MPI - v Operations

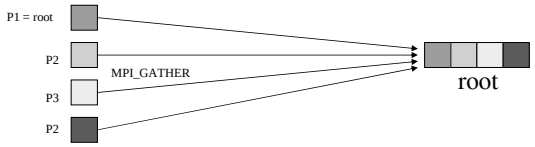
Based on notes by Dr. David Cronk
Innovative Computing Lab
University of Tennessee

Collective Communication: Gather

- A Gather operation has data from all processes collected, or "gathered", at a central process, referred to as the "root"
- Even the root process contributes data
- The root process can be any process, it does not have to have any particular rank
- Every process must pass the same value for the root argument
- Each process must send the same *amount* of data

Collective Communication: Gather

- MPI_GATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)
  - Receive arguments are only meaningful at the root
  - recvcount is the number of elements received from each process
  - Data received at *root* in rank order
  - Root can use MPI_IN_PLACE for sendbuf:
    - data is assumed to be in the correct place in the recvbuf



MPI_Gather

```
int tmp[20];
int res[320];
```

```
for (i = 0; i < 20; i++) {
  do some computation
  tmp[i] = some value;
}
MPI_Gather (tmp, 20, MPI_INT, res,
  20, MPI_INT, 0, MPI_COMM_WORLD);
if (myrank == 0)
  write out results
```

WORKS

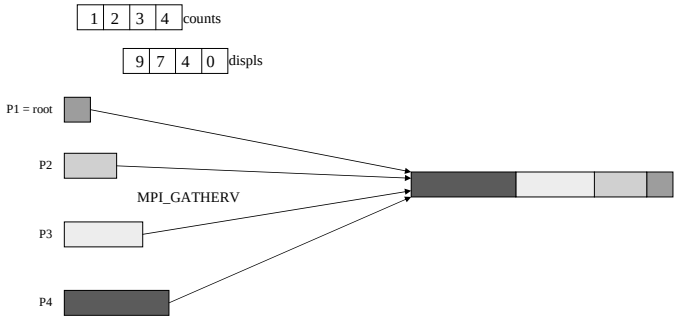
```
for (i = 0; i < 20; i++) {
  do some computation
  if (myrank == 0)
    res[i] = some value
  else tmp[i] = some value
}
if (myrank == 0)
  MPI_Gather (MPI_IN_PLACE,
    20, MPI_INT, res, 20, MPI_INT,
    0, MPI_COMM_WORLD);
  write out results
else
  MPI_Gather (tmp, 20, MPI_INT,
    tmp, 320, MPI_REAL, 0
    MPI_COMM_WORLD);
```

A OK

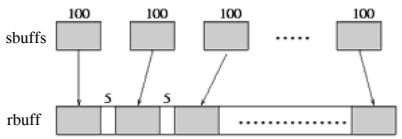
Collective Communication: Gatherv

- MPI_GATHERV (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm,ierr)
  - Vector variant of MPI_GATHER
  - Allows a varying amount of data from each proc
  - allows root to specify where data from each proc goes
  - No portion of the receive buffer may be written more than once
  - The amount of data specified to be received at the root must match amount of data sent by non-roots
  - Displacements are in terms of recvtype
  - MPI_IN_PLACE may be used by root.

Collective Communication: Gatherv (cont)



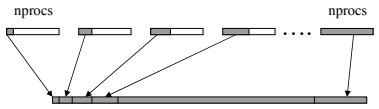
Collective Communication: Gatherv (cont)




```
stride = 105;
root = 0;
for (i = 0; i  < nprocs; i++) {
  displs[i] = i*stride
  counts[i] = 100;
}

MPI_Gatherv (sbuff, 100, MPI_INT,
    rbuff, counts, displs, MPI_INT,
    root, MPI_COMM_WORLD);
```

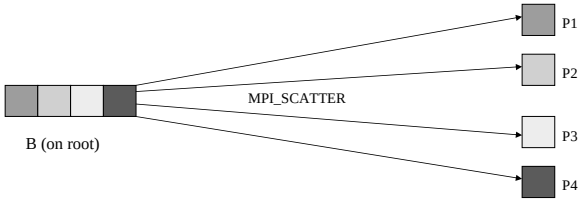

Collective Communication: Gatherv (cont)



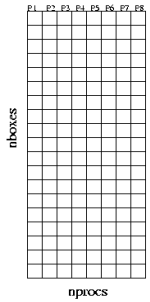

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
scount = myrank+1
displs(1) = 0
rcounts(1) = 1
DO I=2,nprocs
  displs(I) = displs(I-1) + I – 1
  rcounts(I) = I
ENDDO
CALL MPI_GATHERV(sbuff, scount, MPI_INT, rbuff, rcounts,
     displs, MPI_INT, root, MPI_COMM_WORL, ierr)
```

Collective Communication: Scatter

- MPI_SCATTER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, ierr)
  - Opposite of MPI_GATHER
  - Send arguments only meaningful at root
  - Root can use MPI_IN_PLACE for recvbuf



MPI_SCATTER

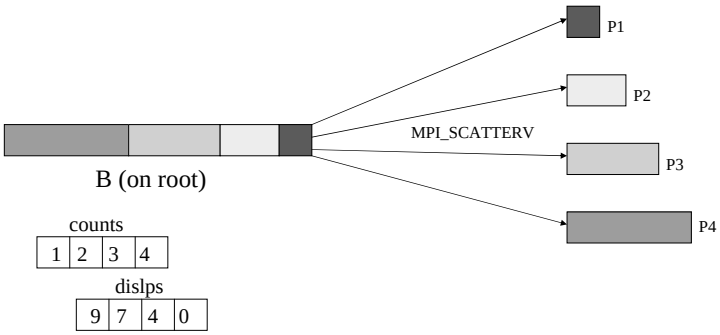



```
IF (MYPE .EQ. ROOT) THEN
  OPEN (25, FILE='filename')
  READ (25, *) nprocs, nboxes
  READ (25, *) mat(i,j) (i=1,nboxes)(j=1,nprocs)
  CLOSE (25)
ENDIF
CALL MPI_BCAST (nboxes, 1, MPI_INTEGER,
&    ROOT, MPI_COMM_WORLD, ierr)
CALL MPI_SCATTER (mat, nboxes, MPI_INT,
&    lboxes, nboxes, MPI_INT, ROOT,
&    MPI_COMM_WORLD, ierr)
```

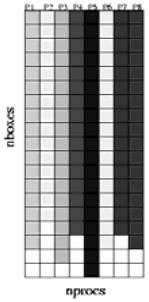

Collective Communication: Scatterv

- MPI_SCATTERV (sendbuf, scounts, displs, sendtype, recvbuf, recvcount, recvtype, ierr)
  - Opposite of MPI_GATHERV
  - Send arguments only meaningful at root
  - Root can use MPI_IN_PLACE for recvbuf
  - No location of the sendbuf can be read more than once

Collective Communication: Scatterv (cont)



MPI_SCATTERV




```
C  mnb = max number of boxes

IF (MYPE .EQ. ROOT) THEN
  OPEN (25, FILE='filename')
  READ (25, *) nprocs
  READ (25, *) (nboxes(I), I=1,nprocs)
  READ (25, *) mat(I,J) (I=1,nboxes(I))(J=1,nprocs)
  CLOSE (25)
  DO I = 1,nprocs
    displs(I) = (I-1)*mnb
  ENDDO
ENDIF
CALL MPI_SCATTER (nboxes, 1, MPI_INT,
   nb, 1, MPI_INT, ROOT, MPI_COMM_WORLD, ierr)
CALL MPI_SCATTERV (mat, nboxes, displs, MPI_INT,
   lboxes, nb, MPI_INT, ROOT,  MPI_COMM_WORLD, ierr)
```

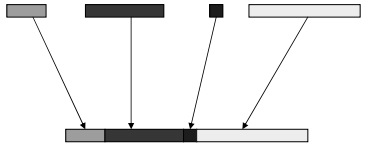

Collective Communication: Allgather

- MPI_ALLGATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)
  - Same as MPI_GATHER, except all processors get the result
  - MPI_IN_PLACE may be used for sendbuf of all processors
  - Equivalent to a gather followed by a bcast

Collective Communication: Allgatherv

- MPI_ALLGATHERV (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm, ierr)
  - Same as MPI_GATHERV, except all processes get the result
  - MPI_IN_PLACE may be used for sendbuf of all processors
  - Similar to a gatherv followed by a bcast
    - If there are "holes" in the receive buffer, bcast would overwrite the "holes"
    - displs need not be the same on each PE

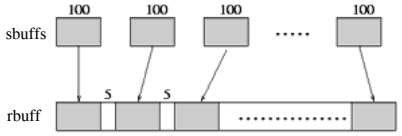
Allgatherv



```
int mycount; /* inited to # local ints */
int counts[4]; /* inited to proper values */
int displs[4];

displs[0] = 0;
for (I = 1; I < 4; I++)
  displs[I] = displs[I-1] + counts[I-1];
MPI_Allgatherv(sbuff, mycount, MPI_INT,
  rbuff, counts, displs, MPI_INT,
  MPI_COMM_WORLD);
```

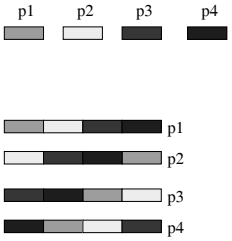
Allgatherv




```
stride = 105;
root = 0;
for (i = 0; i  < nprocs; i++) {
  displs[i] = i*stride
  counts[I] = 100;
}

MPI_Allgatherv (sbuff, 100, MPI_INT,
    rbuff, counts, displs, MPI_INT,
    MPI_COMM_WORLD);
```


Allgatherv

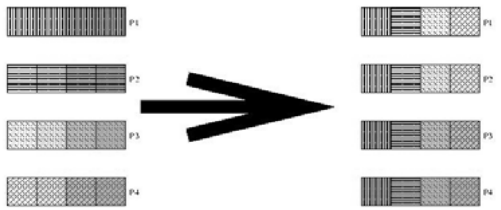



```
MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
MPI_Comm_size (MPI_COMM_WORLD, &nprocs);

for (i=0; i<nprocs; i++) {
  counts[i] = num_elements;
  x = (myrank+i) % nprocs;
  dislps[x] = i;
}
MPI_Allgatherv (sbuff, num_elements, MPI_INT, rbuff,
  counts, dislps, MPI_INT, MPI_COMM_WORLD);
```


Collective Communication: Alltoall (scatter/gather)

- MPI_ALLTOALL (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)



Collective Communication: Alltoallv

- MPI_ALLTOALLV (sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm, ierr)
  - Same as MPI_ALLTOALL, but the vector variant
    - Can specify how many blocks to send to each processor, location of blocks to send, how many blocks to receive from each processor, and where to place the received blocks
    - No location in the sendbuf can be read more than once and no location in the recvbuf can be written to more than once

Collective Communication: Alltoallw

- MPI_ALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounts, rdispls, recvtypes, comm, ierr)
  - Same as MPI_ALLTOALLV, except different datatypes can be specified for data scattered as well as data gathered
    - Can specify how many blocks to send to each processor, location and type of blocks to send, how many blocks to receive from each processor, the type of the blocks received, and where to place the received blocks
    - Displacements are now in terms of bytes rather that types
    - No location in the sendbuf can be read more than once and no location in the recvbuf can be written to more than once

Example

```
      subroutine System_Change_init ( dq, ierror)
!-------------------------------------------------------------------------------
!     Subroutine for data exchange on all six boundaries
!
!     This routine initiates the operations, has to be finished by according
!     Wait/Waitall
!-------------------------------------------------------------------------------
!
      USE globale_daten
      USE comm

      implicit none

      include 'mpif.h'

      double precision, dimension (0:n1+1,0:n2+1,0:n3+1,nc) :: dq

!     local variables

      integer :: handnum, info, position
      integer :: j, k, n, ni
      integer :: ierror
      integer :: size2, size4, size6
!     /* Fortran90 */

      integer, dimension (MPI_STATUS_SIZE,6) :: sendstatusfeld
      integer, dimension (MPI_STATUS_SIZE) :: status

      double precision, allocatable, dimension (:) :: global_dq

      size2 = (n2+2)*(n3+2)*nc * SIZE_OF_REAL.x
      size4 = (n1+2)*(n3+2)*nc * SIZE_OF_REAL.x
      size6 = (n1+2)*(n2+2)*nc * SIZE_OF_REAL.x

      if (.not. rand_ab) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_io),
     &          recvtype(tid_io), tid_io, 10001,
     &          MPI_COMM_WORLD, recvhandle(1), info)
      else
        recvhandle(1) = MPI_REQUEST_NULL
      endif

      if (.not. rand_sing) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_iu),
     &           recvtype(tid_iu), tid_iu, 10002,
     &           MPI_COMM_WORLD, recvhandle(2), info)
      else
        recvhandle(2) = MPI_REQUEST_NULL
      endif
```


Example (cont)

```
      if (.not. rand_zu) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_jo),
     &          recvtype(tid_jo), tid_jo, 10003,
     &          MPI_COMM_WORLD, recvhandle(3), info)
      else
        recvhandle(3) = MPI_REQUEST_NULL
      endif

      if (.not. rand_festk) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_ju),
     &          recvtype(tid_ju), tid_ju, 10004,
     &          MPI_COMM_WORLD, recvhandle(4), info)
      else
        recvhandle(4) = MPI_REQUEST_NULL
      endif

      if (.not. rand_symo) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_ko),
     &          recvtype(tid_ko), tid_ko, 10005,
     &          MPI_COMM_WORLD, recvhandle(5), info)
      else
        recvhandle(5) = MPI_REQUEST_NULL
      endif
```

```
      if (.not. rand_symu) then
          call MPI_IRECV(dq(1,1,1,1), recvcount(tid_ku),
     &          recvtype(tid_ku), tid_ku, 10006,
     &          MPI_COMM_WORLD, recvhandle(6), info)
      else
        recvhandle(6) = MPI_REQUEST_NULL
      endif

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      if (.not. rand_sing) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_iu),
     &          sendtype(tid_iu), tid_iu, 10001,
     &          MPI_COMM_WORLD, sendhandle(1), info)
      else
        sendhandle(1) = MPI_REQUEST_NULL
      endif

      if (.not. rand_ab) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_io),
     &          sendtype(tid_io), tid_io, 10002,
     &          MPI_COMM_WORLD, sendhandle(2), info)
      else
        sendhandle(2) = MPI_REQUEST_NULL
      endif
```


Example (cont)

```
      if (.not. rand_festk) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_ju),
     &          sendtype(tid_ju), tid_ju, 10003,
     &          MPI_COMM_WORLD, sendhandle(3), info)
      else
        sendhandle(3) = MPI_REQUEST_NULL
      endif

      if (.not. rand_zu) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_jo),
     &          sendtype(tid_jo), tid_jo, 10004,
     &          MPI_COMM_WORLD, sendhandle(4), info)
      else
        sendhandle(4) = MPI_REQUEST_NULL
      endif

      if (.not. rand_symu) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_ku),
     &          sendtype(tid_ku), tid_ku, 10005,
     &          MPI_COMM_WORLD, sendhandle(5), info)
      else
        sendhandle(5) = MPI_REQUEST_NULL
      endif
```

```
      if (.not. rand_symo) then
          call MPI_ISEND(dq(1,1,1,1), sendcount(tid_ko),
     &          sendtype(tid_ko), tid_ko, 10006,
     &          MPI_COMM_WORLD, sendhandle(6), info)
      else
        sendhandle(6) = MPI_REQUEST_NULL
      endif

!...  Waitall for the Isends/we have to force to finish them..

      call MPI_WAITALL(6, sendhandle, sendstatusfeld, info)

      ierror = 0

      return
      end
```

Example (Alltoallw)

```
subroutine System_Change ( dq, ierror)
!------------------------------------------------------------------------------
!
!    Subroutine for data exchange on all six boundaries
!
!    uses the MPI-2 function MPI_Alltoallw
!------------------------------------------------------------------------------
!
    USE globale_daten

    implicit none

    include 'mpif.h'

    double precision, dimension (0:n1+1,0:n2+1,0:n3+1,nc) :: dq
    integer :: ierror

    call MPI_Alltoallw ( dq(1,1,1,1), sendcount, senddisps, sendtype,  &
                         dq(1,1,1,1), recvcount, recvdisps, recvtype,  &
                         MPI_COMM_WORLD, ierror )


    end subroutine System_Change
```