Parallel Scientific Computation

MPI 2

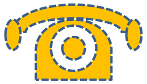
J.-H. Parq
IPCST
Seoul National University

Point-to-Point Communication

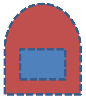
- Blocking communication
  - Your program stops until the message reaches the receiver or a buffer.
- Non-blocking communication
  - Your program proceeds while communicating the message.

Point-to-Point Communication

- Synchronous send
  - Sending a message after the receiver gets ready



- Buffered send
  - Sending a message immediately into a system buffer where the receiver reads later



- Ready send
  - Sending a message immediately, assuming the receiver is ready (Dangerous! Be cautious!)

Blocking Communication

- **Standard** send
  - Synchronous or Buffered
  - Usually, synchronous for large messages and buffered for small ones
  - MPI_SEND(buf, count, datatype, dest, tag, comm, ierror)
  - int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Blocking Communication

- Synchronous send
  - MPI_SSEND / MPI_Ssend
- Buffered send
  - MPI_BSEND / MPI_Bsend
  - If you want to change the system buffer, use
    - MPI_BUFFER_ATTACH / MPI_Buffer_attach
    - MPI_BUFFER_DETACH / MPI_Buffer_detach
- Ready send
  - MPI_RSEND / MPI_Rsend

Blocking Communication

- Receive
  - MPI_RECV(buf, count, datatype, source, tag, comm, status, ierror)
  - int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Point-to-Point Communication

- Wildcards
  - MPI_ANY_SOURCE: for source
  - MPI_ANY_TAG: for tag
- Tag
  - Non-negative integer (0 to MPI_TAG_UB or 32767)
- Special constants
  - MPI_PROC_NULL: none for source or dest
  - MPI_STATUS_IGNORE / MPI_STATUSES_IGNORE
    - None / null array for status in Fortran or C
    - Some versions of Fortran MPI may not have these.

Point-to-Point Communication

- Status
  - Message information
  - In Fortran, it has to be an integer array of size MPI_STATUS_SIZE
  - **Source:** status(MPI_SOURCE) / status.MPI_SOURCE
  - **Tag:** status(MPI_TAG) / status.MPI_TAG
  - **Error:** status(MPI_ERROR) / status.MPI_ERROR
  - **Size:** by a function (to be discussed later)

Example: Ring (Fortran)

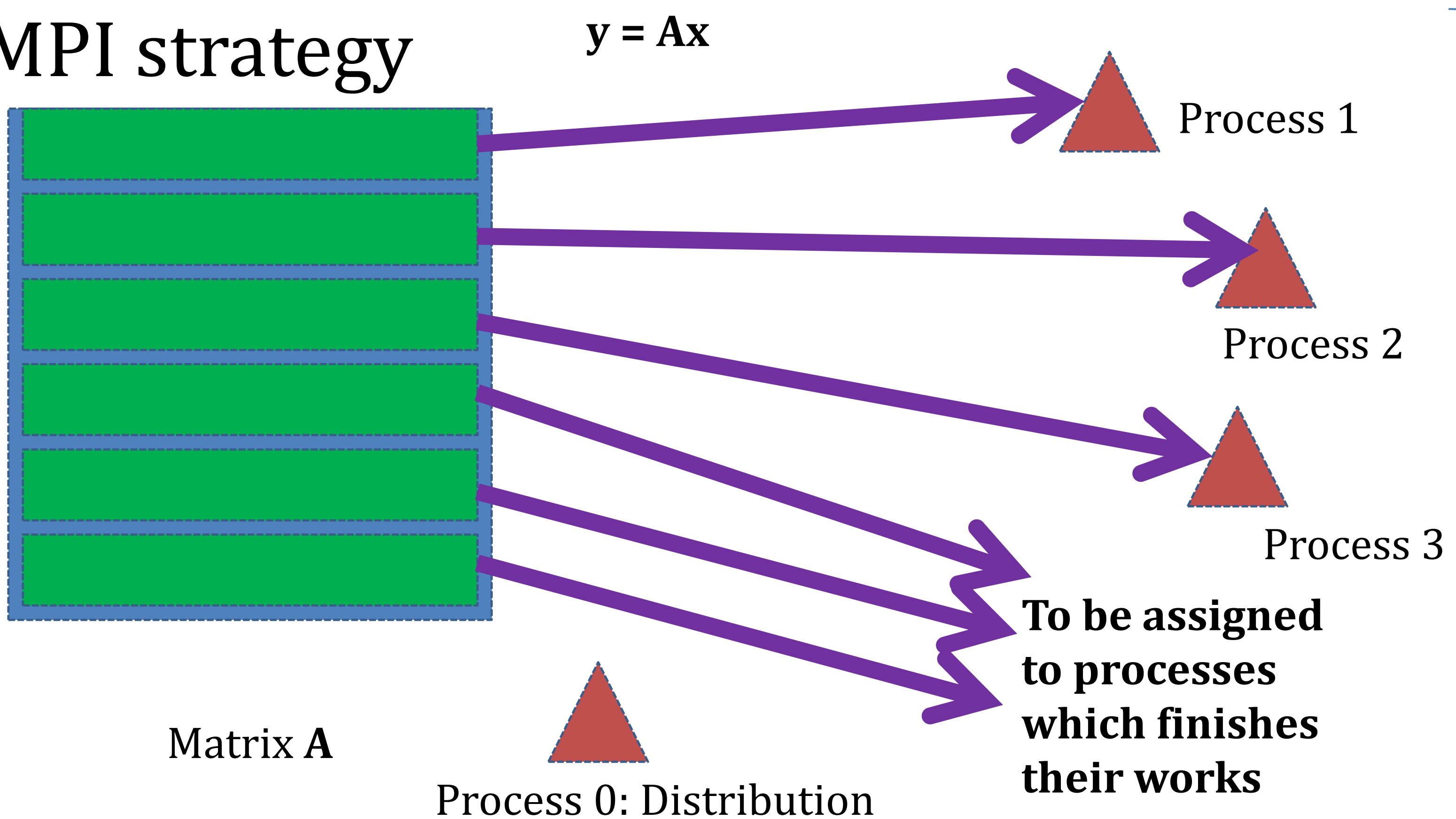
```
IF (Rank == 0) THEN
   Baton = 1
   call MPI_SEND(Baton, 1, MPI_INT, 1, 999, MPI_COMM_WORLD, ierr)
   call MPI_RECV(Baton, 1, MPI_INT, size-1, 999, MPI_COMM_WORLD,
     status, ierr)
   print *, 'Baton', Baton, ': Process', size-1, '--> Process 0‘
ELSE
   call MPI_RECV(Baton, 1, MPI_INT, rank-1, 999, MPI_COMM_WORLD,
     status, ierr)
   print *, 'Baton', Baton, ': Process', rank-1, '--> Process', rank
   call MPI_SEND(Baton, 1, MPI_INT, MOD(rank+1,size), 999,
     MPI_COMM_WORLD, ierr);
END IF
```

Example: Ring (C)

```
if (rank == 0) {
    baton = 1;
    MPI_Send(&baton, 1, MPI_INT, 1, 999, MPI_COMM_WORLD);
    MPI_Recv(&baton, 1, MPI_INT, size-1, 999, MPI_COMM_WORLD,
      &status);
    printf("Baton %d: Process %d --> Process 0", baton, size-1);
}
else {
    MPI_Recv(&baton, 1, MPI_INT, rank-1, 999, MPI_COMM_WORLD,
      &status);
    printf("Baton %d: Process %d --> Process %d", baton, rank-1, rank);
    MPI_Send(&baton, 1, MPI_INT, (rank+1)%size, 999,
      MPI_COMM_WORLD);
}
```

Matrix-Vector Multiplication

- MPI strategy



Matrix-Vector Multiplication (C)

```
if (myid == 0) {
  numsent = 0;
  MPI_Bcast(&x[0], numcol, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  for (i=0; i<numprocs-1 && i<numrow; i++) {
    MPI_Send(&A[i][0],numcol,MPI_DOUBLE,i+1,i,MPI_COMM_WORLD);
    numsent++;
  }
  for (i=0; i<numrow; i++) {
    MPI_Recv(&ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
   MPI_COMM_WORLD, &status);
    sender = status.MPI_SOURCE;
    row = status.MPI_TAG;
    y[row] = ans;
    if (numsent < numrow) {
```

Matrix-Vector Multiplication (C)

```
        MPI_Send(&A[numsent][0], numcol, MPI_DOUBLE, sender,
      numsent, MPI_COMM_WORLD);
        numsent++;
      }
      else MPI_Send(MPI_BOTTOM, 0, MPI_DOUBLE, sender,  numrow,
      MPI_COMM_WORLD);
    }
}
else {
    MPI_Bcast(&x[0], numcol, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    buffer = (double*)malloc(sizeof(double)*numcol);
    po = 1;
    while (myid <= numrow && po) {
      MPI_Recv(buffer, numcol, MPI_DOUBLE, 0, MPI_ANY_TAG,
      MPI_COMM_WORLD, &status);
```

Matrix-Vector Multiplication (C)

```
      row = status.MPI_TAG;
      if (row < numrow) {
         ans = 0.0;
         for (j=0; j<ncol; j++) ans += buffer[j]*x[j];
         MPI_Send(&ans, 1, MPI_DOUBLE, 0, row, MPI_COMM_WORLD);
      }
      else po = 0;
   }
   free(buffer);
}
```

Matrix-Vector Multiplication

- In case of Fortran
  - The most different part is that of sending a row from the master (or distributer)
  - The master process also needs the temporary array (named 'buffer' in the previous example).

```
……
[allocating the 'buffer' array variable if it is not statically declared]
DO I = 1,min(numprocs-1,numrow)
   DO J = 1,numcol
      buffer(J) = A(I,J)
   END DO
   call MPI_SEND(buffer, numcol, MPI_DOUBLE_PRECISION, I, I, &
MPI_COMM_WORLD, ier)
   numsent = numsent + 1
END DO
```

Matrix-Vector Multiplication

```
…….
  IF (numsent > numrow)  THEN
    DO J = 1, numcol
      buffer(J) = A(numsent+1,J)
    END DO
    call MPI_SEND(buffer, numcol, MPI_DOUBLE_PRECISION, sender, &
numsent+1,  MPI_COMM_WORLD, ier)
    numsent = numsent + 1
  ELSE
…….
```

MPI Time Check

1. Command ‘time’
   - In Unix or Linux
   - Total elapsed time
   - Usage) time mpirun –np 8 ./a.exe
2. Time library in C
   - Some Fortran versions also provide time library.
3. MPI functions
   - MPI_WTIME() / MPI_Wtime()
   - MPI_WTICK() / MPI_Wtick()

Latency and Bandwidth

- Linear model
  - Communication time $T_{co} = L + M/B$
- L: latency
  - Delay independent of message size
  - How to find L
    - By 1 byte transmission ($\approx L$)
    - By extrapolation from variation by message size
- B: bandwidth
  - Amount of delivered data (bits) per second

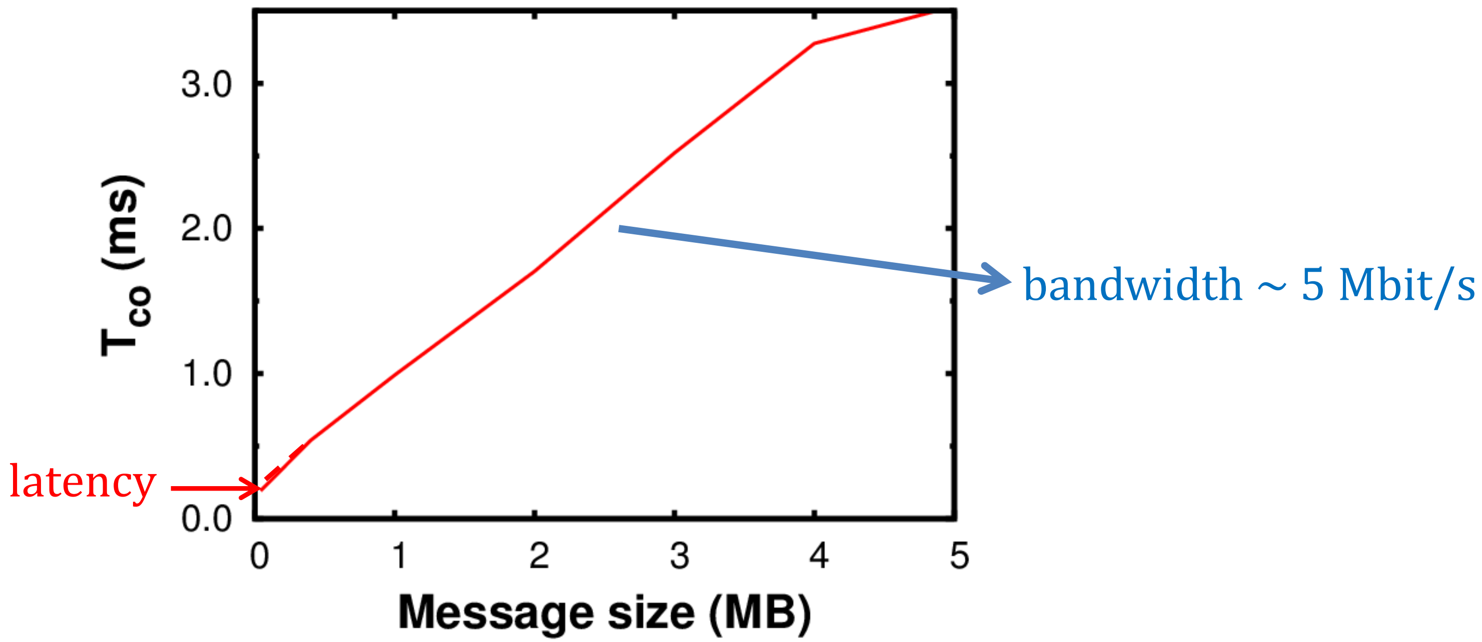
Latency and Bandwidth

- Test code example

```
IF (rank == 0) THEN
   Stime = MPI_WTIME()
   call MPI_SEND(array, count, MPI_INT, 1, tag, comm, ierr)
   call MPI_RECV(array, count, MPI_INT, 1, tag, comm, status, ierr)
   Etime = MPI_WTIME()
   PRINT *, 'Elapsed time =', Etime – Stime
ELSE IF (rank == 1) THEN
   call MPI_RECV(array, count, MPI_INT, 0, tag, comm, status, ierr)
   call MPI_SEND(array, count, MPI_INT, 0, tag, comm, ierr)
END IF
```

Latency and Bandwidth

- Test result example



Message Order

- (send order) = (recv order) for the same tag

- Ex. 1

```
m1 = 1; m2 = 2;
MPI_Send(&m1, ..., 9);
MPI_Send(&m2, ..., 9);
......
MPI_Recv(&m2, ..., 9, &stat);
MPI_Recv(&m1, ..., 9, &stat);
```

➢ Result: m1 = 2 & m2 = 1

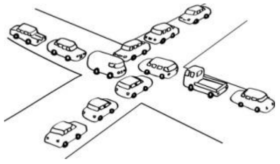
- Ex. 2

```
m1 = 1; m2 = 2;
MPI_Send(&m1, ..., 1);
MPI_Send(&m2, ..., 2);
......
MPI_Recv(&m2, ..., 2, &stat);
MPI_Recv(&m1, ..., 1, &stat);
```

➢ Result: m1 = 1 & m2 = 2

Deadlock

- In case of a ring or exchange of synchronous blocking communication, it's a deadlock if every process is awaiting.
- Ex.)



```
IF (rank == 0) THEN
   MPI_SSEND(buf, count, MPI_REAL, 1, tag, comm, ierror)
   MPI_RECV(buf, count, MPI_REAL 1, tag, comm, status, ierror)
ELSE
   MPI_SSEND(buf, count, MPI_REAL, 0, tag, comm, ierror)
   MPI_RECV(buf, count, MPI_REAL, 0, tag, comm, status, ierror)
END IF
```

Exchange

- Send & Receive
  - MPI_SENDRECV(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status, ierror)
  - int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)
  - They avoid deadlocks.

Exchange

- Examples
  - call MPI_SENDRECV(A(1,e), nx, MPI_DOUBLE_PRECISON, nbrtop, 1, A(1,s-1), nx, MPI_DOUBLE_PRECISON, nbrbottom, 1, comm1d, status, ierr)
  - MPI_Sendrecv(&A[e][0], ny, MPI_DOUBLE, nbrleft, 1, &A[s-1][0], ny, MPI_DOUBLE, nbrright, 1, comm1d, &status);

Non-blocking Communication

- Advantages
  - It enables to **avoid deadlocks**.
  - It **saves time** by executing another work during communication.
- Disadvantages
  - You must be *careful* when you **change values of argument variables or data in your buffer.**
    - Communication or data process can be *influenced*.
  - More **difficult** to develop and fix **codes**

Non-blocking Communication

- Send
  - MPI_ISEND(buf, count, datatype, dest, tag, comm, request, ierror)
  - int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
  - MPI_Issend, MPI_Ibsend, MPI_Irsend: rarely used

Non-blocking Communication

- Receive
  - MPI_IRECV(buf, count, datatype, source, tag, comm, request, ierror)
  - int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
  - The status can be obtained by MPI_Wait.

Non-blocking Communication

- Wait
  - Blocking at that point
  - It awaits until sending or receiving ends.
  - MPI_WAIT(request, status, ierror)
  - int MPI_Wait(MPI_Request *request, MPI_Status *status)
  - Status arguments are ignored for sending.

Non-blocking Communication

- Waitall
  - Waiting for the completion of all sending and receiving with respect to an array of requests
  - MPI_WAITALL(request_count, request_array, status_array, ierror)
  - int MPI_Waitall(int request_count, MPI_Request *request_array, MPI_Status *status_array)

Non-blocking Communication

- Example (Fortran)

```
INTEGER statuses(MPI_STATUS_SIZE,2), req(2)
call MPI_IRECV(A(1,s-1), nx, MPI_DOUBLE_PRECISON, nbrbottom, 1,
     comm1d, req(1), ierr)
call MPI_ISEND(A(1,e), nx, MPI_DOUBLE_PRECISON, nbrtop, 2,
     comm1d, req(2), ierr)
call MPI_WAITALL(2, req, statuses, ierr)
```

- Example (C)

```
MPI_Status statuses[2];
MPI_Request req[2];
MPI_Irecv(&A[s-1][0], ny, MPI_DOUBLE, nbrright, 1, comm1d, &req[0]);
MPI_Isend(&A[e][0], ny, MPI_DOUBLE, nbrleft, 2, comm1d, &req[1]);
MPI_Waitall(2, req, statuses);
```

Non-blocking Communication

- Waitany
  - It stops waiting if any sending or receiving with respect to an array of requests ends.
  - MPI_WAITANY(request_count, request_array, index, status, ierror)
  - int MPI_Waitany(int request_count, MPI_Request *request_array, int *index, MPI_Status *status)

Non-blocking Communication

- Test
  - MPI_Test / MPI_Testall / MPI_Testany
    - Same form as MPI_Wait / MPI_Waitall / MPI_Waitany
  - Not blocking and waiting but just checking completion of sending or receiving
- Probe
  - MPI_Probe / MPI_Iprobe
  - Receiving the status only; It is safe to use MPI_Recv / MPI_Irecv because the sender may still wait for a receiver.
  - Convenient if you want to change the arguments of MPI_Recv / MPI_Irecv tuned to the message

References

- W. Gropp, E. Lusk, and A. Skjellum,
  Using MPI

- C. Evangelinos,
  Parallel Programming for Multicore Machines Using OpenMP and MPI

- MPI forum (www.mpi-forum.org)