UPPSALA
UNIVERSITET

Distributed Computing - MPI

Dimitar Lukarski, Jarmo Rantakokko

Division of Scientific Computing
Department of Information Technology
Uppsala Programming for Multicore Architectures Research Center
(UPMARC)
Uppsala University

Programming of Parallel Computers, Jan, 2014

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

What You Should Know...

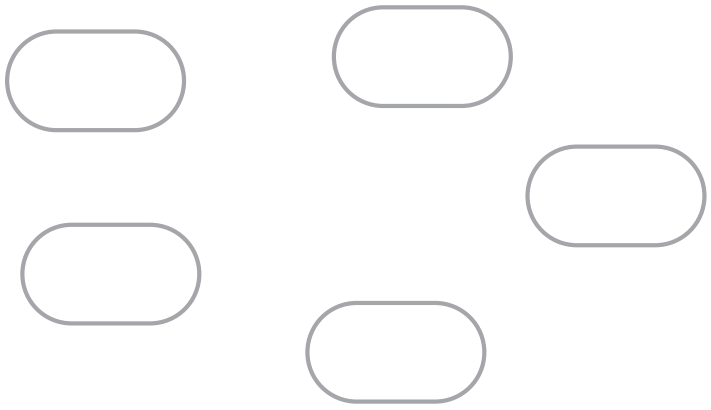
- Programming in FORTRAN/C/C++/Java
- Basics in hardware - CPU, RAM, Network

UPPSALA
UNIVERSITET

Outline

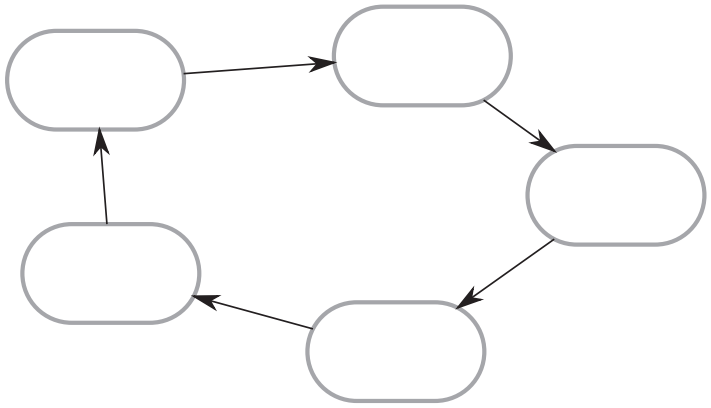
UPPSALA
UNIVERSITET

Distributed computing



UPPSALA
UNIVERSITET

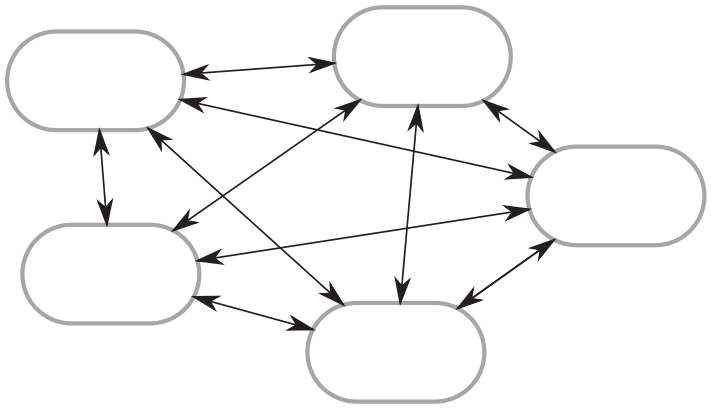
Distributed computing



UPPSALA
UNIVERSITET

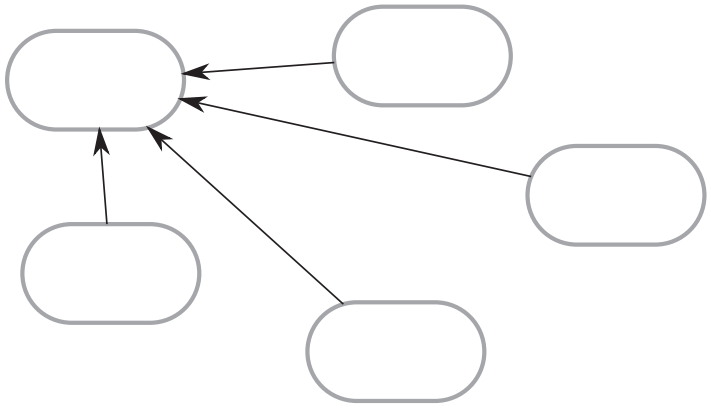
Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Distributed computing



UPPSALA
UNIVERSITET

Distributed computing



UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Distributed computing

How to communicate?

- Sending data
- Receiving data
- Waiting for data
- Waiting for synchronization

UPPSALA
UNIVERSITET

Message Passing Interface (MPI)

- MPI is a library for data and messages passing
- The communication is explicit (the user specify what/how needs to be sent)
- The number of processes is configured at start up

A MPI program:

- Communication functions (Send/Receive/etc)
- C, C++, FORTRAN (+ interface to others)
- To run the program you need a run-time agent (mpirun)
- Example *mpirun -np 2 ./program*

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Brief History

- 1992 - draft of the project
- 1994 - first version MPI 1.0
  - C, FORTRAN 77
  - Point-to-point
  - Global communication, groups
- 1997 - MPI 2.0
  - One-sided communication
  - C++, FORTRAN 90
  - Dynamic management
- 2008/09 - MPI 2.1 and 2.2
- 2012 - MPI-3.0
  - New one-sided communication

UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

The Body of Your Code / C, C++

The program should contain

- *#include <mpi.h>*
  - the include file for MPI
- MPI_Init(&agrc, &argv);
  - Initialize the MPI
  - Pass the arguments from the command line
- MPI_Finalize();
  - Finalize all communication

UPPSALA
UNIVERSITET

Compile and Run Your Code / Linux

Compilation

- *mpicc -O3 -o program program.c*
- *mpiCC -O3 -o program program.cpp*

Execution

- *mpirun -np N ./program*
- where *N* is the number of process (integer number)

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

UPPSALA
UNIVERSITET

Groups and Communicators

A group is an ordered set of processes.

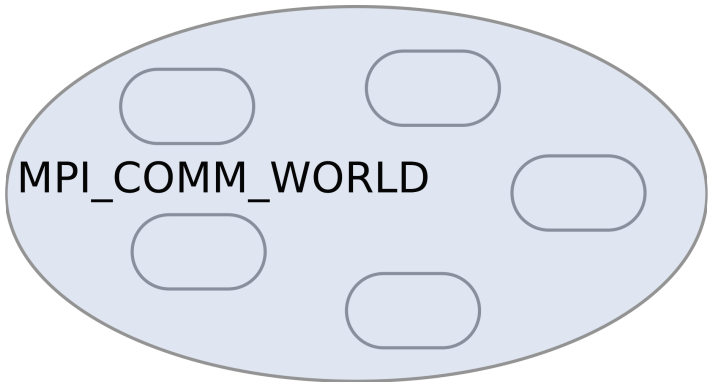
- Each process is associate with ID number - called *rank*
- Processes can belong to different groups

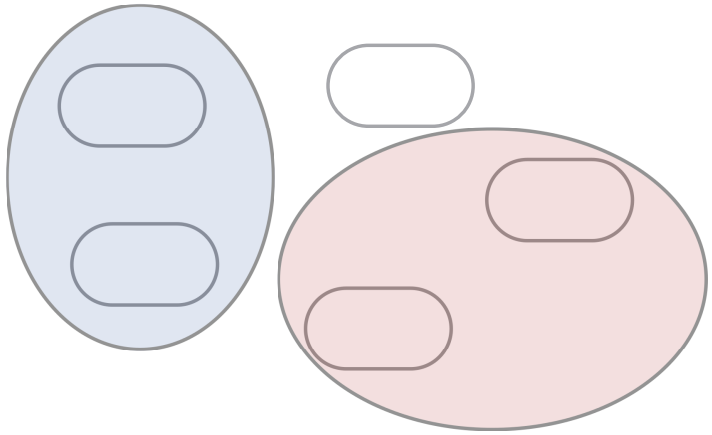
Communicator object connect a group of processes in the MPI session.

- The standard (default) communicator for all MPI process is called *MPI_COMM_WORLD*

UPPSALA
UNIVERSITET

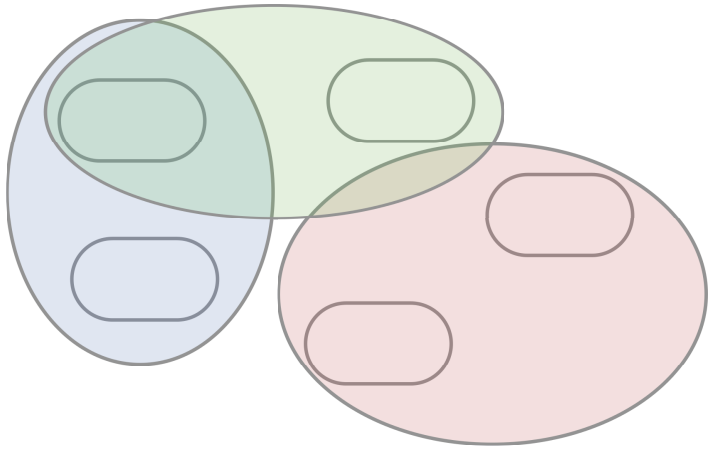
Communicators





UPPSALA
UNIVERSITET

Communicators



UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Communicators

UPPSALA
UNIVERSITET

Communicator Functions

Get the current rank (ID)

- *MPI_Comm_rank()*

Get the size of the communicator

- *MPI_Comm_size()*

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Code Example

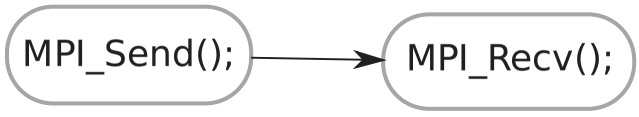
```
#include <stdio.h>
#include <mpi.h>

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

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

 printf("Hello world from process %d of %d\n",
        rank, size );

 MPI_Finalize();
 return 0;
}
```

UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

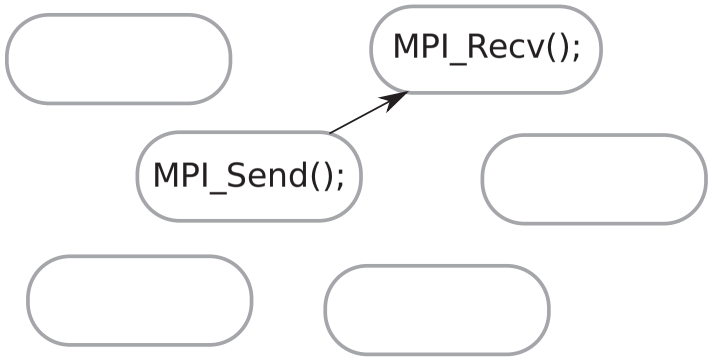
Send and Receive



- Point-to-point communication

UPPSALA
UNIVERSITET

Send - Recv



UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Send

```
MPI_Send(  buff,
           count,
           datatype,
           dest,
           tag,
           comm);
```

void*	buff	Address of the buffer
int	count	Number of elements in the buff
MPI_Datatype	datatype	Data type(MPI_INT,...)
int	dest	Destination (rank)
int	tag	Tag
MPI_Comm	comm	MPI-Communicator

UPPSALA
UNIVERSITET

Recv

```
MPI_Recv(  buff,
           count,
           datatype,
           source,
           tag,
           comm
           status);
```

void*	buff	Address of the buffer
int	count	Number of elements in the buff
MPI_Datatype	datatype	Data type(MPI_INT,...)
int	src	Source (rank)
int	tag	Tag
MPI_Comm	comm	MPI-Communicator
MPI_status*	status	Status

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Code Example

The following code compute the following

- Proc 0 sends data to Proc 1
- Proc 1 sends data to Proc 2
- ...
- Proc N-1 sends data to Proc 0

The data is a double value, where each process adds its rank. Thus, at the end the data will contain the sum of all ranks (0+1+2+3+4+...+N-1).

UPPSALA
UNIVERSITET

Code Example

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

int main(int argc, char *argv[]){
 int size, rank;
 MPI_Status status;
 double bot;

 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 bot = 0.0;
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Code Example

```
if(rank==0){
 MPI_Send(&bot,1,MPI_DOUBLE,1,1,MPI_COMM_WORLD);
 MPI_Recv(&bot,1,MPI_DOUBLE,size-1,1,
              MPI_COMM_WORLD,&status);
 printf("result: %1f\n", bot);

}else {

 MPI_Recv(&bot,1,MPI_DOUBLE,rank-1,1,
 MPI_COMM_WORLD,&status);
 bot += rank;
 MPI_Send(&bot,1,MPI_DOUBLE,
        (rank+1)%size,1,MPI_COMM_WORLD);
}
MPI_Finalize();
return 0;
}
```

UPPSALA
UNIVERSITET

(Non-)Blocking/(A-)Synchronous

Blocking mode

- The current process waits until the its part finished

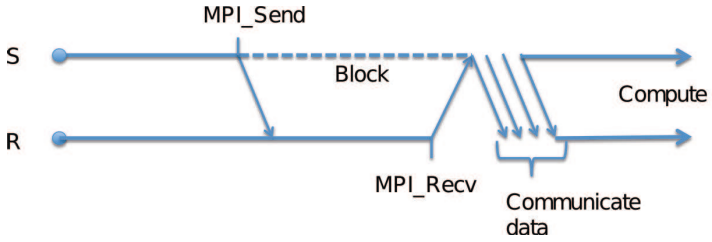
Synchronous mode

- The process waits for the start of the other side

UPPSALA
UNIVERSITET

MPI_Send

Blocking Send (Same as MPI_Ssend)

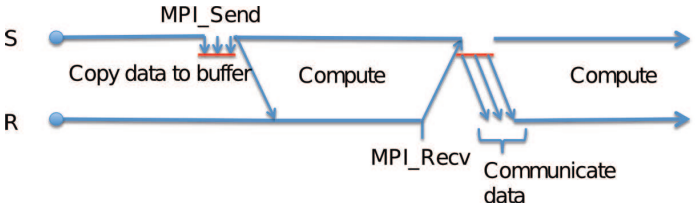


- The processor is blocked until data has been sent
- Receive can be posted before the send
- Receiver waits for the ready-to-send-signal
- Deadlock will occur if no matching send/receive

UPPSALA
UNIVERSITET

MPI_Send

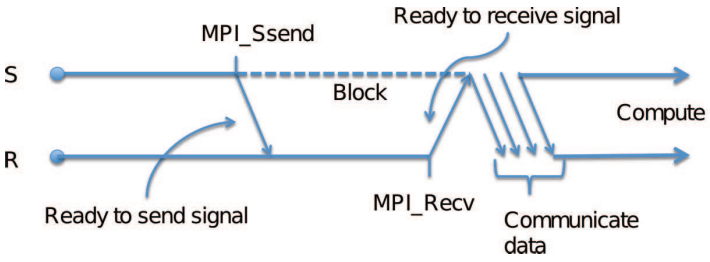
When the data if very small (size<threshold)



UPPSALA
UNIVERSITET

MPI_Ssend

Blocking MPI_Ssend

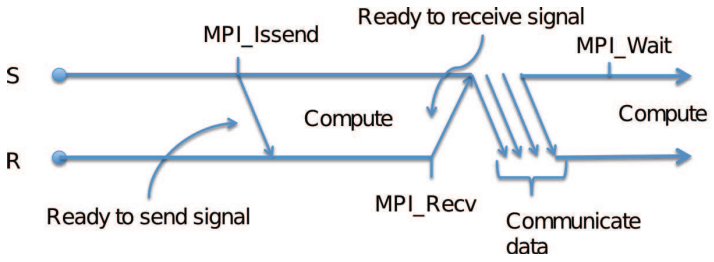


- It will always wait until the receive has been posted on the receiving end.

UPPSALA
UNIVERSITET

MPI_Issend

Non-blocking MPI_Issend

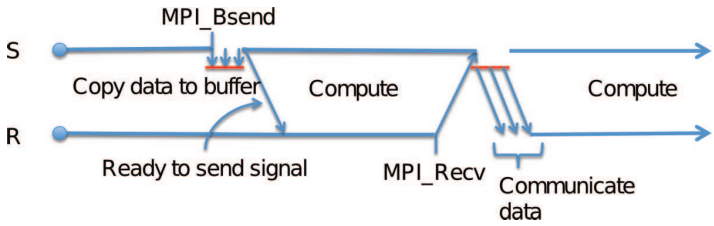


- The sender is not blocked but we need to do an explicit wait to ensure that the data buffer is safe to use (data has been transmitted) with MPI_Wait! Can also do a non-blocking MPI_Test.

UPPSALA
UNIVERSITET

MPI_Bsend

Buffered MPI_Bsend



- It copies data to a user-supplied communication buffer and then returns (no blocking of processor). It is safe to modify the original data and no need to wait for the communication to complete.

UPPSALA
UNIVERSITET

MPI_Bsend

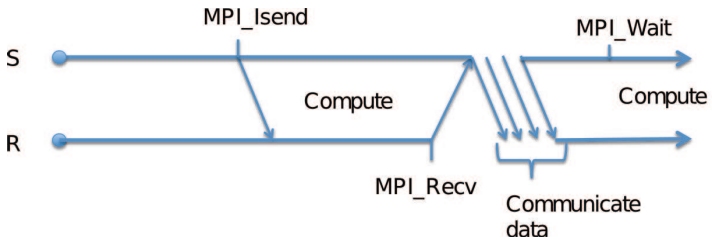
Need to allocate the buffer first (enough space for all pending mess):

- int buflen=totlen*sizeof(double) + MPI_BSEND_OVERHEAD;
- double *buffer=malloc(buflen);
- MPI_Buffer_attach(buffer,buflen);
- MPI_Bsend(data,count,type,dest,tag,comm);

UPPSALA
UNIVERSITET

MPI_Isend

Non-blocking MPI_Isend

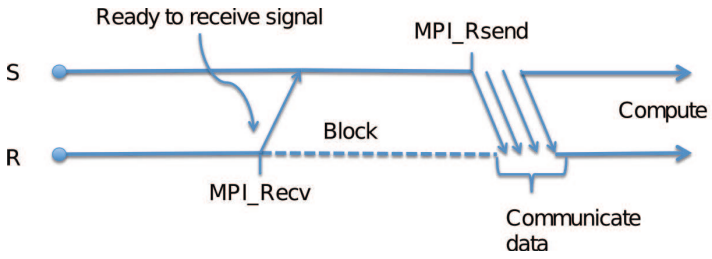


- Same behavior as for non-blocking synchronous send MPI_Issend, need to do an explicit wait/test to ensure that the communication has completed and it is safe to modify the data.

UPPSALA
UNIVERSITET

MPI_Rsend

Ready send MPI_Rsend

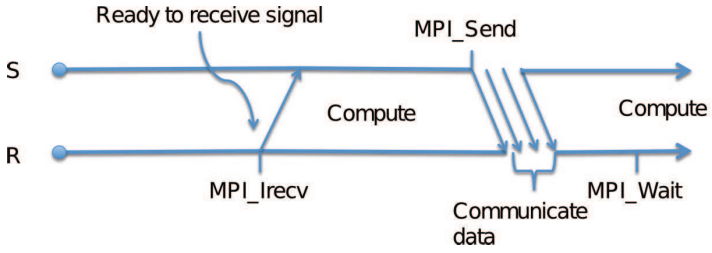


- It sends the message, there must be a receive waiting (if not, the message is lost).

UPPSALA
UNIVERSITET

MPI_Irecv

Non-blocking MPI_Irecv



- The non-blocking receive does not block the processors but then we need to do an explicit wait or test to check if data has arrived.

UPPSALA
UNIVERSITET

Send/Recv

We have four basic communication modes:

- MPI_Send, MPI_Isend – Standard mode
- MPI_Ssend, MPI_Issend – Synchronous mode
- MPI_Rsend, (MPI_Irsend) – Ready mode
- MPI_Bsend, (MPI_Ibsend) – Buffered mode

All modes can be blocked -or- non-blocked Blocking suspends the execution until the message (data) buffer is safe to use (message has been sent/ received/copied). Non-blocking calls returns immediately after initiating the communication.

UPPSALA
UNIVERSITET

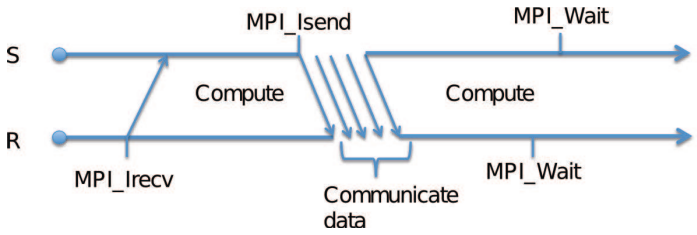
Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Note - Send/Recv

All send calls need to be matched with a receive call!
and
Deadlock will occur if no matching send/receive!

UPPSALA
UNIVERSITET

Tips



- Post a non-blocking send and a non-blocking receive as early as possible and overlap the communication with other computations.
- Using standard non-blocking send and receive is in most cases sufficient and gives good performance.

UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Other Point-to-Point Functions

MPI_Wait

- Wait for request to complete
- Example: MPI_Irecv

MPI_Waitall

- Wait for all requests to complete
- Example: Multiple MPI_Irecv (different tags, senders, etc)

MPI_Waitany

- Wait for any(≥ 1) specififed requests to complete

MPI_Waitsome

- Wait for some (1) specififed requests to complete

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Other Point-to-Point Functions

MPI_Test, MPI_Testall, MPI_Testany, MPI_Testsome

- Same as Wait but
- it only test and returns immedietly

MPI_Probe, MPI_Iprobe

- Check for receiving messages
- Example: getting a receiving status without actually receiving the data

MPI_Cancel

- Abort the operation

UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Global Functions

Barrier
- All processes wait on it

Broadcast
- Send to all processes

Reduce
- Collect data with an operation

Scatter/Gatter
- Send/Receive to/from all

UPPSALA
UNIVERSITET

Barrier

- A global barrier

int MPI_Barrier(MPI_Comm comm);

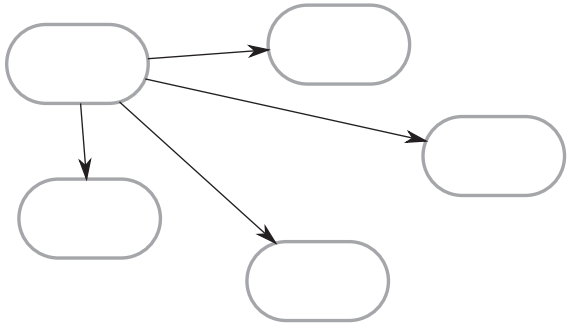
```
...
// parallel computing
...
MPI_Barrier(MPI_COMM_WORLD);
...
/// parallel computing
...
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Broadcast

- A processor sends data to all others



int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm);

UPPSALA
UNIVERSITET

Broadcast

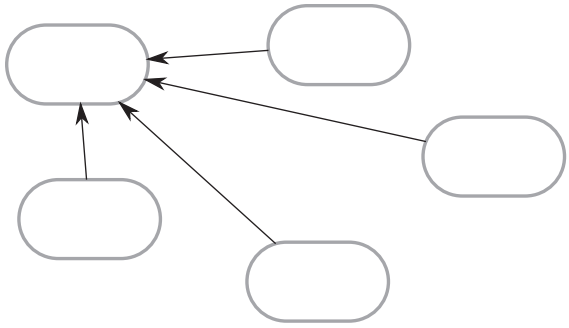
```
...

int global_var;

if (rank == 0)
 global_var = read_file("var.dat");

MPI_Bcast(global_var, 1, MPI_INT, 0,
          MPI_COMM_WORLD);

...
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Reduce

- Reduces values on all processes to a single value



int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm);

UPPSALA
UNIVERSITET

Reduce

```
...
// declare and compute a[]
...
double local_sum = 0.0;

for( int i = 0; i < n; i++)
 local_sum += a[i];

double total;
MPI_Reduce(sum, total, 1, MPI_DOUBLE, MPI_SUM,
           0, MPI_COMM_WORLD);

...
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

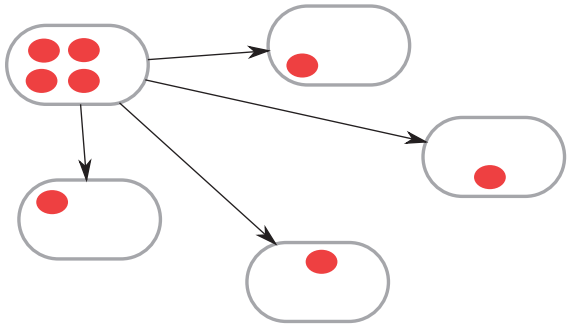
Reduce Operations

MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical XOR
MPI_BXOR	Bitwise XOR
MPI_MAXLOC	Max value and location
MPI_MINLOC	Min value and location

UPPSALA
UNIVERSITET

Scatter

- A processor sends data to all others



int MPI_Scatter(void *sendbuf, int sendcount,
MPI_Datatype sendtype, void *recvbuf, int rcount,
MPI_Datatype recvtype, int root, MPI_Comm comm);

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

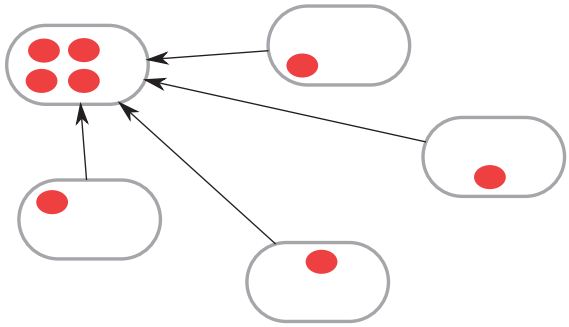
Scatter

```
...
MPI_Comm comm;
int gsize,*sendbuf;
int root, rbuf[100];
...
MPI_Comm_rank( comm, myrank);
if ( myrank == root) {
 MPI_Comm_size( comm, &gsize);
 sendbuf = (int *)malloc(gsize*100*sizeof(int));
}
...
MPI_Scatter(sendbuf, 100, MPI_INT, rbuf, 100,
            MPI_INT, root, comm);
...
```

UPPSALA
UNIVERSITET

Gather

- A processor sends data to all others



int MPI_Gather(void *sendbuf, int sendcount,
MPI_Datatype sendtype, void *recvbuf, int rcount,
MPI_Datatype recvtype, int root, MPI_Comm comm);

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

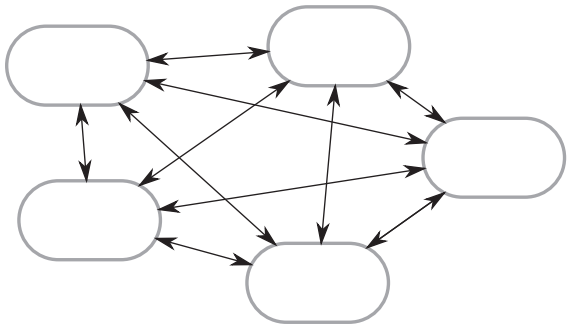
Gather

```
...
MPI_Comm comm;
int gsize,sendarray[100];
int root, myrank, *rbuf;
...
MPI_Comm_rank( comm, myrank);
if ( myrank == root) {
   MPI_Comm_size( comm, &gsize);
   rbuf = (int *)malloc(gsize*100*sizeof(int));
}

MPI_Gather(sendarray, 100, MPI_INT, rbuf, 100,
           MPI_INT, root, comm);
...
```

UPPSALA
UNIVERSITET

MPI_All*

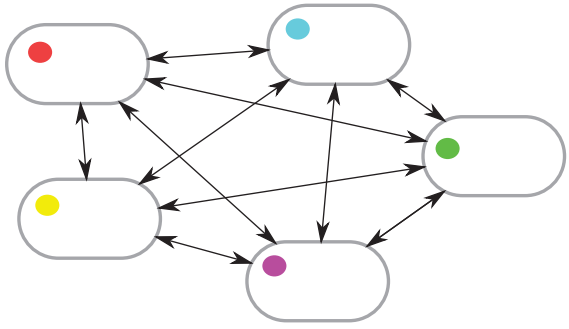
- MPI_Allreduce
- MPI_Allgather
- MPI_Alltoall



UPPSALA
UNIVERSITET

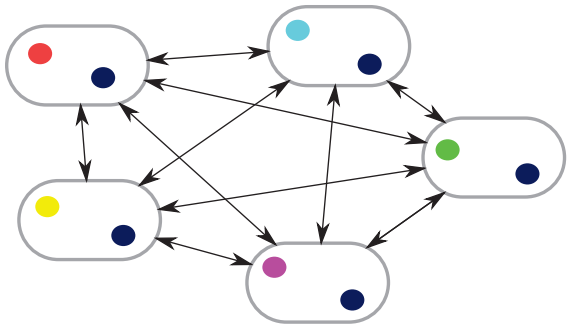
Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_Allreduce



UPPSALA
UNIVERSITET

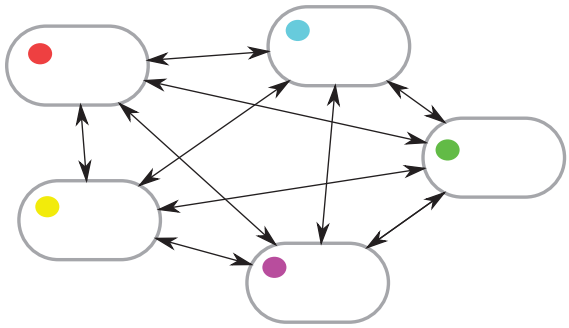
MPI_Allreduce



UPPSALA
UNIVERSITET

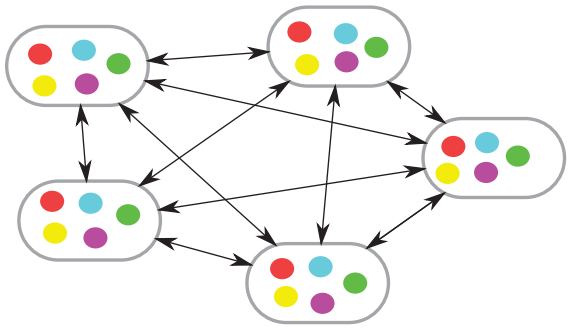
Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_Allgather



UPPSALA
UNIVERSITET

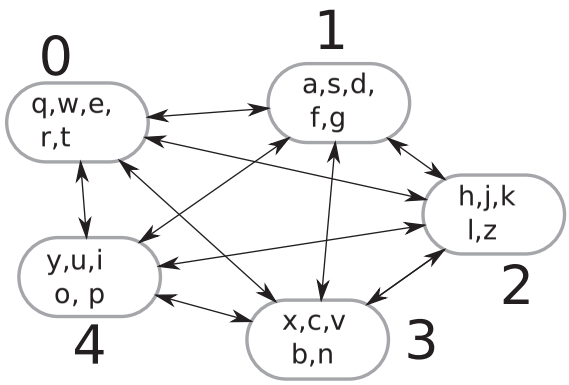
MPI_Allgather



UPPSALA
UNIVERSITET

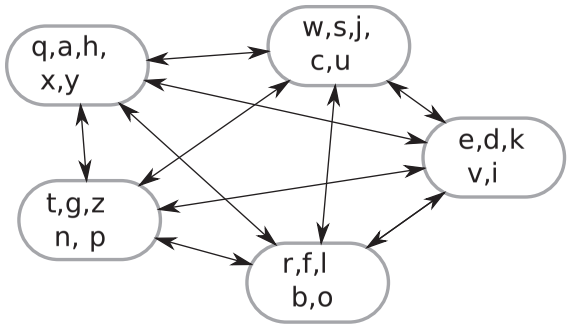
Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_Alltoall



UPPSALA
UNIVERSITET

MPI_Alltoall



UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Timing in MPI

```
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

UPPSALA
UNIVERSITET

MPI 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	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_PACKED	-
MPI_BYTE	-

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Derived Datatype

- Send non-contiguous data
- Reduce the number of communication calls

- MPI_Type_vector() – regular distribution
- MPI_Type_indexed() – irregular distribution
- MPI_Type_struct() – different datatypes

UPPSALA
UNIVERSITET

MPI_TYPE_VECTOR

- MPI_Type_vector(rows,cols,stride,old_type,newtype);
- Makes a row x cols vector
- Cols is number of contiguous elements for each row
- Contents of the new array will be columns of original array

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_TYPE_VECTOR

$A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];$

- MPI_Datatype newtype;
- MPI_Type_vector(2,1,4,MPI_INT,&newtype);
- MPI_Commit(&newtype);
- MPI_Send(&A[0] [1],1,newtype,1,0,comm);
- Sends new array [2 6] to process 1
- MPI_Send(&A[1][3],1,newtype,1,0,comm);
- Sends new array [8 12] to process 1

UPPSALA
UNIVERSITET

MPI_TYPE_VECTOR

$A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]$;

- 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);
- Sends new array [2 3 6 7 10 11] to process 1
- Could be interpreted as matrix with 3 rows, 2 cols

UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_TYPE_INDEX

- MPI_Type_indexed (count, blocklens[], offsets[], old_type,newtype)
- Make blocks at different offsets
- Displacements between successive blocks need not be equal

UPPSALA
UNIVERSITET

MPI_TYPE_INDEX

```
int numtasks, rank, source=0, dest, tag=1;
int blocklengths[2], displacements[2];
float a[16] =
 {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0,
  9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0};
float b[NELEMENTS];
MPI_Status stat;
MPI_Datatype indextype;

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

blocklengths[0] = 4;
blocklengths[1] = 2;
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_TYPE_INDEX

```
displacements[0] = 5;
displacements[1] = 12;

MPI_Type_indexed(2, blocklengths, displacements,
                 MPI_FLOAT, &indextype);
MPI_Type_commit(&indextype);
if (rank == 0) {
 for (int i=0; i < numtasks; i++)
 MPI_Send(a, 1, indextype, i, tag, MPI_COMM_WORLI
}

MPI_Recv(b, NELEMENTS, MPI_FLOAT, source, tag,
         MPI_COMM_WORLD, &stat);
```

UPPSALA
UNIVERSITET

MPI_TYPE_INDEX

```
Output:
rank= 0  b= 6.0 7.0 8.0 9.0 13.0 14.0
rank= 1  b= 6.0 7.0 8.0 9.0 13.0 14.0
rank= 2  b= 6.0 7.0 8.0 9.0 13.0 14.0
rank= 3  b= 6.0 7.0 8.0 9.0 13.0 14.0
```


UPPSALA
UNIVERSITET

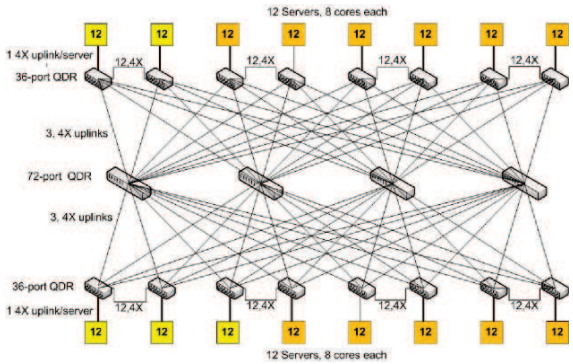
UPPSALA
UNIVERSITET

Topology

Communicator with structure. Makes a mapping to physical topology optimizing (localizing) the communication. Helps in algorithm construction.

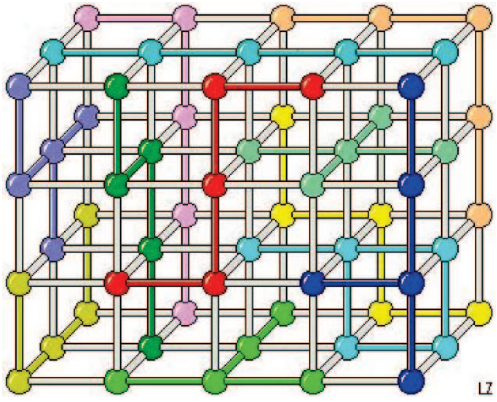
UPPSALA
UNIVERSITET

Network Topology



UPPSALA
UNIVERSITET

Virtual Topology



UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Cartesian Topology

MPI_Cart_create(MPI_Comm old_comm, int ndims, int *dim_size, int *periods, int reorder, MPI_Comm *newcomm);

- Create a cartesian communicator
- with ndim (dimension) and dim_size (size)
- periods - 1 if some of the sides are periodic
- reorder - just ignor it

UPPSALA
UNIVERSITET

MPI_Cart_create

```
MPI_Comm  old_comm, new_comm;
int ndims, reorder, ierr;
int dim_size[2], periods[2];

old_comm = MPI_COMM_WORLD;
ndims = 2;
dim_size[0] = 3;
dim_size[1] = 2;
periods[0] = 1;
periods[1] = 0;
reorder = 1;

MPI_Cart_create(old_comm, ndims, dim_size,
            periods, reorder, &new_comm);
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

Cartesian Topology

Map rank to coordinate

- MPI_Cart_coords();
- Input rank
- Output coordinate

Map coordinate to rank

- MPI_Cart_rank();
- Input coordinate
- Output rank

UPPSALA
UNIVERSITET

MPI_Cart_rank

```
MPI_Cart_create(old_comm, ndims, dim_size,
           periods, reorder, &new_comm);

for (i=0; i<nv; i++)
 for (j=0; j<mv; j++)  {
   coords[0] = i;
   coords[1] = j;
   MPI_Cart_rank(new_comm, coords, &rank);

   printf("%d, %d, %d\n",coords[0],coords[1],
          rank);
  }
```


UPPSALA
UNIVERSITET

Outline
Intro and motivation
Code Body
Communicators
Send and Receive
Other P2P Funcs
Global Funcs
Timing
Datatypes
Topology

MPI_Cart_coords

```
MPI_Cart_create(old_comm, ndims, dim_size,
                periods, reorder, &new_comm);

 for (rank=0; rank<p; rank++) {
   MPI_Cart_coords(new_comm, rank, coords);
   printf("%d, %d\n ",rank, coords);
  }
```