CSCI 455: Lab #8 — Advanced MPI: Group/Communicator Management and Virtual Topologies

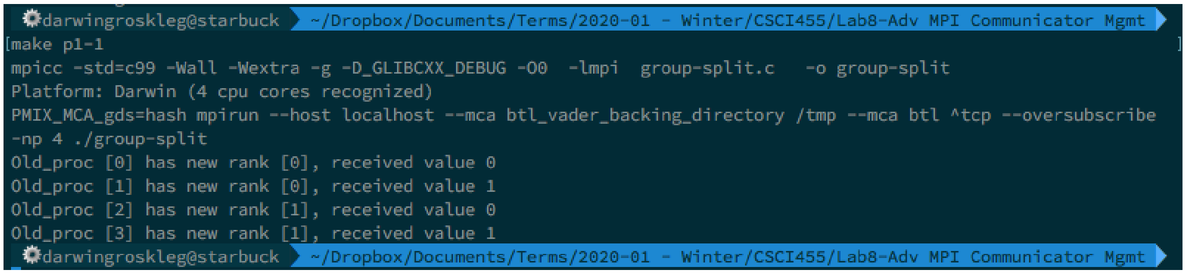
Darwin Jacob Groskleg

Winter 2020

Contents

Part 1: Group/Communicator Management

MPI Split Method



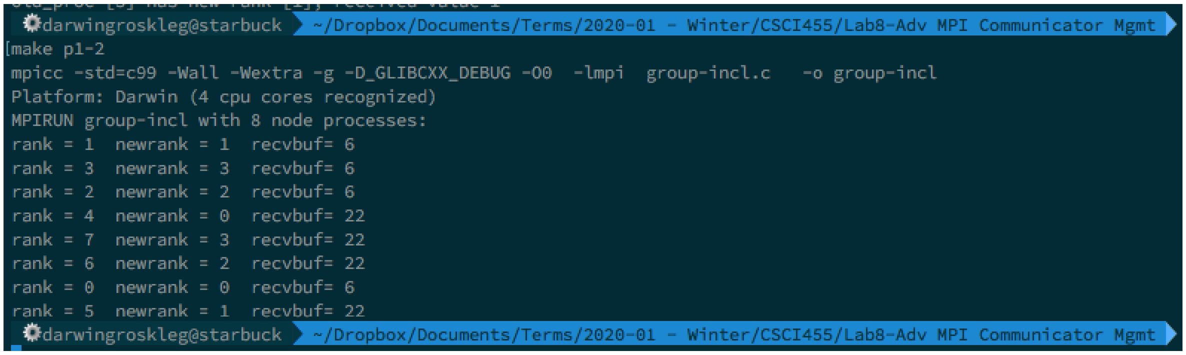
```
darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
[make p1-1                                                                                              ]
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  group-split.c   -o group-split
Platform: Darwin (4 cpu cores recognized)
PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversubscribe
-np 4 ./group-split
Old_proc [0] has new rank [0], received value 0
Old_proc [1] has new rank [0], received value 1
Old_proc [2] has new rank [1], received value 0
Old_proc [3] has new rank [1], received value 1
darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
```

Figure 1: Console output: group-split.c

group-split.c

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int numprocs;
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    /* Color of local process:
     *  decides to which communicator the process will
     *  belong after the split.
     */
    int color = world_rank % 2;

    // Communicator split is a remote opration?
    MPI_Comm New_Comm;
    MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &New_Comm);
    int new_rank;
    MPI_Comm_rank(New_Comm, &new_rank);
    int new_nodes;
    MPI_Comm_size(New_Comm, &new_nodes);

    int broad_val;
    if(new_rank == 0)
        broad_val = color;
    MPI_Bcast(&broad_val, 1, MPI_INT, 0, New_Comm);

    printf("Old_proc [%d] has new rank [%d], received value %d\n",
              world_rank,          new_rank,           broad_val);

    MPI_Comm_free(&New_Comm);
    MPI_Finalize();
    return 0;
}
```

MPI Include Method



```
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
[make p1-2
 mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  group-incl.c   -o group-incl
 Platform: Darwin (4 cpu cores recognized)
 MPIRUN group-incl with 8 node processes:
 rank = 1  newrank = 1  recvbuf= 6
 rank = 3  newrank = 3  recvbuf= 6
 rank = 2  newrank = 2  recvbuf= 6
 rank = 4  newrank = 0  recvbuf= 22
 rank = 7  newrank = 3  recvbuf= 22
 rank = 6  newrank = 2  recvbuf= 22
 rank = 0  newrank = 0  recvbuf= 6
 rank = 5  newrank = 1  recvbuf= 22
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
```

Figure 2: Console output: group-incl.c

group-incl.c

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define NPROCS 8

int main(int argc, char *argv[])  {
    MPI_Init(&argc,&argv);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int numtasks;
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

    if (numtasks != NPROCS) {
        printf("Must specify MP_PROCS= %d. Terminating.\n",NPROCS);
        MPI_Finalize();
        exit(0);
    }

    // extract the original group handle
    MPI_Group  orig_group;
    MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
    MPI_Group  new_group;

    //  divide tasks into two distinct groups based upon world_rank
    //  Group Contruction: happens locally.
    if (world_rank < NPROCS/2) {
        const int ranks1[4]={0,1,2,3};
        MPI_Group_incl(orig_group, 4, ranks1, &new_group);
    }
    else {
        const int ranks2[4]={4,5,6,7};
        MPI_Group_incl(orig_group, 4, ranks2, &new_group);
    }

    // create new new communicator and then perform collective communications
    // Communicator Construction: happens remotely/collectively.
    MPI_Comm   new_comm;
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);

    int sendbuf = world_rank;
    int recvbuf;
    MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, new_comm);

    // get rank in new group
    int new_rank;
    MPI_Group_rank(new_group, &new_rank);


    printf("rank = %d  newrank = %d  recvbuf= %d\n",
           world_rank,     new_rank,     recvbuf);


    MPI_Group_free(&orig_group);
    MPI_Group_free(&new_group);
```

```
    MPI_Comm_free(&new_comm);
    MPI_Finalize();
    return 0;
}
```

Part 2: Virtual Topologies

MPI Cartesian Map with 1 Dimension

```
darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
[make p2-1                                                                                                    ]
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  mpi-cart-1D-get-nbrs.c   -o mpi-cart-1D-get-nbrs
Platform: Darwin (4 cpu cores recognized)
PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversubscribe
-np 4 ./mpi-cart-1D-get-nbrs
PW[0]/[4]: NDims=1, PEdims = [4]
P[0]:     periodic: shift  1: src[3] P[0] dest[1]
P[0]:     periodic: shift  0: src[0] P[0] dest[0]
P[0]:     periodic: shift -1: src[1] P[0] dest[3]
P[1]:     periodic: shift  1: src[0] P[1] dest[2]
P[1]:     periodic: shift  0: src[1] P[1] dest[1]
P[1]:     periodic: shift -1: src[2] P[1] dest[0]
P[3]:     periodic: shift  1: src[2] P[3] dest[0]
P[3]:     periodic: shift  0: src[3] P[3] dest[3]
P[3]:     periodic: shift -1: src[0] P[3] dest[2]

Non-periodic next
P[2]:     periodic: shift  1: src[1] P[2] dest[3]
P[2]:     periodic: shift  0: src[2] P[2] dest[2]
P[2]:     periodic: shift -1: src[3] P[2] dest[1]
P[3]: non-periodic: shift  1: src[2] P[3] dest[-2]
P[3]: non-periodic: shift  0: src[3] P[3] dest[3]
P[3]: non-periodic: shift -1: src[-2] P[3] dest[2]
P[0]: non-periodic: shift  1: src[-2] P[0] dest[1]
P[0]: non-periodic: shift  0: src[0] P[0] dest[0]
P[0]: non-periodic: shift -1: src[1] P[0] dest[-2]
P[1]: non-periodic: shift  1: src[0] P[1] dest[2]
P[1]: non-periodic: shift  0: src[1] P[1] dest[1]
P[1]: non-periodic: shift -1: src[2] P[1] dest[0]
P[2]: non-periodic: shift  1: src[1] P[2] dest[3]
P[2]: non-periodic: shift  0: src[2] P[2] dest[2]
P[2]: non-periodic: shift -1: src[3] P[2] dest[1]
darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
```

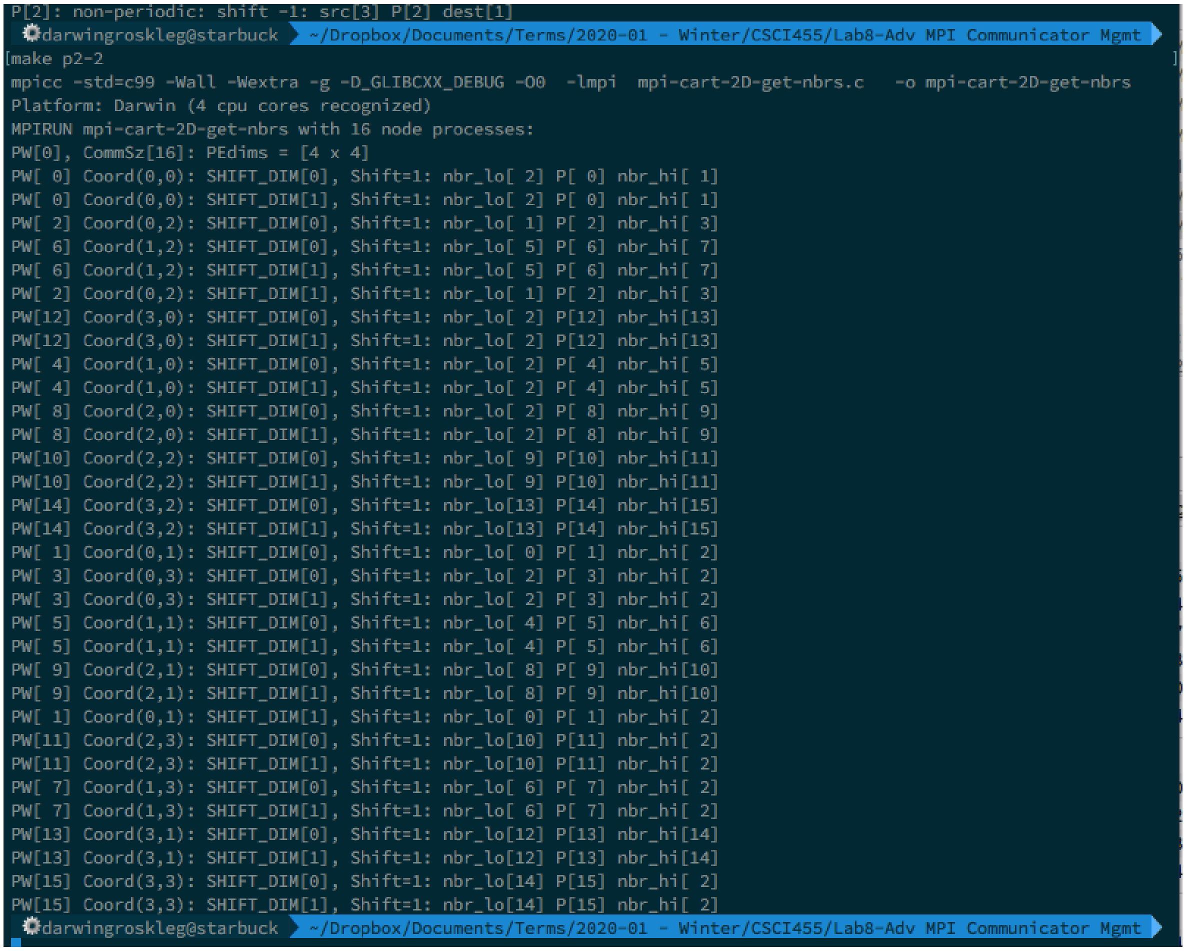
Figure 3: Console output: mpi-cart-1D-get-nbrs.c

mpi-cart-1D-get-nbrs.c

```c
/* mpi-cart-1D-get-nbrs.c
 * ----------------------
 * 1 Dimensional Cartesian Virtual Topology
 *  finds the neighbors in a cartesian communicator
 */
#include <mpi.h>
#include <stdlib.h>
#include <stdio.h>

typedef enum { false, true } bool;

int main( int argc, char *argv[] ) {
    MPI_Init( &argc, &argv );
    int cluster_size;
    MPI_Comm_size( MPI_COMM_WORLD, &cluster_size );
    int rank;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    const int ndims = 1;
    int dims[ndims];
    /* processor dimensions */
    dims[0] = cluster_size;
    /* create Cartesian topology for processes */
    //      nnodes, ndims, dims[]
    MPI_Dims_create(cluster_size, ndims, dims);

    if(rank == 0)
        printf("PW[%d]/[%d]: NDims=%d, PEdims = [%d]\n",
                rank, cluster_size, ndims,  dims[0]);

    int periods[ndims];
    int source, dest;
    bool reorder = true;
    MPI_Comm comm1D;

    /***************************************************************/
    /* Create periodic shift */
    /***************************************************************/
    /* periodic shift is true. */
    periods[0] = true;
    /* create Cartesian mapping */
    MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &comm1D);

    //int errs = 0;
    MPI_Cart_shift(comm1D, dims[0], 1, &source, &dest);
    printf( "P[%d]:     periodic: shift  1: src[%d] P[%d] dest[%d]\n",
             rank,                          source, rank,    dest );
    fflush(stdout);

    MPI_Cart_shift(comm1D, dims[0], 0, &source, &dest);
    printf( "P[%d]:     periodic: shift  0: src[%d] P[%d] dest[%d]\n",
             rank,                          source, rank,    dest );
    fflush(stdout);

    MPI_Cart_shift(comm1D, dims[0], -1, &source, &dest);
```

```
    printf( "P[%d]:     periodic: shift -1: src[%d] P[%d] dest[%d]\n",
              rank,                         source, rank,    dest );
    fflush(stdout);
    MPI_Comm_free( &comm1D );

    /**********************************************************/
    /* Create non-periodic shift */
    /**********************************************************/
    if (rank == 0)
        printf("\nNon-periodic next\n");
    /* periodic shift is false. */
    periods[0] = false;
    MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &comm1D);

    MPI_Cart_shift(comm1D, dims[0], 1, &source, &dest);
    printf( "P[%d]: non-periodic: shift  1: src[%d] P[%d] dest[%d]\n",
              rank,                         source, rank,    dest );
    fflush(stdout);
    MPI_Cart_shift(comm1D, dims[0], 0, &source, &dest);
    printf( "P[%d]: non-periodic: shift  0: src[%d] P[%d] dest[%d]\n",
              rank,                         source, rank,    dest );
    fflush(stdout);
    MPI_Cart_shift(comm1D, dims[0], -1, &source, &dest);
    printf( "P[%d]: non-periodic: shift -1: src[%d] P[%d] dest[%d]\n",
              rank,                         source, rank,    dest );
    fflush(stdout);
    MPI_Comm_free( &comm1D );

    MPI_Finalize();
    return 0;
}
```

MPI Cartesian Map with 2 Dimensions



```
P[2]: non-periodic: shift -1: src[3] P[2] dest[1]
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
[make p2-2                                                                                                    ]
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  mpi-cart-2D-get-nbrs.c   -o mpi-cart-2D-get-nbrs
Platform: Darwin (4 cpu cores recognized)
MPIRUN mpi-cart-2D-get-nbrs with 16 node processes:
PW[0], CommSz[16]: PEdims = [4 x 4]
PW[ 0] Coord(0,0): SHIFT_DIM[0], Shift=1: nbr_lo[ 2] P[ 0] nbr_hi[ 1]
PW[ 0] Coord(0,0): SHIFT_DIM[1], Shift=1: nbr_lo[ 2] P[ 0] nbr_hi[ 1]
PW[ 2] Coord(0,2): SHIFT_DIM[0], Shift=1: nbr_lo[ 1] P[ 2] nbr_hi[ 3]
PW[ 6] Coord(1,2): SHIFT_DIM[0], Shift=1: nbr_lo[ 5] P[ 6] nbr_hi[ 7]
PW[ 6] Coord(1,2): SHIFT_DIM[1], Shift=1: nbr_lo[ 5] P[ 6] nbr_hi[ 7]
PW[ 2] Coord(0,2): SHIFT_DIM[1], Shift=1: nbr_lo[ 1] P[ 2] nbr_hi[ 3]
PW[12] Coord(3,0): SHIFT_DIM[0], Shift=1: nbr_lo[ 2] P[12] nbr_hi[13]
PW[12] Coord(3,0): SHIFT_DIM[1], Shift=1: nbr_lo[ 2] P[12] nbr_hi[13]
PW[ 4] Coord(1,0): SHIFT_DIM[0], Shift=1: nbr_lo[ 2] P[ 4] nbr_hi[ 5]
PW[ 4] Coord(1,0): SHIFT_DIM[1], Shift=1: nbr_lo[ 2] P[ 4] nbr_hi[ 5]
PW[ 8] Coord(2,0): SHIFT_DIM[0], Shift=1: nbr_lo[ 2] P[ 8] nbr_hi[ 9]
PW[ 8] Coord(2,0): SHIFT_DIM[1], Shift=1: nbr_lo[ 2] P[ 8] nbr_hi[ 9]
PW[10] Coord(2,2): SHIFT_DIM[0], Shift=1: nbr_lo[ 9] P[10] nbr_hi[11]
PW[10] Coord(2,2): SHIFT_DIM[1], Shift=1: nbr_lo[ 9] P[10] nbr_hi[11]
PW[14] Coord(3,2): SHIFT_DIM[0], Shift=1: nbr_lo[13] P[14] nbr_hi[15]
PW[14] Coord(3,2): SHIFT_DIM[1], Shift=1: nbr_lo[13] P[14] nbr_hi[15]
PW[ 1] Coord(0,1): SHIFT_DIM[0], Shift=1: nbr_lo[ 0] P[ 1] nbr_hi[ 2]
PW[ 3] Coord(0,3): SHIFT_DIM[0], Shift=1: nbr_lo[ 2] P[ 3] nbr_hi[ 2]
PW[ 3] Coord(0,3): SHIFT_DIM[1], Shift=1: nbr_lo[ 2] P[ 3] nbr_hi[ 2]
PW[ 5] Coord(1,1): SHIFT_DIM[0], Shift=1: nbr_lo[ 4] P[ 5] nbr_hi[ 6]
PW[ 5] Coord(1,1): SHIFT_DIM[1], Shift=1: nbr_lo[ 4] P[ 5] nbr_hi[ 6]
PW[ 9] Coord(2,1): SHIFT_DIM[0], Shift=1: nbr_lo[ 8] P[ 9] nbr_hi[10]
PW[ 9] Coord(2,1): SHIFT_DIM[1], Shift=1: nbr_lo[ 8] P[ 9] nbr_hi[10]
PW[ 1] Coord(0,1): SHIFT_DIM[1], Shift=1: nbr_lo[ 0] P[ 1] nbr_hi[ 2]
PW[11] Coord(2,3): SHIFT_DIM[0], Shift=1: nbr_lo[10] P[11] nbr_hi[ 2]
PW[11] Coord(2,3): SHIFT_DIM[1], Shift=1: nbr_lo[10] P[11] nbr_hi[ 2]
PW[ 7] Coord(1,3): SHIFT_DIM[0], Shift=1: nbr_lo[ 6] P[ 7] nbr_hi[ 2]
PW[ 7] Coord(1,3): SHIFT_DIM[1], Shift=1: nbr_lo[ 6] P[ 7] nbr_hi[ 2]
PW[13] Coord(3,1): SHIFT_DIM[0], Shift=1: nbr_lo[12] P[13] nbr_hi[14]
PW[13] Coord(3,1): SHIFT_DIM[1], Shift=1: nbr_lo[12] P[13] nbr_hi[14]
PW[15] Coord(3,3): SHIFT_DIM[0], Shift=1: nbr_lo[14] P[15] nbr_hi[ 2]
PW[15] Coord(3,3): SHIFT_DIM[1], Shift=1: nbr_lo[14] P[15] nbr_hi[ 2]
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
```

Figure 4: Console output: mpi-cart-2D-get-nbrs.c

mpi-cart-2D-get-nbrs.c

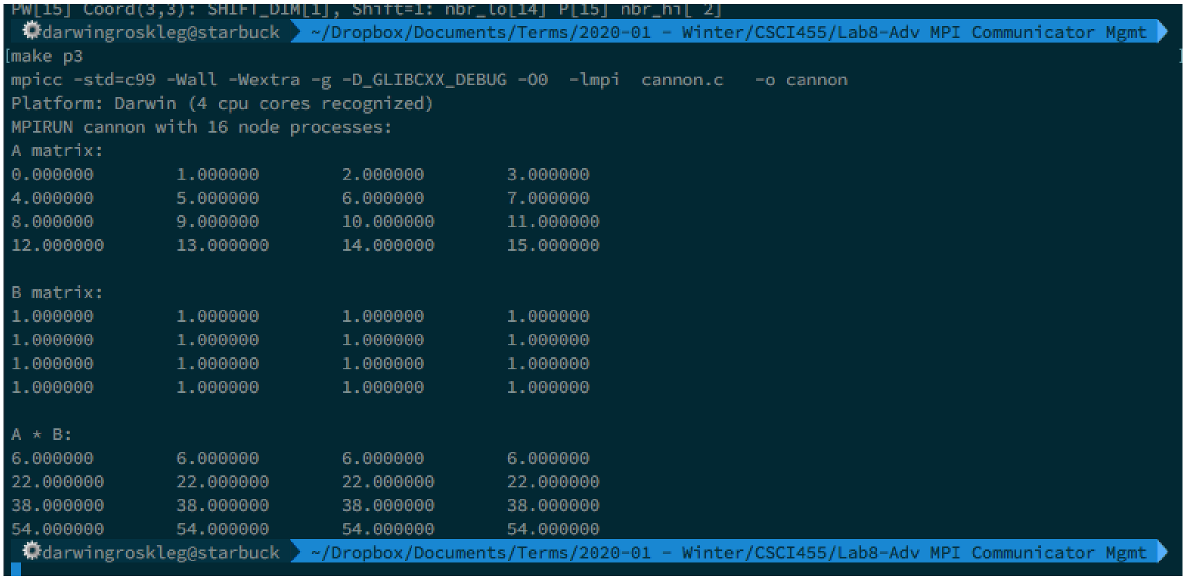
```
/* mpi-cart-2D-get-nbrs.c
 * ----------------------
 * 2 Dimensional Cartesian Virtual Topology
 *  finds the neighbors in a cartesian communicator
 */
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define SHIFT_ROW 0
#define SHIFT_COL 1
#define DISP 1

typedef enum { false, true } bool;

int wrap_row(int  row_width, int col);
int wrap_col(int col_height, int row);

int main(int argc, char *argv[]) {
    //int errs;

    /* start up initial MPI environment */
    MPI_Init(&argc, &argv);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int my_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    int ndims=2;
    int dims[ndims];
    int nrows;
    int ncols;

    /* process command line arguments*/
    if (argc == 3) {
        nrows = atoi (argv[1]);
        ncols = atoi (argv[2]);
        dims[0] = nrows; /* number of rows */
        dims[1] = ncols; /* number of columns */
        if( (nrows*ncols) != size) {
            if( my_rank ==0)
                printf("ERROR: nrows*ncols) = %d * %d = %d != %d\n",
                        nrows, ncols, nrows*ncols, size);
            MPI_Finalize();
            exit(0);
        }
    }
    else {
        nrows = ncols = (int)sqrt(size);
        dims[0] = dims[1] = 0;
    }

    /*************************************************************/
    /* create cartesian topology for processes */
```

```c
    /*************************************************************/
    MPI_Dims_create(size, ndims, dims);
    if(my_rank == 0)
        printf("PW[%d], CommSz[%d]: PEdims = [%d x %d]\n",
                my_rank,     size,       dims[0], dims[1]);

    /* create cartesian mapping */
    int periods[ndims];
    periods[0] = periods[1] = 0; /* periodic shift is .false. */
    int reorder = true;
    MPI_Comm comm2D;
    int ierr = 0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder,
                           &comm2D);
    if (ierr != 0)
        printf("ERROR[%d] creating CART\n", ierr);

    /* find my coordinates in the cartesian communicator group */
    int coord[ndims];
    MPI_Cart_coords(comm2D, my_rank, ndims, coord);

    /* use my cartesian coordinates to find my rank in cartesian group*/
    int my_cart_rank;
    MPI_Cart_rank(comm2D, coord, &my_cart_rank);

    //int source, dest;
    /* get my neighbors; axis is coordinate dimension of shift */
    /* axis=0 ==> shift along the rows: P[my_row-1]: P[me] : P[my_row+1] */
    /* axis=1 ==> shift along the columns P[my_col-1]: P[me] : P[my_col+1] */
    int nbr_i_lo, nbr_i_hi;
    MPI_Cart_shift(comm2D, dims[0], DISP, &nbr_i_lo, &nbr_i_hi);
    nbr_i_lo = wrap_row(ncols, nbr_i_lo);
    nbr_i_hi = wrap_row(ncols, nbr_i_hi);
    printf("PW[%2d] Coord(%d,%d): SHIFT_DIM[%d], Shift=%d: "
                           "nbr_lo[%2d] P[%2d] nbr_hi[%2d]\n",
            my_rank, coord[0], coord[1], SHIFT_ROW,  DISP,
                                nbr_i_lo, my_rank,   nbr_i_hi);

    int nbr_j_lo, nbr_j_hi;
    MPI_Cart_shift(comm2D, dims[1], DISP, &nbr_j_lo, &nbr_j_hi);
    nbr_j_lo = wrap_col(nrows, nbr_j_lo);
    nbr_j_hi = wrap_col(nrows, nbr_j_hi);
    printf("PW[%2d] Coord(%d,%d): SHIFT_DIM[%d], Shift=%d: "
                           "nbr_lo[%2d] P[%2d] nbr_hi[%2d]\n",
            my_rank, coord[0], coord[1], SHIFT_COL,  DISP,
                                nbr_j_lo, my_rank,   nbr_j_hi);
    fflush(stdout);

    MPI_Comm_free( &comm2D );
    MPI_Finalize();
    return 0;
}

int wrap_dim(int dim_width, int idx) {
    if (idx < 0)
        return dim_width + idx;
```

```c
    return idx;
}
int wrap_row(int row_width, int col) {  return wrap_dim(row_width, col); }
int wrap_col(int col_height, int row) { return wrap_dim(col_height, row); }
```

Part 3: Simplified Matrix Multiplication

Using Cannon's algorithm.



```
PW[15] Coord(3,3): SHIFT_DIM[1], Shift=1: nbr_lo[14] P[15] nbr_hi[ 2]
darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
make p3
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  cannon.c   -o cannon
Platform: Darwin (4 cpu cores recognized)
MPIRUN cannon with 16 node processes:
A matrix:
0.000000        1.000000        2.000000        3.000000
4.000000        5.000000        6.000000        7.000000
8.000000        9.000000        10.000000       11.000000
12.000000       13.000000       14.000000       15.000000

B matrix:
1.000000        1.000000        1.000000        1.000000
1.000000        1.000000        1.000000        1.000000
1.000000        1.000000        1.000000        1.000000
1.000000        1.000000        1.000000        1.000000

A * B:
6.000000        6.000000        6.000000        6.000000
22.000000       22.000000       22.000000       22.000000
38.000000       38.000000       38.000000       38.000000
54.000000       54.000000       54.000000       54.000000
darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab8-Adv MPI Communicator Mgmt
```

Figure 5: Console output: cannon.c

cannon.c

```c
/* cannon.c
 * --------
 * Simplified Matrix-Matrix Multiplication
 *
 *  This code is based on Cannon algorithm for matrix matrix multiplication.
 *  The main assumption in Cannon is that both A and B matrix must be square
 *  matrix and number of processors must be equal to the no of elements in
 *  A matrix.
 */
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define ndims 2 // 2 Dimension topology
#define SHIFT_ROW 1     // coord[1] is j
#define SHIFT_COL 0     // coord[0] is i

enum TaskRanks { Master = 0 };
// MxN Matrix: M rows, N columns
typedef struct MatrixSize {
  int m;    // rows    indexed by i
  int n;    // columns indexed by j
} mat_size_t;
mat_size_t get_matrices(float **L, float **R);

int main(int argc, char *argv[]) {
  // Initializing MPI
  MPI_Init(&argc, &argv);
  int size;
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // Read the data only if it is the root process (rank = 0)
  int   row, column;
  float *A = NULL;
  float *B = NULL;
  if (rank == Master) {
    mat_size_t msize = get_matrices(&A, &B);
    row    = msize.m;
    column = msize.n;
  }

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Bcast(&row,    1, MPI_INT, Master, MPI_COMM_WORLD);
  MPI_Bcast(&column, 1, MPI_INT, Master, MPI_COMM_WORLD);

  // set periodicity both vertical and horizontal movement
  //    periodic == true, wraps == true
  int periods[ndims] = {1, 1};
  int dims[ndims] = {row, column};
  int reorder = 1;  // true
  // Create Cartesian mapping of processes, a topological map
  MPI_Comm cart_comm;
  MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods, reorder, &cart_comm);
```

```c

  // Sending/Assigning each A and B element to the individual processor
  //    ASSUMES 1 element per process, enough processes
  float a_ij = 0;
  MPI_Scatter(A,    1, MPI_FLOAT,
              &a_ij, 1, MPI_FLOAT,
              Master, cart_comm);
  float b_ij = 0;
  MPI_Scatter(B,    1, MPI_FLOAT,
              &b_ij, 1, MPI_FLOAT,
              Master, cart_comm);
  //printf("p[%d] a=%f, b=%f\n", rank, a_ij, b_ij);
  MPI_Barrier(MPI_COMM_WORLD);

  // 2 Dimension topology, so 2 coordinates
  int coords[2];
  // get the coordinates in the new Cartesian grid
  MPI_Cart_coords(cart_comm, rank, ndims,   coords);
  // get the new rank in Cartesian group using coords
  int cart_rank;
  MPI_Cart_rank(cart_comm, coords,  &cart_rank);
  //printf("Coordinate of processor rank %d is [%d, %d], new rank is %d\n",
  //                                  rank, coords[0], coords[1], cart_rank);

  float c_ij = 0;
  int msg_tag = 11;
  // neighbor ranks
  int right = 0, left = 0, down = 0, up = 0;
  // Pumping along systolic array:
  //    ASSUMES a square matrix
  for (int ij = 0; ij < row; ij++) {
    // get the shifted source and destination rank horizontally
    MPI_Cart_shift(cart_comm, SHIFT_ROW, ij, &right, &left);
    // get the shifted source and destination rank vertically
    MPI_Cart_shift(cart_comm, SHIFT_COL, ij, &down, &up);
    // send and receive using single buffer:
    //      shift value from RIGHT coordinate to LEFT coordinate
    MPI_Sendrecv_replace(&a_ij, 1, MPI_FLOAT,
            left,  msg_tag,     // rank of dest   (send to   left)
            right, msg_tag,     // rank of source (recv from right)
            cart_comm, MPI_STATUS_IGNORE);
    // send and receive using single buffer:
    //      shift value from DOWN coordinate to UP coordinate
    MPI_Sendrecv_replace(&b_ij, 1, MPI_FLOAT,
            up,   msg_tag,      // rank of dest   (send up)
            down, msg_tag,      // rank of source (recv from below)
            cart_comm, MPI_STATUS_IGNORE);
    // Calculation of matrix multiplication
    c_ij += a_ij * b_ij;
  }

  // allocate memory for C matrix
  float *C = (float *) calloc(sizeof(float), row * row);
  // Gather the multiplication result from every processor
  MPI_Gather(&c_ij, 1, MPI_FLOAT,
                 C, 1, MPI_FLOAT,
```

```
            Master, cart_comm);

  // Printing the result of Matrix multiplication stored in C array
  if (rank == Master) {
    int k = 0;
    printf("\nA * B:\n");
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < column; j++) {
        printf("%f\t", C[k]);
        k++;
      }
      printf("\n");
    }
  }

  MPI_Finalize();
  return 0;
}

// Assumes A and B will have same numberof rows and columns
//  TODO Should only assume that columns(A) == rows(B).
//      A_mxl . B_lxn  ==>  C_mxn
mat_size_t get_matrices(float **L, float **R) {
  int row = 0;
  int column = 0;
  // finding the number of rows & columns in A matrix
  FILE *fp;
  fp = fopen("A.txt", "r");
  int count = 0;
  char ch;
  float n;
  // scan each line
  while (fscanf(fp, "%f", &n) != -1) {
    ch = fgetc(fp);
    if (ch == '\n') {
      row++;
    }
    count++;
  }
  column = count / row;

  // Check to see to have enough processors for the elements
  int cluster_size;
  MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);
  if (count != cluster_size) {
    printf("No of Processors must be equal to %d\nCode terminated\n", count);
    MPI_Finalize();
    fclose(fp);
    exit(1);
  }

  // Jump back to beginning of file for matrix A
  fseek(fp, 0, SEEK_SET);

  // allocate memory for A and B
  float *A = (float *)calloc(sizeof(float), row * column);
```

```c
  float *B = (float *)calloc(sizeof(float), row * column);

  // Scanning and printing Matrix A
  int k = 0;
  printf("A matrix:\n");
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < column; j++) {
      fscanf(fp, "%f", &n);
      A[k] = n;
      printf("%f\t", A[k]);
      k++;
    }
    printf("\n");
  }
  fclose(fp);

  // Scanning and printing Matrix B
  k = 0;
  printf("\nB matrix:\n");
  // read data for B matrix
  fp = fopen("B.txt", "r");
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < column; j++) {
      fscanf(fp, "%f", &n);
      B[k] = n;
      printf("%f\t", B[k]);
      k++;
    }
    printf("\n");
  }
  fclose(fp);

  *L = A;
  *R = B;
  return (mat_size_t){row, column};
}
```