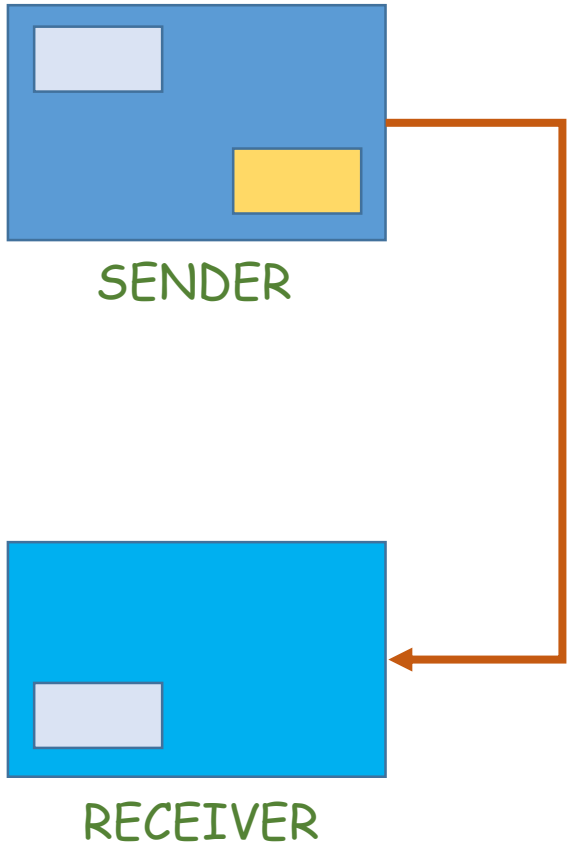
Point-to-point

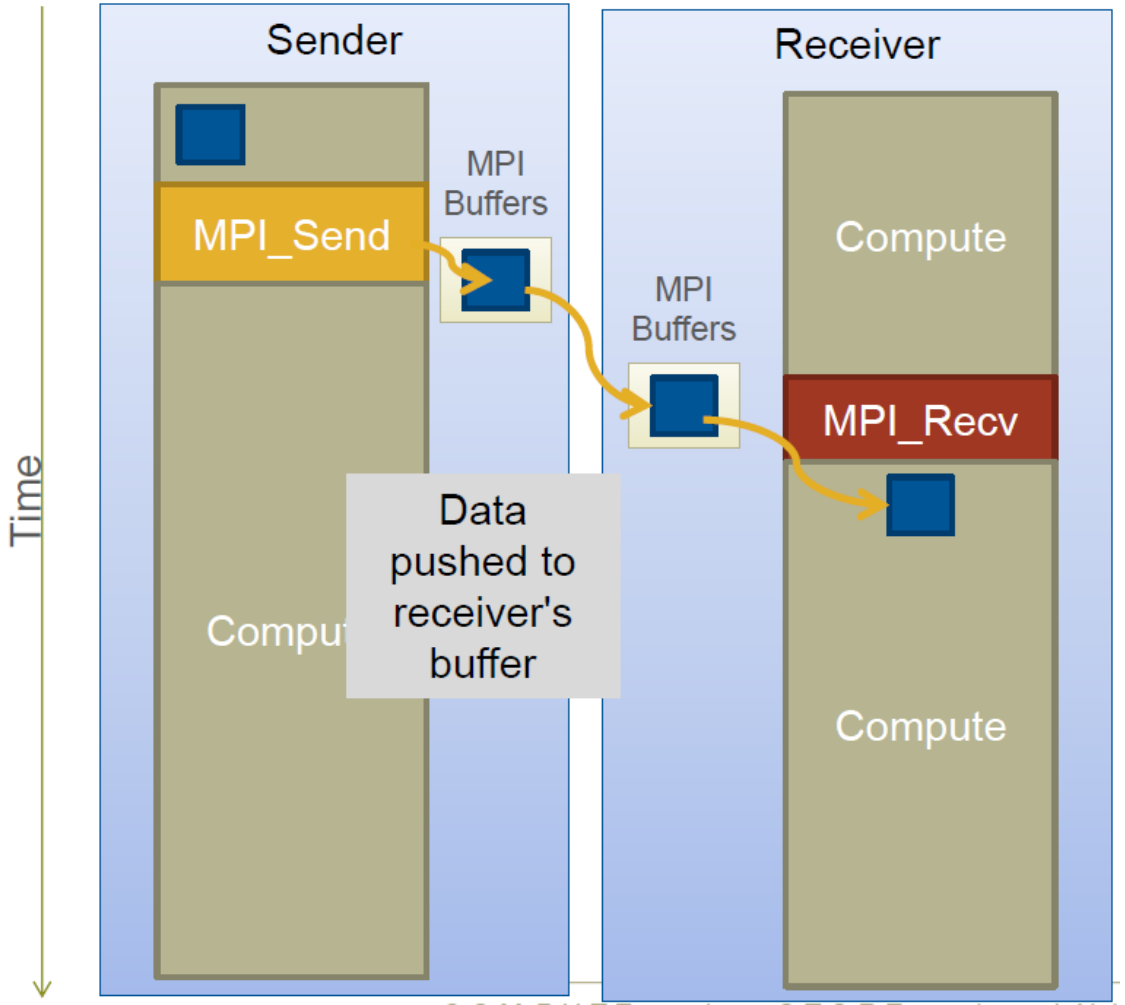
Jan 29, 2021

MPI_Send (Blocking, Standard Mode)

- Does not return until buffer can be reused
- Message buffering can affect this
- Implementation-dependent



Buffering

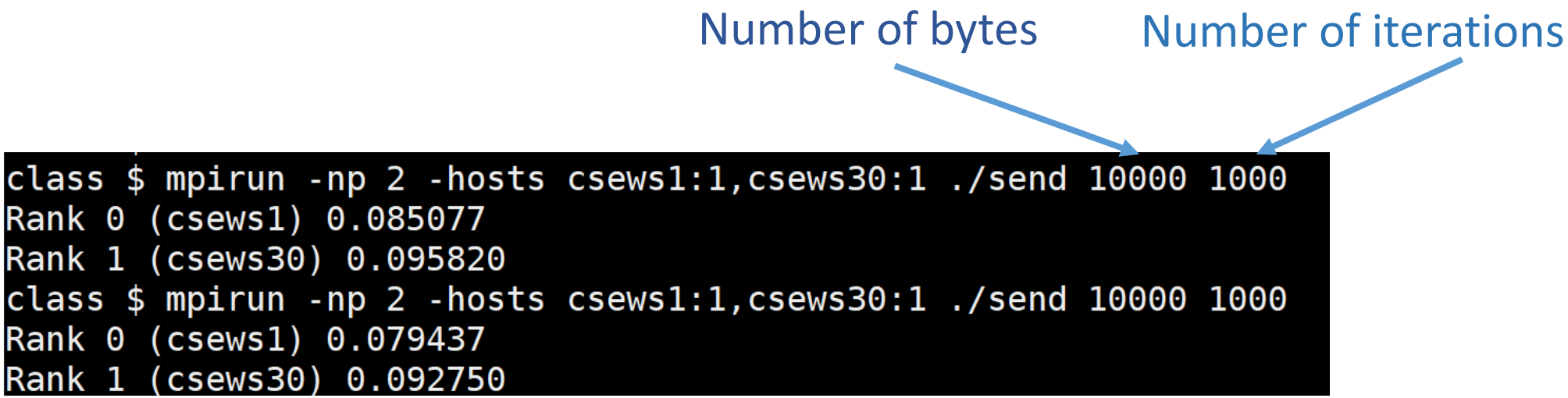


[Source: Cray presentation]

Eager vs. Rendezvous Protocol

- Eager
  - Send completes without acknowledgement from destination
  - MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE (check output of `mpivars`)
  - Small messages, typically 128 KB (at least in MPICH)
- Rendezvous
  - Requires an acknowledgement from a matching receive
  - Large messages

Eager vs. Rendezvous Example



Simple example code: Rank 0 sends 10000 integers to rank 1 1000 times

Matching MPI_Recv Not Posted

```
int main( int argc, char *argv[])
{
  int myrank, size;
  MPI_Status status;
  double sTime, eTime, time;

  MPI_Init(&argc, &argv);

  int count = atoi (argv[1]);
  int buf[count];

  MPI_Comm_rank(MPI_COMM_WORLD, &myrank) ;
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // initialize data
  for (int i=0; i<count; i++)
   buf[i] = myrank+i;

  sTime = MPI_Wtime();
  if (myrank == 0)
   MPI_Send (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD);
  eTime = MPI_Wtime();
  time = eTime - sTime;

  printf ("%lf\n", time);

  MPI_Finalize();
  return 0;

}
```

```
class $ mpirun -np 2 ./norecv 10
0.000007
0.000001
class $ mpirun -np 2 ./norecv 100
0.000015
0.000002
class $ mpirun -np 2 ./norecv 1000
0.000036
0.000001
class $ mpirun -np 2 ./norecv 10000
0.000002
0.000024
class $ mpirun -np 2 ./norecv 100000
0.000001
[mpiexec@csews7] Sending Ctrl-C to processes as requested
[mpiexec@csews7] Press Ctrl-C again to force abort
class $
```

Safety

MPI_Send MPI_Send	MPI_Recv MPI_Recv	Safe
MPI_Send MPI_Recv	MPI_Send MPI_Recv	Unsafe
MPI_Send MPI_Recv	MPI_Recv MPI_Send	Safe
MPI_Recv MPI_Send	MPI_Recv MPI_Send	Unsafe

Other Send Communication Modes

- MPI_Send Standard
  - May complete before matching receive is posted (implementation dependent)
- MPI_Bsend Buffered
  - May complete before matching receive is posted
- MPI_Ssend Synchronous
  - Completes only if a matching receive is posted
- MPI_Rsend Ready
  - Started only if a matching receive is posted

Code

sendmodes.c

```
start_time = MPI_Wtime ();
if (myrank < size-1)
{
  if (option == 1)
   MPI_Send(arr, BUFFER, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
  else if (option == 2)
   MPI_Bsend(arr, BUFFER, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
  else if (option == 3)
   MPI_Ssend(arr, BUFFER, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
}
else
{
  int count, recvarr[size][BUFFER];
  for (int i=0; i<size-1; i++)
    MPI_Recv(recvarr[i], BUFFER, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
}

if (myrank == size-1) printf ("Rank %d: time= %lf\n", myrank, MPI_Wtime () - start_time);
```

MPI_BUFFER_ATTACH/DETACH

MPI_BUFFER_ATTACH (buffer_addr, size)

- Provides to MPI a buffer in the user's memory
- Used by messages sent in buffered mode
- Only one buffer can be attached to a process at a time

MPI_BUFFER_DETACH (buffer_addr, size)

Performance of Send Modes

```
class $ mpirun -np 4 ./sendmodes 10000 1
Rank 3: time= 0.000266
class $ mpirun -np 4 ./sendmodes 10000 2
Rank 3: time= 0.000154
class $ mpirun -np 4 ./sendmodes 10000 3
Rank 3: time= 0.000873
```

MPI_Send
MPI_Bsend
MPI_Ssend

```
class $ mpirun -np 4 ./sendmodes 1000000 1
Rank 3: time= 0.003725
class $ mpirun -np 4 ./sendmodes 1000000 2
Rank 3: time= 0.012123
class $ mpirun -np 4 ./sendmodes 1000000 3
Rank 3: time= 0.004568
```

No handshake (eager) or handshake (rendezvous)
Forced buffering
Forced synchronization

Safety

MPI_Ssend MPI_Ssend	MPI_Recv MPI_Recv	Safe
MPI_Ssend MPI_Recv	MPI_Ssend MPI_Recv	Unsafe/Deadlock
MPI_Ssend MPI_Recv	MPI_Recv MPI_Ssend	Safe

Message Ordering

```
IF (rank.EQ.0) THEN
        MPI_BSEND (buf1, count, MPI_INT, 1, tag1, comm)
        MPI_SSEND (buf2, count, MPI_INT, 1, tag2, comm)
ELSE IF (rank.EQ.1) THEN
        MPI_RECV (buf1, count, MPI_INT, 0, tag2, comm, status)
        MPI_RECV (buf2, count, MPI_INT, 0, tag1, comm, status)
END IF
```

Process 1 receives messages in reverse order

Message Ordering

```
IF (rank.EQ.0) THEN
        MPI_BSEND (buf1, count, MPI_INT, 1, tag, comm)
        MPI_BSEND (buf2, count, MPI_INT, 1, tag, comm)
ELSE IF (rank.EQ.1) THEN
        MPI_RECV (buf1, count, MPI_INT, 0, MPI_ANY_TAG, comm, status)
        MPI_RECV (buf2, count, MPI_INT, 0, tag, comm, status)
END IF
```

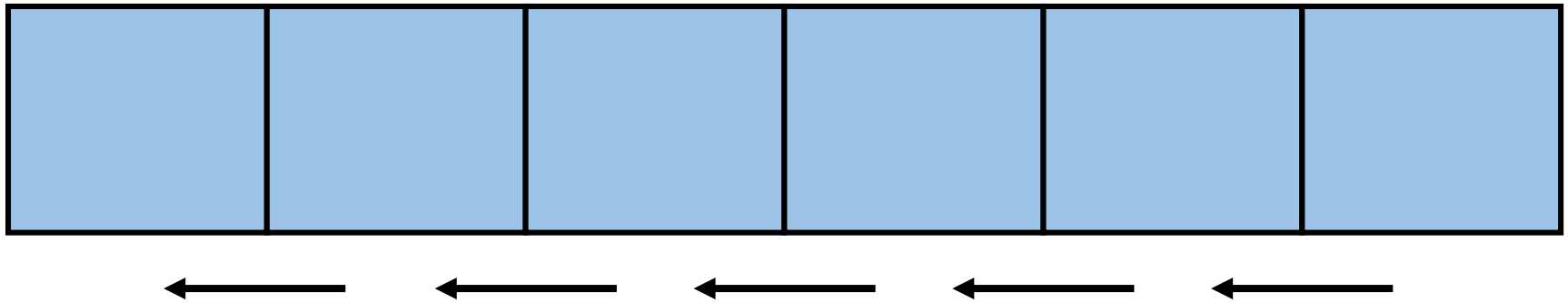
Match must be ensured by the implementation

MPI_Sendrecv

MPI_SENDRECV (sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

- Blocking send and receive
- Same communicator
- Send and receive buffers must be disjoint

MPI_PROC_NULL



P2P Blocking – Performance Bottleneck

- MPI_Send (buf, count, datatype, dest, tag, comm)
- MPI_Recv (buf, count, datatype, source, tag, comm, status)

MPI_Send (1)

Safe but may delay sender

MPI_Recv (0)

P2P Non-blocking

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)
- MPI_Wait (request, status)
- MPI_Waitall (count, request, status)

Code – Non-blocking P2P

```
// send from all ranks to the last rank
start_time = MPI_Wtime ();
if (myrank < size-1)
{
  MPI_Send(arr, BUFSIZE, MPI_INT, size-1, 99, MPI_COMM_WORLD);
}
else
{
  int count, recvarr[size][BUFSIZE];
  for (int i=0; i<size-1; i++)
    MPI_Irecv(recvarr[i], BUFSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request[i]);
  MPI_Waitall (size-1, request, status);
}
time = MPI_Wtime () - start_time;

MPI_Reduce (&time, &max_time, 1, MPI_DOUBLE, MPI_MAX, size-1, MPI_COMM_WORLD);
if (myrank == size-1) printf ("Max time = %lf\n", max_time);
```

Output

```
class $ for i in `seq 1 3` ; do mpirun -np 40 -f hostfile ./p2p 1000000 ; done
Max time = 1.395533
Max time = 1.278514
Max time = 1.352068
class $ for i in `seq 1 3` ; do mpirun -np 40 -f hostfile ./nb_p2p 1000000 ; done
Max time = 1.362506
Max time = 1.244787
Max time = 1.289337
```

```
class $ for i in `seq 1 3` ; do mpirun -np 40 -f hostfile ./p2p 100000000 ; done
Max time = 123.864696
Max time = 123.819141
Max time = 123.850547
class $ for i in `seq 1 3` ; do mpirun -np 40 -f hostfile ./nb_p2p 100000000 ; done
Max time = 122.848520
Max time = 122.804291
Max time = 122.797551
```

Non-blocking Performance

- Standard does not require overlapping communication and computation
- Implementation may use a thread to move data in parallel
- Implementation can delay the initiation of data transfer until “Wait”
- MPI_Test – non-blocking, tests completion, starts progress
- MPIR_CVAR_ASYNC_PROGRESS (MPICH)

Asynchronous Communication Progress

```
class $ mpirun -np 40 -f hostfile ./onetomanyisends 100000000
123.617642
class $ export MPIR_CVAR_ASYNC_PROGRESS=1
class $ mpirun -np 40 -f hostfile ./onetomanyisends 100000000
117.038334
class $ for i in `seq 1 3` ; do mpirun -np 40 -f hostfile ./onetomanyisends 100000000 ; done
116.899904
116.969498
116.891767
class $
```

MPI_Test

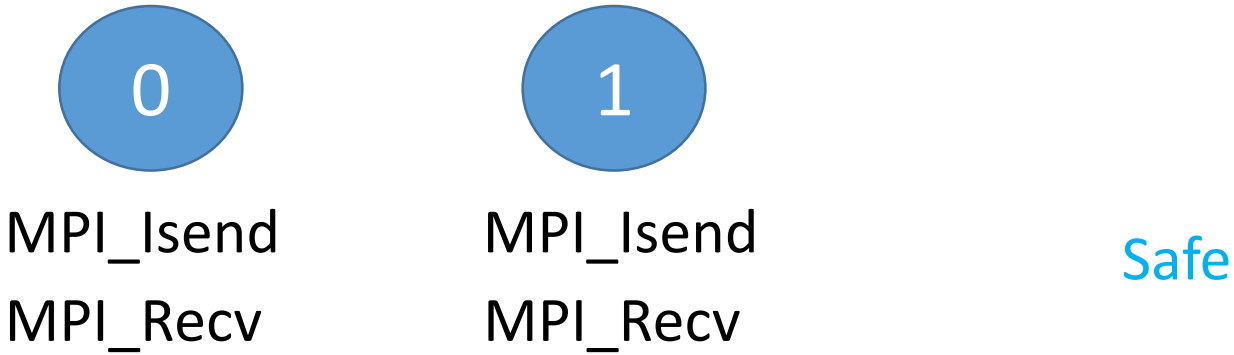
```
sTime = MPI_Wtime();
if (myrank == 0)
 MPI_Isend (buf, count, MPI_INT, 1, 1, MPI_COMM_WORLD, &request);
eTime = MPI_Wtime();
time = eTime - sTime;

if (!myrank) {
   MPI_Test (&request, &flag, &status);
   printf ("%lf %d\n", time, flag);
}
```

```
class $ mpirun -np 2 ./norecv_test 10
0.000011 1
class $ mpirun -np 2 ./norecv_test 100
0.000010 1
class $ mpirun -np 2 ./norecv_test 1000
0.000016 1
class $ mpirun -np 2 ./norecv_test 10000
0.000025 1
class $ mpirun -np 2 ./norecv_test 100000
0.000010 0
```

Non-blocking Point-to-Point Safety

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)
- MPI_Wait (request, status)



Message Ordering

```
IF (rank.EQ.0) THEN
        MPI_ISEND (buf1, count, MPI_INT, 1, 0, comm, req1)
        MPI_ISEND (buf2, count, MPI_INT, 1, 0, comm, req2)
ELSE IF (rank.EQ.1) THEN
        MPI_IRECV (buf1, count, MPI_INT, 0, MPI_ANY_TAG, comm, req1)
        MPI_IRECV (buf2, count, MPI_INT, 0, 0, comm, req2)
END IF
MPI_WAIT (req1, status1)
MPI_WAIT (req2, status2)
```

MPI_Pack

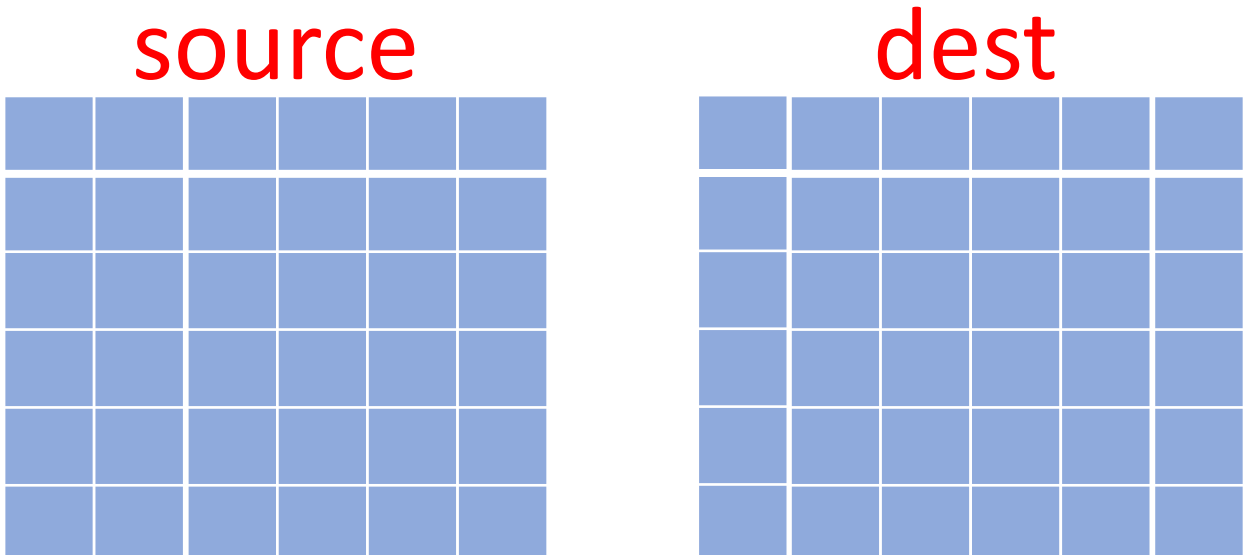
int MPI_Pack (const void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outsize, int *position, MPI_Comm comm)

```
MPI_Pack (&num1, 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Pack (&num2, 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);



MPI_Recv (recvbuf, 2, MPI_INT, source, 0, MPI_COMM_WORLD)
```

MPI_PACK Example




```
for (int r=0; r<6; r++)
  MPI_Pack (&array[r][5], 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);

MPI_Recv (recvColumn, count, MPI_INT, source, 0, MPI_COMM_WORLD, &status);
```

MPI_Unpack

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

```
for (int r=0; r<6; r++)
   MPI_Pack (&array[r][5], 1, MPI_INT, buffer, 1000, &position, MPI_COMM_WORLD);
MPI_Send (buffer, position, MPI_PACKED, dest, 0, MPI_COMM_WORLD);

MPI_Recv (recvBuf, count, MPI_PACKED, source, 0, MPI_COMM_WORLD, &status);
for (int r=0; r<6; r++)
   MPI_Unpack (recvBuf, 1000, &position, &recvArr[r][0], 1, MPI_INT, MPI_COMM_WORLD);
```