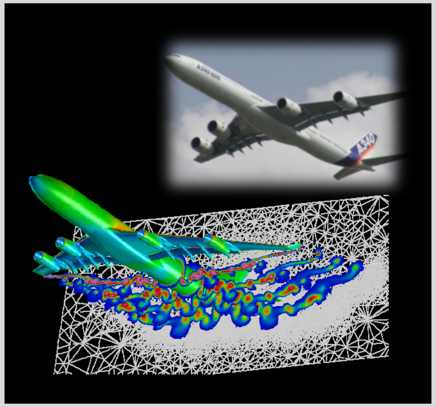


# High Performance Computing

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


PRACTICAL LECTURE 5.1

# Understanding MPI Communicators & Data Structures

October 03, 2019

Webinar


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

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

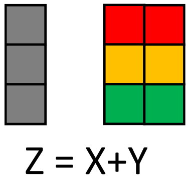
DEEP Projects

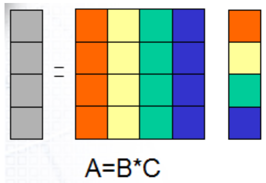
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

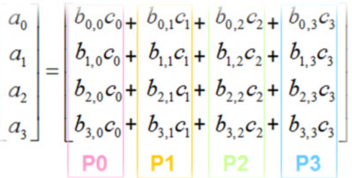
HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

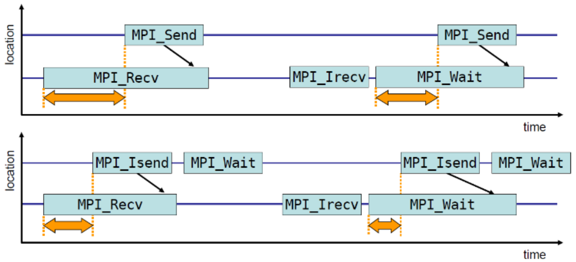
# Review of Lecture 5 – Parallel Algorithms & Data Structures

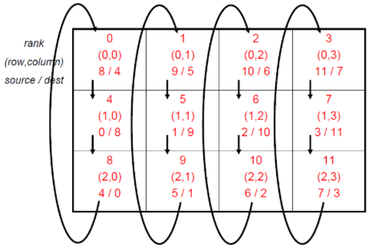
- Selected Parallel Algorithms using MPI



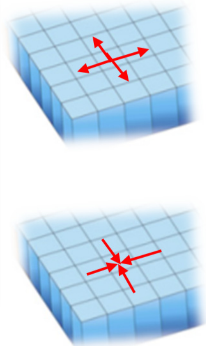



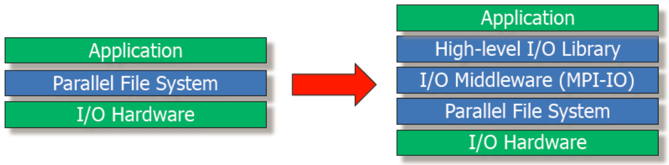


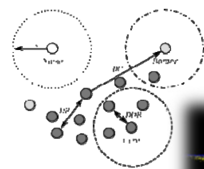


*[1] Metrics tour [2] German MPI Lecture*

- Derived MPI Data Types

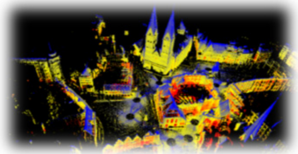

- Parallel I/O & Hierarchical Data Format (HDF)



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

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



# 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

- Non-Blocking Communications & Communicator Examples
  - Blocking vs. Non-Blocking Communication – Revisited & Algorithms
  - Non-Blocking Communication with Isend/Irecv & Wait Functions
  - Understanding MPI Cartesian Communicator Dimensions & Shifts
  - Using Non-Blocking Communication with Cartesian Communicators
  - Simple Application Examples on Jötunn HPC System

- MPI Derived Data Types & Parallel I/O via HDF Examples
  - Simple Examples of MPI Derived Data Types with Applications
  - MPI I/O & Parallel Filesystems using HDF5 – Revisited
  - Data Science Example with Parallel & Scalable HPDBSCAN Algorithm
  - Understanding HDF5 Binary File Format & Using H5Dump Tool
  - HPDBSCAN Clustering of Point Cloud Data Set Bremen on Jötunn HPC System

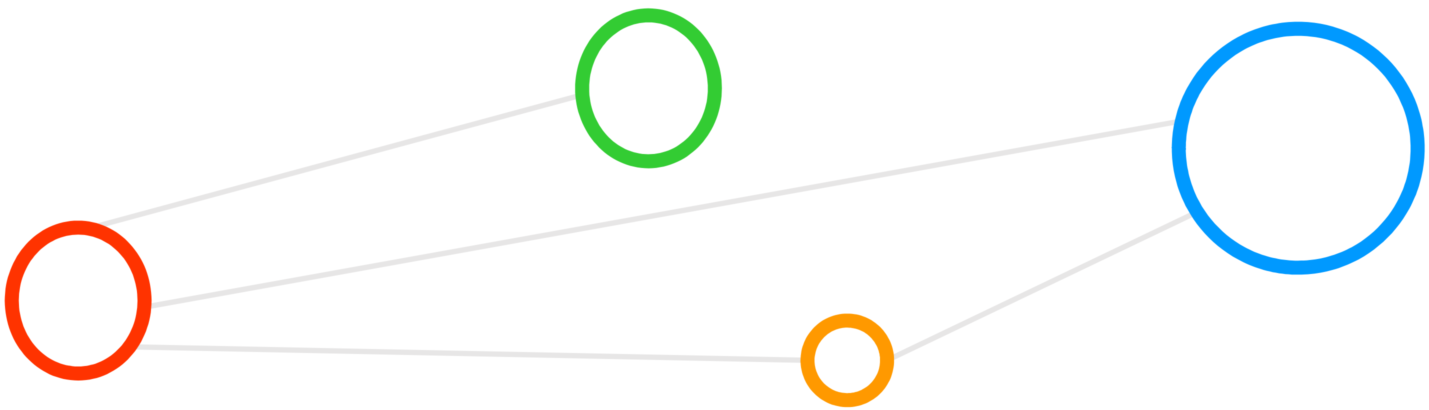
- This lecture is not considered to be a full introduction to MPI programming and the overall MPI functions library and rather focusses on selected commands and concepts particularly relevant for our assignments, e.g. the use of the MPI Cartesian communicator & MPI derived data types & parallel I/O
- The goal of this practical lecture is to make course participants aware of the process of using different communicators in MPI  programs and the use of data structures in MPI that enable many scientific & engineering applications in data sciences & simulation sciences today

# Selected Learning Outcomes – Revisited

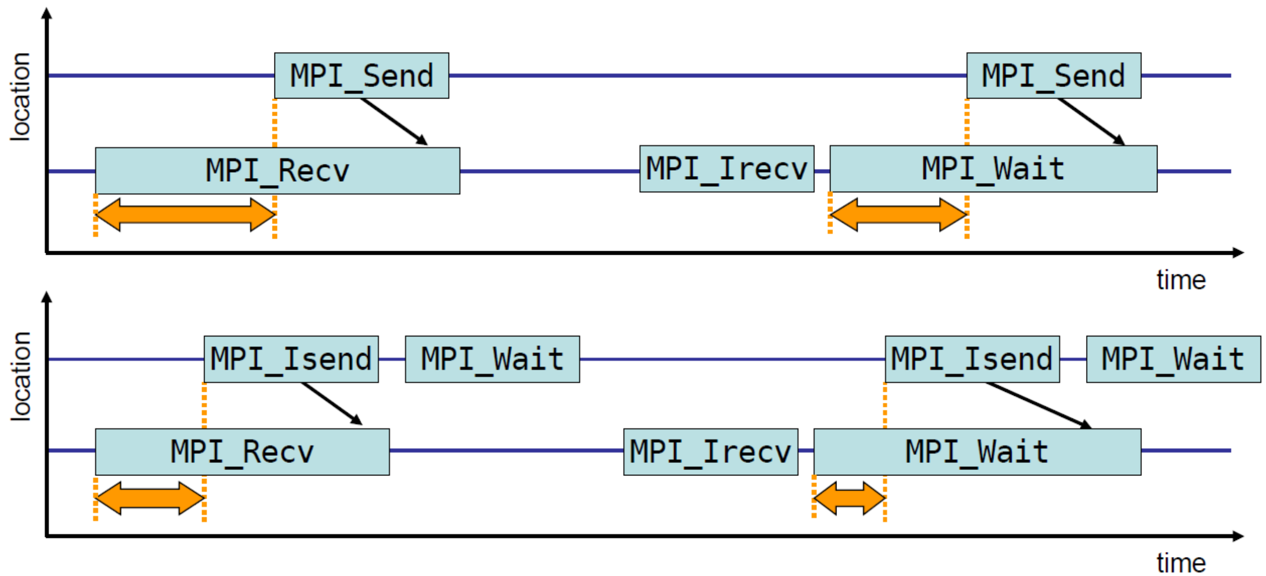
- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming → e.g., scheduling(!)
  - 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

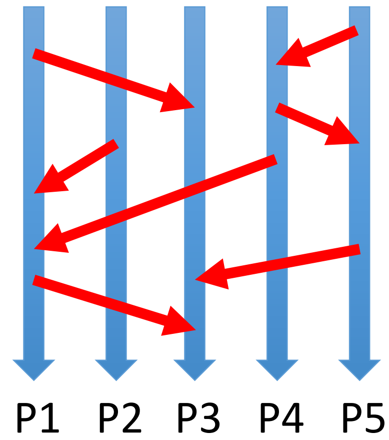
# Non-Blocking Communications & Communicator Examples



# Blocking vs. Non-blocking communication (cf. Lecture 4)

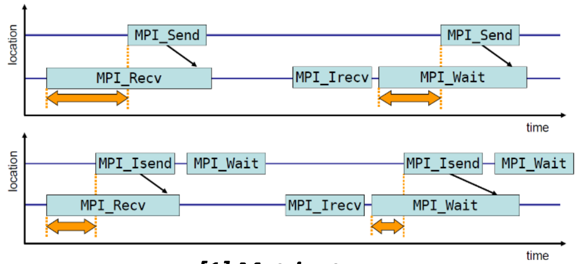



*[1] Metrics tour*

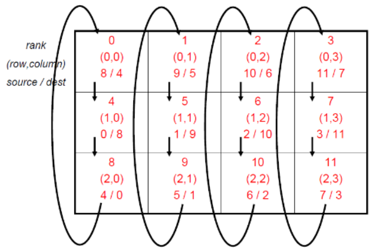



- **Blocking vs. non-blocking: MPI_Send() blocks until data is received; MPI_Isend() continues**
- **The use of these functions can cause different performance problems (e.g. here 'late sender')**
- **MPI_Wait() does wait for a given MPI request to complete before continuing**
- **MPI_Waitall() does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

# Blocking vs. Non-blocking Communication – Parallel Algorithms (cf. Lecture 5)

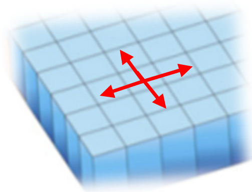


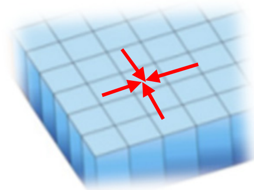

*[1] Metrics tour*




*[2] German MPI Lecture*

```c
// 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[])
```

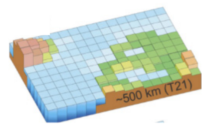
**Input Parameters**

**count**
list length (integer)

**array_of_requests**
array of request handles (array of handles)

**Output Parameters**

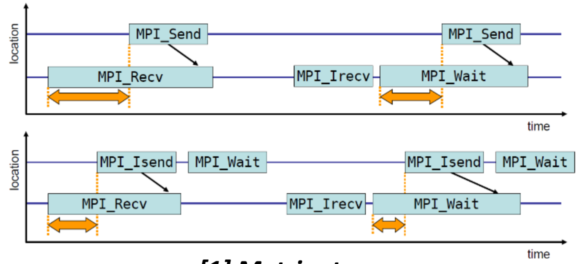
**array_of_statuses**
array of status objects (array of Statuses). May be MPI_STATUSES_IGNORE.




> Lecture 10 shows how MPI non-blocking communication is used in Cartesian communicators for nearest neighbor communications

# MPI Non-Blocking Communication – Motivation & Methods Examples

- Motivation: Non-Blocking Communication
  - Improved performance but harder to program (keep overview)
  - E.g. allows computations and communication to overlap
  - E.g. optimization patterns: e.g. MPI_Isend + MPI_Wait makes no sense (one could simply use send)




*[1] Metrics tour*

- Selected useful methods
  - E.g. MPI_Irecv() non-blocking receive
  - E.g. MPI_Isend() non-blocking send
  - E.g. MPI_Wait() waits for MPI requests
  - E.g. MPI_Get_processor_name() identifies particular piece of hardware (i.e. processor)
  - E.g. MPI_Wtime() is elabsed time / processor
  - Please refer to MPI specifications online for more methods & details

```
int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Request, *request)

int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm, comm, MPI_Request *request)

int MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Get_processor_name( char *name, int *resultlen )

double MPI_Wtime( void )
```

➢ **Lecture 9 will offer more examples where MPI non-blocking communication can influence the performance of parallel applications**

# Non-Blocking Communicaton Application Example – MPI_Isend()

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

main(argc, argv)
int argc;
char *argv[];
{

  int pool_size, my_rank;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &pool_size);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  if (my_rank == 0) {

    char send_buffer[BUFSIZ], my_cpu_name[BUFSIZ];
    int my_name_length;
    MPI_Request request;
    MPI_Status status;

    MPI_Get_processor_name(my_cpu_name, &my_name_length);
    sprintf (send_buffer, "Dear Task 1,\n\
Please do not send any more messages.\n\
Please send money instead.\n\
\tYours faithfully,\n\
\tTask 0\n\
\tRunning on %s\n", my_cpu_name);
    MPI_Isend (send_buffer, strlen(send_buffer) + 1, MPI_CHAR,
               1, 77, MPI_COMM_WORLD, &request);
    printf("hello there user, I've just started this send\n\
and I'm having a good time relaxing.\n");
    MPI_Wait (&request, &status);
    printf("hello there user, it looks like the message has been sent.\n");

    if (request == MPI_REQUEST_NULL) {
      printf("\tthe send request is MPI_REQUEST_NULL now\n");
    } else {
      printf("\tthe send request still lingers\n");
    }

  }
```

- Our rank 0 processor takes the role of a nonblocking sender

- The method MPI_Get_processor_name() identifies a particular piece of hardware w.r.t. to the concrete processor name (e.g., maybe same as gethostname()
- We use the MPI_Get_processor_name() function output to include in the message to another task where exactly the processor is running right now that performs later the non-blocking send

- The method MPI_Isend() begins a non-blocking send operation and outputs a request
- Here it is initiated by rank 0 and is send to rank 1 in MPI_COMM_WORLD

```
int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm, comm, MPI_Request *request)
```

- MPI_Wait()  waits for an MPI request to complete

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

*[4] Non-Blocking Application Example*

# Non-Blocking Communicaton Application Example – MPI_Irecv()

```
 else if (my_rank == 1) {

   char recv_buffer[BUFSIZ], my_cpu_name[BUFSIZ];
   int my_name_length, count;
   MPI_Request request;
   MPI_Status status;

   MPI_Get_processor_name(my_cpu_name, &my_name_length);
   MPI_Irecv (recv_buffer, BUFSIZ, MPI_CHAR, 0, 77, MPI_COMM_WORLD,
              &request);
   printf("hello there user, I've just started this receive\n\
on %s, and I'm having a good time relaxing.\n", my_cpu_name);
   MPI_Wait (&request, &status);
   MPI_Get_count (&status, MPI_CHAR, &count);
   printf("hello there user, it looks like %d characters \
have just arrived:\n", count );
   printf("%s", recv_buffer);

   if (request == MPI_REQUEST_NULL) {
     printf("\tthe receive request is MPI_REQUEST_NULL now\n");
   } else {
     printf("\tthe receive request still lingers\n");
   }

 }

 MPI_Finalize();
}
```

- **Our rank 1 processor takes the role of a nonblocking receiver**

- **The method MPI_Get_processor_name() identifies a particular piece of hardware w.r.t. to the concrete processor name (e.g., maybe same as gethostname()**
- **We use the MPI_Get_processor_name() function output to show later in a local message where the receiver is running right now that performs later the non-blocking receive**

- **The method MPI_Irecv () begins a non-blocking receive operation & outputs a request**
- **Here it is initiated by rank 0 and is send to rank 1 in MPI_COMM_WORLD**

```
int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm, comm, MPI_Request *request)
```

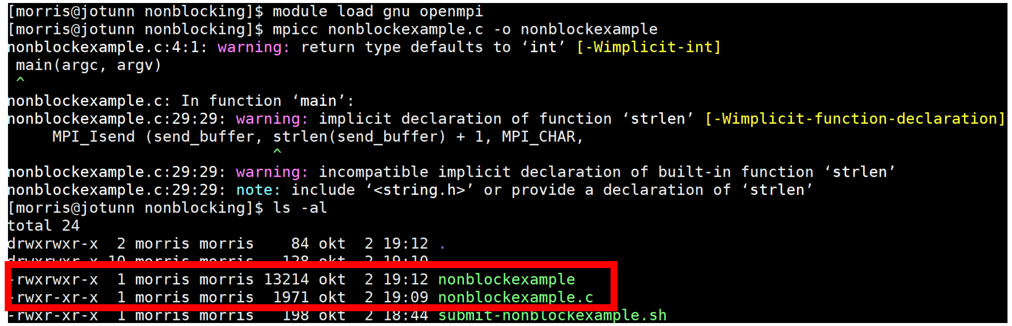
- **MPI_Wait() waits for an MPI request to complete**

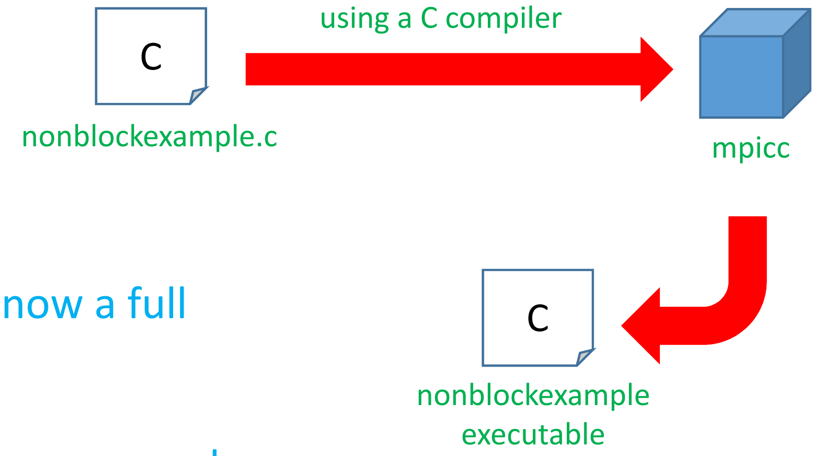
```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

***[4] Non-Blocking Application Example***

# Load the right Modules for Compilers & Compile C & MPI Program

- Using modules to get the right C compiler for compiling broadcast.c
  - ‘module load gnu openmpi‘
  - Note: there are many C compilers available, we here pick one for our particular HPC course that works with the Message Passing Interface (MPI)
  - Note: If there are no errors, the file nonblockexample is now a full C program executable that can be started by an OS
  - New: C program with MPI message exchanges (cf. Lecture 2 – Parallel Programming with MPI)




*[5] Icelandic HPC Machines & Community*

# Parallel Processing – Executing an MPI Program with MPIRun & Script

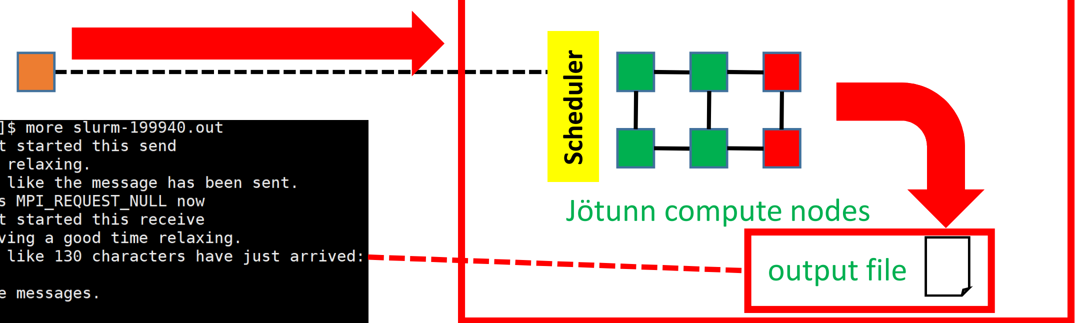
- Submission using the Scheduler – Update(!)
  - Example: SLURM on Jötunn HPC system
  - Scheduler allocated 2 cores as requested
  - MPIRun & scheduler distribute the executable on the right nodes
  - Note the outputs from the two ranks that perform nonblocking send and receive operations

**▪ The job script parameter #SBATCH –N X indicates the NUMBER X OF NODES; allocation by scheduler then depends on HPC system setup**
**▪ The job script parameter #SBATCH –n X indicates the NUMBER X OF CORES; allocation by scheduler then depends on HPC system setup**
**▪ Both parameters #SBATCH –n X and #SBATCH –N X can be combined in the job script if needed to fine-tune the requirements for how much cores are needed on how many nodes**

```
#!/bin/bash
#SBATCH -J nonblockexample
#SBATCH -n 2
#SBATCH --mail-user=morris@hi.is
#SBATCH --mail-type=end
module load gnu openmpi
mpirun /home/morris/2019-HPC-Course/nonblocking/nonblockexample
```

```
[morris@jotunn nonblocking]$ qstat
Job id               Name             Username        Time Use S Queue
-------------------- ---------------- --------------- -------- - ------
199590               hello-mpi-exampl jfb3            00:00:00 Q normal
199591               hello-mpi-exampl jfb3            00:00:00 Q normal
199592               hello-mpi-exampl jfb3            00:00:00 Q normal
199593               hello-mpi-exampl jfb3            00:00:00 Q normal
199595               pingpong         jfb3            00:00:00 Q normal
199596               pingpong         jfb3            00:00:00 Q normal
199840               pingpong         jfb3            00:00:00 Q normal
199841               pingpong         jfb3            00:00:00 Q normal
199940               nonblockexample  morris          00:00:00 C normal
```

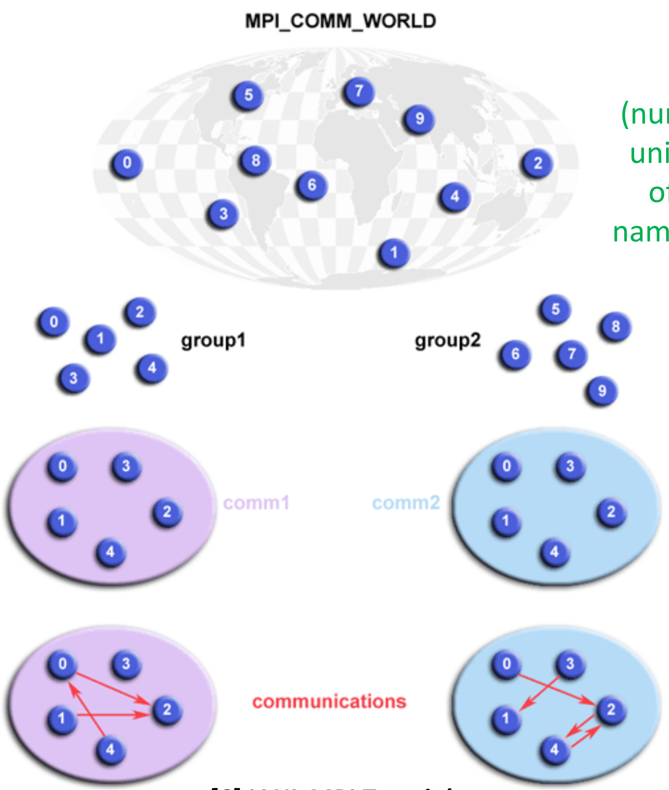
```
[morris@jotunn nonblocking]$ sbatch submit-nonblockexample.sh
Submitted batch job 199940
```




```
[morris@jotunn nonblocking]$ more slurm-199940.out
hello there user, I've just started this send
and I'm having a good time relaxing.
hello there user, it looks like the message has been sent.
         the send request is MPI_REQUEST_NULL now
hello there user, I've just started this receive
on compute-2-0, and I'm having a good time relaxing.
hello there user, it looks like 130 characters have just arrived:
Dear Task 1,
Please do not send any more messages.
Please send money instead.
         Yours faithfully,
         Task 0
         Running on compute-2-0
         the receive request is MPI_REQUEST_NULL now
```

```
[morris@jotunn nonblocking]$ ls -al
total 28
drwxrwxr-x  2 morris morris   107 okt  2 19:17 .
drwxrwxr-x 10 morris morris   128 okt  2 19:10 ..
-rwxrwxr-x  1 morris morris 13214 okt  2 19:12 nonblockexample
-rwxr-xr-x  1 morris morris  1971 okt  2 19:09 nonblockexample.c
-rw-rw-r--  1 morris morris   526 okt  2 19:17 slurm-199940.out
-rwxr-xr-x  1 morris morris   199 okt  2 19:17 submit-nonblockexample.sh
```

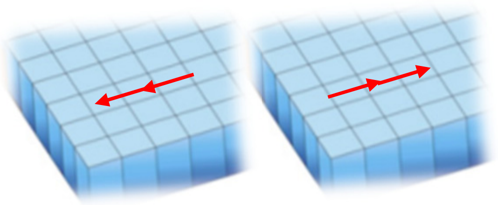
# MPI Communicators – Create MPI Cartesian Communicators (cf. Lecture 4)




*[6] LLNL MPI Tutorial*

(numbers reflect unique identity of processor named 'MPI rank)

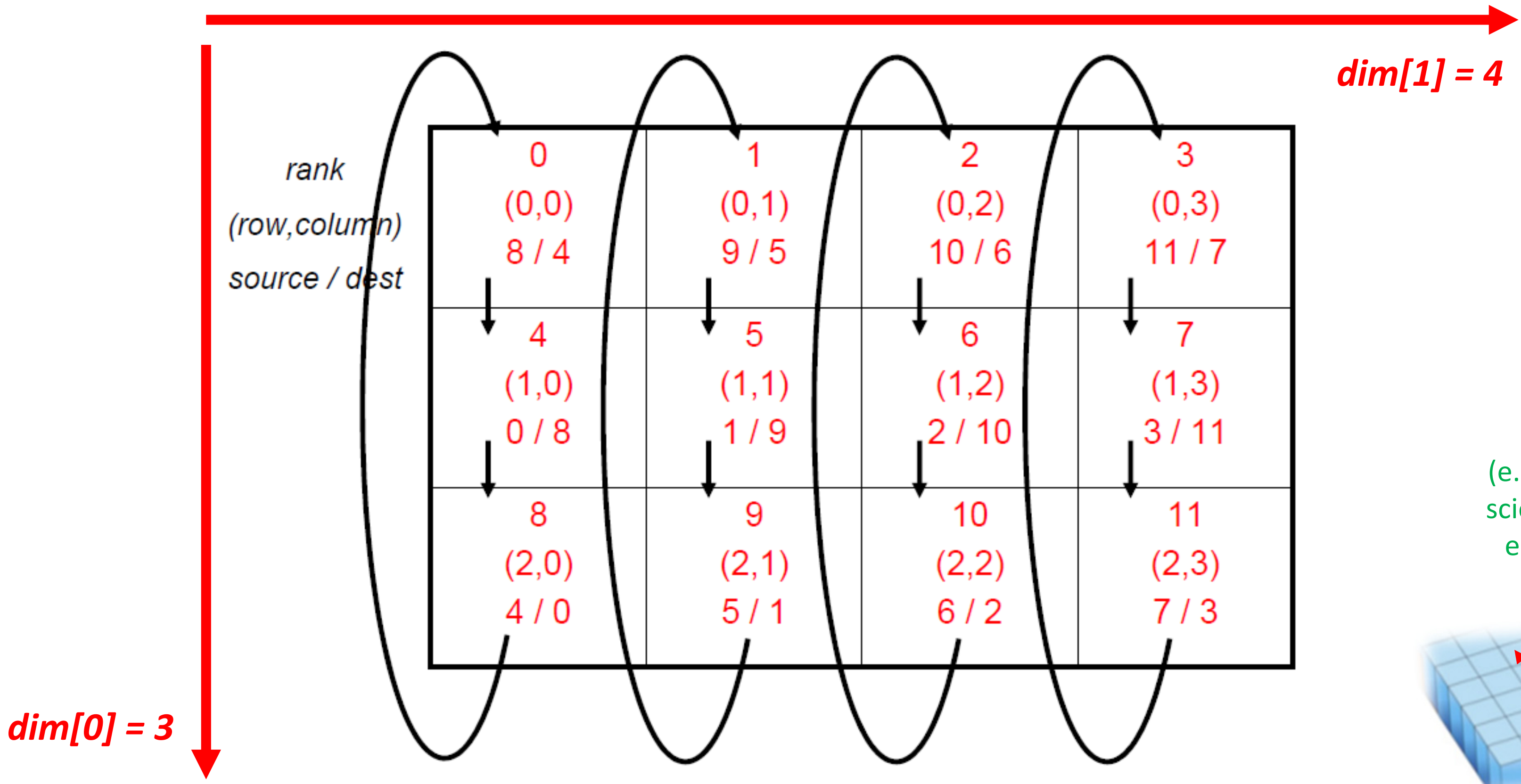
- Create (sub-)groups of the processes & virtual groups of processes
  - E.g. optimized for cartesian topology `MPI_Cart_create()`
  - Creates a new communicator out of MPI_COMM_WORLD
  - Dims: array with length for each dimension
  - Periods: logical array specifying whether the grid is periodic or not
  - Reorder: Allow reordering of ranks in output communicator



(e.g. using MPI messages in scientific simulations and/or engineering applications)

> Assignment #3 will make use of the cartesian communicator in a simple application example that includes the moving of boats & fish

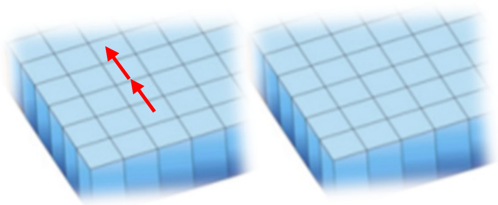
# Cartesian Communicator Example – Conceptual View (cf. Lecture 4)




('cartesian structure')

*modified from [2] German MPI Lecture*

(e.g. using MPI messages in scientific simulations and/or engineering applications)



# Cartesian Communicator Example – Source-code View (cf. Lecture 4)

```c
#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);
  dims[0]=3; dims[1] = 4;
  periods[0]=true; periods[1]=true;
  reorder = false;
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims,
            periods, reorder, &comm_2d);
  MPI_Cart_coords(comm_2d, rank, 2, &coords);
  MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);
  a = rank; b = 1;
  MPI_Sendrecv(a, 1, MPI_REAL, dest, 13, b, 1,
            MPI_REAL, source, 13, comm_2d, &status);
  MPI_Finalize();
  return 0;
}
```

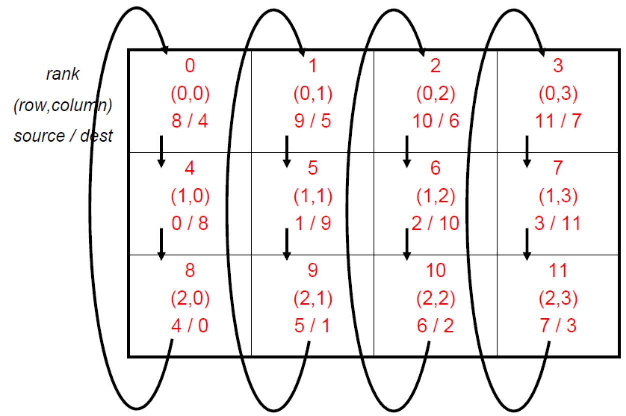
- **Preparing parameter dims as array with length for each dimension (here 3 x 4)**
- **Preparing parameter periods as logical array specifying whether the cartesian grid is period**
- **Preparing parameter reorder as not reordering of ranks in output communicator**

- **MPI_Cart_create() creates a new communicator (cartesian structure) according to specified dimensions in variables**

- **MPI_Cart_coords() obtains process coordinates in cartesian topology – note that this JUST obtaines the current process coordinates – no actual shift is done yet**
- **MPI_Cart_shift() obtains 'ranks' for shifting data in cartesian topology – note that this JUST prepares for a shift understanding which ranks are affected by shift**

- **A real shift is done using a typical MPI message exchange with the obtained ranks and in the space of the Cartesian communicator**

*[2] German MPI Lecture*



# Cartesian Communicator Example – MPI_Cart_create()

```c
#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);

  int dims[2] = {3,4};
  int periods[2] = {1,1};
  int coords[2];
  int reorder = 1;

  int source, dest, a, b;

  MPI_Comm comm_2d;
  MPI_Status status;

  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_2d);
```

- C uses the definition that false is ‘exact value 0‘ and true is ‘unequal 0‘ (e.g. 1)
- The usefulness of the different levels of periodicity depends on the application logic of the corresponding scientific simulation
- Setting the periodic or non-periodic levels influences the shifts patterns

## MPI_Cart_create

Makes a new communicator to which topology information has been attached

### Synopsis

```c
int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[],
                    const int periods[], int reorder, MPI_Comm *comm_cart)
```

### Input Parameters

**comm_old**
: input communicator (handle)

**ndims**
: number of dimensions of cartesian grid (integer)

**dims**
: integer array of size ndims specifying the number of processes in each dimension

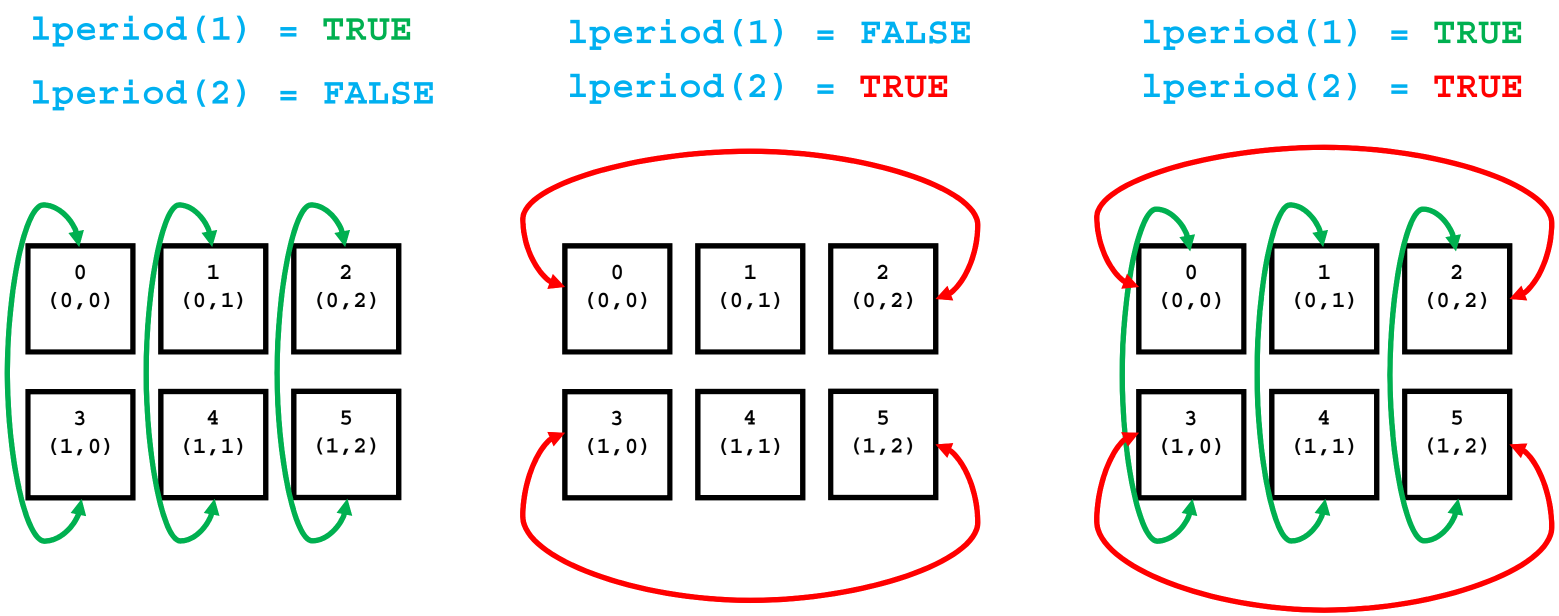
**periods**
: logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension

**reorder**
: ranking may be reordered (true) or not (false) (logical)

### Output Parameters

**comm_cart**
: communicator with new cartesian topology (handle)

# Cartesian Communicators – Periodicity in Detail




- **C uses the definition that false is ‘exact value 0‘ and true is ‘unequal 0‘ (e.g. 1)**
- **The usefulness of the different levels of periodicity depends on the application logic of the corresponding scientific simulation**
- **Setting the periodic or non-periodic levels influences the shifts patterns**

*[2] German MPI Lecture*

# Cartesian Communicator Example – MPI_Cart_coords()

```
#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);

  int dims[2] = {3,4};
  int periods[2] = {1,1};
  int coords[2];
  int reorder = 1;

  int source, dest, a, b;

  MPI_Comm comm_2d;
  MPI_Status status;

  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_2d);

  MPI_Cart_coords(comm_2d, rank, 2, coords);
```

- **Obtains coordinate from each process from the cartesian communicator**

## MPI_Cart_coords

Determines process coords in cartesian topology given rank in group

### Synopsis

```
int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[])
```

### Input Parameters

**comm**
: communicator with cartesian structure (handle)

**rank**
: rank of a process within group of `comm` (integer)

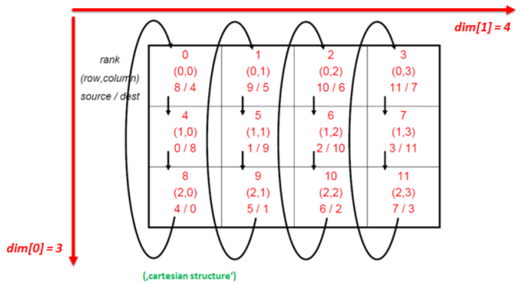
**maxdims**
: length of vector `coords` in the calling program (integer)

### Output Parameters

**coords**
: integer array (of size `ndims`) containing the Cartesian coordinates of specified process (integer)

# Cartesian Communicator Example – MPI_Cart_shift()

```c
#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);

  int dims[2] = {3,4};
  int periods[2] = {1,1};
  int coords[2];
  int reorder = 1;

  int source, dest, a, b;

  MPI_Comm comm_2d;
  MPI_Status status;

  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_2d);

  MPI_Cart_coords(comm_2d, rank, 2, coords);

  MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);
```




- (just!) prepares a 'shift' to neighbours down in the Grid (in this example) according cartesian setup (obtain the rank of them)
- Think: dimension[0] = 'direction' of the Grid – think better coordinate dimension here
- Think: Upwards in dimension means 'down'

## MPI_Cart_shift

Returns the shifted source and destination ranks, given a shift direction and amount

### Synopsis

```c
int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *rank_source,
                   int *rank_dest)
```

### Input Parameters

**comm**
: communicator with cartesian structure (handle)

**direction**
: coordinate dimension of shift (integer)

**disp**
: displacement (> 0: upwards shift, < 0: downwards shift) (integer)

### Output Parameters

**rank_source**
: rank of source process (integer)

**rank_dest**
: rank of destination process (integer)

### Notes

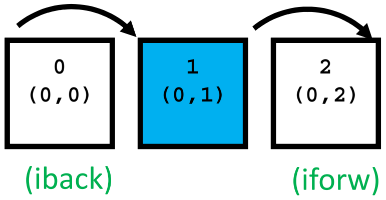
The `direction` argument is in the range `[0,n-1]` for an n-dimensional Cartesian mesh.

# Cartesian Communicators – Standard Shifts

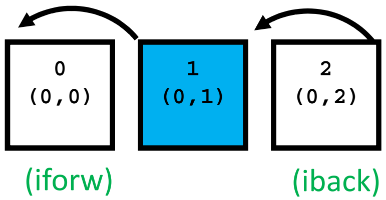
`MPI_Cart_shift(comm, dir, disp, iback, iforw, ierr)`

```
int MPI_Cart_shift(
  MPI_Comm comm,
  int direction,
  int displ,
  int *source,
  int *dest
);
```

- Positive Shift
  - `disp = +1`




- Negative Shift
  - `disp = -1`




- **Shifts prepares the communication with neighbours with send/receive operations along each different directions and obtain ranks to be used in send/receive operations**

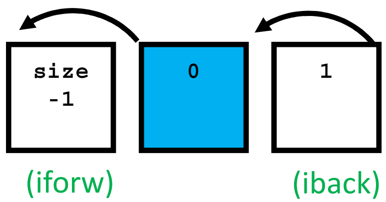
*[2] German MPI Lecture*

# Cartesian Communicators – Problematic Shifts

`MPI_CART_SHIFT(comm, dir, disp, iback, iforw, ierr)`

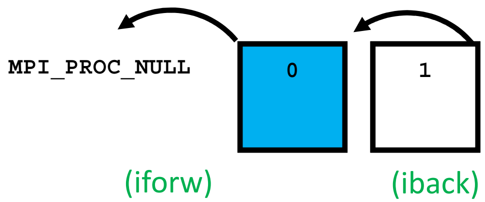
```
int MPI_Cart_shift(
  MPI_Comm comm,
  int direction,
  int displ,
  int *source,
  int *dest
);
```

- Negative Shift (periodic)
  - `disp = -1`




- **Size-1 indicates that the next shift is going to perform a 'turnaround / period' given a periodic cartesian communicator setup**

- Off-end Shift (non-periodic)
  - `disp = -1`




- **MPI_PROC_NULL 'as dummy process' indicates here that the next shift is leaving the defined dimension of the cartesian communicator in a non-periodic setup**

*[2] German MPI Lecture*

# Cartesian Communicator Example – MPI_Sendrecv()

```c
#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);

  int dims[2] = {3,4};
  int periods[2] = {1,1};
  int coords[2];
  int reorder = 1;

  int source, dest, a, b;

  MPI_Comm comm_2d;
  MPI_Status status;

  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_2d);

  MPI_Cart_coords(comm_2d, rank, 2, coords);

  MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);

  a = rank; b = 100;

  MPI_Sendrecv(&a, 1, MPI_INT, dest, 13, &b, 1, MPI_INT, source, 13, comm_2d, &status);
```

- **We send a as rank information**
- **We initialize b = 100 to check if we receive something**
- **MPI_Sendrecv sends a to dest and obtains b from source: dest and source prepared from MPI_Cart_shift()**

## MPI_Sendrecv

Sends and receives a message

### Synopsis

```c
int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                int dest, int sendtag,
                void *recvbuf, int recvcount, MPI_Datatype recvtype,
                int source, int recvtag,
                MPI_Comm comm, MPI_Status *status)
```

### Input Parameters

**sendbuf**
: initial address of send buffer (choice)

**sendcount**
: number of elements in send buffer (integer)

**sendtype**
: type of elements in send buffer (handle)

**dest**
: rank of destination (integer)

**sendtag**
: send tag (integer)

**recvcount**
: number of elements in receive buffer (integer)

**recvtype**
: type of elements in receive buffer (handle)

**source**
: rank of source (integer)

**recvtag**
: receive tag (integer)

**comm**
: communicator (handle)

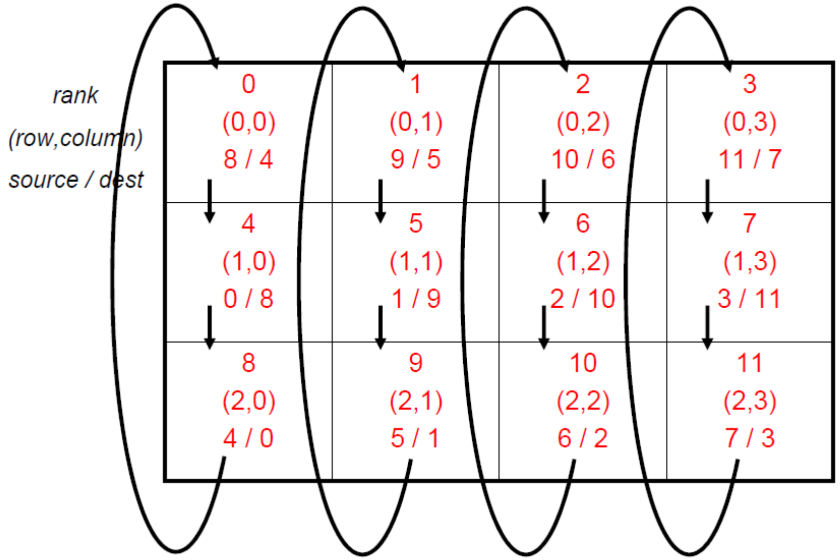
### Output Parameters

**recvbuf**
: initial address of receive buffer (choice)

**status**
: status object (Status). This refers to the receive operation.

# Cartesian Communicator Example – Create Outputs

```c
int source, dest, a, b;

MPI_Comm comm_2d;
MPI_Status status;

MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &comm_2d);

MPI_Cart_coords(comm_2d, rank, 2, coords);

MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);

a = rank; b = 100;

MPI_Sendrecv(&a, 1, MPI_INT, dest, 13, &b, 1, MPI_INT, source, 13, comm_2d, &status);

printf("rank %d source is %d\n", rank, source);
printf("rank %d dest is %d\n", rank, dest);
printf("rank %d coordinates are %d %d\n", rank, coords[0], coords[1]);

printf("rank %d send to dest = %d the value %d \n", rank, dest, a);
printf("rank %d received from source = %d the value %d \n", rank, source, b);
printf("----\n");

MPI_Finalize();
return 0;
}
```




*[2] German MPI Lecture*

- **(just!) prepares a 'shift' to neighbours down in the Grid (in this example) according cartesian setup (obtain the rank of them)**
- **Think: dimension[0] = 'direction' of the Grid – think better coordinate dimension here**
- **Think: Upwards in dimension means 'down'**
- **Coordinates are printed as well as the sending and receiving of information**

- **We send a as rank information**
- **We initialize b = 100 to check if we receive something**
- **MPI_Sendrecv sends a to dest and obtains b from source: dest and source prepared from MPI_Cart_shift()**

# Cartesian Communicator Example – Compile & Submit Batch Script & Output

```
[morris@jotunn cartesian]$ module load gnu openmpi
[morris@jotunn cartesian]$ mpicc cart-example.c -o cart-example
[morris@jotunn cartesian]$ ls -al
total 36
drwxrwxr-x  2 morris morris   121 okt  2 20:22 .
drwxrwxr-x 12 morris morris  4096 okt  2 20:21 ..
-rwxrwxr-x  1 morris morris 13025 okt  2 20:22 cart-example
-rwxr-xr-x  1 morris morris  1027 okt  2 20:07 cart-example.c
-rw-rw-r--  1 morris morris  1822 okt  2 20:05 slurm-199941.out
-rw-rw-r--  1 morris morris   599 okt  2 20:07 slurm-199942.out
-rwxr-xr-x  1 morris morris_  193 okt  2 20:20 submit-cart-example.sh
```

(the number of 12 cores is ok and fits grid dimension of communicator)

```
#!/bin/bash
#SBATCH -J cart-example
#SBATCH -n 12
#SBATCH --mail-user=morris@hi.is
#SBATCH --mail-type=end
module load gnu openmpi
mpirun /home/morris/2019-HPC-Course/cartesian/cart-example
```

```
[morris@jotunn cartesian]$ sbatch submit-cart-example.sh
Submitted batch job 199941
```

(below is an error as an example that occurs when the number of processes not match the required grid dimension of the cartesian communicator)

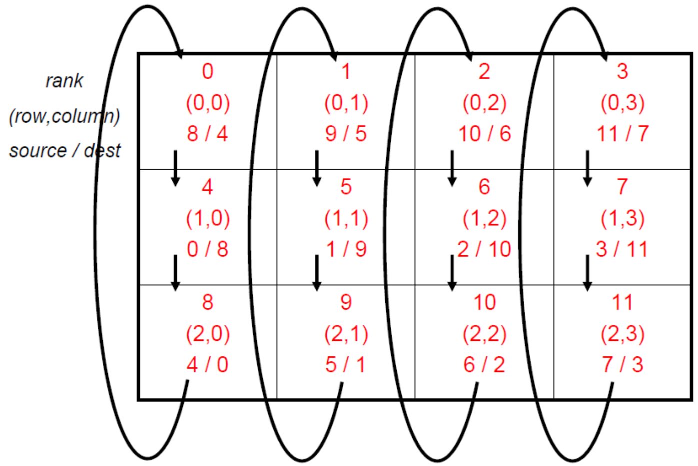
```
#!/bin/bash
#SBATCH -J cart-example
#SBATCH -n 11
#SBATCH --mail-user=morris@hi.is
#SBATCH --mail-type=end
module load gnu openmpi
mpirun /home/morris/2019-HPC-Course/cartesian/cart-example
```

```
[morris@jotunn cartesian]$ more slurm-199942.out
[compute-2-0:8696] *** An error occurred in MPI_Cart_create
[compute-2-0:8696] *** reported by process [3418685441,1]
[compute-2-0:8696] *** on communicator MPI_COMM_WORLD
[compute-2-0:8696] *** MPI_ERR_ARG: invalid argument of some other kind
[compute-2-0:8696] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[compute-2-0:8696] ***    and potentially your MPI job)
[compute-2-0:08693] 10 more processes have sent help message help-mpi-errors.txt / mpi_errors_are_fatal
[compute-2-0:08693] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
```

# Cartesian Communicator Example – Understanding Output

```
rank 0 source is 8
rank 0 dest is 4
rank 0 coordinates are 0 0
rank 0 send to dest = 4 the value 0
rank 0 received from source = 8 the value 8
----
rank 1 source is 9
rank 1 dest is 5
rank 1 coordinates are 0 1
rank 1 send to dest = 5 the value 1
rank 1 received from source = 9 the value 9
----
rank 2 source is 10
rank 2 dest is 6
rank 2 coordinates are 0 2
rank 2 send to dest = 6 the value 2
rank 2 received from source = 10 the value 10
----
rank 3 source is 11
rank 3 dest is 7
rank 3 coordinates are 0 3
rank 3 send to dest = 7 the value 3
rank 3 received from source = 11 the value 11
----
rank 4 source is 0
rank 4 dest is 8
rank 4 coordinates are 1 0
rank 4 send to dest = 8 the value 4
rank 4 received from source = 0 the value 0
----
```

```
rank 7 source is 3
rank 7 dest is 11
rank 7 coordinates are 1 3
rank 7 send to dest = 11 the value 7
rank 7 received from source = 3 the value 3
----
rank 8 source is 4
rank 8 dest is 0
rank 8 coordinates are 2 0
rank 8 send to dest = 0 the value 8
rank 8 received from source = 4 the value 4
----
rank 9 source is 5
rank 9 dest is 1
rank 9 coordinates are 2 1
rank 9 send to dest = 1 the value 9
rank 9 received from source = 5 the value 5
----
rank 10 source is 6
rank 10 dest is 2
rank 10 coordinates are 2 2
rank 10 send to dest = 2 the value 10
rank 10 received from source = 6 the value 6
----
rank 11 source is 7
rank 11 dest is 3
rank 11 coordinates are 2 3
rank 11 send to dest = 3 the value 11
rank 11 received from source = 7 the value 7
```



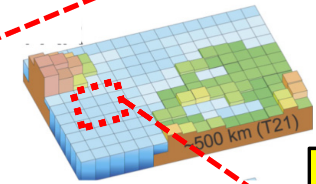

*[2] German MPI Lecture*

# Using Non-Blocking Communication with Cartesian Communicators (1)

```
#include <stdio.h>
#include <mpi.h>
#define SIZE 16
#define UP 0
#define DOWN 1
#define LEFT 2
#define RIGHT 3

int main (int argc, char** argv) {
  int numtasks, rank, source, dest, outbuf, i, tag=1;
  int inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL, MPI_PROC_NULL,
        MPI_PROC_NULL};
  int nbrs[4];
  int dims[2] = {4,4}, periods[2] = {0,0}, reorder=0;
  int coords[2];
  MPI_Comm cartcomm;
  ...
  MPI_Request reqs[8];
  MPI_Status stats[8];
   ...
  MPI_Init(&argc, &argv);
  ... // starting with MPI program...
}
```

- 'constants for numbers': offer here better code readability, not a must

- Prepares variables to be used in asynchronous communication;
- MPI_PROC_NULL indicates a 'rank' for a so-called 'dummy process'




- Prepares variables related to our 2D problem,4x4 with 4 neighbours
- Prepares variables for creating a Cartesian communicator later

- Prepares variables used for non-blocking MPI

*'simplified demo code'*

*modified from [11] MPI Tutorial*

# Using Non-Blocking Communication with Cartesian Communicators (2)

```
...
MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,
                  reorder, &cartcomm);
  MPI_Comm_rank(cartcomm, &rank);
  ...
  MPI_Cart_coords(cartcomm, rank, 2, coords);
  ...
  MPI_Cart_shift(cartcomm, 0, 1,
                 &nbrs[UP], &nbrs[DOWN] );
  MPI_Cart_shift(cartcomm, 1, 1,
                 &nbrs[LEFT], &nbrs[RIGHT] );

  printf(“rank= %d coords= %d %d“ having
    neighbours(u,d,l,r)=%d %d %d %d \n“,
    rank, coords[0], coords[1],
    nbrs[UP], nbrs[DOWN], nbrs[LEFT], nbrs[RIGHT]);

  // do some work with MPI communication operations...
  ...
  MPI_Finalize();
  return 0;
}
```

was 0

was &d

mpicc: stray \302 in program: remove invalid characters due to copy&paste

- Creates a cartesian coordinator based on our above initialized variables, here 2D → 4x4
- Obtains rank from each process, here from the cartesian communicator
- Obtains coordinate from each process from the cartesian communicator
- (just!) prepares a ‘shift’ to neighbours up and down as well as left and right according cartesian setup (obtain the rank of them)
- Prints out the neighbours with cooresponding rank

*‘simplified demo code’*

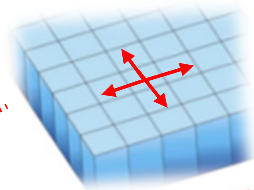
***modified from [11] MPI Tutorial***

# Using Non-Blocking Communication with Cartesian Communicators (3)

```
// 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];

  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
}
...
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;
}
```

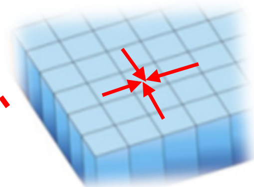
was &inbuf

- Each process has 4 neighbours so sends out 4 pieces of information and receives 4 pieces of information → 8 overall



- Loop: 4 x asynchronous communication: a non-blocking send using the 'shift' rank information indirectly via dest

- Loop: 4 x asynchronous communication: a non-blocking receive using the 'shift' rank information indirectly via source



- Synchronization: Wait for all 8 asynchronous communications to be finalized & printout data

*'simplified demo code'*

*modified from [11] MPI Tutorial*

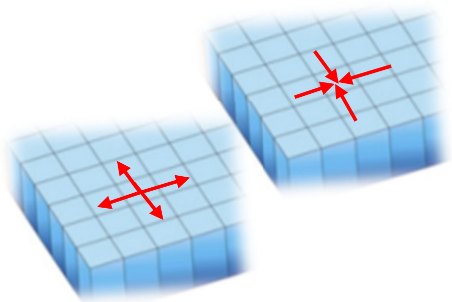
# Using Non-Blocking Communication with Cartesian Communicators (4)

```c
#include <stdio.h>
#include <mpi.h>
#define SIZE 16
#define UP 0
#define DOWN 1
#define LEFT 2
#define RIGHT 3

int main (int argc, char** argv) {
  int numtasks, rank, source, dest, outbuf, i, tag=1;
  int inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL, MPI_PROC_NULL,
        MPI_PROC_NULL};
  int nbrs[4];
  int dims[2] = {4,4}, periods[2] = {1,1}, reorder=1;
  int coords[2];
  MPI_Comm cartcomm;

  MPI_Request reqs[8];
  MPI_Status stats[8];

  // starting with MPI program
  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,reorder, &cartcomm);

  MPI_Comm_rank(cartcomm, &rank);

  MPI_Cart_coords(cartcomm, rank, 2, coords);

  MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN] );
  MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT] );

  printf("rank= %d coords= %d %d having neighbours(u,d,l,r)=%d %d %d %d \n", rank,
         coords[0], coords[1], nbrs[UP], nbrs[DOWN], nbrs[LEFT], nbrs[RIGHT]);
```

```c
  // 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;
}
```

```
[morris@jotunn cart2d]$ pwd
/home/morris/2019-HPC-Course/cart2d
[morris@jotunn cart2d]$ vi submit-cart2d.sh
[morris@jotunn cart2d]$ module load gnu openmpi
[morris@jotunn cart2d]$ mpicc cart2d.c -o cart2d
[morris@jotunn cart2d]$ ls -al
total 28
drwxrwxr-x  2 morris morris    57 okt  2 22:29 .
drwxrwxr-x 12 morris morris  4096 okt  2 20:21 ..
-rwxrwxr-x  1 morris morris 13052 okt  2 22:29 cart2d
-rwxr-xr-x  1 morris morris  1674 okt  2 19:40 cart2d.c
-rwxr-xr-x  1 morris morris   184 okt  2 22:29 submit-cart2d.sh
```

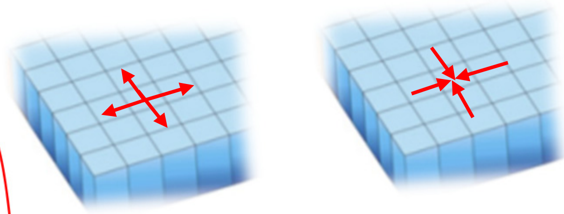
```bash
#!/bin/bash
#SBATCH -J cart2d-example
#SBATCH -n 16
#SBATCH --mail-user=morris@hi.is
#SBATCH --mail-type=end
module load gnu openmpi
mpirun /home/morris/2019-HPC-Course/cart2d/cart2d
```

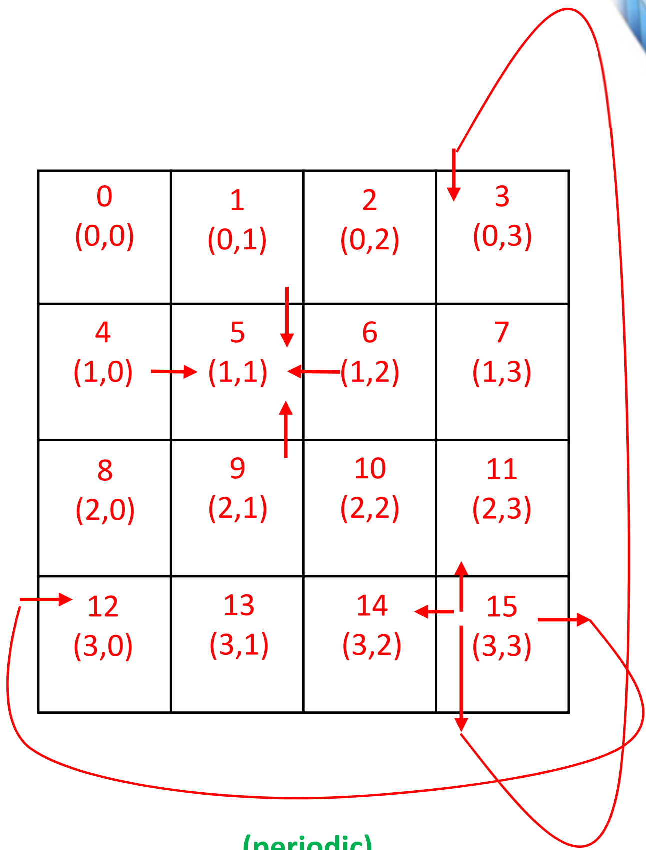


# Using Non-Blocking Communication with Cartesian Communicators (5)

```
[morris@jotunn cart2d]$ qstat
Job id              Name             Username        Time Use S Queue
------------------- ---------------- --------------- -------- - -------
199590              hello-mpi-exampl jfb3            00:00:00 Q normal
199591              hello-mpi-exampl jfb3            00:00:00 Q normal
199592              hello-mpi-exampl jfb3            00:00:00 Q normal
199593              hello-mpi-exampl jfb3            00:00:00 Q normal
199595              pingpong         jfb3            00:00:00 Q normal
199596              pingpong         jfb3            00:00:00 Q normal
199840              pingpong         jfb3            00:00:00 Q normal
199841              pingpong         jfb3            00:00:00 Q normal
199948              cart2d-example   morris          00:00:00 C normal
```

```
[morris@jotunn cart2d]$ ls -al
total 32
drwxrwxr-x  2 morris morris    80 okt  2 22:32 .
drwxrwxr-x 12 morris morris  4096 okt  2 20:21 ..
-rwxrwxr-x  1 morris morris 13052 okt  2 22:29 cart2d
-rwxr-xr-x  1 morris morris  1674 okt  2 19:40 cart2d.c
-rw-rw-r--  1 morris morris  1612 okt  2 22:32 slurm-199948.out
-rwxr-xr-x  1 morris morris   185 okt  2 22:32 submit-cart2d.sh
```

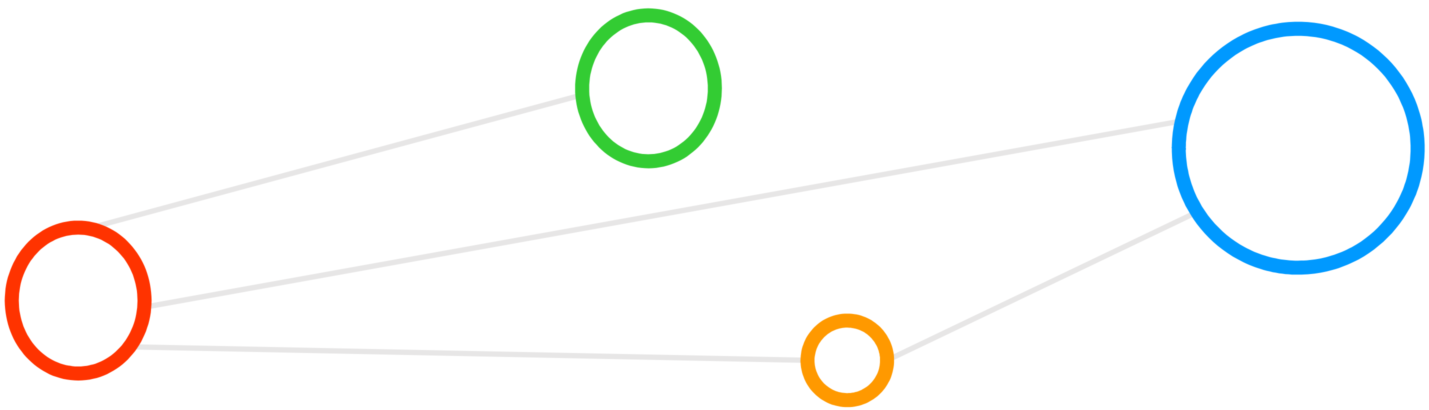





(periodic)

```
[morris@jotunn cart2d]$ more slurm-199948.out
rank= 6 coords= 1 2 having neighbours(u,d,l,r)=2 10 5 7
rank= 6 has received (u,d,l,r)= 2 10 5 7
rank= 7 coords= 1 3 having neighbours(u,d,l,r)=3 11 6 4
rank= 7 has received (u,d,l,r)= 3 11 6 4
rank= 8 coords= 2 0 having neighbours(u,d,l,r)=4 12 11 9
rank= 8 has received (u,d,l,r)= 4 12 11 9
rank= 9 coords= 2 1 having neighbours(u,d,l,r)=5 13 8 10
rank= 9 has received (u,d,l,r)= 5 13 8 10
rank= 10 coords= 2 2 having neighbours(u,d,l,r)=6 14 9 11
rank= 10 has received (u,d,l,r)= 6 14 9 11
rank= 11 coords= 2 3 having neighbours(u,d,l,r)=7 15 10 8
rank= 11 has received (u,d,l,r)= 7 15 10 8
rank= 12 coords= 3 0 having neighbours(u,d,l,r)=8 0 15 13
rank= 12 has received (u,d,l,r)= 8 0 15 13
rank= 13 coords= 3 1 having neighbours(u,d,l,r)=9 1 12 14
rank= 13 has received (u,d,l,r)= 9 1 12 14
rank= 14 coords= 3 2 having neighbours(u,d,l,r)=10 2 13 15
rank= 14 has received (u,d,l,r)= 10 2 13 15
rank= 15 coords= 3 3 having neighbours(u,d,l,r)=11 3 14 12
rank= 15 has received (u,d,l,r)= 11 3 14 12
rank= 0 coords= 0 0 having neighbours(u,d,l,r)=12 4 3 1
rank= 0 has received (u,d,l,r)= 12 4 3 1
rank= 1 coords= 0 1 having neighbours(u,d,l,r)=13 5 0 2
rank= 1 has received (u,d,l,r)= 13 5 0 2
rank= 2 coords= 0 2 having neighbours(u,d,l,r)=14 6 1 3
rank= 2 has received (u,d,l,r)= 14 6 1 3
rank= 3 coords= 0 3 having neighbours(u,d,l,r)=15 7 2 0
rank= 3 has received (u,d,l,r)= 15 7 2 0
rank= 4 coords= 1 0 having neighbours(u,d,l,r)=0 8 7 5
rank= 4 has received (u,d,l,r)= 0 8 7 5
rank= 5 coords= 1 1 having neighbours(u,d,l,r)=1 9 4 6
rank= 5 has received (u,d,l,r)= 1 9 4 6
```

# MPI Derived Data Types & Parallel I/O via HDF Examples



# MPI Derived Datatypes – MPI_Type_contigous() Example

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

(like a vector: datatype consists of a number of contiquos items of the same datatype)

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

*[12] RookieHPC*

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    MPI_Datatype double_int_type;
    MPI_Type_contiguous(2, MPI_INT, &double_int_type);
    MPI_Type_commit(&double_int_type);
    enum role_ranks { SENDER, RECEIVER };
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    switch(my_rank)
    {
        case SENDER:
        {
            int buffer_sent[2] = {12345, 67890};
            printf("MPI process %d sends values %d and %d.\n",
                   my_rank, buffer_sent[0], buffer_sent[1]);
            MPI_Send(&buffer_sent, 1, double_int_type, RECEIVER, 0, MPI_COMM_WORLD);
            break;
        }
        case RECEIVER:
        {
            int received[2];
            MPI_Recv(&received, 1, double_int_type, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("MPI process %d received values: %d and %d.\n", my_rank, received[0], received[1]);
            break;
        }
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

```
[morris@jotunn derived-contigous]$ ls -al
total 28
drwxrwxr-x  2 morris morris  113 okt  3 07:35 .
drwxrwxr-x 15 morris morris 4096 okt  3 06:42 ..
-rwxrwxr-x  1 morris morris 8832 okt  3 07:35 derived-contigous
-rw-rw-r--  1 morris morris 1045 okt  3 07:33 derived-contigous.c
-rw-rw-r--  1 morris morris   92 okt  3 07:35 slurm-199953.out
-rwxr-xr-x  1 morris morris  208 okt  3 07:35 submit-derived-contigous.sh
[morris@jotunn derived-contigous]$ more slurm-199953.out
MPI process 0 sends values 12345 and 67890.
MPI process 1 received values: 12345 and 67890.
```

# MPI Derived Datatypes – MPI_Type_vector() Example

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

```
     Full array            What we want
                              to send
  +-----+-----+-----+   +-----+-----+-----+
  |  0  |  1  |  2  |   |  -  |  1  |  -  |
  +-----+-----+-----+   +-----+-----+-----+
  |  3  |  4  |  5  |   |  -  |  4  |  -  |
  +-----+-----+-----+   +-----+-----+-----+
  |  6  |  7  |  8  |   |  -  |  7  |  -  |
  +-----+-----+-----+   +-----+-----+-----+

  How to extract a column with a vector type:

                   distance between the
               start of each block: 3 elements
             <---------------> <--------------->
             |                 |                 |
          start of          start of          start of
          block 1           block 2           block 3
             |                 |                 |
             V                 V                 V
    +-----+-----+-----+-----+-----+-----+-----+-----+-----+
    |  -  |  1  |  -  |  -  |  4  |  -  |  -  |  7  |  -  |
    +-----+-----+-----+-----+-----+-----+-----+-----+-----+
          <--->             <--->             <--->
         block 1           block 2           block 3

  Block length: 1 element
  Element: MPI_INT
```

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

***[12] RookieHPC***

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    enum rank_roles { SENDER, RECEIVER };
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    switch(my_rank)
    {
        case SENDER:
        {
            MPI_Datatype column_type;
            MPI_Type_vector(3, 1, 3, MPI_INT, &column_type);
            MPI_Type_commit(&column_type);
            int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
            MPI_Request request;
            printf("MPI process %d sends values %d, %d and %d.\n",
                   my_rank, buffer[0][1], buffer[1][1], buffer[2][1]);
            MPI_Send(&buffer[0][1], 1, column_type, RECEIVER, 0, MPI_COMM_WORLD);
            break;
        }
        case RECEIVER:
        {
            int received[3];
            MPI_Recv(&received, 3, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("MPI process %d received values: %d, %d and %d.\n",
                   my_rank, received[0], received[1], received[2]);
            break;
        }
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

```
[morris@jotunn derived-vector]$ ls -al
total 28
drwxrwxr-x  2 morris morris  104 okt  3 06:55 .
drwxrwxr-x 15 morris morris 4096 okt  3 06:42 ..
-rwxrwxr-x  1 morris morris 8825 okt  3 06:54 derived-vector
-rw-rw-r--  1 morris morris 1124 okt  3 06:52 derived-vector.c
-rw-rw-r--  1 morris morris   82 okt  3 06:55 slurm-199951.out
-rwxr-xr-x  1 morris morris  199 okt  3 06:54 submit-derived-vector.sh
[morris@jotunn derived-vector]$ more slurm-199951.out
MPI process 0 sends values 1, 4 and 7.
MPI process 1 received values: 1, 4 and 7.
```

# MPI Derived Datatypes – MPI_Type_indexed() Example

```
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);
```


```
  Full array          What we want
                         to send

+-----+-----+-----+  +-----+-----+-----+
|  0  |  1  |  2  |  |  0  |  -  |  -  |
+-----+-----+-----+  +-----+-----+-----+
|  3  |  4  |  5  |  |  3  |  4  |  -  |
+-----+-----+-----+  +-----+-----+-----+
|  6  |  7  |  8  |  |  6  |  7  |  8  |
+-----+-----+-----+  +-----+-----+-----+
```

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

```
How to extract the lower triangle with an indexed type:


       +---------------------------- displacement for
       |                             block 2: 6 elements
       |                                   |
       +---------- displacement for        |
       |           block 2: 3 elements     |
       |                 |                 |
 displacement for        |                 |
block 1: 0 elements      |                 |
       |                 |                 |
       V                 V                 V
       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
       |  0  |  -  |  -  |  3  |  4  |  -  |  6  |  7  |  8  |
       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
        <--->             <--------->       <--------------->
       block 1            block 2             block 3
      1 element          2 elements          3 elements

Element: MPI_INT
```

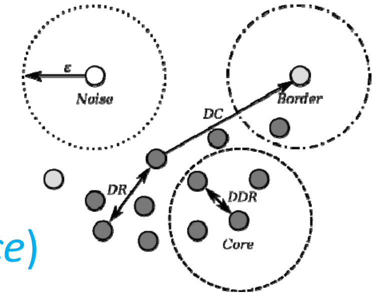
*[12] RookieHPC*

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    enum rank_roles { SENDER, RECEIVER };
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    switch(my_rank)
    {
        case SENDER:
        {
            MPI_Datatype triangle_type;
            int lengths[3] = { 1, 2, 3 };
            int displacements[3] = { 0, 3, 6 };
            MPI_Type_indexed(3, lengths, displacements, MPI_INT, &triangle_type);
            MPI_Type_commit(&triangle_type);
            int buffer[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
            MPI_Request request;
            printf("MPI process %d sends values:\n%d\n%d %d\n%d %d %d\n", my_rank,
                   buffer[0][0], buffer[1][0], buffer[1][1], buffer[2][0], buffer[2][1], buffer[2][2]);
            MPI_Send(buffer, 1, triangle_type, RECEIVER, 0, MPI_COMM_WORLD);
            break;
        }
        case RECEIVER:
        {
            int received[6];
            MPI_Recv(&received, 6, MPI_INT, SENDER, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("MPI process %d received values:\n%d\n%d %d\n%d %d %d\n",
                   my_rank, received[0], received[1], received[2], received[3], received[4], received[5]);
            break;
        }
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

```
[morris@jotunn derived-indexed]$ ls -al
total 28
drwxrwxr-x  2 morris morris  107 okt  3 07:10 .
drwxrwxr-x 15 morris morris 4096 okt  3 06:42 ..
-rwxrwxr-x  1 morris morris 8827 okt  3 07:10 derived-indexed
-rw-rw-r--  1 morris morris 1349 okt  3 07:08 derived-indexed.c
-rw-rw-r--  1 morris morris   83 okt  3 07:10 slurm-199952.out
-rwxr-xr-x  1 morris morris  202 okt  3 07:10 submit-derived-indexed.sh
[morris@jotunn derived-indexed]$ more slurm-199952.out
MPI process 0 sends values:
0
3 4
6 7 8
MPI process 1 received values:
0
3 4
6 7 8
```

# Data Science Example: DBSCAN Clustering Algorithm – Revisited (cf. Lecture 5)

- 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*)
- 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




(MinPoints = 4)

(DR = Density Reachable)

(DDR = Directly Density Reachable)

(DC = Density Connected)

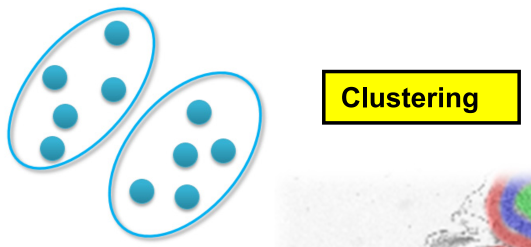
*[7] Ester et al.*

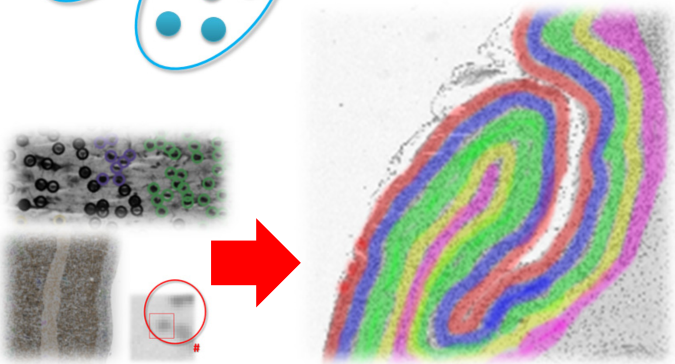
```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=ml-hpc-1

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
```

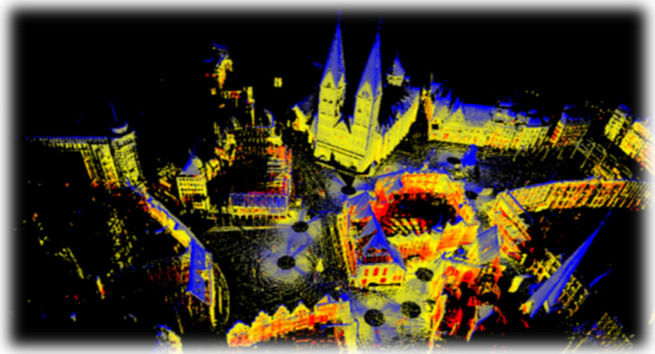
- **Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm that requires only two parameters and has no requirement to specify number of clusters**
- **Parameter Epsilon: Algorithm looks for a similar point within a given search radius Epsilon**
- **Parameter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

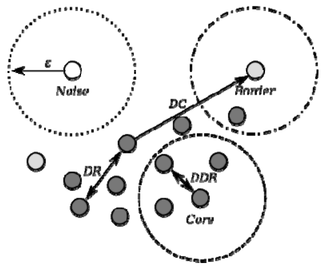
➢ **Lecture 8 provides more details about using MPI and OpenMP for data science algorithms used in clustering and classification of data**

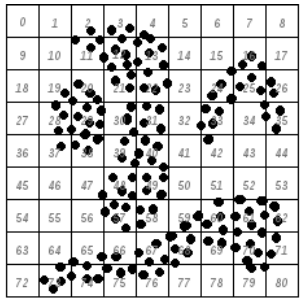
# ‘Big Data‘ Science Example – Parallel & Scalable Clustering Algorithm – Revisited



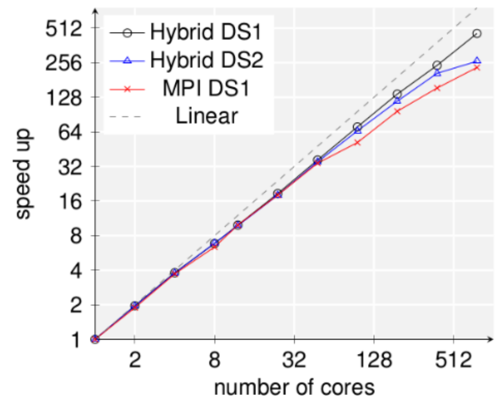



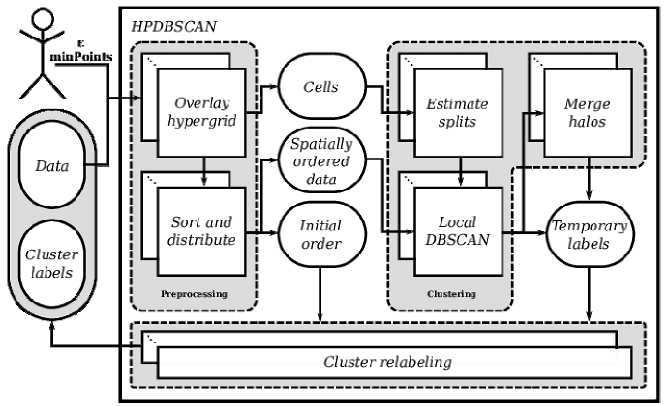


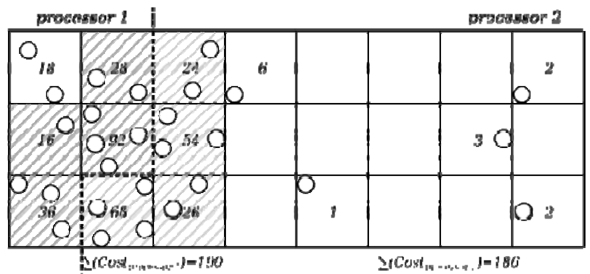


Overlayed spatial grid



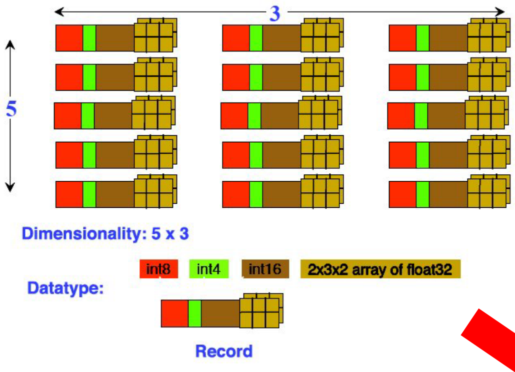



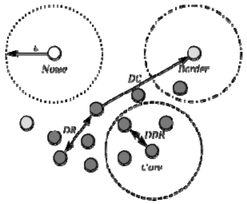

*[8] M. Goetz and M. Riedel et al,*
*Proceedings IEEE Supercomputing Conference, 2015*

# High-Level I/O Hierarchical Data Format (HDF) for Data Structures – Revisited

- 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)

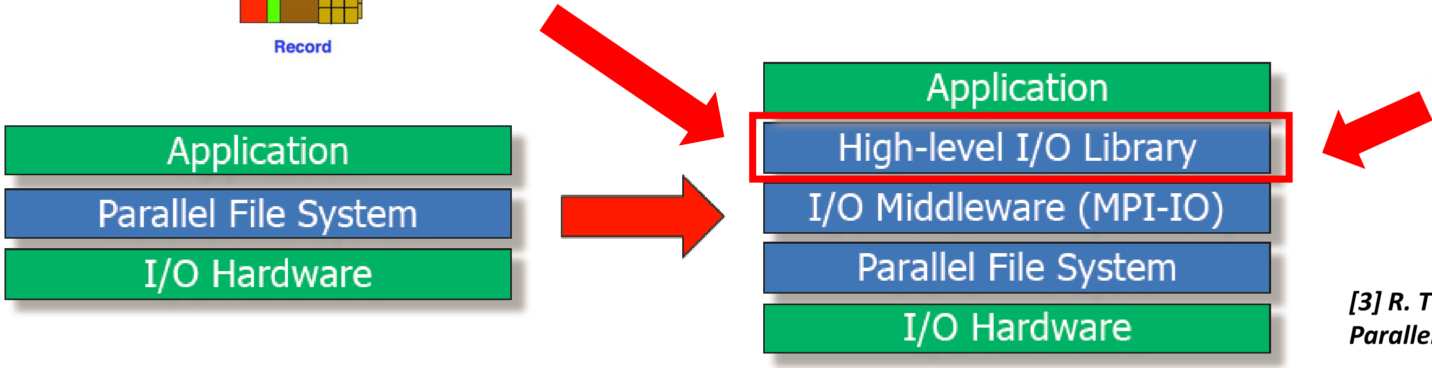


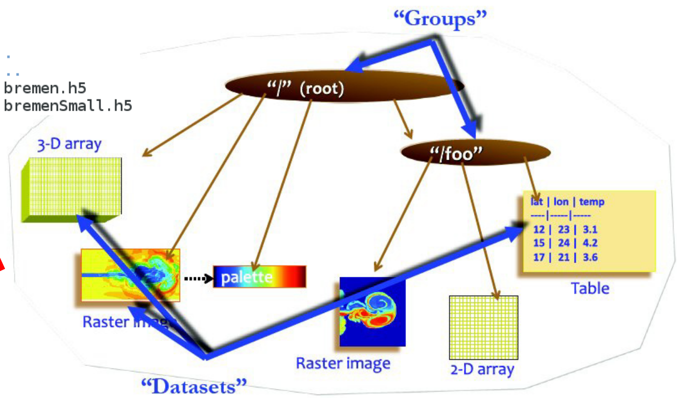

(application example parallel & scalable clustering with HPDBSCAN using Bremen data in HDF5)



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



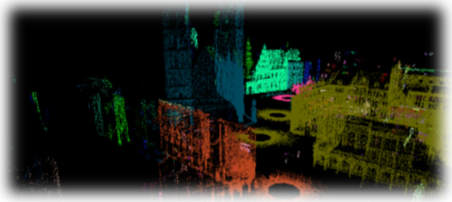




*[3] R. Thakur, PRACE Training, Parallel I/O and MPI I/O*

*[9] HDF@ I/O workshop*

# High-Level I/O Hierarchical Data Format (HDF) for Data Structures & H5Dump

- Point cloud data stored in HDF5 → create a local copy → read/write!
  - Binary file format → not normal text, e.g., using more fails, but h5dump works



```
[morris@jotunn hpdbscan]$ pwd
/home/morris/2019-HPC-Course/hpdbscan
[morris@jotunn hpdbscan]$ ls -al
total 1342196
drwxrwxr-x  2 morris morris       4096 okt  2 13:16 .
drwxrwxr-x 12 morris morris       4096 okt  2 20:21 ..
-rwxr-xr-x  1 morris morris 1302382632 okt  2 13:16 bremen.h5
-rwxr-xr-x  1 morris morris   72002416 okt  2 20:40 bremenSmall.h5
-rw-rw-r--  1 morris morris          0 okt  2 13:13 HPDBSCAN-199934.err
-rw-rw-r--  1 morris morris        490 okt  2 13:14 HPDBSCAN-199934.out
-rw-rw-r--  1 morris morris          0 okt  2 13:14 HPDBSCAN-199935.err
-rw-rw-r--  1 morris morris        492 okt  2 13:17 HPDBSCAN-199935.out
-rwxr-xr-x  1 morris morris        535 okt  2 13:14 submit-clustering-bremen.sh
```

```
[morris@jotunn hpdbscan]$ h5dump
usage: h5dump [OPTIONS] files
  OPTIONS
     -h,   --help         Print a usage message and exit
     -V,   --version      Print version number and exit
--------------- File Options ---------------
     -n,   --contents     Print a list of the file contents and exit
                          Optional value 1 also prints attributes.
     -B,   --superblock   Print the content of the super block
     -H,   --header       Print the header only; no data is displayed
     -f D, --filedriver=D Specify which driver to open the file with
     -o F, --output=F     Output raw data into file F
     -b B, --binary=B     Binary file output, of form B
     -O F, --ddl=F        Output ddl text into file F
                          Do not use filename F to suppress ddl display
```

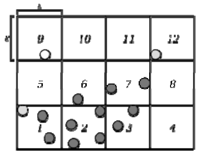
```
[morris@jotunn hpdbscan]$ more bremenSmall.h5
▒HDF




D<P▒EAZ▒▒t▒▒BC▒*▒▒V\E▒b&D▒fF▒▒▒▒▒▒×|▒D▒▒g▒+▒▒À▒▒@▒▒▒▒▒▒
ts▒a3▒▒QE▒▒▒D^LB▒E켧 ▒@^L▒D▒W▒▒R▒▒▒0M▒3a3EC.▒▒0▒*▒>0<▒▒+"▒▒7▒▒▒▒▒-Ń▒E▒/E▒ЍCD▒▒f▒+
```

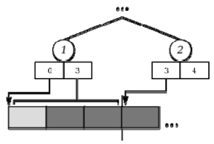
```
[morris@jotunn hpdbscan]$ h5dump -n bremenSmall.h5
HDF5 "bremenSmall.h5" {
FILE_CONTENTS {
 group      /
 dataset    /COLORS
 dataset    /Clusters
 dataset    /DBSCAN
 }
}
```

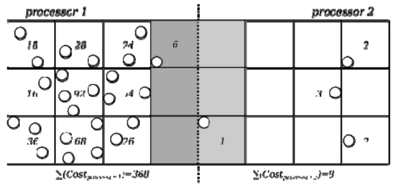
*[10] h5Dump Tool Description*

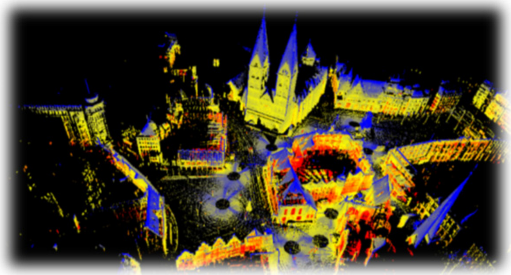
# ‘Big Data‘ Science Example: Using High-Level I/O Hierarchical Data Format (HDF)

- 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

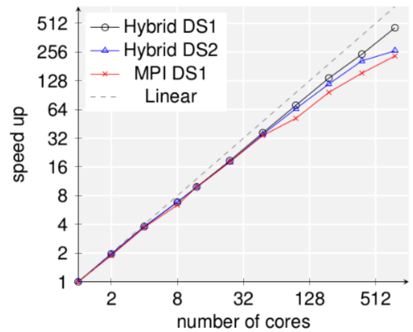






```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=ml-hpc-1

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
```

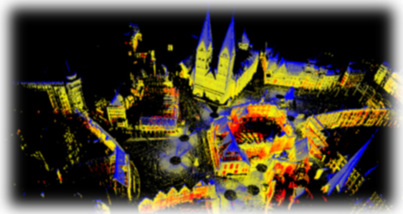



*[8] M. Goetz and M. Riedel et al, Proceedings IEEE Supercomputing Conference, 2015*

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

# HPDBSCAN Clustering of Point Cloud Data Set Bremen on Jötunn HPC System

- Submit
  - Using your own copy of datasets in HDF5 format (read & write!), /Clusters are 0
  - Using installed modules for HPDBSCAN
  - Using typical batch system script specifying DBSCAN parameters



```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH -n 4

# load modules
module load gnu/5.3.0
module load hdf5/1.8.17
module load openmpi/1.10.2
module load HPDBSCAN/mpi

# executable
HPDBSCAN=dbscan

# your own copy of bremen small
BREMENSMALLDATA=/home/morris/2019-HPC-Course/hpdbscan/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/home/morris/2019-HPC-Course/hpdbscan/bremen.h5

mpirun $HPDBSCAN -m 300 -e 500 $BREMENSMALLDATA
```

```
[morris@jotunn hpdbscan]$ h5dump -d /Clusters bremenSmall.h5
HDF5 "bremenSmall.h5" {
DATASET "/Clusters" {
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( 3000000 ) / ( 3000000 ) }
   DATA {
   (0): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (23): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (46): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (69): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (92): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (115): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
[morris@jotunn hpdbscan]$ sbatch submit-clustering-bremen.sh
Submitted batch job 199947
[morris@jotunn hpdbscan]$ qstat
Job id             Name             Username        Time Use S Queue
------------------ ---------------- --------------- -------- - ---------------
199590             hello-mpi-exampl jfb3            00:00:00 Q normal
199591             hello-mpi-exampl jfb3            00:00:00 Q normal
199592             hello-mpi-exampl jfb3            00:00:00 Q normal
199593             hello-mpi-exampl jfb3            00:00:00 Q normal
199595             pingpong         jfb3            00:00:00 Q normal
199596             pingpong         jfb3            00:00:00 Q normal
199840             pingpong         jfb3            00:00:00 Q normal
199841             pingpong         jfb3            00:00:00 Q normal
199947             HPDBSCAN         morris          00:00:00 R normal
```

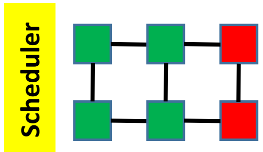
***[5] Icelandic HPC Machines & Community***

# HPDBSCAN Clustering – Understanding Two Outputs: Text & HDF5 File

```
[morris@jotunn hpdbscan]$ more HPDBSCAN-199947.out
Calculating Cell Space...
        Computing Dimensions... [OK] in 0.054227
        Computing Cells...      [OK] in 0.022971
        Sorting Points...       [OK] in 0.133213
        Distributing Points...  [OK] in 0.111046
DBSCAN...
        Local Scan...           [OK] in 139.057375
        Merging Neighbors...    [OK] in 0.010089
        Adjust Labels ...       [OK] in 0.010476
        Rec. Init. Order ...    [OK] in 0.556270
        Writing File ...        [OK] in 0.170748
Result...
        21      Clusters
        2974394 Cluster Points
        25606   Noise Points
        2949094 Core Points
Took: 140.453795s
```

```
[morris@jotunn hpdbscan]$ h5dump -d /Clusters bremenSmall.h5
HDF5 "bremenSmall.h5" {
DATASET "/Clusters" {
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( 3000000 ) / ( 3000000 ) }
   DATA {
   (0): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (23): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (46): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (69): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (92): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   (115): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

- The input data for the parallel & scalable HPDBSCAN clustering algorithm is a HDF5 file and all the processors read in parallel chunks of the data
- The HDF5 file before the execution of HPDBSCAN has 0 as Cluster Ids for its specific initialization




Jötunn compute nodes

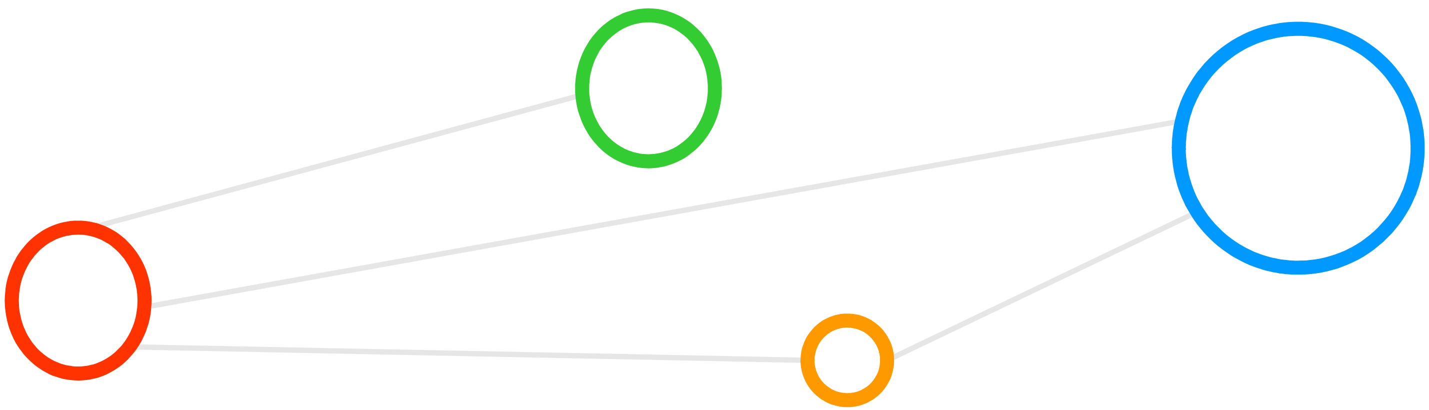
```
[morris@jotunn hpdbscan]$ h5dump -d /Clusters bremenSmall.h5
HDF5 "bremenSmall.h5" {
DATASET "/Clusters" {
   DATATYPE  H5T_STD_I64LE
   DATASPACE  SIMPLE { ( 3000000 ) / ( 3000000 ) }
   DATA {
   (0): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (8): -45205, -45205, -45205, -45205, -45205, 4825, -45205, -45205, -45205,
   (17): -45205, -4108, -45205, -45205, -45205, -45205, -45205, -45205,
   (25): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (33): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (41): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (49): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (57): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (65): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (73): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (81): -45205, -45205, -45205, -490063, -45205, -45205, -45205, -45205,
   (89): -45205, -45205, -45205, -45205, -45205, -45205, -45205, 45205,
   (97): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (105): -45205, -45205, -45205, -45205, -45205, -45205, -45205, -45205,
   (113): -45205, -45205, -45205, -45205, -45205, -45205, -490063, -45205,
```

- The standard out of the HPDBSCAN parallel & scalable DBSCAN clustering algorithm is not the result of the DBSCAN clustering algorithm and only shows meta information such as the numbers of clusters found, noise, and running time

- The real outcome of the parallel & scalable HPDBSCAN algorithm is directly written into the HDF5 file assigning for each point cloud data element a specific cluster ID, or using minus numbers to indicate noise points (no real clusters)

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

# Lecture Bibliography



# Lecture Bibliography


- [1] M. Geimer et al., ‘SCALASCA performance properties: The metrics tour’
- [2] German Lecture ‘Umfang von MPI 1.2 und MPI 2.0‘
- [3] Rajeev Thakur, Parallel I/O and MPI-IO, Online: http://www.training.prace-ri.eu/uploads/tx_pracetmo/pio1.pdf
- [4] Non-Blocking Communication Example, Online: http://beige.ucs.indiana.edu/B673/node153.html
- [5] Icelandic HPC Machines & Community, Online: http://ihpc.is
- [6] LLNL MPI Tutorial, Online: https://computing.llnl.gov/tutorials/mpi/
- [7] 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
- [8] 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
- [9] Michael Stephan,‘Portable Parallel IO - ‘Handling large datasets in heterogeneous parallel environments‘, Online: http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/parallelio-2014/parallel-io-hdf5.pdf?__blob=publicationFile
- [10] H5Dump Tool, Online: https://support.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Dump
- [11] Blaise Barney, Lawrence Livermore National Laboratory, ‘MPI Tutorial’, Online: https://computing.llnl.gov/tutorials/mpi/
- [12] Rookie HPC MPI, Online: https://www.rookiehpc.com/mpi/index.php

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