

Tutorial on MPI: The Message-Passing Interface

William Gropp



Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
gropp@mcs.anl.gov

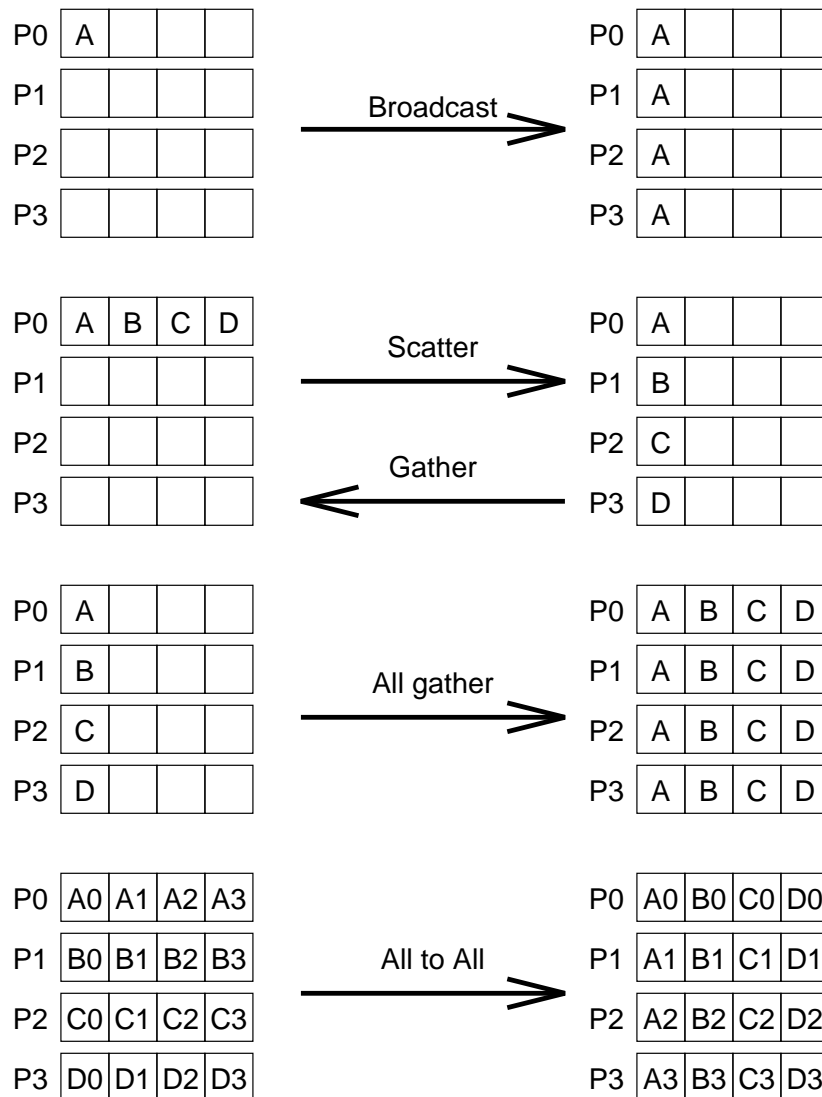
Collective Communications in MPI

- Communication is coordinated among a group of processes.
- Groups can be constructed “by hand” with MPI group-manipulation routines or by using MPI topology-definition routines.
- Message tags are not used. Different communicators are used instead.
- No non-blocking collective operations.
- Three classes of collective operations:
 - synchronization
 - data movement
 - collective computation

Synchronization

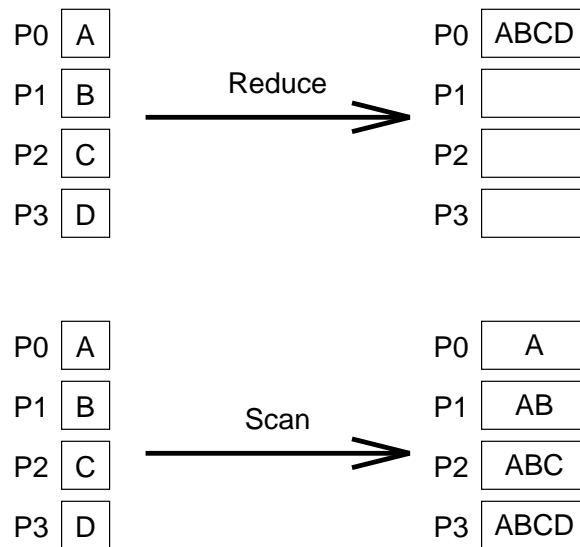
- `MPI_Barrier(comm)`
- Function blocks until all processes in `comm` call it.

Available Collective Patterns



Schematic representation of collective data movement in MPI

Available Collective Computation Patterns



Schematic representation of collective data movement in MPI

MPI Collective Routines

- Many routines:

Allgather	Allgatherv	Allreduce
Alltoall	Alltoallv	Bcast
Gather	Gatherv	Reduce
ReduceScatter	Scan	Scatter
Scatterv		

- All versions deliver results to all participating processes.
- V versions allow the chunks to have different sizes.
- Allreduce, Reduce, ReduceScatter, and Scan take both built-in and user-defined combination functions.

Built-in Collective Computation Operations

MPI Name	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_LAND	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or (xor)
MPI_BAND	Bitwise and
MPI_BOR	Bitwise or
MPI_BXOR	Bitwise xor
MPI_MAXLOC	Maximum value and location
MPI_MINLOC	Minimum value and location

Defining Your Own Collective Operations

```
MPI_Op_create(user_function, commute, op)
MPI_Op_free(op)
```

```
user_function(invec, inoutvec, len, datatype)
```

The user function should perform

```
inoutvec[i] = invec[i] op inoutvec[i];
```

for i from 0 to $len-1$.

`user_function` can be non-commutative (e.g., matrix multiply).

Sample user function

For example, to create an operation that has the same effect as MPI_SUM on Fortran double precision values, use

```
subroutine myfunc( invec, inoutvec, len, datatype )
  integer len, datatype
  double precision invec(len), inoutvec(len)
  integer i
  do 10 i=1,len
10   inoutvec(i) = invec(i) + inoutvec(i)
  return
end
```

To use, just

```
integer myop
call MPI_Op_create( myfunc, .true., myop, ierr )
call MPI_Reduce( a, b, 1, MPI_DOUBLE_PRECISION, myop, ... )
```

The routine MPI_Op_free destroys user-functions when they are no longer needed.

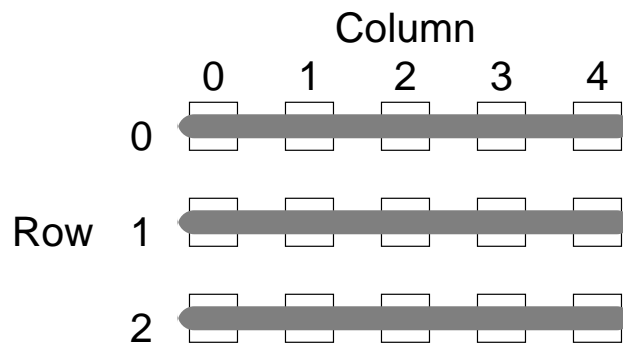
Defining groups

All MPI communication is relative to a *communicator* which contains a *context* and a *group*. The group is just a set of processes.

Subdividing a communicator

The easiest way to create communicators with new groups is with `MPI_COMM_SPLIT`.

For example, to form groups of rows of processes



use

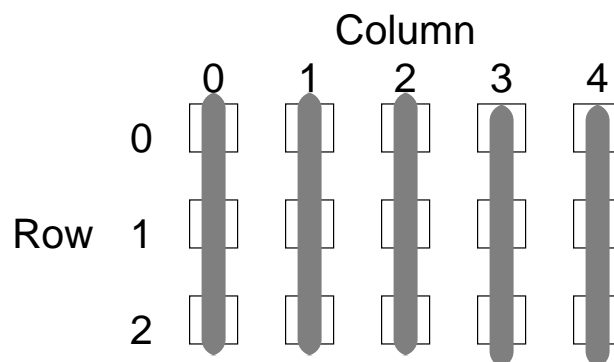
```
MPI_Comm_split( oldcomm, row, 0, &newcomm );
```

To maintain the order by rank, use

```
MPI_Comm_rank( oldcomm, &rank );  
MPI_Comm_split( oldcomm, row, rank, &newcomm );
```

Subdividing (con't)

Similarly, to form groups of columns,



use

```
MPI_Comm_split( oldcomm, column, 0, &newcomm2 );
```

To maintain the order by rank, use

```
MPI_Comm_rank( oldcomm, &rank );  
MPI_Comm_split( oldcomm, column, rank, &newcomm2 );
```

Manipulating Groups

Another way to create a communicator with specific members is to use `MPI_Comm_create`.

```
MPI_Comm_create( oldcomm, group, &newcomm );
```

The group can be created in many ways:

Creating Groups

All group creation routines create a group by specifying the members to take from an existing group.

- `MPI_Group_incl` specifies specific members
- `MPI_Group_excl` excludes specific members
- `MPI_Group_range_incl` and `MPI_Group_range_excl` use ranges of members
- `MPI_Group_union` and `MPI_Group_intersection` creates a new group from two existing groups.

To get an existing group, use

```
MPI_Comm_group( oldcomm, &group );
```

Free a group with

```
MPI_Group_free( &group );
```