Lecture 14
The Rest of MPI

EN 600.320/420

Instructor: Randal Burns

2 and 4 April 2012

Department of Computer Science, *Johns Hopkins University*

Using MPI

- I bought an entire book
  - I should get one lecture out of it

- *Using MPI: Portable Parallel Programming with the Message Passing Interface.* William Gropp, Ewing Lusk, and Anthony Skjellum, MIT Press, 1999.

Chapter 1: Background

Some stuff I learned and interesting observations about message passing

- Ease of debugging
  - While debuggers are better on shared-memory systems, most common source of bugs (overwriting shared memory) is more easily addressed
  - One process handles memory at each location
- Universality
  - Message passing is the lowest common denominator of parallel computing
  - MPI programs run on every architecture

Chapter 2: Introduction

- Message tagging
  - Used to deal w/ out of order delivery.
  - Process receives messages in the order it wants.

- Transmitting sparse data
  - Collection (gather) of data performed by MPI systems
    - Late binding, minimize buffering and avoid a copy
    - E.g., send a column in row-major array

Tags vs. Communicator

```
MPI_Send ( address, count, datatype,
           destination, tag, comm )
```

- Tag is an application defined concept used to determine delivery order
  - Specify a tag, get the message you desire, regardless of delivery order
  - There are wildcards to receive all messages (Reg. exps?)
- Communicator specifies a subset of nodes running a parallel application
  - Has a rank and size
  - Default MPI_COMM_WORLD

Broadcast/Reduce

- An alternative to point-to-point messaging
  - For programs with 1->all->1 structure

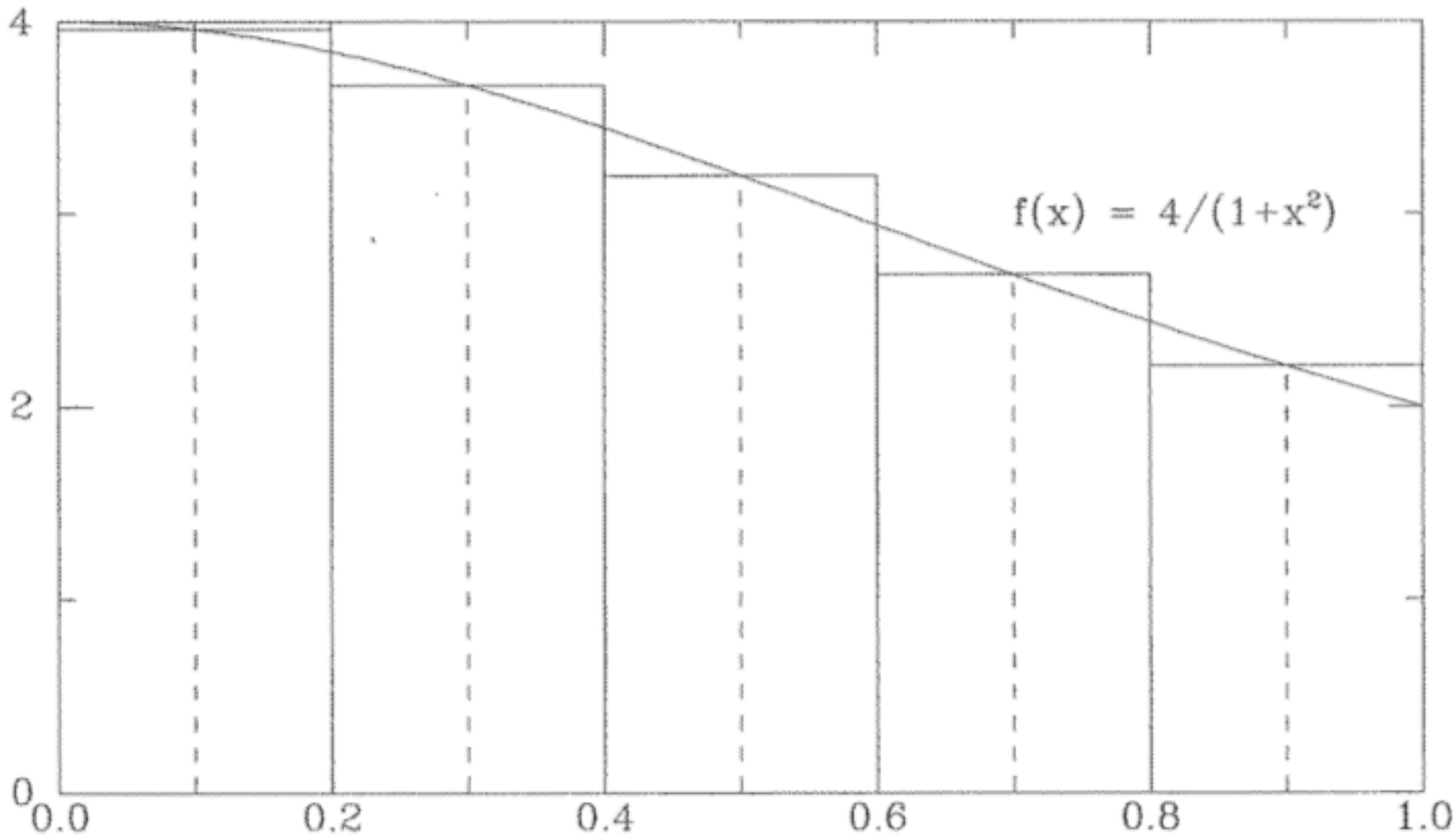
```
int MPI_Init(int *argc, char ***argv)

int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

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

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

int MPI_Finalize()
```

Ch.3: Using MPI in Simple Programs

- Using MPI_Reduce

```
h = 1.0 / (double) n;
sum = 0.0;
for (i = rank + 1; i <= n; i += size) {
    x = h * ((double)i - 0.5);
    sum += (4.0 / (1.0 + x*x));
}
mypi = h * sum;

MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI::DOUBLE,
                       MPI::SUM, 0);
if (rank == 0)
    cout << "pi is approximately " << pi
         << ", Error is " << fabs(pi - PI25DT)
         << endl;
```

Estimating PI



Ch.3: Using MPI in Simple Programs

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

- Gather data from all nodes
- Accumulate to root
- Using the specified operation
  - Aggregates: mean, sum
  - Extrema: min, max
  - User defined functions

Reduce Functions

- All operations are “algebraic” in that they can be applied in any order.

Representation	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical and
MPI_BAND	Bit-wise and
MPI_LOR	Logical or
MPI_BOR	Bit-wise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bit-wise exclusive or
MPI_MAXLOC	Maximum value and corresponding index
MPI_MINLOC	Minimum value and corresponding index

More MPI Features

- Advanced messaging features
  - Scatter (send-to-all)
  - Gather (receive-from-all)
- Data types
  - Arrays, vectors, sparse structures
- Topologies and grids
  - Automatic spatial decomposition
  - Automatic parallelization
  - Cartesian grids

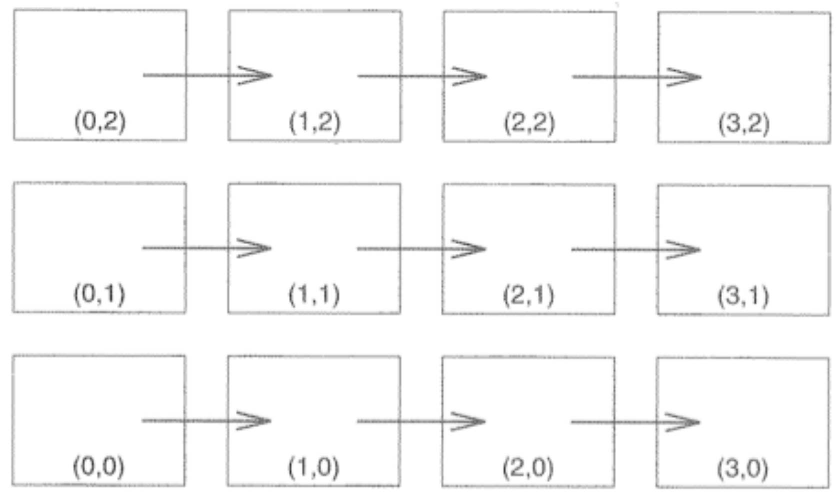
Chapter 4: Intermediate MPI

- MPI supports Cartesian decomposition and access
  - Allow MPI system to pick topology, rather than programmer
  - Useful for universality or complex topologies (maybe)

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,int *isperiodic,
                    int reorder, MPI_Comm *new_comm)

int MPI_Cart_shift(MPI_Comm comm, int direction, int displ,int *src, int *dest)

int MPI_Cart_get(MPI_Comm comm, int maxdims, int *dims, int *isperiodic,
                 int *coords)

int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)
```

Chapter 4: Intermediate MPI

- Arrows show the shift by one from MPI_cart_shift()




```
integer dims(2)
logical isperiodic(2), reorder

dims(1)       = 4
dims(2)       = 3
isperiodic(1) = .false.
isperiodic(2) = .false.
reorder       = .true.
ndim          = 2
call MPI_CART_CREATE( MPI_COMM_WORLD, ndim, dims, isperiodic, &
                      reorder, comm2d, ierr )
```

Chapter 4: Intermediate MPI

- Create a virtual topology
  - Defined by the MPI runtime
- Access functions to map
  - Get rank by coordinates
  - Get coordinates by rank
  - Find neighbors by shift

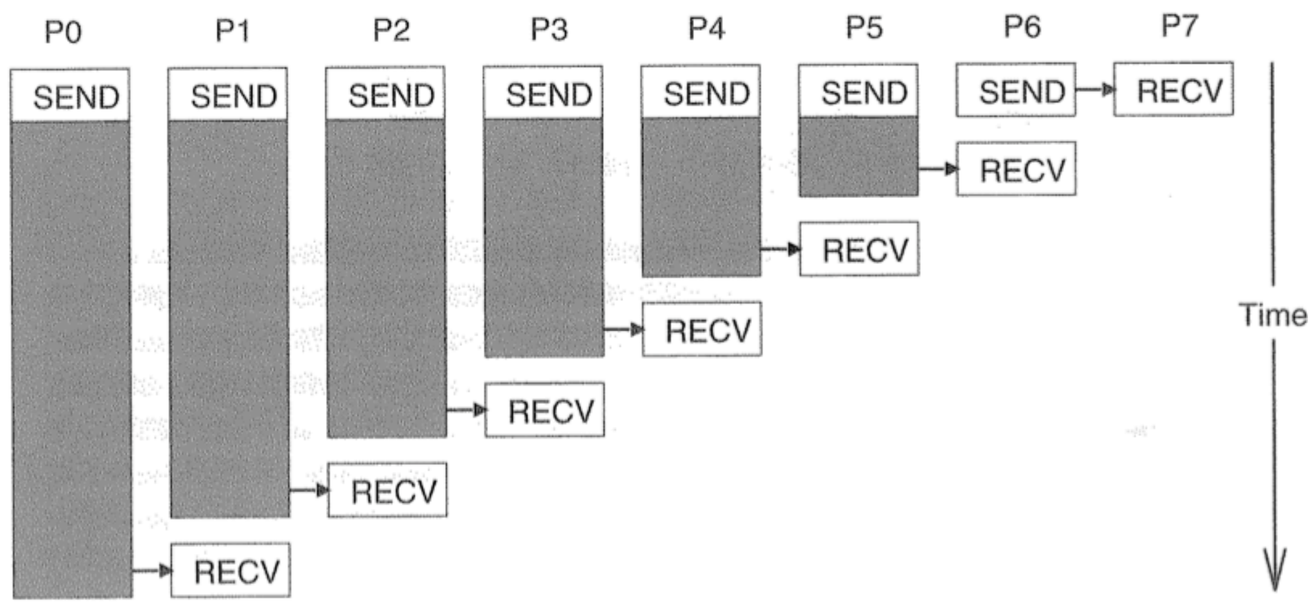
```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,int *isperiodic,
                    int reorder, MPI_Comm *new_comm)

int MPI_Cart_shift(MPI_Comm comm, int direction, int displ,int *src, int *dest)

int MPI_Cart_get(MPI_Comm comm, int maxdims, int *dims, int *isperiodic,
                 int *coords)

int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)
```

Chapter 4: Intermediate MPI

- Cartesian coordinates in MPI are rarely used
  - Vendors don't do a great job implementing
  - Codes tend to be hand optimized for specific machines
- But, they are becoming more necessary
  - As cluster machines have routing hierarchies
  - For machines with multiple networks with complex topologies (BG/L)

Chapter 4: Intermediate MPI

- For pairwise exchange, MPI_SendRecv
  - Always non-blocking
  - Useless: doesn't implement pairwise communication



Ch. 5: Adv. Msg. Passing in MPI

- Type Maps
  - General representation of sparse data
  - Typemap = { (type0,offset0),(type1,offset1),…,(typen,offsetn)}
- Abbreviated type maps
  - Vector: regular gaps
  - Indexed: same type, list of offsets
- Gather, the opposite of broadcast
  - MPI_Gather: collect all data at one process
  - MPI_AllGather: all to all communication, gather then broadcast
- Support for dynamically allocated data
  - Pack and unpack

Defining Type Maps

- MPI bindings for contiguous typemaps

```
int MPI_Type_contiguous(int count, MPI_Datatype oldtype,
                        MPI_Datatype *newtype)

int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent)

int MPI_Type_size(MPI_Datatype datatype, MPI_Aint *size)

int MPI_Type_lb(MPI_Datatype datatype, MPI_Aint *displacement)

int MPI_Type_ub(MPI_Datatype datatype, MPI_Aint *displacement)
```

MPI_TYPE_VECTOR

29	30	31	32	33	34	35
22	23	24	25	26	27	28
15	16	17	18	19	20	21
8	9	10	11	12	13	14
1	2	3	4	5	6	7

```
MPI_TYPE_VECTOR( 5, 1, 7, MPI_DOUBLE_PRECISION, newtype, ierr )
```

MPI_TYPE_indexed

```
MPI_Type_indexed( n_to_move, elmsize, elmoffset,
                  particletype, &sendtype );
MPI_Type_commit( &sendtype );
MPI_Send( myparticles, 1, sendtype, dest, tag, comm );
MPI_Type_free( &sendtype );
```

Figure 5.4
Sketch of code to move particles from one process to another

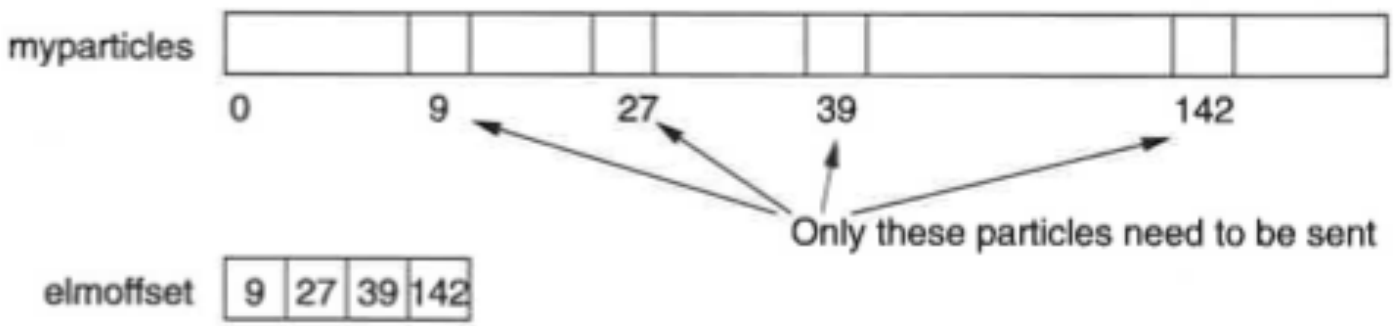


Figure 5.5
Illustration of the array_of_displacements argument in Figure 5.4

All Type Maps

- Contiguous: an array with type width indexing
- Vector: strided access within arrays
- Hvector: vector with byte offsets
  - For heterogeneous data, e.g. an int64 among float32
- Indexed: a list of specific offset
- Hindexed: indexed by bytes rather than type widths
- Struct: general typemap

Pack and Unpack

- Reduce number of messages for irregular data
  - Can be easier/more concise than a type map
  - Supports dynamic data better than type maps
  - Incrementally add/remove data from a contiguous buffer
  - Send data type MPI_Packed

```
int MPI_Pack(void* inbuf, int incount, MPI_Datatype datatype, void *outbuf,
             int outsize, int *position,  MPI_Comm comm)

int MPI_Unpack(void* inbuf, int insize, int *position, void *outbuf, int outcount,
             MPI_Datatype datatype, MPI_Comm comm)

int MPI_Pack_size(int incount, MPI_Datatype datatype, MPI_Comm comm,
             int *size)
```

V

- YOU HAD A WHOLE SECTION ON COLLECTIVE OPERATIONS AND REDUCE FROM PACHECO AND RUNGER AND YOU LOST THEM.

Chapter 6: Parallel Libraries

- Extensions
  - PDEs
  - Linear algebra
  - FFTW

- MPI supports many features to aid in the construction of parallel libraries
  - Create communicators other than COMM_WORLD
  - Use tags for message ordering

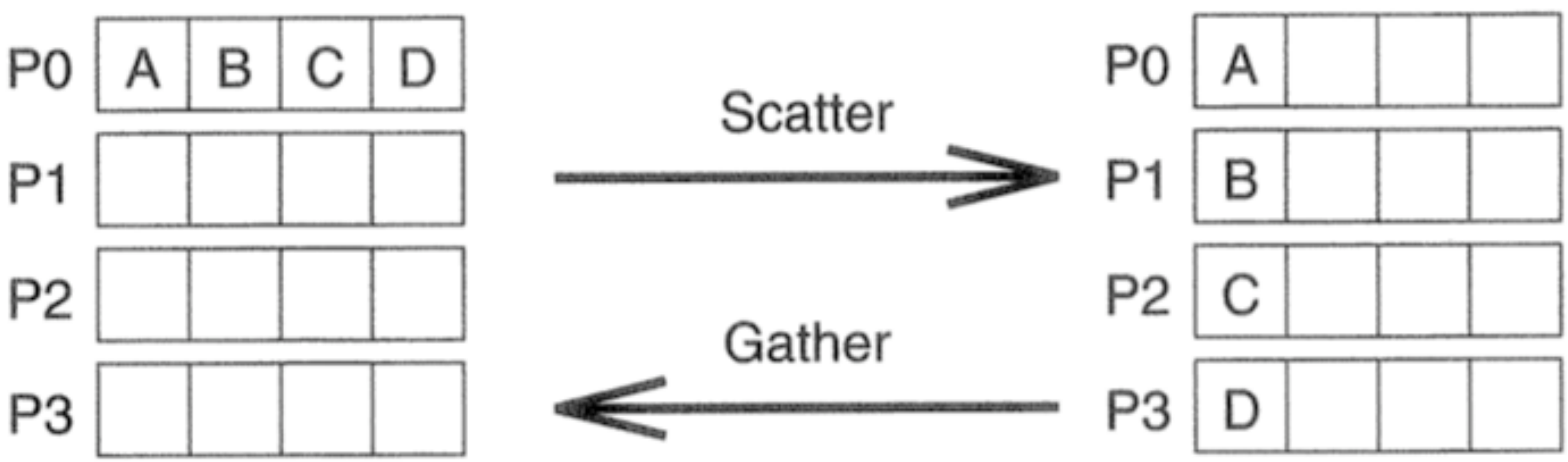
Ch. 7: Other Features of MPI

Weirdness that has led to the failure of MPI 2.0

- Shared-memory support
- Inter-domain communication

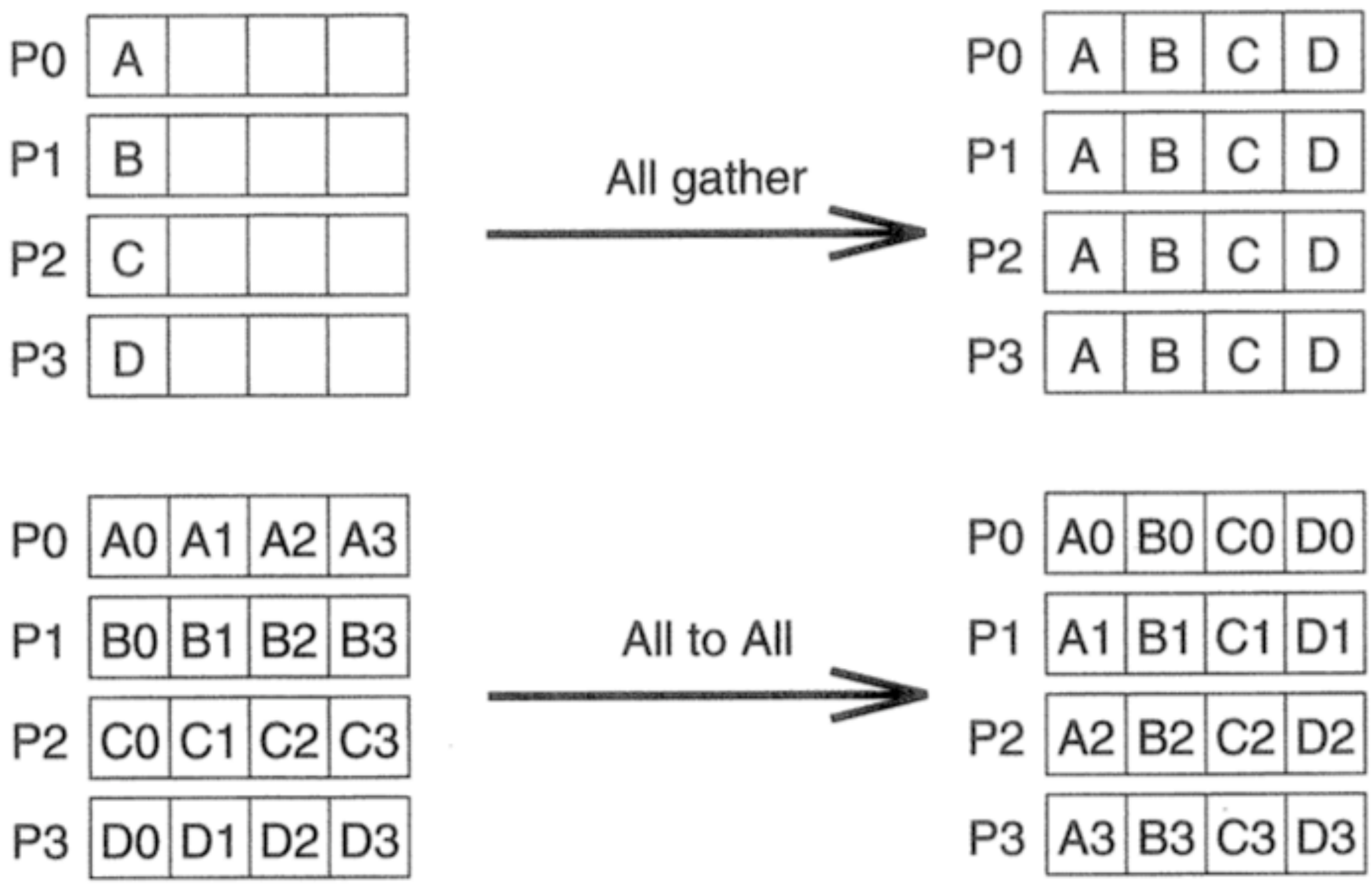
Goodness

- Scatter/Gather support



Ch. 7: Other Features of MPI

- Scatter/Gather variants



Other Stuff

- Ch. 8: Understanding How MPI Implementations Work
- Ch. 9: Comparing MPI w/ Other Systems for IPC
- Ch. 10: Beyond Message Passing
  - Parallel I/O
  - MPI 2.0