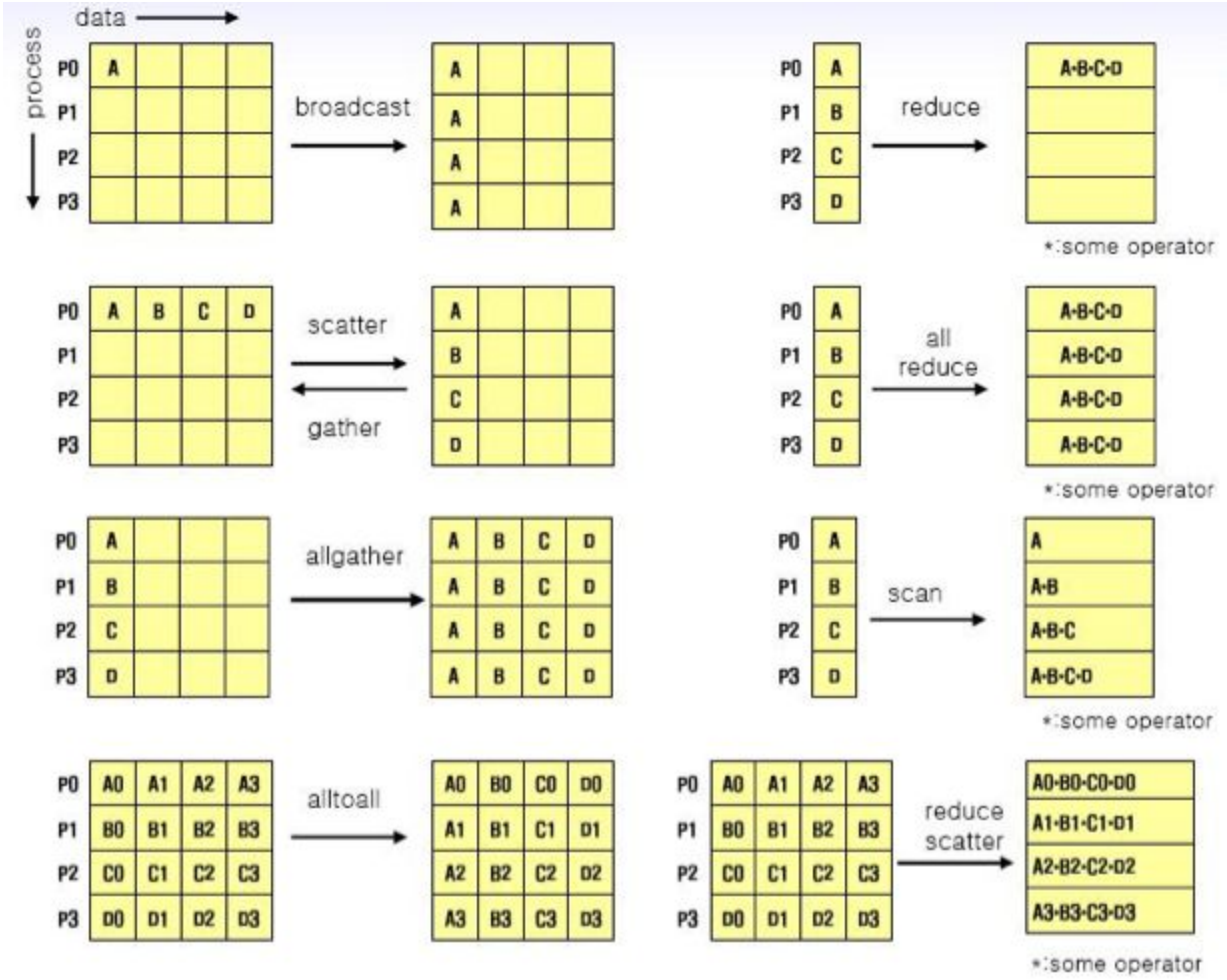
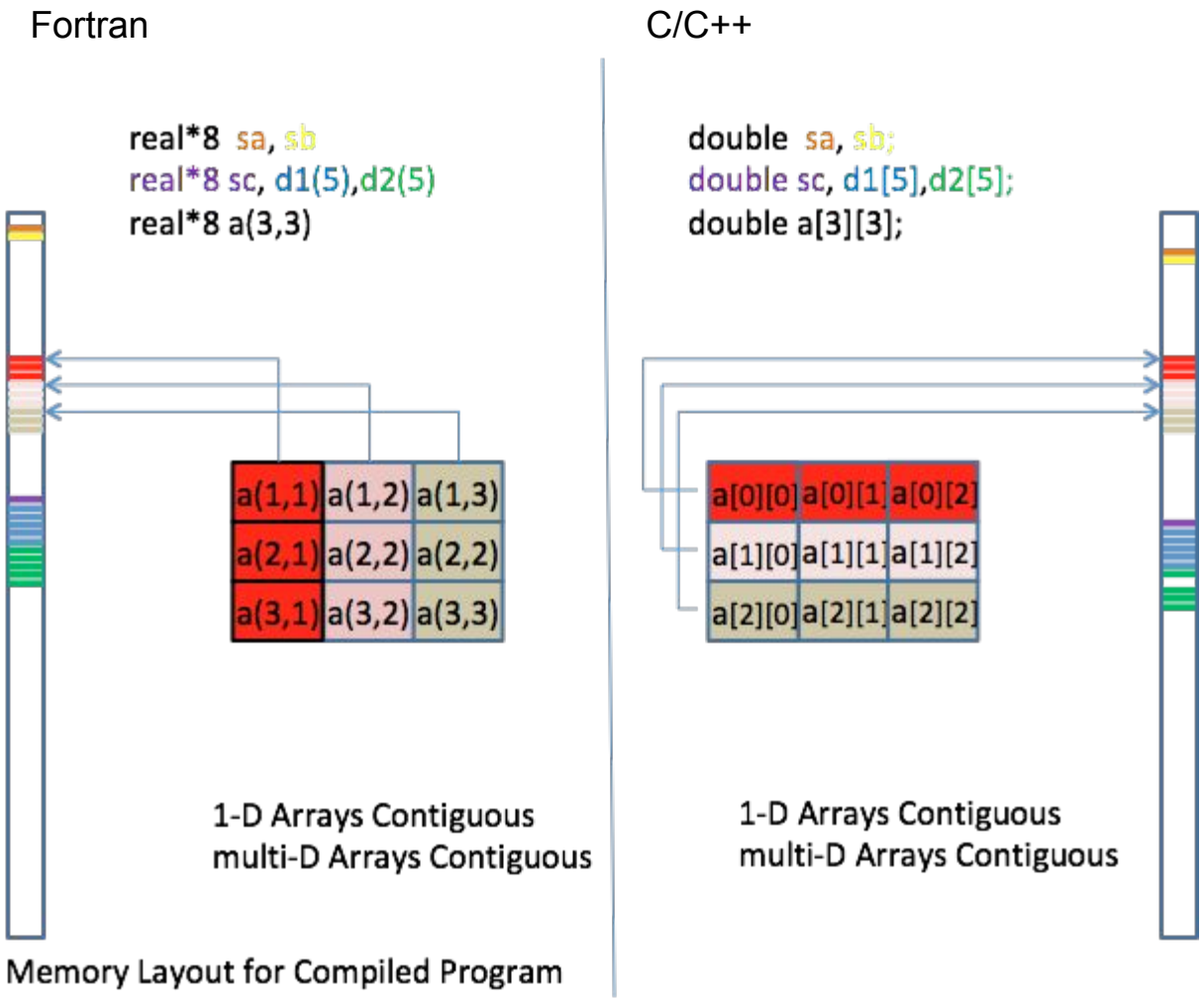
Parallel Computing for Science & Engineering
Introduction to MPI, Advanced Collectives
Spring 2018

Instructors:
Charlie Dey, TACC
Lars Koesterke, TACC

Collectives, Summary



Contiguous Data and Alignment



Building a Matrix with MPI_Gather in Fortran

```
 program gather
! Build matrix A from column vectors v; 4 processors, A=4x4.
! MAP: A = [v0,v1,v2,v3]  vi = column vector from process I.
!
    integer,parameter :: N=4
    real*8            :: a(N,N),v(N)
    include 'mpif.h'
    call mpi_init(ierr)
    call mpi_comm_rank(MPI_COMM_WORLD,mype,ierr)
    call mpi_comm_size(MPI_COMM_WORLD,npes,ierr)
    if(npes.ne.N) stop
! Vector Syntax (each element of v assigned mype)
    v=mype
    call mpi_gather(v,N,MPI_REAL8,     &
                    a,N,MPI_REAL8, 0,MPI_COMM_WORLD,ierr)
    if(mype.eq.0) write(6,'(4f5.0)') ((a(i,j),j=1,N),i=1,4)
    call mpi_finalize(ierr)

end program
```

Building a Matrix with MPI_Gather in C

```
#include <mpi.h>
#include <stdio.h>
#define  N 4
main(int argc, char **argv){
/* Build matrix A from ROW vectors v; 4 processors, A=4x4.
   MAP: A = [v0,v1,v2,v3]  vi = vector ROW from process i. */
   int npes, mype, ierr;
   int i, j;
   double a[N][N], v[N];
      ierr = MPI_Init(&argc, &argv);
      ierr = MPI_Comm_size(MPI_COMM_WORLD, &npes);
      ierr = MPI_Comm_rank(MPI_COMM_WORLD, &mype);
      if(npes != N){ printf("Use %d PEs\n",N); exit(9);}

      for(i=0; i<N; i++) v[i] = (double) mype; /* Fill v with PE# */

 /*Gather up ROW vecs into matrix "a" on PE 0.*/
      ierr = MPI_Gather(v,N,MPI_DOUBLE,
                        a,N,MPI_DOUBLE, 0,MPI_COMM_WORLD);
   if(mype == 0)
    for(i=0; i<N; i++){
     for(j=0; j<N; j++) printf("%5f ", a[i][j]);
     printf("\n"); }
   ierr = MPI_Finalize();
}
```

Building a Matrix with MPI_Gather in C (w/ malloc)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define  N 4
main(int argc, char **argv){
/* Build matrix A from ROW vectors v; 4 processors, A=4x4.
   MAP: A = [v0,v1,v2,v3]  vi = vector ROW from process i. */
   int i,j, npes, mype, ierr;
   double *amemblk, **a, v[N];
      ierr = MPI_Init(&argc, &argv);
      ierr = MPI_Comm_size(MPI_COMM_WORLD, &npes);
      ierr = MPI_Comm_rank(MPI_COMM_WORLD, &mype);
      if(npes != N){ printf("Use %d PEs\n",N); exit(9);}
      amemblk = (double * ) malloc(N*N*sizeof(double  ));
      a       = (double **) malloc(  N*sizeof(double *));
      for(i = 0; i < N; i++) a[i] = &amemblk[i*N];

      for(i=0; i<N; i++) v[i] = (double) mype; /* Fill v with PE# */

      ierr = MPI_Gather(v,       N,MPI_DOUBLE,
                        &a[0][0],N,MPI_DOUBLE, 0,MPI_COMM_WORLD);
   if(mype == 0)
    for(i=0; i<N; i++){
    for(j=0; j<N; j++) printf("%5f ", a[i][j]); printf("\n"); }
   ierr = MPI_Finalize();
}
```

Scatter - Work - Gather

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int numnodes, myid, mpi_err;         /*globals*/
#define mpi_root 0

void my_init(int  *argc, char ***argv) {
    mpi_err = MPI_Init(argc,argv);
    mpi_err = MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
    mpi_err = MPI_Comm_rank( MPI_COMM_WORLD, &myid);
}
```

Scatter - Work - Gather

```
int main(int argc, char *argv[]){

   int *myray, *send_ray, *back_ray;
   int count, size,mysize,i,k,j,total;

   my_init(&argc,&argv);

   count=4;                    /*each task get 4 elements*/
   myray=(int*)malloc(count*sizeof(int));

   if(myid == mpi_root){       /*create send data*/
      size=count*numnodes;
      send_ray=(int*)malloc(    size*sizeof(int));
      back_ray=(int*)malloc(numnodes*sizeof(int));
      for(i=0;i<size;i++) send_ray[i]=i;
   }
```

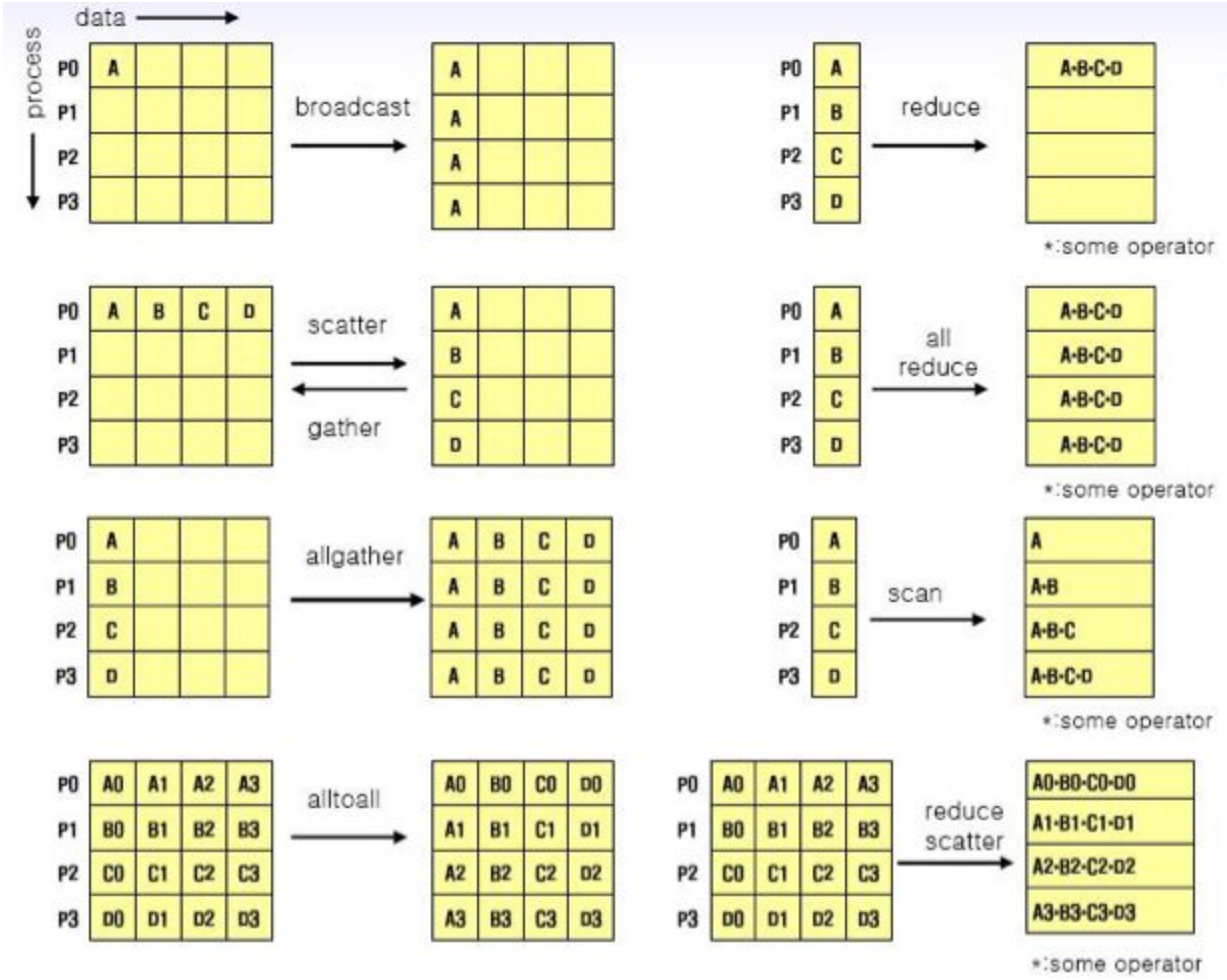
Scatter - Work - Gather

```
  mpi_err=MPI_Scatter(send_ray,count, MPI_INT,
                          myray,count, MPI_INT, mpi_root,MPI_COMM_WORLD);

   total=0;                                              /*partial sum*/
   for(i=0; i<count; i++) total=total+myray[i];
   printf("myid= %d total= %d\n ",myid, total);
                                                        /*send back sum*/
   mpi_err = MPI_Gather(&total,   1, MPI_INT,
                        back_ray, 1, MPI_INT, mpi_root, MPI_COMM_WORLD);

   if(myid == mpi_root){
      total=0;
      for(i=0; i<numnodes; i++) total=total+back_ray[i];
      printf("results from all processors= %d \n ",total);
   }
   mpi_err = MPI_Finalize();
}
```

Collectives, Summary



MPI Advanced Collectives

- `MPI_Alltoall`: Each processor sends and receives data to/from all others

MPI Advanced Collectives

MPI_Alltoall:

C

```
ierr = MPI_Alltoall(&sbuf[0], scnt, stype, &rbuf[0], rcnt,
rtype, comm);
```

Fortran

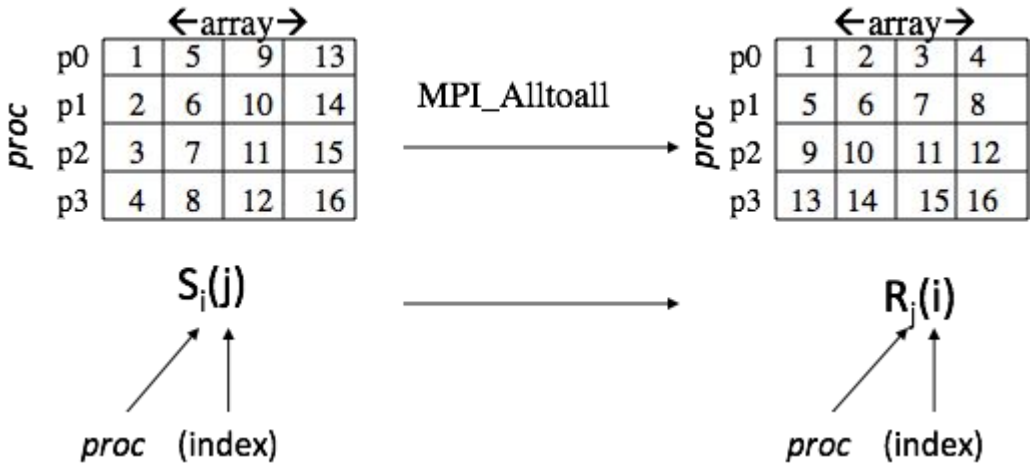
```
call MPI_Alltoall( sbuf, scnt, stype, rbuf, rcnt, rtype,
comm, ierr)
```

Parameters

```
scnt: # of elements sent from each processor
sbuf: sending array of size scnt*number_of_processors
rcnt: # of elements obtained from each proc.
rbuf: receiving array, size rcnt*number_of_processors
```

MPI Advanced Collectives

MPI_Alltoall



MPI Advanced Collectives

Introducing "V" Operators, allows the size of data in the send and receive buffers to vary on each processor

- `MPI_Gatherv`: Gather different amounts of data from each processor to the root processor
- `MPI_Allgatherv`: Gather different amounts of data from each processor and sends all data to each
- `MPI_Scatterv`: Send different amounts of data to each processor from the root processor
- `MPI_Alltoallv`: Send and receive different amounts of data form all processors

MPI Advanced Collectives

MPI_Gatherv:

C

```
ierr = MPI_Gatherv(&sbuf[0], scnt, stype,
                   &rbuf[0], &rcnts[0], &rdispls[0], rtype,
                   root, comm);
```

Fortran

```
call MPI_Gatherv(sbuf, scnt, stype,
                 rbuf, rcnts, rdispls, rtype,
                 root, comm, ierr)
```

Parameters

scnt: # of elements sent from each processor

sbuf: sending array of size scnt

rcnt: ***array*** of counts to be received from each processor:
  1st element # from processor 0, 2nd from processor 1, etc

rdispls: ***array*** of displacements (offsets)

MPI Advanced Collectives

`MPI_Gatherv:`

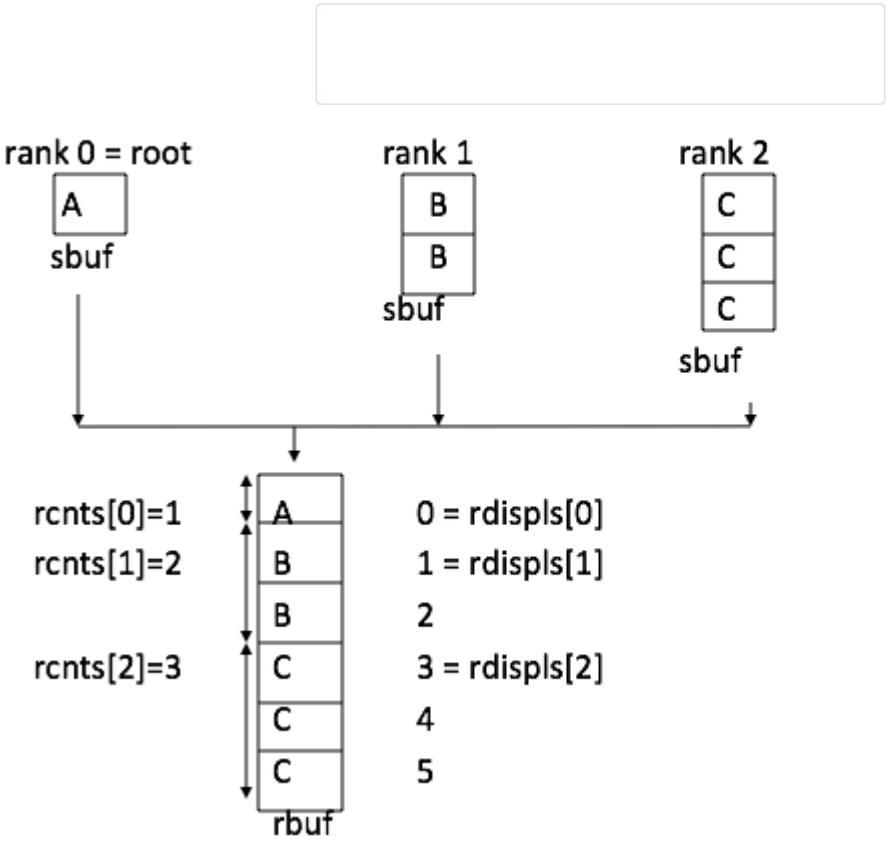
C

```
MPI_Gatherv(&sbuf[0], scnt, stype,
            &rbuf[0], &rcnts[0], &rdispls[0], rtype,
            root, comm);
```

- `abuf` and `scnt` are placed in `rbuf` in rank order:
  - `rcnts(i)` elements, starting at offset `rdispls(i)`
  - for i = {0,…,n-1} of group of n tasks.
- Size of data send by rank i and received in root rcnts(i) must be equal.
- receiving variables not “significant” on non-root

MPI Advanced Collectives

`MPI_Gatherv:`



MPI Advanced Collectives

MPI_Gatherv:

C

```
...
   MPI_Comm comm;
   int gsize,sendarray[100];
   int root, *rbuf, stride;
   int *displs,i,*rcounts;
...
   MPI_Comm_size( comm, &gsize);
   rbuf = (int *)malloc(gsize*stride*sizeof(int));
   displs = (int *)malloc(gsize*sizeof(int));
   rcounts = (int *)malloc(gsize*sizeof(int));
   for (i=0; i<gsize; ++i) {
       displs[i] = i*stride;
       rcounts[i] = 100;
   }
   MPI_Gatherv( sendarray, 100, MPI_INT, rbuf, rcounts,
   displs, MPI_INT, root, comm);
```

MPI Advanced Collectives

- `MPI_Alltoallv`: Send and receive different amounts of data form all processors

MPI Advanced Collectives

MPI_Alltoallv:

C

```
ierr = MPI_Alltoallv(&sbuf[0],&scnts[0],&sdispls[0],stype,
                     &rbuf[0],&rcnts[0],&rdispls[0],rtype,
                     comm);
```

Fortran

```
call MPI_Alltoallv(sbuf, scnts, sdispls, stype,
                   rbuf, rcnts, rdispls, rtype,
                   comm, ierr)
```

Parameters

scnt: # of elements sent from each processor

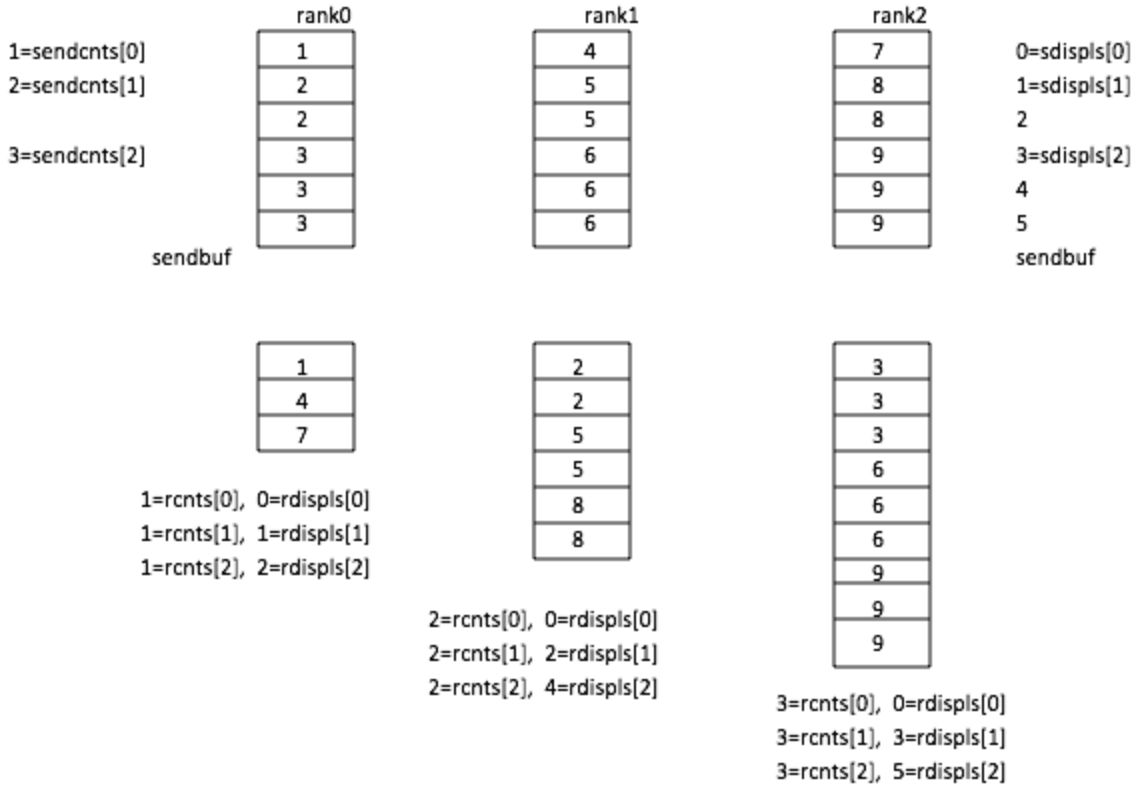
sbuf: sending array of size scnt

rcnt: ***array*** of counts to be received from each processor:
  1st element # from processor 0, 2nd from processor 1, etc

rdispls: ***array*** of displacements (offsets)

MPI Advanced Collectives

`MPI_Alltoallv:`



MPI Advanced Collectives

MPI_Alltoallv:

C

```
int main(int argc,char *argv[]){
    int *sray,*rray;
    int *sdisp,*scounts,*rdisp,*rcounts;
    int ssize,rsize,i,k,j;
    float z;

   mpi_err = MPI_Init(argc,argv);
   mpi_err = MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
   mpi_err = MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    scounts=(int*)malloc(sizeof(int)*numnodes);
    rcounts=(int*)malloc(sizeof(int)*numnodes);
    sdisp=(int*)malloc(sizeof(int)*numnodes);
    rdisp=(int*)malloc(sizeof(int)*numnodes);

    seed_random(myid);
```

MPI Advanced Collectives

```
/* find out how much data to send */
     for(i=0;i<numnodes;i++){
          random_number(&z);
          scounts[i]=(int)(5.0*z)+1;
     }
/* tell the other processors how much data is coming */
     mpi_err = MPI_Alltoall(   scounts,1,MPI_INT,
                               rcounts,1,MPI_INT,
                               MPI_COMM_WORLD);
```

MPI Advanced Collectives

```
/* calculate displacements and the size of the arrays */
    sdisp[0]=0;
    for(i=1;i<numnodes;i++){
        sdisp[i]=scounts[i-1]+sdisp[i-1];
    }
    rdisp[0]=0;
    for(i=1;i<numnodes;i++){
        rdisp[i]=rcounts[i-1]+rdisp[i-1];
    }
    ssize=0;
    rsize=0;
    for(i=0;i<numnodes;i++){
        ssize=ssize+scounts[i];
        rsize=rsize+rcounts[i];
    }
```

MPI Advanced Collectives

```
/* allocate send and rec arrays */
    sray=(int*)malloc(sizeof(int)*ssize);
    rray=(int*)malloc(sizeof(int)*rsize);
    for(i=0;i<ssize;i++)
        sray[i]=myid;
/* send/rec different amounts of data to/from each processor */
    mpi_err = MPI_Alltoallv(  sray,scounts,sdisp,MPI_INT,
                              rray,rcounts,rdisp,MPI_INT,
                              MPI_COMM_WORLD);

    mpi_err = MPI_Finalize();
}
```

MPI

- Derived Data Types
- Parallel IO
- MPI Quiz
- Homework