Communication Optimization for Parallel Processing

Lecture 5

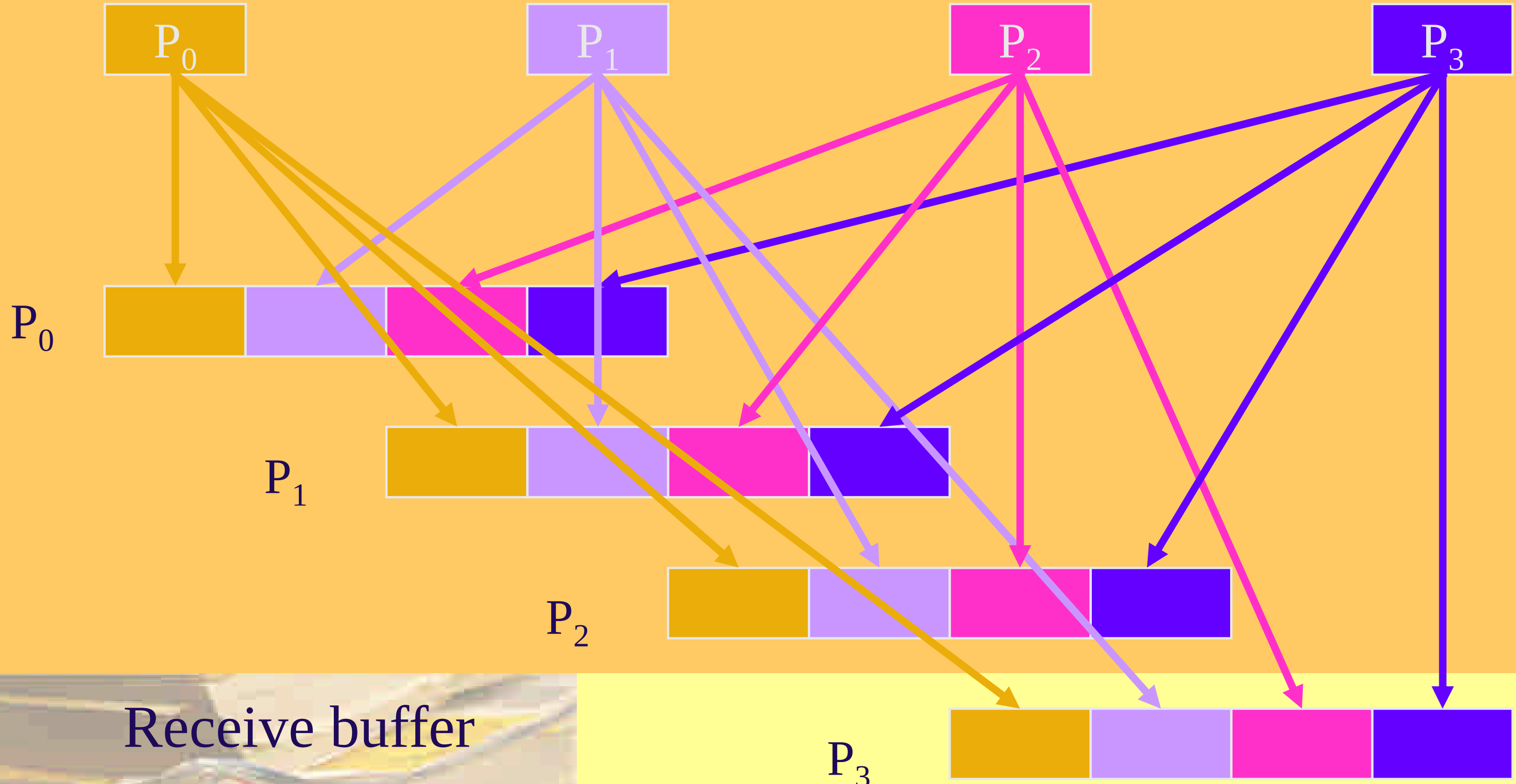
Pangfeng Liu, Department of Computer Science and Information Engineering, National Taiwan University.

Advanced Point-to-Point Communication

- Simultaneous send and receive
- Synchronous and asynchronous comunication

Everyone Gathers the Data

Send buffer

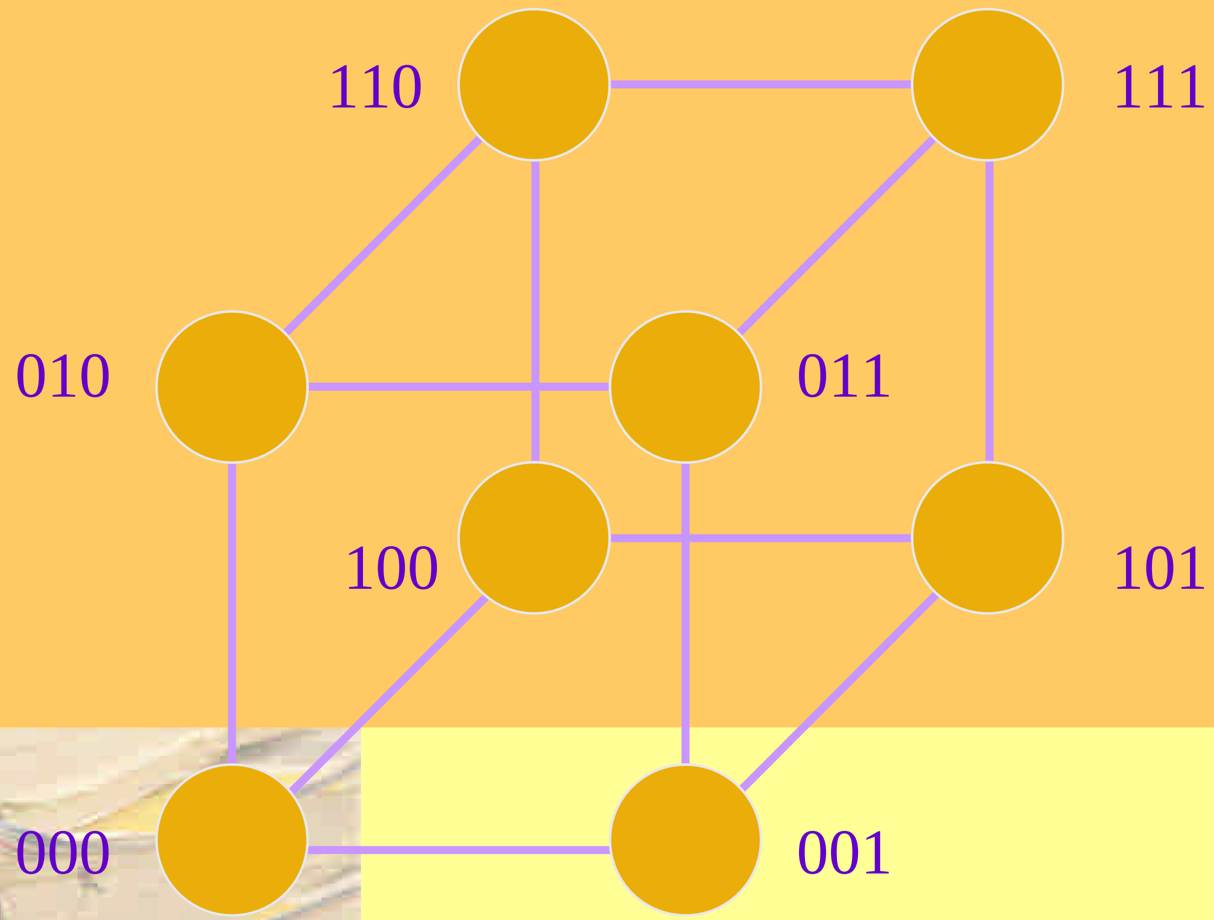


Receive buffer

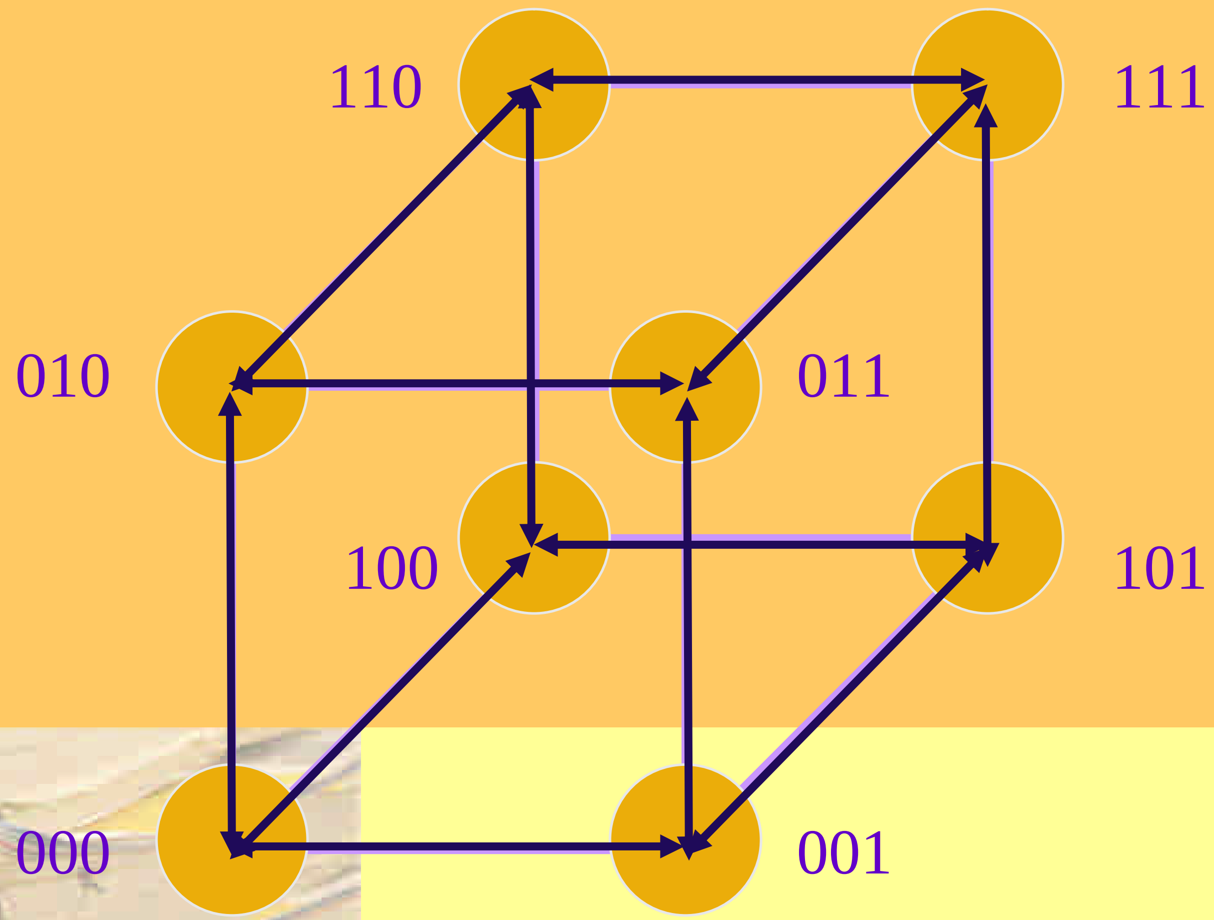
Hypercube

- A hypercube has *2^d* nodes.
  - *d* is the dimension
- Each hypercube node can be easily identify by a binary number
- A node *n* is connected to *d* nodes whose binary number differ from *n* in exactly one bit.
  - Hamming distance

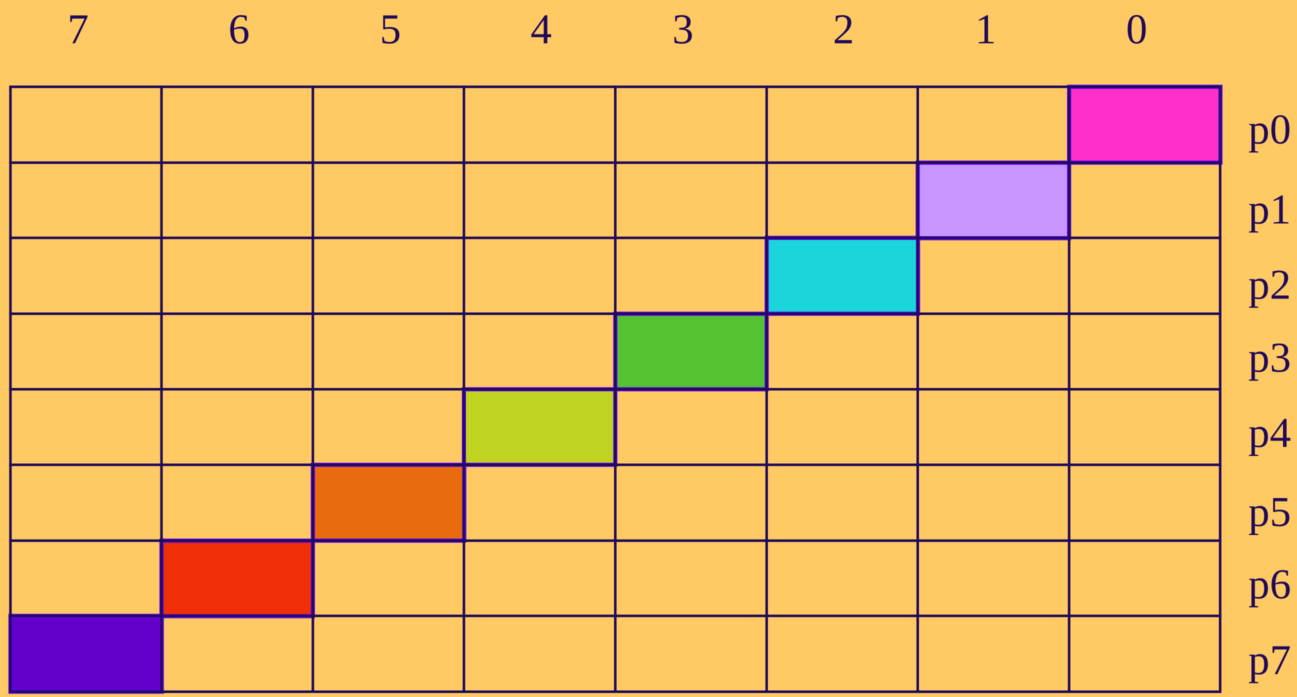
Hypercube



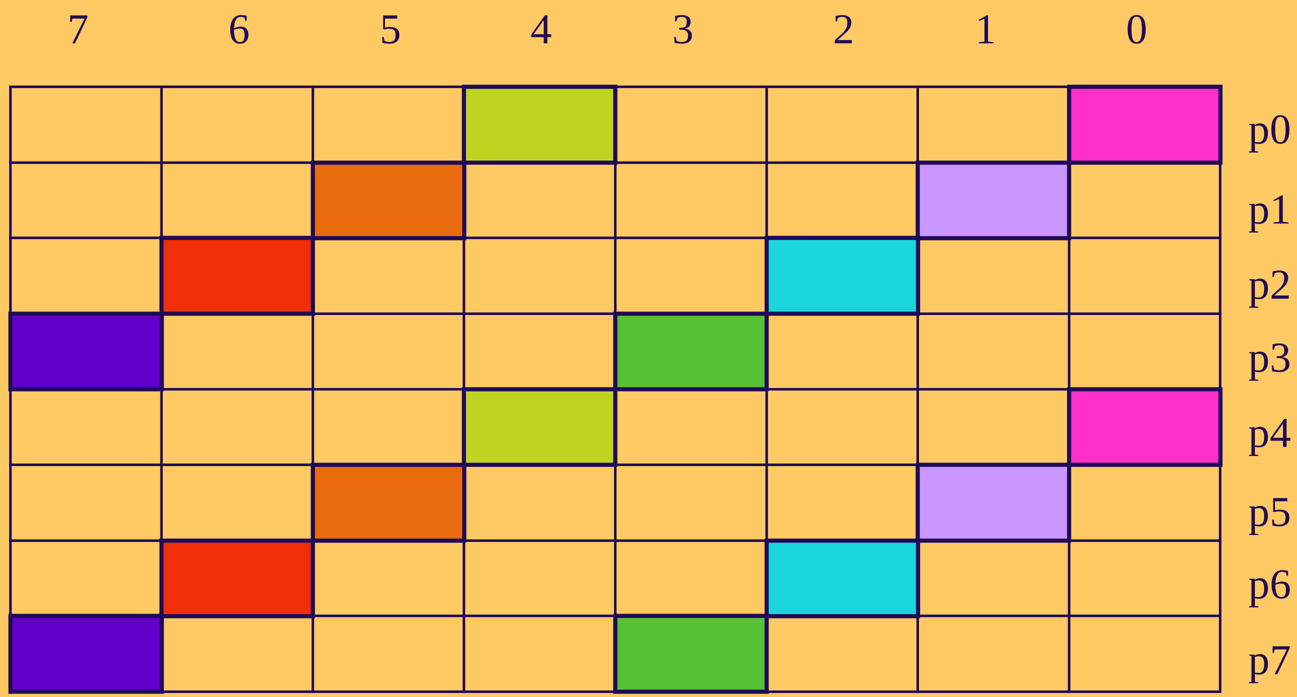
Hypercube All-gather



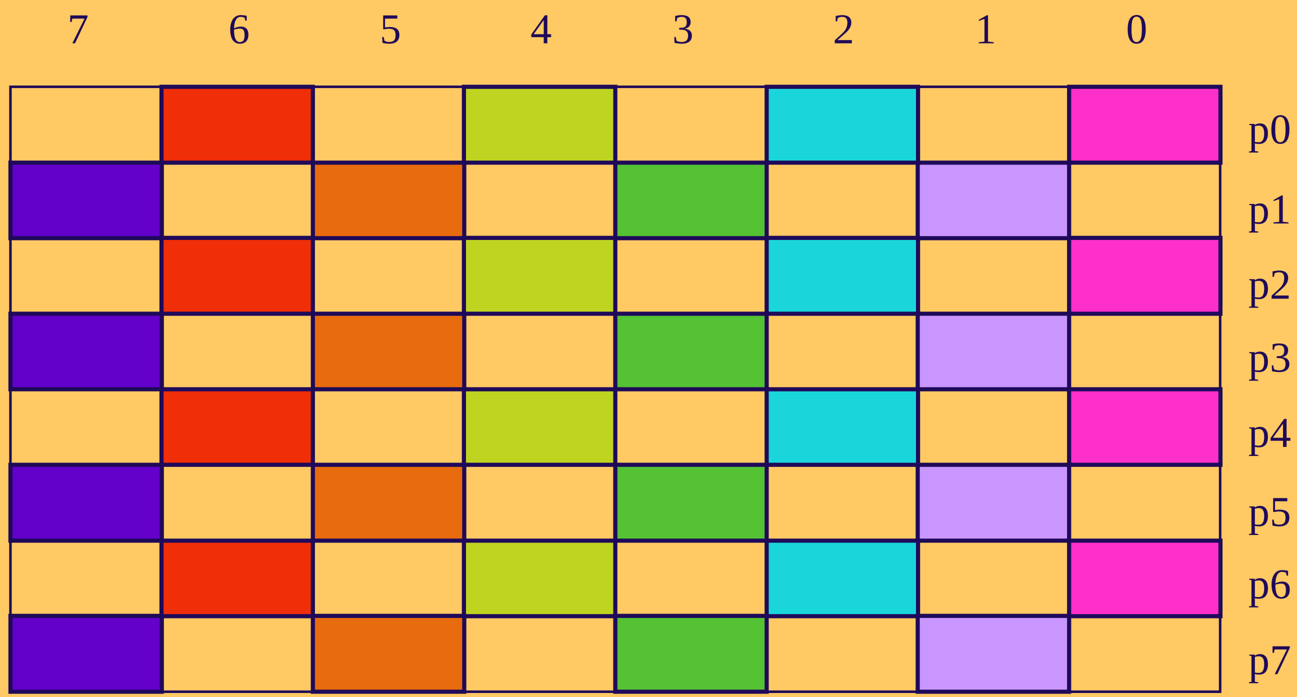
Initial Configuration



Initial Configuration



Initial Configuration



Initial Configuration

Hypercube All-gather

- Has *d* phases, and each phase corresponds to a hypercube dimension.
- Use MPI_Send and MPI_Recv to exchange data between neighboring processes.
- Use “exclusive or” to compute the index of the neighbor.
- The calculation of sending and receiving offset is important.
- Details about type are removed.

The Main Program

```
#define MAX 128
#define LOCAL_MAX 128

main(int argc, char* argv[]) {
   int p, my_rank, I, blocksize;
   float x[LOCAL_MAX], y[MAX];
   MPI_Comm  io_comm;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_dup(MPI_COMM_WORLD, &io_comm);
   Cache_io_rank(MPI_COMM_WORLD, io_comm);
   Cscanf(io_comm,"Enter the local array size","%d", &blocksize);

   while(blocksize > 0) {
      for (i = 0; i < blocksize; i++)
         x[i] = (float) my_rank;
      Allgather_cube(x, blocksize, y, MPI_COMM_WORLD);
      Print_arrays(io_comm, "Gathered_arrays", y, blocksize);
      Cscanf(io_comm,"Enter the local array size", "%d", &blocksize);
   }
   MPI_Finalize();
}  /* main */
```

All-Gather Hypercube Style

```
void  Allgather_cube(float x[], int blocksize, float y[], MPI_Comm comm)
{
   MPI_Comm_size(comm, &p);
   MPI_Comm_rank(comm, &my_rank);

   for (i = 0; i < blocksize; i++)
      y[i + my_rank * blocksize] = x[i];

   d = log_base2(p);
   eor_bit = 1 << (d-1);
   and_bits = (1 << d) - 1;

   for (stage = 0; stage < d; stage++) {
      partner = my_rank ^ eor_bit;
      send_offset = (my_rank & and_bits) * blocksize;
      recv_offset = (partner & and_bits) * blocksize;

      MPI_Send(y + send_offset, 1, hole_type, partner, 0, comm);
      MPI_Recv(y + recv_offset, 1, hole_type, partner, 0, comm, &status);

      eor_bit = eor_bit >> 1;
      and_bits = and_bits >> 1;
   }
} /* Allgather_cube */
```

Send-Receive

- Two processors want to exchange information.
- The order by which the two processors send and receive is critical.
  - Deadlock could occur.

Processor A	Processor B
Send data to B; Receive data from B	Send data to A; Receive data from A

Processor A	Processor B
Receive data from B; Send data to B;	Send data to A; Receive data from A;

Temporary Buffering

- Data could be overwritten.
- Much like the case to exchange the values of two variables.
  - temp = a; a = b; b = temp;

Processor A

Receive data from B and place into temp;
Send data to B;
Put temp into data;

Processor B

Send data to A;
Receive data from A;

Send/Recv Interface

- MPI_Sendrecv
- Send and receive data simultaneously.
- Two sets of parameters
  - Sending
    - void *send_buf, int send_count, MPI_Datatype,
    - int dest, int sendtag
  - Receiving
    - void * recv_buf, int recv_count, MPI_Datatype,
    - int source, int recvtag

Send/Recv Interface

- MPI_Sendrecv_replace
- Similar to MPI_Sendrecv, but can specify the same buffer as sending *and* receiving.
- Recall that in Fox's algorithm, we use MPI_Sendrecv_replace to shift the sub-matrix within a column.

Non-Blocking Message Passing

- Blocking communication routines do not return until the communication finishes.
- This “blocking” effect may cause deadlock, and inflexibility in programming.
- Non-blocking communication routines return immediately.
  - The MPI system starts processing the buffers, so data within the buffer should not be modified.
  - A “handle” must be provided to test whether the communication has finished or not.

Programming Interface

- MPI_Isend
  - Send a message without waiting for receiver.
  - The routine returns immediately, after the MPI system has been informed that it can start copying data out of the sending buffer.
  - Has a similar interface as MPI_Send, but with an extra output parameter MPI_Request *request.

Programming Interface

- MPI_Irecv
  - Start receiving a message without waiting for sender.
  - The routine returns immediately, after the MPI system has been informed that it can start copying data into the receiving buffer.
  - Has a similar interface as MPI_Recv, but with an extra output parameter MPI_Request *request.

Query the End Condition

- MPI_Wait
- Wait for a non-blocking communication to finish.
  - Requires a MPI_Request *request to identify the communication to wait for.

Non-Blocking Hypercube All-gather

- Has log n phases, and each phase corresponds to a hypercube dimension.
- Use MPI_Isend and MPI_Irecv to send/receive data in non-blocking mode.
- Use MPI_Wait to wait for the non-blocking communication.
  - Both non-blocking send and receive must finish before going into the next phase.

Non-Blocking Hypercube All-gather

```
void  Allgather_cube(float x[], int blocksize, float y[], MPI_Comm comm)
{
   int i, d, p, my_rank;
   unsigned eor_bit;
   unsigned and_bits;
   int stage, partner;
   MPI_Datatype  hole_type;
   int send_offset, recv_offset;
   MPI_Status status;
   MPI_Request send_request;
   MPI_Request recv_request;

   MPI_Comm_size(comm, &p);
   MPI_Comm_rank(comm, &my_rank);

   /* Copy x into correct location in y */
   for (i = 0; i < blocksize; i++)
      y[i + my_rank*blocksize] = x[i];
```

Non-Blocking Send/Recv

```
    d = log_base2(p);
    eor_bit = 1 << (d-1);
    and_bits = (1 << d) - 1;

    partner = my_rank ^ eor_bit;
    send_offset = (my_rank & and_bits)*blocksize;
    recv_offset = (partner & and_bits)*blocksize;

    for (stage = 0; stage < d; stage++) {
      MPI_Isend(y + send_offset, 1, hole_type, partner, 0, comm, &send_request);
      MPI_Irecv(y + recv_offset, 1, hole_type, partner, 0, comm, &recv_request);

      if (stage < d-1) {
        eor_bit >>= 1;
        and_bits >>= 1;
        partner = my_rank ^ eor_bit;
        send_offset = (my_rank & and_bits) * blocksize;
        recv_offset = (partner & and_bits) * blocksize;
      }
      MPI_Wait(&send_request, &status);
      MPI_Wait(&recv_request, &status);
    }  /* for */
} /* Allgather_cube */
```