COMP 605: Introduction to Parallel Computing
Quiz 2:
Module 2: Message Passing Interface (MPI)

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 04/10/17
Posted: 03/16/17
Updated: 03/30/17

Table of Contents

Assignemtn

This assignment is an exercise in learning about MPI Halo exchange. You will modify an existing halo exchange program to dynamically allocate distributed data arrays and to use MPI Cartesian communicator groups to update/exchange data on the different cores.

- Use source code based on MPI Halo Exchange Tutorial & code: `http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/exchange/C/exchange/main.html`
- Source Code provided on tuckoo: */COMP605/q3/mpi-exch.c*

Approach 1: Modify the MPI Exchange program:

(a) Process command line args to set dimensions of an [*MxN*] matrix.

(b) Modify code so that the array dimension variables are dynamic

(c) Modify code to handle more than 4 processors, such that $Rem(M/size) == 0$, where $size = processors$.

(d) Dynamically allocate *xlocal*: a 2D Matrix of size [*MxN*].

(e) Add test code to check values of input variables and results.

(f) Print out small size arrays (e.g. $M <= 10$)

(g) Integrate the *mpi-cart-2d-col.c* example code into your *mpi-exchange.c* code:

- Use the1D *comm1D_col* communicator sub group to exchange data between the horizontal slabs (rows).
- Use MPI_Cart_shift() to get neighbors (replace manual calc of up/down nbrs)
- Use *comm1D_col* in the *MPI_Sendrecv* calls to exchange data between processors.

(h) Verify that your results are working correctly (data exchanged).

(i) Time all code

Approach 2: Modify *mpi-cart-2d-col.c*

(a) Process command line args to set the dimensions of the [*MxN*] matrix.

(b) Modify code to handle more than 4 processors, such that $Rem(M/size) == 0$, where size = processors.

(c) Add test code to check values of input variables and results.

(d) Print out small size arrays (e.g. $M <= 10$)

(e) Insert exchange code into *mpi-cart-2d-col.c*

(f) Dynamically allocate a *xlocal* 2D Matrix of size [*MxN*]

(g) Use the1D *comm1D_col* communicator sub group to exchange data between the horizontal slabs (rows).

(h) Verify that your results are working correctly (data exchanged).

(i) Time all code

General Test Case Instructions

- All tests should be run for the following conditions:
  - Number of PEs: $[4, 8, 16]$
  - 4 ProbSizes, $[MxN]$, such that
    - REM$(M/size) = 0$
    - Range spans approximately 10^1 to 10^7
  - Timing measurements of critical exchange blocks
  - When populating the test array, use the 2D communicator rank (my_cart_rank). This will result in array values that are different than in the original code, and that is OK. Examples of output from my code is provided after the $mpi - exch.c$ otput slides.
  - Verify that your results are working correctly (data exchanged):
  - Explain/show why the 1D communicator should be a column communicator and not a row communicator.

Writeup & Turn-in Instructions

Report should include:

- Measure T_{wall} for all cases
- Calculate halo exchange bandwidth.
- Tables and Plot(s) of data
- Additional items in General Instructions (below).
- All code must be run on the student cluster as batch jobs.
- Place code, data, and PDF copy of your report in:
  /home/605/accountname/quizes/q2
- Report should Include:
  - answers to key questions & comments.
  - printout of code
  - relevant results, tables, plots

MPI Halo Exchange References

References:

- Gropp Halo Exch Lecture:
  `http://wgropp.cs.illinois.edu/courses/cs598-s16/`
- MPI Halo Exch Tutorial & code:
  `http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/exchange/C/exchange/main.html`
- MPI Comm Example Code located on tuckoo:
  /COMP605/mpi.ex/comms/cart
- See lectures on Performance and MPI Communications.:
  - `http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf`
  - `http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Comms-Perf.pdf`
  - `http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/ParallelPerformance.pdf`

Hints & Comments

- Currently, the code only runs on 4 processors, and max dimension of 12. To make it more dynamic:
  - replace *maxn=12* with dynamic variables *M,N*, then redefine dimensions for array xlocal.
  - MPI_Comm_size gets the number of processors. replace all hard coded values to use *size*
  - process command line args for M and N, then allocate arrays
- *mpi-exch.c* adds ghost cells at the top/bottom of the local array. What type of decomposition is this and along what dimension is the exchange?
- simple exchange is between the processors:

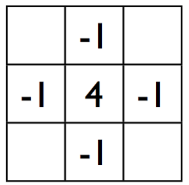
$$[P_{i-1}] \leftrightarrow [P_i] \leftrightarrow [P_{i+1}]$$

Using Cartesian systems

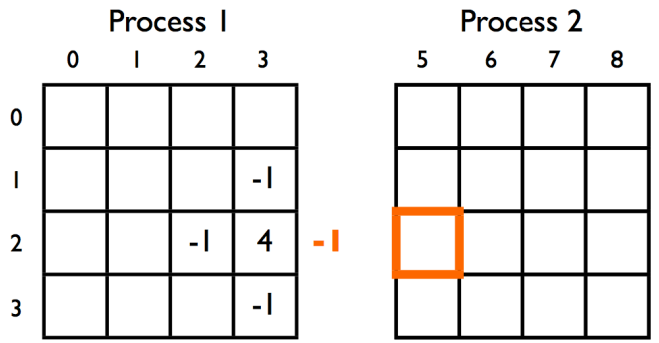
- In this project, you will create a 2D Cartesian mapping of your processors.
- The next slides show graphically what this decomposition looks like when it is overlaid onto your computational mesh.
- The process is to create a 2D communicator, and then create a subcommunicator group that will exchange data along a row of processors or a column.

1D MPI Halo Exchange

- Stencil computation in geometrically decomposed grids.
- The set of neighboring points that influence the calculations of a point is often called a *stencil*.
- If the computation needs data from another neighbor, then there must be an exchange of data.



(a) 5-Point Stencil

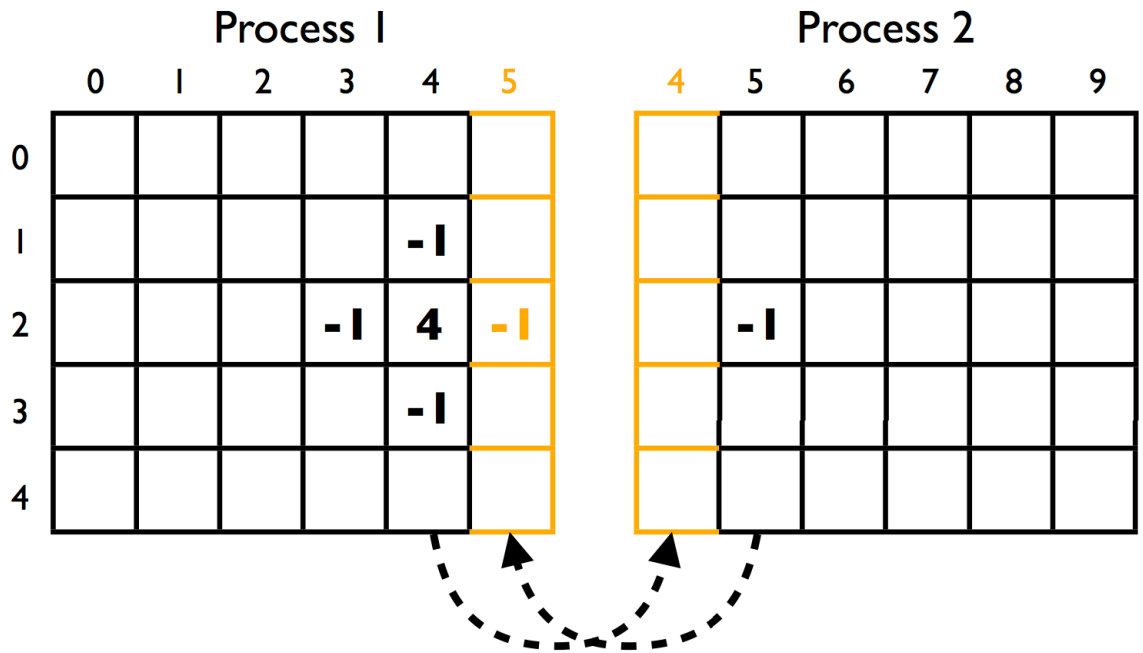


(b) Stencil that needs a cell from its neighbor

Source: F. B. Kjolstad and M. Snir, Ghost Cell Pattern, in Proceedings of the 2010 Workshop on Parallel Programming Patterns

1D MPI Halo Exchange

Ghost Cell Layout for Column Exchange



- Each core receives a chunk/vector of data from neighbor.
- Dark cells are inner compute cells; light/tan cells are ghost cells.
- Neighbor data is stored in the ghost cells.

Source: F. B. Kjolstad and M. Snir

Deadlock-freeI Halo Exchange

Alternate which cores do the exchange

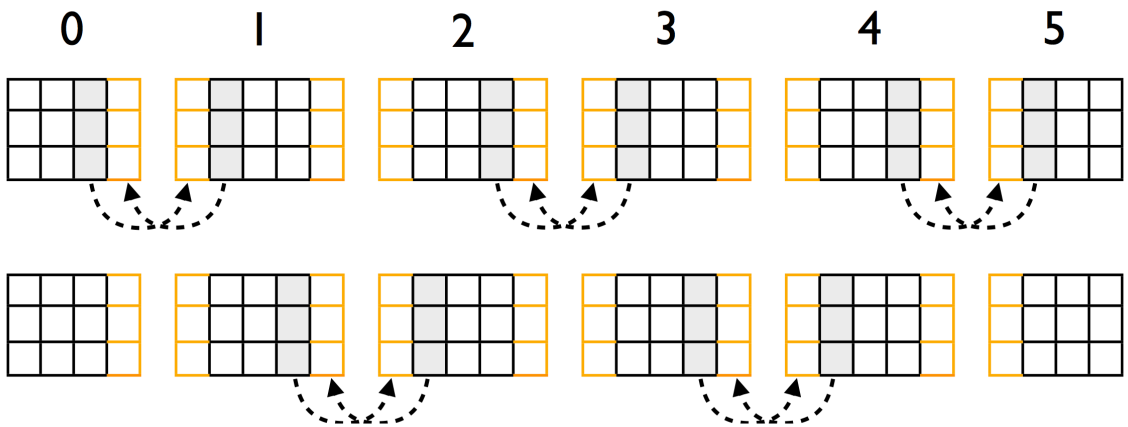


Figure 5: Deadlock-free border exchanges

```
void exchange_horizontal_borders () {
     if (x coord % 2 == 0) {
          exchange_east_border ();
          exchange_westUborder ();
     }
     else   {
          exchange west border ();
          exchange east border ();
     }
}
```

Source: F. B. Kjolstad and M. Snir, Ghost Cell Pattern, in Proceedings of the 2010 Workshop on Parallel Programming Patterns

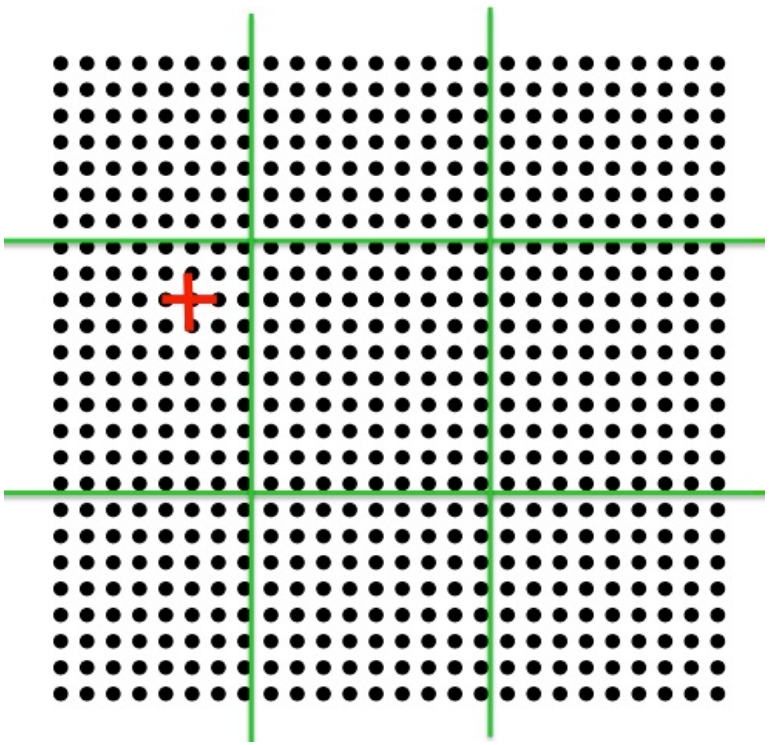
MPI_SendRecv(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

IN/OUT	Datatype	Desc
IN	sendbuf	initial address of send buffer (choice)
IN	sendcount	number of elements in send buffer (integer)
IN	sendtype	type of elements in send buffer (handle)
IN	dest	rank of destination (integer)
IN	sendtag	send tag (integer)
OUT	recvbuf	initial address of receive buffer (choice)
IN	recvcount	number of elements in receive buffer (integer)
IN	recvtype	type of elements in receive buffer (handle)
IN	source	rank of source (integer)
IN	recvtag	receive tag (integer)
IN	comm	communicator (handle)
OUT	status	status object (Status)

2D MPI Halo Exchange

Stencil computation in geometrically decomposed grids.

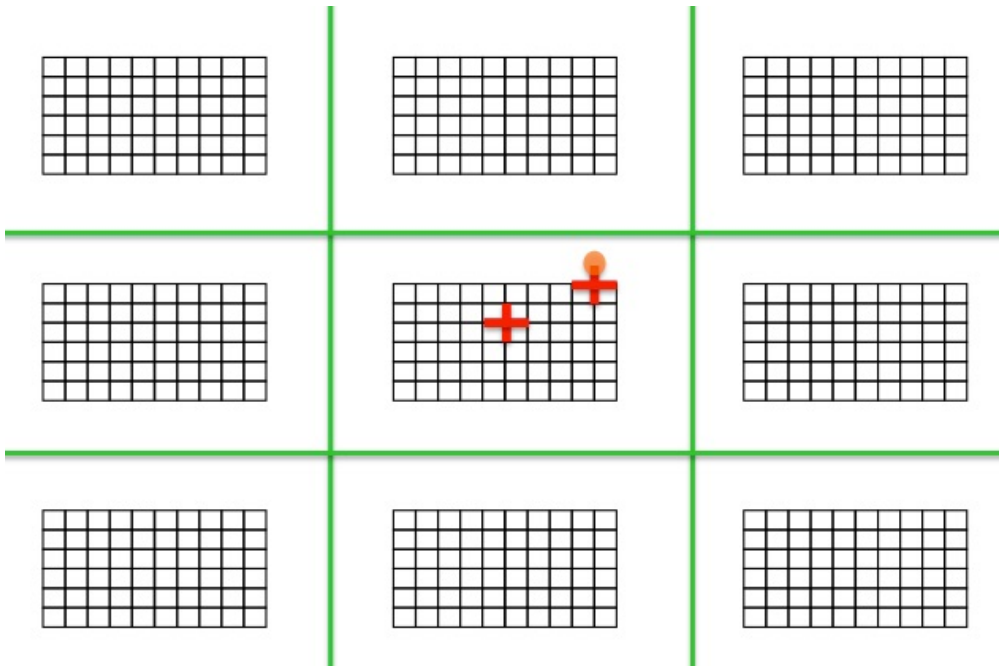
- Each circle is a mesh point in a 2D computation
- Each point involves the four neighbors
- The red plus is called the methods stencil
- Some computations may require neighbors data points.
- Data Distribution: decompose mesh into equal sized (work) pieces.



Source: Gropp Halo Exch Lecture: `http://wgropp.cs.illinois.edu/courses/cs598-s16/`

Stencil computation in geometrically decomposed grids: data transfers.

- Data Distribution: decompose mesh into equal sized (work) pieces.
- Distribute to workers/cores
- Track inner vs. boundary points
- Provide access to remote data through a halo exchange.



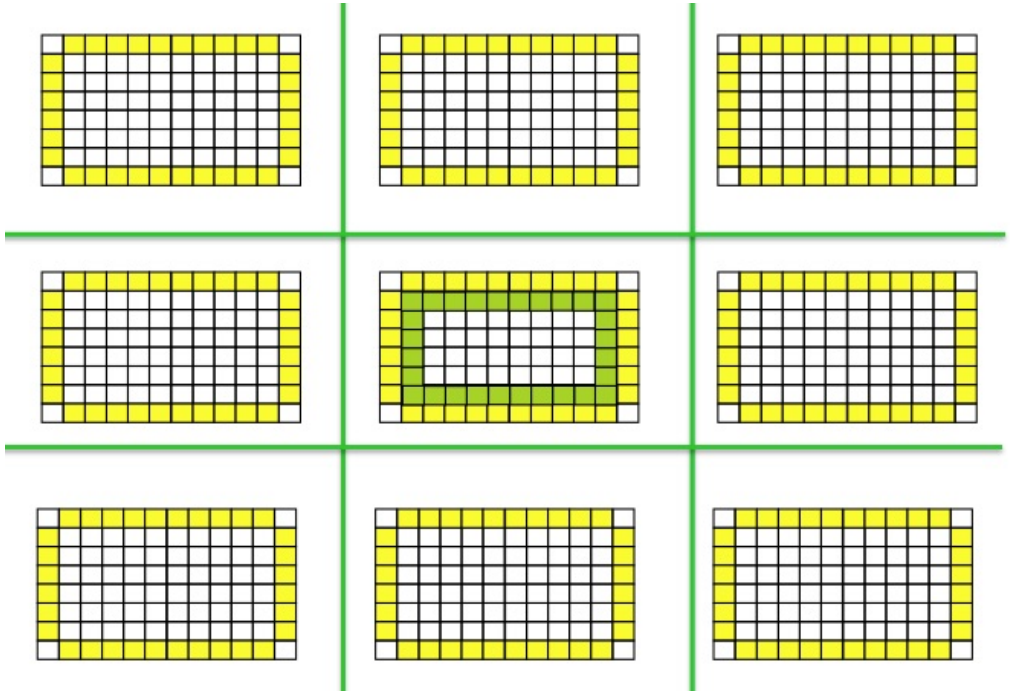


Diagram of inner (white), halo (yellow), exchange (green) cells.
You can exchange data using MPI_COMM_WORLD, or by creating 2D or 1D (row or col) communication groupts.

Source: Gropp Halo Exch Lecture: `http://wgropp.cs.illinois.edu/courses/cs598-s16/`

MPI Exchange Code Example

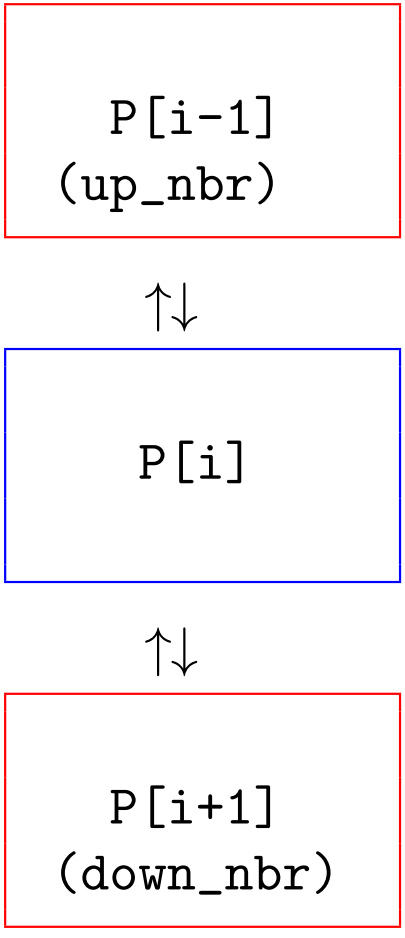
```
#include <stdio.h>
#include "mpi.h"
/*
 * MPI Halo Exchange tutorial:
 *     http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/exchange/C/exchange/main.html
 *
 * From MPI Tutorial:
 *      http://www.mcs.anl.gov/research/projects/mpi/tutorial/
 */
/* This example handles a 16 x 16 mesh, on 4 processors only. */
#define maxn 16

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value, size, errcnt, toterr, i, j;
    int up_nbr, down_nbr;
    MPI_Status status;
    double x[maxn][maxn];
    double xlocal[(maxn/4)+2][maxn];

    MPI_Init( &argc, &argv );

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

    if (size != 4) MPI_Abort( MPI_COMM_WORLD, 1 );
```

MPI Exchange Code Example

```
/* xlocal[][0] is lower ghostpoints, xlocal[][maxn+2] is upper */

/* Fill the data as specified */
 for (i=1; i<=maxn/size; i++)
     for (j=0; j<maxn; j++)
         xlocal[i][j] = rank;
 for (j=0; j<maxn; j++) {
     xlocal[0][j] = -1;
     xlocal[maxn/size+1][j] = -1;
 }

 for (i=0; i<maxn/size; i++) {
     printf("BE[%d]: I=[%d]:",rank, i);
     for (j=0; j<maxn; j++) {
        printf(" %4.0f ",xlocal[i][j]);
     }
     printf("\n");
 }
 printf("\n");
```

Deadlock Free Halo Exchange

```
// Processors 0 and 1 exchange, 2 and 3 exchange, etc.
//       Then 1 and 2 exchange, 3 and 4, etc.
//       The formula for this is
//                if (even) exchng up else down
//                if (odd)  exchng up else down
// Note the use of xlocal[i] for &xlocal[i][0] */
// Note that we use MPI_PROC_NULL to remove the if statements that
       would be needed without MPI_PROC_NULL */
up_nbr = rank + 1;
if (up_nbr >= size) up_nbr = MPI_PROC_NULL;
down_nbr = rank - 1;
if (down_nbr < 0) down_nbr = MPI_PROC_NULL;
if ((rank % 2) == 0) {    //  exchange up
        MPI_Sendrecv( xlocal[maxn/size], maxn, MPI_DOUBLE, up_nbr, 0,
                      xlocal[maxn/size+1], maxn, MPI_DOUBLE, up_nbr, 0,
                      MPI_COMM_WORLD, &status );
}
else {   // exchange down
        MPI_Sendrecv( xlocal[1], maxn, MPI_DOUBLE, down_nbr, 0,
                      xlocal[0], maxn, MPI_DOUBLE, down_nbr, 0,
                      MPI_COMM_WORLD, &status );
}

// Do the second set of exchanges
if ((rank % 2) == 1) {  // exchange up
        MPI_Sendrecv( xlocal[maxn/size], maxn, MPI_DOUBLE, up_nbr, 1,
                      xlocal[maxn/size+1], maxn, MPI_DOUBLE, up_nbr, 1,
                      MPI_COMM_WORLD, &status );
    }
else {   // exchange down
        MPI_Sendrecv( xlocal[1], maxn, MPI_DOUBLE, down_nbr, 1,
                      xlocal[0], maxn, MPI_DOUBLE, down_nbr, 1,
                      MPI_COMM_WORLD, &status );
}
```



MPI Exchange Code Example

```
    /* Check that we have the correct results */
    errcnt = 0;
    for (i=1; i<=maxn/size; i++)
        for (j=0; j<maxn; j++)
            if (xlocal[i][j] != rank) errcnt++;
    for (j=0; j<maxn; j++) {
        if (xlocal[0][j] != rank - 1) errcnt++;
        if (rank < size-1 && xlocal[maxn/size+1][j] != rank + 1) errcnt++;
    }

     for (i=0; i<maxn/size; i++) {
        printf("AE[%d]: I=[%d]:",rank, i);
        for (j=0; j<maxn; j++) {
           printf(" %4.0f ",xlocal[i][j]);
        }
        printf("\n");
    }
    MPI_Reduce( &errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
    if (rank == 0) {
        if (toterr)
            printf( "! found %d errors\n", toterr );
        else
            printf( "No errors\n" );
    }

    MPI_Finalize( );
    return 0;
}
```

Array data before the exchange between upper and lower neighbors. Array data is based on rank

```
BE[3]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[3]: I=[4]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[3]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[2]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[1]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
-------------------------------------------------------------------------------------------------
BE[2]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[2]: I=[4]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[3]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[2]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[1]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
-------------------------------------------------------------------------------------------------
BE[1]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[1]: I=[4]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
BE[1]: I=[3]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
BE[1]: I=[2]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
BE[1]: I=[1]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
BE[1]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
-------------------------------------------------------------------------------------------------
BE[0]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[0]: I=[4]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
BE[0]: I=[3]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
BE[0]: I=[2]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
BE[0]: I=[1]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
BE[0]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```

Array data after the exchange between upper and lower neighbors

```
AE[3]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
AE[3]: I=[4]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[3]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[2]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[1]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[0]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
-------------------------------------------------------------------------------------------------
AE[2]: I=[5]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[2]: I=[4]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[3]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[2]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[1]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[0]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
-------------------------------------------------------------------------------------------------
AE[1]: I=[5]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[1]: I=[4]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[1]: I=[3]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[1]: I=[2]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[1]: I=[1]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[1]: I=[0]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
-------------------------------------------------------------------------------------------------
AE[0]: I=[5]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[0]: I=[4]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
AE[0]: I=[3]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
AE[0]: I=[2]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
AE[0]: I=[1]:    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
AE[0]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```

```
BE[3]: I=[2]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[1]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
BE[3]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
-------------------------------------------------------------------------------------------------
BE[2]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[2]: I=[4]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[3]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[2]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[1]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
BE[2]: I=[0]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
-------------------------------------------------------------------------------------------------
BE[1]: I=[5]:   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
BE[1]: I=[4]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
BE[1]: I=[3]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
```

```
AE[3]: I=[2]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[1]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[3]: I=[0]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
-------------------------------------------------------------------------------------------------
AE[2]: I=[5]:    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
AE[2]: I=[4]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[3]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[2]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[1]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[2]: I=[0]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
-------------------------------------------------------------------------------------------------
AE[1]: I=[5]:    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
AE[1]: I=[4]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
AE[1]: I=[3]:    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
```

2D matrix allocation for MPI Communications

1. Init MPI env
2. Init Cart communicators (cartComm,rowComm,colComm)
3. On ROOT: Process command line args, set up key vars, bcast to child nodes.
4. Allocate Global memory for matrices needed on ROOT
5. Allocate Local memory for matrices needed on nodes
6. Distribute data from ROOT to other nodes
7. Execute functions
8. Gather distributed results for analysis/storage/printing

Init MPI env for Cartesian topology

```
MPI_Comm rowComm;
MPI_Comm colComm;
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
          periodic, 1, &gridComm);
MPI_Comm_rank(gridComm, &IMyRank);
MPI_Cart_coords(gridComm, IMyRank, ndims, IMyCoords);
MPI_Cart_rank(gridComm, IMyCoords, &IGridRank);

// Setup rowComm
int freeCoords[] = {0, 1};
MPI_Cart_sub(gridComm, freeCoords, &rowComm);
MPI_Comm_rank(colComm, &colRank);

// Setup column colComm
freeCoords[] = {1, 0};
MPI_Cart_sub(gridComm, freeCoords, &colComm);
MPI_Comm_rank(rowComm, &rowRank);
```

Code stubb for: matrix memory allocation

```
//set up rowSize & colSize
double localA[rowSize][colSize];
double localB[rowSize][colSize];
double localC[rowSize][colSize];
```

OR

```
double** localA = (double **)malloc(rowCnt * sizeof(double *));
for(i = 0; i < rowCnt; i++){
    localA[i] = (double *)malloc(colCnt * sizeof(double));
    for(j = 0; j < colCnt; j++){
        localA[i][j] = 0;
    }
}
```

Data Distribution to nodes

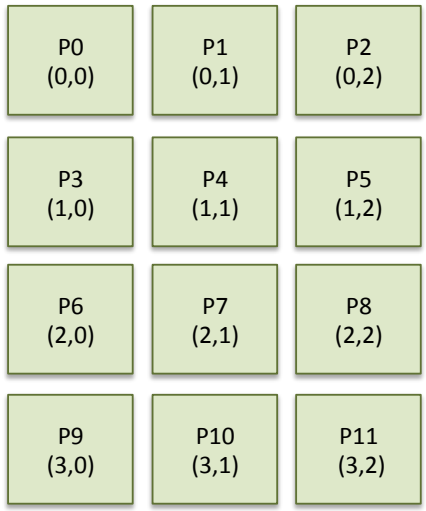
- Most students used linearized arrays to distribute distribute matrices by rows then by columns
- Algorithm

```
if MPI_COMM_WORLD rank == o
    linearize the matrix by rows into a vector
if rowComm rank = 0
    \\  the root of each rowComm calls this
    \\ to distribute the rows by chunks to nodes
    call MPI_Scatter(. . ., tmpVect, . . . )
    Reconvert tmpVector back to a 2D array

// all nodes:
for each row in my rowComm group
    call MPI_Scatter(. . ., tmpVect, . . . ) to scatter each row of data
              evenly across procs into the rowComm group
    Assign  localArray[row][*] = tmpVect
```

OUTPUT: *mpi-cart-2D.c*:

PE[4x3] topology (using default MPI PE arrangement) and creating the comm2D cartesian communicator.

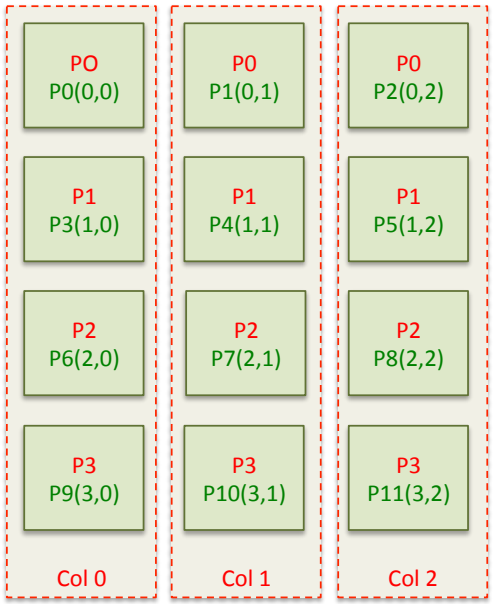
```
[comms]%  mpirun -np 12 ./mpi-cart-2d

PW[0]/[12]: PEdims = [4 x 3]

PW[0]:  my_cart_rank PCM[0],  my coords = (0,0)
PW[1]:  my_cart_rank PCM[1],  my coords = (0,1)
PW[2]:  my_cart_rank PCM[2],  my coords = (0,2)
-----------------------------------------------
PW[3]:  my_cart_rank PCM[3],  my coords = (1,0)
PW[4]:  my_cart_rank PCM[4],  my coords = (1,1)
PW[5]:  my_cart_rank PCM[5],  my coords = (1,2)
-----------------------------------------------
PW[6]:  my_cart_rank PCM[6],  my coords = (2,0)
PW[7]:  my_cart_rank PCM[7],  my coords = (2,1)
PW[8]:  my_cart_rank PCM[8],  my coords = (2,2)
-----------------------------------------------
PW[9]:  my_cart_rank PCM[9],  my coords = (3,0)
PW[10]: my_cart_rank PCM[10], my coords = (3,1)
PW[11]: my_cart_rank PCM[11], my coords = (3,2)
```



Source: MThomas, COMP 605 lectures:
http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf

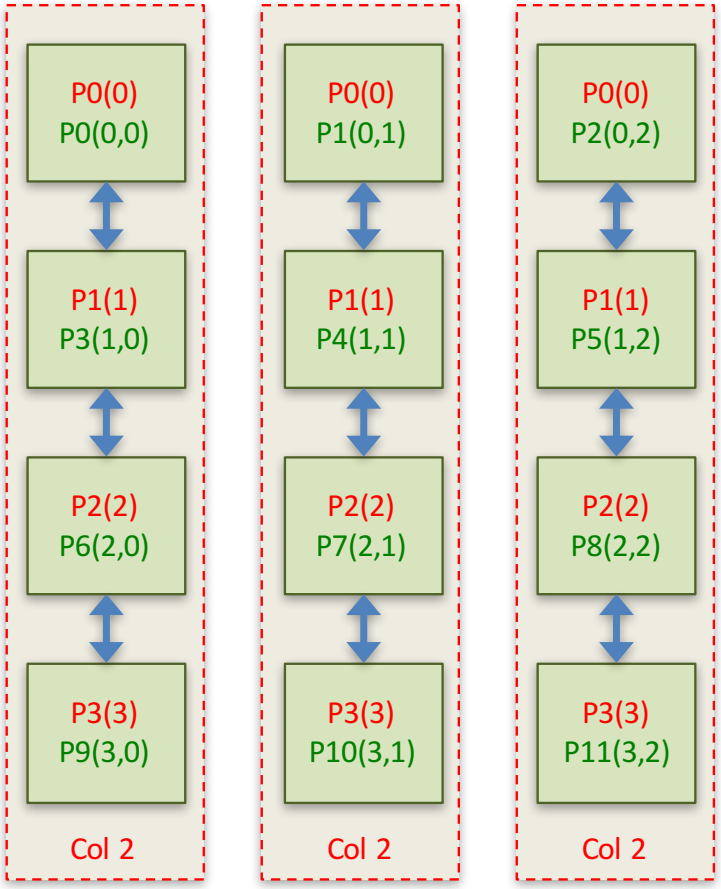
OUTPUT:: *mpi-cart-2D-col*: PE[4x3] topology, MPI defined PE dims. Processor arrangement obtained by using *MPI_Cart_sub* In this case, the PEs along a column will exchange data with upper/lower neighbors because they are in a group.

```
[comms]% % mpirun -np 12 ./mpi-cart-2D-col
Name of comm world is: MPI_COMM_WORLD
PW[0]/[12]: PEdims = [4 x 3]
Name of comm world is: comm2D
------------------------------------------------------------------------
PW[0], PWC(0,0)::ColComm::PC[0],CC(0),Root=0,Name=comm1D_col_0,CSum=18
PW[3], PWC(1,0)::ColComm::PC[1],CC(1),Root=0,Name=comm1D_col_0,CSum=18
PW[6], PWC(2,0)::ColComm::PC[2],CC(2),Root=0,Name=comm1D_col_0,CSum=18
PW[9], PWC(3,0)::ColComm::PC[3],CC(3),Root=0,Name=comm1D_col_0,CSum=18
------------------------------------------------------------------------
PW[1], PWC(0,1)::ColComm::PC[0],CC(0),Root=1,Name=comm1D_col_1,CSum=22
PW[4], PWC(1,1)::ColComm::PC[1],CC(1),Root=1,Name=comm1D_col_1,CSum=22
PW[7], PWC(2,1)::ColComm::PC[2],CC(2),Root=1,Name=comm1D_col_1,CSum=22
PW[10],PWC(3,1)::ColComm::PC[3],CC(3),Root=1,Name=comm1D_col_1,CSum=22
------------------------------------------------------------------------
PW[2], PWC(0,2)::ColComm::PC[0],CC(0),Root=2,Name=comm1D_col_2,CSum=26
PW[5], PWC(1,2)::ColComm::PC[1],CC(1),Root=2,Name=comm1D_col_2,CSum=26
PW[8], PWC(2,2)::ColComm::PC[2],CC(2),Root=2,Name=comm1D_col_2,CSum=26
PW[11],PWC(3,2)::ColComm::PC[3],CC(3),Root=2,Name=comm1D_col_2,CSum=26
------------------------------------------------------------------------
```



Source: MThomas, COMP 605 lectures: `http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf`

- Exchange pattern for the PE[4x3] topology, MPI defined PE dims.
- Processor arrangement obtained by using *MPI_Cart_sub* to create the 1D column groups.
- The figure shows the the ranks for each core:
  - colComm1D group (in red)
  - comm2D cartesian group (green)
  - coords in the 1D and 2D decomposition.
- In this case, the PEs along a column will exchange data with upper/lower neighbors because they are in a group.



Source: MThomas, COMP 605 lectures: `http://edoras.sdsu.edu/~mthomas/sp17.605/lectures/MPI-Cart-Comms-and-Topos.pdf`

Output from code before (BE) and after (AE) halo exchange for the column group "Col1"

```
----------------------------------------
BE[1]: I=[0]:   -1    -1    -1    -1
BE[1]: I=[1]:    1     1     1     1
BE[1]: I=[2]:    1     1     1     1
BE[1]: I=[3]:    1     1     1     1
BE[1]: I=[4]:   -1    -1    -1    -1
----------------------------------------
BE[4]: I=[0]:   -1    -1    -1    -1
BE[4]: I=[1]:    4     4     4     4
BE[4]: I=[2]:    4     4     4     4
BE[4]: I=[3]:    4     4     4     4
BE[4]: I=[4]:   -1    -1    -1    -1
----------------------------------------
BE[7]: I=[0]:   -1    -1    -1    -1
BE[7]: I=[1]:    7     7     7     7
BE[7]: I=[2]:    7     7     7     7
BE[7]: I=[3]:    7     7     7     7
BE[7]: I=[4]:   -1    -1    -1    -1
----------------------------------------
BE[10]: I=[0]:   -1    -1    -1    -1
BE[10]: I=[1]:   10    10    10    10
BE[10]: I=[2]:   10    10    10    10
BE[10]: I=[3]:   10    10    10    10
BE[10]: I=[4]:   -1    -1    -1    -1
----------------------------------------
```

```
----------------------------------------
AE[1]: I=[0]:   -1    -1    -1    -1
AE[1]: I=[1]:    1     1     1     1
AE[1]: I=[2]:    1     1     1     1
AE[1]: I=[3]:    1     1     1     1
AE[1]: I=[4]:    4     4     4     4
----------------------------------------
AE[4]: I=[0]:    1     1     1     1
AE[4]: I=[1]:    4     4     4     4
AE[4]: I=[2]:    4     4     4     4
AE[4]: I=[3]:    4     4     4     4
AE[4]: I=[4]:    7     7     7     7
----------------------------------------
AE[7]: I=[0]:    4     4     4     4
AE[7]: I=[1]:    7     7     7     7
AE[7]: I=[2]:    7     7     7     7
AE[7]: I=[3]:    7     7     7     7
AE[7]: I=[4]:   10    10    10    10
----------------------------------------
AE[10]: I=[0]:    7     7     7     7
AE[10]: I=[1]:   10    10    10    10
AE[10]: I=[2]:   10    10    10    10
AE[10]: I=[3]:   10    10    10    10
AE[10]: I=[4]:   -1    -1    -1    -1
----------------------------------------
```