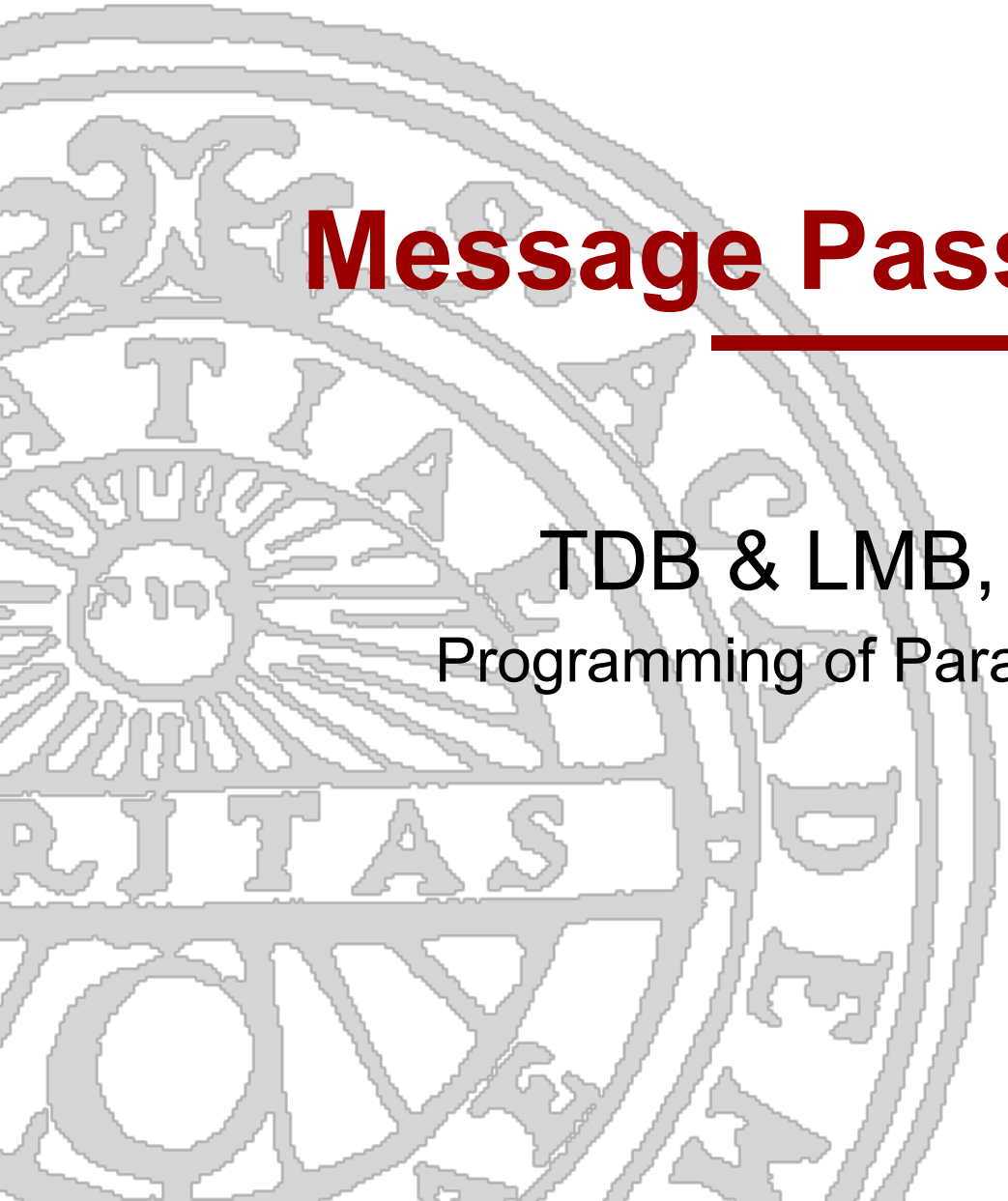


MPI

Message Passing Interface

Jing Liu

TDB & LMB, Uppsala University

Programming of Parallel Computers, Jan 2016

UPPSALA UNIVERSITET

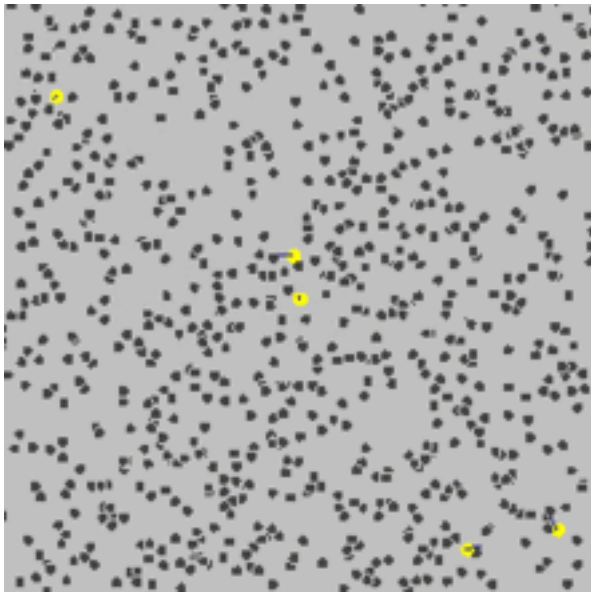
Dep. of Information Technology

MPI -- 1

- communicator & group
- asynchronous VS synchronous
- 4 modes of blocking send
- non-blocking send/ receive calls
- MPI wait/ test/ probe/ cancel functions

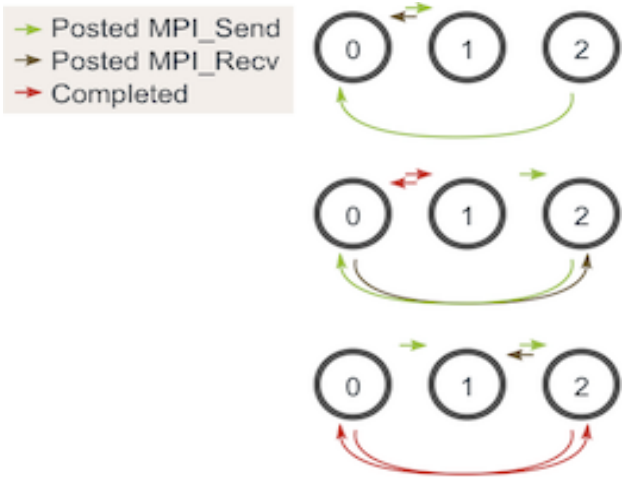
P2P case study

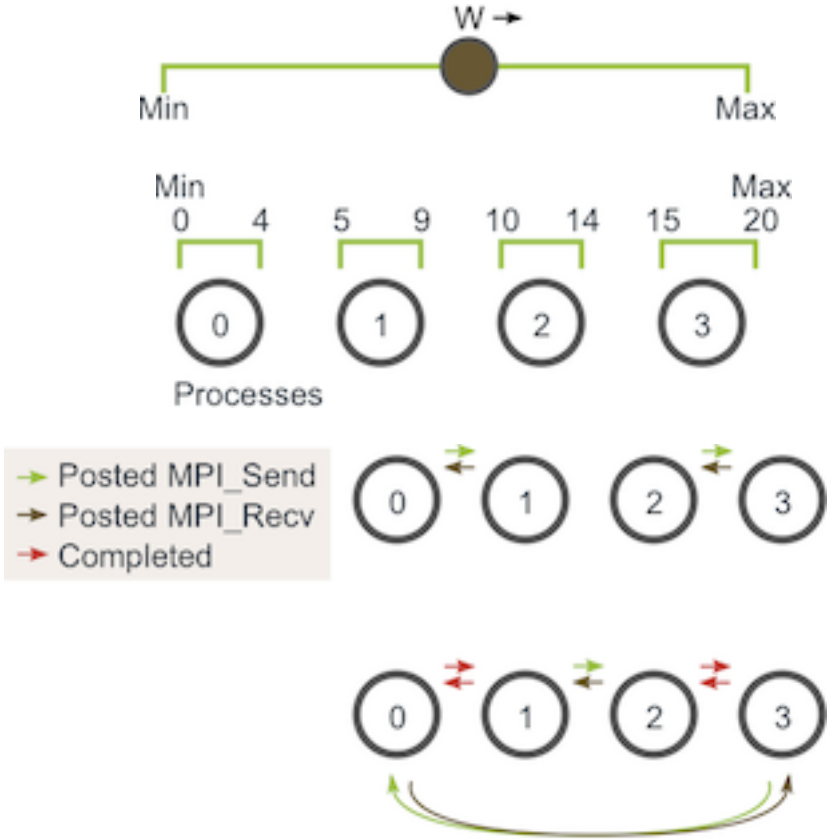
- Random walk and particle tracing
  - Particle moves randomly in a fluid – Brownian motion
  - Performing parallel particle tracing can be very difficult
    - Randomly move among grids
    - Unbalanced computation
  - Random walk



P2P case study

- A simple random walk in MPI
  - MPI_Send
  - MPI_Recv
  - MPI_Probe





UPPSALA UNIVERSITET

Dep. of Information Technology

Outline

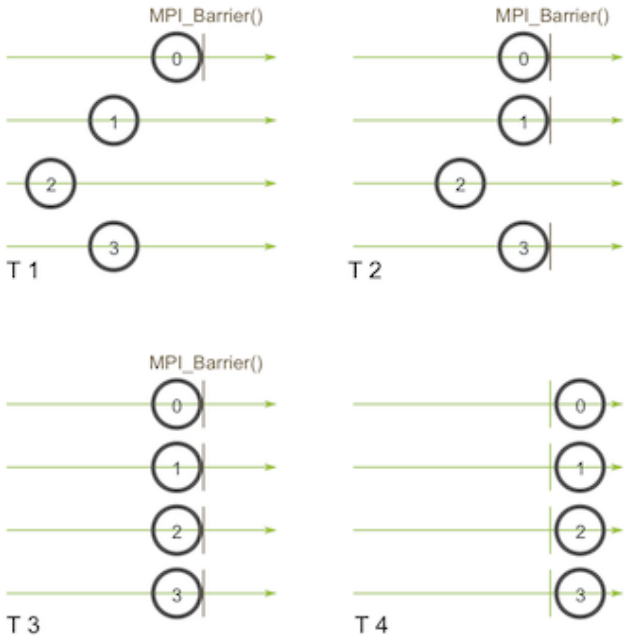
- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Global functions

- Global functions act on every process in a group (communicator).
  - Barrier: all process wait
  - Broadcast: send to all
  - Reduce: collect data
  - Scatter: send to all
  - Gather: receive from all

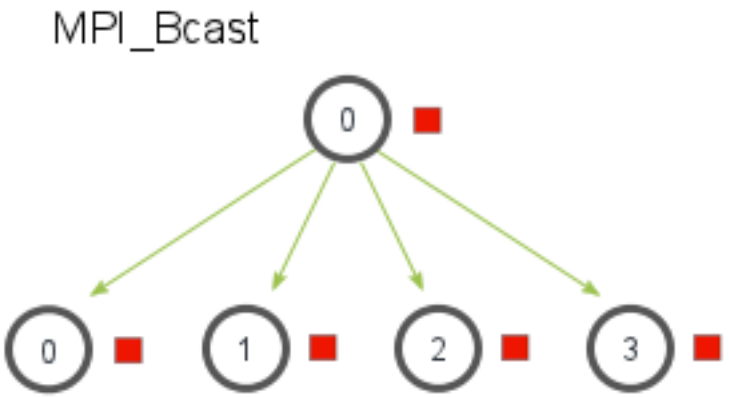
Global functions (cont.)

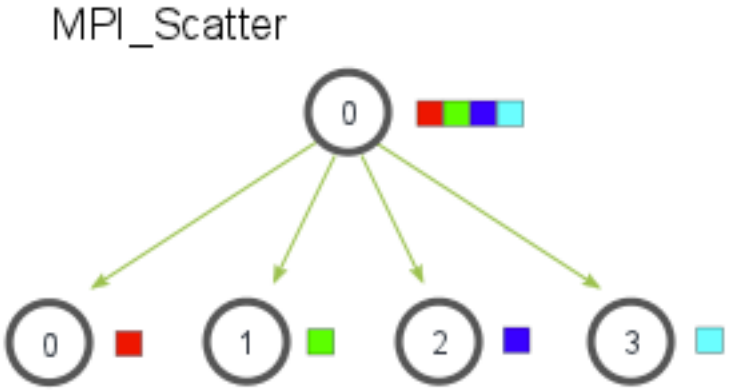
- int MPI Barrier(MPI_Comm comm);
  - Wait until every processes post this function



Global functions(cont.)

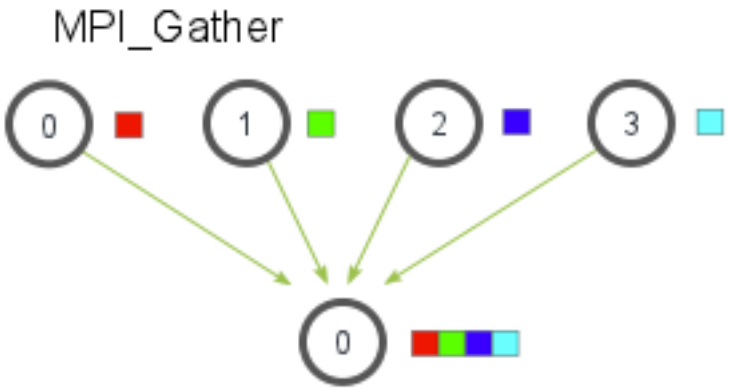
- Broadcast
  - int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm);
- Scatter
  - int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int rcount, MPI_Datatype recvtype, int root, MPI_Comm comm);





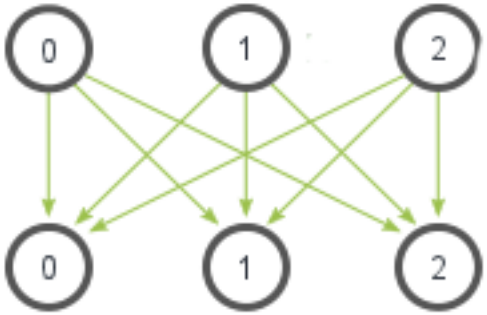
Global functions(cont.)

- MPI_Gather
  - Inverse of MPI_Scatter
  - Highly useful for parallel sorting and searching
- int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int rcount, MPI_Datatype recvtype, int root, MPI_Comm comm);



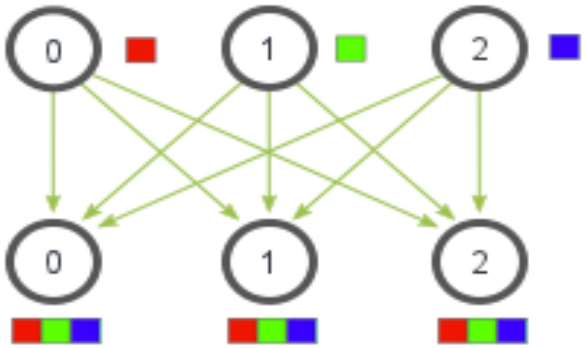
Global functions(cont.)

- MPI_All*
  - Many-to-Many communication pattern
    - MPI_Allreduce
    - MPI_Allgather
    - MPI_Alltoall

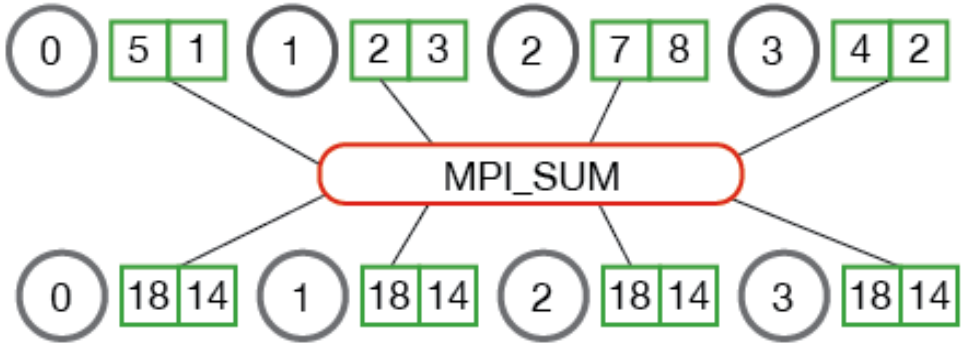


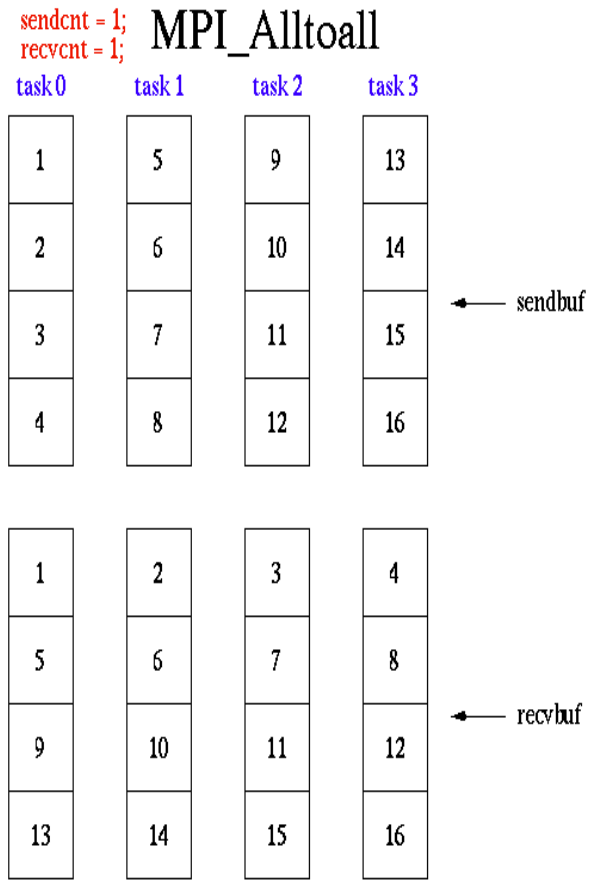
Global functions(cont.)

MPI_Allgather



MPI_Allreduce





Global functions(cont.)

- For the MPI_*reduce
  - MPI_MAX – Returns the maximum element.
  - MPI_MIN – Returns the minimum element.
  - MPI_SUM – Sums the elements.
  - MPI_PROD – Multiplies all elements.
  - MPI_LAND – Performs a logical “and” across the elements.
  - MPI_LOR – Performs a logical “or” across the elements.
  - MPI_BAND – Performs a bitwise “and” across the bits of the elements.
  - MPI_BOR – Performs a bitwise “or” across the bits of the elements.
  - MPI_MAXLOC – Returns the maximum value and the rank of the process that owns it.
  - MPI_MINLOC – Returns the minimum value and the rank of the process that owns it.

Global functions—case study

- Some optimization is done with the global functions
- Compare MPI_Bcast to for loop Send/ Recv
- mpirun -n 16 ./DemoBcast2 1000000 10

  Data size = 4000000 bytes, Trials = 10 times

  Avger SR_bcast time = 0.015581 seconds

  Avg MPI_Bcast time = 0.004403 seconds

UPPSALA UNIVERSITET

Dep. of Information Technology

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

UPPSALA UNIVERSITET

Dep. of Information Technology

Datatypes

MPI datatypeÅ	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED	unsigned int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_PACKED	-
MPI_BYTE	-

Datatypes (cont.)

- Derived Datatype
  - **MPI_Type_contiguous**
    - Produces a new datatype by making count copies of an existing data type.
  - **MPI_Type_vector**
    - Similar to contiguous, but allows for regular gaps (stride) in the displacements.
  - **MPI_Type_indexed**
    - An array of displacements of the input data type is provided as the map for the new data type.

Datatypes (cont.)

- Derived Datatype (cont.)
  - **MPI_Type_struct**
    - The most general of all derived datatypes. The new data type is formed according to completely defined map of the component data types.
- Allocate / de-allocate datatype
  - int MPI_Type_commit (MPI_datatype *datatype)
  - int MPI_Type_free (MPI_datatype *datatype)

Read more at Derived Data Types with MPI

Datatypes (cont.)

- int MPI_Type_contiguous(int count, MPI_Datatype old_type, MPI_Datatype *new_type_p);
  - MPI_Datatype newtype;
  - MPI Type vector(15,MPI_INT,&newtype);
  - MPI_Commit(&newtype);
  - MPI_Send(&A[0],1,newtype,1,123,comm);
  - MPI_Recv(B,1,newtype, 0, 123,comm, &status);
  - if A is an array from 1 to 20, what B will be? 1~15

Datatypes (cont.)

- int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)
  - MPI_Datatype newtype;
  - MPI Type vector(3,2,4,MPI_INT,&newtype);
  - MPI_Commit(&newtype);
  - MPI_Send(&A[0][1],1,newtype,1,0,comm)

1	2	3	4
5	6	7	8
9	10	11	12

- Sends new array [2 3 6 7 10 11] to process 1
  - count * blocklength = # of elements to be sent

Datatypes (cont.)

- int MPI_Type_indexed(int count, int blocklens[], int indices[], MPI_Datatype old_type, MPI_Datatype *newtype);
  - blocklens = {1,2,3}
  - indices = {4,5,6}
  - A is an array from 0 to 20
  - S sends MPI_Send(&A[10],1,type,…)
  - R calls MPI_Recv(B,1,type…) / MPI_Recv(B, 6,MPI_INT,…)

Datatypes (cont.)

- int MPI_Type_indexed(int count, const int *blocklengths, const int *disp, MPI_Datatype oldtype, MPI_Datatype *newtype)

- int MPI_Type_struct(int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)
  - read example using MPI_Type_struct here

UPPSALA UNIVERSITET

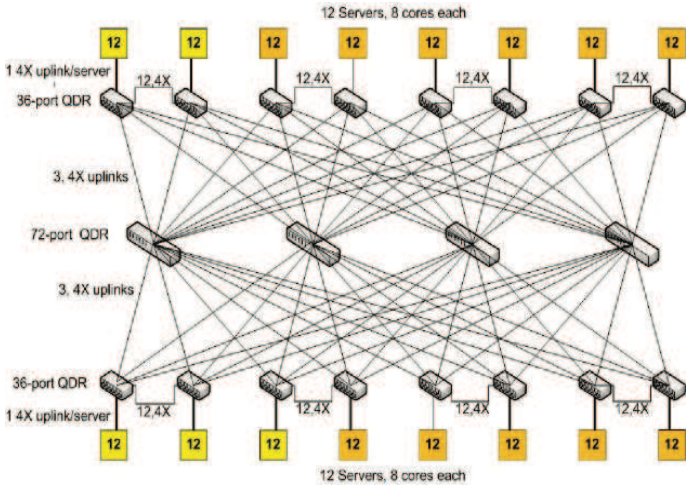
Dep. of Information Technology

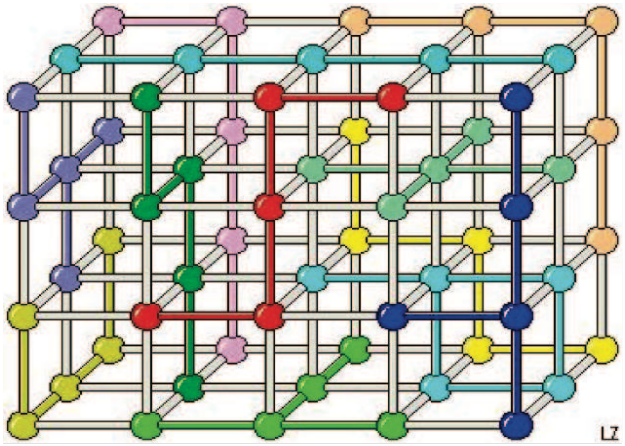
Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Topology

- Virtual Topology VS physical topology
  - Use different communicator
  - Optimizing your algorithm communications





Topology (cont)

- Cartesian Topology

int MPI_Cart_create (MPI_Comm commold , int ndims , int *dims , int * periods, int reorder, MPI_Comm *commcart)

  - Creates a new communicator with Cartesian topology of arbitrary dimension
    - commold input communicator
    - ndims number of dimensions of Cartesian grid
    - dims 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 of initial processes may be reordered (true) or not (false)
    - comm cart communicator with new Cartesian topology

Topology (cont)

- old_comm = MPI_COMM_WORLD;
- ndims = 2;
- dim_size[0] = 3;
- dim_size[1] = 2;
- periods[0] = 0;
- periods[1] = 0;
- reorder = 0;
- MPI_Cart_create(old_comm, ndims, dim_size,periods, reorder, &new_comm);

UPPSALA UNIVERSITET

Dep. of Information Technology

Topology (cont)

- MPI_Cart_rank
  - Determines process rank in communicator given Cartesian location
- MPI_Cart_coords
  - Determines process coordinates in Cartesian topology given rank in group
- Demo time!!

UPPSALA UNIVERSITET

Dep. of Information Technology

MPI TIMING

- MPI_Wtime returns an elapsed time (in second) on the calling processor

```
double start_time;
start_time = MPI_Wtime();
...
// Compute
...
double finish_time;
finish_time = MPI_Wtime();
// Elapsed time
double elapsed_time;
elapsed_time = finish_time - start_time;
```

UPPSALA UNIVERSITET

Dep. of Information Technology

Read more on:

- MPI APIs and examples
- MPICH official website
- OpenMPI v1.8