

MPI Tutorial

Desert Code Camp

Anthony DiGirolamo
anthony.d@asu.edu

Arizona State University

Saturday November 7, 2009

Introduction

- Alternatives
- Versions
- Why?

Basics

- Initialization
- Communicators
- Point to Point
- Broadcast
- Collectives
 - Synchronization
- Data Movement
- Collective Computations
- Compiling
- Running

Advanced

- Blocking
- Non-Blocking
- Sendrecv
- More Collective Comm.
 - Scatterv
 - Allgatherv
- Datatypes
- Communicators & Groups

Further Reading

What is MPI?

What is message passing?

- ▶ Literally, the sending and receiving of messages between tasks
- ▶ Commonly used in distributed systems
- ▶ Capabilities include sending data, performing operations on data, and synchronization between tasks

Memory Model

Each process has its own address space, and no way to get at another's, so it is necessary to send/receive data

Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point
Broadcast
Collectives
Synchronization
Data Movement
Collective Computations
Compiling
Running

Advanced

Blocking
Non-Blocking
Sendrecv
More Collective Comm.
Scatterv
Allgatherv
Datatypes
Communicators & Groups

Further Reading

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ OpenMP
- ▶ PGAS languages
(Partitioned Global Address Space)
 - ▶ UPC
 - ▶ Co-Array Fortran
- ▶ Global Arrays Library
- ▶ Hybrids with Message Passing are possible

MPI-1 - Message Passing Interface (v. 1.2)

- ▶ library standard defined by committee of vendors, implementers, and parallel programmers
- ▶ used to create parallel SPMD codes based on explicit message passing

Available on almost all parallel machines with C/C++ and Fortran bindings (and occasionally with other bindings)

About 125 routines, total

- ▶ 6 basic routines
- ▶ the rest include routines of increasing generality and specificity

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Includes features left out of MPI-1

- ▶ one-sided communications
- ▶ dynamic process control
- ▶ more complicated collectives

Implementations

- ▶ not quickly undertaken after the standard document was released (in 1997)
- ▶ now OpenMPI, MPICH2 (and its descendants), and the vendor implementations are pretty complete or fully complete

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Why Learn MPI?

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ MPI is a standard
 - ▶ Public domain version easily to install
 - ▶ Vendor-optimized version available on most communication hardware
- ▶ MPI applications can be fairly portable
- ▶ MPI is expressive: MPI can be used for many different models of computation, therefore can be used with many different applications.
- ▶ MPI is “assembly language of parallel processing”: low level but efficient.

Initialization and Termination

- ▶ Must include header files
- ▶ All processes must initialize and finalize MPI (each is a collective call).

C

```
#include <mpi.h>
int main(int argc, char *argv[] ) {
    int ierr;
    ierr = MPI_Init(&argc, &argv);
    ...
    ierr = MPI_Finalize();
}
```

Fortran

```
program init_finalize
    include 'mpif.h'
    call mpi_init(ierr)
    ...
    call mpi_finalize(ierr)
end program
```

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Communicators

- ▶ collections of processes that can communicate with each other
- ▶ Multiple communicators can co-exist
- ▶ Each communicator answers two questions:
 - ▶ How many processes exist in this communicator?
 - ▶ Which process am I?
- ▶ MPI_COMM_WORLD encompasses all processes and is defined by default

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Setting up Communicators

C

```
#include <mpi.h>
int main(int argc, char *argv[] ) {
    int ierr, myrank, procs;

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &procs);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    ierr = MPI_Finalize();
}
```

Fortran

```
program init_finalize
    include 'mpif.h'

    call mpi_init(ierr)
    call mpi_comm_size(MPI_COMM_WORLD, procs ,ierr)
    call mpi_comm_rank(MPI_COMM_WORLD, myrank ,ierr)

    call mpi_finalize(ierr)
end program
```

Introduction

[Alternatives](#)

[Versions](#)

[Why?](#)

Basics

[Initialization](#)

[Communicators](#)

[Point to Point](#)

[Broadcast](#)

[Collectives](#)

[Synchronization](#)

[Data Movement](#)

[Collective Computations](#)

[Compiling](#)

[Running](#)

Advanced

[Blocking](#)

[Non-Blocking](#)

[Sendrecv](#)

[More Collective Comm.](#)

[Scatter](#)

[Allgather](#)

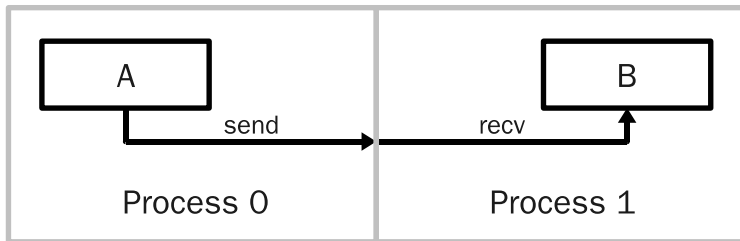
[Datatypes](#)

[Communicators & Groups](#)

Further Reading

Point to Point Communication

- ▶ Sending data from one point (process/task) to another point (process/task)
- ▶ One task sends while another receives



- ▶ **MPI_Send():** A blocking call which returns only when data has been sent from its buffer
- ▶ **MPI_Recv():** A blocking receive which returns only when data has been received onto its buffer

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Point to Point Communication

Parameters:

- ▶ void* data: actual data being passed
- ▶ int count: number of type values in data
- ▶ MPI_Datatype type: data type of data
- ▶ int dest/src: rank of the process this call is sending to or receiving from. src can also be wildcard MPI_ANY_SOURCE
- ▶ int tag: simple identifier that must match between sender/receiver, or the wildcard MPI_ANY_TAG
- ▶ MPI_Comm comm: communicator that must match between sender/receiver – no wildcards
- ▶ MPI_Status* status: returns information on the message received (receiving only), e.g. which source if using MPI_ANY_SOURCE
- ▶ INTEGER ierr: place to store error code (Fortran only. In C/C++ this is the return value of the function call)

C

```
MPI_Send(data, count, type, dest, tag, comm)
MPI_Recv(data, count, type, src, tag, comm, &status)
```

Fortran

```
mpi_send(data, count, type, dest, tag, comm, ierr)
mpi_recv(data, count, type, src, tag, comm, status, ierr)
```

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Point to Point Communication

Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgatherv](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

- Recall that all tasks execute the same code, conditionals are often needed

```
_____ C _____  
MPI_Comm_rank(comm, &myrank);  
  
if (myrank==0) {  
    MPI_Send( /*buffer*/, /*target=*/ 1, /*tag=*/ 0, comm);  
} else if (myrank==1) {  
    MPI_Recv( /*buffer*/, /*source=*/ 0, /*tag=*/ 0, comm);  
}
```

Examples

C

```
#include "mpi.h"
main(int argc, char **argv){
    int myrank, procs, ierr; double a[2];
    MPI_Status status;
    MPI_Comm icomm = MPI_COMM_WORLD;
    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(icom, &myrank);
    ierr = MPI_Comm_size(icom, &procs);
    if(myrank == 0){
        a[0] = myrank; a[1] = myrank+1;
        ierr = MPI_Send(a,2,MPI_DOUBLE, 1,9, icomm);
    }
    else if (myrank == 1){
        ierr = MPI_Recv(a,2,MPI_DOUBLE, 0,9,icom,&status);
        printf("PE %d, A array= %f %f\n",myrank,a[0],a[1]);
    }
    MPI_Finalize();
}
```

Introduction

- Alternatives
- Versions
- Why?

Basics

- Initialization
- Communicators

Point to Point

- Broadcast
- Collectives
- Synchronization
- Data Movement
- Collective Computations
- Compiling
- Running

Advanced

- Blocking
- Non-Blocking
- Sendrecv
- More Collective Comm.
- Scatterv
- Allgatherv
- Datatypes
- Communicators & Groups

Further Reading

Examples

Fortran

```
program sr
  include "mpif.h"
  real*8, dimension(2) :: A
  integer, dimension(MPI_STATUS_SIZE) :: istat
  icomm = MPI_COMM_WORLD
  call mpi_init(ierr)
  call mpi_comm_rank(icomm,mype,ierr)
  call mpi_comm_size(icomm,np ,ierr);

  if(mype.eq.0) then
    a(1) = real(ipe); a(2) = real(ipe+1)
    call mpi_send(A,2,MPI_REAL8, 1,9,icomm, ierr)
  else if (mype.eq.1) then
    call mpi_recv(A,2,MPI_REAL8, 0,9,icomm, istat,ierr)
    print*,"PE ",mype,"received A array =",A
  endif

  call mpi_finalize(ierr)
end program
```

Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgatherv](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

- ▶ What if one processor wants to send to everyone else?

```
      C
if (mytid == 0 ) {
    for (tid=1; tid<ntids; tid++) {
        MPI_Send( (void*)a, /* target= */ tid, ... );
    }
} else {
    MPI_Recv( (void*)a, 0, ... );
}
```

- ▶ Implements a very naive, serial broadcast
- ▶ Too primitive
 - ▶ leaves no room for the OS / switch to optimize
 - ▶ leaves no room for fancy algorithms
- ▶ Too slow: calls may wait for completion.

Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point

Broadcast

Collectives
Synchronization
Data Movement
Collective Computations
Compiling
Running

Advanced

Blocking
Non-Blocking
Sendrecv
More Collective Comm.
Scatterv
Allgatherv
Datatypes
Communicators & Groups

Further Reading

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ Collective Communication
 - ▶ Communication pattern that involves all processes within a communicator
- ▶ There are three basic types of collective communications:
 - ▶ Synchronization
 - ▶ Data movement
 - ▶ Computation w/ movement
- ▶ MPI Collectives are all blocking
- ▶ MPI Collectives do not use message tags

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

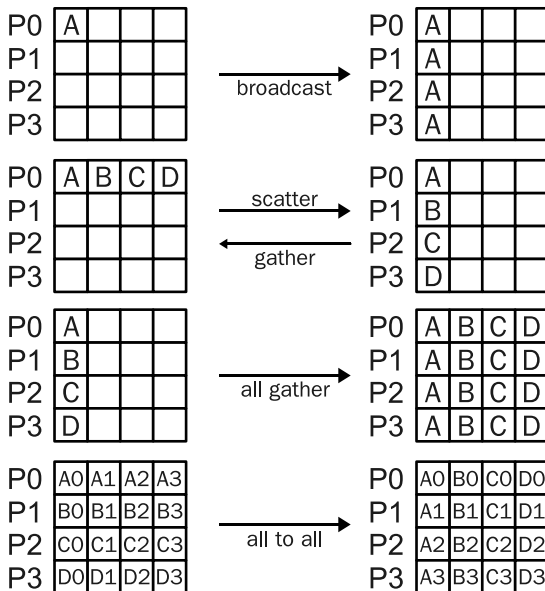
Further Reading

Barrier

- ▶ `MPI_Barrier(comm)`
- ▶ `mpi_barrier(comm, ierr)`

Function blocks until all tasks in comm call it.

Collective Data Movements



Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point
Broadcast
Collectives
Synchronization

Data Movement

Collective Computations
Compiling
Running

Advanced

Blocking
Non-Blocking
Sendrecv
More Collective Comm.
Scatterv
Allgatherv
Datatypes
Communicators & Groups

Further Reading

Broadcast

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ `MPI_Bcast(data, count, type, root, comm)`
- ▶ `mpi_bcast(data, count, type, root, comm, ierr)`

Sends data from the root process to all communicator members.

Collective Computations

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

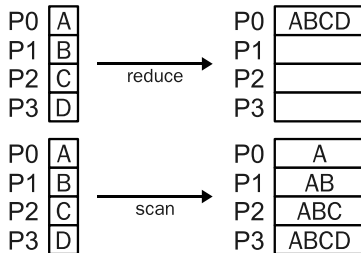
Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading



Collective Computations

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ `MPI_Reduce(data, result, count, type, op, root, comm)`
- ▶ `mpi_reduce(data, result, count, type, op, root, comm, ierr)`

Operations

- ▶ `MPI_SUM`
- ▶ `MPI_PROD`
- ▶ `MPI_MAX`
- ▶ `MPI_MIN`
- ▶ `MPI_MAXLOC`
- ▶ `MPI_MINLOC`

Example

C

```
#include <mpi.h>
#define WCOMM MPI_COMM_WORLD
main(int argc, char **argv){
    int npes, mype, ierr;
    double sum, val; int calc, knt=1;
    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_size(WCOMM, &npes);
    ierr = MPI_Comm_rank(WCOMM, &mype);

    val = (double) mype;

    ierr = MPI_Allreduce(&val, &sum, knt,
                        MPI_DOUBLE, MPI_SUM, WCOMM);

    calc = (npes-1 + npes%2) * (npes/2);
    printf(" PE: %d sum=%5.0f calc=%d\n", mype, sum, calc);
    ierr = MPI_Finalize();
}
```

Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgatherv](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

Example

Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point
Broadcast
Collectives
Synchronization
Data Movement

Collective Computations

Compiling
Running

Advanced

Blocking
Non-Blocking
Sendrecv
More Collective Comm.
Scatterv
Allgatherv
Datatypes
Communicators & Groups

Further Reading

Fortran

```
program sum2all
  include 'mpif.h'

  icomm = MPI_COMM_WORLD
  knt = 1
  call mpi_init(ierr)
  call mpi_comm_rank(icommm,myype,ierr)
  call mpi_comm_size(icommm,npes,ierr)
  val = dble(myype)

  call mpi_allreduce(val,sum,knt,MPI_REAL8,MPI_SUM,icommm,ierr)

  ncalc=(npes-1 + mod(npes,2))*(npes/2)
  print*,' pe#, sum, calc. sum = ',myype,sum,ncalc
  call mpi_finalize(ierr)
end
```

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ Generally use a special compiler or compiler wrapper script
 - ▶ not defined by the standard
 - ▶ consult your implementation
 - ▶ handles correct include path, library path, and libraries
- ▶ MPICH-style (the most common)
 - ▶ `mpicc -o foo foo.c`
 - ▶ `mpif77 -o foo foo.f`

Running MPI Programs

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

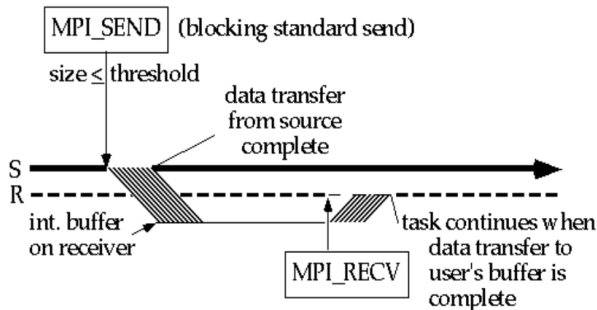
Communicators & Groups

Further Reading

- ▶ MPI programs require some help to get started
 - ▶ what computers should I run on?
 - ▶ how do I access them?
- ▶ MPICH-style
 - ▶ `mpirun -np 10 -machinefile mach ./foo`
- ▶ When batch schedulers are involved, all bets are off
- ▶ On Ranger
 - ▶ `ibrun ./foo`
and the scheduler handles the rest
- ▶ On Saguaro
 - ▶ create a jobscript that calls `mpirun` somewhere
 - ▶ `qsub ./jobscript`

Blocking Send / Receive

- ▶ MPI_Send, does not return until buffer is safe to reuse: either when buffered, or when actually received. (implementation / runtime dependent)
- ▶ Rule of thumb: send completes only if receive is posted



Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

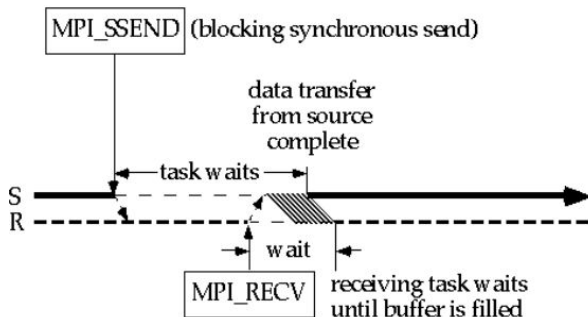
Datatypes

Communicators & Groups

Further Reading

Synchronous Mode

MPI_Ssend, which does not return until matching receive has been posted (non-local operation).



Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

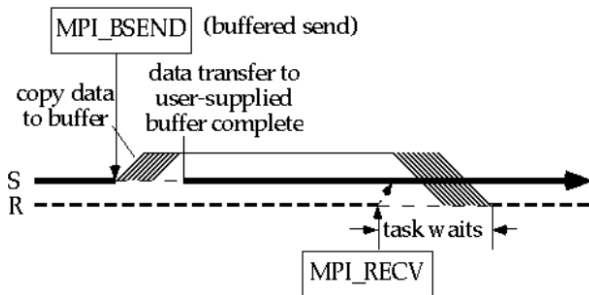
[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgatherv](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

Blocking Send / Receive

Buffered Mode

MPI_Bsend, which completes as soon as the message buffer is copied into user-provided buffer



Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

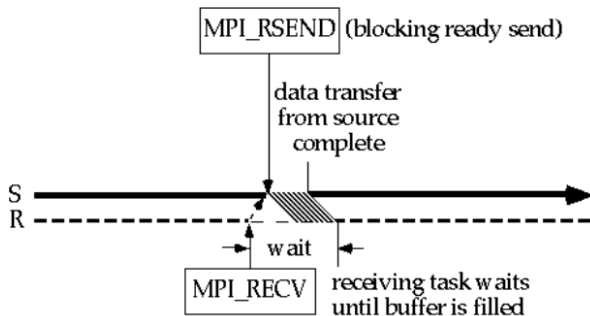
[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgatherv](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

Blocking Send / Receive

Ready Mode

MPI_Rsend returns immediately assuming that a matching receive has been posted



Deadlocks occur when all tasks are waiting for events that haven't been initiated yet. It's most common with blocking communication.

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Blocking Send / Receive

- ▶ **Ready Mode** has least total overhead. However the assumption is that receive is already posted.
- ▶ **Synchronous mode** is portable and safe. It does not depend on order (ready mode) or buffer space (buffered mode). However it incurs substantial overhead.
- ▶ **Buffered mode** decouples sender from receiver. No sync. overhead on sending task and order of execution does not matter (ready mode). User can control size of message buffers and total amount of space. However additional overhead may be incurred by copy to buffer and buffer space can run out.
- ▶ **Standard Mode** is implementation dependent. Small messages are generally buffered (avoiding sync. overhead) and large messages are usually sent synchronously (avoiding the required buffer space)

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

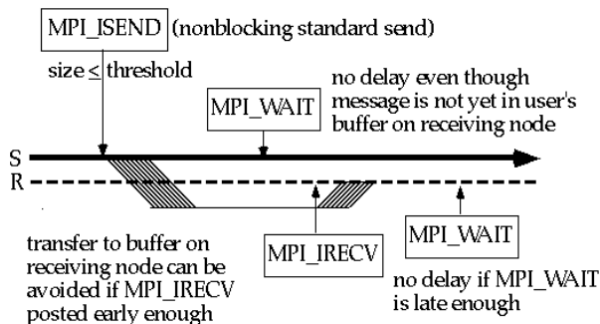
Datatypes

Communicators & Groups

Further Reading

Non-Blocking Send / Receive

- ▶ Nonblocking communication: calls return, system handles buffering
- ▶ `MPI_Isend`, completes immediately but user must check status before using the buffer for same (tag/receiver) send again; buffer can be reused for different tag/receiver.
- ▶ `MPI_Irecv`, gives a user buffer to the system; requires checking whether data has arrived.



Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

The Sendrecv Operation

MPI_Sendrecv

- ▶ both in one call, source and dest can be the same
- ▶ Example 1: exchanging data with one other node; target and source the same
- ▶ Example 2: chain of processors
 - ▶ Operate on data
 - ▶ Send result to next processor, and receive next input from previous processor in line

Example

```
MPI_SENDRECV(  
    sendbuf, sendcount, sendtype, dest, sendtag,  
    recvbuf, recvcount, recvtype, source, recvtag,  
    comm, status)
```

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

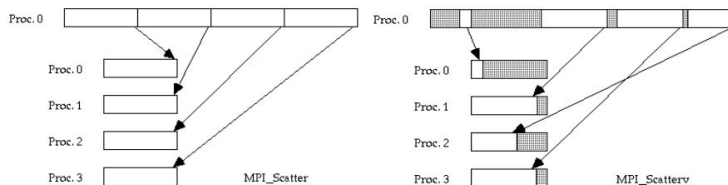
Datatypes

Communicators & Groups

Further Reading

- ▶ MPI_{Scatter,Gather,Allscatter,Allgather}v
- ▶ What does v stand for?
 - ▶ varying size, relative location of messages
- ▶ Advantages
 - ▶ more flexibility
 - ▶ less need to copy data into temp. buffers
 - ▶ more compact
- ▶ Disadvantage
 - ▶ Somewhat harder to program (more bookkeeping)

Scatter vs Scatterv



Example

```
CALL mpi_scatterv(SENDBUF, SENDCOUNTS, DISPLS, SENDTYPE,  
                 RECVBUF, RECVCOUNTS, RECVTYPE, ROOT, COMM, IERR)
```

- ▶ **SENDCOUNTS(i)** is the number of items of type **SENDTYPE** to send from process **ROOT** to process **i**. Defined on **ROOT**.
- ▶ **DISPLS(i)** is the displacement from **SENDBUF** to the beginning of the **i**-th message, in units of **SENDTYPE**. Defined on **ROOT**.

Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point
Broadcast
Collectives
Synchronization
Data Movement
Collective Computations
Compiling
Running

Advanced

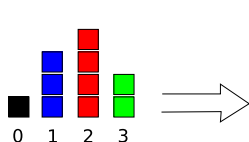
Blocking
Non-Blocking
Sendrecv
More Collective Comm.

Scatterv

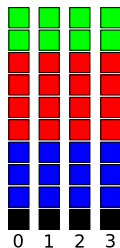
Allgatherv
Datatypes
Communicators & Groups

Further Reading

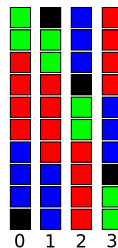
Allgatherv Example



AllgatherV takes data of varied length from each task and distributes it to all tasks.



AllgatherV can also alter the order of data based on task.



Example

```
MPI_Allgatherv(  
    void *sendbuf, int sendcount, MPI_Datatype sendtype,  
    void *recvbuf, int *recvcounts, int *displs,  
    MPI_Datatype recvtype, MPI_Comm comm )
```

Introduction

[Alternatives](#)[Versions](#)[Why?](#)

Basics

[Initialization](#)[Communicators](#)[Point to Point](#)[Broadcast](#)[Collectives](#)[Synchronization](#)[Data Movement](#)[Collective Computations](#)[Compiling](#)[Running](#)

Advanced

[Blocking](#)[Non-Blocking](#)[Sendrecv](#)[More Collective Comm.](#)[Scatterv](#)[Allgather](#)[Datatypes](#)[Communicators & Groups](#)

Further Reading

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ MPI basic data-types are predefined for contiguous data of single type
- ▶ What if application has data of mixed types, or non-contiguous data?
 - ▶ existing solutions of multiple calls or copying into buffer and packing etc. are slow, clumsy and waste memory
 - ▶ better solution is creating/deriving datatypes for these special needs from existing datatypes
- ▶ Derived Data-types can be created recursively and at run-time
- ▶ Automatic packing and unpacking

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

- ▶ Elementary: Language-defined types
- ▶ Contiguous: Vector with stride of one
- ▶ Vector: Separated by constant stride
- ▶ Hvector: Vector, with stride in bytes
- ▶ Indexed: Array of indices (for scatter/gather)
- ▶ Hindexed: Indexed, with indices in bytes
- ▶ Struct: General mixed types (for C structs etc.)

Introduction

Alternatives
Versions
Why?

Basics

Initialization
Communicators
Point to Point
Broadcast
Collectives
Synchronization
Data Movement
Collective Computations
Compiling
Running

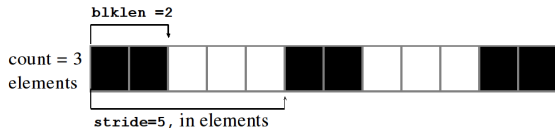
Advanced

Blocking
Non-Blocking
Sendrecv
More Collective Comm.
Scatterv
Allgatherv

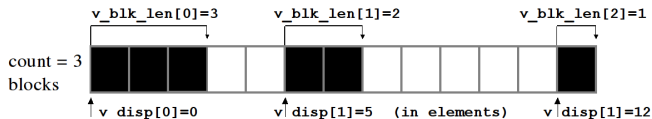
Datatypes

Communicators & Groups

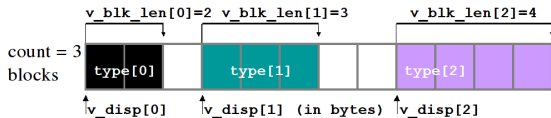
Further Reading



Vector
(strided)



Indexed



“Struct”

Sending a Row in Fortran

MPI_Type_vector

Allows creating non-contiguous vectors with constant stride

Syntax

```
mpi_type_vector(count, blocklen, stride,  
                oldtype, vtype, ierr)  
mpi_type_commit(vtype, ierr)
```

	ncols			
nrows	1	6	11	16
	2	7	12	17
	3	8	13	18
	4	9	14	19
	5	10	15	20

Example

```
call MPI_Type_vector(ncols, 1, nrows,  
                    MPI_DOUBLE_PRECISION, vtype, ierr)  
call MPI_Type_commit(vtype, ierr)  
call MPI_Send( A(nrows,1) , 1 , vtype )
```

Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

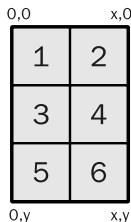
Further Reading

Communicators and Groups

- ▶ All MPI communication is relative to a communicator which contains a context and a group.
- ▶ The group is just a set of processes.
- ▶ Groups can be created by using the union, difference, intersection, and range operations.

Data Decomposition

How can we construct a communicator with a 2D decomposition? For example: X by Y and 6 processor elements



Introduction

Alternatives

Versions

Why?

Basics

Initialization

Communicators

Point to Point

Broadcast

Collectives

Synchronization

Data Movement

Collective Computations

Compiling

Running

Advanced

Blocking

Non-Blocking

Sendrecv

More Collective Comm.

Scatterv

Allgatherv

Datatypes

Communicators & Groups

Further Reading

MPI-1 Standard

<http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html>

MPI-2 Standard

<http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.htm>

MPI Guides

<http://www.clustermonkey.net/content/category/5/13/28/>

Thank you!

Introduction

- Alternatives
- Versions
- Why?

Basics

- Initialization
- Communicators
- Point to Point
- Broadcast
- Collectives
 - Synchronization
- Data Movement
- Collective Computations
- Compiling
- Running

Advanced

- Blocking
- Non-Blocking
- Sendrecv
- More Collective Comm.
 - Scatterv
 - Allgatherv
- Datatypes
- Communicators & Groups

Further Reading