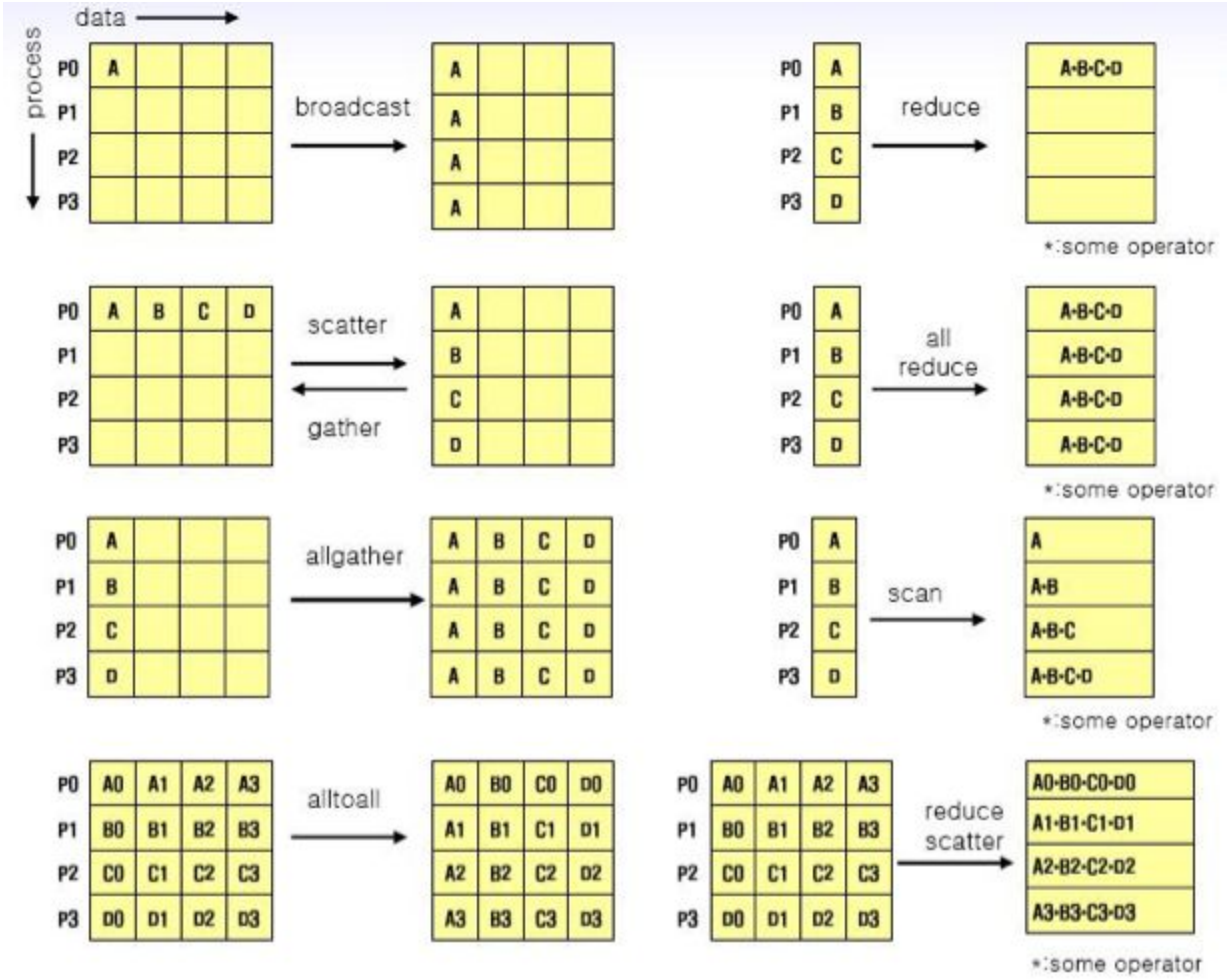
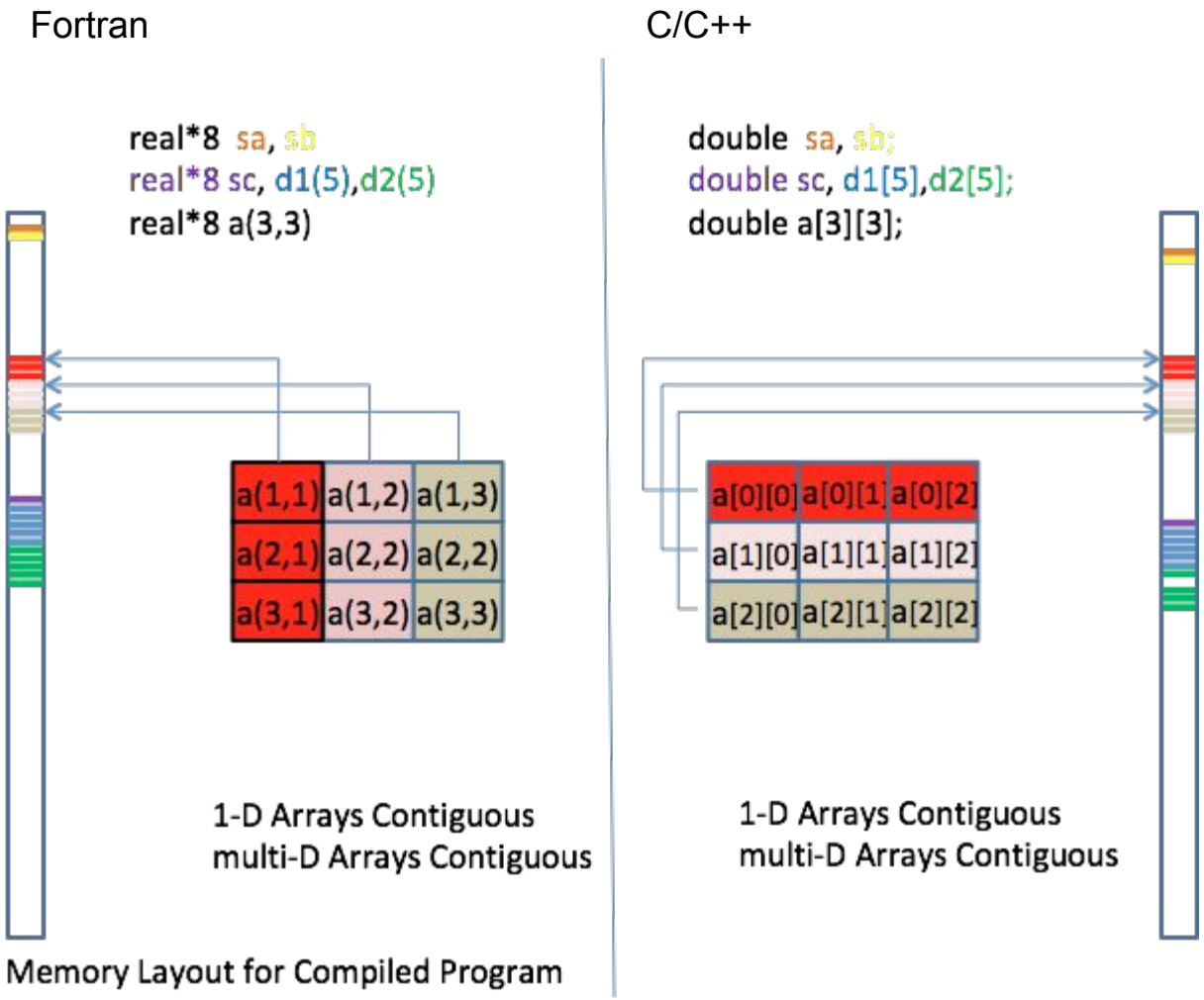
Parallel Computing for Science & Engineering
Introduction to MPI, Advanced Collectives
Spring 2018

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;
   }
```

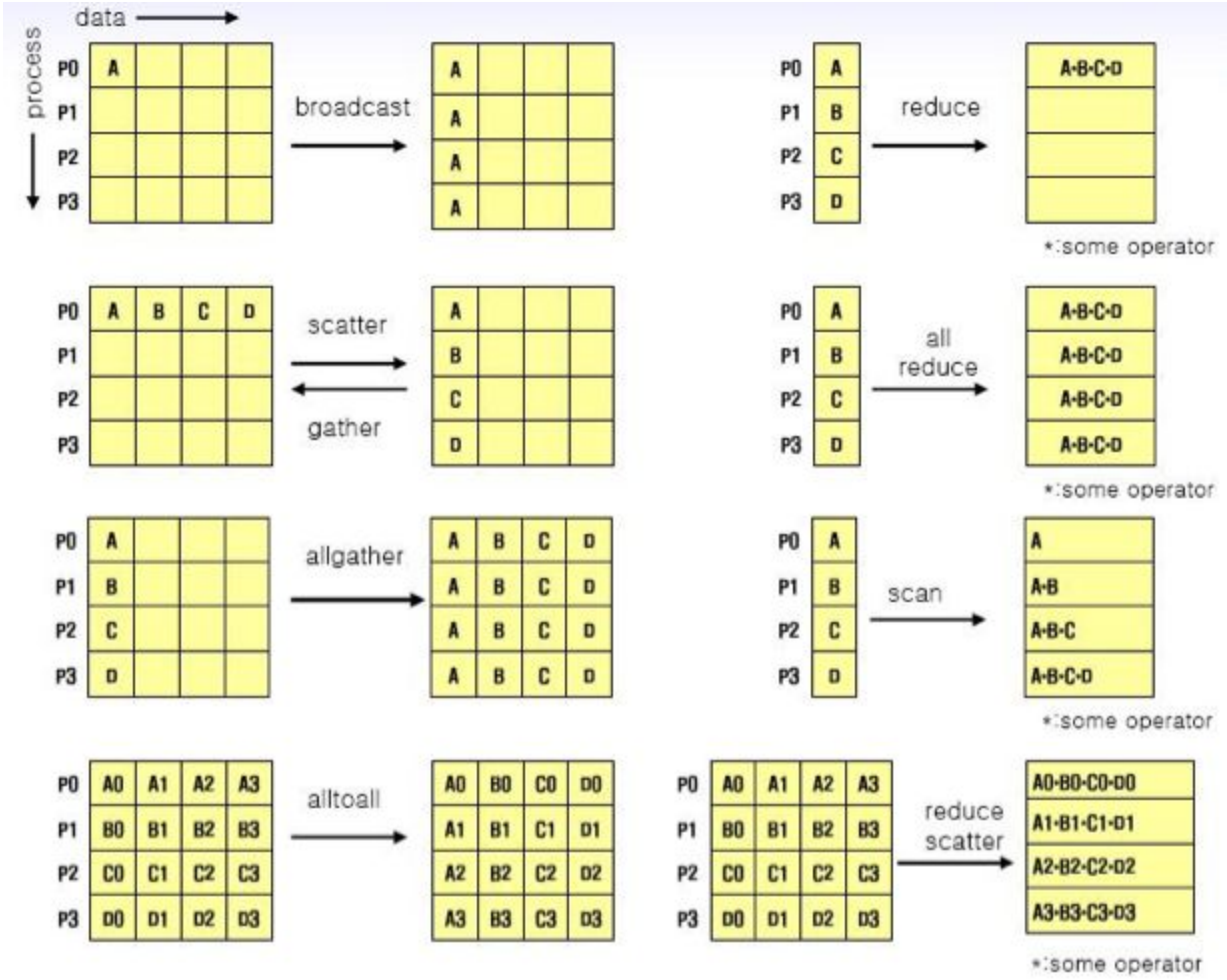
Scatter - Work - Gather

```
  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 Collectives

- `MPI_AllGather:` gather, and leave the result everywhere.
- `MPI_Allreduce:` do a reduction, but leave the result everywhere.

MPI Collectives

MPI_Allgather:

C

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

Fortran

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

Parameters

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

MPI Collectives

MPI_Allreduce:

C

```
ierr = MPI_Allreduce(&sbuf[0],&rbuf[0],cnt,type,op,comm)
```

Fortran

```
call MPI_Allreduce(sbuf, rbuf, cnt,type,op,comm,ierr)
```

Parameters

sbuf: array to reduce

rbuf: receive buffer

cnt: sbuf and rbuf size

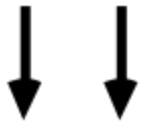
type: datatype

operation: binary operator

MPI Collectives

MPI_Reduce, Global Sum

	X(0)	X(1)	X(2)
TASK 1	A0	B0	C0
TASK 2	A1	B1	C1
TASK 3	A2	B2	C2



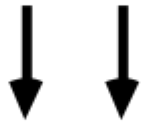
	X(0)	X(1)	X(2)
TASK 1	A0+A1+A2	B0+B1+B2	C0+C1+C2
TASK 2			
TASK 3			

TACC
THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

MPI Collectives

MPI_Allreduce, Global Sum

	X(0)	X(1)	X(2)
TASK 1	A0	B0	C0
TASK 2	A1	B1	C1
TASK 3	A2	B2	C2



	X(0)	X(1)	X(2)
TASK 1	A0+A1+A2	B0+B1+B2	C0+C1+C2
TASK 2	A0+A1+A2	B0+B1+B2	C0+C1+C2
TASK 3	A0+A1+A2	B0+B1+B2	C0+C1+C2

MPI Collectives

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

MPI 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 to each processor

sbuf: is an array of size scnt*np (np=# of processes)

rcnts: # of elements obtained from each processor

rbuf: size rcnt*np

Note: send buffer and receive buffer must be of size = scnt * np