Non-blocking Communications

東京大学情報基盤中心　准教授　片桐孝洋

Takahiro Katagiri, Associate Professor,
Information Technology Center, The University of Tokyo

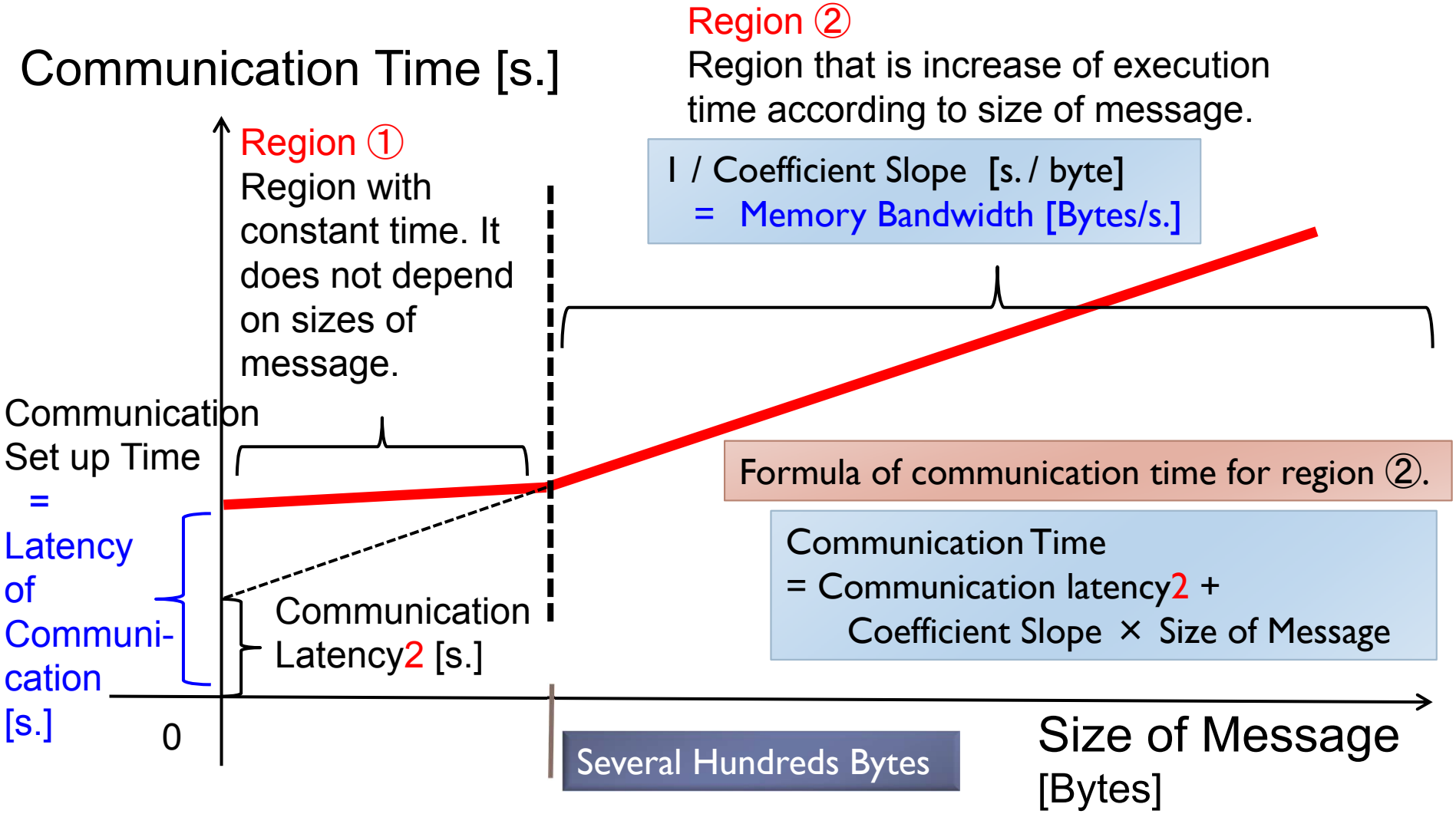
台大数学科学中心　科学計算冬季学校

Agenda

1. Technical terms of MPI for 1-to-1 communications
2. Execution of sample program (Non-blocking Communication)
3. Lessons

How to optimize communications

Size of Message and Times of Communication



Note: Optimization of Communications (1/2)

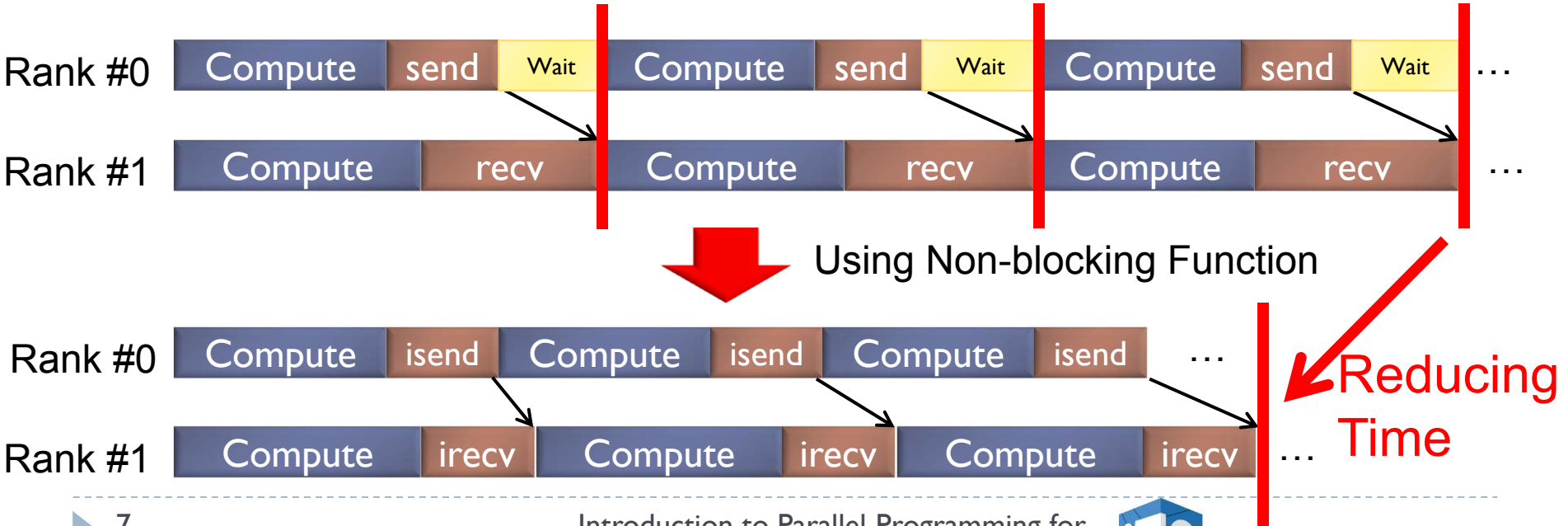
- Knowing pattern of communications in your application in viewpoint of follows to optimize the communication.
  - Whether <Region ①> or <Region ②>?
  - How many times of communications does it happen?
- In case of region ①:
  - "Commutation Latency" is majority of execution time.
  - Reduce times of communications.
    - E.g.) Integrate communications that are sending with small size of messages.
- In case of region ②:
  - "Communication Time" is majority of execution time.
  - Reduce size of massages.
  - E.g.) Do redundant computing and increase computation complexity if it reduces size of messages.

Example of Communication to be Region①

- Sending size of message for reduction (MPI_Allreduce) of dot product is one unit of double precision, thus 8 bytes.
- With respect to 8 bytes, it is same time between MPI_Allreduces between 8 bytes and several bytes.
  - ⇒Integrating several times of dot products can be reduce communication time.
- E.g.) Dot products in Conjugate Gradient (CG) method, which is an iterative solver for linear equations.
  - Simple implementation, there are three dot products per iteration.
  - Hence communication latency is majority for dot products.
  - If we can use multiple iterations for one time, communication time of dot products can be reduced by 1/k time.
    - However it is difficult to converge by using simple implementation.
    - This is hot topic for HPC. It is known as Communication Avoiding CG (CACG).

Note: Optimization of Communications (2/2)

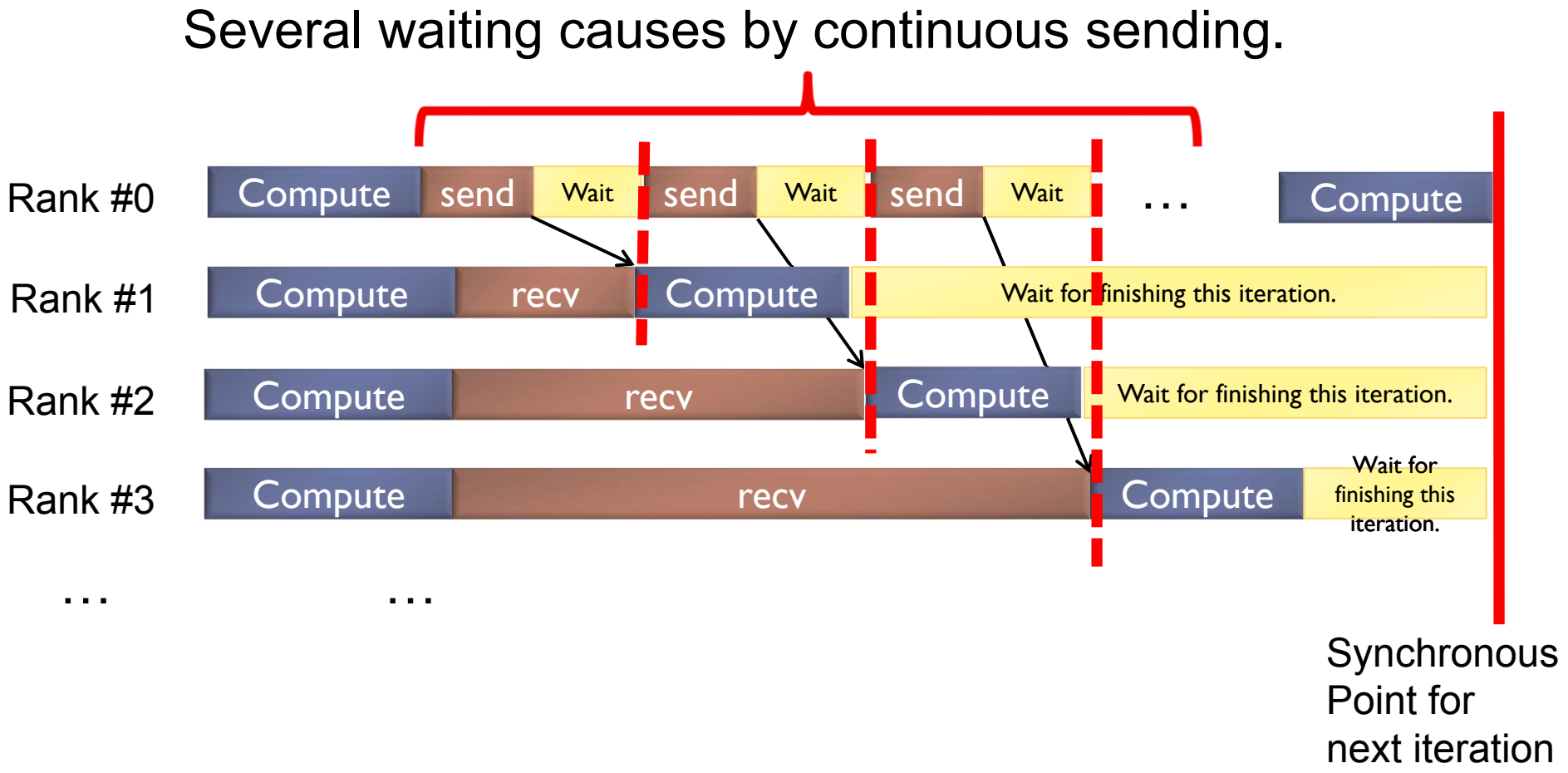
- Reducing "Synchronization Points" contributes fast execution.
  - To use "non-blocking" function of MPI
  - E.g.) Blocking Function MPI_SEND()
    → Non-blocking function MPI_ISEND()
  - Communication and computation simultaneously



Non-blocking Communications: Isend, Irecv, and persistent communication

An Example: Worst Case with Blocking Communication

- If rank #0 has sending data to be used:



Technical Terms of Non-blocking Communication of MPI

Blocking and Non-blocking

1. **Blocking**
   - Do not return when sending/receiving data is stored to buffer area, and until it is reusable for the buffer area.
   - Assure consistency of data on the buffer area.
2. **Non-blocking**
   - Return as soon as possible that whether sending/receiving data is stored to buffer area, or not.
   - Do not assure consistency of data on the buffer area.
     - Keeping consistency of data is duty for users.

Local and Non-local

- Local
  - To finalize procedure, it depends on only process that is executing.
  - Process that do not communicate with the other processes.
- Non-local
  - To finalize procedure, it may depend on MPI procedures on the other processes.
  - Process may not communicate with the other processes.

Communication Modes (In case of sending)

1. Standard Communication Mode (Non-Local) : Default
   - Buffering for sending message is controlled by MPI system.
     - If the massage is buffered: Finalize the sending before finalizing target receiving.
     - If the massage is not buffered: Wait until finalizing the sending.
2. Buffered Communication Mode (Local)
   - Do buffering every time. If there is no area to do buffering, an error is returned.
3. Synchronous Communication Mode (Non-Local)
   - Wait until that buffer area can be reused, and target receiving starts.
4. Ready Communication Mode (The process its own is local)
   - This is executable that target receiving is issued in calling time. Otherwise, an error is returned.
     - Since it can remove "hand shakings" for communication, it can establish high performance.

An Example－MPI_Send

- MPI_Send Function
  - Blocking
  - Standard Communication Mode (Non-Local)
    - Do not return until that buffer area is safe.
    - If buffer area can be allocated:
      Message are buffered. Sending can be finalized before corresponding receiving is calling.
    - If buffer area cannot be allocated:
      Sending cannot be finalized until corresponding receiving is calling, and message is sent to corresponding receiver completely.

Non-blocking Function

- ierr = MPI_Isend(sendbuf, icount, datatype, idest, itag, icomm, irequest);

  - sendbuf : Specify first address of sending array.
  - icount : Integer type. Specify number of elements of sending array.
  - datatype : Integer type. Specify type of sending array.
  - idest : Integer type. Specify rank for process that is issued corresponding receive in icomm.
  - itag : Integer type. Specify tag for receive massage.

Non-blocking Function

- icomm : Integer type. Specify communicator.
  - In default, "MPI_COMM_WORLD" can be specified.
- irequest : MPI_Request type. (An array of Integer type.) An identifier of the sending message is stored. (A communication handler)
- ierr : Integer type. An error code is stored.

Function of Checking for Sending or Receiving

- ierr = MPI_Wait(irequest, istatus);

  - irequest : MPI_Request type. (A array of integer type.) Specify identifier of the sending message (A message handler).
  - istatus : MPI_Status type. (A array of integer type.) Status of receiving is stored.
    - Declare array that number of elements is MPI_STATUS_SIZE.
    - Rank of sending process is stored in istatus[MPI_SOURCE] and its tag is stored in istatus[MPI_TAG].

An Example－MPI_Isend

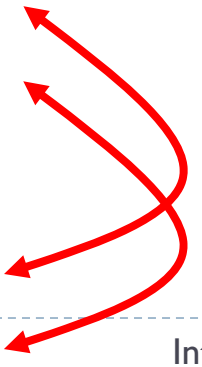
- MPI_Isend Function
  - Non-blocking
  - Standard Communication Mode (Non-Local)
    - Return whether status of communication buffer area.
    - If buffer area can be allocated, massage is buffered, and sending is finalized before corresponding receive is calling.
    - If buffer area cannot be allocated, sending cannot be finalized until that corresponding receive is called, and sending message is copied to receiving area completely.
    - We should understand that this behavior is a case when MPI_Wait function is calling.

Note

- We can understand with the followings
  - MPI_Send Function
    - MPI_Wait function is included in the function;
  - MPI_Isend Function
    - MPI_Wait function is not included in the function. And return to user program as soon as possible;

Note of Parallelization (MPI_Send and MPI_Recv)

- If MPI_Send is called in all processes in advance of receive, process is halted in the place. (cf. Standard Communication Mode) （To describe exactly, it can work in a limited case)
  - In MPI_Send, buffer area cannot allocated due to memory consumption.
  - The process should be waited until buffer area can be reused. A spin-wait (busy wait) is happen.
- To avoid this, implement the following for an example.
  - If number of rank can be devisable with 2:
    - MPI_Send();
    - MPI_Recv();
  - The others:
    - MPI_Recv();
    - MPI_Send();



Corresponding each

TIPS for Non-blocking Functions

- Knowing type of messages without receiving all messages.
  - In case of changing implementation with respect to type of receiving messages.
  - MPI_Probe function (Blocking)
  - MPI_Iprobe function (Non-blocking)
  - MPI_Cancel function (Non-blocking and Local)

MPI_Probe Function

- ierr = MPI_Probe(isource, itag, icomm, istatus) ;
  - isource: Integer type. Specify sending rank.
    - "MPI_ANY_SOURCE" (Integer type) is also describable.
  - itag: Integer type. A number of tag.
    - "MPI_ANY_TAG" (Integer type) is also describable.
  - icomm: Integer type. Communicator.
  - istatus : Status object.
  - If there is message with rank of "isource" and tag of "itag", the function returns.

MPI_Iprobe Function

- ierr = MPI_Iprobe(isource, itag, icomm, iflag, istatus) ;
  - isource: Integer type. Specify sending rank.
    - "MPI_ANY_SOURCE" (Integer type) is also describable.
  - itag: Integer type. A number of tag.
    - "MPI_ANY_TAG" (Integer type) is also describable.
  - icomm: Integer type. Communicator.
  - iflag: Logical type. If there is message with rank of "isource" and tag of "itag", it returns with true.
  - istatus : Status object.

MPI_Cancel Function

- ierr = MPI_Cancel(irequest);

  - irequest: integer type. Communication handler.

- Return as soon as possible before canceling target massage.
- To specify the cancelation, it should be finalized that MPI_Request_free function, MPI_Wait function, and MPI_Test function, or arbitrary functions to operate it.

An Example: Non-blocking Communication (C Language)

```
if (myid == 0)  {
      …
    for (i=1;  i<numprocs;  i++)  {
        ierr = MPI_Isend( &a[0],  N,  MPI_DOUBLE,  i,
            i_loop,  MPI_COMM_WORLD,  &irequest[i] );
    }
 }  else  {
    ierr = MPI_Recv( &a[0],  N,  MPI_DOUBLE,  0,  i_loop,
                MPI_COMM_WORLD,  &istatus );
}
   Computation with a[ ];
if (myid == 0)   {
     for (i=1;  i<numprocs;  i++)  {
         ierr = MPI_Wait(&irequest[i],  &istatus);
     }
}
```

Rank #0 process sends array with length N and type of double from process that rank #1 to rank #numprocs-1.

Processes that rank #1 to rank #numprocs-1 wait for receiving from rank #0.

Rank #0 starts computation unless waiting for receiving of the other ranks

Process of rank #0 is doing spin-wait (busy wait) until finalizing sending data to processes that from rank #1 to rank #numprocs-1.

An Example: Non-blocking Communication (Fortran Language)

```
if (myid .eq. 0)  then
     …
   do i=1, numprocs - 1
      call MPI_ISEND( a,  N,  MPI_DOUBLE_PRECISION,
         i,  i_loop,  MPI_COMM_WORLD,  irequest, ierr )
   enddo
else
      call  MPI_RECV( a,  N,  MPI_DOUBLE_PRECISION ,
         0,  i_loop, MPI_COMM_WORLD,  istatus,  ierr )
endif
   Computation with a( ).
if (myid .eq.  0) then
   do  i=1,  numprocs - 1
      call  MPI_WAIT(irequest(i),  istatus, ierr )
   enddo
endif
```

Rank #0 process sends array with length N and type of double precision from process that rank #1 to rank #numprocs-1.

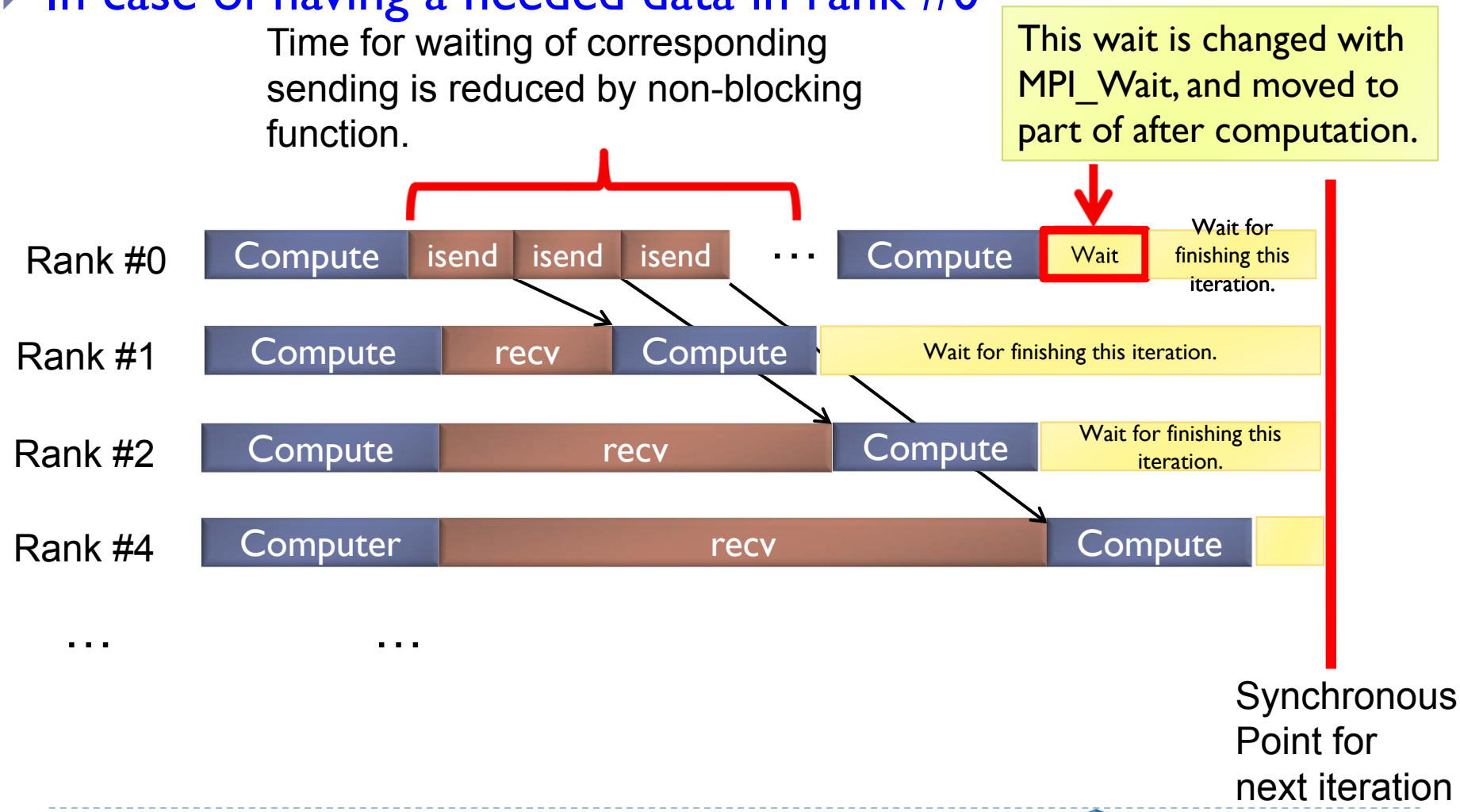
Processes that rank #1 to rank #numprocs-1 wait for receiving from rank #0.

Rank #0 starts computation unless waiting for receiving of the other ranks

Process of rank #0 is doing spin-wait (busy wait) until finalizing sending data to processes that from rank #1 to rank #numprocs-1.

Improvement by Non-blocking Communication

- In case of having a needed data in rank #0



Persistent Communication (1/2)

- If implementation of MPI_ISEND is not supporting to start sending data after calling the function, there is no effect for non-blocking communication.
- However, some implementations do not start sending data for MPI_ISEND until time of calling MPI_WAIT.
  - In this case, there is no effect for non-blocking communication.
- Using "Persistent Communication" may improve the effect of non-blocking communication.
  - MPI-1 supports the persistent communication. Hence usually it is available for your environment.
    - Note: There is different problem that implementation of persistent communication is supporting the above function (communication overlapping) or not. However its performance is better or same in theoretically.

Persistent Communication (2/2)

- How to use persistent communications?
  1. Call an initialization function to set sending information before entering target loop.
  2. Write MPI_START in the point of MPI_SEND.
  3. Function to specify synchronization point, such as MPI_WAIT, MPI_ISEND or same sending functions can be described.
- By using MPI_SEND_INIT to initialize communication information, there is no settings process in MPI_START.
  - In case of multiple sending to same rank, performance is increased or same to a non-blocking function.
- Examples:
  - Explicit methods for domain decomposition method.
  - Implicit methods for iterative solver.

An Example: Persistent Communication (C Language)

```
MPI_Status istatus;
MPI_Request  irequest[numprocs];
…
if (myid == 0) {
  for (i=1; i<numprocs; i++) {
     ierr = MPI_Send_init (a,  N,  MPI_DOUBLE_PRECISION,  i,
                  0,  MPI_COMM_WORLD,  irequest[i] );
  }
}
…
if (myid == 0) {
  for (i=1; i<numprocs; i++) {
    ierr = MPI_Start ( irequest );
  }
}

/* After this , it is same as example of Isend. */
```

Initialize information of sending data before entering main loop.

The massage is sent in here.

An Example: Persistent Communication (Fortran Language)

```
integer istatus(MPI_STATUS_SIZE)
integer irequest(0:MAX_RANK_SIZE)
…
if (myid .eq. 0) then
  do i=1, numprocs-1
    call MPI_SEND_INIT (a, N, MPI_DOUBLE_PRECISION, i,
         0, MPI_COMM_WORLD, irequest(i), ierr)
  enddo
endif
…
if (myid .eq. 0) then
  do i=1, numprocs-1
    call MPI_START (irequest, ierr )
  enddo
endif/* After this , it is same as example of Isend. */
```

Initialize information of sending data before entering main loop.

The massage is sent in here.

Execute a sample program (Non-blocking Communication)

Note: Sample Program of LU Decomposition

- File name of C/Fortran Languages

  lsend-fx.tar

- Change queue name from **lecture** to **lecture6**

In job script file "isend.bash". Then enter "pjsub."

- lecture : Queue for time of out of the lecture.
- lecture6: Queue for time in the lecture.

Execute sample program of MPI_Isend (Common with C and Fortran Languages)

- Type the following commands.

```
$ cp /home/z30082/ISend-fx.tar ./
$ tar xvf ISend-fx.tar
$ cd Isend
```

- Choose one:

```
$ cd C  : For C language users.
$ cd F  : For Fortran language users.
```

- The followings are common. Type them.

```
$ make
$ pjsub isend.bash
```

- After execution, type the follow.

```
$ cat isend.bash.oXXXXXX
```

Output

- The following is obtained. (C Language)

Execution time using MPI_Isend : 30.3248 [sec.]

Explanation of sample program (C Language)

```
if (myid == 0)  {
      …
    for (i=1;  i<numprocs;  i++)  {
        ierr = MPI_Isend( &a[0],  N,  MPI_DOUBLE,  i,
            i_loop,  MPI_COMM_WORLD,  &irequest[i] );
    }
 }  else  {
    ierr = MPI_Recv( &a[0],  N,  MPI_DOUBLE,  0,  i_loop,
                MPI_COMM_WORLD,  &istatus );
}
…
if (myid == 0)   {
     for (i=1;  i<numprocs;  i++)  {
         ierr = MPI_Wait(&irequest[i],  &istatus);
     }
}
```

Rank #0 process sends array with length N and type of double from process that rank #1 to rank #191.

Processes that rank #1 to rank #191 wait for receiving from rank #0.

Process of rank #0 is doing spin-wait (busy wait) until finalizing sending data to processes that from rank #1 to rank #191.

Explanation of sample program (Fortran Language)

```
if (myid .eq. 0)  then
      …
    do i=1, numprocs - 1
       call MPI_ISEND( a,  N,  MPI_DOUBLE_PRECISION,
          i,  i_loop,  MPI_COMM_WORLD,  irequest, ierr )
    enddo
else
       call  MPI_RECV( a,  N,  MPI_DOUBLE_PRECISION ,
          0,  i_loop, MPI_COMM_WORLD,  istatus,  ierr )
endif
  …
if (myid .eq.  0) then
     do  i=1,  numprocs - 1
        call  MPI_WAIT(irequest(i),  istatus, ierr )
     enddo
endif
```

Rank #0 process sends array with length N and type of double precision from process that rank #1 to rank #191.

Processes that rank #1 to rank #191 wait for receiving from rank #0.

Process of rank #0 is doing spin-wait (busy wait) until finalizing sending data to processes that from rank #1 to rank #191.

Lesson

1. Explain that blocking communication in MPI is not always synchronization communication.
2. Survey and summarize functions of blocking and non-blocking of MPI in viewpoint of communication mode.
3. Survey effective condition for sending size of messages, such as N is from 0 to an upper value, for blocking communication (MPI_Send function) to non-blocking communication (MPI_Isend function) by using parallel computers. Then discuss results.