MPI Lab

Steve Lantz
Susan Mehringer

Introduction to Parallel Computing
May 23, 2011

MPI Lab

- Parallelization (Calculating π in parallel)
  - How to split a problem across multiple processors
  - Broadcasting input to other nodes
  - Using MPI_Reduce to accumulate partial sums
- Sharing Data Across Processors (Updating ghost cells)
  - How ghost cells are used in finite difference problems
  - Using Sendrecv for deadlock-free transfers involving simultaneous Sends and Receives on a node

Getting Started

- Login to ranger.tacc.utexas.edu **-or-** lonestar.tacc.utexas.edu
- Untar the lab source code

```
login3% cd $HOME
login3% tar xf ~train100/labs/mpi_lab.tar
```

- Part 1: Calculating π

```
cd $HOME/mpi_lab/pi
```

- Part 2: Ghost Cell Update

```
cd $HOME/mpi_lab/ghosts
```

Part 1: Calculating π – Basic Course of Action

- Objective: parallelize serial π calculation, starting with serial code (serial_pi.c or serial_pi.f90).

```
for (i=1; i<=n; i++) {
  x  = h * ( (double)(i) - 0.5e0 );
  sum = sum + f(x); }
```

```
do i = 1, n
  x = h * (dble(i) - 0.5_KR8)
  sum = sum + f(x)
end do
```

- Each processor will perform a partial sum for x_i, x_{i+N}, x_{i+2N}, x_{i+3N} ,… where N is the processor count, and i is the rank.

```
for (i=myid+1; i<=n; i=i+numprocs) {
  x = h * ( (double)(i) - 0.5e0 );
  sum = sum + f(x); }
```

```
do i = myid+1, n, numprocs
  x = h * (dble(i) - 0.5_KR8)
  sum = sum + f(x)
end do
```

- Accumulate and add partial sums on processor 0.

```
ierr = MPI_Reduce(&part_pi,&pi,1,MPI_DOUBLE,          MPI_SUM,0,MPI_COMM_WORLD     )
call   MPI_Reduce(mypi,     pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)
```

Calculating π – MPI_Init and Finalize

- Modify the serial_pi.f or serial_pi.c file.
  - cp serial_pi.f90 pi.f90 or cp serial_pi.c pi.c
  - Include MPI startup and finalization routines at the beginning and end of pi.c/f90. Also include declaration statements for the rank and number of processors (myid and numprocs, respectively)

```
C: #include "mpi.h" or  F90:  include "mpif.h"
 …MPI_Init(…)
 …MPI_Comm_rank(MPI_COMM_WORLD...)
 …MPI_Comm_size(MPI_COMM_WORLD…)
```

Initialization

```
 …
```

Serial Code

```
 …MPI_Finalize(…)
```

End of Code

Don't forget:
Declare myid, numprocs, and ierr as ints in C, integers in Fortran
Use "call" and an error argument in FORTRAN; error is a return value in C code
Use myid and numprocs for the rank and processor count

Calculating π – Read & Form Partial Sums

- Have rank 0 processor read n, the total # elements to integrate
  - Make the read statement conditional, only on root, with:
    if (myid == 0) read…
  - Broadcast n to the other nodes
    MPI_Bcast(n,1,<datatype>,0,MPI_COMM_WORLD…)
    Use MPI_INTEGER and MPI_INT for Fortran and C datatypes, respectively (use &n address for C)
- Specify integral elements for each processor
  - F90: do i = 1,n → do i = myid+1, n, numprocs
  - C: for(i=1; i<=n; i++) → for(i=myid+1; i<=n; i=i+numprocs)

Calculating π – MPI_Reduce Partial Sums

- Assign the sum from each rank to a partial sum
  - declare part_pi as a double [real(KR8) in F90]
  - after the loop, replace “pi = h * sum” with:
    part_pi = h * sum; followed by
- Sum the partial sums with an MPI_Reduce call

  …MPI_Reduce(part_pi,pi,1,<type>,MPI_SUM,0,
  MPI_COMM_WORLD…)

  where <type> is MPI_DOUBLE or MPI_DOUBLE_PRECISION for C and F90, respectively; use addresses &part_pi and &pi in C code
- Write out π & calc. pi, from rank 0 proc (use if)
  - if (myid == 0) print…

Calculating π – Testing the Code

- Compile code (see parallel_pi.c or .f90 for solution)

```
mpif90 -O3 pi.f90
mpicc -O3 pi.c
```

- Prepare job (edit 'job' in current directory)

  Modify the processor count:
  - Set the # of processors/node, #way, to 16 for Ranger (or 12 for Lonestar)
  - The last argument, divided by 16 (or 12), is the number of nodes

  Add a line to identify your account (optional)

```
#$ -A 20110523HPC
```

  Create a file called "input" including the total elements (n) on the first line:

```
echo 2000 > input
```

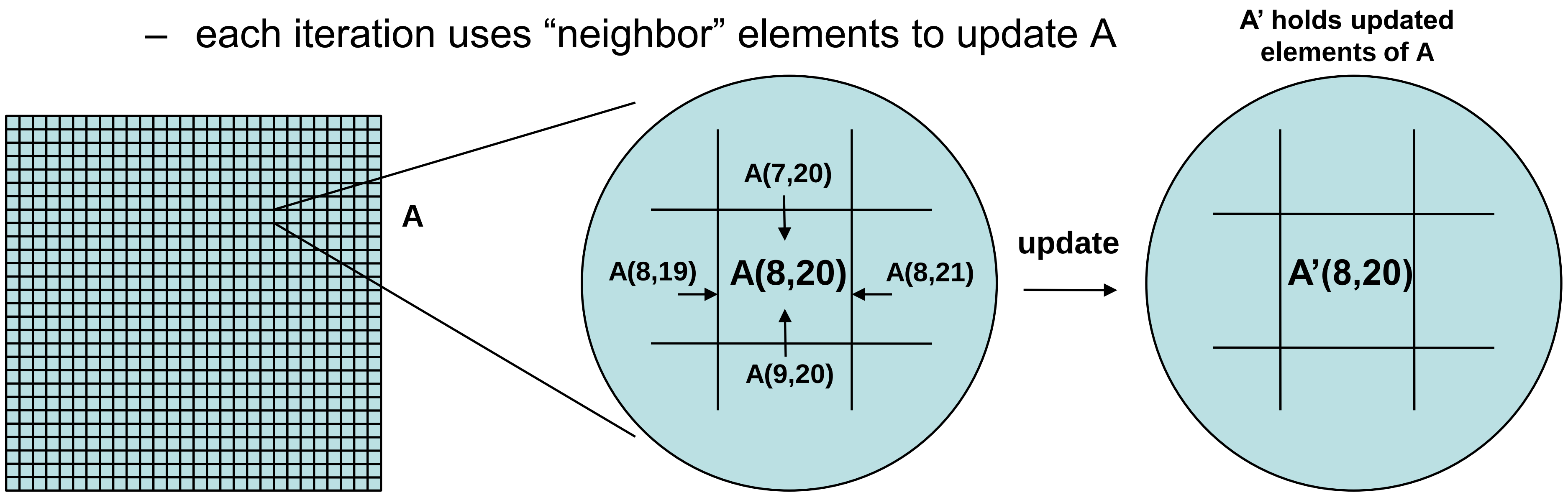
- Submit job

```
qsub job
showq -u
```

Part 2:
Sharing Data Across Processors

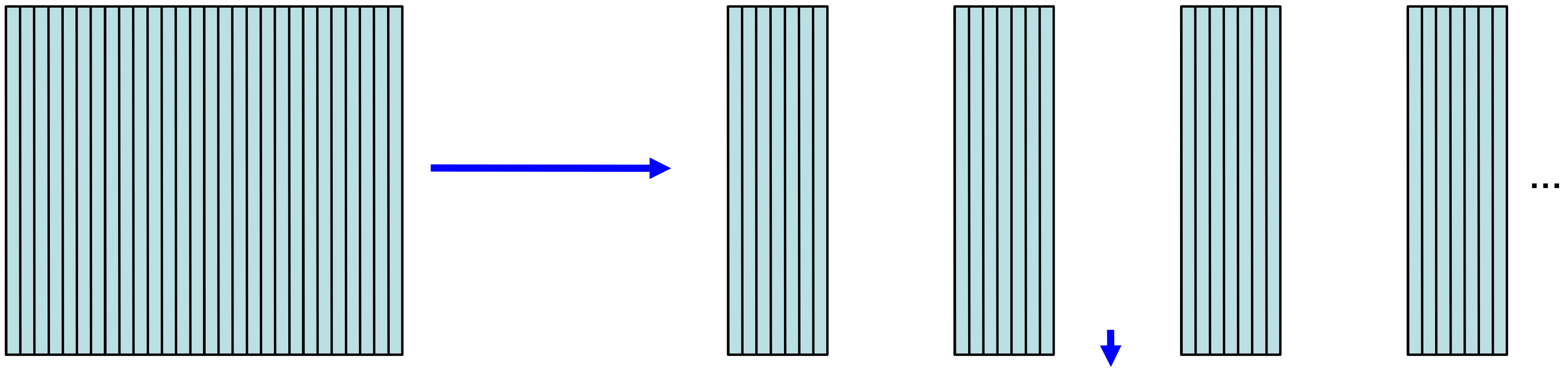
Overview

- Solve 2-D partial differential equation (finite difference)
  - represent x-y domain as 2-D grid of points*
  - solution matrix=A(x,y)
  - initialize grid elements with guess
  - iteratively update solution matrix (A) until converged
  - each iteration uses “neighbor” elements to update A

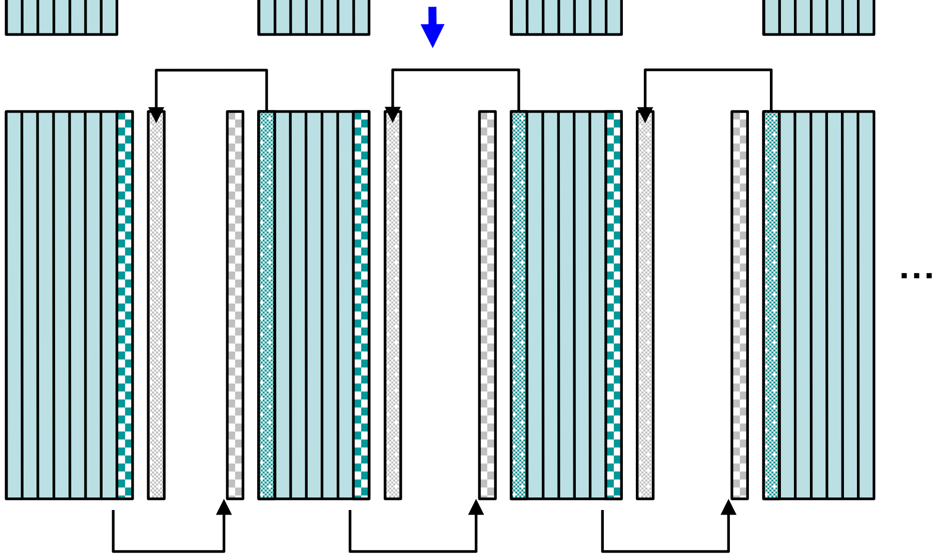


Domain Decomposition

Decompose 2-D grid into column blocks across p processors



Need to duplicate edge columns on neighbor processors & send updated values after each iteration. That is, create ghost columns (gray) from real columns (patterned) on neighbor processors.



Sharing Data Across Processors – Serial to Parallel

- From a simple serial code, decompose a domain (matrix) into column slices for each processor, include ghost cells, and create a subroutine for transferring real (calculated) columns to ghost column on the neighbor processor. Extend the A matrix to hold the neighbors: A(N,N)→A(N,N+2).
- Instructions:

```
cd $HOME/mpi_lab/ghosts
cp serial.c myghost.c       (for C programmers)
cp serial.f90 myghost.f90 (for F90 programmers)
```

(ghost_1d.c/f90 are example, completed codes)

Outline: Serial To Parallel

serial code (serial) → parallel code (myghost)

```
main program
   matrix A



loop
   jacob_update(A)

end loop



end main
jacob_update
```

```
main program
   matrix A  {include ghosts in A}

initialize MPI, get rank size

loop
   jacob_update(A)
   ghost_exchange(A)
end loop

finalize MPI

end main
jacob_update modify for ghosts
routine  ghost_exchange
```

Domain Decomposition

- Look over the serial.c or serial.f90 code.
  - The code loops over a jacob_update routine which simply increases all values in a matrix (to emulate a stencil update in a Finite Difference code).

Fortran

```
real*8 :: A(n,n)
…

do iter = 1,LOOPS
   call jacob_update(a,n,iter)
end do
…
subroutine jacob_update()
    A(i,j) = iter
```

C

```
#define A(i,j)  a( (i-1) + (j-1)*n )
double a[n*n];

for(iter=1; iter<=LOOPS; iter++){
   jacob_update(a,n,iter)
}
…
Void jacob_update(){
   A(i,j) = (double) (iter);
```

Matrix Layout – Serial Code

columns

$$\begin{array}{c|cccc|l} \text{rows} & 1 & 2 & 3 & 4 & j \\ \hline 1 & 1 & 5 & 9 & 13 & \\ 2 & 2 & 6 & 10 & 14 & \\ 3 & 3 & 7 & 11 & 15 & \\ 4 & 4 & 8 & 12 & 16 & \\ \hline i & & & & & \end{array}$$

Fortran

```
indexing: { i = 1,n; j=1,n }

real*8 :: A(n,n);

A(i,j)…
```

C

```
indexing: { i = 1,n; j=1,n }

#define A(i,j)  a( (i-1) + (j-1)*n )
double  a[ n*n ];

A(i,j)...
```

Domain Decomposition

Matrix Layout with Ghost Cells

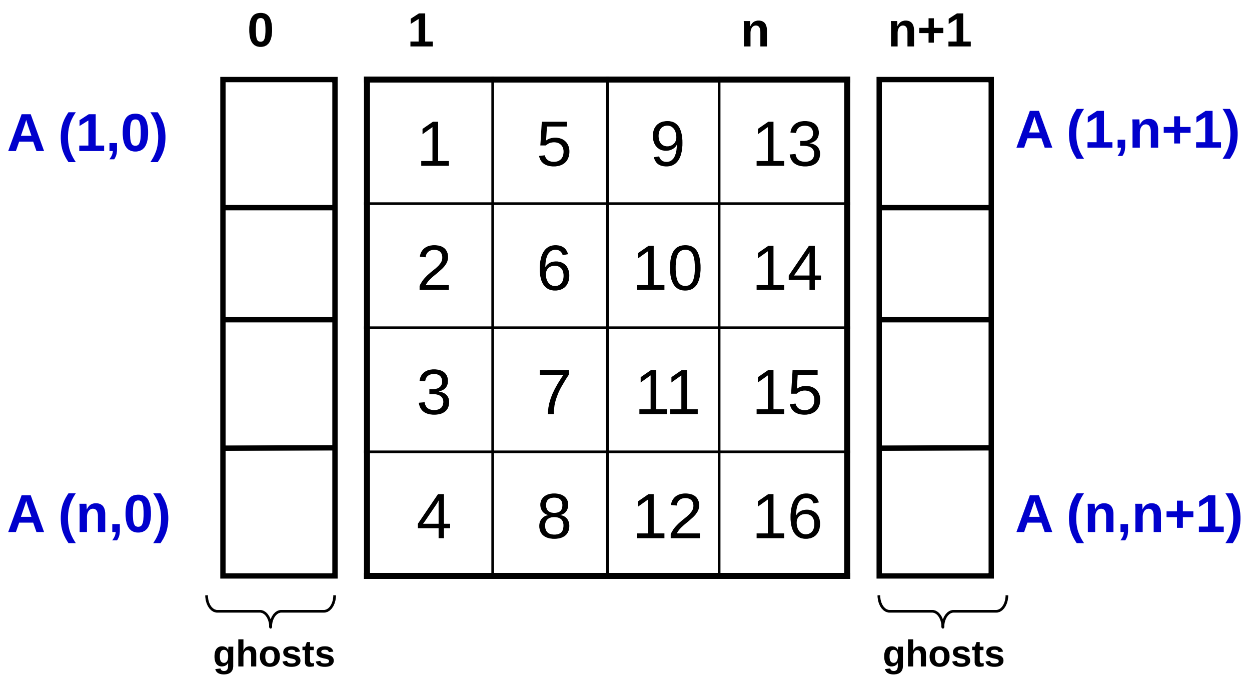
Redefine array for easy ghost access

```
real*8 :: A(n, 0:n+1)
```

Fortran

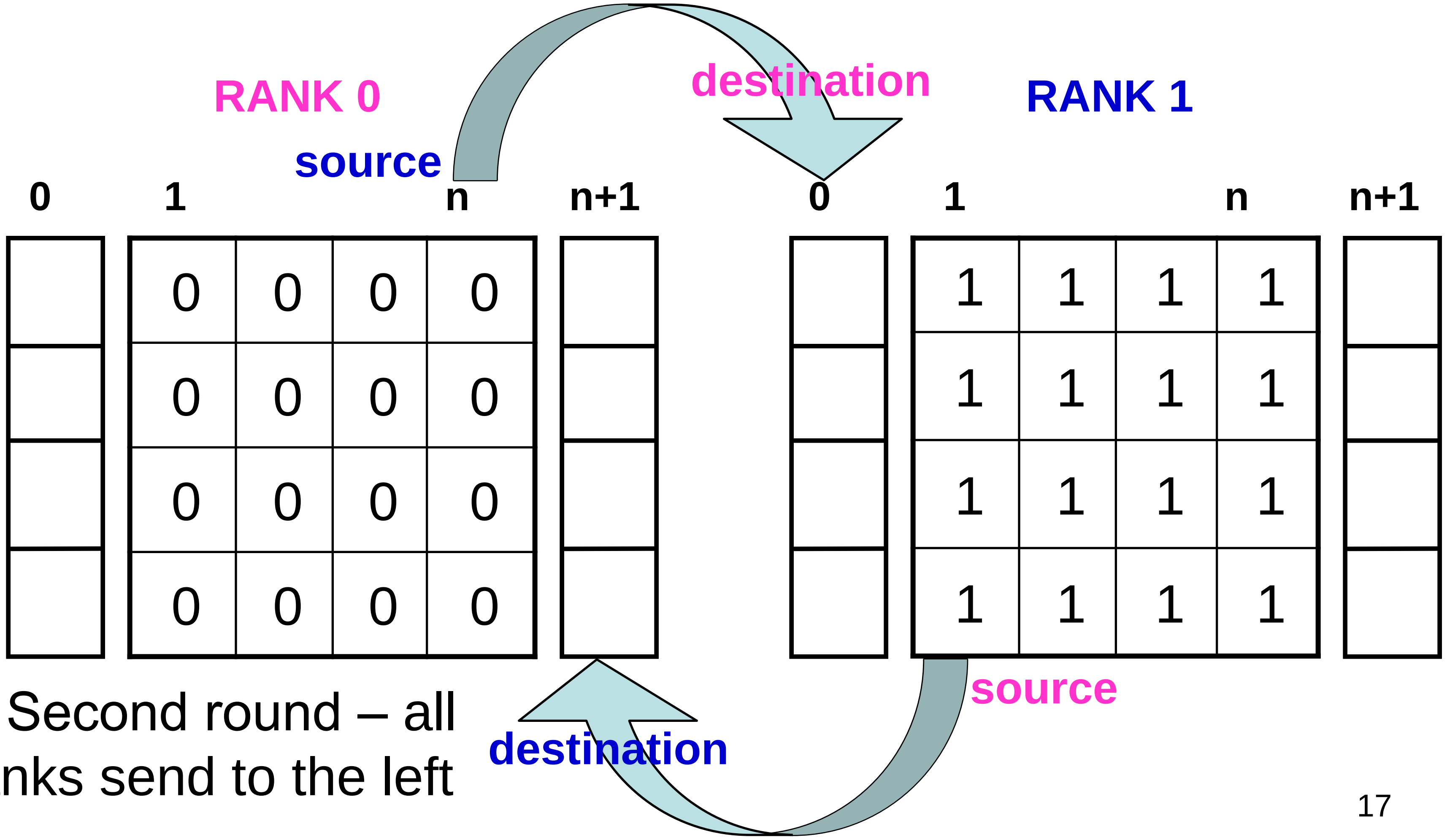
```
#define A(i,j)  a( (i-1) + (j)*n )
double a[n*(n+2)];
```

C



Exchange Using Sendrecv

First round – all ranks send to the right…



…Second round – all ranks send to the left

Domain Decomposition

- Include the usual MPI_Init & MPI_Finalize statements:

```
define ierr, irank, nranks as integers

…MPI_Init(…);
…MPI_Comm_rank(MPI_COMM_WORLD, irank*,...);
…MPI_Comm_size(MPI_COMM_WORLD, nranks*…);
…
…MPI_Finalize(…);
```

(Don’t forget to include mpif.h or mpi.h.)
(Don’t forget to declare irank and nranks.)
***   &irank and &nranks for C code**

Domain Decomposition

- Create a subroutine for the exchange:
  ghost_exchange(a,n,iter,irank,nranks)
- Create destination and source numbers for the exchange

```
idest = irank +  1;
isrc  = irank -  1;
if(idest == nranks) idest = MPI_PROC_NULL;
if(isrc  ==     -1) isrc  = MPI_PROC_NULL;
```

C prototype: void ghost_exchange(double *a, int n, int iter, int irank, int nranks);
include type statements for idest, isrc (integers)

Domain Decomposition

- Send right data column to right neighbor, into its left ghost column.

  **MPI_Sendrecv(A(1, n), n, <type>, idest, 8, A(1, 0), n, <type>,
  isrc, 8, MPI_COMM_WORLD, status,…);**

  See top arrow(s) of slide 17. Use &A(1,n), &A(1,0), &status for C.

- Send left data columns to left neighbor, into its right ghost column.

  **MPI_Sendrecv(A(1, 1), n, <type>, isrc, 9, A(1, n+1), n, <type>,
  idest, 9, MPI_COMM_WORLD, status,…);**

  See bottom arrow(s) of slide 17. Use &A(1,1), &A(1,n+1), &status for C.

  C declaration: MPI_Status status F90: integer status(MPI_STATUS_SIZE)

Domain Decomposition – jacob_update Changes

- Ghost column 0 & n+1 accommodated by C #define:

```
#define A(i,j)  a( (i-1) + (j-1)*n )      →      #define A(i,j)  a( (i-1) + (j)*n )
double  a[N*N];                                  double  a[N*(N+2)];

for(i=1; i<=n; i++){                no           for(i=1; i<=n; i++){
                                    change
  for(j=1; j<=n; j++){                →            for(j=1; j<=n; j++){
    A(i,j) = (double) (iter);                        A(i,j) = (double) (iter);
} }                                              } }
```

- Ghost column 0 & n+1 accommodated by F90 array declaration:

```
                                    no
                                    change
A(1:N, 1:N) = iter;                   →            A(1:N, 1:N) = iter;
```

Because new indexing in declaration accommodates ghost vectors:

```
real*8  :: A(1:n, 1:n)                →            real*8 :: A(n, 0:n+1)
```

Domain Decomposition – Testing the Code

- Compile code (see ghost_1d.c or .f90 for finished parallel version)

```
mpif90 -O3 myghost.f90
mpicc -O3 myghost.c
```

- Prepare job

  Modify the processor count:

  – Set the # of processors/node, #way, to 16 for Ranger (12 for Lonestar)

  – The last argument, divided by 16 (or 12), is the number of nodes

  Add a line to identify your account (optional)

```
#$ -A 20110523HPC
```

- Submit job

```
qsub job
showq -u
```