

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

LECTURE 5

 @Morris Riedel

 @MorrisRiedel

 @MorrisRiedel

Parallel Algorithms & Data Structures

September 23, 2019

Room V02-156



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



JÜLICH
Forschungszentrum

JÜLICH
SUPERCOMPUTING
CENTRE



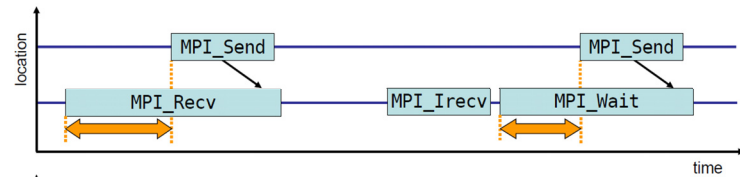
HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES



HELMHOLTZ
ARTIFICIAL INTELLIGENCE
COOPERATION UNIT

Review of Lecture 4 – Advanced MPI Techniques

Selected Options for MPI Communications

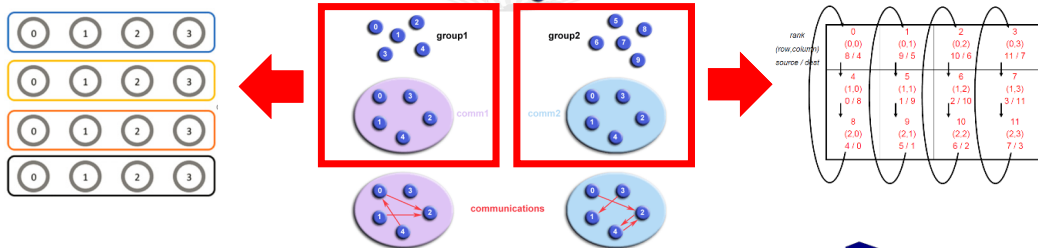


(blocking vs. non-blocking communication)

MPI_Comm_Split() to create sub-group communicator

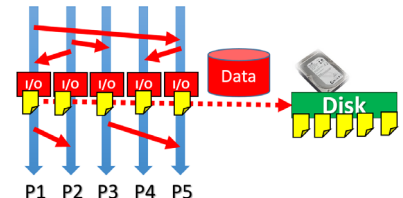
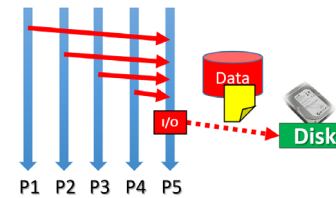


MPI_Cart_create() to create cartesian communicator

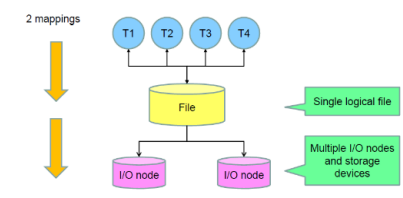
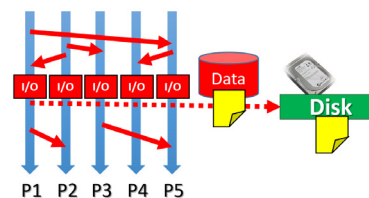
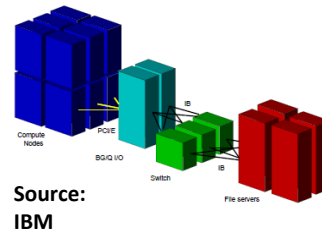
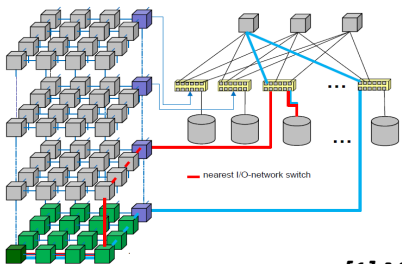


[23] German MPI Lecture

Parallel & Scalable I/O Methods



(how MPI communication is affected by the hardware network impacts application performance)



[1] Metrics tour [2] LLNL MPI Tutorial [3] Introduction to Groups & Communicators [4] HPC Best Practices @ IO Workshop modified from [5] Parallel I/O

Outline of the Course

1. High Performance Computing
2. Parallel Programming with MPI
3. Parallelization Fundamentals
4. Advanced MPI Techniques
5. Parallel Algorithms & Data Structures
6. Parallel Programming with OpenMP
7. Graphical Processing Units (GPUs)
8. Parallel & Scalable Machine & Deep Learning
9. Debugging & Profiling & Performance Toolsets
10. Hybrid Programming & Patterns

11. Scientific Visualization & Scalable Infrastructures
12. Terrestrial Systems & Climate
13. Systems Biology & Bioinformatics
14. Molecular Systems & Libraries
15. Computational Fluid Dynamics & Finite Elements
16. Epilogue

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

- Practical Topics
- Theoretical / Conceptual Topics

Outline

■ Selected Parallel Algorithms

- Vector Addition in MPI using MPI Collectives
- Matrix – Vector Multiplication in MPI using MPI Collectives
- Fast Fourier Transform (FFT) Library Tool using MPI Communicators
- Using Non-Blocking Communication in Simulation Sciences
- Advanced Parallel & Scalable Algorithm Examples in Context

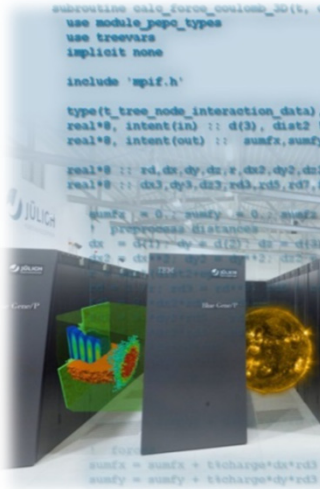
■ Selected Data Structures

- Tree-based Data Structures & Particle Interaction Examples
- Basic MPI Datatypes and Arrays & Multi-dimensional Datasets
- Derived MPI Datatypes & Small Examples
- Relationships to Parallel IO & Hierarchical Data Format (HDF)
- Data Science example using Parallel I/O for 'Big Data' Clustering

- Promises from previous lecture(s):
- *Lecture 4:* Lecture 5 offers more details on using blocking & non-blocking MPI communication in simulations and data science applications
- *Lecture 4:* Lecture 5 offers more details on using Parallel I/O and portable data formats in various simulation sciences & data science applications

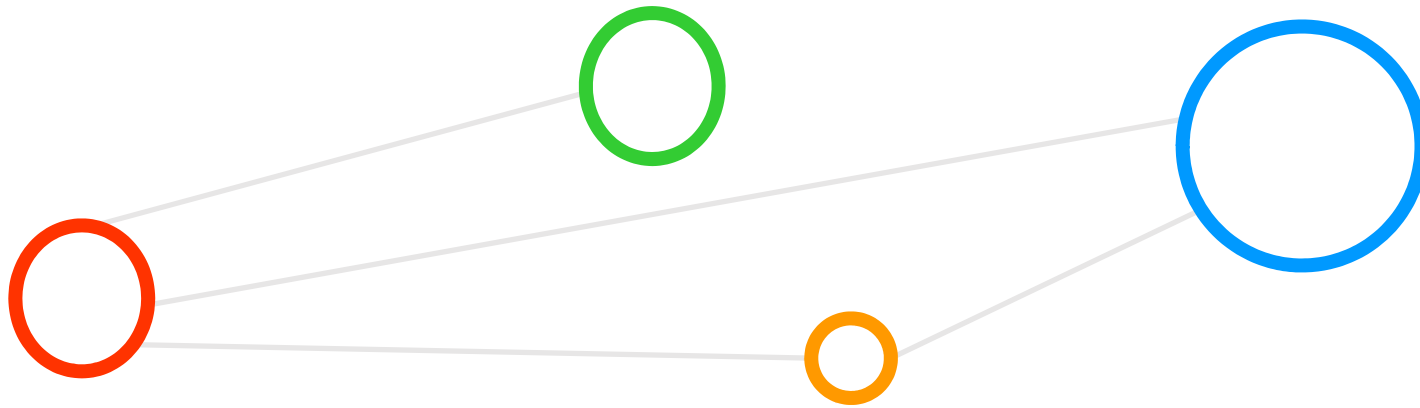


Selected Learning Outcomes

- 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 ...
 - **Program and use HPC programming paradigms**
 - **Take advantage of innovative scientific computing simulations & technology**
 - **Work with technologies and tools to handle parallelism complexity**
- 

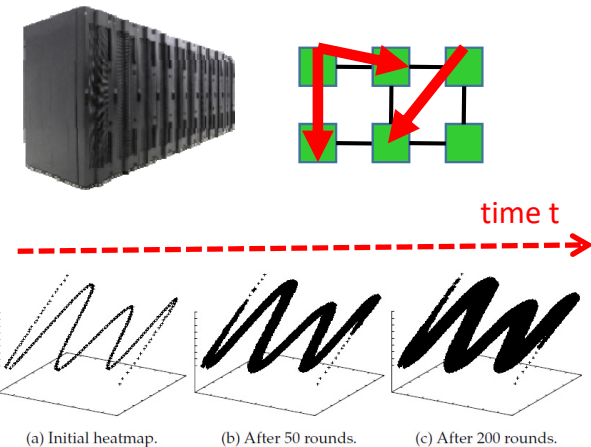
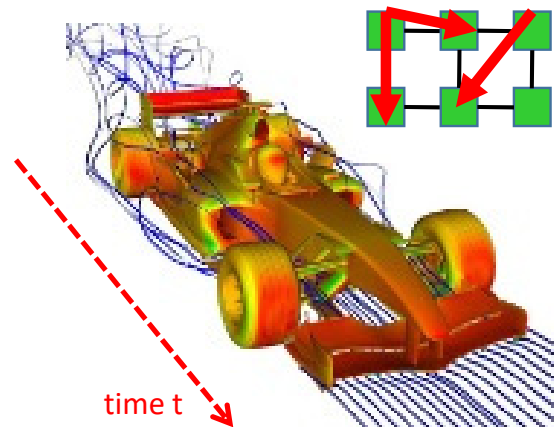
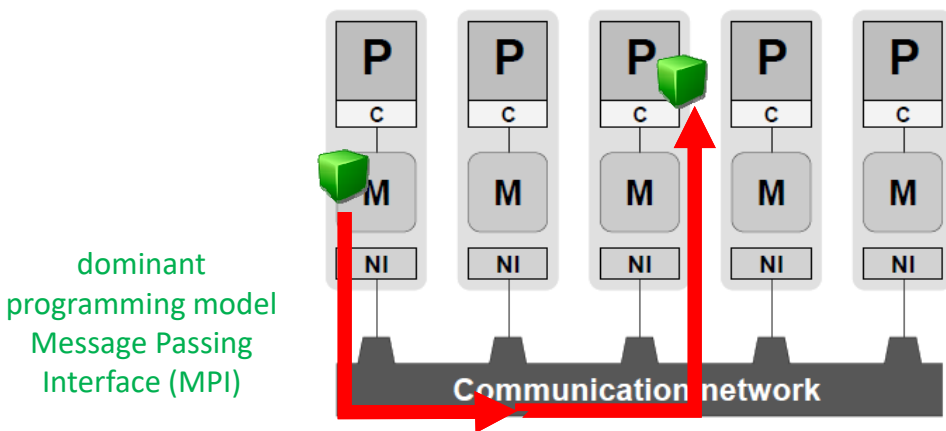


Selected Parallel Algorithms



Distributed-Memory Computers – Revisited (cf. Lecture 1)

- A distributed-memory parallel computer establishes a 'system view' where no process can access another process' memory directly



■ Features

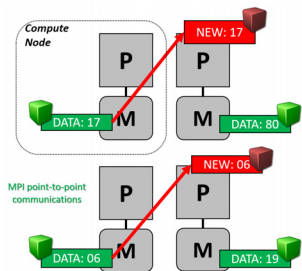
- Processors communicate via **Network Interfaces (NI)**
- NI mediates the connection to a **Communication network**
- This setup is rarely used → a programming model view today

[12] Modified from
Caterham F1 team

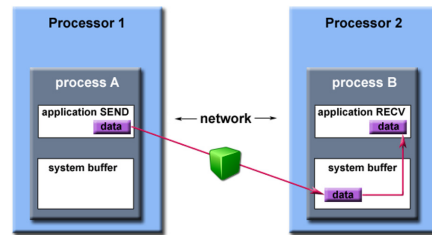
[13] Introduction to High Performance Computing
for Scientists and Engineers

Parallel Programming with MPI & Basic Building Blocks (cf. Lecture 2)

- Message Passing Interface (MPI) Concepts
- MPI Parallel Programming Basics



MPI
Point
to
Point
Communication



C
hello.c

using a C compiler
using a job script

Scheduler



```
#include <stdio.h>
#include <mpi.h>

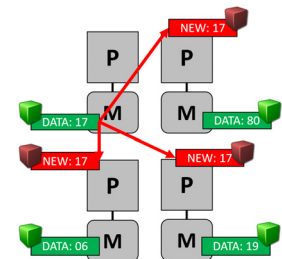
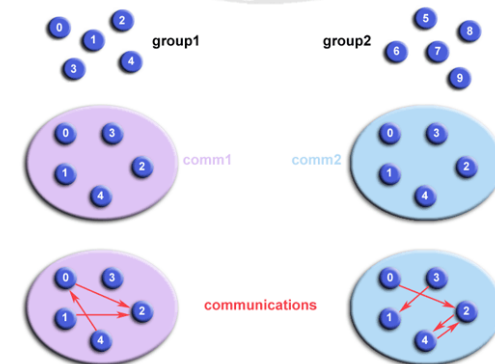
int main(int argc, char** argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

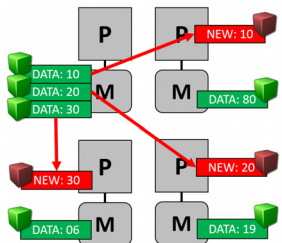
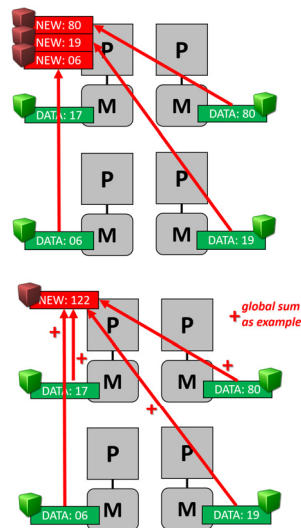
    printf("Hello World, I am %d out of %d\n",
           rank, size);

    MPI_Finalize();

    return 0;
}
```



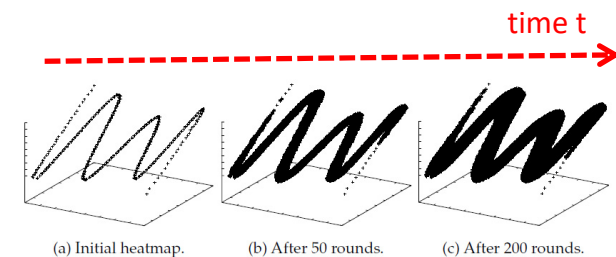
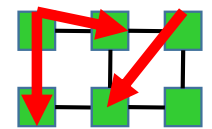
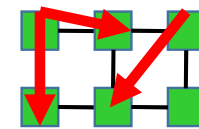
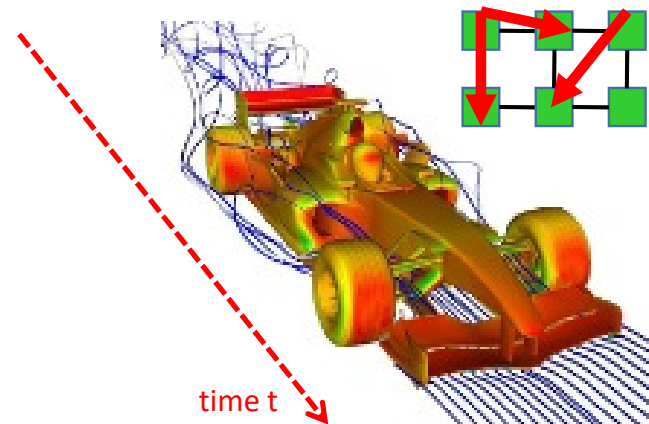
MPI
Collective
Communication



[2] LLNL MPI Tutorial

Formula Race Car Design & Room Heat Dissipation – Revisited (cf. Lecture 3)

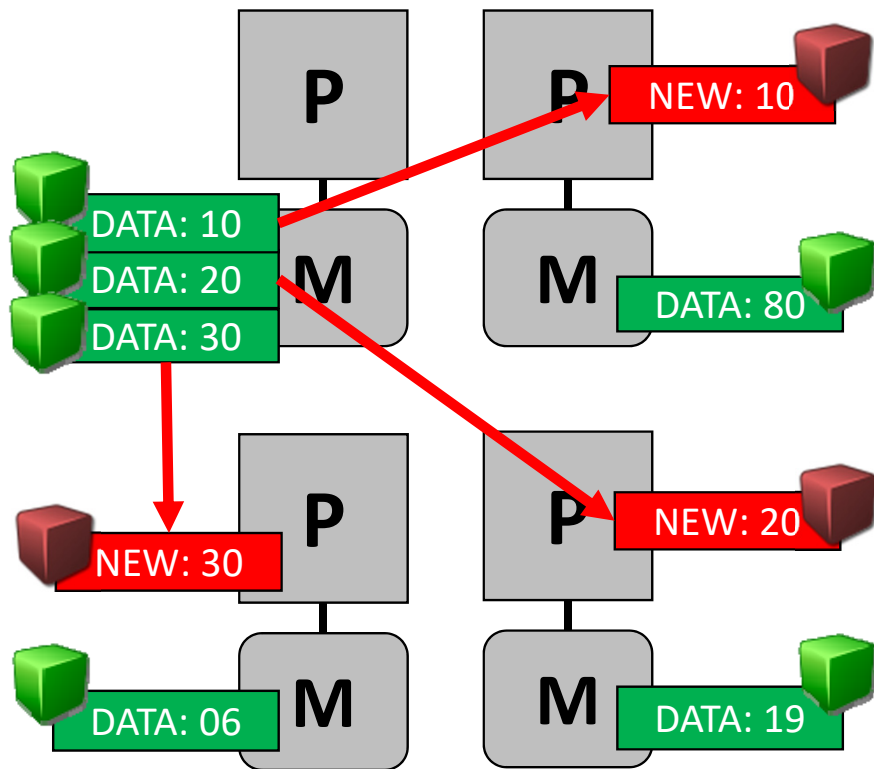
- Pro: Network communication is relatively hidden and supported
 - Contra: Programming with MPI still requires using ‘parallelization methods’
 - Not easy: Write ‘technical code’ well integrated in ‘problem-domain code’
- Example: Race Car Simulation & Heat dissipation in a Room
 - Apply a good parallelization 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)



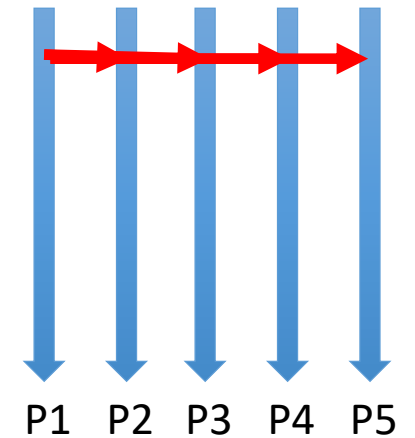
[6] Modified from
Caterham F1 team

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

Collective Functions: Scatter (one-to-many) – Parallel Algorithm Example

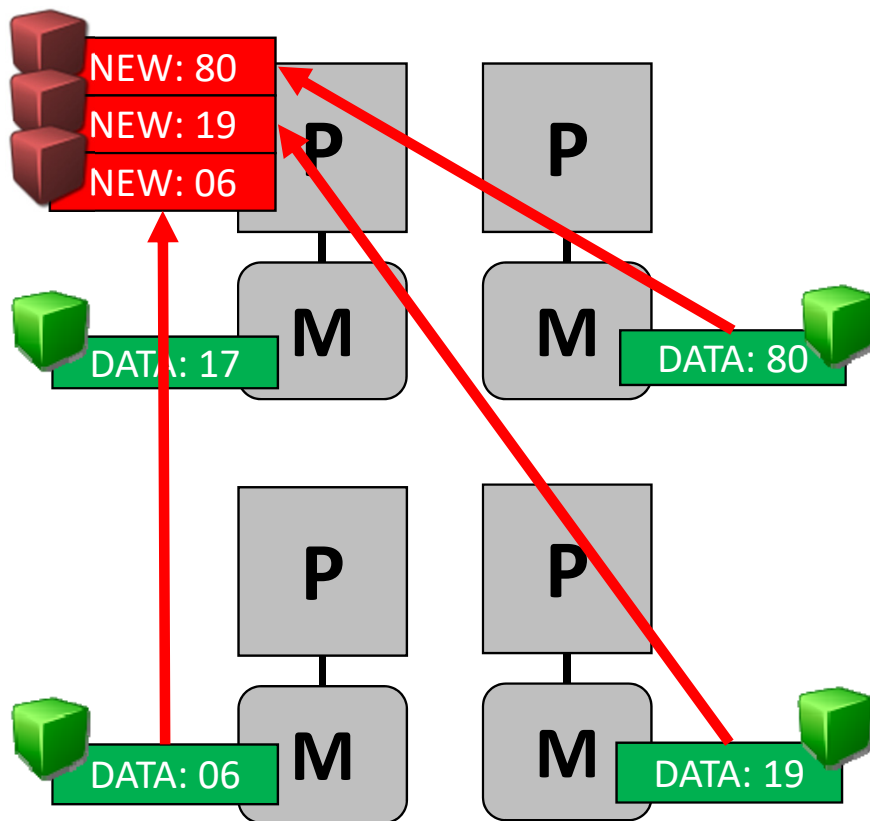


$$Z = X + Y$$

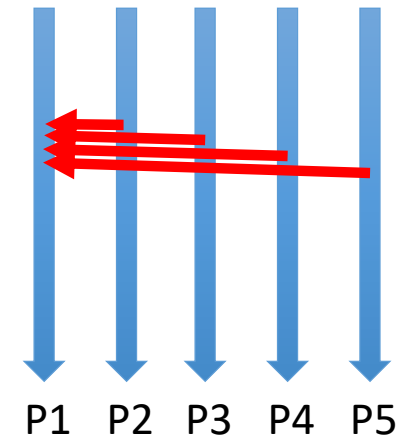


- Scatter distributes different data to many or even all other processors

Collective Functions: Gather (many-to-one) – Parallel Algorithm Example



$$Z = X + Y$$



- Gather collects data from many or even all other processors to one specific processor

Vector Addition in MPI using MPI Collectives

```
#include <mpi.h>

#define CHUNK = 1 /* simple example */

int main(int argc, char **argv) {
    int rank, size, n, i;
    double x[3], y[3], z[3];
    double xpart[CHUNK], ypart[CHUNK], zpart[CHUNK];

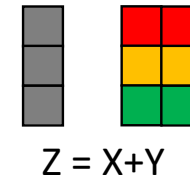
    n = atoi(argv[1]); /* Get input size */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* ...x,y,buff preparations with real values for vectors, e.g. via rank 0 ... */
    MPI_Scatter(x,CHUNK,MPI_DOUBLE,xpart,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);
    MPI_Scatter(y,CHUNK,MPI_DOUBLE,ypart,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);

    for (i=0; i<CHUNK; i++)
        zpart[i] = xpart[i] + ypart[i];
    /* Collect the result */
    MPI_Gather(zpart,CHUNK,MPI_DOUBLE,z,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);
    MPI_Finalize();

    return 0;
}
```

*'simplified
demo code'*



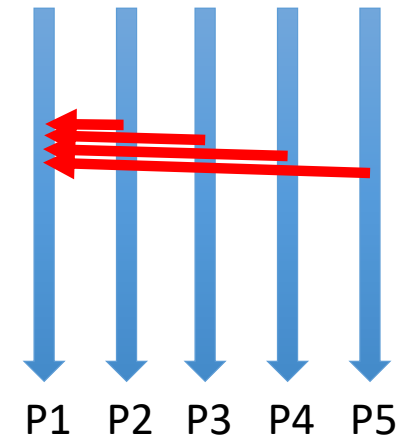
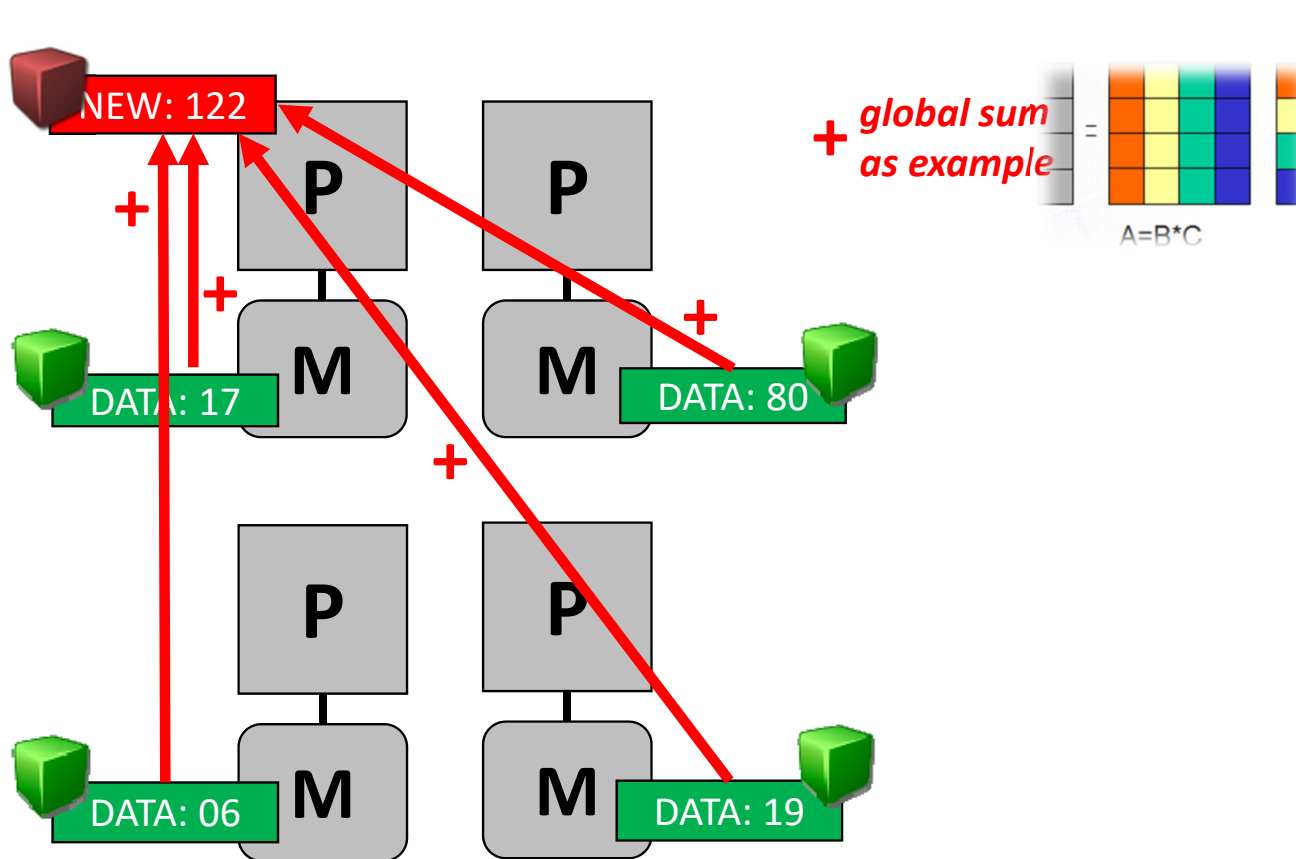
- Scatter distributes different data (x,y) to many or even all other processors in the communicator

- Gather collects data from many or even all other processors to one specific processor in the communicator

```
int MPI_Scatter(
    void *sendbuf,
    int sendcnt,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcnt,
    MPI_Datatype recvtype,
    int root,
    MPI_Comm comm
);
```

```
int MPI_Gather(
    void *sendbuf,
    int sendcnt,
    MPI_Datatype sendtype,
    void *recvbuf,
    int recvcnt,
    MPI_Datatype recvtype,
    int root,
    MPI_Comm comm
);
```

Collective Functions: Reduce (many-to-one) – Parallel Algorithm Example



- Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a global minimum or maximum, sum, or product of the different data located at different processors

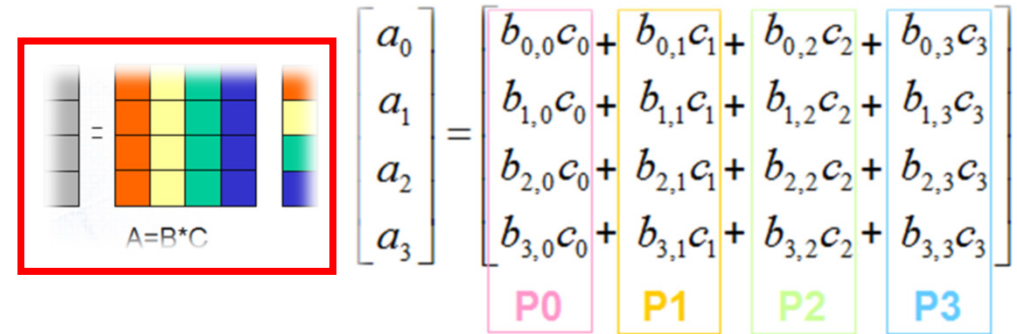
Matrix-Vector Multiplication in MPI using MPI Collectives – Required Variables

```
#include <stdio.h>
#include <mpi.h>
#define NCOLS = 4 /* matrix columnwise domain decomposition */
int main(int argc, char **argv) {
    int i,j,k,l, rank, size;

    float A[NCOLS];
    float Apart[NCOLS];
    float Bpart[NCOLS];
    float C[NCOLS];
    float A_exact[NCOLS];
    float B[NCOLS][NCOLS];
    float Cpart[1];

    root = 0;

    /* preparing MPI environment and initialization of matrix */
    ...
}
```



$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + b_{0,1}c_1 + b_{0,2}c_2 + b_{0,3}c_3 \\ b_{1,0}c_0 + b_{1,1}c_1 + b_{1,2}c_2 + b_{1,3}c_3 \\ b_{2,0}c_0 + b_{2,1}c_1 + b_{2,2}c_2 + b_{2,3}c_3 \\ b_{3,0}c_0 + b_{3,1}c_1 + b_{3,2}c_2 + b_{3,3}c_3 \end{bmatrix}$$

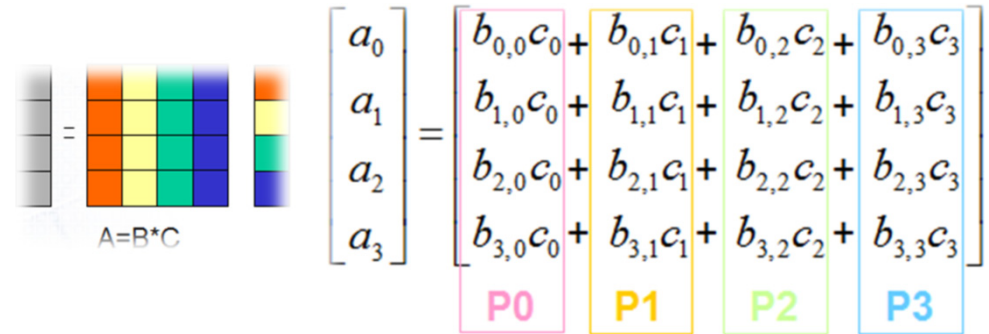
P0 P1 P2 P3

*'simplified
demo code'*

[14] Parallel Algorithms

Matrix-Vector Multiplication in MPI using MPI Collectives – MPI Setup

```
#include <stdio.h>
#include <mpi.h>
#define NCOLS = 4 /* matrix columnwise domain decomposition */
int main(int argc, char **argv) {
    ...
    /* required variables are defined */
    ...
    /* preparing MPI environment and initialization of matrix */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) { /* initialize matrix B */
        B[0][0] = 1; B[0][1] = 2; B[0][2] = 3; B[0][3] = 4; B[1][0] = 4; B[1][1] = -5; B[1][2] = 6;
        B[1][3] = 4; B[2][0] = 7; B[2][1] = 8; B[2][2] = 9; B[2][3] = 2; B[3][0] = 3; B[3][1] = -1;
        B[3][2] = 5; B[3][3] = 0;
        /* initialize vector C */
        C[0] = 1; C[1] = -4; C[2] = 7; C[3] = 3;
    } /* Program continues with parallel calculations using collectives */
    ...
}
```



$$A = B \cdot C$$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + b_{0,1}c_1 + b_{0,2}c_2 + b_{0,3}c_3 \\ b_{1,0}c_0 + b_{1,1}c_1 + b_{1,2}c_2 + b_{1,3}c_3 \\ b_{2,0}c_0 + b_{2,1}c_1 + b_{2,2}c_2 + b_{2,3}c_3 \\ b_{3,0}c_0 + b_{3,1}c_1 + b_{3,2}c_2 + b_{3,3}c_3 \end{bmatrix}$$

P0 P1 P2 P3

*'simplified
demo code'*

[14] Parallel Algorithms

Matrix-Vector Multiplication in MPI using MPI Collectives – Scatter Data

```
#include <stdio.h>
#include <mpi.h>
#define NCOLS = 4 /* matrix columnwise domain decomposition */
int main(int argc, char **argv) { ...
    /* Program continues with parallel calculations using collectives */
    /* Put up a barrier until I/O is complete */
    MPI_Barrier(MPI_COMM_WORLD);
    /* Scatter matrix B */
    MPI_Scatter(B,NCOLS,MPI_FLOAT,Bpart,NCOLS,MPI_FLOAT,0,MPI_COMM_WORLD);
    /* Scatter vector C */
    MPI_Scatter(C,1,MPI_FLOAT,Cpart,1,MPI_FLOAT, 0,MPI_COMM_WORLD);
    /* Do the vector-scalar multiplication */
    for(j=0; j < NCOLS; j++)
        Apart[j] = Cpart[0] * Bpart[j];
    /* Reduce to matrix A */
    MPI_Reduce(Apart,A,NCOLS,MPI_FLOAT,MPI_SUM, 0,MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + b_{0,1}c_1 + b_{0,2}c_2 + b_{0,3}c_3 \\ b_{1,0}c_0 + b_{1,1}c_1 + b_{1,2}c_2 + b_{1,3}c_3 \\ b_{2,0}c_0 + b_{2,1}c_1 + b_{2,2}c_2 + b_{2,3}c_3 \\ b_{3,0}c_0 + b_{3,1}c_1 + b_{3,2}c_2 + b_{3,3}c_3 \end{bmatrix}$$

P0 P1 P2 P3

- Barrier enables a synchronization among all processors and blocks until all processors reach this line of code

- Scatter distributes different data (x,y) to many or even all other processors in the communicator
- Each processor has a column of matrix B (named as Bpart)
- Each processor has an element of column vector C (named Cpart)

'simplified demo code'

[14] Parallel Algorithms

Matrix-Vector Multiplication in MPI using MPI Collectives – Reduce Results

```
#include <stdio.h>
#include <mpi.h>

#define NCOLS = 4 /* matrix columnwise domain decomposition */

int main(int argc, char **argv) { ...

    /* Program continues with parallel calculations using collectives */

    /* Put up a barrier until I/O is complete */
    MPI_Barrier(MPI_COMM_WORLD);

    /* Scatter matrix B */
    MPI_Scatter(B, NCOLS, MPI_FLOAT, Bpart, NCOLS, MPI_FLOAT, 0, MPI_COMM_WORLD);

    /* Scatter vector C */
    MPI_Scatter(C, 1, MPI_FLOAT, Cpart, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);

    /* Do the vector-scalar multiplication */
    for(j=0; j < NCOLS; j++)
        Apart[j] = Cpart[0] * Bpart[j];

    /* Reduce to matrix A */
    MPI_Reduce(Apart, A, NCOLS, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Finalize();

    return 0;

}
```

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + b_{0,1}c_1 + b_{0,2}c_2 + b_{0,3}c_3 \\ b_{1,0}c_0 + b_{1,1}c_1 + b_{1,2}c_2 + b_{1,3}c_3 \\ b_{2,0}c_0 + b_{2,1}c_1 + b_{2,2}c_2 + b_{2,3}c_3 \\ b_{3,0}c_0 + b_{3,1}c_1 + b_{3,2}c_2 + b_{3,3}c_3 \end{bmatrix}$$

P0 P1 P2 P3

- Each processor performs an independent vector-scalar multiplication (based on their Bpart and Cpart contents)

- Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a global minimum or maximum, sum, or product of the different data located at different processors
- Each processor has a part of the result vector A (named Apart) and is reduced on rank 0 as sum

'simplified demo code'

[14] Parallel Algorithms

Fast Fourier Transform (FFT) – Algorithm & Parallel Library Applications

- Example Applications
 - Digital signal processing and solving Partial Differential Equations (PDE)
 - Algorithms for quick multiplication of large integers
 - **Fourier series → study of periodic phenomena...**
- **Discrete Fourier Transform (DFT) – $O(N^2)$**
 - Obtained by decomposing a sequence of values into components of different frequencies (... analysis of non-periodic phenomena...)
 - Computing it directly from the mathematical definition is often too slow to be practical
- **Fast Fourier Transform (FFT) – $O(N \log N)$**
 - FFT is a way to compute the DFT and its inverse, but more quickly
- **Tool Fast Fourier Transform in the West (FFTW)**
 - Multi-threaded C subroutine library with fortran interface (free software)
 - FFTW versions include parallel transforms (shared & distributed memory)

[16] *FFTW Manual*

FFTW Library Tool – Parallel Algorithm with MPI Example

```
#include <fftw3-mpi.h>
```

```
int main(int argc, char **argv) {  
    const ptrdiff_t N0 = ..., N1 = ...;  
    fftw_plan plan;  
    fftw_complex *data;  
    ptrdiff_t alloc_local, local_n0, local_0_start, i, j;  
    MPI_Init(&argc, &argv);  
    fftw_mpi_init();  
    /* get local data size and allocate */  
    alloc_local = fftw_mpi_local_size_2d (N0, N1, MPI_COMM_WORLD, &local_n0, &local_0_start);  
    data = fftw_alloc_complex(alloc_local);  
    /* create plan for in-place forward DFT */  
    plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD, FFTW_FORWARD, FFTW_ESTIMATE);  
    /* initialize data to some function my_function(x,y) */  
    for (i = 0; i < local_n0; ++i)  
        for (j = 0; j < N1; ++j)  
            data[i*N1 + j] = my_function(local_0_start + i, j);  
    /* compute transforms, in-place, as many times as desired */  
    fftw_execute(plan);  
    fftw_destroy_plan(plan);  
    MPI_Finalize();  
    return 0; }  
}
```

- Using a library tool like here for the Fast Fourier Transform (FFT) could mean that MPI messages are mostly abstracted away and only communication elements remain

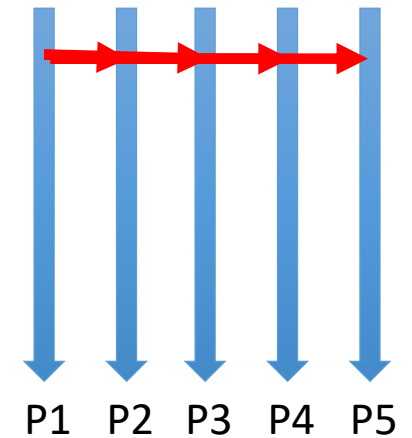
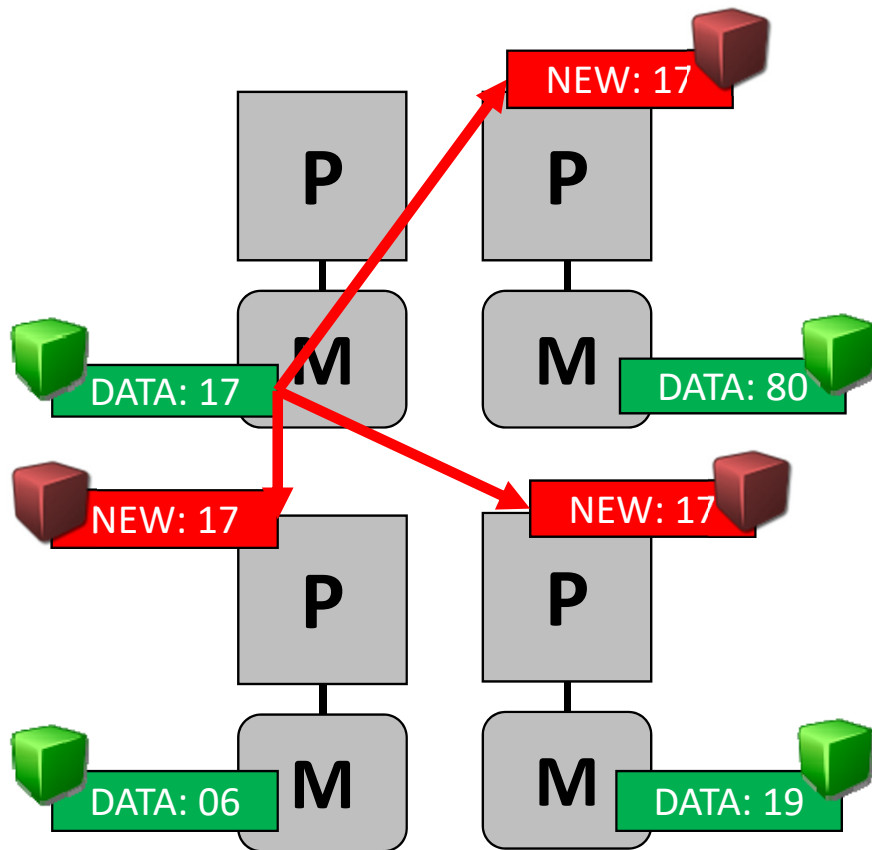
- Does not allocate the entire 2 dimensional array on each process, instead function `fftw_mpi_local_size_2d` finds out what *portion* of the array resides on each processor and this is used to know how much space to allocate
- *1d block distribution* of the data (cf. Lecture 2), distributed along the first dimension; e.g. 100 × 200 complex DFT, distributed over 4 processes: 25 × 200 slices / process of the data, 0 : 0 - 24, 1: 25 – 49 , 2 : 50 – 74, 3 : 75 – 99

- Communicator `MPI_COMM_WORLD` indicates here which processes will participate in the transform

*'simplified
demo code'*

[16] FFTW Manual

Collective Functions : Broadcast (one-to-many) – Parallel Algorithm Example



- Broadcast distributes the same data to many or even all other processors

Computational Steering of (Iterative) Parallel Algorithms using MPI


■ Particle Simulations using PEPC library (see above)

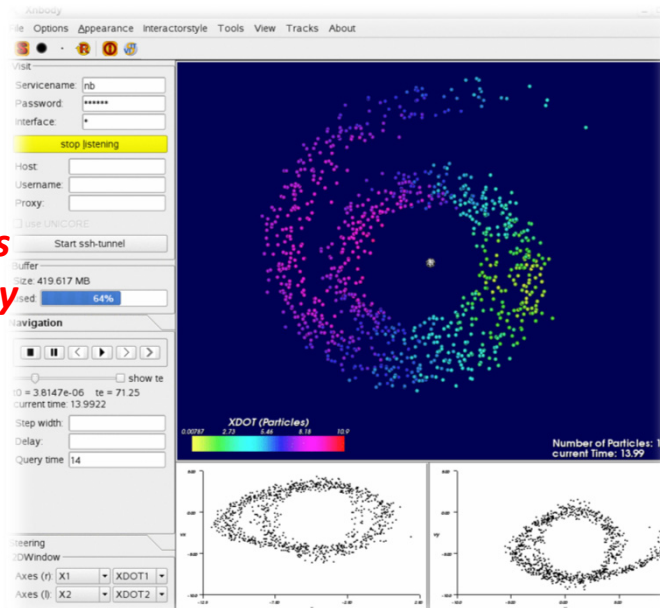
- E.g. research star cluster dynamics in astrophysics or particle acceleration simulations via laser pulses
- E.g. Iterations over time using **nbody6++ parallel algorithm**
- Steering: changing parameters during the run-time of simulation

```
call flvisit_nbody2_steering_recv(  
& VISITDPARM1,VISITDPARM2,VISITDPARM3,  
& VISITDPARM4,VISITIPARM1,VISITIPARM2,...)  
...  
if(LVISIT_ACTIVE.eq.1) Then  
  VDISTANCE=VISITDPARM4  
  write(*,*) 'VISCON: VDISTANCE=',VDISTANCE  
endif  
...  
IF(VISITDPARM2.gt.0) THEN  
  DTADJ = VISITDPARM2  
END IF  
IF(VISITDPARM3.gt.0) THEN  
  DELTAT = VISITDPARM3  
END IF
```

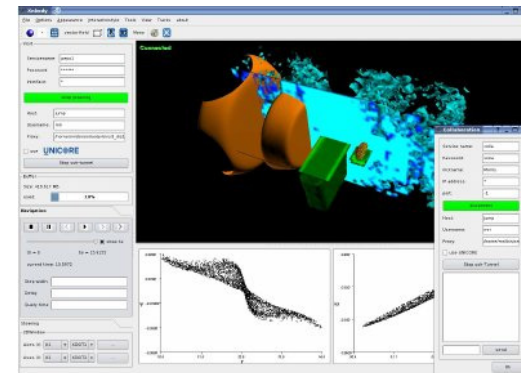
```
CALL MPI_BCAST(DTADJ,1,MPI_DOUBLE_PRECISION,  
0,& MPI_COMM_WORLD,ierr)  
CALL MPI_BCAST(VISITDPARM3,1,  
MPI_DOUBLE_PRECISION,0,& MPI_COMM_WORLD,...)
```


**change
parameters
interactively**


**visualize
status**

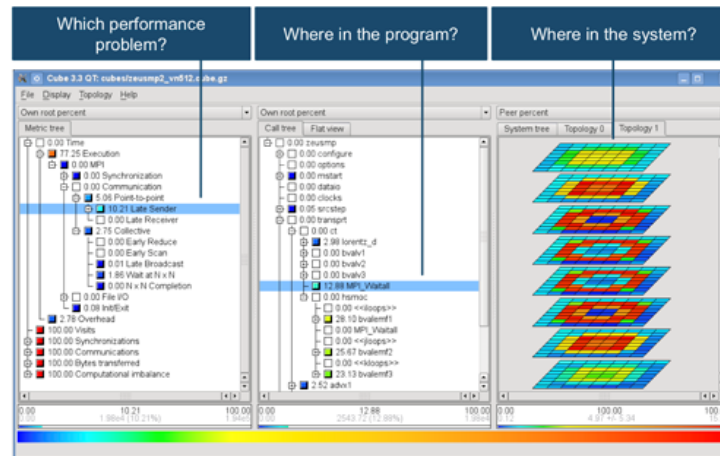


[17] M. Riedel et al.,
computational steering, 2007



Performance Analysis is a Key Field in HPC – Revisited

- 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



[21] SCALASCA Performance Tool

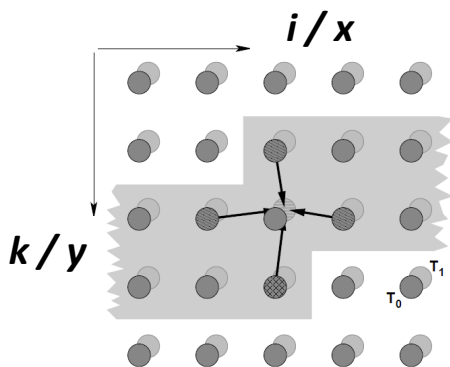
➤ Lecture 9 will give details on how to measure performance in parallel programmes & and related tools using various applications

Data Parallelism: Formulas Across Domain Decomposition

- From the problem to computational data structures

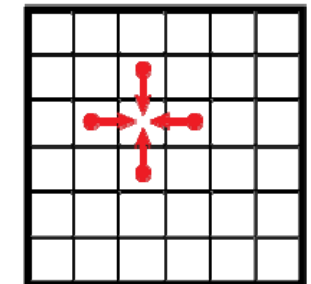
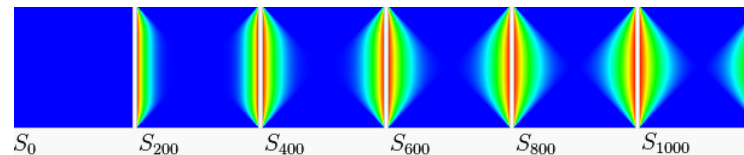
- Apply an 'isotropic lattice' technique

```
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
```



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

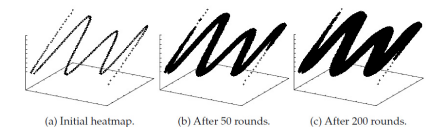
$$\frac{\partial \Phi}{\partial t} = \Delta \Phi$$



Modified from [13] *Introduction to High Performance Computing for Scientists and Engineers*

[20] Wikipedia on 'stencil code'

'change over time'
diffusion equation



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

Large-scale Computing Infrastructures & Course-Grained Parallel Algorithms

- Large computing systems are often embedded in infrastructures
 - Grid computing for **distributed data storage and processing** via middleware
 - The success of Grid computing was renowned when being mentioned by Prof. Rolf-Dieter Heuer, CERN Director General, in the context of the Higgs Boson Discovery:
- Other large-scale distributed infrastructures exist
 - Partnership for Advanced Computing in Europe (PRACE) → EU HPC
 - Extreme Engineering and Discovery Environment (XSEDE) → US HPC

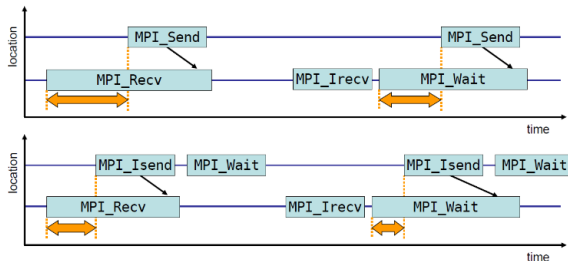
■ 'Results today only possible due to extraordinary performance of Accelerators – Experiments – Grid computing'



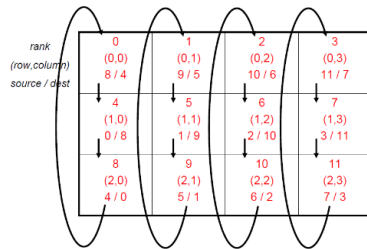
[18] Grid Computing Video

➤ Lecture 11 will give in-depth details on scalable approaches in large-scale HPC infrastructures and how to use them with middleware

Blocking vs. Non-blocking communication – Parallel Algorithms Example



[1] Metrics tour



[23] German MPI Lecture

```
// do some work with MPI communication operations...
// e.g. exchanging simple data with all neighbours

outbuf = rank;

for (i=0; i<4;i++) {
    dest=nbrs[i];
    source=nbrs[i];

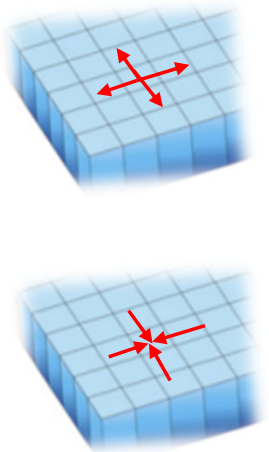
    // perform non-blocking communication
    MPI_Isend(&outbuf, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &reqs[i]);
    MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag, MPI_COMM_WORLD, &reqs[i+4]); // 4 as a kind of offset
}

// wait for non-blocking communication to be completed for output
MPI_Waitall(8, reqs, stats);

printf("rank= %d has received (u,d,l,r)= %d %d %d %d \n", rank,
        inbuf[UP], inbuf[DOWN], inbuf[LEFT], inbuf[RIGHT] );

MPI_Finalize();

return 0;
```



- **Blocking vs. non-blocking:** MPI_Send() blocks until data is received; MPI_Isend() continues
- The use of these functions can cause different performance problems (e.g. here 'late sender')
- MPI_Wait() does wait for a given MPI request to complete before continuing
- MPI_Waitall() does wait for all given MPI requests (e.g. waiting for message) to complete before continuing

MPI_Waitall

Waits for all given MPI Requests to complete

Synopsis

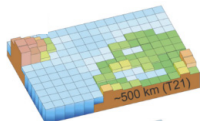
```
int MPI_Waitall(int count, MPI_Request array_of_requests[],
               MPI_Status array_of_statuses[])
```

Input Parameters

count
list length (integer)
array_of_requests
array of request handles (array of handles)

Output Parameters

array_of_statuses
array of status objects (array of Statuses). May be MPI_STATUSES_IGNORE.

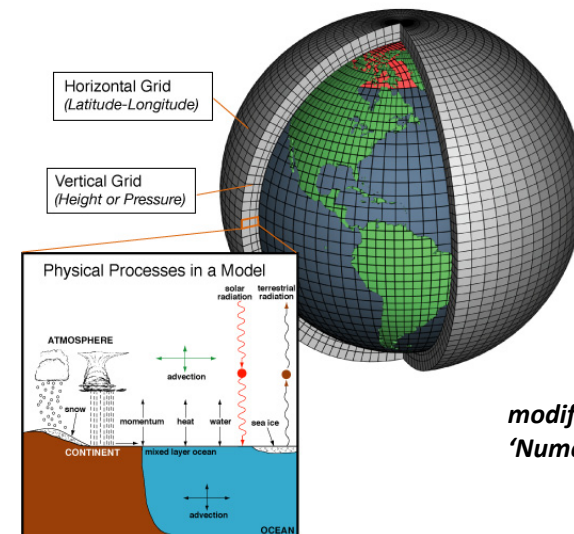


➤ Lecture 12 will provide more details on using blocking vs non-blocking communication in terrestrial systems & HPC climate simulations

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 atmospheric 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 [19] Wikipedia on 'Numerical Weather Prediction'

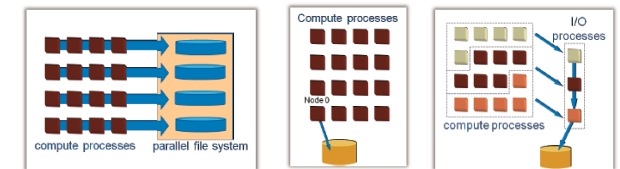
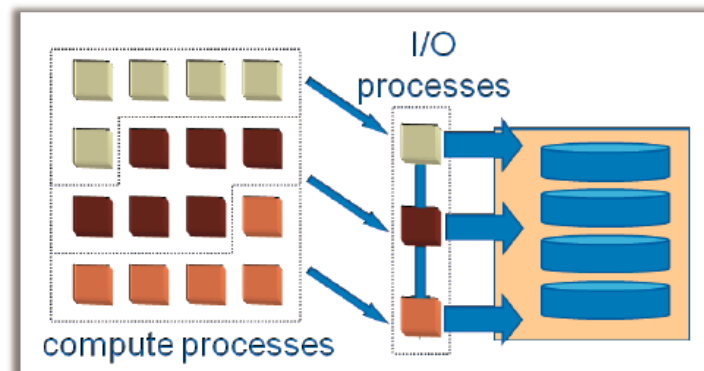
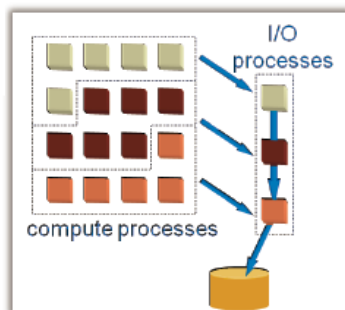
➤ Lecture 12 will provide more details on using different domain decompositions for terrestrial systems and climate simulations on HPC

Climate – WRF Model Parallel Application – pNetCDF

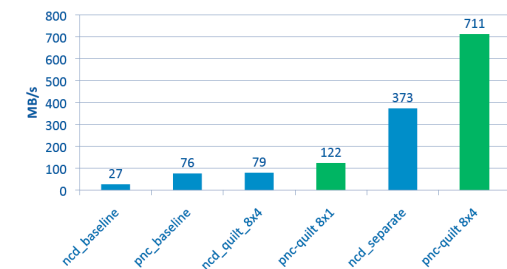
- Need for Parallel I/O (cf. Lecture 4)
 - WRF is output-bound ('writes costs much')
- Use Serial & parallel NetCDF
 - Provides an I/O layer implemented with parallel NetCDF (pNetCDF)
 - I/O performance gain is considerable against using not pNetCDF

- Parallel Network Common Data Form (NETCDF) is designed to store & organize array-oriented data
- Portable data formats are needed to efficiently process data in heterogeneous HPC environments
- Parallel NetCDF can be used to significantly improve I/O output performance of WRF codes

Serial NetCDF collected and written by gangs of MPI tasks (quilting)
Parallel NetCDF written to single files by all MPI tasks in a gang



(different options that do not scale)



[22] Opportunities for WRF Model Acceleration

Monte Carlo Parallel Algorithms

■ Scientific case:

- Understanding protein folding in computational biophysics for an increased understanding of human body
- Proteins perform functions within living organisms (e.g. respond to stimuli)
- Proteins differ in their sequence of amino acids, results in different foldings
- Correct and unique 3D structure is essential to the functions of proteins
- Process of protein folding as a parallel computing application



■ Using Monte Carlo simulations

- Simulations that use stochastic methods to generate new configurations of a system
- Initial conditions of particles, then Monte Carlo 'moves' that changes configuration of particles

➤ Lecture 13 will provide more details on using different & scalable parallel algorithms for systems biology & bioinformatics applications

- Scientific case:

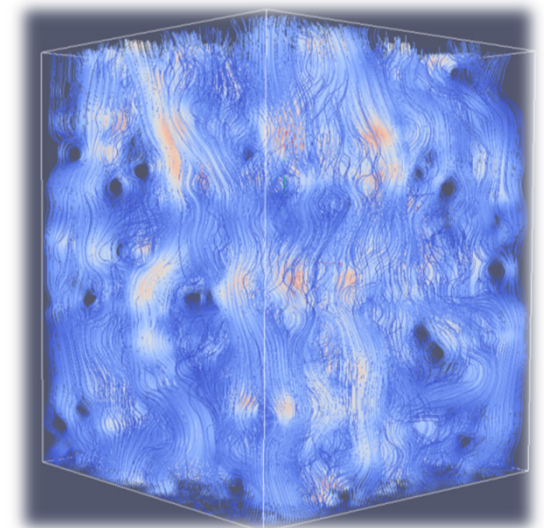
- Understanding physical movements of atoms and molecules in the context of n-body simulations

- Molecular dynamics algorithms for interacting ‘particles’

- Determine trajectories of atoms and molecules
- Numerically solving the Newton’s equations of motion
- Forces between particles and potential energy is parallel computed according to molecular mechanics force field methods

- Using a library

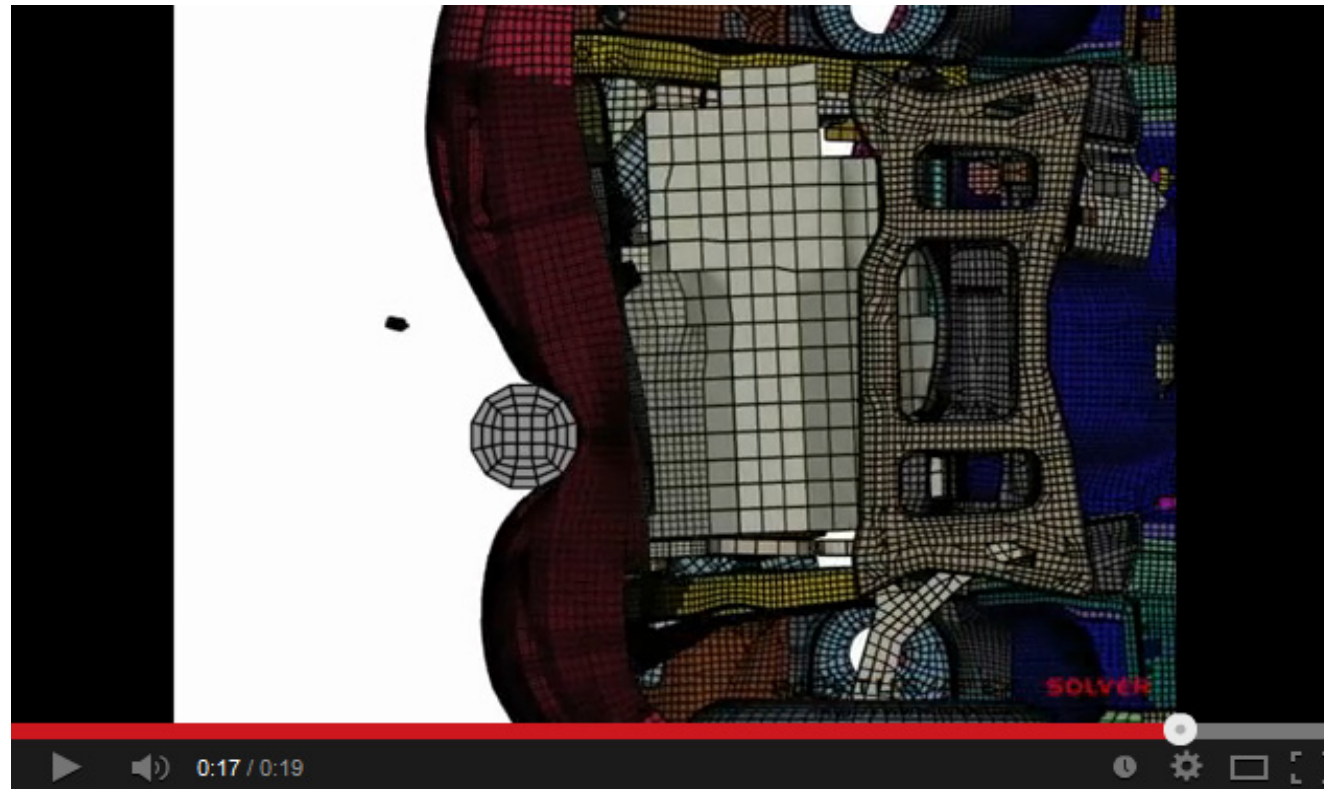
- E.g. [MP2C code](#): particle-based hydrodynamics (fluid simulations)



Flow field in a gas diffusion membrane

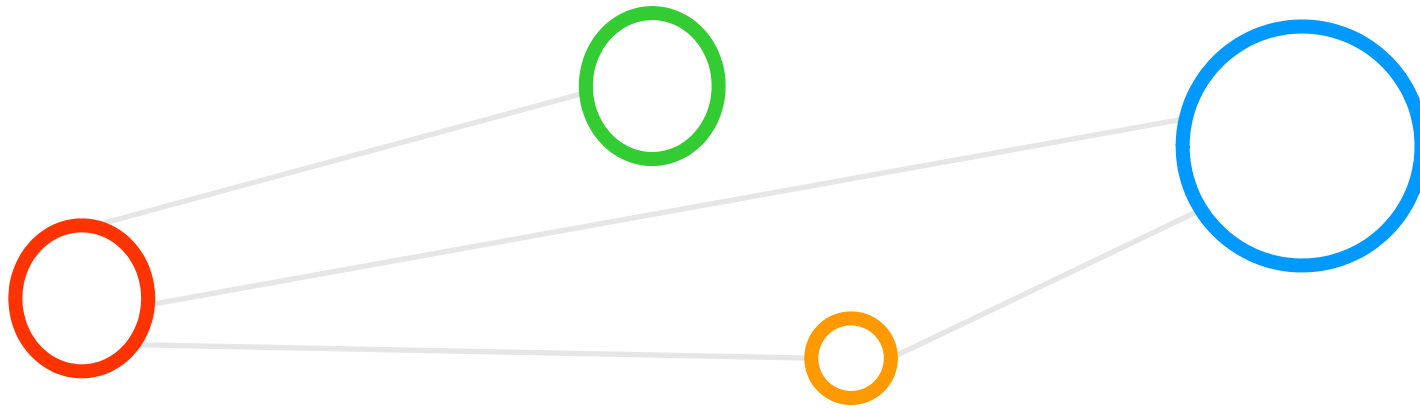
➤ Lecture 14 will give in-depth details on parallel and scalable molecular systems algorithms, tools, methods, and the use of libraries

[Video] Finite Element Simulation Example in Product Engineering



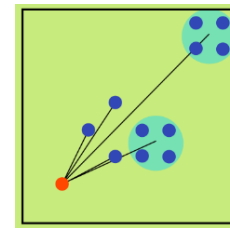
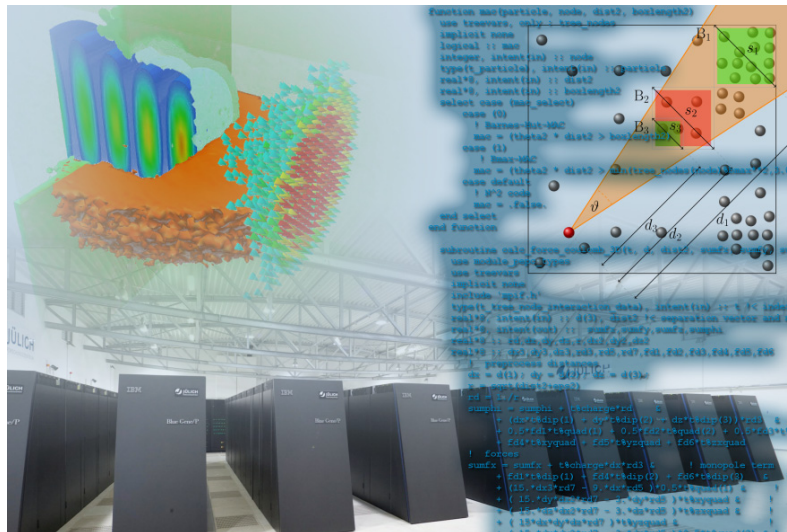
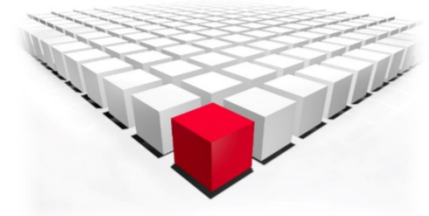
[15] Finite element simulation of full scale car crash

Selected Data Structures

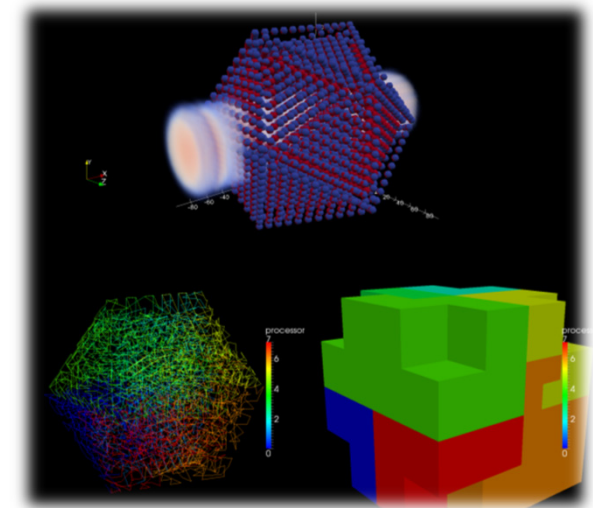


Tree-Code Parallel Algorithms – Example N-body & Particle Simulations

- **Tree codes** – ‘another form of smart domain decomposition’
 - E.g. to speed up ‘**N-body simulations**’ with long range interactions
 - Enable realistic simulations of n-body systems **with increasing particles**
 - Offers the **understanding why data structures** & domain decomposition are important to be jointly considered in parallel algorithms



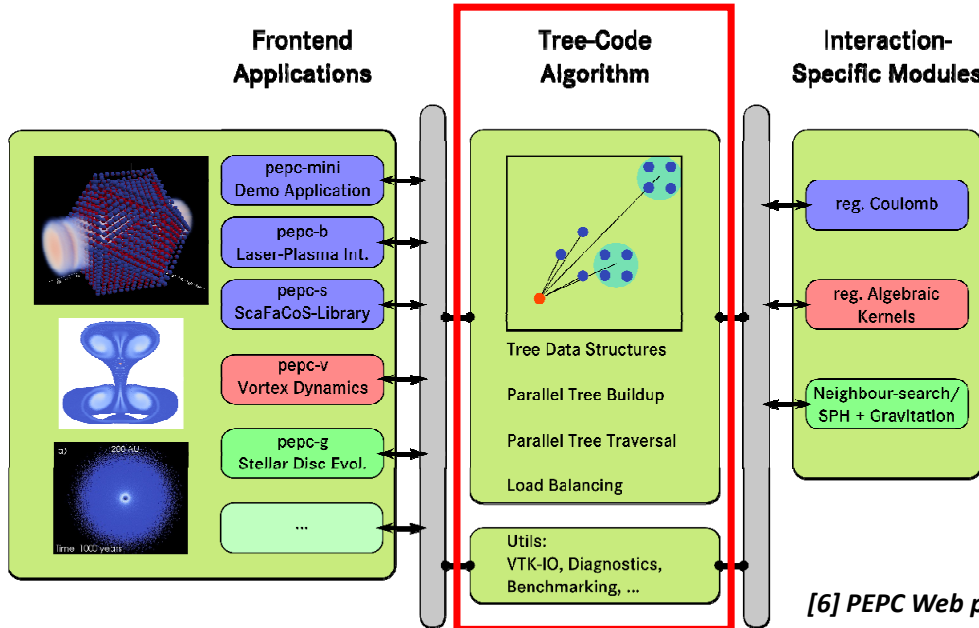
[6] PEPC Web page



Tree-Code Parallel Algorithms – Reduce Number of Required Particle Interactions

- Pretty Efficient Parallel Coulomb (PEPC) Solver

- Implementation of classical ‘Barnes-Hut Tree Code’ for N-body problems
- Divides ‘simulation space’ into cubic cells as ‘octree’ to reduce computing
- Particles in nearby cells are treated individually
- Particles in distant cells are treated as a single large particle

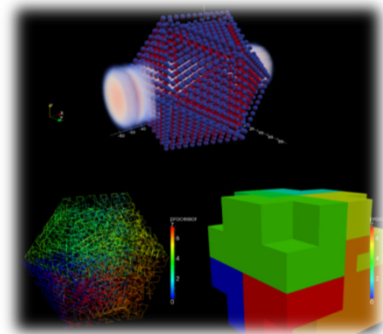
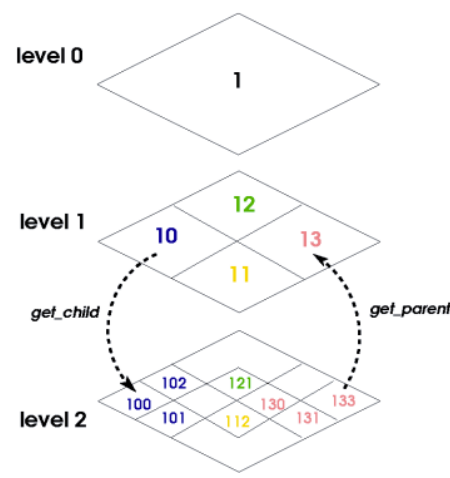


[6] PEPC Web page

■ In physics-based simulation science applications a tree-code parallel algorithm can significantly reduce the number of particle pair interactions that must be computed

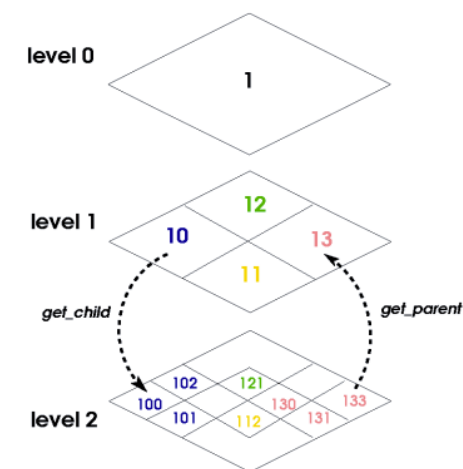
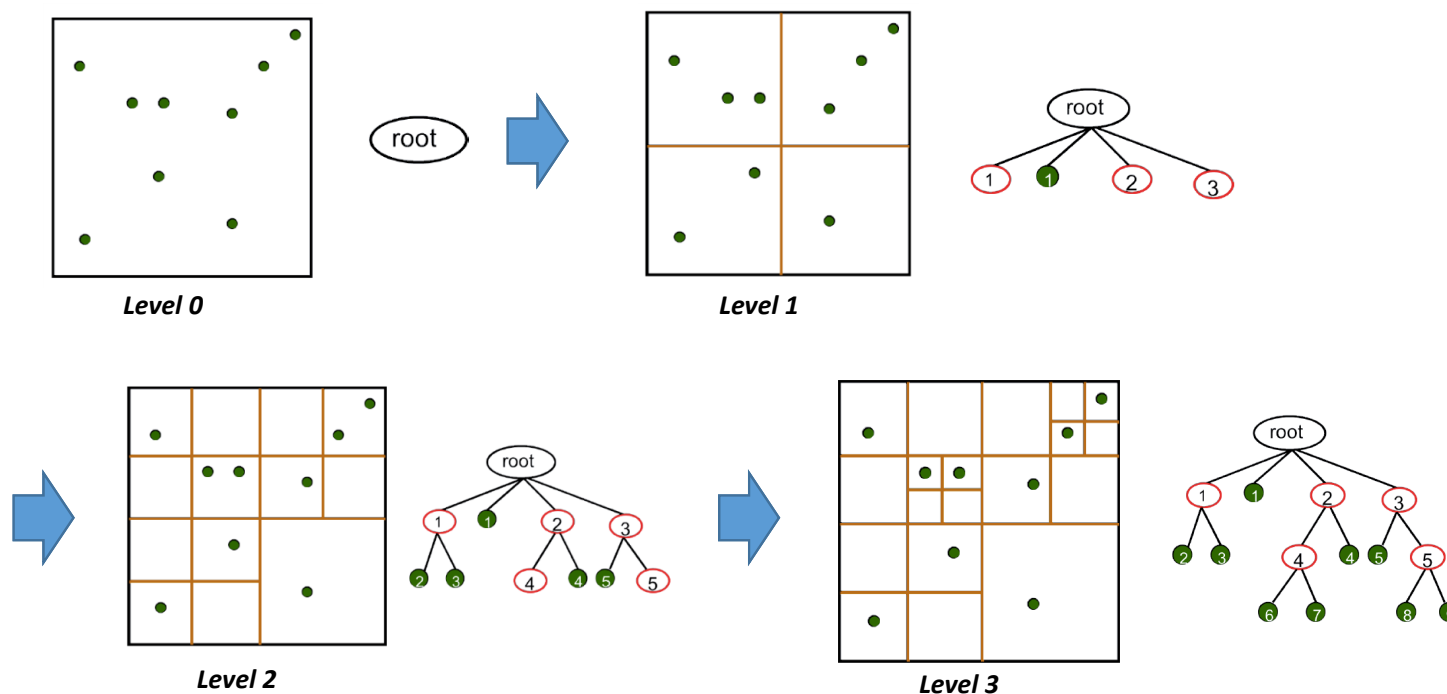
■ Particles in nearby cells are treated individually in complex computations

■ Particles in distant cells are treated as a single large particle to reduce interactions



Tree-Code Parallel Algorithms – Particle Space & Domain Decomposition Example

- Example: Parallel Tree Buildup & Tree Data Structures
 - Interaction between particles based on 'known physical laws'



[6] PEPC Web page

Basic MPI Datatypes & Multi-Dimensional Datasets

■ Basic MPI Datatypes (aka 'intrinsic or primitive' data types)

- Simple, used in many applications and work towards 'code portability'
- `MPI_INT`; `MPI_CHAR`; `MPI_LONG`; `MPI_FLOAT`; `MPI_DOUBLE`; ...
- E.g. need to match `MPI_Datatype` in `MPI_Send` and `MPI_Recv` operations
- E.g. value described by a data-type, a count and memory location
- Challenge: the data must be contiguous in memory (here `void* buf`)

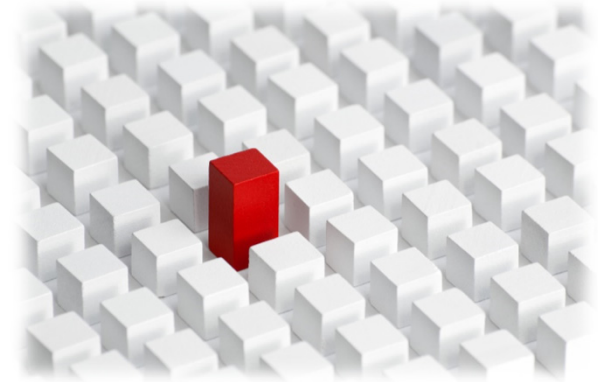
```
Int MPI_Send( void* buf, int count, MPI_Datatype datatype,  
             int dest, int tag, MPI_Comm comm)
```

```
Int MPI_Recv( void* buf, int count, MPI_Datatype datatype,  
             int source, int tag, MPI_Comm comm, MPI_Status* status)
```

- Some applications require data structures that are more sophisticated data types and formats
- Several applications require another simple mechanism for exchanging common data than just basic MPI datatypes or simple multi-dimensional datasets

■ Arrays for multi-dimensional datasets

- Typically used in conjunction with one and only one basic data type
- Flexible and used in a wide variety of applications (e.g. matrices, etc.)



Derived MPI Datatypes – Principles

- Motivation: convenient & efficient & suited for application needs
 1. **Construct** a new datatype using dedicated MPI routines (see below)
 2. **Commit** the new datatype – `MPI_Type_Commit()`
 3. **Take advantage** of the new datatype, e.g. in send/receive operations
- MPI derived datatype (e.g. when you need it often)
 - Represents a ‘map for understanding’ and interpreting message data
 - Transforms an ‘old datatype’ and building constraints to a ‘new datatype’
 - Note: although the old datatype will remain, easier to use the new datatype
- MPI construction routines

- Derived MPI datatypes are constructed from existing other datatypes (e.g. basic data types)
- Used to avoid repeated sends of varied basic types (i.e. slow, clumsy, and error prone)
- Enable a suitable memory layout for complex data structures that consist of several different types



(how do we send a string with 3 chars?)

Derived MPI Datatypes – MPI_Type_contiguous()

- Allocations of a datatype into contiguous locations

```
MPI_Type_contiguous( 3, oldtype, newtype );
```



(simple example, but as this works we can go more sophisticated)

```
...
int buffer[100];
MPI_Type_contiguous(100, MPI_CHAR, &stringtype);
MPI_Type_commit(&stringtype);

/* sending and receiving party
understand the data structure type*/

if (rank==0) {
    MPI_Send(buffer,1,stringtype,1, 123, MPI_COMM_WORLD);
} else {
    MPI_Recv(buffer,1,stringtype,0,123, MPI_COMM_WORLD, &status);
}
...
```

```
int MPI_Type_contiguous(
    int count,
    MPI_Datatype old_type,
    MPI_Datatype *new_type_p
);
```

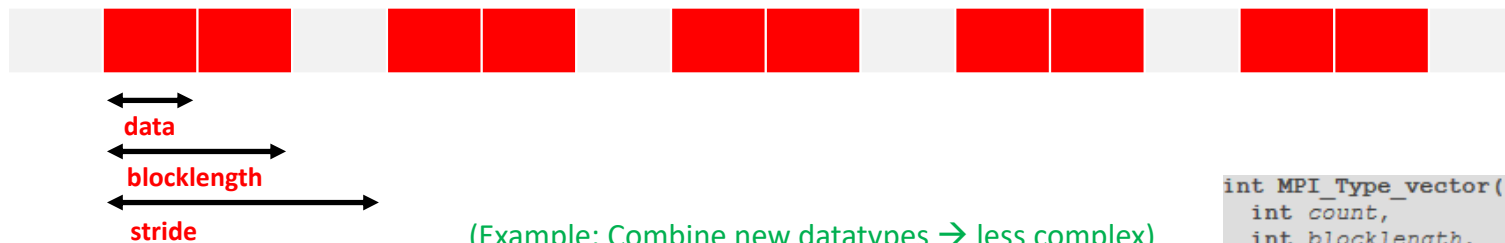
```
int MPI_Send(
    void *buf,
    int count,
    MPI_Datatype datatype,
    int dest,
    int tag,
    MPI_Comm comm
);
```

```
int MPI_Recv(
    void *buf,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm comm,
    MPI_Status *status
);
```

Derived MPI Datatypes – MPI_Type_vector()

- Allocations of a datatype into block-wise locations
 - Locations consist of equally spaced blocks
 - Stride: number of elements between start of each block (integer)

```
MPI_Type_vector( 5, 2, 3, oldtype, newtype );
```



(Example: Combine new datatypes → less complex)

```
...  
MPI_Type_contiguous(3, MPI_INT, &goodnumber);  
MPI_Type_commit(&goodnumber);  
MPI_Type_vector(3,2,3, goodnumber, &lotsofgoodnumbers);  
MPI_Type_commit(&lotsofgoodnumbers);  
...  
MPI_Send(buffer, 1, lotsofgoodnumbers, 1, 123, MPI_COMM_World);  
...
```

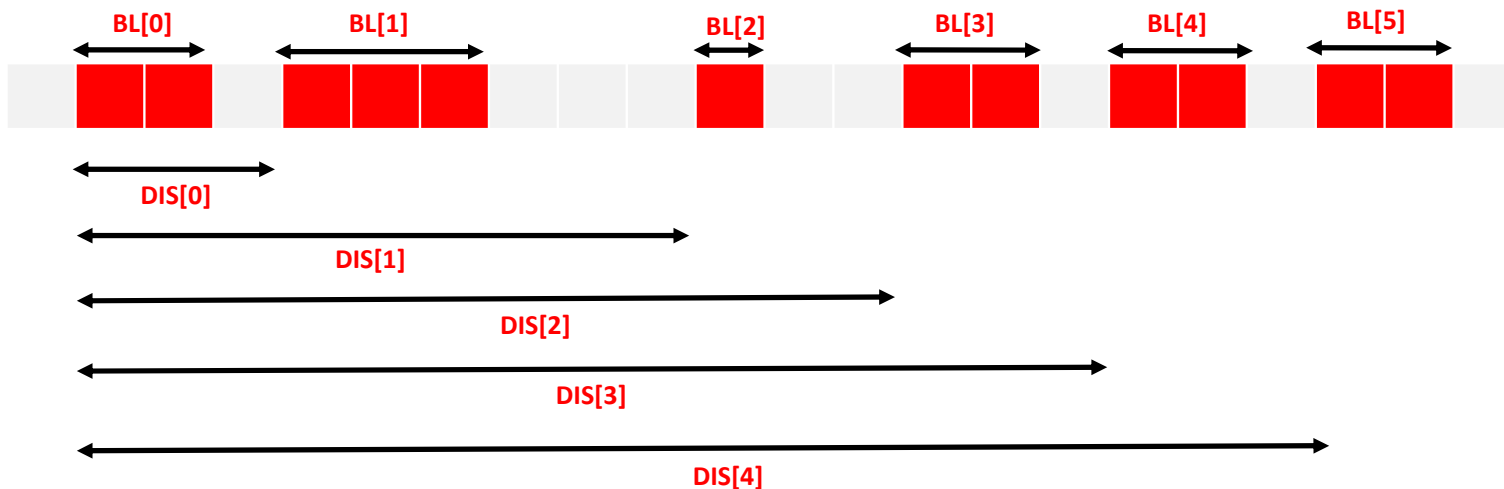
```
int MPI_Type_vector(  
    int count,  
    int blocklength,  
    int stride,  
    MPI_Datatype old_type,  
    MPI_Datatype *newtype_p  
);
```

```
int MPI_Send(  
    void *buf,  
    int count,  
    MPI_Datatype datatype,  
    int dest,  
    int tag,  
    MPI_Comm comm  
);
```

Derived MPI Datatypes – MPI_Type_indexed()

- Allocations of a datatype into a (non uniform) sequence of blocks
 - Each blocks can contain a **different number of copies**
 - Each block can have a **different displacement**

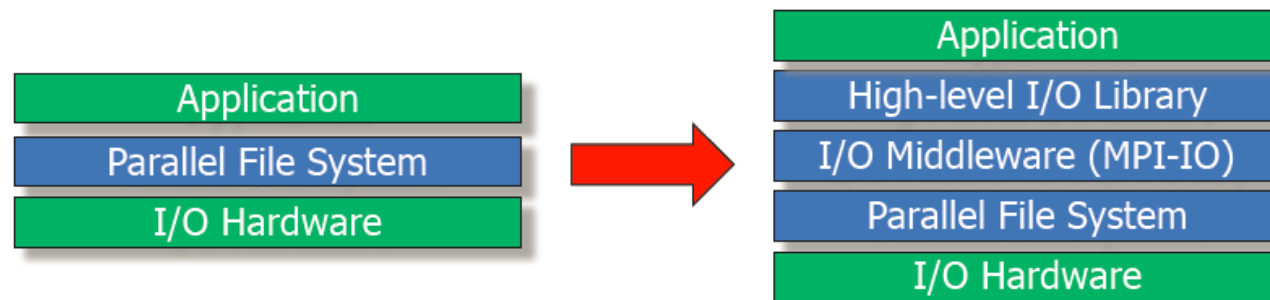
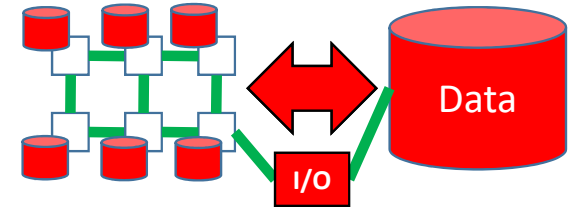
```
array_of_blocklengths[] = {2,3,1,2,2,2} /* below BL */  
array_of_displacements[] = {3,9,12,15,18} /* below DIS */  
  
MPI_Type_indexed ( 6, array_of_blocklengths, array_of_displacements, oldtype, newtype);
```



```
int MPI_Type_indexed(  
    int count,  
    int blocklens[],  
    int indices[],  
    MPI_Datatype old_type,  
    MPI_Datatype *newtype  
);
```

MPI I/O & Parallel Filesystems – Revisited (cf. Lecture 4)

- **Understanding** and **tuning** parallel I/O is needed with ‘big data’
 - Leverage aggregate communication and I/O bandwidth of client machines
- Support: **Add additional software components/libraries layers**
 - Coordination of file access & mapping of application model to I/O model
 - Components and libraries get increasingly specialized / layer
 - High-Level I/O libraries like NetCDF or Hierarchical Data Format (HDF) are standards in the community



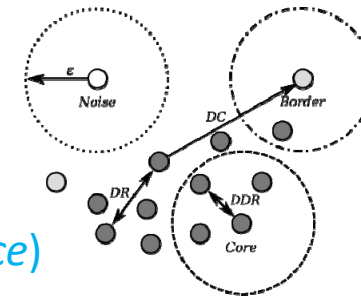
Parallel Filesystems are just one part out of three in the whole I/O process

[8] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

Data Science Example: DBSCAN Clustering Algorithm

■ DBSCAN Algorithm

- Introduced 1996 and most cited clustering algorithm
- Groups number of similar points into clusters of data
- Similarity is defined by a distance measure (e.g. *euclidean distance*)



(MinPoints = 4)

(DR = Density Reachable)

(DDR = Directly Density Reachable)

(DC = Density Connected)

■ Distinct Algorithm Features

- Clusters a variable number of clusters (cf. K-Means Clustering with K clusters)
- Forms arbitrarily shaped clusters (except 'bow ties')
- Identifies inherently also outliers/noise

[11] Ester et al.

- Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm that requires only two parameters and has no requirement to specify number of clusters
- Parameter Epsilon: Algorithm looks for a similar point within a given search radius Epsilon
- Parameter minPoints: Algorithm checks that cluster consist of a given minimum number of points

```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=ml-hpc-1

export OMP_NUM_THREADS=4

# location executable
HPDBSCAN=/homea/hpclab/train001/tools/hpdbscan/dbscan

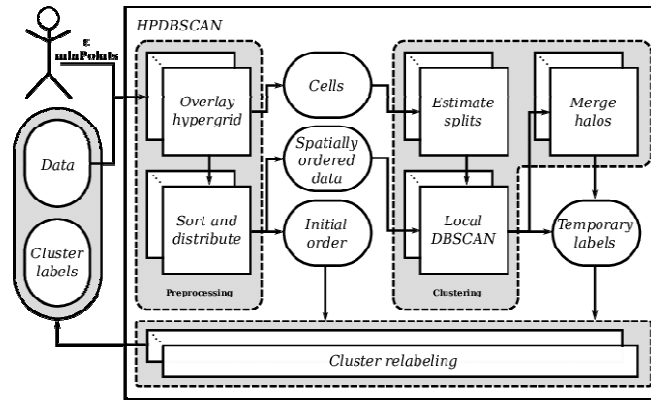
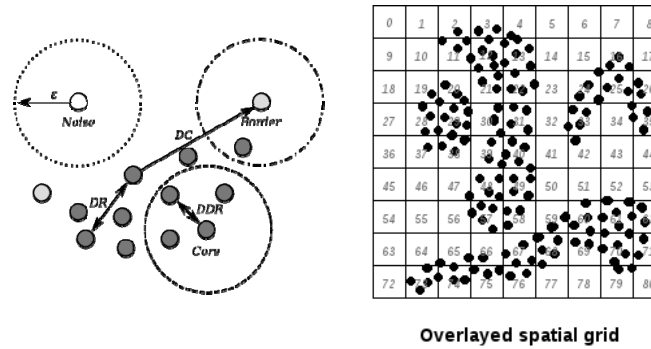
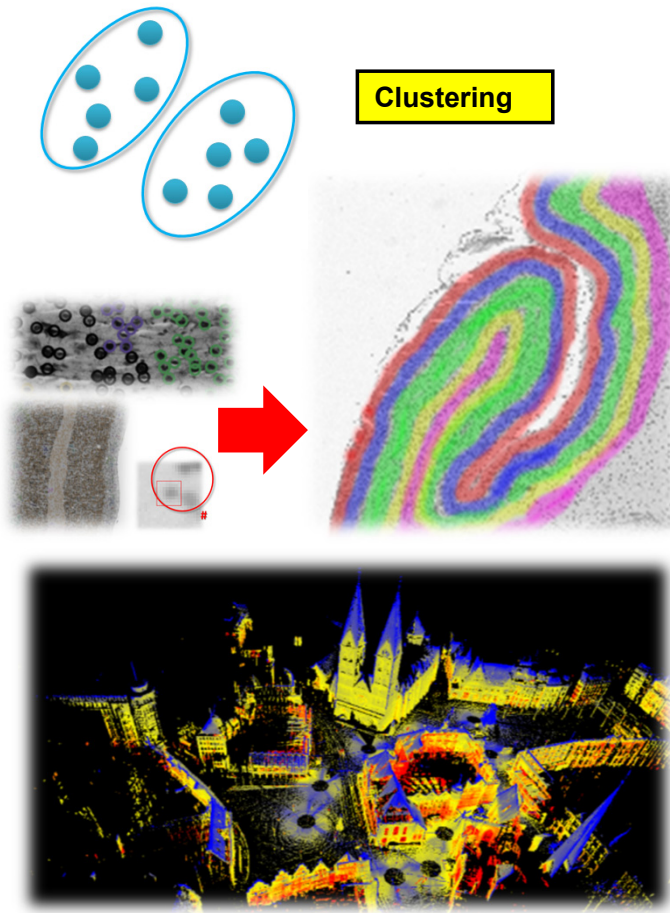
# your own copy of bremen small
BREMENSMALLDATA=/homea/hpclab/train001/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/homea/hpclab/train001/bremen.h5

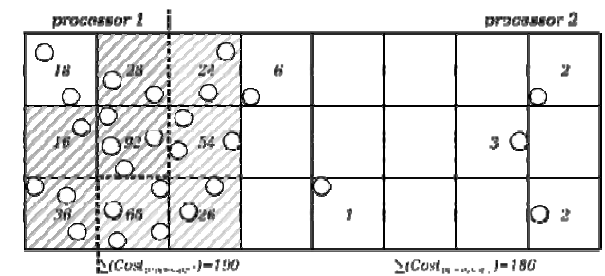
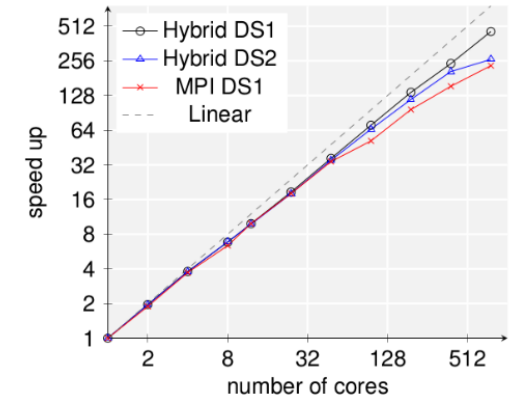
srun $HPDBSCAN -m 100 -e 300 -t 12 $BREMENSMALLDATA
```

➤ Lecture 8 provides more details about using MPI and OpenMP for data science algorithms used in clustering and classification of data

'Big Data' Science Example – Parallel & Scalable Clustering Algorithm – Revisited



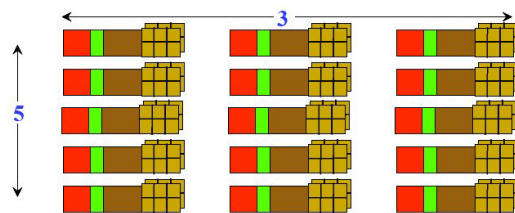
[9] M. Goetz and M. Riedel et al,
Proceedings IEEE Supercomputing Conference, 2015



Using High-Level I/O Hierarchical Data Format (HDF) for Data Structures

■ Simple ‘compound type’ example:

- Array of data records with some descriptive information (5x3 dimension)
- HDF5 data structure type with int(8); int(4); int(16); 2x3x2 array (float32)

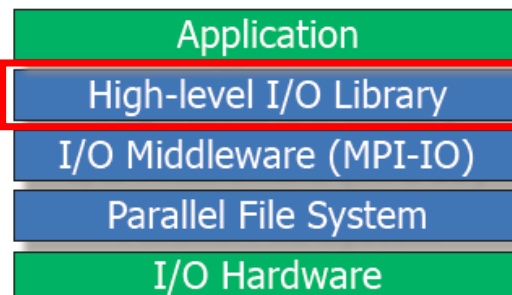
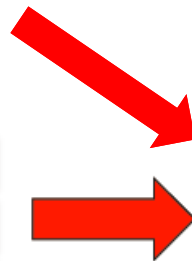
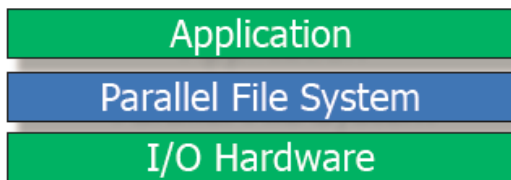


Dimensionality: 5 x 3

Datatype: int8 int4 int16 2x3x2 array of float32

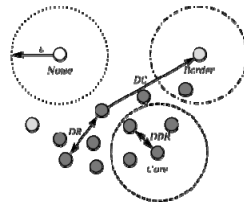


Record

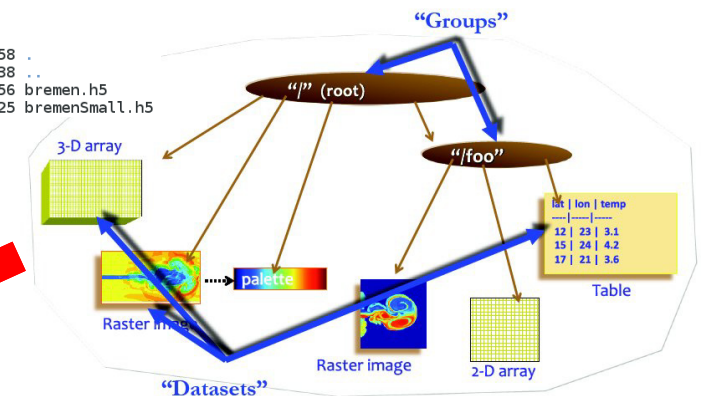


(application example parallel & scalable clustering with HPDBSCAN using Bremen data in HDF5)

```
[train001@jrl07 bremen]$ pwd
/homea/hpclab/train001/data/bremen
[train001@jrl07 bremen]$ ls -al
total 1342208
drwxr-xr-x 2 train001 hpc lab      512 Jan 14 09:58 .
drwxr-xr-x 4 train001 hpc lab      512 Jan 14 08:38 ..
-rw-r--r-- 1 train001 hpc lab 1302382632 Jan 14 09:56 bremen.h5
-rw-r--r-- 1 train001 hpc lab 72002416 Jan 14 08:25 bremenSmall.h5
```



- The Hierarchical Data Format (HDF) is a technology suite that enables the work with extremely large and complex data collections
- A HDF version 5 file is a container to organize data objects – it looks like a filesystem within a file



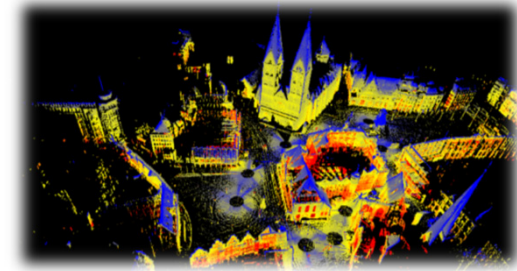
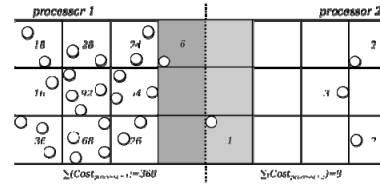
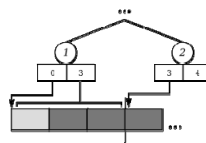
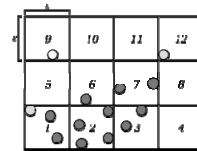
[8] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

[10] HDF@ I/O workshop

'Big Data' Science Example: Using High-Level I/O Hierarchical Data Format (HDF)

■ Parallelization Strategy

- Chunk data space equally
- Overlay with hypergrid
- Apply cost heuristic
- Redistribute points (data locality)
- Execute DBSCAN locally
- Merge clusters at chunk edges
- Restore initial order



■ Data organization

- Use of HDF5
(cf. Lecture 5)
- Cluster Id stored
in HDF5 file

```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=m1-hpc-1

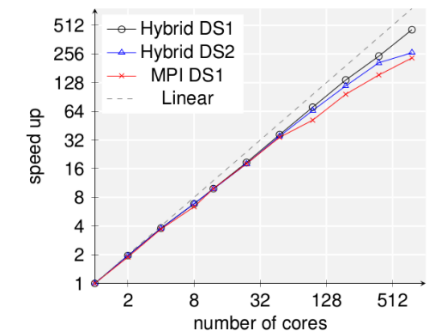
export OMP_NUM_THREADS=4

# location executable
HPDBSCAN=/homea/hpclab/train001/tools/hpdbscan/dbscan

# your own copy of bremen small
BREMENSMALLDATA=/homea/hpclab/train001/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/homea/hpclab/train001/bremen.h5

srun $HPDBSCAN -m 100 -e 300 -t 12 $BREMENSMALLDATA
```



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

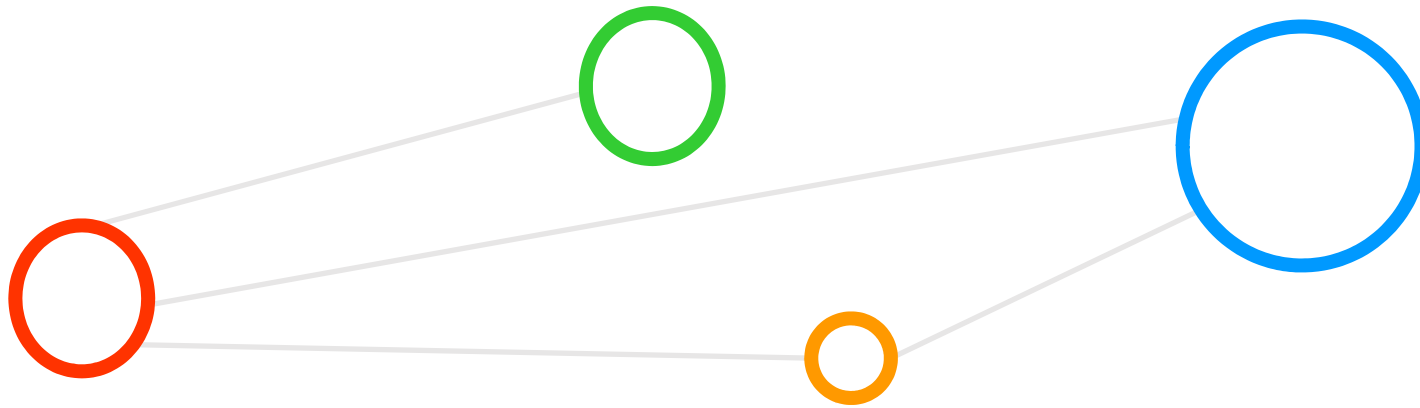
➤ Lecture 8 provides more details about using MPI and OpenMP for data science algorithms used in clustering and classification of data

[Video] Aerospace Engineering Industry Simulations



[7] ANSYS, Aerospace Industry demands

Lecture Bibliography



Lecture Bibliography (1)

- [1] M. Geimer et al., 'SCALASCA performance properties: The metrics tour'
- [2] LLNL MPI Tutorial, Online:
<https://computing.llnl.gov/tutorials/mpi/>
- [3] Introduction to Groups and Communicators, Online:
<http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/>
- [4] Wolfgang Frings, 'HPC I/O Best Practices at JSC', Online:
http://www.fz-juelich.de/ias/jsc/DE/Leistungen/Dienstleistungen/Dokumentation/Praesentationen/folien-parallel-io-2014_table.html?nn=469624
- [5] Parallel I/O, YouTube Video, Online:
<http://www.youtube.com/watch?v=cXbEVsExU9c>
- [6] PEPC Web page, FZ Juelich, Online:
http://www.fz-juelich.de/ias/jsc/EN/AboutUs/Organisation/ComputationalScience/Simlabs/slpp/SoftwarePEPC/_node.html
- [7] YouTube Video, 'Aerospace Industry Demands Accurate, Fast, and Reliable Simulation Technology', Online:
<http://www.youtube.com/watch?v=otj39Zk1-aM>
- [8] Rajeev Thakur, Parallel I/O and MPI-IO, Online:
http://www.training.prace-ri.eu/uploads/tx_pracetmo/pio1.pdf
- [9] 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
- [10] Michael Stephan, 'Portable Parallel IO - Handling large datasets in heterogeneous parallel environments', Online:
http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/parallel-io-2014/parallel-io-hdf5.pdf?__blob=publicationFile
- [11] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." Kdd. Vol. 96. 1996, Online:
<https://dl.acm.org/citation.cfm?id=3001507>

Lecture Bibliography (2)

- [12] Caterham F1 Team Races Past Competition with HPC, Online:
<http://insidehpc.com/2013/08/15/caterham-f1-team-races-past-competition-with-hpc>
- [13] Introduction to High Performance Computing for Scientists and Engineers,
Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [14] Parallel Algorithms Underlying MPI Implementations, Online:
http://www.slidefinder.net/p/parallel_algorithms_underlying_mpi_implementations/13-parallelalgorithmsunderlyingmpiimplementations/17131238/p2
- [15] YouTube Video, 'Finite element simulation of full scale car crash based 100% on solid elements', Online:
<http://www.youtube.com/watch?v=NrvuFiDqn5A>
- [16] FFTW Tool, M. Frigo & S.G. Johnson, Online:
http://www.fftw.org/fftw3_doc/
- [17] M. Riedel, Th. Eickermann, S. Habbinga, W. Frings, P. Gibbon, D. Mallmann, F. Wolf, A. Streit, Th. Lippert, Felix Wolf, Wolfram Schiffmann, Andreas Ernst, Rainer Spurzem, Wolfgang E. Nagel, 'Computational Steering and Online Visualization of Scientific Applications on Large-Scale HPC Systems within e-Science Infrastructures', pp.483-490, Third IEEE International Conference on e-Science and Grid Computing, 2007, Online:
https://www.researchgate.net/publication/4309569_Computational_Steering_and_Online_Visualization_of_Scientific_Applications_on_Large-Scale_HPC_Systems_within_e-Science_Infrastructures
- [18] How EMI Contributed to the Higgs Boson Discovery, YouTube Video, Online:
<http://www.youtube.com/watch?v=FgcoLUys3RY&list=UUz8n-tukF1S7fqI19KOAAhw>
- [19] Wikipedia on 'Numerical Weather Prediction', Online:
http://en.wikipedia.org/wiki/Numerical_weather_prediction
- [20] Wikipedia on 'stencil code', Online:
http://en.wikipedia.org/wiki/Stencil_code
- [21] Scalasca Performance Analysis Tool, Online:
<http://www.scalasca.org/>

Lecture Bibliography (3)

- [22] John Michalakes et al., 'Opportunities for WRF Model Acceleration', Online:
<http://www2.mmm.ucar.edu/wrf/users/workshops/WS2012/ppts/1.4.pdf>
- [23] German Lecture 'Umfang von MPI 1.2 und MPI 2.0'

