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 Ghosts Cells are used in Finite Difference problems
  - How to use SendRecv for deadlock-free transfers involving simultaneous Sends and Receives on a node.

Getting Started

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

```
lslogin3$ cd
lslogin3$ tar xvf ~train00/mpi_lab.tar
```

- Part 1: Calculating PI

```
cd mpi_lab/pi
```

- Part 2: Ghost Cell Update

```
cd mpi_lab/ghosts
```

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 to 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…)

Top of Code

…

Serial Code

…MPI_Finalize(…)

End of Code

Don’t forget:

(declare myid, numprocs and ierrs as ints in C and integers in fortran
Use “call” and an error argument in FORTRAN; and use error return in C code.
use myid and numprocs for the rank and processor count)

(and error argument in f90 codes)

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

    Use MPI_INTEGER and MPI_INT for Fortran and C datatypes, respectively.

- Figure out a way to break up the integral locally based on the total number of MPI tasks

Calculating π MPI-reduce partial sums, print

- Assign the sum from each rank to a partial sum
- Sum the partial sums using your favorite MPI collective
- Write out π from rank 0 proc
  - if (myid == 0) print…

Calculating π

- Compile code:

  mpif90 –O3 pi.f90

  mpicc –O3 pi.c

- Prepare job

  Modify the processor count:

  Keep the # of processors per node set to 16 (keep the "16way)

  The last argument, divided by 16, is the number of nodes.

  #$ -pe 16way 16 → change 16 to 32, 48, 64, etc.

  – create a file called "input" and include the total number of elements (n) on the first line (then play with the # of intervals and processor count to look at scalability)

    echo 2000 >input

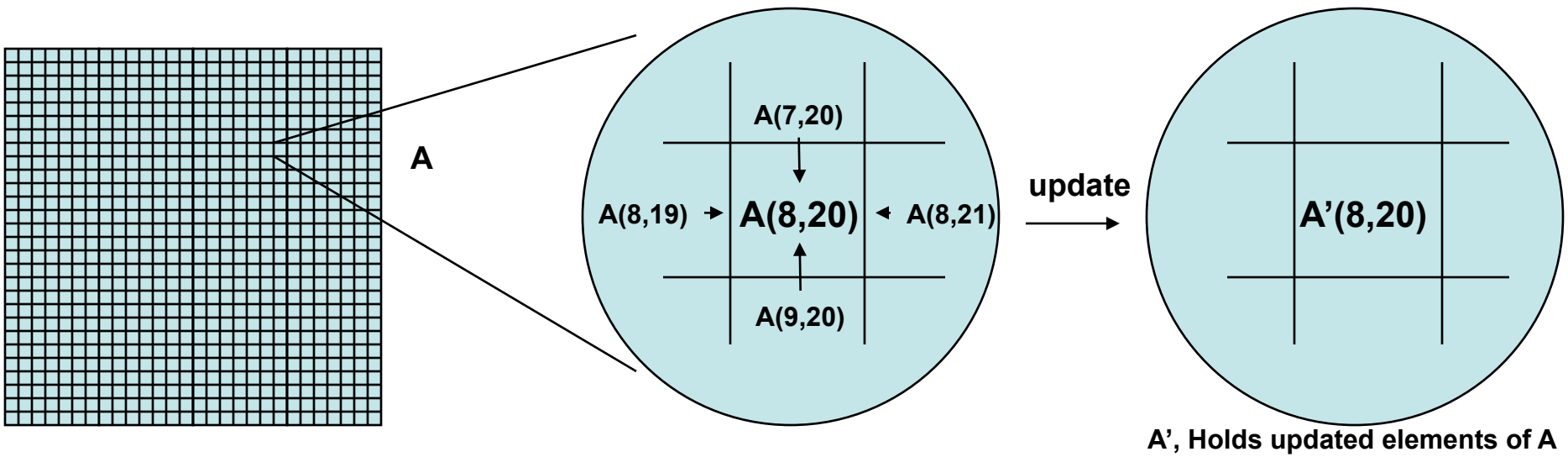
- Submit job

  qsub job

Part 2: Sharing Data Across Processors -- Overview

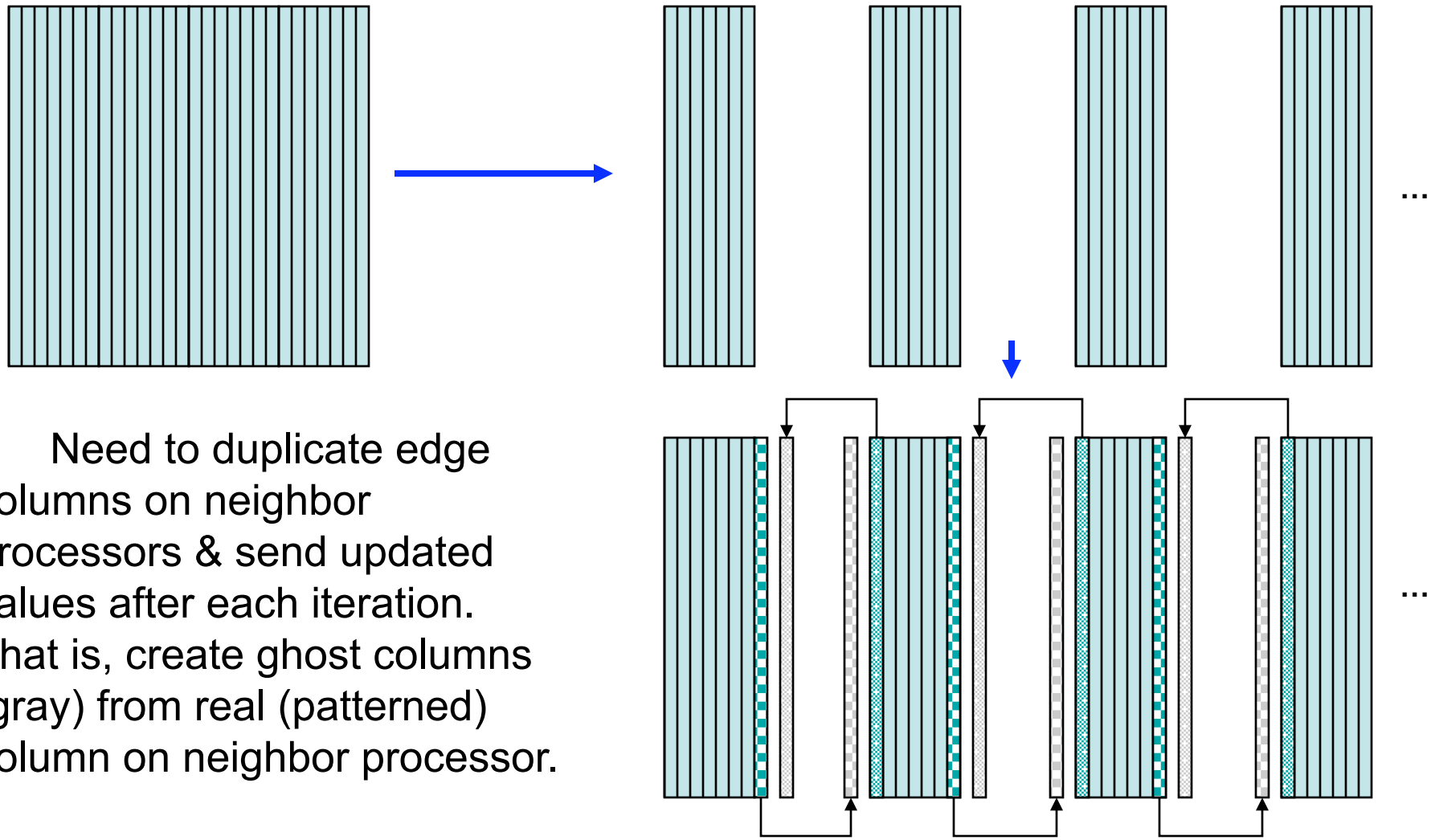
Solve 2-D partial differential equation (finite difference)

- 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.



Sharing Data Across Processors -- 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 (patterned) column on neighbor processor.

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)

Sharing Data Across Processors – Outline: Serial To Parallel

serial code → parallel code (myghost)

```
main program
   matrix A



loop
   jacobi_update(A)

end loop



end main
jacobi_update
```

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

initialize MPI, get rank size

loop
   jacobi_update(A)
   ghost_exchange(A)
end loop

finalize MPI

end main
jacobi_update modify for ghost
routine  ghost_exchange
```

Domain Decomposition

- Look over the serial.c or serial.f90 code.
  - The code loops over a jacobi_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)
..

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

C

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

do iter = 1,LOOPS
   call jacob_update(a,n,iter)
end do
…
Void jacobi_update(){
   A(i,j) = (double) (iter);
```

Domain Decomposition
Matrix Layout– Serial Code

rows \ columns	1	2	3	4 (j)
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15
4 (i)	4	8	12	16

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

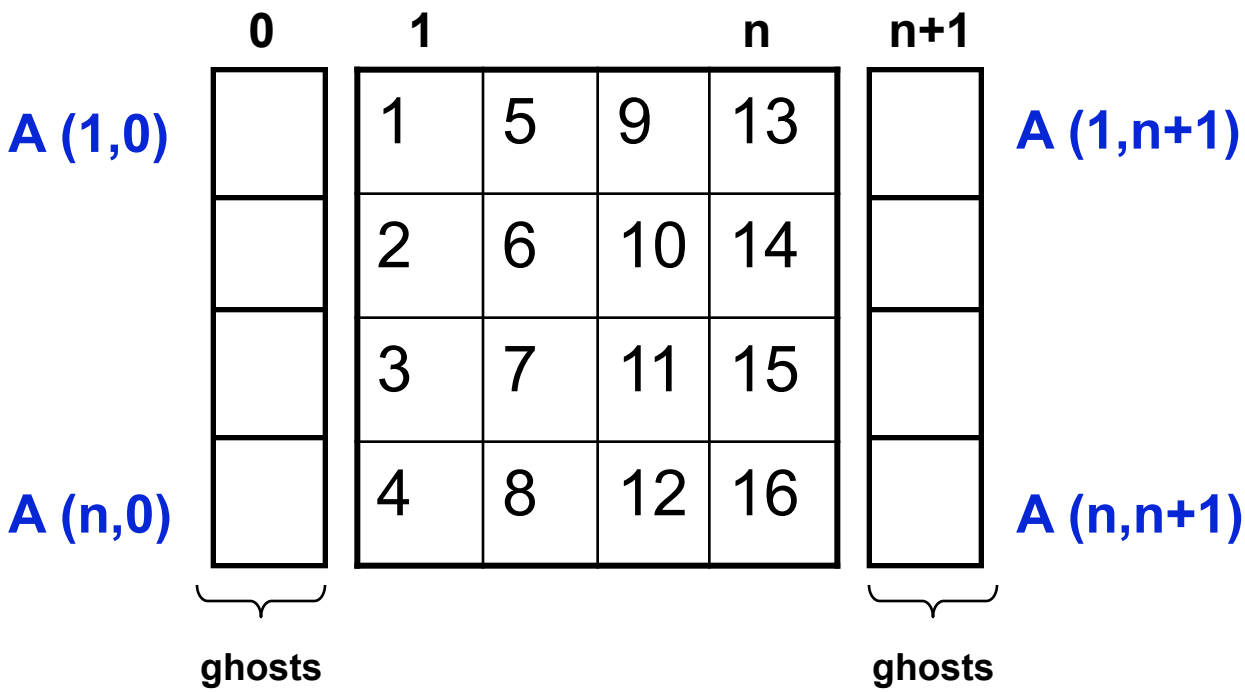
Redefine Array for easy ghost access

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

Fortran

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

C



Domain Decomposition

- Compile code:

  mpif90 –O3 myghost.f90

  mpicc –O3 myghost.c

- Prepare job

  Modify the processor count:

  Keep the # of precessors per node set to 16 (keep the “16way)

  The last argument, divided by 16, is the number of nodes.

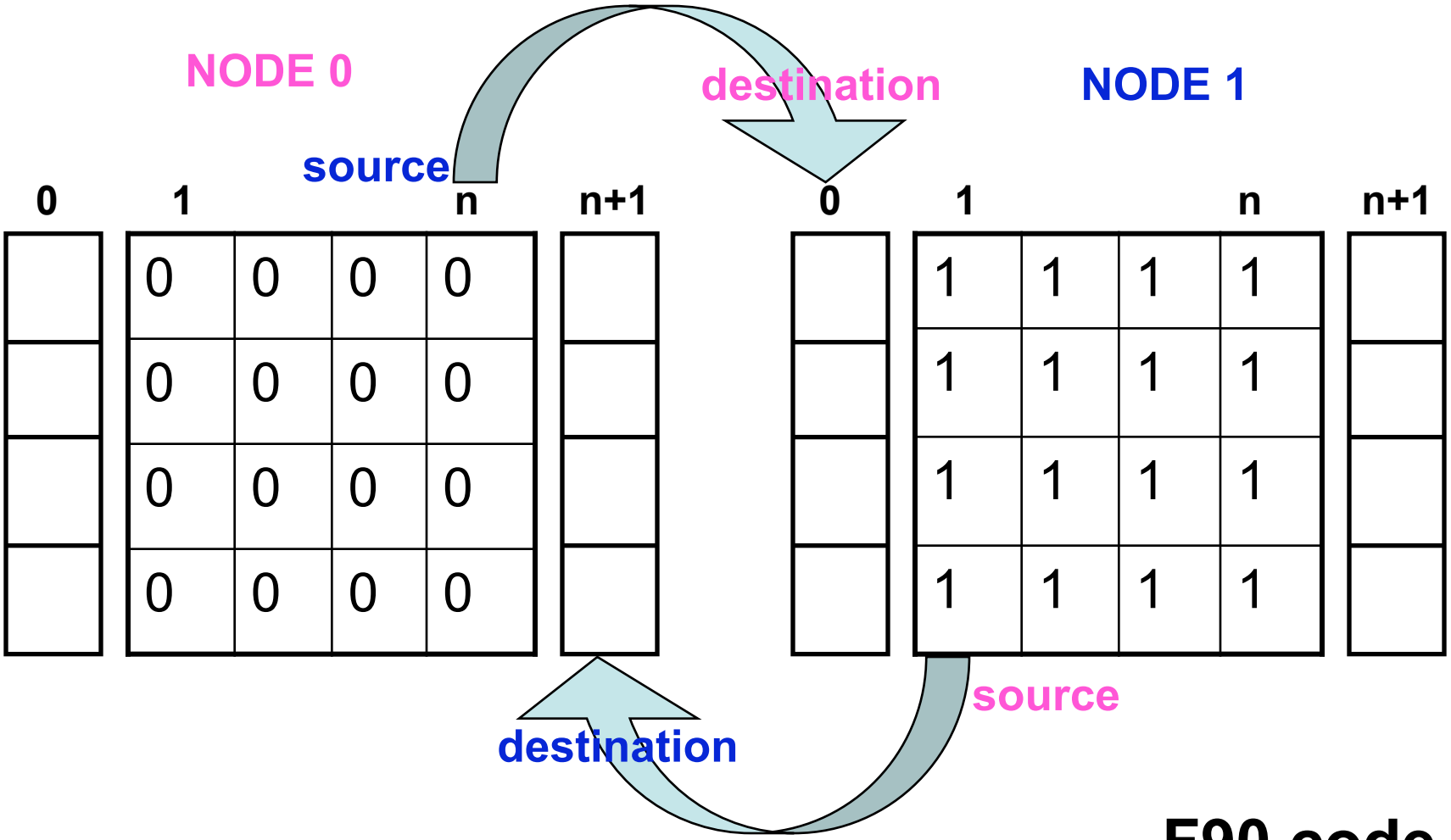
  #$ -pe 16way 16 → change 16 to 32, 48, 64, etc.

- Submit job

  qsub job

- See ghost_1d.c (.f90) for finished parallel version.

Domain Decomposition
Node Exchange



F90 code

Domain Decomposition

- Include the usual MPI init & finalize statements:

```
define ierr, irank, nranks at 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, into the 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 previous slide. Use &A(1,n) and &A(1,0) for C code.

- Send left data columns to left, into the 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 previous slide. Use &A(1,1), &A(1,n+1) &status for C.

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

Domain Decomposition

jacobi_update modifications

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

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

→

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

```
for(i=1; i<=n; i++){
  for(j=1; j<=n; j++){
    A(i,j) = (double) (iter);
  } }
```

no change

```
for(i=1; i<=n; i++){
  for(j=1; j<=n; j++){
    A(i,j) = (double) (iter);
  } }
```

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

```
A(1:N, 1:N) = iter;
```

no change

```
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)
```