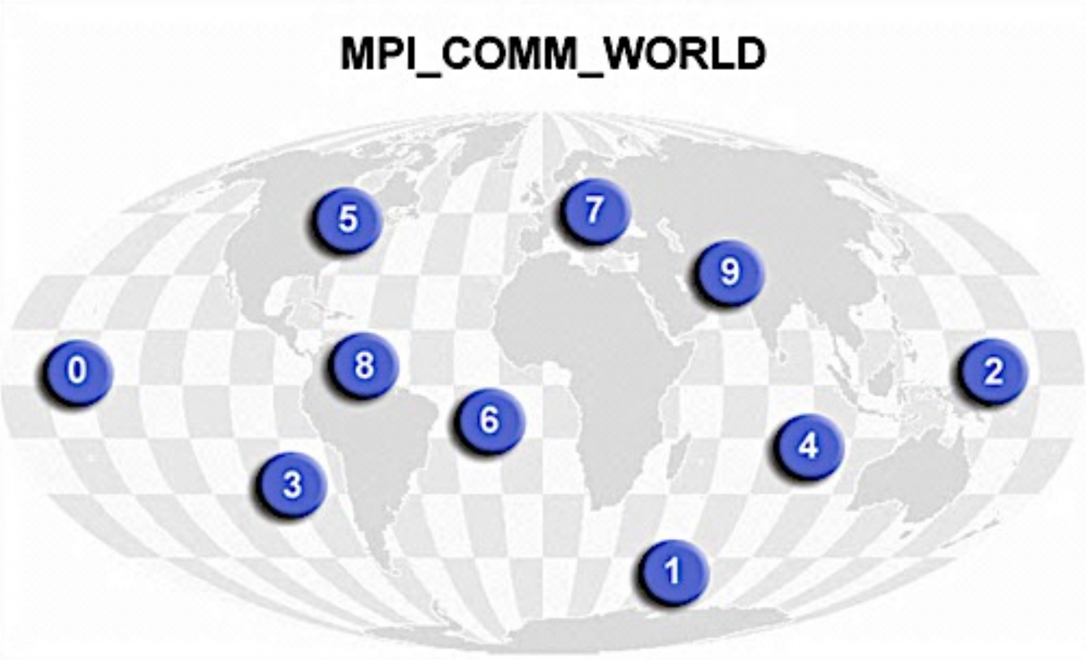
Portability - There is little or no need to modify your source code when you port your application to a different computer.

Performance!!!

E.g., on Mac OS you can install MPI via macports: `sudo port install mpich`

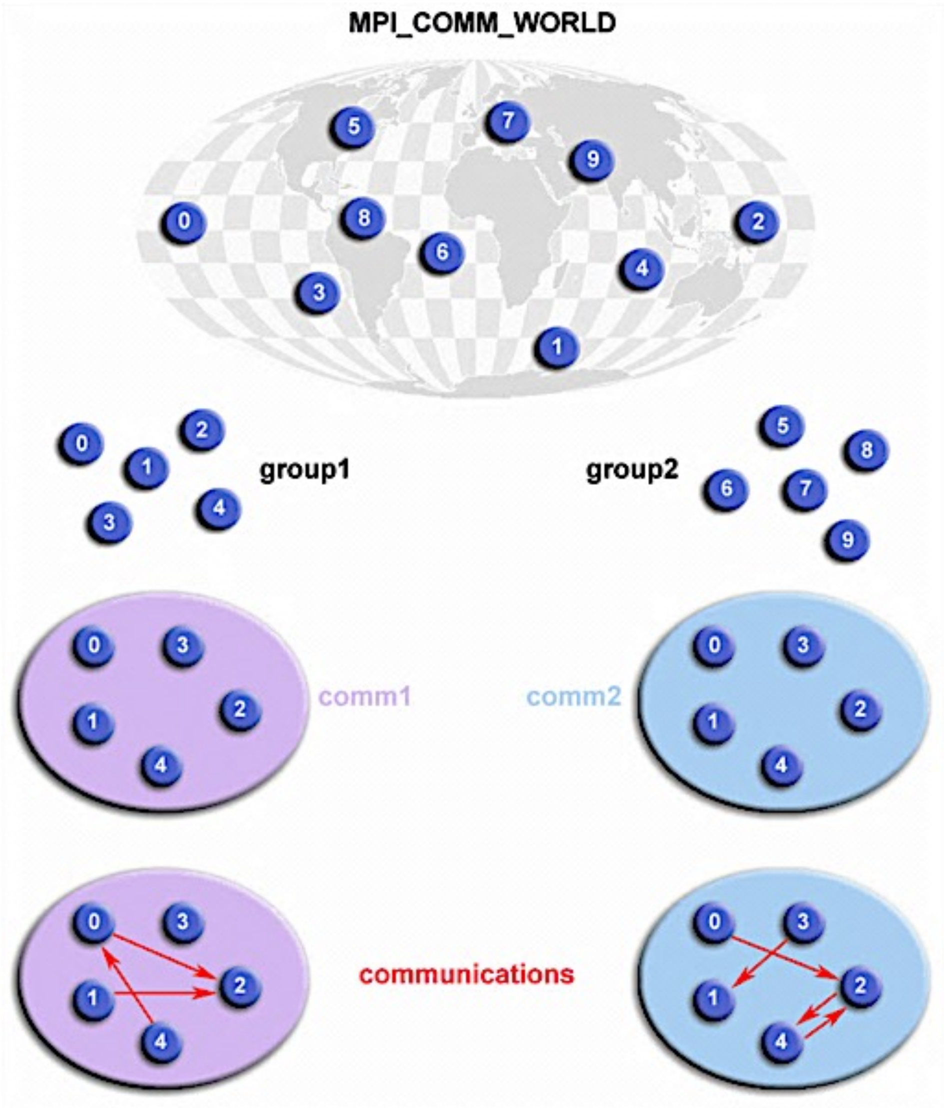
Message Passing Interface (MPI)

MPI uses objects called **communicators** and **groups** to define which collection of processes may communicate with each other



MPI processes are called "ranks"

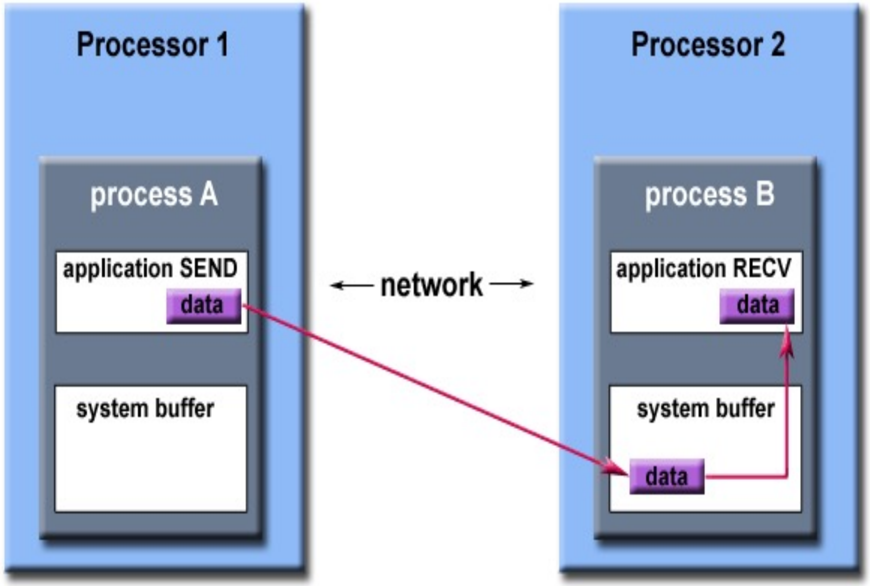
MPI **communicators** and **groups**



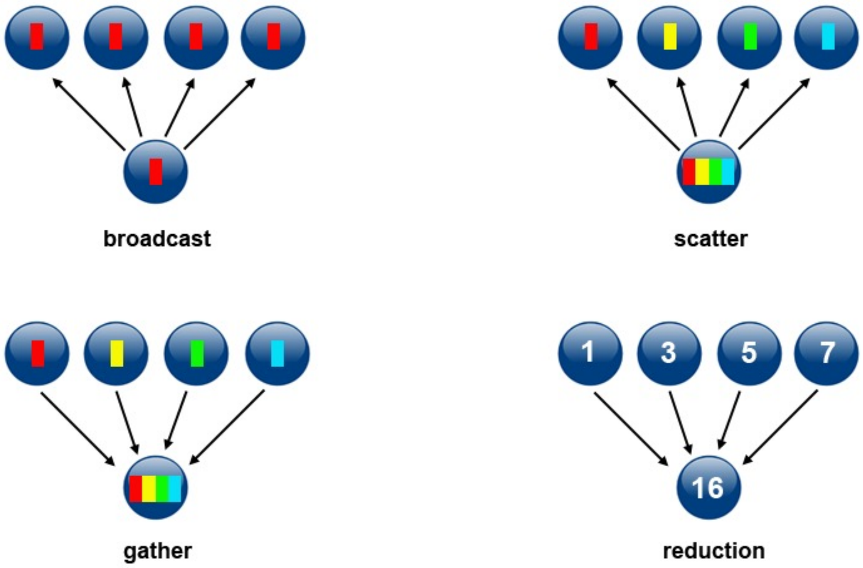
Parallel computing – MPI example

MPI parallelization – 2 main communication types

Point-to-point communication



Collective communication

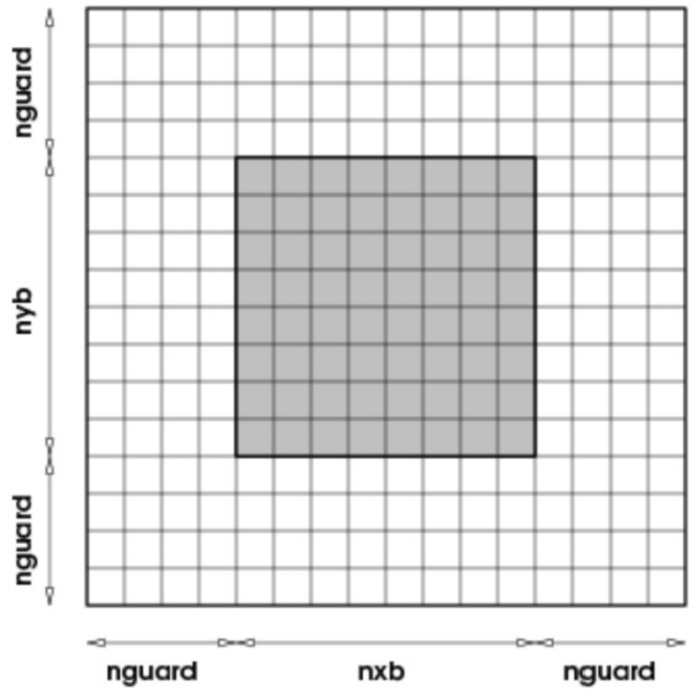


Now MPI example code …

Parallel computing – domain decomposition

MPI parallelization – domain decomposition

For example in the FLASH hydro-dynamical code: "Blocks"

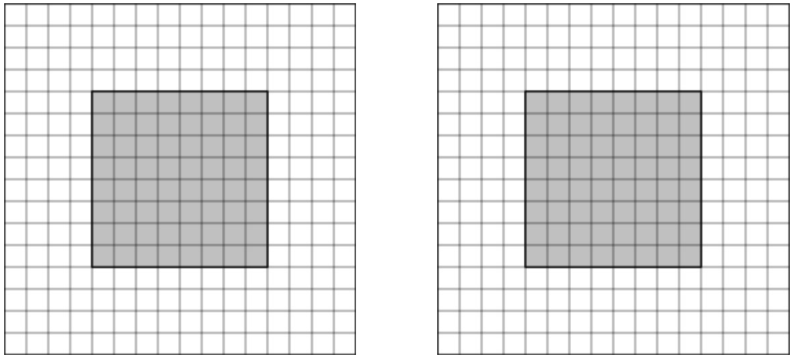


In hydro codes: space versus time decomposition

Parallel computing – domain decomposition

MPI parallelization – domain decomposition

For example in the FLASH hydro-dynamical code: “Blocks”

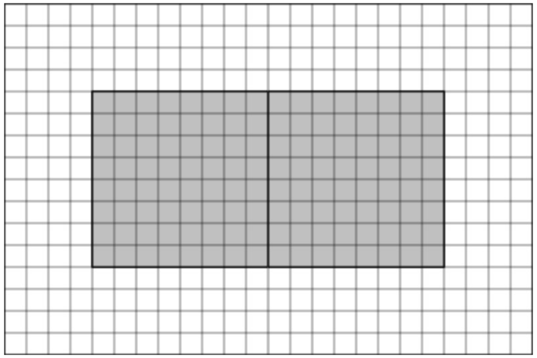


In hydro codes: space versus time decomposition

Parallel computing – domain decomposition

MPI parallelization – domain decomposition

For example in the FLASH hydro-dynamical code: “Blocks”



In hydro codes: space versus time decomposition