ALBUQUERQUE
High Performance Computing Center

The University of New Mexico

MPI Workshop - III

Research Staff

Cartesian Topologies in MPI

and

Passing Structures in MPI

Week 3 of 3

Schedule

- Course Map
- Fix environments to run MPI codes
- CartesianTopology
  - MPI_Cart_create
  - MPI_ Cart _rank, MPI_Cart_coords
  - MPI_ Cart _shift
  - MPI_Gatherv, MPI_Reduce, MPI_Sendrecv
- Passing Data Structures in MPI
  - MPI_Type_struct, MPI_Vector, MPI_Hvector

Course Map

	Week 1 Point to Point Basic Collective	Week 2 Collective Communications	Week 3 Advanced Topics
MPI functional routines	MPI_SEND (MPI_ISEND) MPI_RECV (MPI_IRECV) MPI_BCAST MPI_SCATTER MPI_GATHER	MPI_BCAST MPI_SCATTERV MPI_GATHERV MPI_REDUCE MPI_BARRIER	MPI_DATATYPE MPI_HVECTOR MPI_VECTOR MPI_STRUCT MPI_CART_CREATE
MPI Examples	Helloworld Swapmessage Vector Sum	Pi Matrix/vector multiplication Matrix/matrix mulplication	Poisson Equation Passing Structures/ common blocks Parallel topologies in MPI

Poisson Equation on a 2D Grid periodic boundary conditions

$$\nabla^2\phi(x,y) = -4\pi\rho(x,y)$$

$$\rho(x,y) = \frac{a}{\pi}\left(e^{-a[(x-L/4)^2+y^2]} - e^{-a[(x-3L/4)^2+y^2]}\right)$$

Serial Poisson Solver

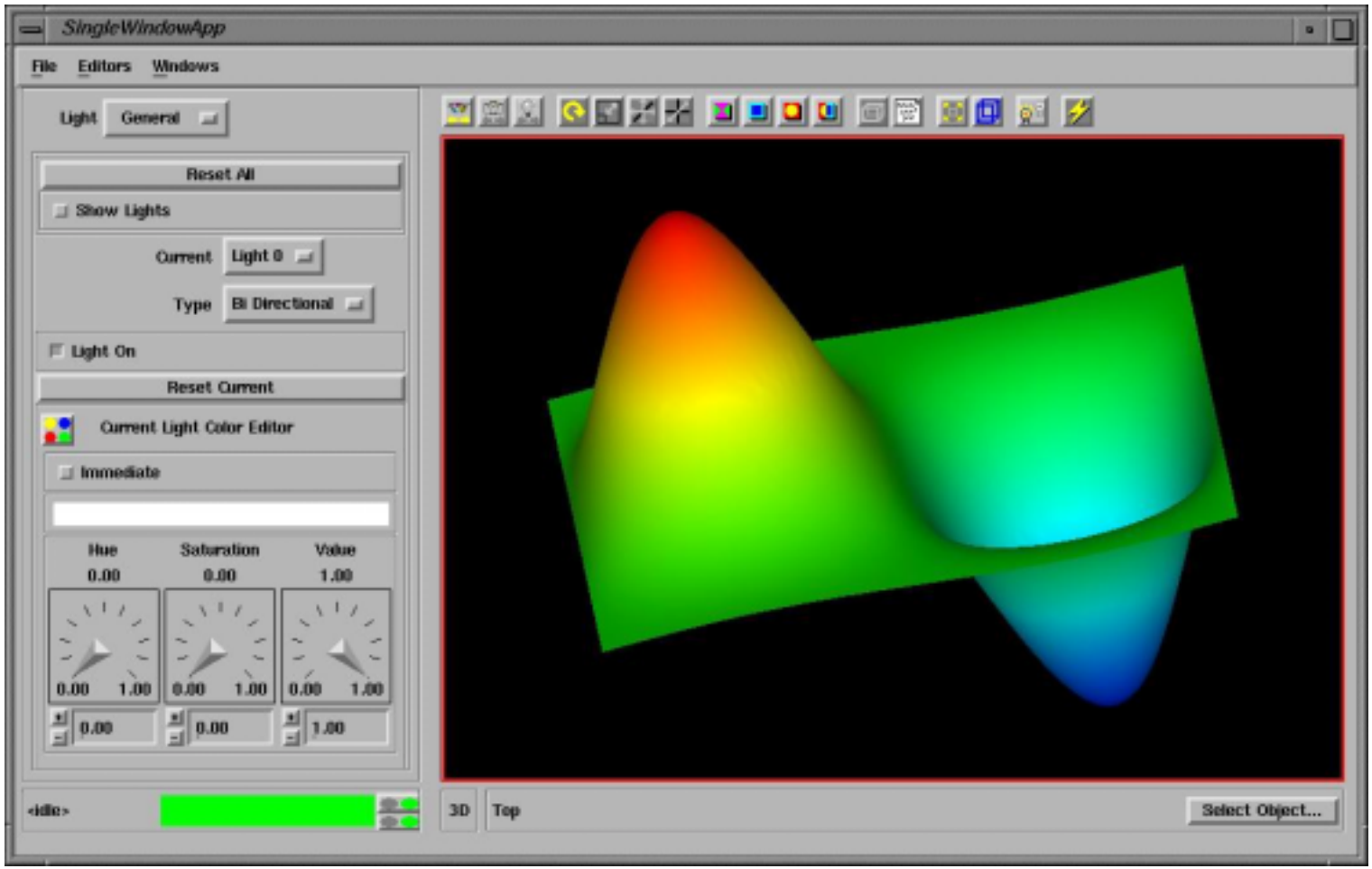
- **F90 Code**
  - **N^2 matrix for ρ and ϕ**
  - **Initialize ρ.**
  - **Discretize the equation**

$$\phi_{i,j} = \pi \Delta x^2 \rho_{i,j} + \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1})$$

  - **iterate until convergence**
  - **output results**

$$\sum_{i,j} |\phi_{i,j}^{new} - \phi_{i,j}^{old}| / \sum_{i,j} |\rho_{i,j}| < \varepsilon$$

Serial Poisson Solver: Solution



Serial Poisson Solver (cont)

```
do j=1, M
   do i = 1, N     !Fortran: access down columns first
      phi(i,j) = rho(i,j) +
```

$\phi_{i+1,j}$ `.25 * ( phi_old( modulo(i,N)+1, j )`

$\phi_{i-1,j}$ `+ phi_old( modulo(i-2,N)+1, j )`

$\phi_{i,j+1}$ `+ phi_old( i, modulo(j,N)+1 )`

$\phi_{i,j-1}$ `+ phi_old( i, modulo(j-2,N)+1 )`

```
                )
   enddo
enddo
```

Parallel Poisson Solver in MPI domain decomp. 3 x 5 processor grid

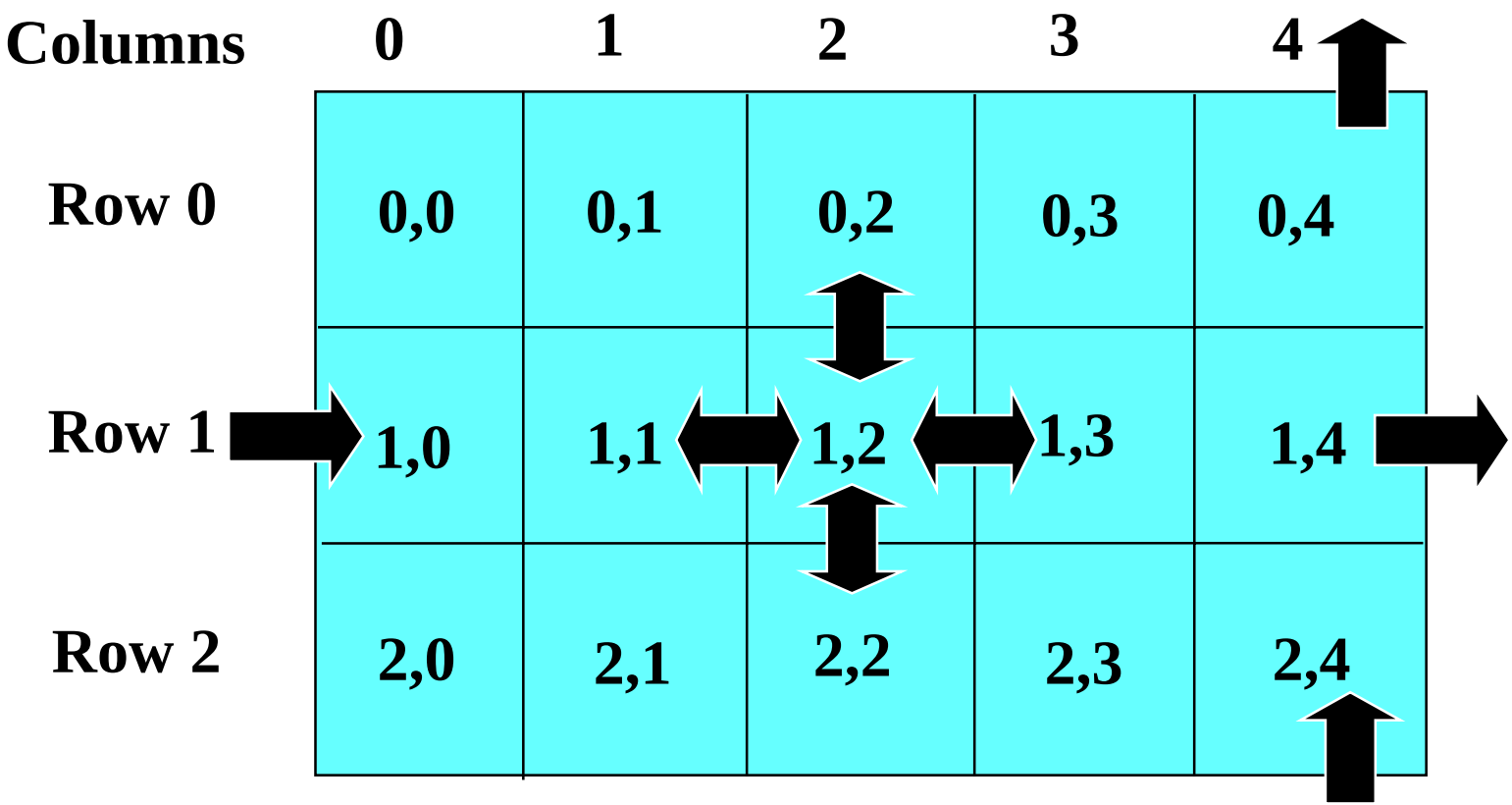
Columns	0	1	2	3	4
Row 0	0,0	0,1	0,2	0,3	0,4
Row 1	1,0	1,1	1,2	1,3	1,4
Row 2	2,0	2,1	2,2	2,3	2,4

Parallel Poisson Solver in MPI
Processor Grid: e.g. 3 x 5, N=M=64

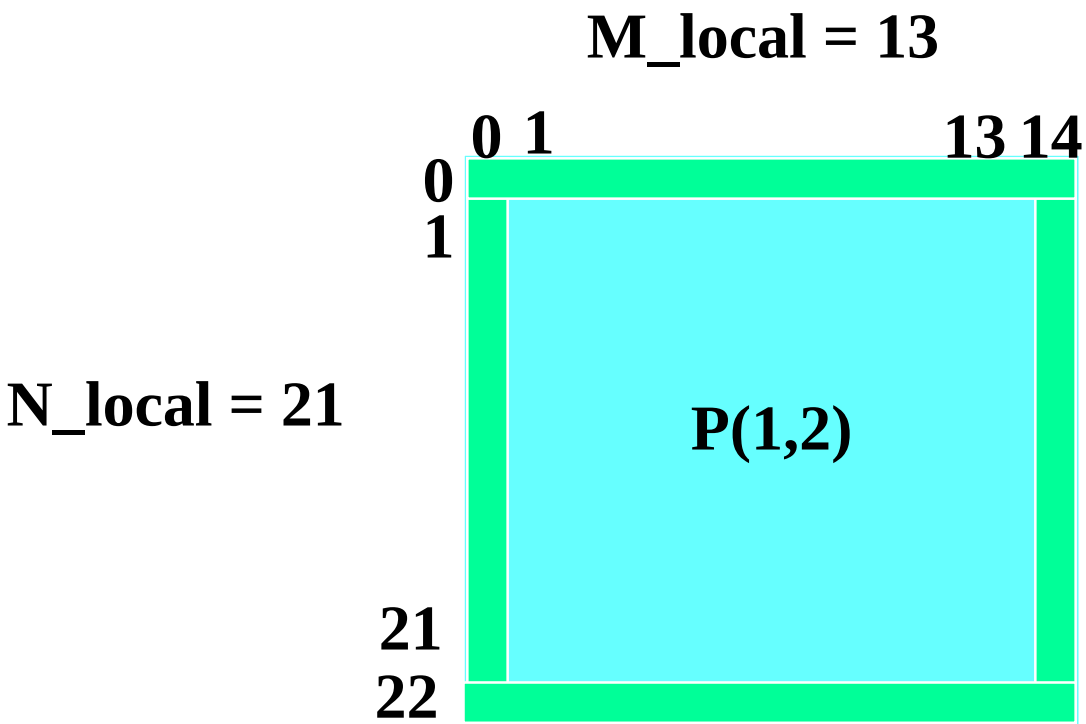
Columns	0	1	2	3	4
M: 1 – 13	14 – 26	27 – 39	40 – 52	53 – 64	
N: 1 – 22	23 – 43	44 – 64			
Row 0	0,0	0,1	0,2	0,3	0,4
Row 1	1,0	1,1	1,2	1,3	1,4
Row 2	2,0	2,1	2,2	2,3	2,4

Parallel Poisson Solver in MPI

Boundary data movement each iteration



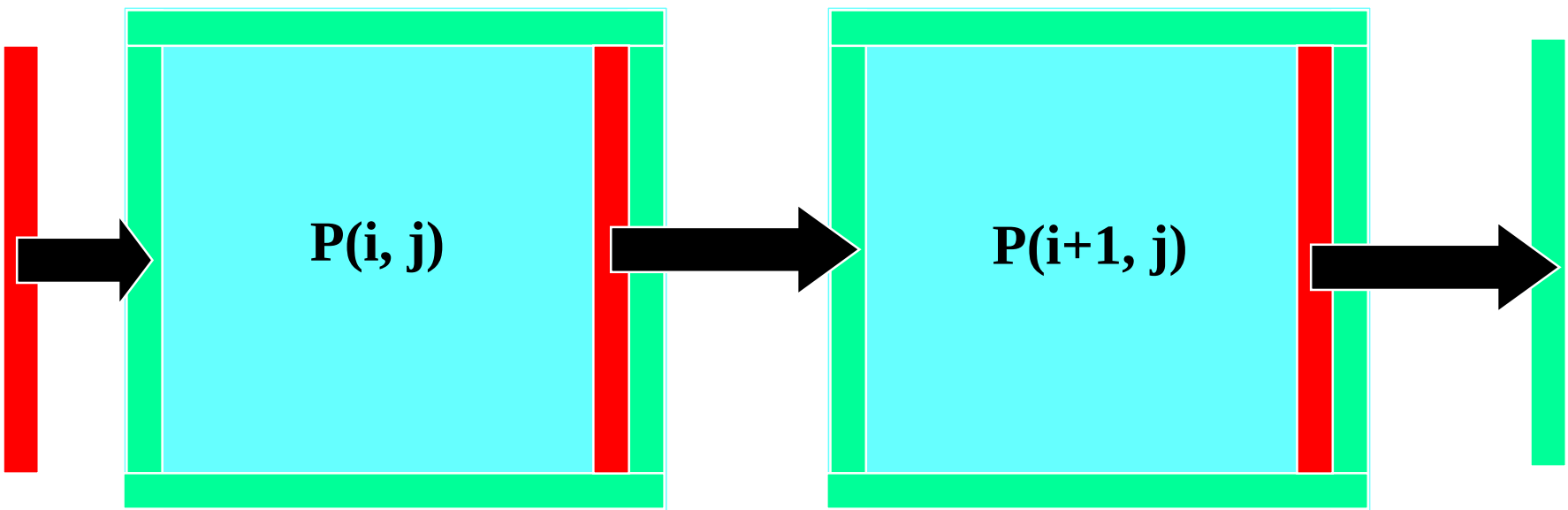
Ghost Cells: Local Indices



Data Movement
e.g. Shift Right, (East)

--> boundary data

--> ghost cells



Communicators and Topologies

- A *Communicator* is a set of processors which can talk to each other
- The basic communicator is *MPI_COMM_WORLD*
- One can create new groups or subgroups of processors from *MPI_COMM_WORLD* or other communicators
- MPI allows one to associate a Cartesian or Graph topology with a communicator

MPI Cartesian Topology Functions

- **MPI_CART_CREATE(old_comm, nmbr_of_dims, dim_sizes(), wrap_around(), reorder, cart_comm, ierr)**
  - **old_comm = MPI_COMM_WORLD**
  - **nmbr_of_dims = 2**
  - **dim_sizes() = (np_rows, np_cols) = (3, 5)**
  - **wrap_around = (.true., .true.)**
  - **reorder = .false. (generally set to .true.)**
    - **allows system to reorder the procr #s for better performance**
  - **cart_comm = grid_comm (name for new communicator)**

MPI Cartesian Topology Functions

- **MPI_CART_RANK(comm, coords(), rank, ierr)**
  - **comm = grid_comm**
  - **coords() = (coords(1), coords(2)), e.g. (0,2) for P(0,2)**
  - **rank = processor rank inside grid_com**
  - **returns the rank of the procr with coordinates coords()**
- **MPI_CART_COORDS(comm, rank, nmbr_of_dims, coords(), ierr)**
  - **nmbr_of_dims = 2**
  - **returns the coordinates of the procr in grid_comm given its rank in grid_comm**

MPI Cartesian Topology Functions

- **MPI_CART_SUB(grid_comm, free_coords(), sub_comm, ierr)**
  - **grid_comm = communicator with a topology**
  - **free_coords() : (.false., .true.) -> (i fixed, j varies), i.e. row communicator**
  - **: (.true., .false.) -> (i varies, j fixed), i.e. column communicator**
  - **sub_comm = the new sub communicator (say row_comm or col_comm)**

MPI Cartesian Topology Functions

- **MPI_CART_SHIFT(grid_comm, direction, displ, rank_recv_from, rank_send_to, ierr)**
  - **grid_comm = communicator with a topology**
  - **direction : = 0 → i varies, → column shift; N or S**
  - **: = 1 → j varies, → row shift ; E or W**
  - **disp : how many procrs to shift over (+ or -)**
  - **e.g. N shift: direction=0, disp= -1,**
  - **S shift: direction=0, disp= +1**
  - **E shift: direction=1, disp= +1**
  - **W shift: direction=1, disp= -1**

MPI Cartesian Topology Functions

- **MPI_CART_SHIFT(grid_comm, direction, displ, rank_recv_from, rank_send_to, ierr)**
  - **MPI_CART_SHIFT *does not* actually perform any data transfer. It returns two ranks.**
  - **rank_recv_from: the rank of the procr from which the calling procr will receive the new data**
  - **rank_send_to: the rank of the procr to which data will be sent from the calling procr**
  - **Note: MPI_CART_SHIFT does the modulo arithmetic if the corresponding dimensions has wrap_around(*) = .true.**

Parallel Poisson Solver N=upward shift in colms


```
!N or upward shift
!P(i+1,j) →(recv from)→ P(i,j) →(send to)→ P(i-1,j)
direction =  0 !i varies
disp      = -1 !i → i-1
top_bottom_buffer = phi_old_local(1,:)
call MPI_CART_SHIFT( grid_comm, direction, disp,
                     rank_recv_from, rank_send_to, ierr)
call MPI_SENDRECV( top_bottom_buffer, M_local+1,
                MPI_DOUBLE_PRECISION, rank_send_to, tag,
                     bottom_ghost_cells, M_local,
                MPI_DOUBLE_PRECISION, rank_recv_from, tag,
                     grid_comm, status, ierr)
phi_old_local(N_local+1,:) = bottom_ghost_cells
```

Parallel Poisson Solver
Main computation

```
do j=1, M_local
   do i = 1, N_local
      phi(i,j) = rho(i,j) +
                    .25 * (  phi_old( i+1, j )
                           + phi_old( i-1, j )
                           + phi_old( i, j+1 )
                           + phi_old( i, j-1 )
                          )
   enddo
enddo
```

$\phi_{i+1,j}$

$\phi_{i-1,j}$

$\phi_{i,j+1}$

$\phi_{i,j-1}$

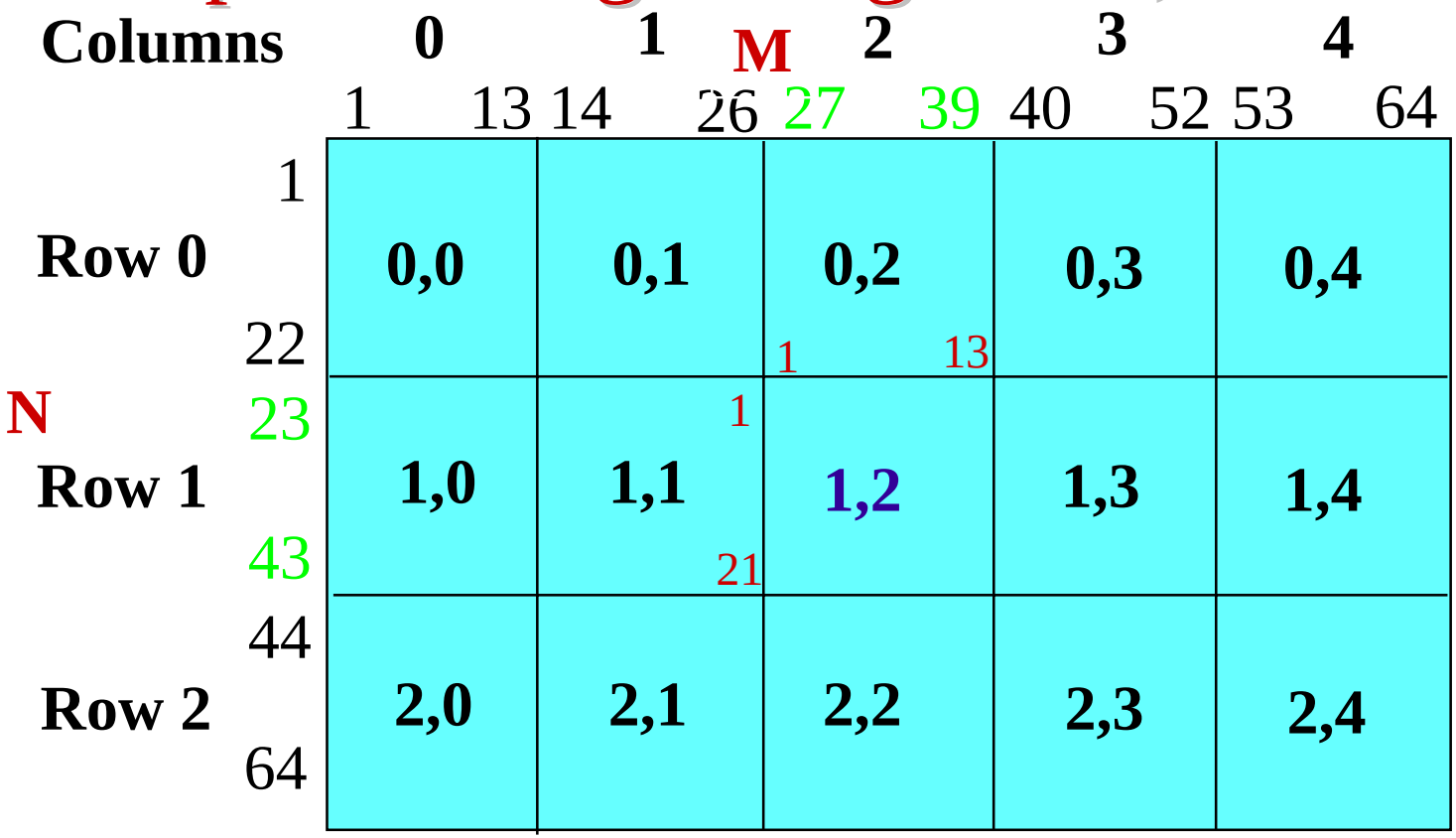
Note: indices are all within range now due to ghost cells

Parallel Poisson Solver: Global vs. Local Indices

```
 i_offset=0;
 do i = 1, coord(1)
    i_offset = i_offset + nmbr_local_rows(i)
enddo
j_offset=0;
do j = 1, coord(2)
   j_offset = j_offset + nmbr_local_cols(j)
enddo
 do j = j_offset+1, j_offset + M_local  !global indices
    y = (real(j)-.5)/M*Ly-Ly/2
    do i = i_offset+1, i_offset+1 + N_local !global "
       x = (real(i)-.5)/N*Lx
       makerho_local(i-i_offset,j-j_offset) = f(x,y)
     enddo
 enddo
```

!store with local indices

Parallel Poisson Solver in MPI
processor grid: e.g. 3 x 5, N=M=64

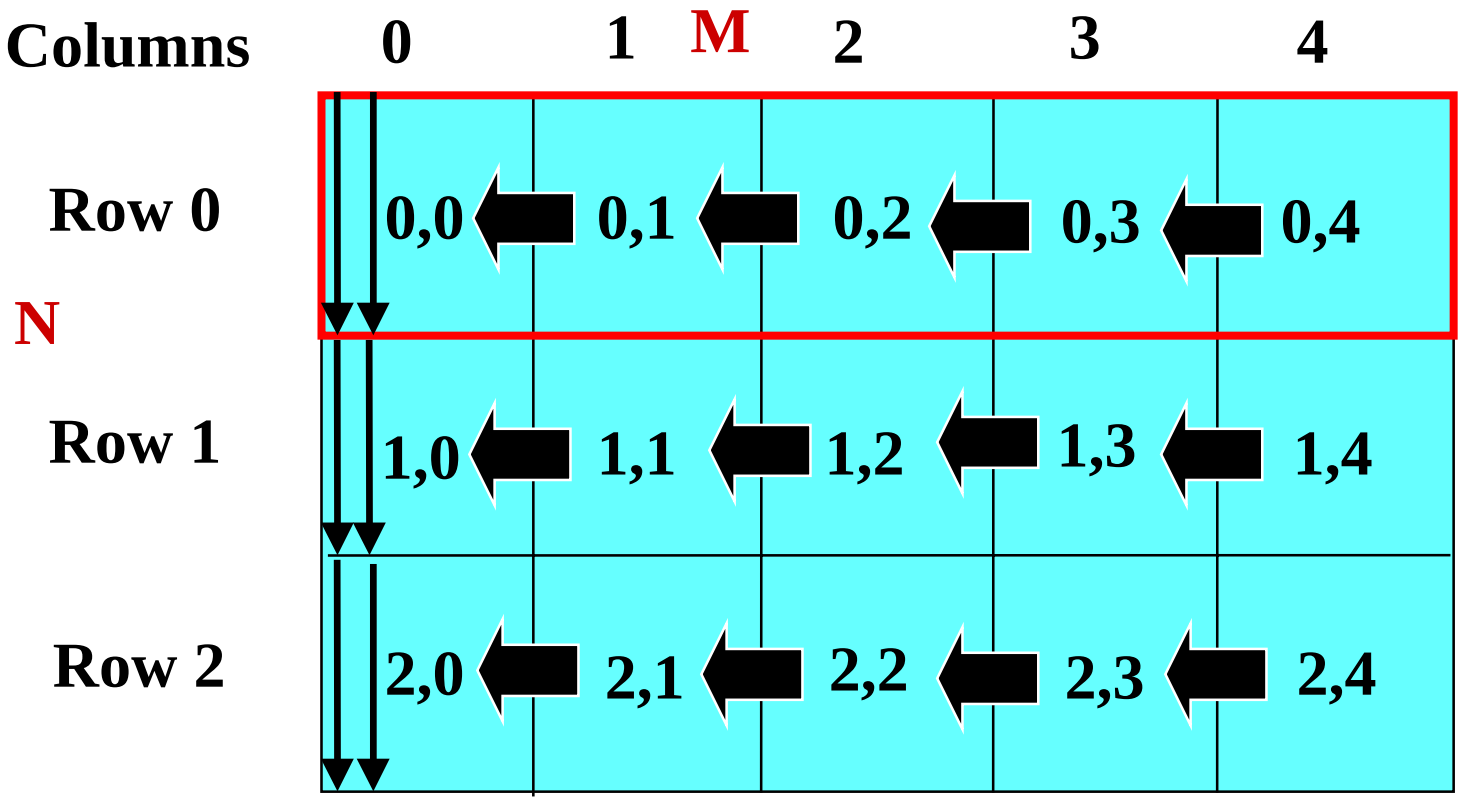


MPI Reduction & Communication Functions

- **Point-to-point communications in N,S,E,W shifts**
  - **MPI_SENDRECV(sendbuf, sendcount,sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source,recvtag, comm, status ierr)**
- **Reduction operations in the computation**
  - **MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, operation, ierr)**
  - **operation = MPI_SUM, MPI_MAX, MPI_MIN_LOC, ...**

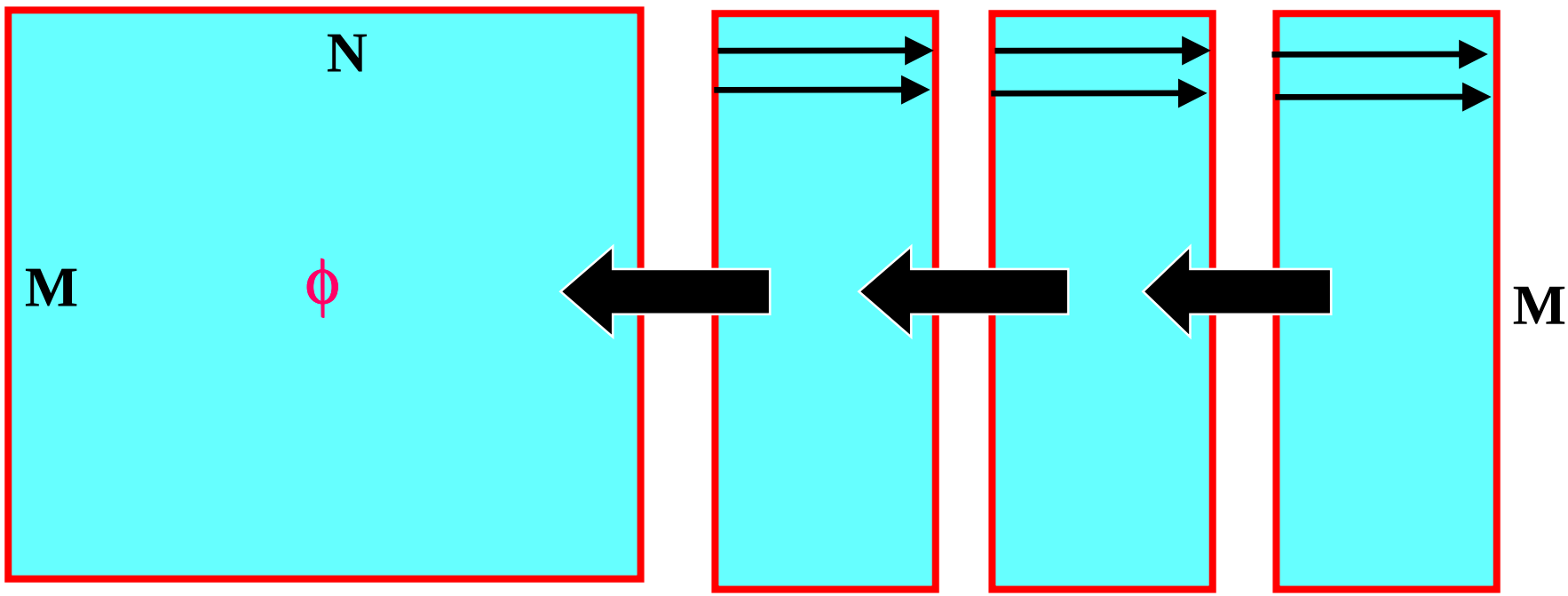
I/O of final results

Step 1: in row_comm, Gatherv the columns into matrices of size (# rows) x M



I/O of final results

Step 2: transpose the matrix;
in row_comm, Gatherv the columns in M x N matrix.
Result is the Transpose of the matrix for ϕ.



ALBUQUERQUE
High Performance Computing Center

The University of New Mexico

MPI Workshop - III

Research Staff

Passing Structures in MPI

Week 3 of 3

Topic

- **Passing Data Structures in MPI: C version**
  - **MPI_Type_struct**
  - **also see more general features MPI_PACK, MPI_UNPACK**
  - **See the code MPI_Type_struct_annotated.c**
    - **in** `/nfs/workshops/mpi/advanced_mpi_examples`

A structure in C

```
#define IDIM   2
#define FDIM   3
#define CHDIM 10
struct buf2{
     int     Int[IDIM];
     float   Float[FDIM];
     char    Char[CHDIM];
     float   x, y, z;
     int     i,j,k,l,m,n;
} sendbuf, recvbuf;
```

Create a structure datatype in MPI

```
struct buf2{
      int     Int[IDIM];
      float   Float[FDIM];
      char    Char[CHDIM];
      float   x,y,z;
      int     i,j,k,l,m,n;
} sendbuf, recvbuf;

MPI_Datatype type[NBLOCKS];  /* #define NBLOCKS 5 */
int       blocklen[NBLOCKS];

type=[MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_FLOAT,  MPI_INT];
blocklen=[IDIM,FDIM,CHDIM,3,6];
```

Create a structure datatype in MPI (cont)

```
struct buf2{
      int     Int[IDIM];
      float   Float[FDIM];
      char    Char[CHDIM];
      float   x,y,z;
      int     i,j,k,l,m,n;
} sendbuf, recvbuf;
MPI_Aint disp;    /* to align on 4-byte boundaries */
MPI_Address(&sendbuf.Int[0],disp);
MPI_Address(&sendbuf.Float[0],disp+1);
MPI_Address(&sendbuf.Char[0],disp+2);
MPI_Address(&sendbuf.x,disp+3);
MPI_Address(&sendbuf.i,disp+4);
```

Create a structure datatype in MPI (cont)

```
int base;
MPI_Datatype NewType; /* new MPI type we're creating */
MPI_Status status;

base = disp[0];          /* byte alignment */
for(i=0;i<NBLOCKS;i++) disp[i] -= base;

/* MPI calls to create new structure datatype.
   Since this is the 90's we need more than to just want
   this new datatype, we must commit to it.
*/
MPI_Type_struct(NBLOCKS, blocklen, disp, type, &NewType);
MPI_Type_commit(&NewType);
```

Use of the new structure datatype in MPI

```
/* Right Shift:
    rank-1 mod(NP)→ rank → rank+1 mod(NP)
*/

MPI_Comm_size(MPI_COMM_WORLD, &NP);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

sendto      = (rank+1)%NP;
recvfrom    = (rank-1+NP)%NP;

MPI_Sendrecv(&sendbuf, 1, NewType, sendto,   tag,
             &recvbuf, 1, NewType, recvfrom, tag,
                          MPI_COMM_WORLD, &status);
```

References: Some useful books

- ***MPI: The complete reference***
  - **Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongara, MIT Press**
  - `/nfs/workshops/mpi/docs/mpi_complete_reference.ps.Z`
- ***Parallel Programming with MPI***
  - **Peter S. Pacheco, Morgan Kaufman Publishers, Inc**
- ***Using MPI: Portable Parallel Programing with Message Passing Interface***
  - **William Gropp, E. Lusk and A. Skjellum, MIT Press**