

# MPI Lab

- **Parallelization** (Calculating  $\pi$  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
```

# Part 1: Calculating $\pi$ – Basic Course of Action

- Objective: parallelize serial  $\pi$  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 partial sum:  
for  $x_i, x_{i+N}, x_{i+2N}, x_{i+3N}, \dots$  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(mympi, pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0, MPI_COMM_WORLD,ierr)
```

# Calculating $\pi$ – 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...)
...
...MPI_Finalize(...)
```

Top of Code

Serial Code

End of Code

Don't forget:

(declare myid, numprocs and ierr 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 $\pi$ – 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 $\pi$ MPI-reduce partial sums, print

- 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  $\pi$  & calc. `pi`, from rank 0 proc (use if)
  - if (`myid == 0`) print...

# Calculating $\pi$

- 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.

If the number of cores is not a multiple of 16, then set the MY\_NSLOTS environment variable with the number of cores. E.g. “setenv MY\_NSLOTS 31”. But the last argument of the `-pe` option must be a multiple of 16, so as to determine the number of nodes.

- create a file called “input” and include the total number of elements (n) on the first line

`echo 2000 >input`

- Submit job

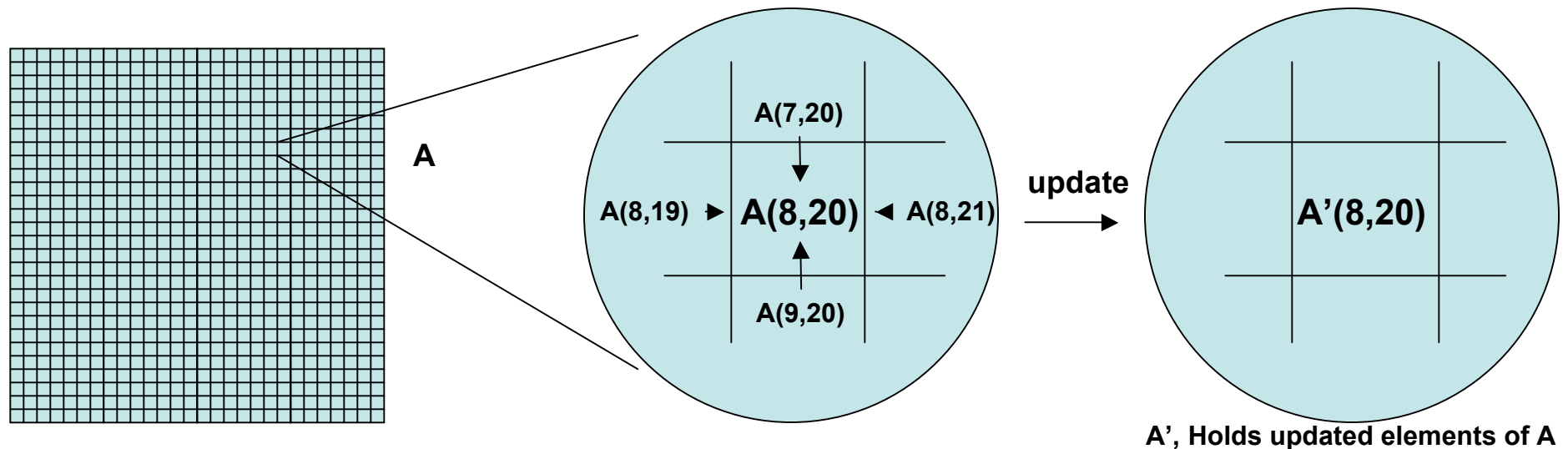
`qsub job`

- See `parallel_pi.c` (.f90) for finished parallel versions.

## Part 2: Sharing Data Across Processors -- Overview

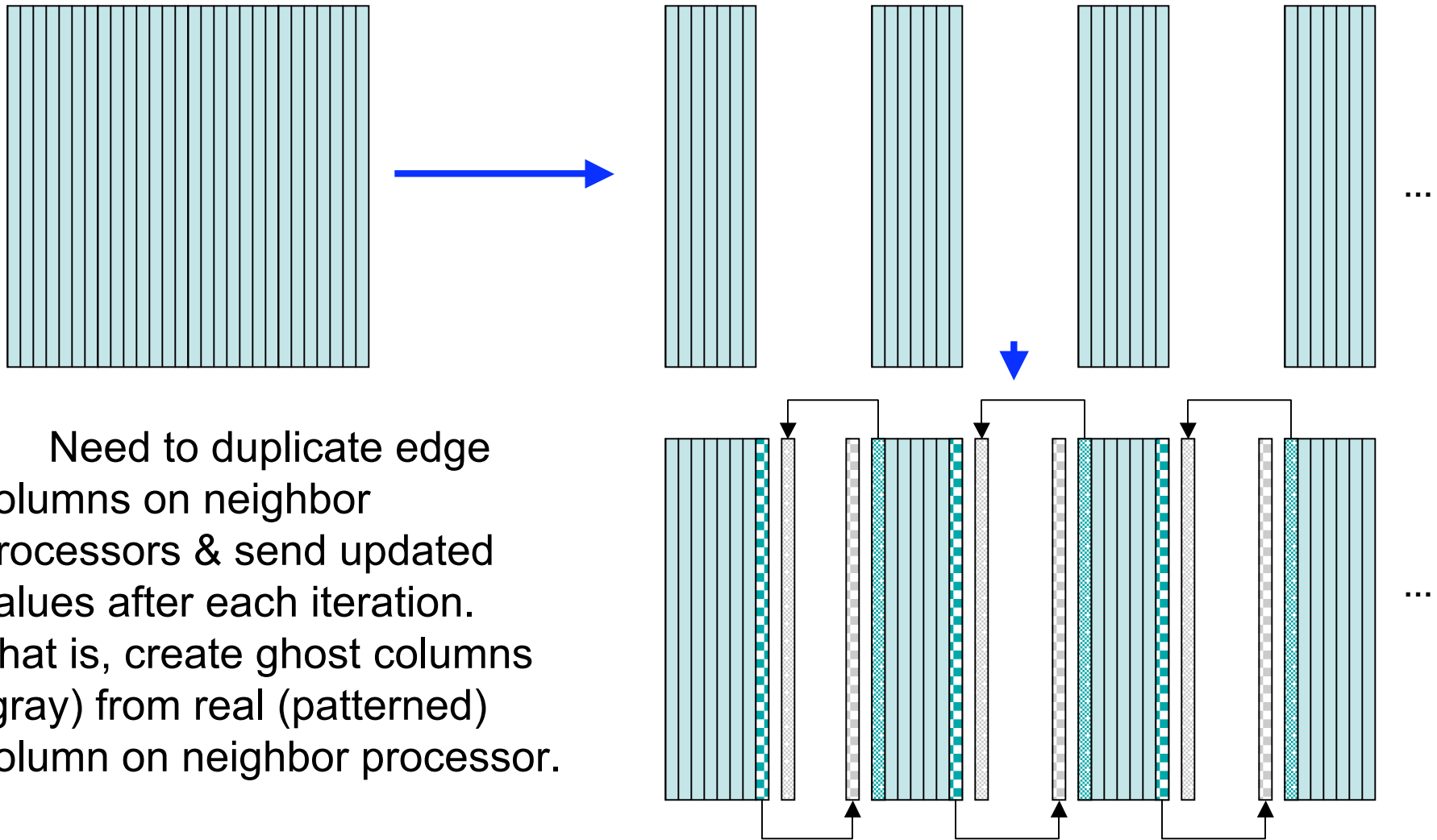
Solve 2-D partial differential equation (finite difference)

- 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:  $A(N,N) \rightarrow A(N,N+2)$ .

## 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

```
main program
  matrix A

loop
  jacobi_update(A)
end loop

end main
jacobi_update
```

→ parallel code (myghost)

```
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

|                  |   | columns |   |    |            |
|------------------|---|---------|---|----|------------|
|                  |   | 1       | 2 | 3  | 4 <b>j</b> |
| rows<br><b>i</b> | 1 | 1       | 5 | 9  | 13         |
|                  | 2 | 2       | 6 | 10 | 14         |
|                  | 3 | 3       | 7 | 11 | 15         |
|                  | 4 | 4       | 8 | 12 | 16         |

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

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

**A(i,j)...**

**Fortran**

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

```
#define A(i,j) a( (i-1) + (j-1)*n )
```

```
double a[ n*n ];
```

**A(i,j)...**

**C**

# Domain Decomposition

## Matrix Layout with Ghost Cells

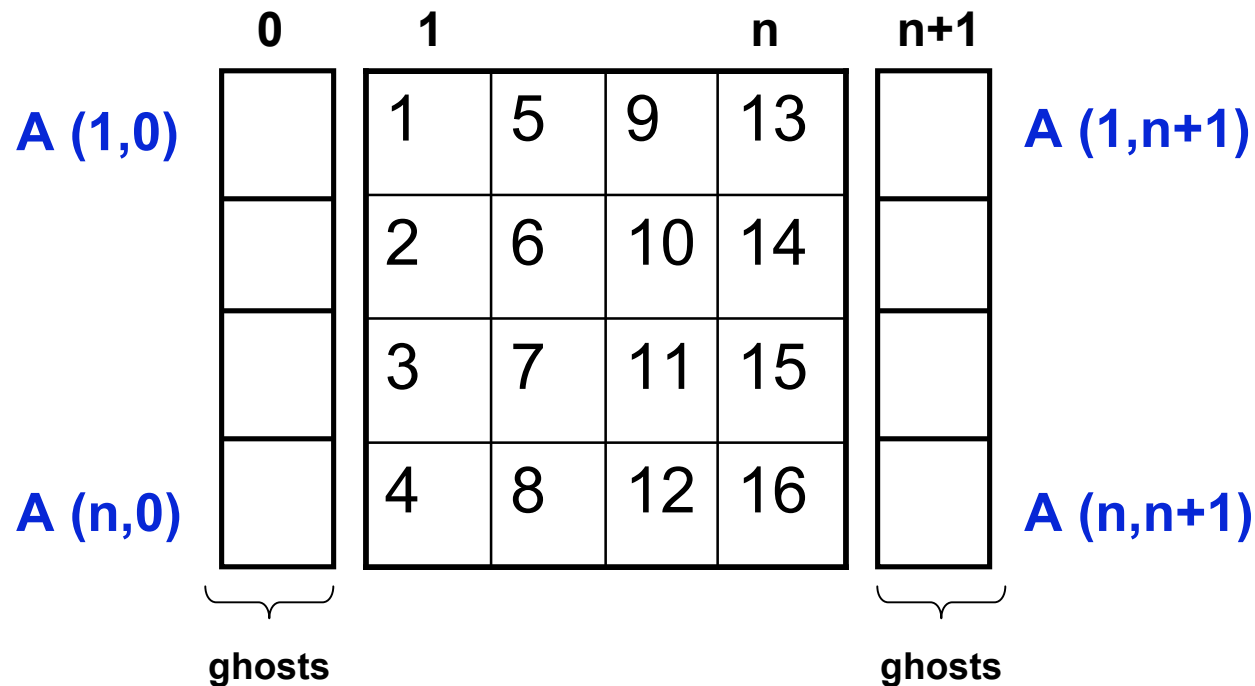
Redefine Array for easy ghost access

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

Fortran

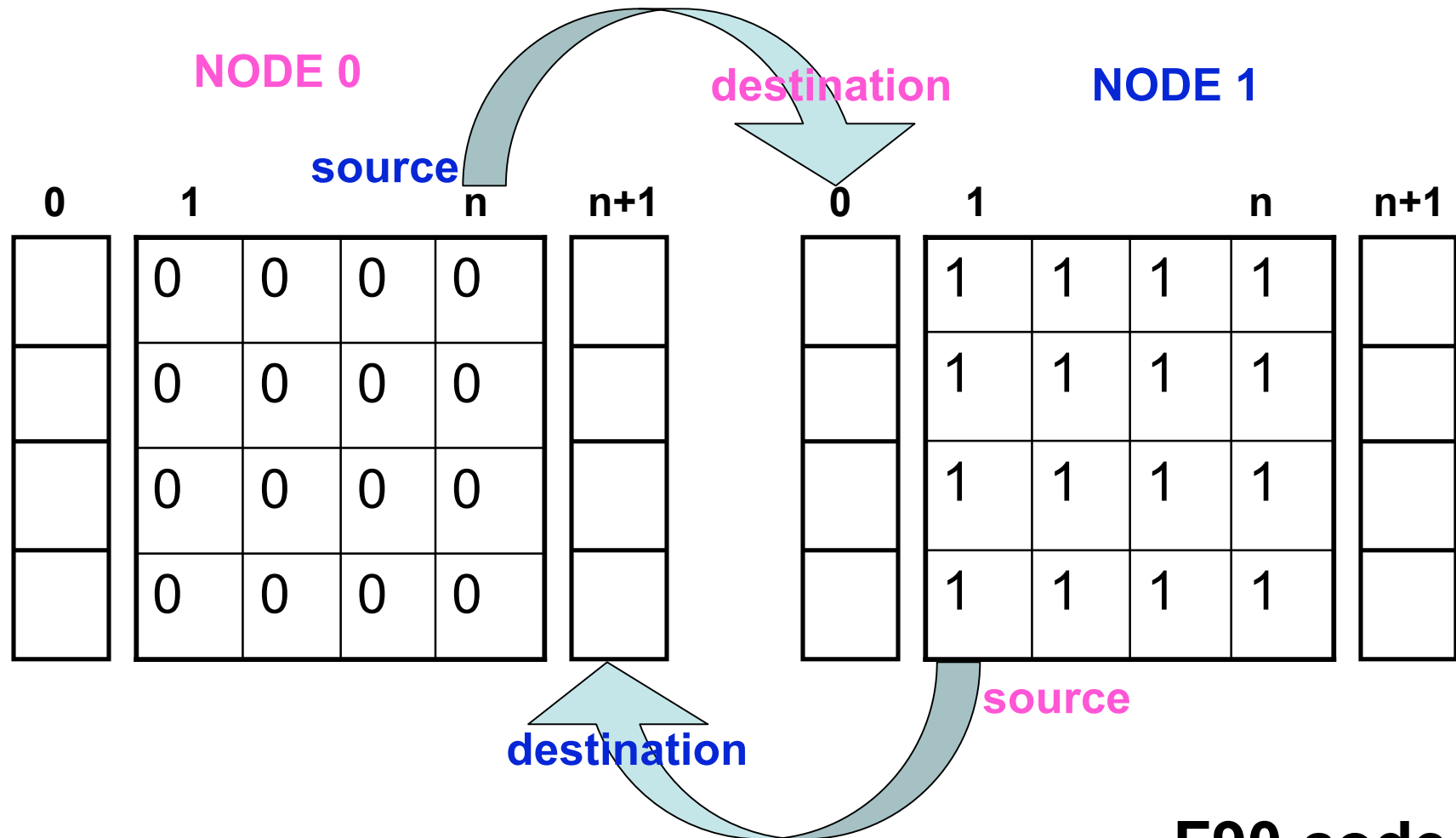
```
#define A(i,j) a( (i-1) + (j)*n )  
double a[n*(n+2)];
```

C



# Domain Decomposition

## 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:

|                                                                                                                                                                                  |                      |                                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#define A(i,j) a( (i-1) + (j-1)*n )<br/>double a[N*N];<br/><br/>for(i=1; i&lt;=n; i++){<br/>  for(j=1; j&lt;=n; j++){<br/>    A(i,j) = (double) (iter);<br/>  }<br/>}</pre> | $\longrightarrow$    | <pre>#define A(i,j) a( (i-1) + (j)*n )<br/>double a[N*(N+2)];<br/><br/>for(i=1; i&lt;=n; i++){<br/>  for(j=1; j&lt;=n; j++){<br/>    A(i,j) = (double) (iter);<br/>  }<br/>}</pre> |
|                                                                                                                                                                                  | <u>no<br/>change</u> |                                                                                                                                                                                    |

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

|                                |                      |                                |
|--------------------------------|----------------------|--------------------------------|
| <pre>A(1:N, 1:N) = iter;</pre> | <u>no<br/>change</u> | <pre>A(1:N, 1:N) = iter;</pre> |
|--------------------------------|----------------------|--------------------------------|

Because new indexing in declaration accommodates ghost vectors:

|                                  |                   |                                  |
|----------------------------------|-------------------|----------------------------------|
| <pre>real*8 :: A(1:n, 1:n)</pre> | $\longrightarrow$ | <pre>real*8 :: A(n, 0:n+1)</pre> |
|----------------------------------|-------------------|----------------------------------|

# Domain Decomposition

- Compile code:

`mpif90 -O3 myghost.f90`

`mpicc -O3 myghost.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.

If the number of cores is not a multiple of 16, then set the `MY_NSLOTS` environment variable with the number of cores. E.g. “`setenv MY_NSLOTS 31`”. But the last argument of the `-pe` option must be a multiple of 16, so as to determine the number of nodes.

- Submit job

`qsub job`

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