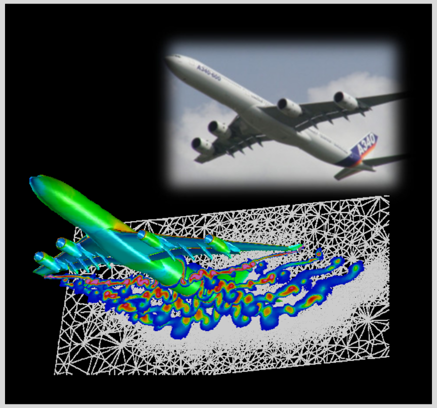


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Prof. Dr. – Ing. Morris Riedel
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland
Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

@Morris Riedel @MorrisRiedel @MorrisRiedel

LECTURE 3

Parallelization Fundamentals

September 12, 2019
Room V02-258

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

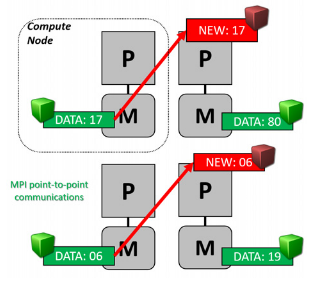
DEEP Projects

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

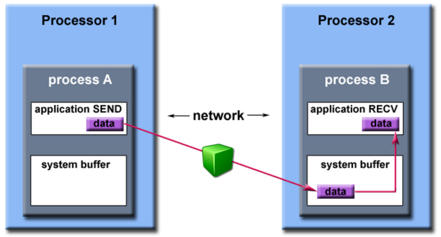
HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

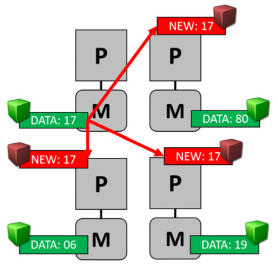
Review of Lecture 2 – Parallel Programming with MPI

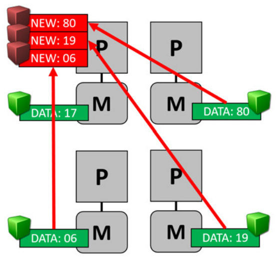
- Message Passing Interface (MPI) Concepts



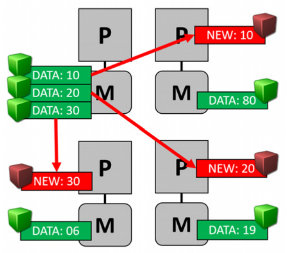
MPI Point to Point Communication

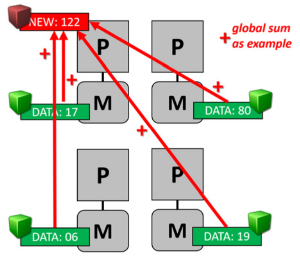




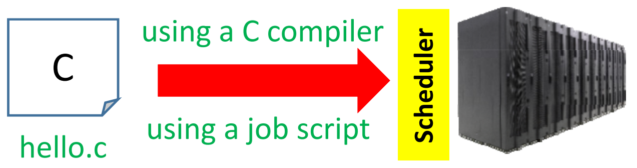


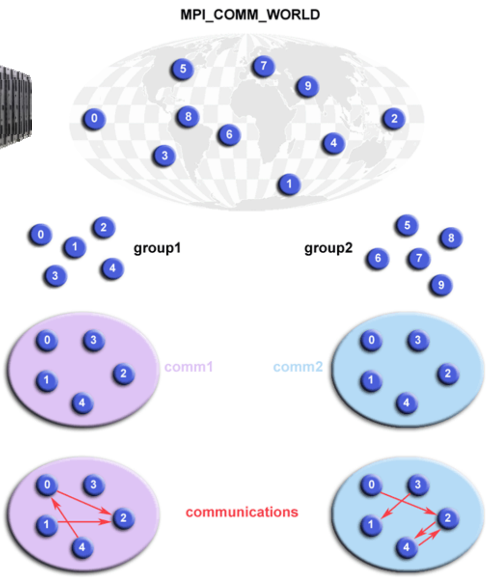
MPI Collective Communication





- MPI Parallel Programming Basics





```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char** argv)
{
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf("Hello World, I am %d out of %d\n",
         rank, size);
  MPI_Finalize();
  return 0;
}
```

[1] LLNL MPI Tutorial

Outline of the Course

+ additional practical lectures & Webinars for our hands-on assignments in context

- 🟩 Practical Topics
- 🟦 Theoretical / Conceptual Topics

Outline

- Common Strategies for Parallelization
  - Simple Parallel Computing Examples Multi-core & Many-core
  - Parallelization Methods & Domain Decomposition
  - Halo / Ghost Layer in Simulation Sciences & Data Sciences
  - Single Program Multiple Data vs. Multiple Program Multiple Data
  - Data Parallelism & Functional Parallelism Methods

- Parallelization Terms & Theory
  - Moore's Law & Parallelization Reasons
  - Speedup & Load Imbalance Terminology
  - Role of Serial Elements
  - Scalability Metrics & Performance
  - Amdahl's Law & Performance Analysis

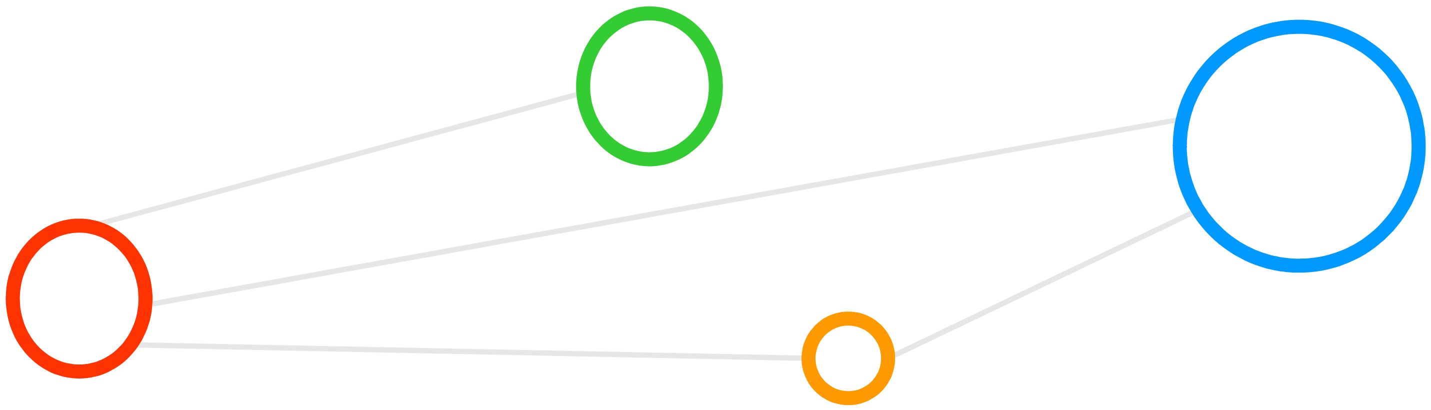
- **Promises from previous lecture(s):**
- ***Lecture 1:*** **Lecture 3 will give in-depth details on parallelization fundamentals & performance term relationships & theoretical considerations**
- ***Lecture 2:*** **Lecture 3 will provide more details on MPI application examples with a particular focus on parallelization fundamentals**

Selected Learning Outcomes

- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming
  - HPC environment tools that support programming or analyze behaviour
  - Different abstractions of parallel computing on various levels
  - Foundations and approaches of scientific domain-specific applications

- Students are able to …
  - Programm and use HPC programming paradigms
  - Take advantage of innovative scientific computing simulations & technology
  - Work with technologies and tools to handle parallelism complexity

Common Strategies for Parallelization



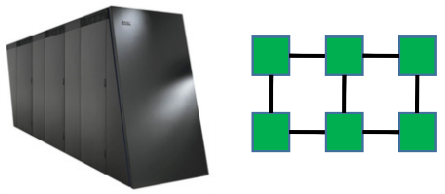
Parallel Computing – Revisited (cf. Lecture 1)

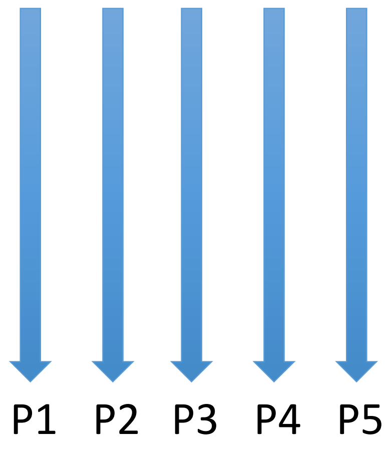
- All modern supercomputers depend heavily on parallelism
  - Parallelism can be achieved with many different approaches

> ▪ **We speak of parallel computing whenever a number of 'compute elements' (e.g. cores) solve a problem in a cooperative way**

[5] Introduction to High Performance Computing for Scientists and Engineers

- Often known as 'parallel processing' of some problem space
  - Tackle problems in parallel to enable the 'best performance' possible
  - Includes not only parallel computing, but also parallel input/output (I/O)
- 'The measure of speed' in High Performance Computing matters
  - Common measure for parallel computers established by TOP500 list
  - Based on benchmark for ranking the best 500 computers worldwide



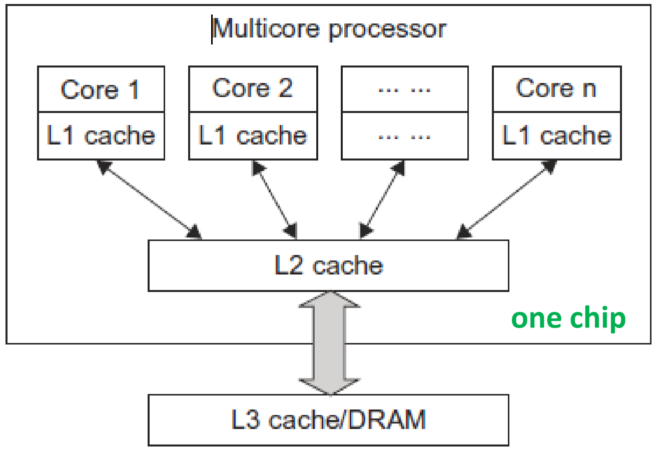


[2] TOP500 Supercomputing Sites

Multi-core CPU Processors – Revisited (cf. Lecture 1)

- Significant advances in CPU (or microprocessor chips)
  - Multi-core architecture with dual, quad, six, or n processing cores
  - Processing cores are all on one chip

- Multi-core CPU chip architecture
  - Hierarchy of caches (on/off chip)
  - L1 cache is private to each core; on-chip
  - L2 cache is shared; on-chip
  - L3 cache or Dynamic random access memory (DRAM); off-chip

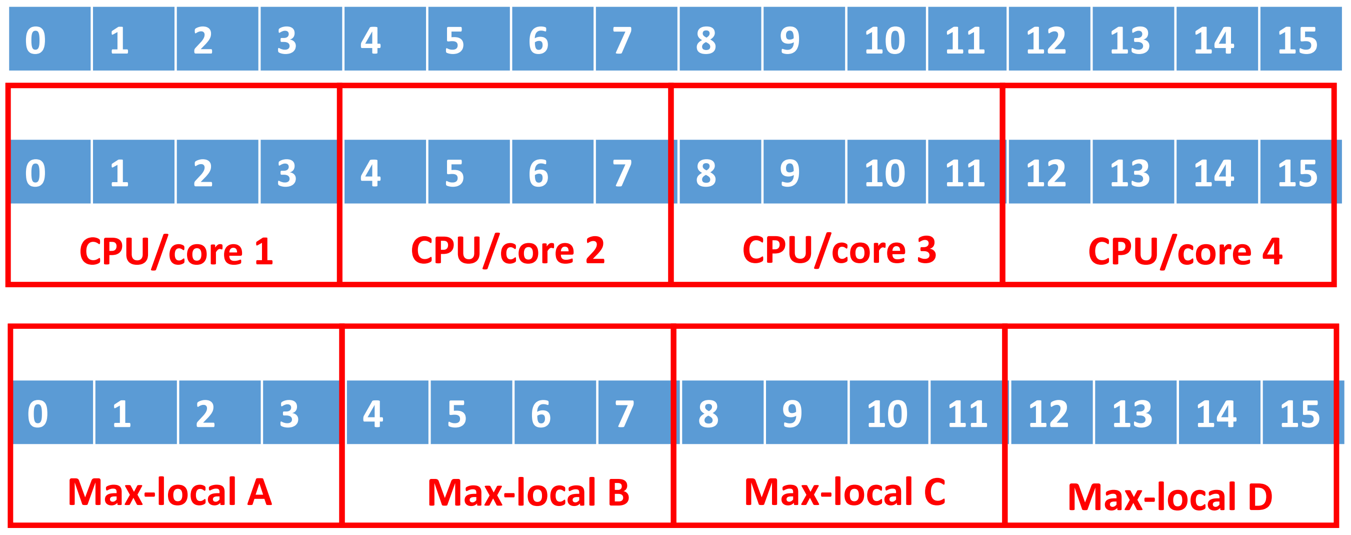


[3] Distributed & Cloud Computing Book

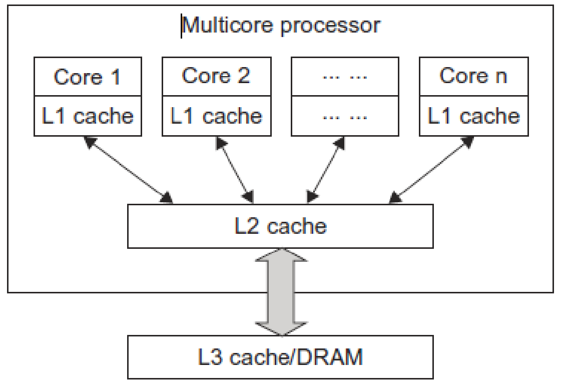
- **Clock-rate for single processors increased from 10 MHz (Intel 286) to 4 GHz (Pentium 4) in 30 years**
- **Clock rate increase with higher 5 GHz unfortunately reached a limit due to power limitations / heat**
- **Multi-core CPU chips have quad, six, or n processing cores on one chip and use cache hierarchies**

Simple Visual Parallel Computing Example on Multi-Core CPUs

- Example: Find largest (maximum) element in array (e.g., 10^15 elements, here only 2^4)
  - Think how the data elements can be divided onto CPUs/cores → ‘data domain decomposition‘
  - Think what each CPUs/cores should do → ‘computational-intensive processes or calculations‘



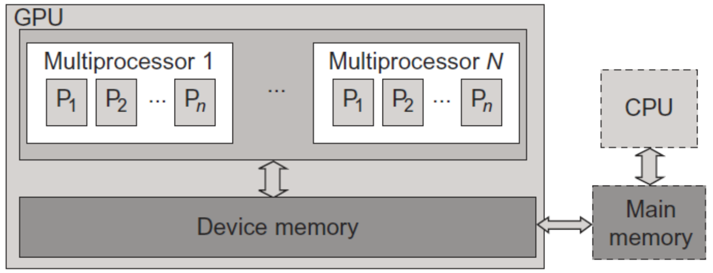
Max-global = Max (Max-local A,B,C,D)



[3] Distributed & Cloud Computing Book

Many-core GPGPUs – Revisited (cf. Lecture 1)

- Use of very many simple cores
  - High throughput computing-oriented architecture
  - Use massive parallelism by executing a lot of concurrent threads slowly
  - Handle an ever increasing amount of multiple instruction threads
  - CPUs instead typically execute a single long thread as fast as possible
- Many-core GPUs are used in large clusters and within massively parallel supercomputers today
  - Named General-Purpose Computing on GPUs (GPGPU)
  - Different programming models emerge



[3] Distributed & Cloud Computing Book

- **Graphics Processing Unit (GPU) is great for data parallelism and task parallelism**
- **Compared to multi-core CPUs, GPUs consist of a many-core architecture with hundreds to even thousands of very simple cores executing threads rather slowly**

Simple Visual Parallel Computing Example on Many-Core GPUs

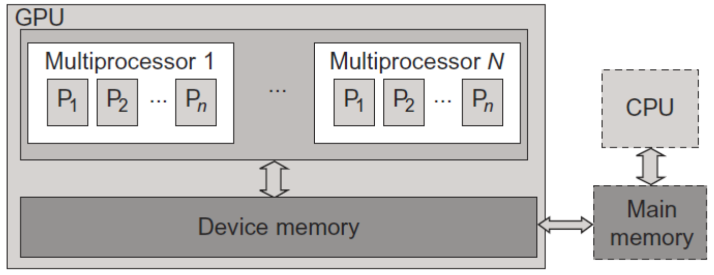
- General Purpose Graphical Processing Unit (GPGPU)
  - Designed to compute large numbers of floating point operations in parallel, but with moderate performance

- **Step 'zero': Data is loaded via the main memory of the CPU (i.e., host CPU memory) to the device memory of the GPU accessed by the many cores**

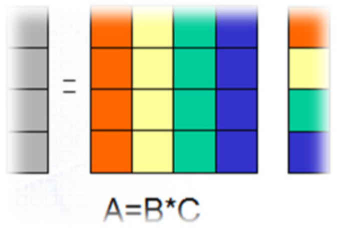
- **Step one: each GPU core has a column of matrix B (named as Bpart)**
- **Step one: each GPU core has an element of column vector C (named Cpart)**

- **Step two: Each GPU core performs an independent vector-scalar multiplication (i.e., independently based on their Bpart and Cpart contents)**

- **Step three: Each GPU core has a part of the result vector A (named Apart) and is written in device memory; results go to the main memory of CPU**



[3] Distributed & Cloud Computing Book



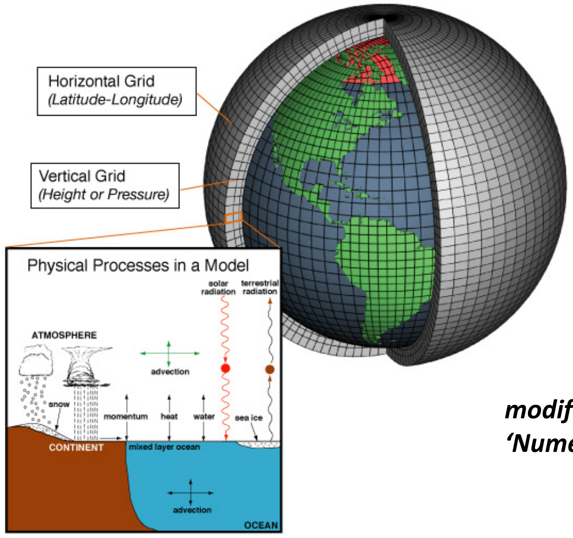
(nice parallelization possible via independent computing)

$$
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} =
\begin{array}{cccc}
b_{0,0}c_0 + & b_{0,1}c_1 + & b_{0,2}c_2 + & b_{0,3}c_3 \\
b_{1,0}c_0 + & b_{1,1}c_1 + & b_{1,2}c_2 + & b_{1,3}c_3 \\
b_{2,0}c_0 + & b_{2,1}c_1 + & b_{2,2}c_2 + & b_{2,3}c_3 \\
b_{3,0}c_0 + & b_{3,1}c_1 + & b_{3,2}c_2 + & b_{3,3}c_3 \\
\text{P0} & \text{P1} & \text{P2} & \text{P3}
\end{array}
$$

Complex Climate Example – Numerical Weather Prediction (NWP) & Forecast

- Application areas
  - Global & regional short-term weather forecast models in operations
  - Perform long-term climate prediction research (e.g. climate change, polar research, etc.)
- NWP model characteristics
  - Use ordinary/partial differential equations (PDEs) (i.e. use laws of physics, fluids, motion, chemistry)
  - Domain decomposition example: 3D grid cells
  - Computing/cell: winds, heat transfer, solar radiation, relative humidity & surface hydrology
  - Interactions with neighboring cells: used to calculate atmosopheric properties over time

- **Numerical Weather Prediction (NWP) uses mathematical models of the atmosphere and oceans to predict the weather based on current weather observations (e.g. weather satellites) as inputs**
- **Performing complex calculations necessary for NWP requires supercomputers (limit ~6 days) using HPC techniques**
- **NWP belongs to the field of numerical methods that obtain approximate solutions to problems → certain uncertainty remains**

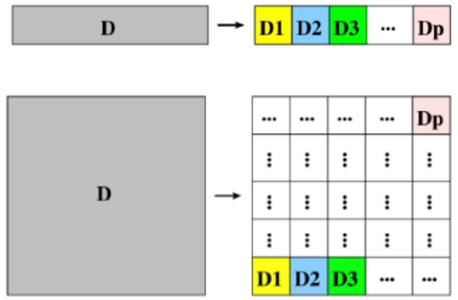


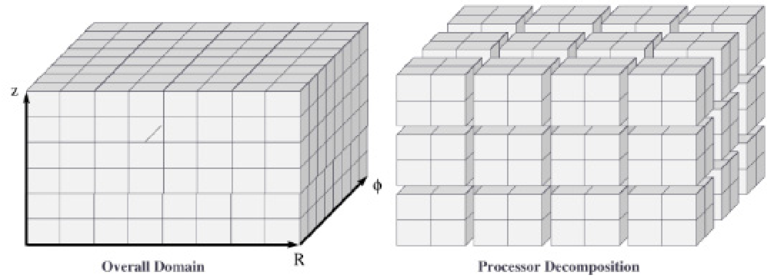
modified from [7] Wikipedia on 'Numerical Weather Prediction'

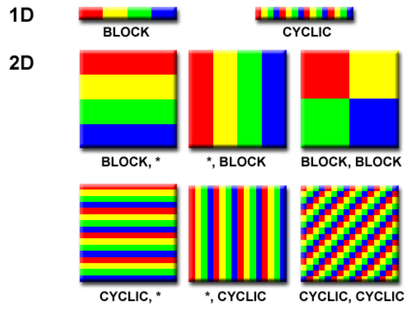
➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial systems and climate simulations on HPC**

Parallelization Methods & Domain Decomposition – Many Approaches

- Data Parallelism





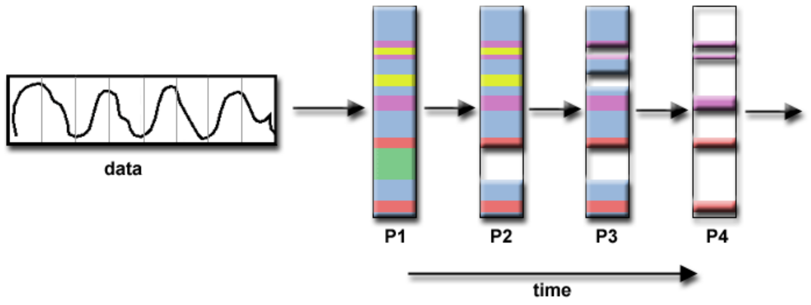


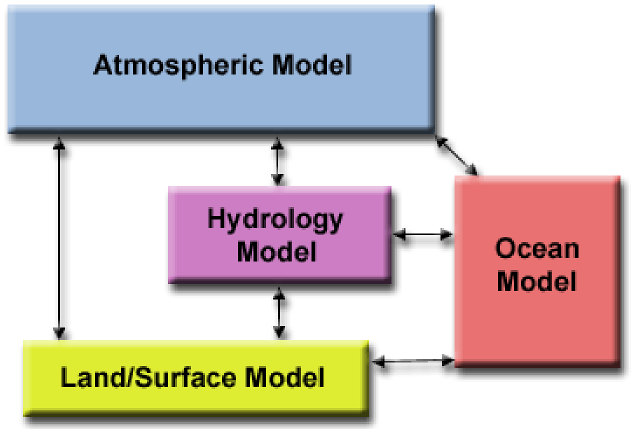
(different forms of domain decomposition methods)

[4] 2013 SMU HPC Summer Workshop

[5] Parallel Computing Tutorial

- Functional Parallelism

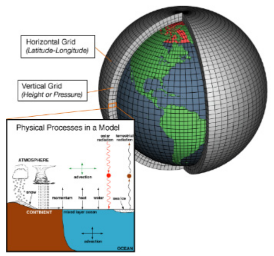




Parallelization Methods in Detail

- Data Parallelism (aka SPMD)
  - N processors/cores work on 'different parts of the data'
  - E.g. Medium-grained loop parellelization
  - E.g. Domain decomposition

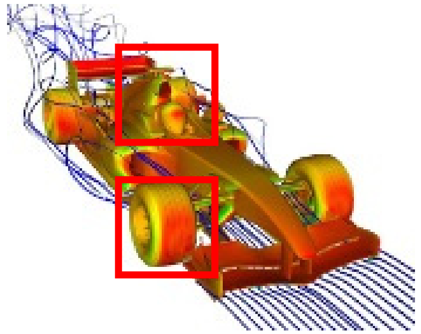
- In the Single Program Multiple Data (SPMD) paradigm each processor executes the same 'code' but with different data


Horizontal Grid (Latitude-Longitude)
Vertical Grid (Height or Pressure)
Physical Processes in a Model

modified from [7] Wikipedia on 'Numerical Weather Prediction'

- Functional Parallelism (aka MPMD)
  - N processors/cores work on on 'different sub-tasks' of the problem
  - Processors/cores work jointly together by exchanging data and do synchronization
  - E.g. Master-worker scheme
  - E.g. Functional decomposition

- In the Multiple Program Multiple Data (MPMD) paradigm each processor executes different 'code' with different data



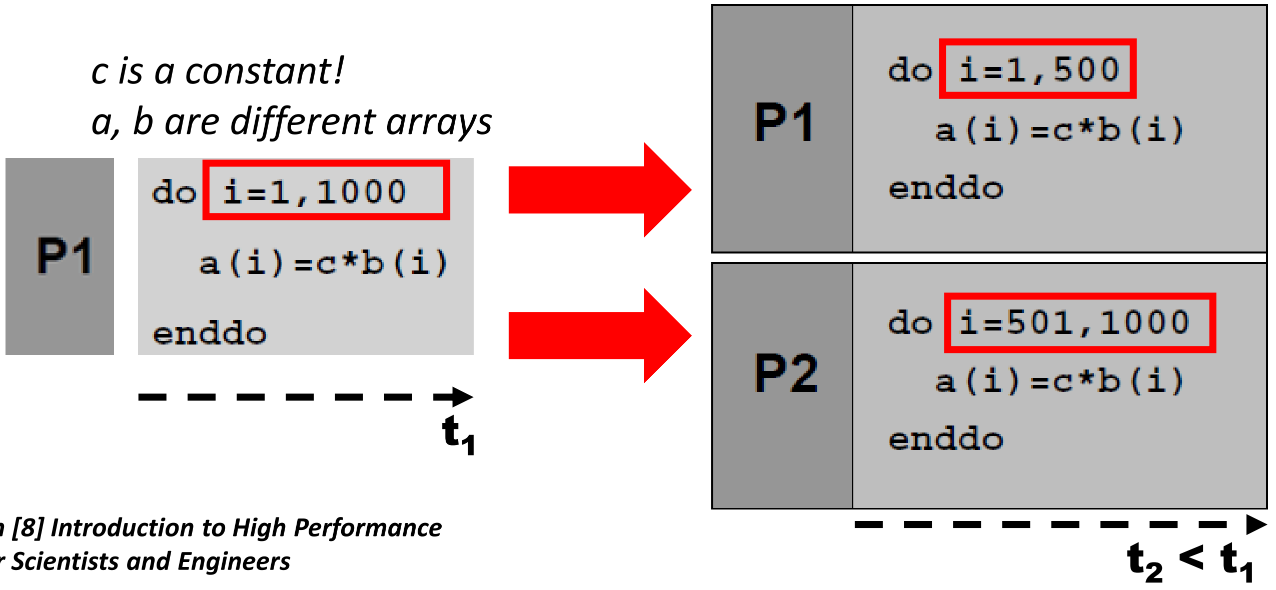
[6] Modified from Caterham F1 team

➢ **Lecture 12-15 will provide details on applied parallelization methods within parallel applications & domain/functional decomposition**

Data Parallelism: Medium-grained Loop Parallelization

- Idea: Computations performed on individual array elements are independent of each other
  - Good for parallel execution by N processors (e.g., using shared memory parallel programming)

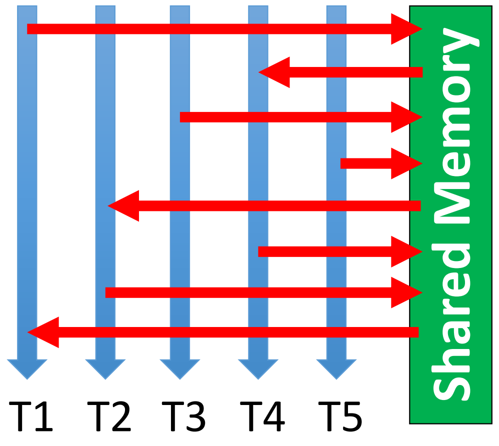
c is a constant!
a, b are different arrays



```
do i=1,1000
  a(i)=c*b(i)
enddo
```

```
do i=1,500
  a(i)=c*b(i)
enddo
```

```
do i=501,1000
  a(i)=c*b(i)
enddo
```



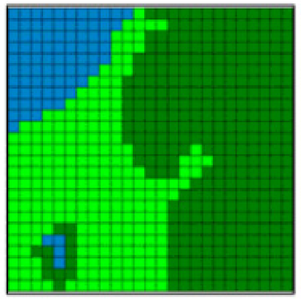
Modified from [8] Introduction to High Performance Computing for Scientists and Engineers

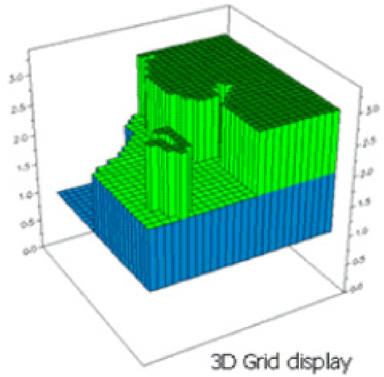
➢ **Lecture 6 will offer more elaborate shared memory parallel programming examples in context of different HPC application domains**

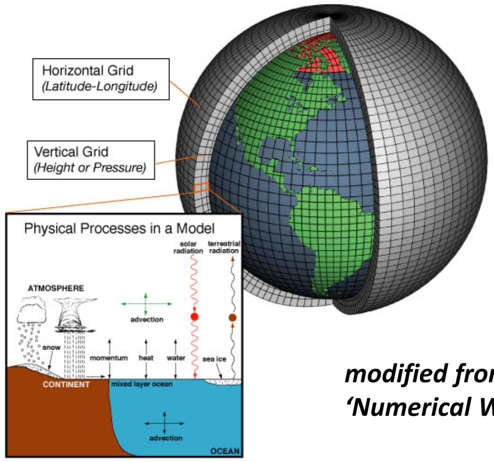
Data Parallelism: Domain Decomposition

- Approach
  - Simplified picture of reality with a 'computational domain' represented as a 'grid' (rather course-grained) or a 'mesh'
  - Grids define discrete positions for the physical quantities of the complete domain
  - Grids are not always Cartesian and often adapted to the numerical constraints of a certain algorithm in question
  - The supercomputer then simulates the reality with observables (e.g. certain physical variables) on this grid using N processors in parallel

- **Work distribution: Assign N parts of the grid to N processors**
- **In parallel computing a Grid distribution can be related to solving variables in linear equations (or find best estimates of values)**

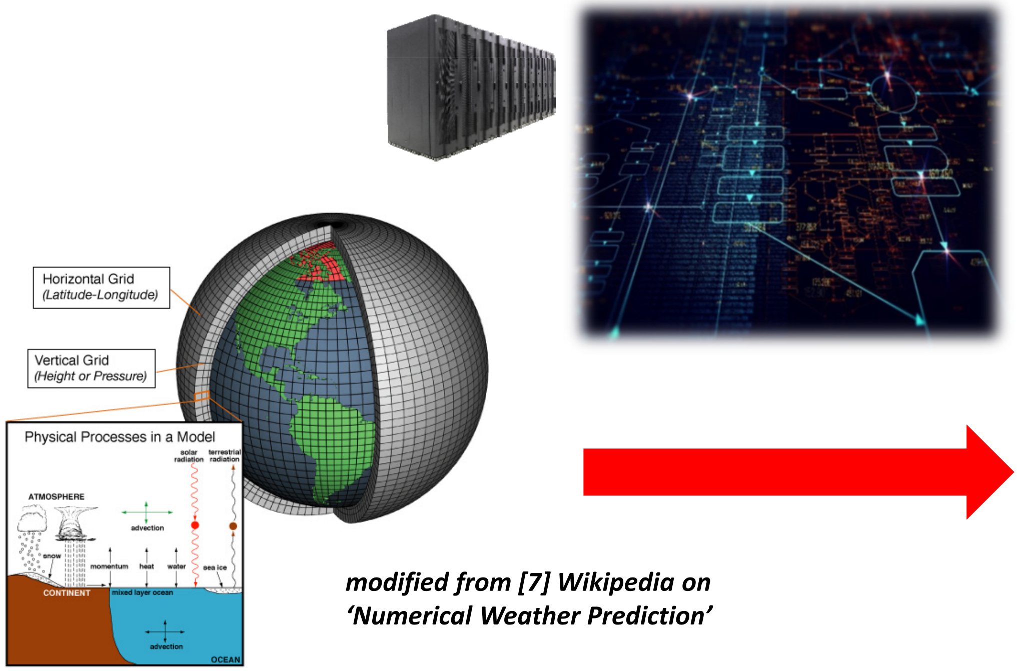

2D Grid display


3D Grid display

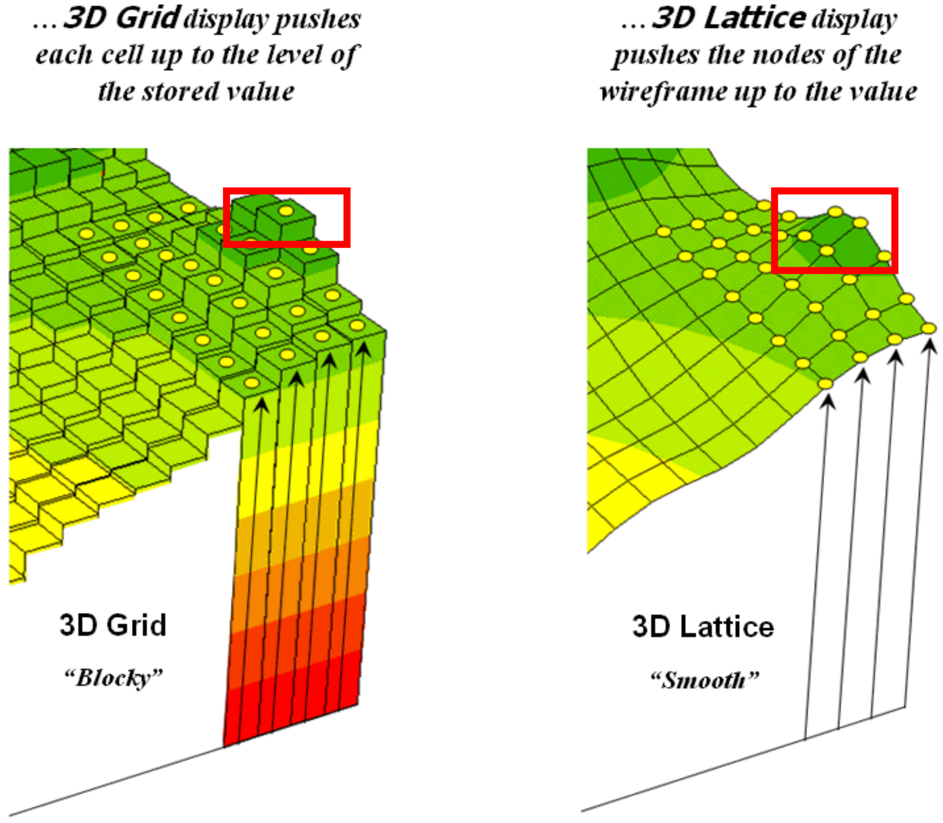


modified from [7] Wikipedia on 'Numerical Weather Prediction'

Domain Decomposition Examples: Grid vs. Lattice Approach



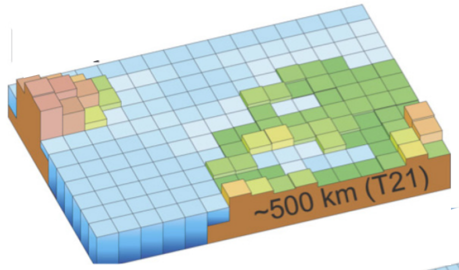
modified from [7] Wikipedia on 'Numerical Weather Prediction'


…3D Grid display pushes each cell up to the level of the stored value
…3D Lattice display pushes the nodes of the wireframe up to the value
3D Grid
"Blocky"
3D Lattice
"Smooth"

[9] Map Analysis - Understanding Spatial Patterns and Relationships, Book

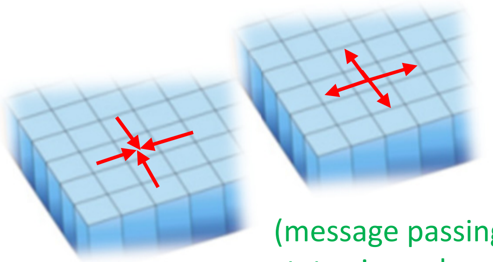
Terrestrial Systems Example – Towards Realistic Simulations – Granularity

- Scientific computing with HPC simulates ‘ ~realistic behaviour ‘
  - Apply common patterns over time & simulate based on numerical methods
  - Increasing granularity (e.g. domain decomposition) needs more computing

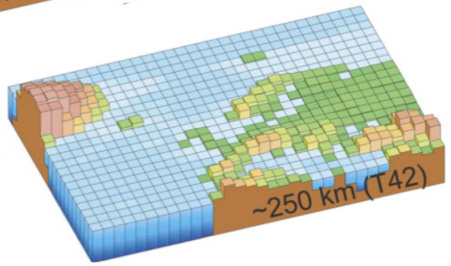


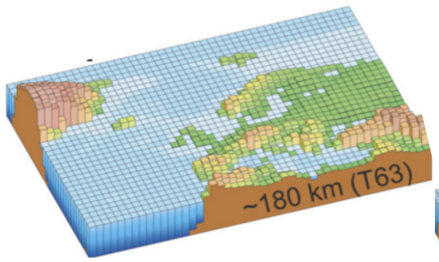
(introduce more and more physical parameters over time…)

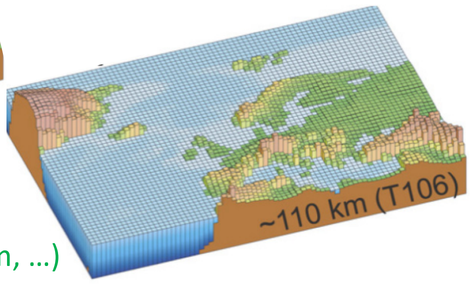
(compute more physical laws…)



(message passing of status in each cell)







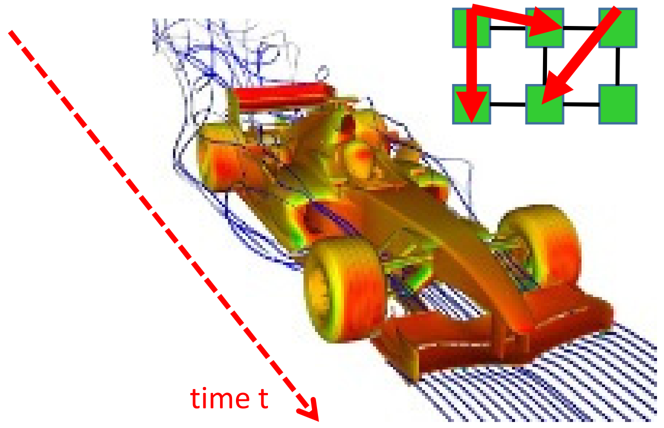
(add scientific domain studies:
e.g. rainfall, ocean waves, wind, oil, storms…)

(add objects to study: boats, fish, birds, people, oil platform, …)

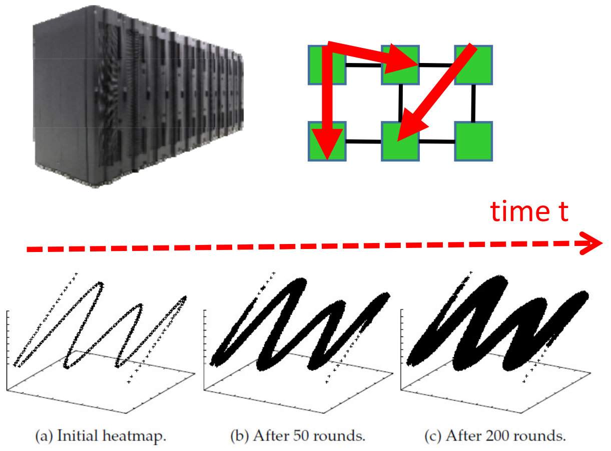
➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

Application Example: Formula Race Car Design & Room Heat Dissipation Revisited

- Pro: Network communication is relatively hidden and supported
  - Contra: Programming with MPI still requires using 'parallelization methods'
  - Not easy: Write 'technical code' well integrated in 'problem-domain code'
- Example: Race Car Simulation & Heat dissipation in a Room
  - Apply a good parallelization method (e.g. domain decomposition)
  - Write manually good MPI code for (technical) communication between processors (e.g. across 1024 cores)
  - Integrate well technical code with problem-domain code (e.g. computational fluid dynamics & airflow)



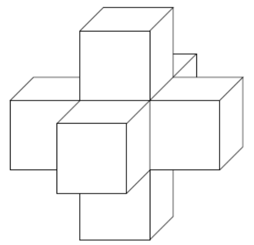
[6] Modified from Caterham F1 team

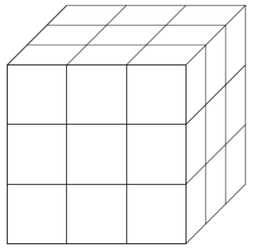


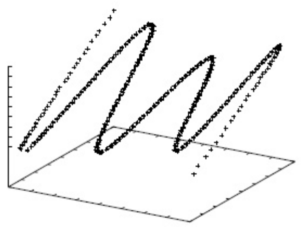
[8] Introduction to High Performance Computing for Scientists and Engineers

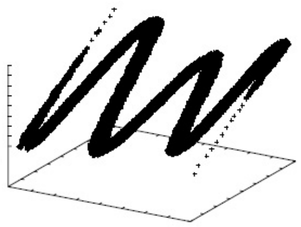
Data Parallelism: Domain Decomposition & Simple Application Example

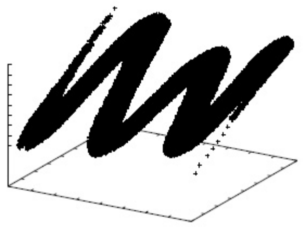
- Parallelizing a two-dimensional Jacobi solver → Heat in a room application example
  - Jacobi method is a known 'iterative method' in numerical simulations (iterative: step by step closer to the solution with approximations)
  - Application example: heat dissipation & heatmap, e.g., in a lecture room

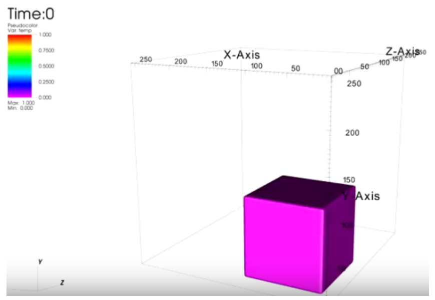

(a) Simulation one, including 6 neighbor cell.

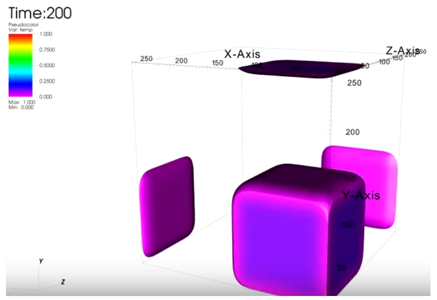

(b) Simulation two, including 26 neighbor cells.

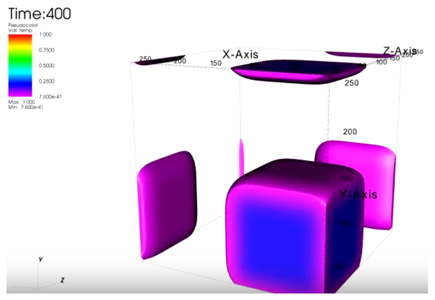

(a) Initial heatmap.


(b) After 50 rounds.


(c) After 200 rounds.







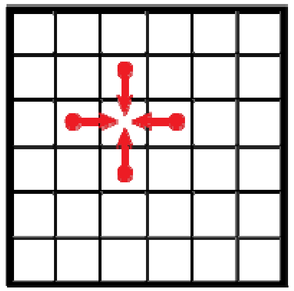
[10] Templates for the solution of linear systems

[11] YouTube Video, heat dissipation Jacobi Method

Data Parallelism: Formulas Across Domain Decomposition

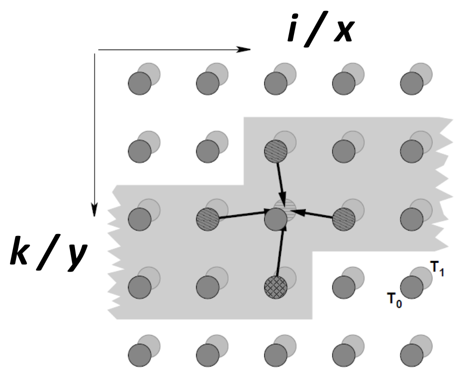
- From the problem to computational data structures
  - Apply an ‘isotropic lattice‘ technique

```
do k = 1,kmax
  do i = 1,imax
    ! four flops, one store, four loads
    phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                   + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25
  enddo
enddo
```



Modified from [8] Introduction to High Performance Computing for Scientists and Engineers

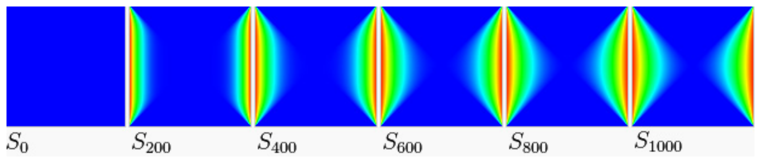
[10] Wikipedia on ‘stencil code’

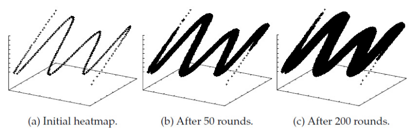


$$\frac{\delta\Phi(x_i,y_i)}{\delta t} = \frac{\Phi(x_{i+1},y_i)+\Phi(x_{i-1},y_i)-2\Phi(x_i,y_i)}{(\delta x)^2} + \frac{\Phi(x_i,y_{i-1})+\Phi(x_i,y_{i+1})-2\Phi(x_i,y_i)}{(\delta y)^2}$$

‘change over time’ diffusion equation

$$\frac{\partial\Phi}{\partial t} = \Delta\Phi$$

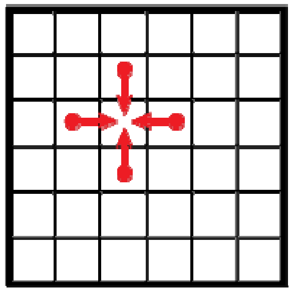


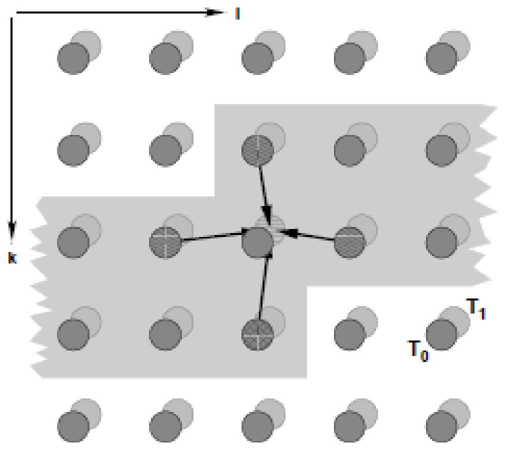


➢ **Lecture 10 on Hybrid Programming and Patterns will offer more details on stencil methods & patterns in simulation science applications**

Data Parallelism: Domain Decomposition & Equations

- Example: Parallelizing a two-dimensional Jacobi solver
  - Jacobi method is a known 'iterative method' in numerical simulations (iterative: step by step closer to the solution with approximations)
  - Solving n linear equations with n unknown variables, diagonal dominance
    - Picking start values and iterate towards a ~final solutions (reducing errors/step)
  - Goal: Update physical variables on a 'N x N grid' until approximations good enough (maybe only solution to 97%, but enough & shorter time)
  - Domain decomposition for N processors subdivides the computational domain in N subdomains





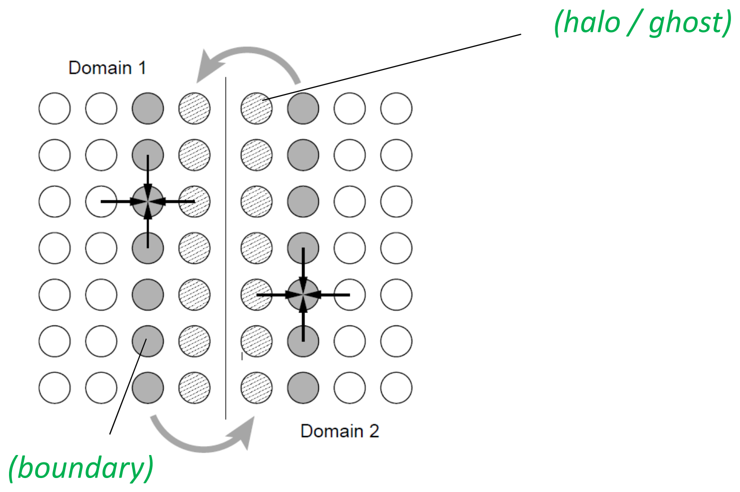
Find (approximate) values for K and I → update arrays

In each time step (e.g. T_1) re-using values from previous iteration (e.g. T_0)

[8] Introduction to High Performance Computing for Scientists and Engineers

Data Parallelism: Domain Decomposition & Halo/Ghost Layers/Cells

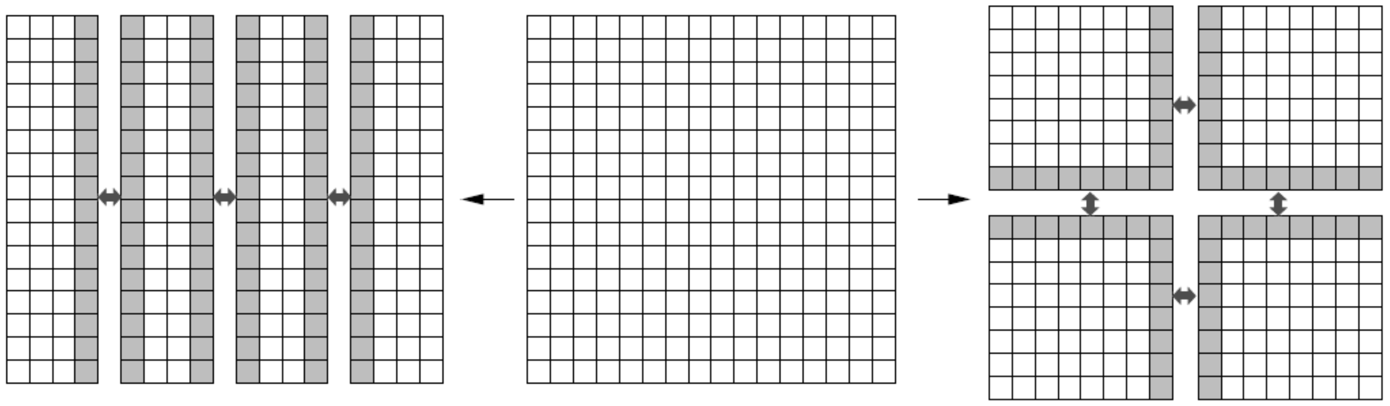
- Two-dimensional Jacobi solver in context of parallel systems:
  - Shared-memory and complete domain fits into memory
  - Relatively easy: all grid sites in all domains can be updated before the processors have to synchronize at the end of the sweep (i.e. time step)
- Distributed-memory with no access to ‘neighbours memory’
  - Complex: updating the boundary sites of one domain requires data from adjacent domain(s)
  - Idea: before a domain update (next step), all boundary values needed for the upcoming sweep must be communicated to the relevant neighboring domains
  - We need to store this data somewhere, so extra grid points introduced (*halo/ghost layers/cells*)



[8] Introduction to High Performance Computing for Scientists and Engineers

Data Parallelism: Domain Decomposition & Communication

- Two-dimensional Jacobi solver in context of communication cost:
  - Often choosing the optimal domain decomposition is application-specific
  - Next neighbour interactions needed and can vary (more/less shaded cells)
  - Simple: Cutting in four stripes domains (left) incurs more communication
  - Optimal decomposition: four domains (right) incurs less communication

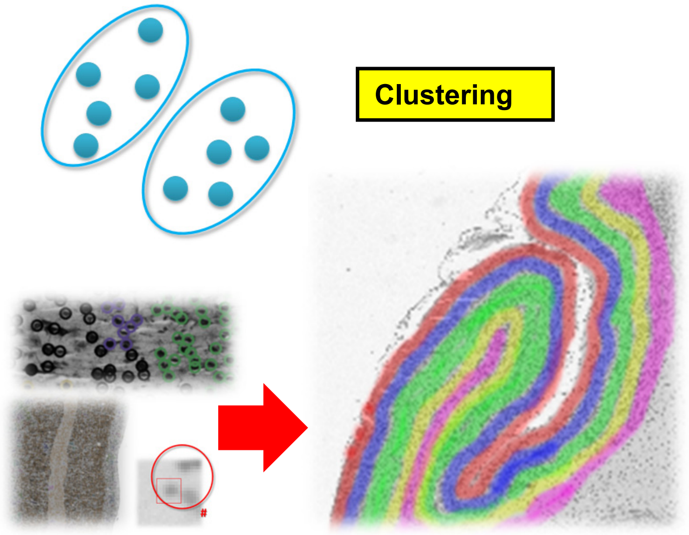


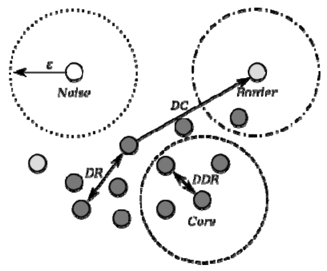
3 * 16 = 48

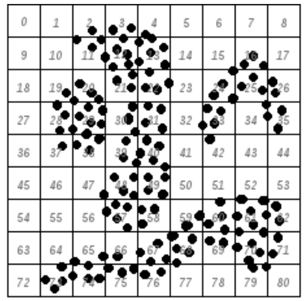
4 * 8 = 32

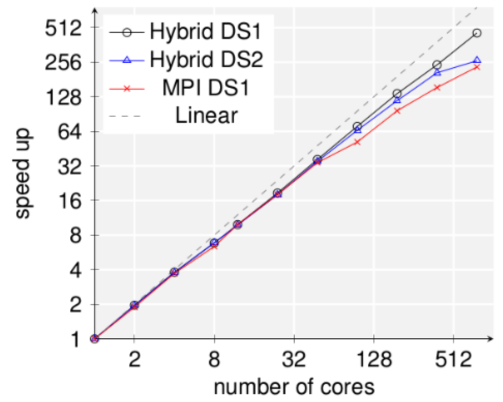
[8] Introduction to High Performance Computing for Scientists and Engineers

Data Parallelism Example: Smart Domain Decomposition in Data Sciences

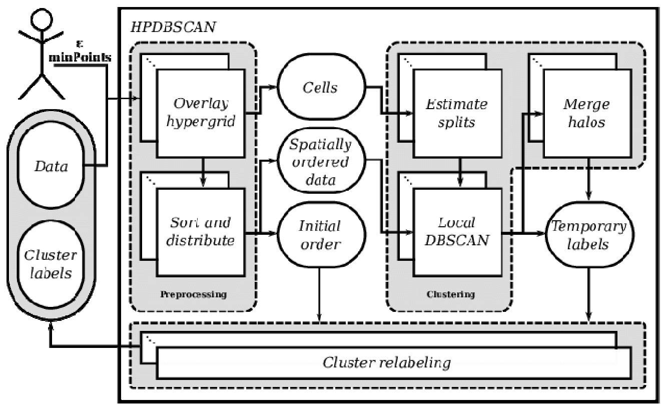




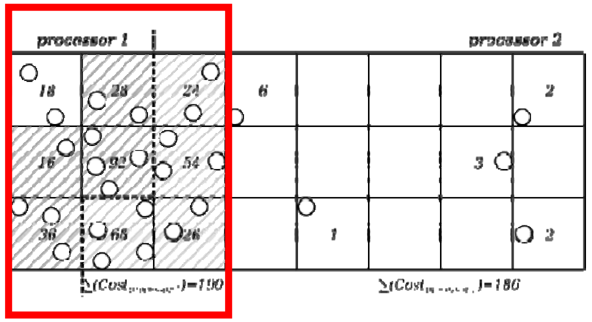

Overlayed spatial grid



[13] M. Goetz and M. Riedel et al,
Proceedings IEEE Supercomputing Conference, 2015



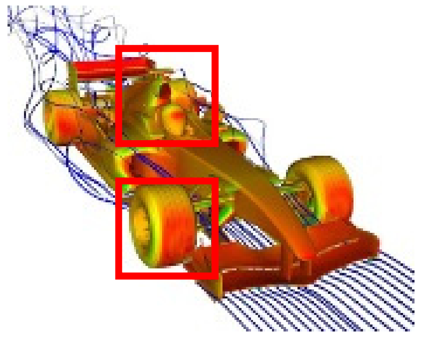
cluster merge across halo regions/layers



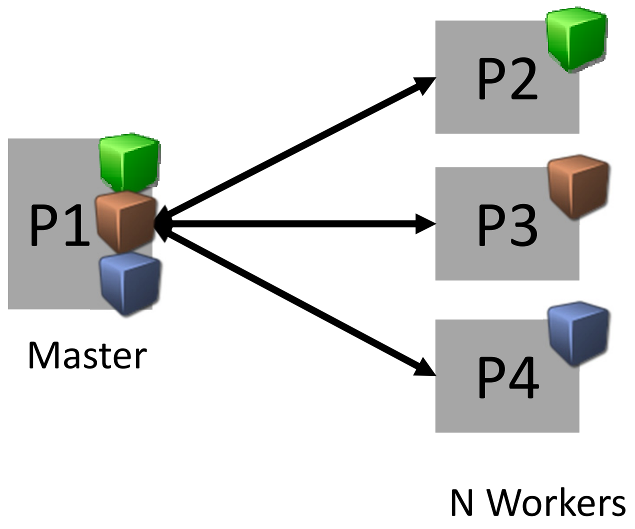
- **Lecture 8 will provide more details on MPI application examples with a particular focus on parallel and scalable machine learning**

Functional Parallelism: Master-Worker Scheme

- Idea:
  - One processor performs administrative tasks while others solve a particular problem jointly together
- Master
  - Distributes work and collects results from workers
  - Could be single bottleneck
- N Workers (old: slaves)
  - Whenever a worker has finished a package it stops or requests a new task from the master depending on the application
- Example: Find largest element in array
  - Which CPU/core does the global max?



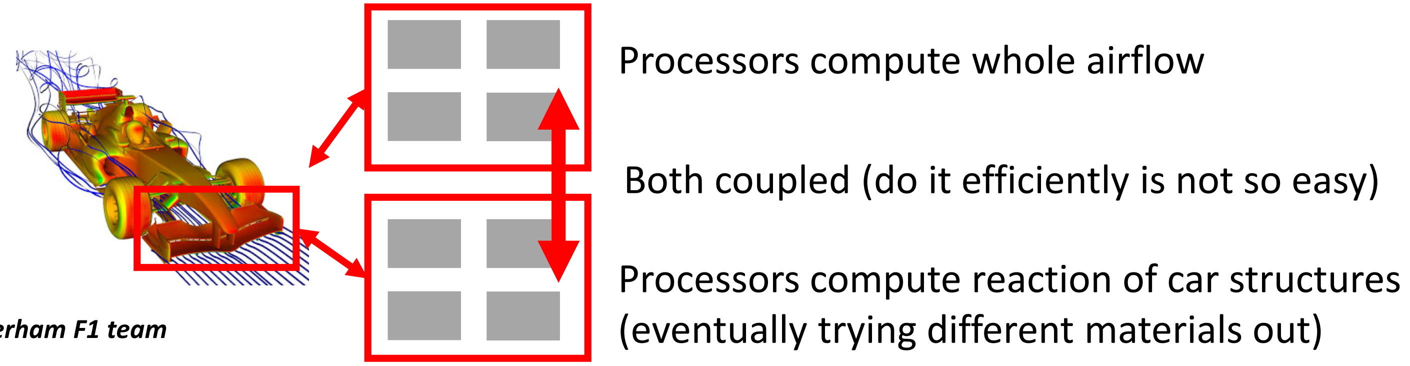
[6] Modified from Caterham F1 team



➢ **Lecture 15 will offer more details on HPC applications that perform computational fluid dynamics (CFD)**

Functional Parallelism: Functional Decomposition

- Approach
  - Couple different running codes to compute functions that jointly are used to solve a higher-level problem
- Example: Multi-physics simulation of a race car to be coupled with communication layer
  - (Multi-physics problems gaining popularity since they reflect better reality)
  - Air flow around race car with Computational Fluid Dynamics (CFD) code
  - Parallel finite element simulation could describe the reaction of the flexible structures of the car body to the computed air flow (involves accurate geometry and material properties in context)

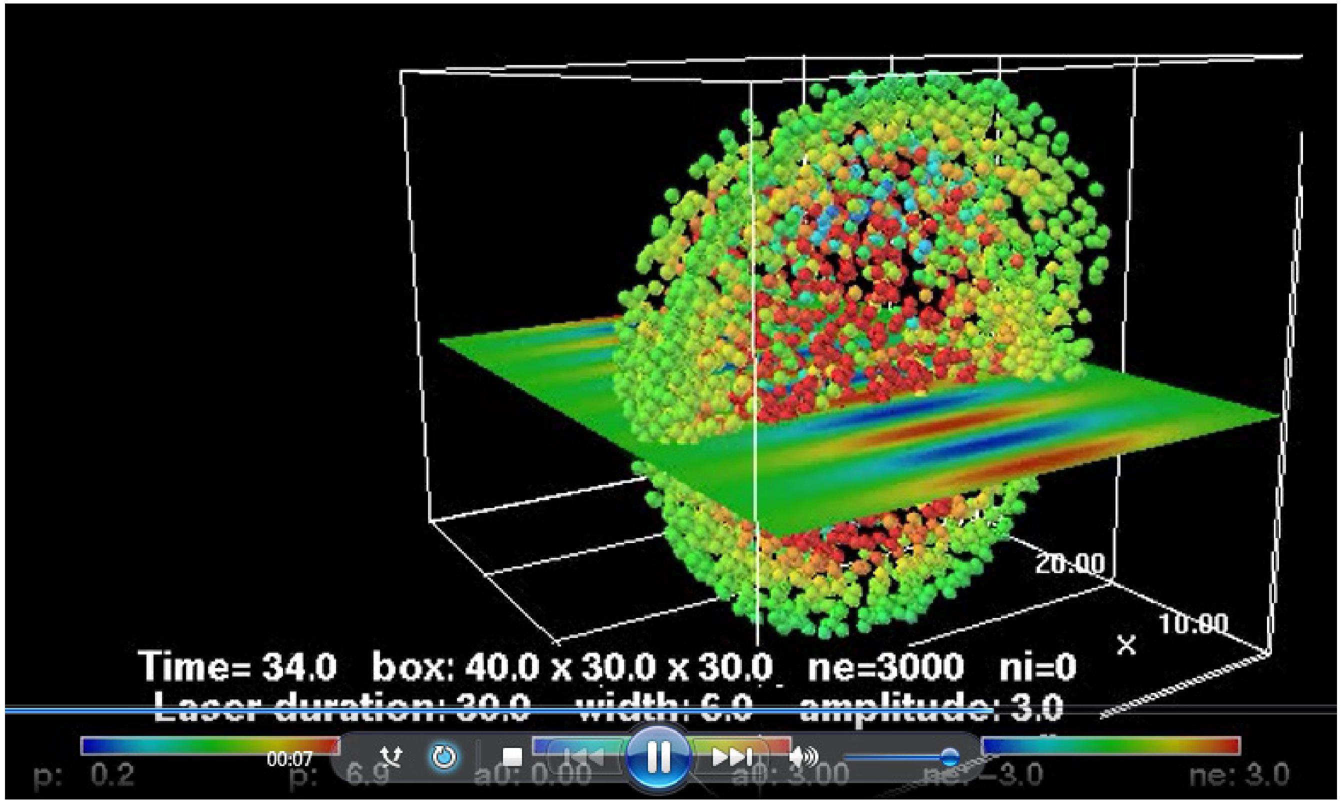


Processors compute whole airflow
Both coupled (do it efficiently is not so easy)
Processors compute reaction of car structures (eventually trying different materials out)

[6] Modified from Caterham F1 team

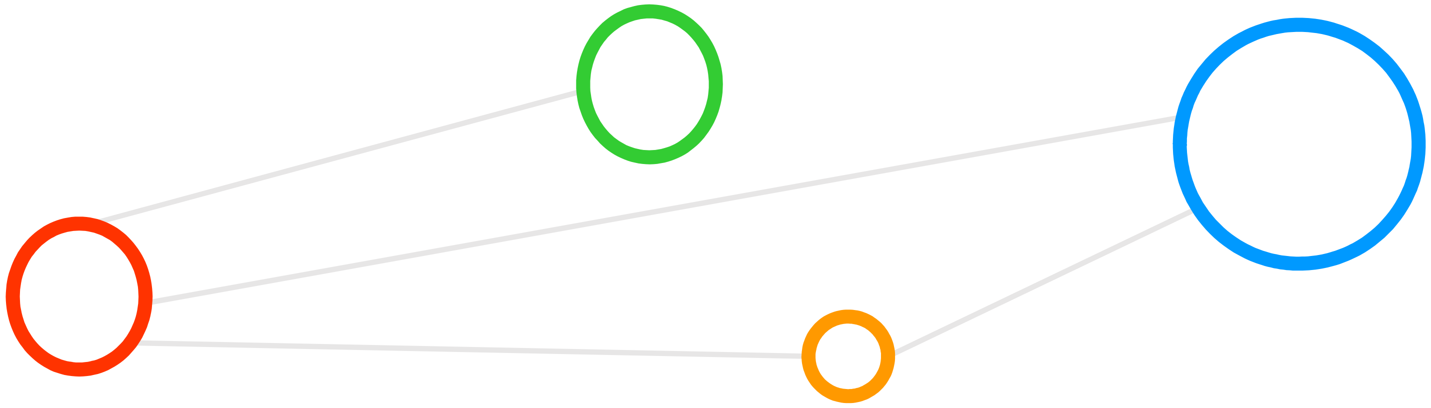
➢ **Lecture 15 will offer more details on HPC applications that perform computational fluid dynamics (CFD) on domain decompositions**

[Video] PEPC – Particle Acceleration Application


20.00
10.00
x
Time= 34.0 box: 40.0 x 30.0 x 30.0 ne=3000 ni=0
Laser duration: 30.0 width: 6.0 amplitude: 3.0
p: 0.2
p: 6.9
a0: 0.00
a0: 3.00
ne: −3.0
ne: 3.0
00:07

[14] PEPC Video Application Example

Parallelization Terms & Theory

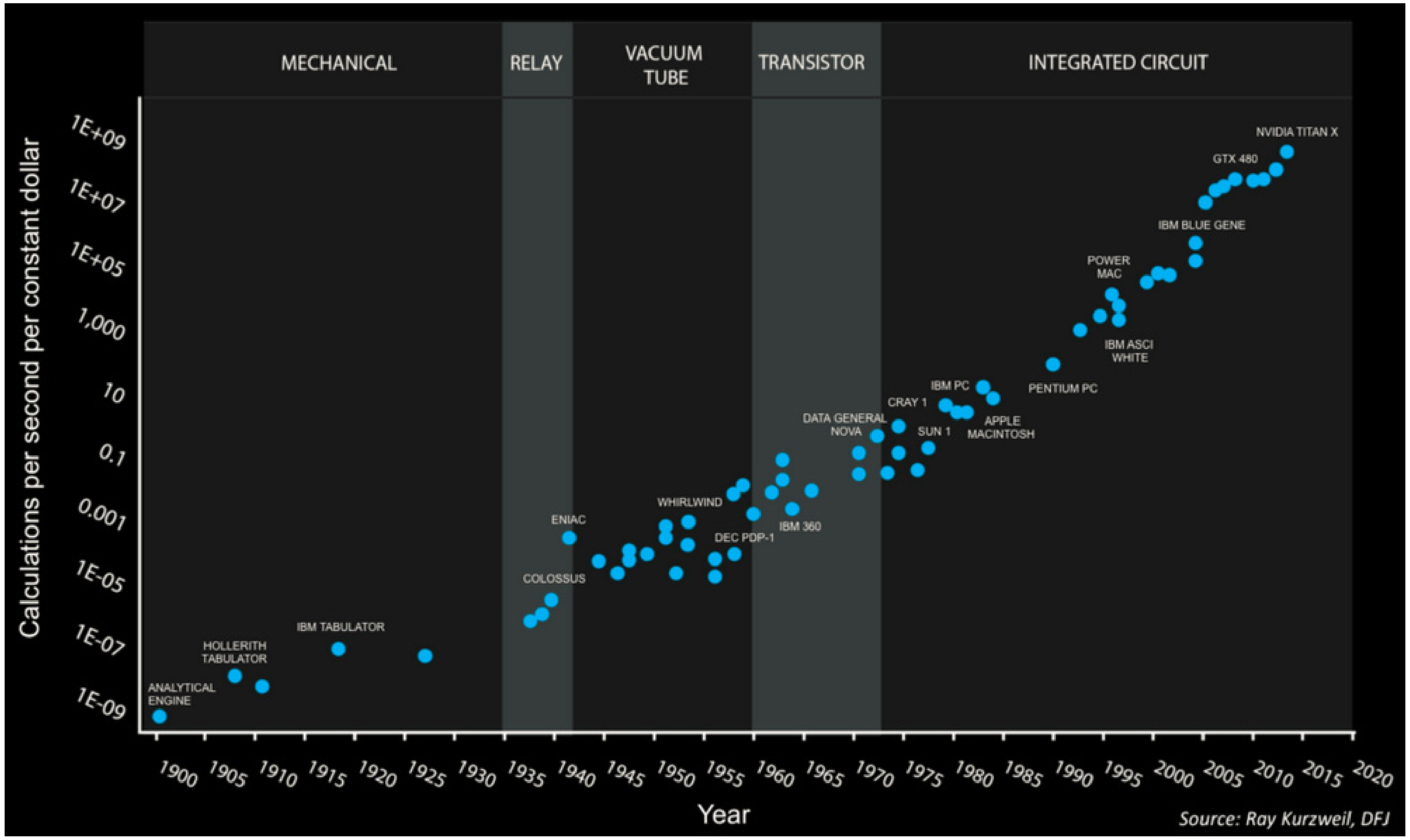


Parallelization in High Performance Computing

- Parallelization in HPC is essential due to the following capabilities
  - Perform calculations, visualizations, and data processing…
  - … at an incredible, ever-increasing speed
  - … at an unprecedented granularity and / or accuracy

- **HPC uses parallel computing in order to tackle problems & increase insights**
- **HPC can perform virtual experiments that are too dangerous or too expensive**
- **HPC enables simulation of real-world phenomena not possible otherwise**
- **HPC automates re-occuring processing of large quantities of data or many equations**

Moore’s Law



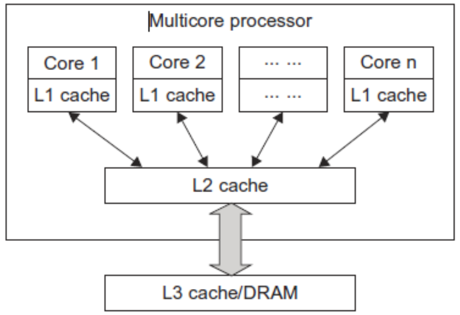
(seven last dots are actually
many-core GPGPUs,
cf. Lecture 1)

[15] Wikipedia ‘Moore’s Law’

- **Moore’s Laws says that the number of transistors on integrated circuits doubles approximately every two years (exponential growth)**

Reasons for Parallelization

- The concept of 'parallelization' getting more mainstream today
  - Supercomputers (that are massively parallel computers today)
  - Multi-core PCs and Laptops (with increasing amount of cores, 2x, 4x, etc.)
  - Many-core GPUs not only used for graphics but also for general processing

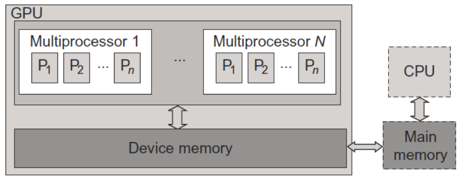


- Two major reasons to engage in parallelization

▪ A single core is too slow to perform the required task(s) in a certain constrained amount of time

▪ The available memory on a single system is not sufficient to tackle a problem in a required granularity or precision.

Derived from [8] Introduction to High Performance Computing for Scientists and Engineers

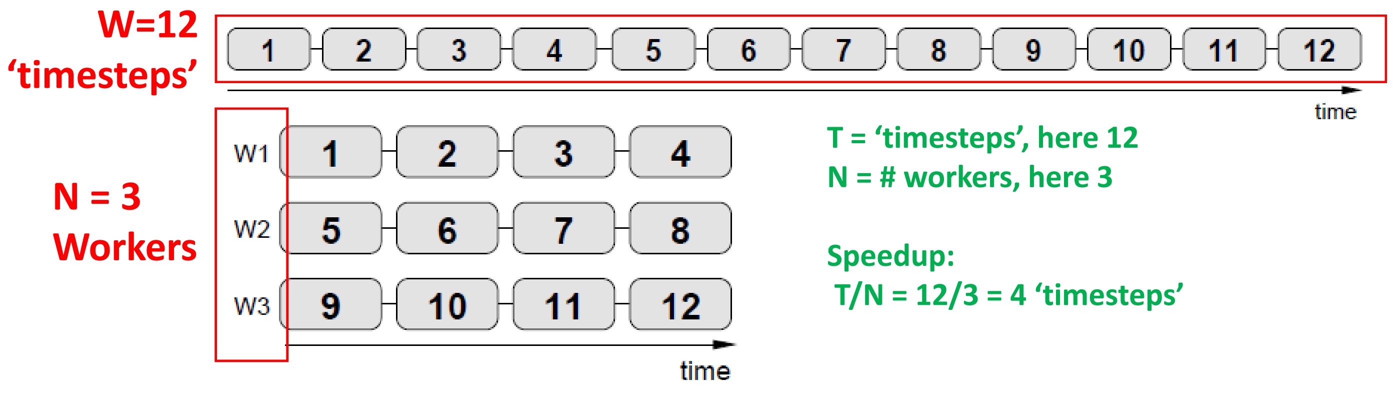


- The reason influences the choosen 'parallelization method(s)'
  - Example: SPMD or MPMD

[3] Distributed & Cloud Computing Book

Parallelization Goal: Speedup Term

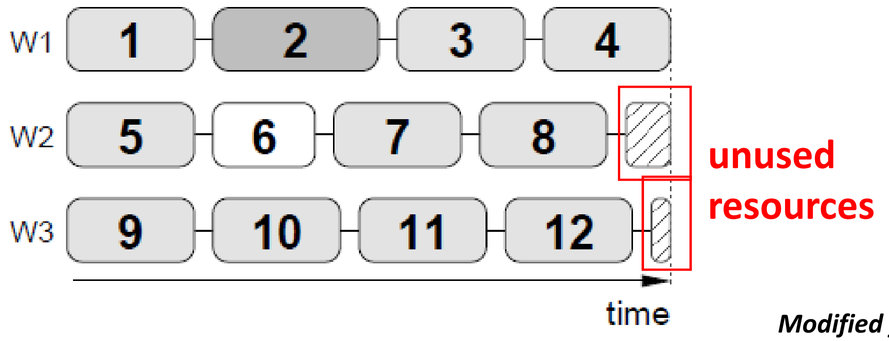
- Consider simple situation: All processing units execute their assigned work in exactly the same amount of time
  - Solving a problem would take Time T sequentially (1 Worker essentially)
  - Having *N* workers solve the problem now ideally only in *T/N*
  - This is a speedup of N



Modified from [8] Introduction to High Performance Computing for Scientists and Engineers

Parallelization Challenge: Load Imbalance Term

- Consider a more realistic situation: Not all workers might execute their tasks in the same amount of time
  - Reason: The problem simply can not be properly partitioned into pieces with equal complexity
  - Nearly worst case: All but a few have nothing to do but wait for the latecomers to arrive (because of different execution times)



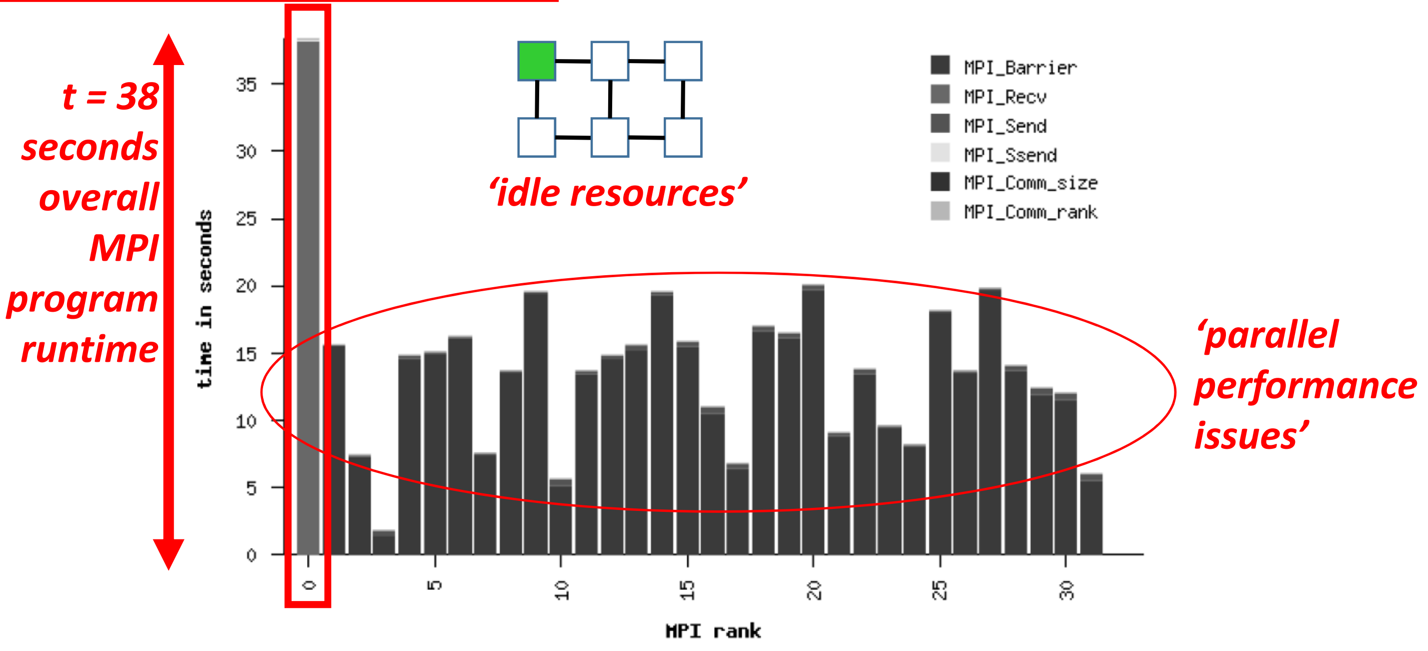
- Load imbalance hampers performance, because some resources are underutilized

Modified from [8] Introduction to High Performance Computing for Scientists and Engineers

Load Imbalance Example

- Parallel Programming Problems
  - Wrong assumptions in distributed-memory programming
  - Cost and side effects of the programmed communications

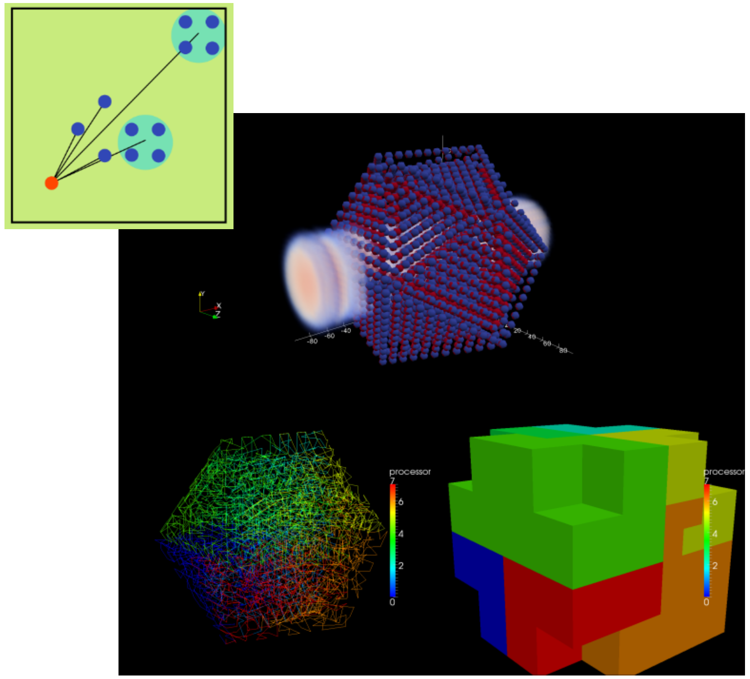
- General Problems
  - Serial execution limits
  - Load Imbalance
  - Unnecessary synchronizations

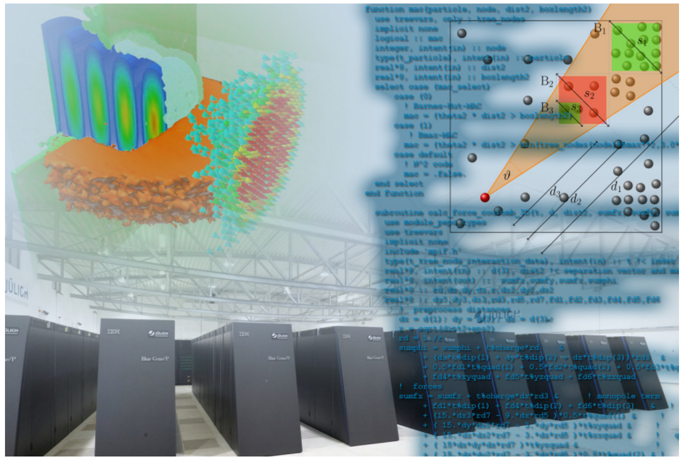


[8] Introduction to High Performance Computing for Scientists and Engineers

Parallelization Challenges: Optimal Domain Decompositions

- Tree codes – ‘another form of smart domain decomposition‘
  - E.g. to speed up *N*-body simulations with long range interactions





- **Fast and/or high performance means many n floating point operations (FLOP) per one second**

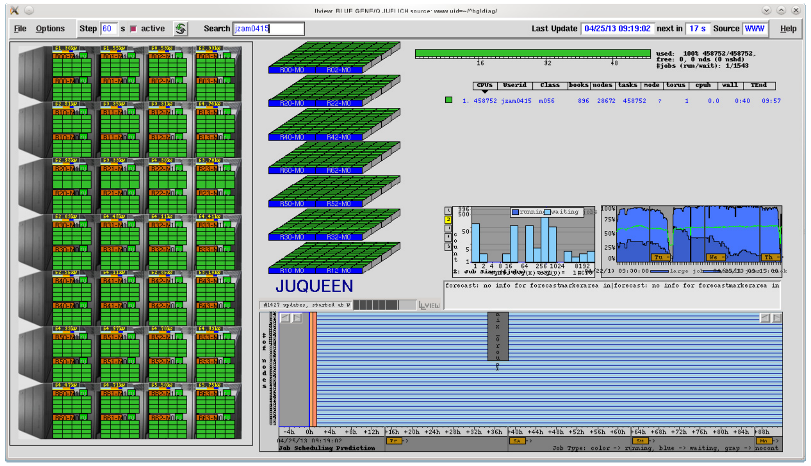
[15] PEPC Webpage

Towards Fast & Scalable Applications

- Many factors influence the scalablility of an application
  - Benefits of smart domain decomposition methods is just one factor
  - E.g. PEPC Tree-code on whole BlueGene/Q

[17] Wikipedia on 'scalability'

[15] PEPC Webpage



```
used:  100% 458752/458752,
free:  0, 0 nds (0 nshd)
#jobs (run/wait): 1/1543
```

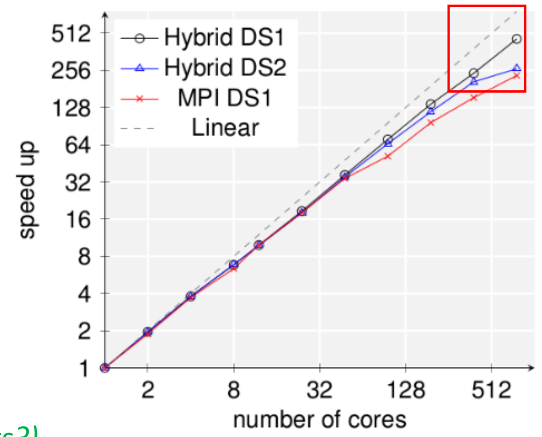
CPUs	Userid	Class	books	nodes	tasks	mode	torus	cpuh	wall	TEnd
1. 458752	jzam0415	m056	896	28672	458752	?	1	0.0	0:40	09:57

- **Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth**
- **Measure the execution time T of a parallel programm for speed-up & scaling**
- **Scaling a parallel program means how the performance of it changes as the number of processors increases using two metrics: Strong/Weak Scaling**

- Raises several questions and challenges
  - What means faster? How we get to an application that is scalable?

Terminologies: Speed-Up & Scaling

- Achieving speed-up
  - Increasing cores, e.g. 2 instead of 1: 50% improve?
  - Reality: No! 'Tail off' → loose of parallel efficiency
  - Q: How many cores we can use in our code?
    A: Scaling: change number of processors – goals?
  - Different types of scaling for different parallization goals



[13] M. Goetz and M. Riedel et al, Proceedings IEEE Supercomputing Conference, 2015

- Scalability Metric: Strong Scaling *(obvious/intuitive way → what parallelization benefits?)*
  - Total problem size stays the same – increases number of processors
  - More difficult to achieve, but (often) more useful than weak scaling
- Scalability Metric: Weak Scaling *(room for improvement → what is possible to do?)*
  - Total problem size increases with the rate of processors – keeping work per processor the same → Getting to the limits, what is possible to do?
  - Think about is to get towards reality → getting better simulations

Scalability Metrics: Strong Scaling

- $\mathbf{T_f^S}$ = single worker serial runtime for a fixed problem size:

$$\boxed{T_f^S = s + p}$$

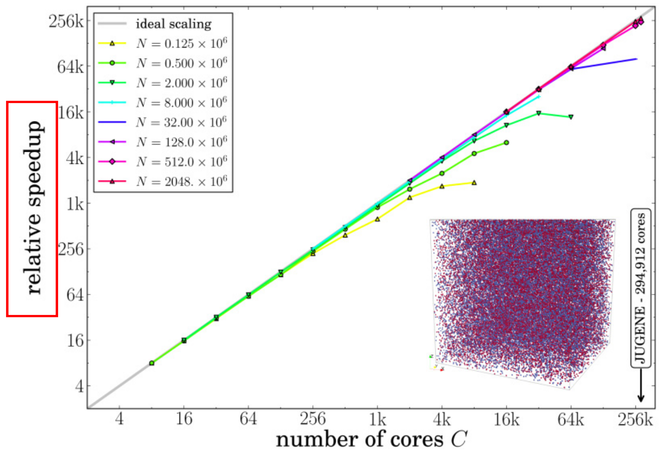
- $\mathbf{T_f^P}$ = time to tackle the fixed problem size with N parallel workers:

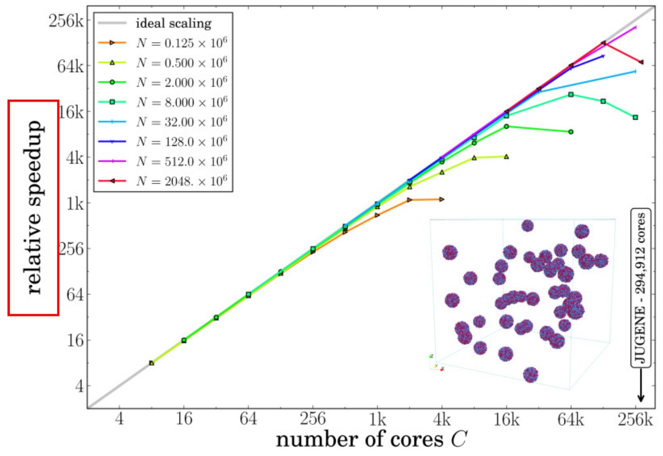
$$\boxed{T_f^P = s + p/N}$$

- $\mathbf{T_f^P}$ means then: Strong scaling ('very good'!)
  - The amount of work stays constant…
  - … with increasing number of workers N
  - Goal of parallelization:
    Minimization of time to solution for a problem

- **Scalability metrics quantify how well a task can be parallelized for different goals**
- **Two major quantities in HPC are named as 'Strong Scaling' and 'Weak Scaling'**
- **Tail off in parallel efficiency in strong scaling: parallel overhead gets visible as processors do less and less (of fixed problem size) when increasing numbers of processors more**

Strong Scaling Example





- Tree-code PEPC
  - Execution system: BlueGene/P (JUGENE@Juelich)
  - Homogenous particle distribution (top, different use cases)
  - Inhomogenous particle distribution (bottom, different use cases)
  - Different test particle setups used
- Y: Total particle numbers N
  - Note change of N in plots
- X: Total compute core number C

- **Strong Scaling: How the time to solution varies with the number of processors for a fixed total problem size**

[15] PEPC Webpage

Scalability Metrics: Weak Scaling

- Goal of parallelization: tackle problem too big for single machine
  - E.g. available memory is a limiting factor
  - Idea: scale the problem size at least with some power of N; α positive
- T_v^S = serial runtime for a scaled (variably-sized) problem:

$$\boxed{T_v^S = s + p * N^{\alpha}}$$

- T_v^P = parallel runtime for a scaled (variably-sized) problem:

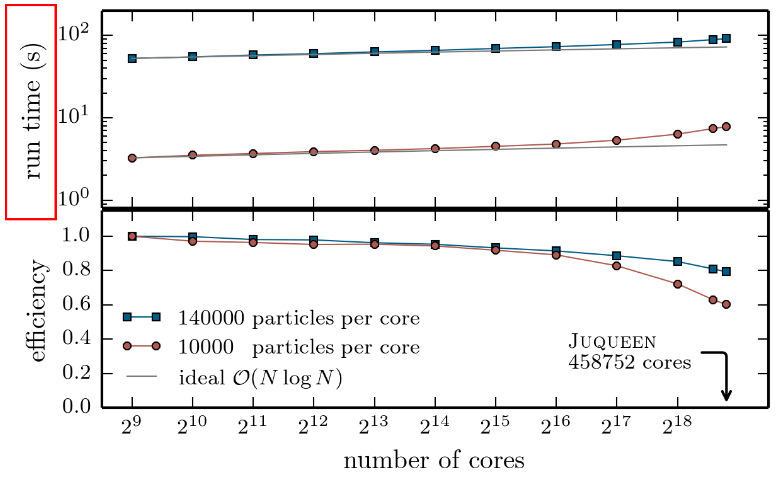
$$\boxed{T_v^P = s + p * N^{\alpha - 1}}$$

- T_v^P means then: Weak scaling ('not very good'!)
  - If special case $\alpha = 1 \rightarrow N^0 = 1 \rightarrow$ you did not parallelize something really

- **Tail off in weak scaling: Reasons can be communication overheads by adding more and more processors and problems sizes that in turn increases the runtime**

Weak Scaling Example – Gustafson‘s Law

- Tree-code PEPC
  - Execution system: BlueGene/Q (JUQUEEN@Juelich)
- Weak scaling plot (top)
  - Two numbers of particles kept constant; test particle setups constant
- Parallel efficiency (below)



- **Gustafson’s law: ‘We need larger problems for larger numbers of CPUs’**
- **Assumption is that the parallel part is proportional to our problem size**
- **Result: Bigger problems ‘just‘ scale better, since serial parts get more insignificant**

(work that each processor needs to do stays fixed by increasing cores)

- **For comparison: Strong Scaling: How the time to solution varies with the number of processors for a fixed total problem size**

- **Weak Scaling: How the time to solution varies with the number of processors for a fixed problem size/processor**

[15] PEPC Webpage

Application Speedup – Amdahl’s Law

- Scalability is dependend from the serial application parts
  - More related to the ‘strong scaling‘ of a parallel program

$$S_f = \frac{P_f^P}{P_f^S} = \frac{1}{s + \boxed{\frac{1-s}{N}}} \Rightarrow$$

0 for $N \rightarrow \infty$

[18] G. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, 1967

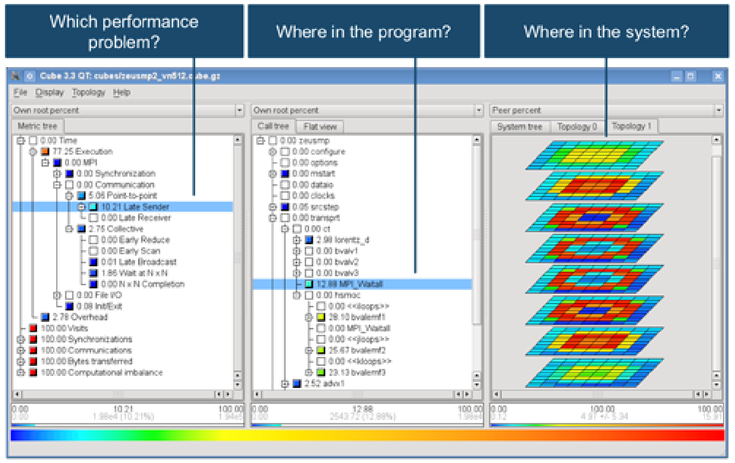
- **1 - s** is the ‘parallizable part’ of the problem
- When unlimited workers in place we have $\boldsymbol{N \rightarrow \infty}$
- Amdahl’s law limits application speedup thus to 1/*s*

> - **Amdahl’s laws says that scaling of massively parallel applications is hindered by the domination of its serial parts (strong scaling)**

> - **For comparison: Gustafson’s law says we ‘just’ need larger problems thus no limits by serial parts (weak scaling)**

Performance Analysis is a Key Field in HPC

- Analysis is typically performed using (automated) software tools
  - Measure and analyze the runtime behaviour of parallel programs
  - Identifies potential performance bottlenecks
  - Offer performance optimization hints and views of the location in time
  - Guides exploring causes of bottlenecks in communication/synchronization



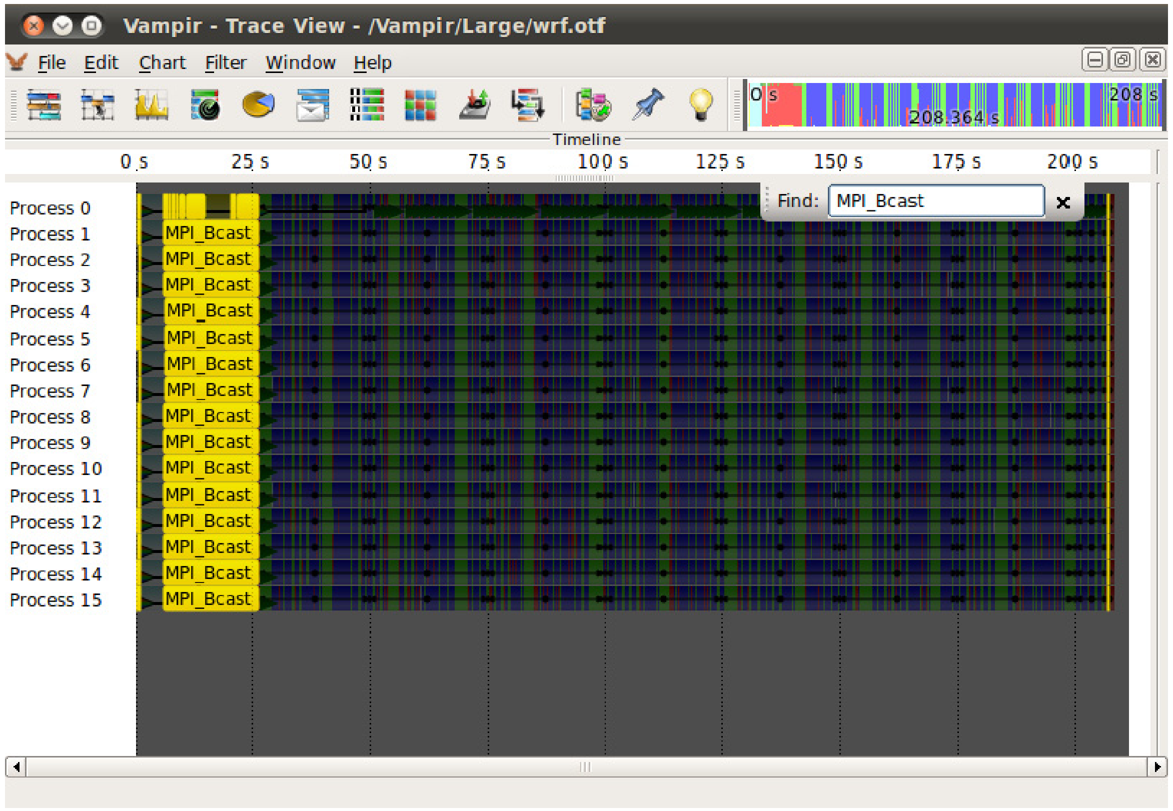
```
0.00 Time
  77.25 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        5.06 Point-to-point
          10.21 Late Sender
          0.00 Late Receiver
        2.75 Collective
          0.00 Early Reduce
          0.00 Early Scan
          0.01 Late Broadcast
          1.86 Wait at N x N
          0.00 N x N Completion
      0.00 File I/O
      0.08 Init/Exit
  2.78 Overhead
100.00 Visits
100.00 Synchronizations
100.00 Communications
100.00 Bytes transferred
100.00 Computational imbalance
```

```
0.00 zeusmp
  0.00 configure
  0.00 options
  0.00 mstart
  0.00 dataio
  0.00 clocks
  0.05 srcstep
  0.00 transprt
    0.00 ct
      2.98 lorentz_d
      0.00 bvalv1
      0.00 bvalv2
      0.00 bvalv3
      12.88 MPI_Waitall
      0.00 hsmoc
        0.00 <<iloops>>
        28.10 bvalemf1
        0.00 MPI_Waitall
        0.00 <<jloops>>
        25.67 bvalemf2
        0.00 <<kloops>>
        23.13 bvalemf3
    2.52 advx1
```

[20] SCALASCA Performance Tool

> ➢ **Lecture 9 will give details on how to measure performance in parallel programms & and related tools using various applications**

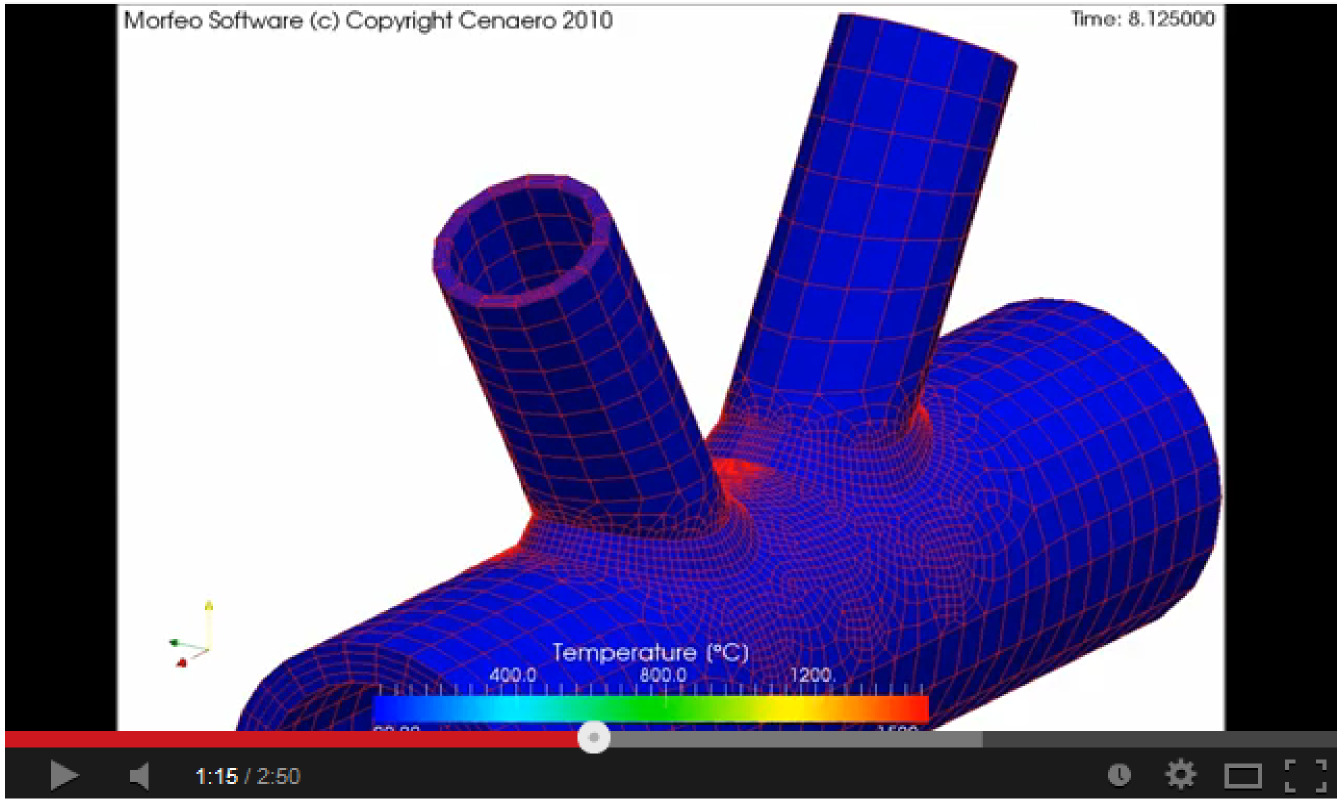
Performance Analysis in Distributed-Memory Programming



[21] Vampir Performance Tool

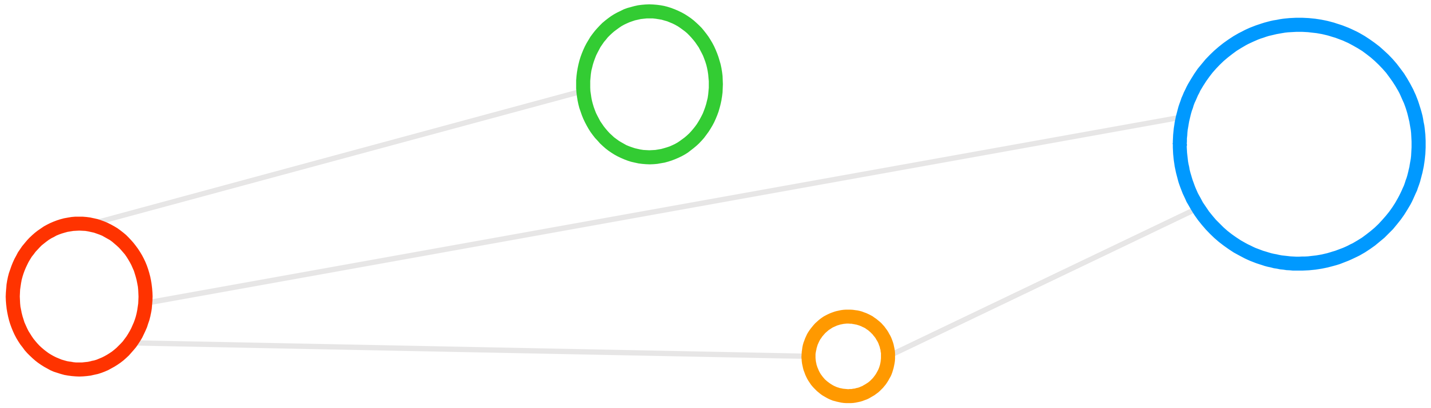
➢ **Lecture 9 will give details on how to measure performance in parallel programms & and related tools using various applications**

[Video] Parallelization From Theory to Practice



[19] Power! Youtube Video

Lecture Bibliography



Lecture Bibliography (1)

- [1] LLNL MPI Tutorial, Online:
  https://computing.llnl.gov/tutorials/mpi/
- [2] TOP500 Supercomputing Sites, Online:
  http://www.top500.org/
- [3] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book, Online:
  http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [4] 2013 SMU HPC Summer Workshop, Session 8: Introduction to Parallel Computing, Online:
  http://dreynolds.math.smu.edu/SMUHPC_workshop/session_8.html
- [5] Introduction to Parallel Computing Tutorial, Online:
  https://computing.llnl.gov/tutorials/parallel_comp/
- [6] Caterham F1 Team Races Past Competition with HPC, Online:
  http://insidehpc.com/2013/08/15/caterham-f1-team-races-past-competition-with-hpc
- [7] Wikipedia on 'Numerical Weather Prediction', Online:
  http://en.wikipedia.org/wiki/Numerical_weather_prediction
- [8] Introduction to High Performance Computing for Scientists and Engineers,
  Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [9] Map Analysis, Understanding Spatial Patterns and Relationships, Joseph K. Berry, Online:
  http://www.innovativegis.com/basis/Books/MapAnalysis/Default.htm
- [10] Templates for the solution fo linear systems, building blocks for iterative methods, book, Online:
  http://www.netlib.org/linalg/html_templates/Templates.html
- [11] Jacobi Heat Dissipation, Online:
  https://www.youtube.com/watch?v=jBbanIGoIhE

Lecture Bibliography (2)

- [12] Wikipedia on ‘stencil code‘, Online:
  http://en.wikipedia.org/wiki/Stencil_code
- [13] M. Goetz, C. Bodenstein, M. Riedel, ‘HPDBSCAN – Highly Parallel DBSCAN’, in proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, 2015, Online:
  https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN
- [14] PEPC Video Application Example, FZ Juelich, Online:
  http://www.fz-juelich.de/ias/jsc/EN/AboutUs/Organisation/ComputationalScience/Simlabs/slpp/SoftwarePEPC/_node.html
- [15] Wikipedia ‘Moore’s Law’, Online:
  http://en.wikipedia.org/wiki/Moore's_law
- [16] PEPC Webpage, FZ Juelich, Online:
  http://www.fz-juelich.de/ias/jsc/EN/AboutUs/Organisation/ComputationalScience/Simlabs/slpp/SoftwarePEPC/_node.html
- [17] Wikipedia Scalability, Online:
  http://en.wikipedia.org/wiki/Scalability
- [18] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS ’67 (Spring): Proceedings of the April 18-20, 1967, Spring Joint Computer Conference (ACM, New York, NY, USA), 483–485.
- [19] Power! | Copyright GeonX 2013, Geon Technologies, Online:
  http://www.youtube.com/watch?v=nEDOSGC3wFs
- [20] Scalasca Performance Analysis Tool, Online:
  http://www.scalasca.org/
- [21] VAMPIR Performance Analysis Tool, Online:
  http://www.vampir.eu/

Data
Science
Big
Research
science
methods
big
simulations
processing
technologies
analysis
computational
TB
European
computing
Supercomputing
Scientific
Computational
E.g
Centre
climate
Computing
nodes
centers
storage
ENES
EUDAT
Project
HPC
Juelich
datasets
performance
research
databases
brain
increasing
community
data-intensive
Service
Climate
modelling
structures
System
DLCLs
DLCL
access
Hardware
management
distributed
large-scale
services
measurement
within
fusion
new
funding
concepts
Policy-based
cross-disciplinary
resources
forms
year
like
Cross-Disciplinary
EU
device
Analysis
Computer
expertise
disciplines
results
Approaches
construction
Fusion
Enable
Modelling
different
Environment
Key
scientific
important
basis
Infrastructure
files
High
Test
PB
DKRZ
Use
etc
formats
model
international
SMAQ
engineering
References
via
physical
IPCC
Common
Methods
Using
Work
images
often
levels
external
MapReduce
create
GB
codes
use
JSC
Level
stored
Resources
analyze
Large
Summary
Matters
Concrete
system
Landscape
e.g
acquisition
compute
hundreds
Nutshell
Services
network
AR5
Technologies
Earth
directory
cores
Structure
General
current
project
Understanding
Germany
Provide
NASA
Energy
Simulation
systems
day
Processing
Health