Programming Using the Message Passing Paradigm

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University

Thank Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar for providing slides.

Topic Overview

- Topologies and Embedding
- Groups and Communicators
- Collective Communication and Computation Operations

Topologies and Embeddings

- MPI allows a programmer to organize processors into logical k-dimensional meshes.
- The processor ids in `MPI_COMM_WORLD` can be mapped to other communicators (corresponding to higher-dimensional meshes) in many ways.
- The goodness of any such mapping is determined by the interaction pattern of the underlying program and the topology of the machine.
- MPI does not provide the programmer any control over these mappings.

Topologies and Embeddings

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

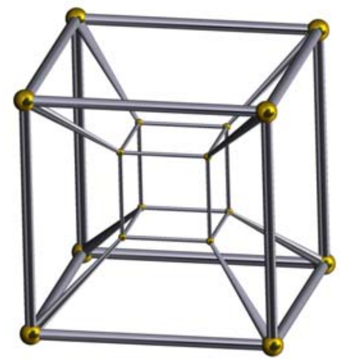
(a) Row−major mapping

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

(b) Column−major mapping

0	3	4	5
1	2	7	6
14	13	8	9
15	12	11	10

(c) Space−filling curve mapping



(d) 4-d Hybercube.

Different ways to map a set of processes to a two-dimensional grid. (a) and (b) show a row- and column-wise mapping of these processes, (c) shows a mapping that follows a space-filling curve (dotted line), and (d) shows a mapping in which neighboring processes are directly connected in a hypercube.

Creating and Using Cartesian Topologies

- We can create cartesian topologies using the function:

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,
          int *periods, int reorder, MPI_Comm *comm_cart)
```

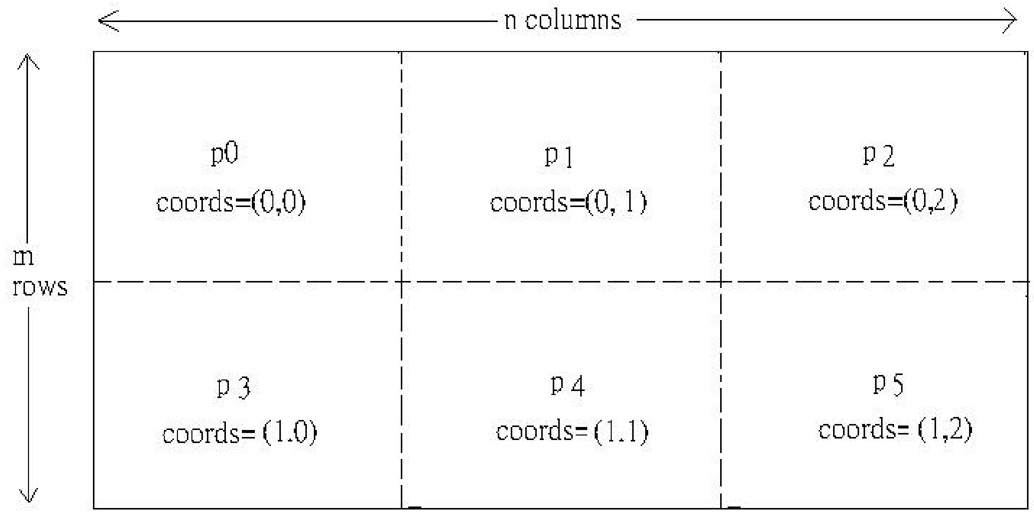
  This function takes the processes in the old communicator and creates a new communicator with `dims` dimensions.

- Each processor can now be identified in this new cartesian topology by a vector of dimension `dims`.

- ndims = number of dimensions.
- `*dims` → the number of elements in each dimension.
- periods → nonzero (True) for wrapping.
- reorder → reorder the processor.

Creating and Using Cartesian Topologies

```
dims[0]=2;
dims[1]=3;
periods[0]=periods[1]=1;
```



2D array distributed on a 2x3 process grid

- Since sending and receiving messages still require (one-dimensional) ranks, MPI provides routines to convert ranks to cartesian coordinates and vice-versa.

```
int MPI_Cart_coord(MPI_Comm comm_cart, int rank, int maxdims,
        int *coords)
int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int *rank)
```

Creating and Using Cartesian Topologies

- The most common operation on cartesian topologies is a shift. To determine the rank of source and destination of such shifts, MPI provides the following function:

```
int MPI_Cart_shift(MPI_Comm comm_cart, int dir, int s_step,
        int *rank_source, int *rank_dest)
MPI_Cart_shift(comm_2d,1,-1,&shiftsource,&shiftdest);
// 1: in 2nd dim (row-wise), -1: to the left
```

- `dir`: 0 (Ist dim, column-wise $\downarrow$), 1(2nd dim, row-wise$\rightarrow$).
- `s_step`: Positive, Ist dim, column-wise $\downarrow$, 2nd dim, row-wise$\rightarrow$.

Groups and Communicators

- In many parallel algorithms, communication operations need to be restricted to certain subsets of processes.

- MPI provides mechanisms for partitioning the group of processes that belong to a communicator into subgroups each corresponding to a different communicator.

Groups and Communicators

- In many parallel algorithms, processes are arranged in a virtual grid, and in different steps of the algorithm, communication needs to be restricted to a different subset of the grid.

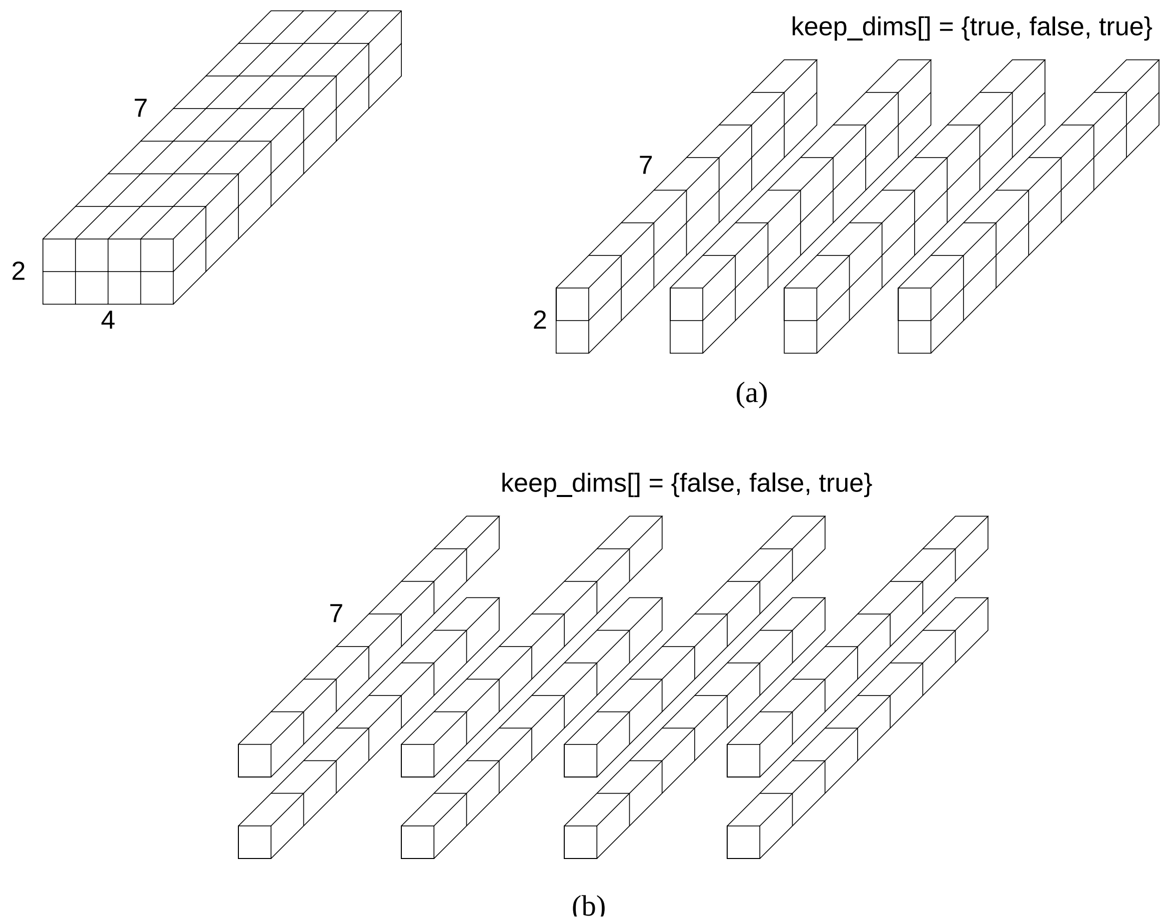
- MPI provides a convenient way to partition a Cartesian topology to form lower-dimensional grids:

```
int MPI_Cart_sub(MPI_Comm comm_cart, int *keep_dims,
        MPI_Comm *comm_subcart)
keep_dims[0]=1;keep_dims[1]=0;
// 3 column sub-topologies are created
```

- If `keep_dims[i]` is true (non-zero value in C) then the `i`th dimension is retained in the new sub-topology.

- The coordinate of a process in a sub-topology created by `MPI_Cart_sub` can be obtained from its coordinate in the original topology by disregarding the coordinates that correspond to the dimensions that were not retained.

Groups and Communicators



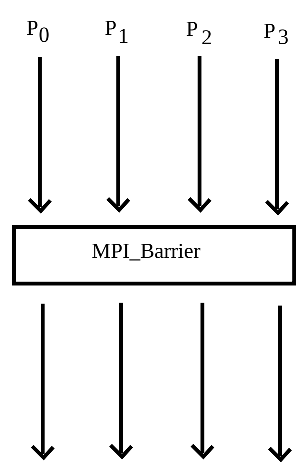
Splitting a Cartesian topology of size $2 \times 4 \times 7$ into (a) four subgroups of size $2 \times 1 \times 7$, and (b) eight subgroups of size $1 \times 1 \times 7$.

- Homework: Hypercube example.

Collective Communication and Computation Operations

- MPI provides an extensive set of functions for performing common collective communication operations.
- Each of these operations is defined over a group corresponding to the communicator.
- All processors in a communicator must call these operations.
- The barrier synchronization operation is performed in MPI using:

```
int MPI_Barrier(MPI_Comm comm)
```



Compute the Global Sum

```
int *global_sum;
....
global_sum=(int*)malloc((npes)*sizeof(int));
for(i=0;i<4;i++){
  a[i]= myrank*4 + i;
  local_sum += a[i];
}
printf("My rank %d a= %d,%d,%d,%d sum= %d \n",myrank,a[0],a[1],a[2],a[3],local_sum);
```

```
if(myrank !=0)
  MPI_Send(&local_sum,1,MPI_INT,0,1,MPI_COMM_WORLD);
else
  global_sum[0]= local_sum;
if(myrank == 0){
  for(i=1;i<npes;i++)
  MPI_Recv(global_sum+i,1,MPI_INT,i,1,MPI_COMM_WORLD,&status);
}
MPI_Barrier(MPI_COMM_WORLD);
if(myrank ==0){
  for(i=1;i<npes;i++)
    global_sum[0] += global_sum[i];
  printf("My rank %d sum = %d \n",myrank, global_sum[0]);
}
MPI_Bcast(global_sum,1,MPI_INT,0,MPI_COMM_WORLD);
printf("My rank %d sum = %d \n",myrank, global_sum[0]);
  ....
```

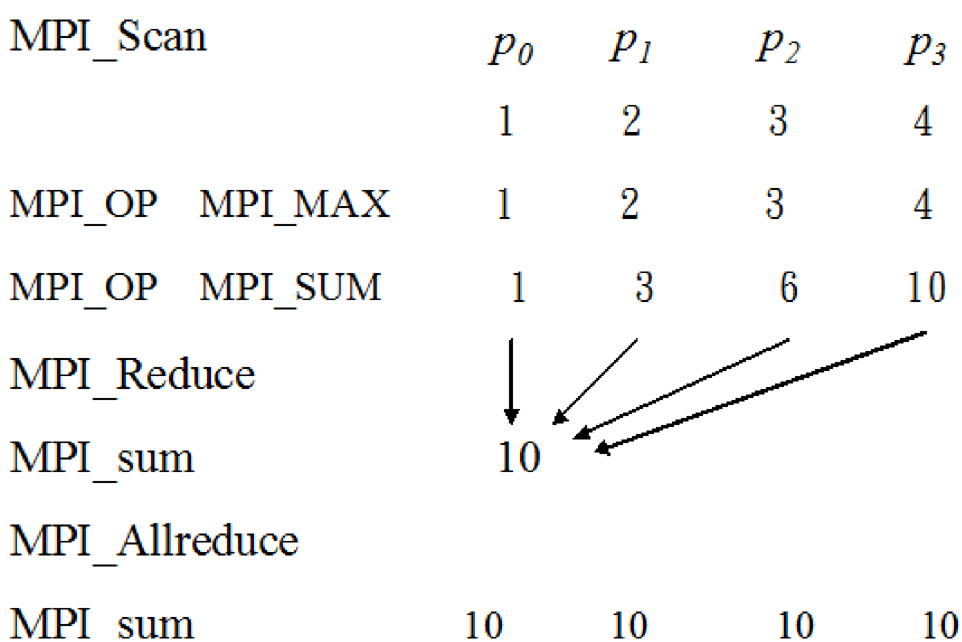
Collective Communication Operations

- The one-to-all broadcast operation is:

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

- The all-to-one reduction operation is:

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



Predefined Reduction Operations

Operation	Meaning	Datatypes
`MPI_MAX`	Maximum	C integers and floating point
`MPI_MIN`	Minimum	C integers and floating point
`MPI_SUM`	Sum	C integers and floating point
`MPI_PROD`	Product	C integers and floating point
`MPI_LAND`	Logical AND	C integers
`MPI_BAND`	Bit-wise AND	C integers and byte
`MPI_LOR`	Logical OR	C integers
`MPI_BOR`	Bit-wise OR	C integers and byte
`MPI_LXOR`	Logical XOR	C integers
`MPI_BXOR`	Bit-wise XOR	C integers and byte
`MPI_MAXLOC`	max-min value-location	Data-pairs
`MPI_MINLOC`	min-min value-location	Data-pairs

Collective Communication Operations

MPI datatypes for data-pairs used with the `MPI_MAXLOC` and `MPI_MINLOC` reduction operations.

MPI Datatype	C Datatype
`MPI_2INT`	pair of `int`s
`MPI_SHORT_INT`	`short` and `int`
`MPI_LONG_INT`	`long` and `int`
`MPI_LONG_DOUBLE_INT`	`long double` and `int`
`MPI_FLOAT_INT`	`float` and `int`
`MPI_DOUBLE_INT`	`double` and `int`

Collective Communication Operations

- If the result of the reduction operation is needed by all processes, MPI provides:

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

- To compute prefix-sums, MPI provides:

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

- Homework: Using collective operators to write a mpi code to compute standard deviation of n numbers using m processors.

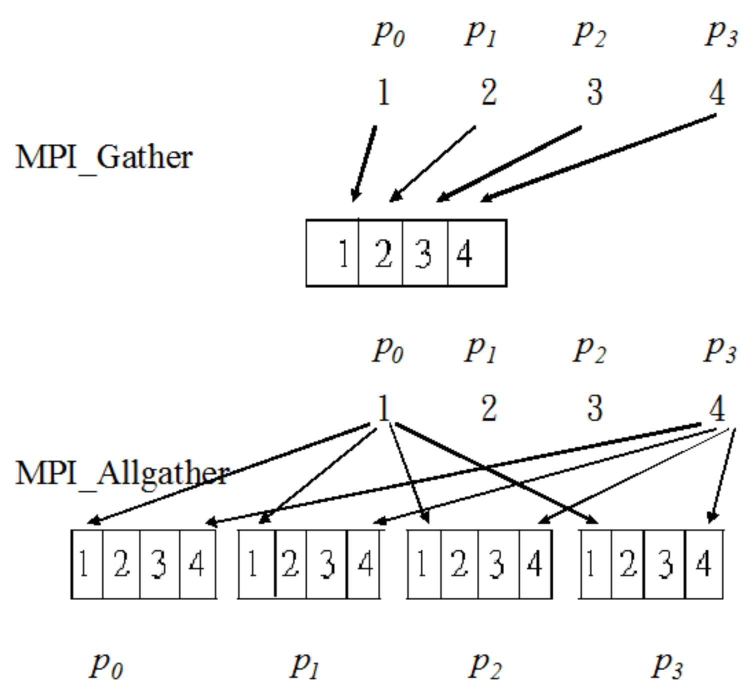
Collective Communication Operations

- The gather operation is performed in MPI using:

```
int MPI_Gather(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, int target, MPI_Comm comm)
```

- MPI also provides the `MPI_Allgather` function in which the data are gathered at all the processes.

```
int MPI_Allgather(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, MPI_Comm comm)
```



Collective Communication Operations

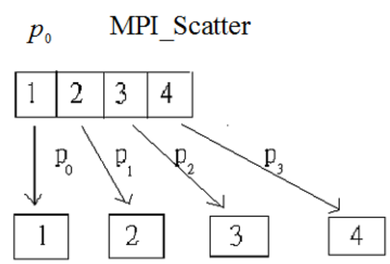
The corresponding scatter operation is:

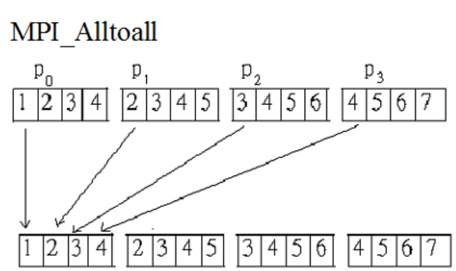
```
int MPI_Scatter(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, int source, MPI_Comm comm)
```

- The all-to-all personalized communication operation is performed by:

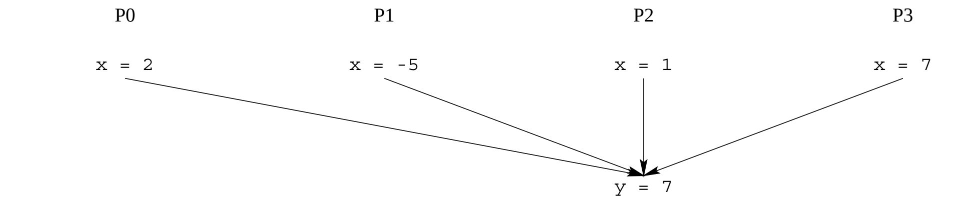
```
int MPI_Alltoall(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, MPI_Comm comm)
```

- Using this core set of collective operations, a number of programs can be greatly simplified.

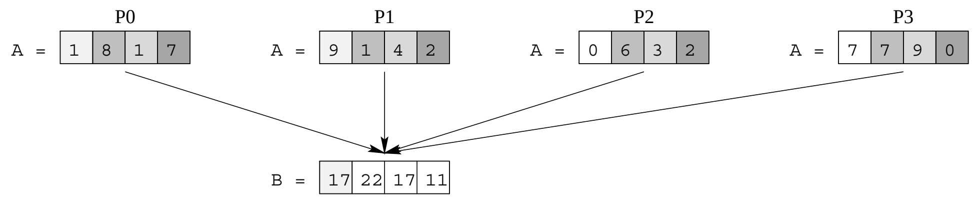




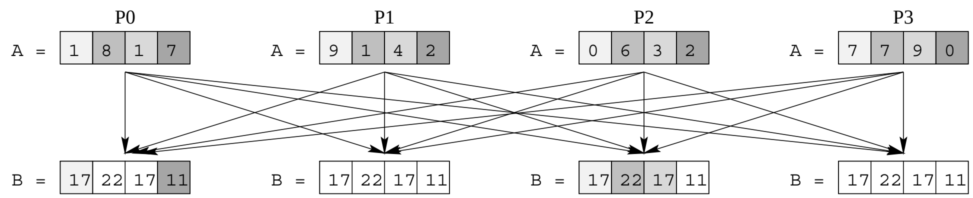
Collective Communication Operations



(a) Reduce(x, y, P2, MAX)



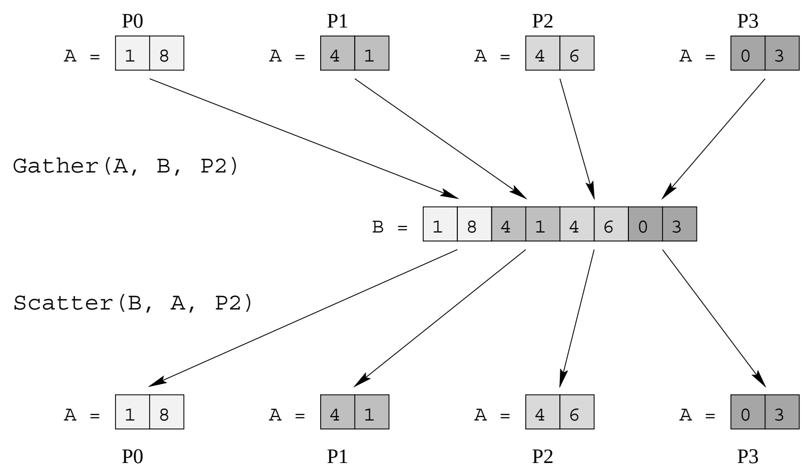
(b) Reduce(A, B, P1, SUM)



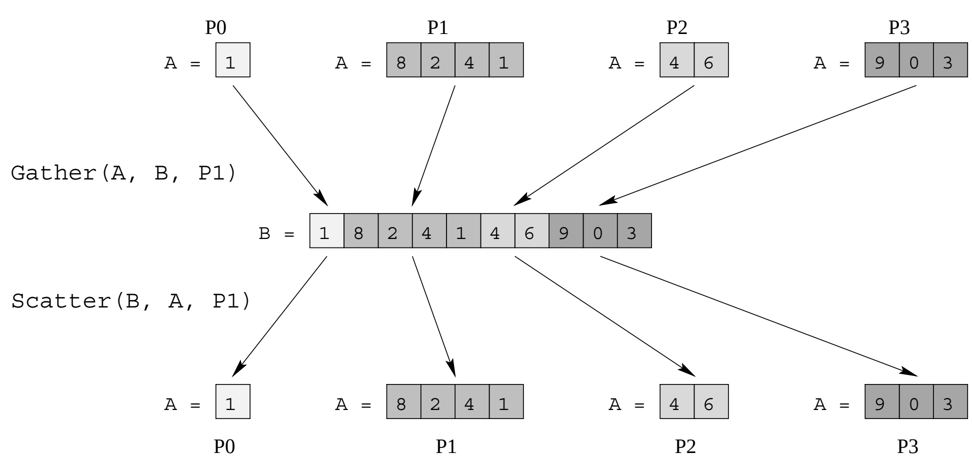
(c) AllReduce(A, B, SUM)

Collective Communication Operations

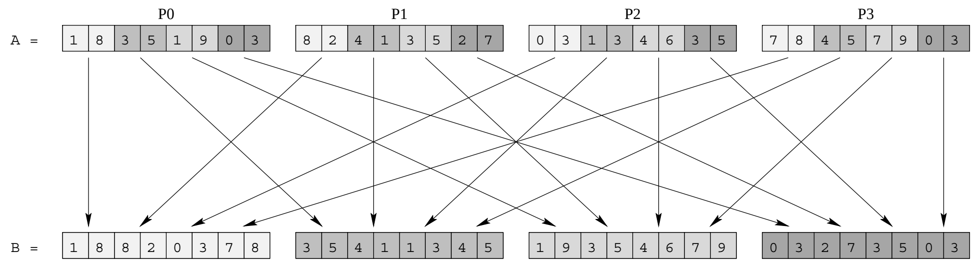
(a) Equal-Size Gather and Scatter Operations



(b) Unequal-Size Gather and Scatter Operations



Collective Communication Operations



AllToAll Operation of Equal Size Data