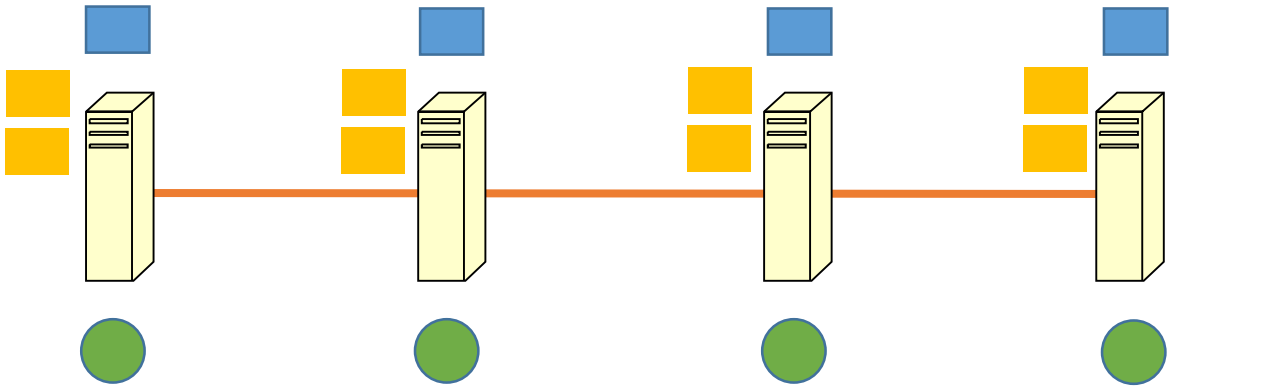
Collectives

Feb 12, 2021

Blocking Collectives

- MPI_Barrier
- MPI_Bcast
- MPI_Gather
- MPI_Scatter
- MPI_Allgather
- MPI_Alltoall
- MPI_Scan
- MPI_Reduce
- MPI_Allreduce

Parallel Sum



Core

Process

Memory

for i = 0 to N/P sum += a[i] * a[i]	for i = N/P to 2N/P sum += a[i] * a[i]	for i = 2N/P to 3N/P sum += a[i] * a[i]	for i = 3N/P to N sum += a[i] * a[i]

Rank = 0 Rank = 1 Rank = 2 Rank = 3

Parallel Sum

```
// local computation at every process
for i = N/P * rank ; i < N/P * (rank+1) ; i++
    localsum += a[i] * a[i]

// collect localsum, add up at one of the ranks
MPI_Reduce (&localsum, … , MPI_SUM, …)
```

Using Send/Recv

```
int recvarr[numtasks];

// receive partial sums at rank 0
stime = MPI_Wtime();
if (rank)
{
  MPI_Send(&sum, 1, MPI_INT, 0, rank, MPI_COMM_WORLD);
}
else
{
  for (int r=1; r<numtasks; r++)
  {
    MPI_Recv(&recvarr[r], 1, MPI_INT, r, r, MPI_COMM_WORLD, &status);
  }
}
etime = MPI_Wtime();
cotime = etime - stime;

stime = MPI_Wtime();
// HOMEWORK
// Add the partial sums
for (int r=1; r<numtasks; r++)
  sum += recvarr[r];
etime = MPI_Wtime();
ctime += etime - stime;

if (!rank)
printf ("%d %lf %lf\n", sum, ctime, cotime);
```

Using Reduce

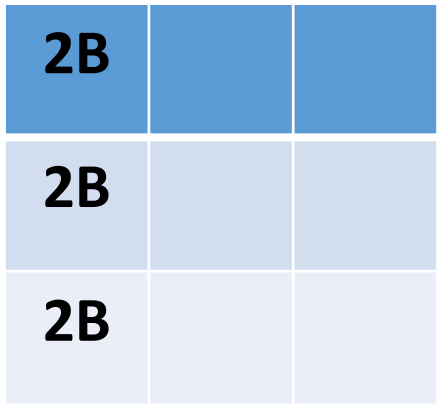
```
// local sum computation
sum=0.0;
stime = MPI_Wtime();
for (i=sidx; i<sidx+N/numtasks ; i++)
    sum += a[i] * a[i];
etime = MPI_Wtime();
ctime = etime - stime;

int globalsum;

stime = MPI_Wtime();
MPI_Reduce (&sum, &globalsum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
etime = MPI_Wtime();
cotime = etime - stime;


if (!rank)
printf ("%d %lf %lf\n", globalsum, ctime, cotime);
```

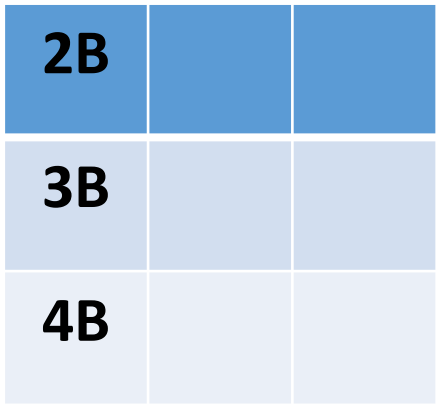
Timing

```
class $ for i in `seq 1 5`; do mpirun -np 3 ./parsum 6000 ; done
10000 0.000007 0.000301
10000 0.000006 0.000309
10000 0.000010 0.000015
10000 0.000006 0.000281
10000 0.000010 0.000041
class $ for i in `seq 1 5`; do mpirun -np 3 ./parsumreduce 6000 ; done
10000 0.000011 0.000172
10000 0.000010 0.000034
10000 0.000010 0.000033
10000 0.000010 0.000025
10000 0.000010 0.000028
class $ for i in `seq 1 5`; do mpirun -np 30 -hosts csews3:10,csews5:10,csews6:10 ./parsum 6000000 ; done
1711000000 0.001006 0.025789
1711000000 0.001034 0.010492
1711000000 0.001007 0.023931
1711000000 0.001029 0.038722
1711000000 0.001028 0.024971
class $ for i in `seq 1 5`; do mpirun -np 30 -hosts csews3:10,csews5:10,csews6:10 ./parsumreduce 6000000 ; done
1711000000 0.001003 0.011949
1711000000 0.002717 0.001540
1711000000 0.001092 0.009885
1711000000 0.003251 0.003160
1711000000 0.001091 0.012467
```

Vector Variants

- Communicate unequal amount of data to/from each process involved in the collective function call



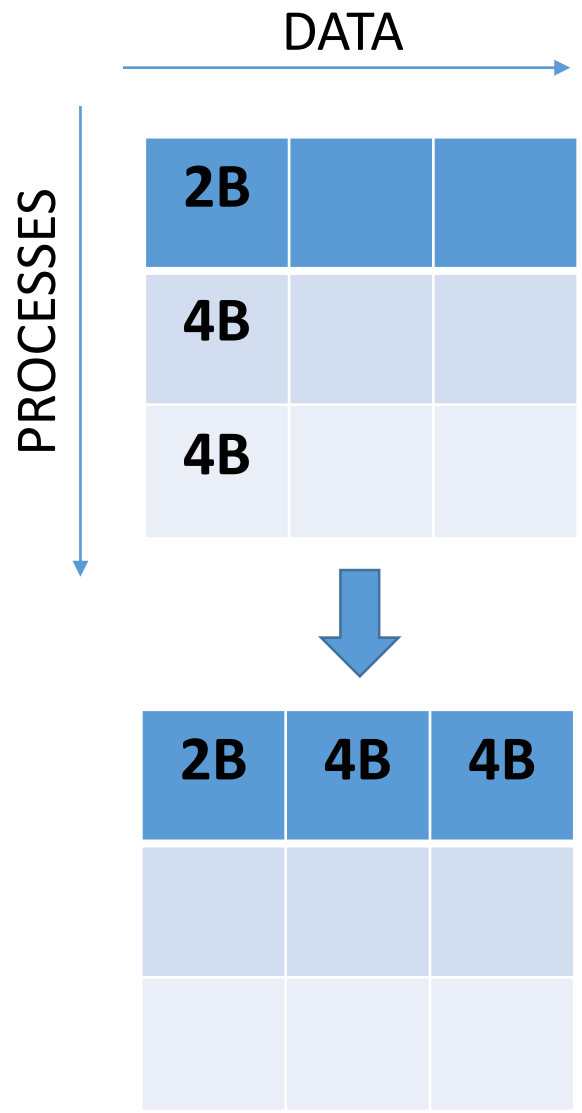


Gatherv

- Root gathers different amounts of data from the other processes
- int MPI_Gatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data

MPI_Recv (recvbuf+displs[i], recvcounts[i], recvtype, i, i, comm, &status) at root for i^{th} process

MPI_Send at non-root



MPI_Gatherv Code

```
int message[arrSize];
int countArray[numtasks], displArray[numtasks];

int displ = 0;                      // note that root process is 0 here

// this information is needed by the root
if (!rank)
 for (i = 0; i < numtasks; i++) {
  countArray[i] = arrSize*(i+1);         // depends on the counts, root may need to get it from the processes
  displArray[i] = displ;
  displ += countArray[i];
  printf ("%d %d %d\n", i, countArray[i], displArray[i]);
 }

if (!rank)
  printf ("\n");

// every process initializes their local array
srand(time(NULL));
for (i = 0; i < arrSize; i++) {
  message[i] = i; // (double)rand() / (double)RAND_MAX;
}

int recvMessage[displ];    // significant at the root process

// receive different counts of elements from different processes
MPI_Gatherv (message, arrSize, MPI_INT, recvMessage, countArray, displArray, MPI_INT, 0, MPI_COMM_WORLD);

if (!rank)
 for (i = 0; i < displ; i++) {
   printf ("%d %d\n", i, recvMessage[i]);
 }
```

Demo

```
class $ mpirun -np 2 ./gatherv 2
0 2 0
1 4 2

0 0
1 1
2 0
3 1
4 2
5 3
class $ mpirun -np 3 ./gatherv 2
0 2 0
1 4 2
2 6 6

0 0
1 1
2 0
3 1
4 2
5 3
6 0
7 1
8 2
9 3
10 4
11 5
class $
```

Bug

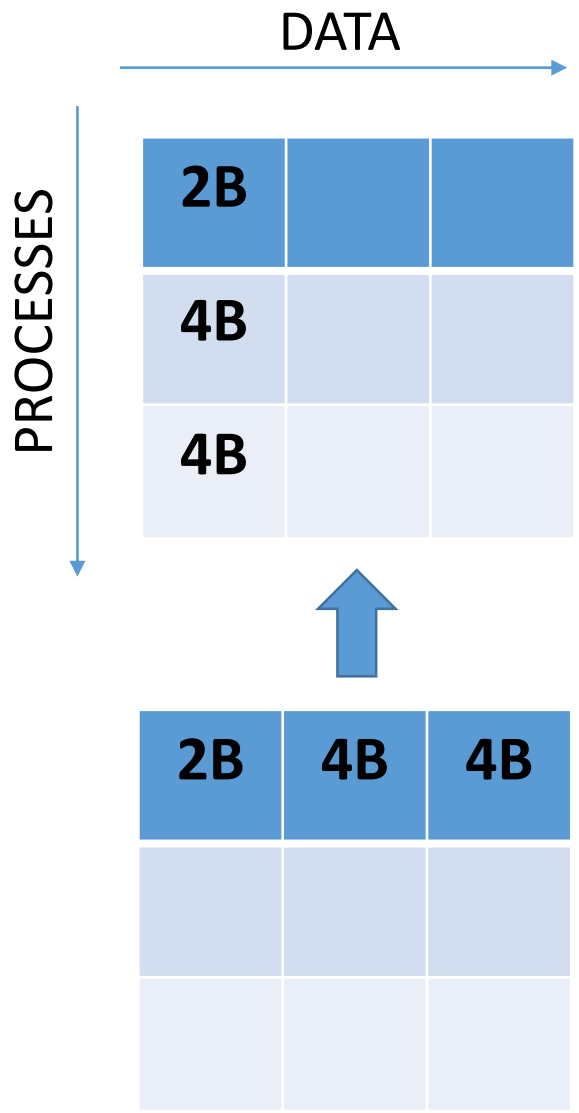
```
Fatal error in PMPI_Gatherv: Message truncated, error stack:
PMPI_Gatherv(435).....................: MPI_Gatherv failed(sbuf=0x7ffd379c4250, scount=2, MPI_INT, rbuf=0x7ffd379c4210, rcnts=0x7ffd379c4240,
 displs=0x7ffd379c4230, MPI_INT, root=0, MPI_COMM_WORLD) failed
MPIR_Gatherv_impl(235)................:
MPIR_Gatherv(151).....................:
MPIDI_CH3_PktHandler_EagerShortSend(363): Message from rank 1 and tag 4 truncated; 16 bytes received but buffer size is 8
class $ vi gatherv.c
class $ !mpicc
mpicc -o gatherv gatherv.c
class $ !mpirun
mpirun -np 2 ./gatherv 2
0 2 0
1 4 2
class $
```

Example

```
class $ time mpirun -np 3 ./gatherv 2

real    0m0.007s
user    0m0.004s
sys     0m0.010s
class $ time mpirun -np 30 -hosts csews4:10,csews2:10,csews3:10 ./gatherv 200

real    0m0.695s
user    0m0.026s
sys     0m0.006s
class $ time mpirun -np 30 -hosts csews4:10,csews2:10,csews3:10 ./gatherv 20000

real    0m1.857s
user    0m0.033s
sys     0m0.002s
class $ time mpirun -np 60 -hosts csews4:10,csews2:10,csews3:10,csews5:10,csews6:10,csews7:10 ./gatherv 20000

real    0m2.965s
user    0m12.571s
sys     0m1.559s
class $ time mpirun -np 60 -hosts csews4:10,csews2:10,csews3:10,csews5:10,csews6:10,csews7:10 ./gatherv 800000

real    1m5.621s
user    7m2.519s
sys     1m2.687s
class $
```

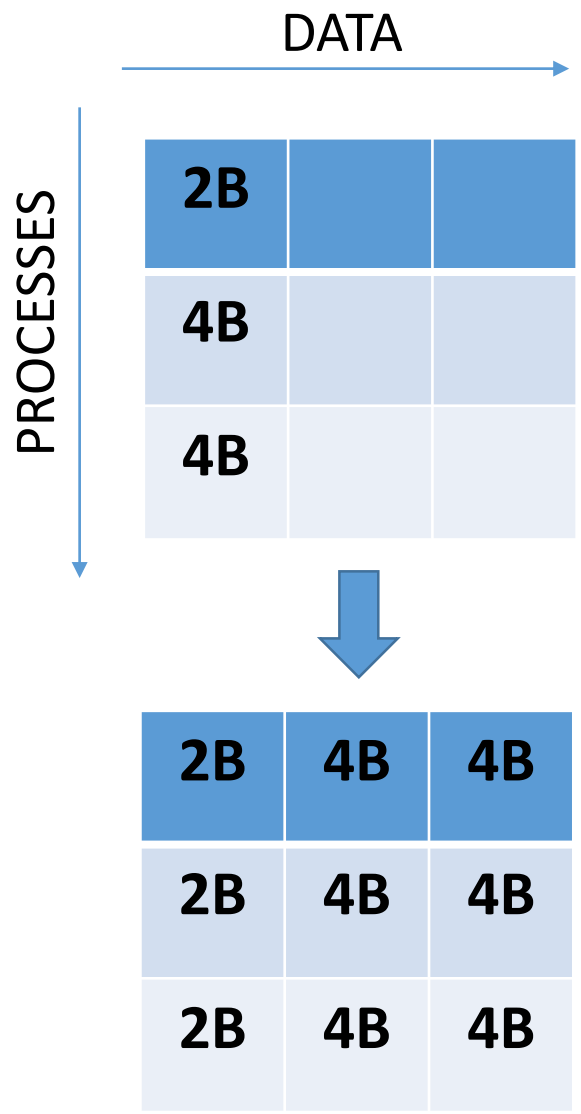
MPI_Scatterv

- Root scatters different amounts of data to the other processes
- int MPI_Scatterv (const void *sendbuf, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- sendcounts – Number of elements to be sent to each process
- displs – Displacement (relative to sendbuf) at which the data to be sent resides



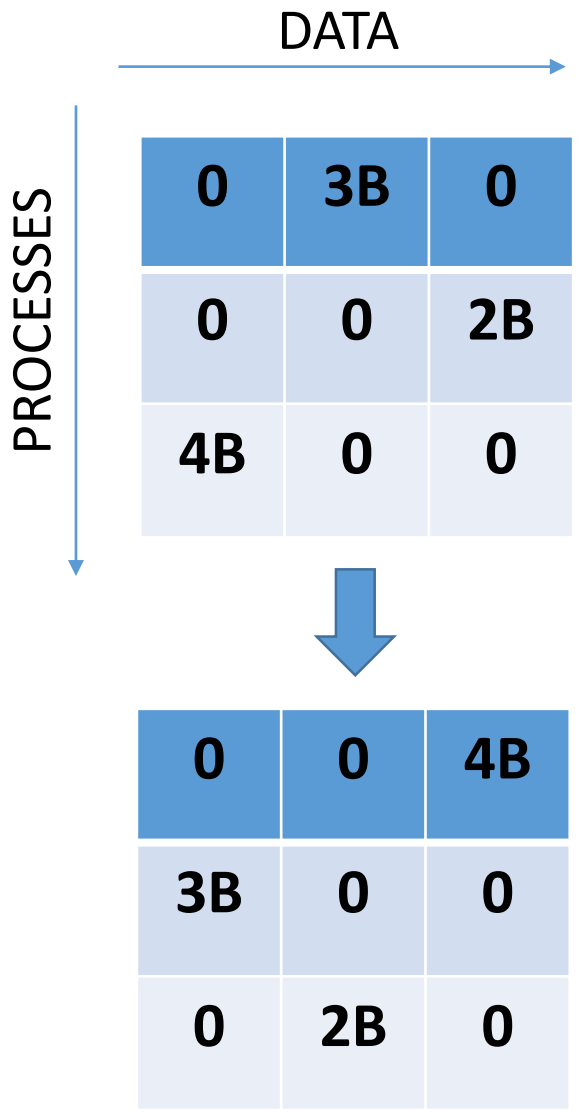
Allgatherv

- All processes gather values of different lengths from all processes
- int MPI_Allgatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm)
- recvcounts – Number of elements to be received from each process
- displs – Displacement at which to place received data

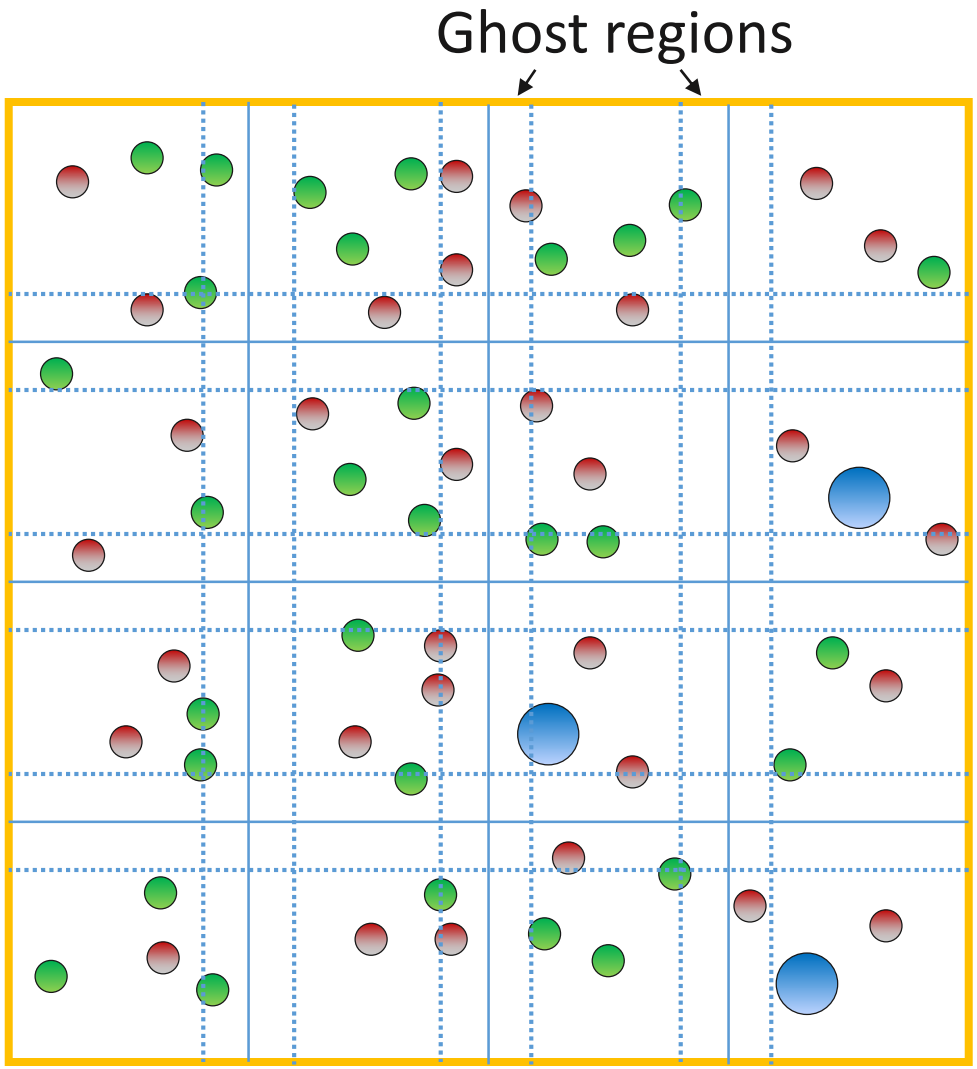


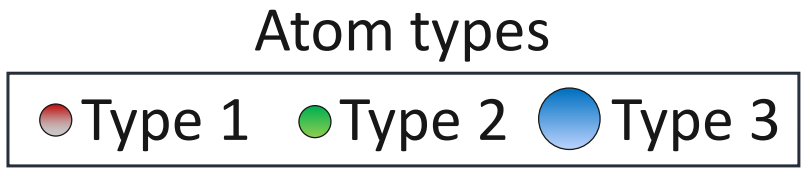
Alltoallv

- Every process sends data of different lengths to other processes
- int MPI_Alltoallv (sendbuf, sendcount, sdispls, sendtype, recvbuf, recvcount, rdispls, recvtype, comm)
- Output parameter – recvbuf
- It’s not necessary to receive some data from all processes, i.e. some entries of count and displs may be 0



Practical Use





Non-blocking Collectives

- Introduced in MPI-3
- Benefit of non-blocking point-to-point
- Overlap communication and computation
- Reduce synchronization
- Improve performance for overlapping communicators
- How do we ensure completion?
  - MPI_Wait (request, status)

Non-blocking Collectives

- MPI_Ibcast (buffer, count, datatype, root, comm, request)
- MPI_Igather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, request)
- MPI_Igatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, request)
- MPI_Ialltoall (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, request)
- …

MPI_Ibcast

```
int main( int argc, char *argv[])
{
  int count = atoi(argv[1]);
  int myrank, buf[count];
  double buf_can_modify[count];
  MPI_Status status;
  MPI_Request request;

  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

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

  // has to be called by all processes
  MPI_Ibcast(buf, count, MPI_INT, 1, MPI_COMM_WORLD, &request);

  // some random expensive computations
  for (int j=0; j<count; j++)
    for (int i=0; i<count; i++)
      buf_can_modify[i] = i*12022021/(myrank+1)*pow(2,i);

  MPI_Wait (&request, &status);

  MPI_Finalize();
  return 0;

}
```

Execution Times

```
class $ for i in `seq 1 5` ; do time mpirun -np 40 -f ../hostfile ./ibcast 100000000 ; done

real    0m14.848s
user    2m27.926s
sys     0m5.847s

real    0m14.547s
user    2m25.309s
sys     0m6.080s

real    0m13.946s
user    2m19.674s
sys     0m5.541s

real    0m14.135s
user    2m20.602s
sys     0m5.846s

real    0m13.508s
user    2m17.507s
sys     0m5.713s
```

```
real    0m16.064s
user    1m50.779s
sys     0m5.527s

real    0m20.102s
user    2m26.007s
sys     0m6.418s

real    0m20.122s
user    2m25.546s
sys     0m6.555s

real    0m20.256s
user    2m26.897s
sys     0m6.586s

real    0m18.734s
user    2m13.618s
sys     0m7.214s
```

MPICH_ASYNC_PROGRESS=1

MPI Examples

```
If (myrank == 0) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1);
  MPI_Wait (&req1, MPI_STATUS_IGNORE);
  MPI_Send (buf, count, MPI_INT, 1, tag, comm);
}
If (myrank == 1) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1);
  MPI_Recv (buf, count, MPI_INT, 0, tag, comm, MPI_STATUS_IGNORE);
  MPI_Wait (&req1, MPI_STATUS_IGNORE);
}
```

Will it run for n=2?

Valid code

MPI Examples

```
If (myrank == 0) {
  MPI_Ibcast (&buf1, 1, MPI_INT, 0, comm, req1[0]);
  MPI_Ibcast (&buf2, 2, MPI_INT, 0, comm, req1[1]);
}
If (myrank == 1) {
  MPI_Ibcast (&buf2, 1, MPI_INT, 0, comm, req1[0]);
  MPI_Ibcast (&buf1, 2, MPI_INT, 0, comm, req1[1]);
}
MPI_Waitall(2, req1, MPI_STATUSES_IGNORE);
```

Correct for n=2?

Valid code