```
// Conway’s Game of Life
// Global variable include file
//
// CSCI 4576/5576 High Performance Scientific Computing
// Matthew Woitaszek

// <soapbox>
// This file contains global variables: variables that are defined throughout
// the entire program, even between multiple independent source files. Of
// course, global variables are generally bad, but they’re useful here because
// it allows all of the source files to know their rank and the number of MPI
// tasks. But don’t use it lightly.
//
// How it works:
//  * One .cpp file -- usually the one that contains main(), includes this file
//    within #define __MAIN, like this:
//      #define __MAIN
//      #include globals.h
//      #undef __MAIN
//  * The other files just "#include globals.h"

typedef enum { SERIAL, ROW, GRID } dist;

#ifdef __MAIN
int                     rank;
int                     np;
int                     my_name_len;
char                    my_name[255];
#else
extern int              rank;
extern int              np;
extern int              my_name_len;
extern char             *my_name;
#endif

//
// Conway globals
//
#ifdef __MAIN

int                     nrows;          // Number of rows in our partitioning
int                     ncols;          // Number of columns in our partitioning
int                     my_row;         // My row number
int                     my_col;         // My column number

// Local logical game size
int                     fake_data_size;
int                     local_width;    // Width and height of game on this processor
int                     local_height;
int                     global_width;
int                     global_height;
int                     N;

// Local physical field size
int                     field_width;        // Width and height of field on this processor
int                     field_height;       // (should be local_width+2, local_height+2)
unsigned char           *env_a;             // 1D character array to represent our 1st 2D environment
unsigned char           *env_b;             // 1D character array to represent our 2nd 2D environment
unsigned char           *out_buffer;        // 1D character array to represent our global 2D environment + padding
dist                     dist_type;

#else
extern int              nrows;
extern int              ncols;
extern int              my_row;
extern int              my_col;

extern int              fake_data_size;
extern int              local_width;
extern int              local_height;
extern int              global_width;
extern int              global_height;
extern int              N;

extern int              field_width;
extern int              field_height;
extern unsigned char    *env_a;
extern unsigned char    *env_b;
extern unsigned char    *out_buffer;

extern dist              dist_type;

#endif
```

```
/*
 * Helper function file to be included in main
 * Written by Adam Ross
 *
 */

void print_usage();
void print_matrix(unsigned char *matrix);
void print_padded_matrix(unsigned char *matrix);
void print_global_matrix(unsigned char *matrix);
void swap(unsigned char **a, unsigned char **b);
unsigned char *Allocate_Square_Matrix();
int count_alive(unsigned char *matrix);
int Calc_Confidence_Interval_stop(double *timing_data, int n);
```

```c
typedef enum { false, true } bool; // Provide C++ style ’bool’ type in C
bool readpgm( char *filename );
```

```
/* $Id: pprintf.h,v 1.3 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

// Modified by Michael Oberg, 2015/10/01 to support both C or C++

#ifdef __cplusplus
extern "C" int init_pprintf(int);
extern "C" int pp_set_banner(char *);
extern "C" int pp_reset_banner();
extern "C" int pprintf(char *, ...);
#endif

extern int init_pprintf(int);
extern int pp_set_banner(char *);
extern int pp_reset_banner();
extern int pprintf(char *, ...);
```

```
/* MT1 - Midterm Part I: Conway’s Game of Line
 *
 *
 * Name: Adam Ross
 *
 * Input: -i filename, -d distribution type <0 - serial, 1 - row, 2 - grid>
 *        -s turn on asynchronous MPI functions, -c <#> if and when to count living
 * Output: Various runtime information including bug counting if turned on
 *
 *
 * Note: a Much of this code, namely the pgm reader and most of the support libraries
 * is credited to: Dr. Matthew Woitaszek
 *
 * Written by Adam Ross, modified from code supplied by Michael Oberg, modified from code su
pplied by Dr. Matthew Woitaszek
 */

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <math.h>
#include <string.h>
#include "mpi.h"

// Include global variables. Only this file needs the #define
#define __MAIN
#include "globals.h"
#undef __MAIN

// User includes
#include "pprintf.h"
#include "pgm.h"
#include "helper.h"

int main(int argc, char* argv[]) {
    unsigned short      i, j;
    unsigned short      neighbors =         0;
    int                 half_height;
    int                 top_dest,
                        top_source,
                        bot_dest ,
                        bot_source,
                        left_dest,
                        left_source,
                        right_dest,
                        right_source =     5280;
    MPI_Status          status;
    MPI_Request         ar, br, lr, rr;
    MPI_File            out_file;
    int                 counting =          -1;
    int                 count =             0;
    int                 total =             0;
    int                 n =                 0;
    int                 option =            -1;
    bool                async =             false;
    bool                writing =           false;
    int                 iter_num =          1000;
    char                *filename;
    char                frame[47];
    int                 gsizes[2], distribs[2], dargs[2], psizes[2];
    MPI_Datatype        ext_array;
    MPI_Datatype        darray;
    MPI_Datatype        column;
```

```
double              start;
double              finish;
double              *timing_data;
double              avg =               0;

fake_data_size = 0;

// Parse commandline
while ((option = getopt(argc, argv, "d:an:c:i:ws:")) != -1) {
    switch (option) {
         case ’d’ :
             dist_type = atoi(optarg);
             break;
         case ’a’ :
             async = true;
             break;
         case ’n’ :
             iter_num = atoi(optarg);
             break;
         case ’c’ :
             counting = atoi(optarg);
             break;
         case ’i’ :
             filename = optarg;
             break;
         case ’w’ :
             writing = true;
             break;
         case ’s’ :
             fake_data_size = atoi(optarg);
             break;
         default:
             print_usage();
             exit(1);
    }
}

// Initialize MPI
MPI_Init(&argc, &argv);

// Get the communicator and process information
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &np);

// Print rank and hostname
MPI_Get_processor_name(my_name, &my_name_len);
printf("Rank %i is running on %s\n", rank, my_name );

// Initialize the pretty printer
init_pprintf(rank);
pp_set_banner("main");

timing_data = (double *) calloc(iter_num, sizeof(double));

if (rank == 0) {
    pprintf("Welcome to Conway’s Game of Life!\n");
}

//
// Determine the partitioning
//
if (dist_type < GRID) {
    if (!rank)
```

```
            pprintf("Row or Serial distribution selected.\n");
        ncols = 1;
        nrows = np;
        my_col = 0;
        my_row = rank;
    } else {
        if (!rank)
            pprintf("Grid distribution selected.\n");
        nrows = (int)sqrt(np);
        ncols = (int)sqrt(np);
        my_row = rank / nrows;
        my_col = rank - my_row * nrows;

        //pprintf("Num rows%d\tNum cols %d\tMy row %d\tMy col %d\n", nrows, ncols, my_row, m
y_col);
    }

    if (np != nrows * ncols) {
        if (!rank)
             pprintf("Error: %ix%i partitioning requires %i np (%i provided)\n",
                  nrows, ncols, nrows * ncols, np );
        MPI_Finalize();
        return 1;
    }


    // Now, calculate neighbors (N, S, E, W, NW, NE, SW, SE)
    // ... which means you ...


    // Read the PGM file. The readpgm() routine reads the PGM file and, based
    // on the previously set nrows, ncols, my_row, and my_col variables, loads
    // just the local part of the field onto the current processor. The
    // variables local_width, local_height, field_width, field_height, as well
    // as the fields (field_a, field_b) are allocated and filled.
    if (!readpgm(filename)) {
        if (rank == 0)
            pprintf("An error occured while reading the pgm file\n");
        MPI_Finalize();
        return 1;
    }

    // Set half array values for async work
    half_height = (local_height / 2) + 1;

    // Set up darray create properties
    gsizes[0] = global_height; /* no. of rows in global array */
    gsizes[1] = global_width; /* no. of columns in global array*/
    distribs[0] = MPI_DISTRIBUTE_BLOCK;
    distribs[1] = MPI_DISTRIBUTE_BLOCK;
    dargs[0] = MPI_DISTRIBUTE_DFLT_DARG;
    dargs[1] = MPI_DISTRIBUTE_DFLT_DARG;
    psizes[0] = nrows; /* no. of processes in vertical dimension of process grid */
    psizes[1] = ncols; /* no. of processes in horizontal dimension of process grid */

    // Create darray and commit
    MPI_Type_create_darray(np, rank, 2, gsizes, distribs, dargs, psizes, MPI_ORDER_C, MPI_UN
SIGNED_CHAR, &darray);
    MPI_Type_commit(&darray);

    // Create data type to extract useful data out of padding
    MPI_Type_vector(local_height, local_width, field_width, MPI_UNSIGNED_CHAR, &ext_array);
    MPI_Type_commit(&ext_array);
```

```

    // Build MPI datatype vector of every Nth item - i.e. a column
    MPI_Type_vector(local_height, 1, field_width, MPI_UNSIGNED_CHAR, &column);
    MPI_Type_commit(&column);

    // allocate memory to print whole stages into pgm files for animation
    if (rank == 0) {
        out_buffer = Allocate_Square_Matrix(global_width, global_height);
    }

    // Count initial living count
    if (counting != -1) {
        count = count_alive(env_a);
        pprintf("Bugs alive at the start: %d\n", count);

        MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        if (rank == 0) {
            pprintf("%i total bugs alive at the start.\n", total);
        }
    }

    // calculate pairings
    if (dist_type > SERIAL) {
        // calculate pairings
        if (dist_type == ROW) { // row distro
            top_dest = bot_source = rank - 1;
            top_source = bot_dest = rank + 1;

            if (rank == 0) { // rank 0, no need to send
                top_dest = MPI_PROC_NULL;
                bot_source = MPI_PROC_NULL;
            } else if (rank == (np - 1)) { // rank np-1 no need to send
                top_source = MPI_PROC_NULL;
                bot_dest = MPI_PROC_NULL;
            }
        } else if (dist_type == GRID) {
        // calculate pairings
            top_dest = bot_source = rank - nrows;
            top_source = bot_dest = rank + nrows;
            left_dest = right_source = rank - 1;
            left_source = right_dest = rank + 1;

            if (my_row == 0) { // top row no need to send up
                top_dest = MPI_PROC_NULL;
                bot_source = MPI_PROC_NULL;
            } else if (my_row == sqrt(np) - 1) { // rank bottom row no need to send down
                top_source = MPI_PROC_NULL;
                bot_dest = MPI_PROC_NULL;
            }
            if (my_col == 0) {
                left_dest = MPI_PROC_NULL;
                right_source = MPI_PROC_NULL;
            } else if (my_col == sqrt(np) - 1) {
                left_source = MPI_PROC_NULL;
                right_dest = MPI_PROC_NULL;
            }
            //pprintf("top: %d\tbot %d\tleft %d\tright %d\tProc %d\n", top_dest, bot_dest, l
eft_dest, right_dest, MPI_PROC_NULL);
        }
    }

    while(n < iter_num) {
        if (writing) {
            for (int k = 1; k < field_height - 1; k++) {
                for (int a = 1; a < field_width - 1; a++) {
                    if (!env_b[k * field_width + a]) {
                        env_a[k * field_width + a] = 255;
                    } else {
                        env_a[k * field_width + a] = 0;
                    }
                }
            }

            sprintf(frame, "/oasis/scratch/comet/adamross/temp_project/%d.pgm", n);
            MPI_File_open(MPI_COMM_WORLD, frame, MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_N
ULL, &out_file);

            char header[20];
            sprintf(header, "P5\n%d %d\n%d\n", global_width, global_height, 255);
            int header_len = strlen(header);

            if (rank == 0) {
                //write header
                //MPI_File_set_view(out_file, 0,  MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "nat
ive", MPI_INFO_NULL);
                MPI_File_write(out_file, &header, header_len, MPI_UNSIGNED_CHAR, MPI_STATUS_
IGNORE);
            }

            // write data
            //MPI_File_set_view(out_file, 15 + rank * local_width + local_width, MPI_UNSIGNE
D_CHAR, darray, "native", MPI_INFO_NULL);
            MPI_File_set_view(out_file, header_len, MPI_UNSIGNED_CHAR, darray, "native", MPI
_INFO_NULL);

            //MPI_File_write(out_file, env_a, (local_height * local_width), ext_array, &stat
us);
            MPI_File_write(out_file, &env_a[field_width + 12], 1, ext_array, &status);
            MPI_File_close(&out_file);

            for (int k = 1; k < field_height - 1; k++) {
                for (int a = 1; a < field_width  - 1; a++) {
                    if (!env_a[k * field_width + a]) {
                        env_a[k * field_width + a] = 0;
                    } else {
                        env_a[k * field_width + a] = 1;
                    }
                }
            }
        }

        start = MPI_Wtime();

        //Uncomment to produce pgm files per frame in serial file system
        //MPI_Gather(&env_b[field_width + 1], 1, ext_array, out_buffer, local_width * local_
height, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
        /*if (rank == 0) {
            print_global_matrix(out_buffer);
        }*/

        /*if (rank == 0) {
            for (int k = 0; k < global_height; k++) {
                for (int a = 0; a < global_width; a++) {
                    if (!out_buffer[k * global_width + a]) {
                        out_buffer[k * global_width + a] = 255;
```

```
                    } else {
                        out_buffer[k * global_width + a] = 0;
                    }
                }
            }

            sprintf(frame, "%d.pgm", n);
            FILE *file = fopen(frame, "w");
            fprintf(file, "P5\n");
            fprintf(file, "%d %d\n", global_width, global_height);
            fprintf(file, "%d\n", 255);
            fwrite(out_buffer, sizeof(unsigned char), global_width * global_height, file);
            fclose(file);
        }*/

        ////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////

        // do upper half minus edges, check if need recv
        // do lower half minus edges, check is need recv
        // do upper row
        // do columns
        // do lower row

        if (async && dist_type == ROW && n < iter_num - 1) {
            // Aschrnous enabled, receive from the last iteration or inital setup
            MPI_Irecv(&env_a[(field_height - 1) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, top_source, 0, MPI_COMM_WORLD, &ar);
            MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_sourc
e, 0, MPI_COMM_WORLD, &br);

            MPI_Isend(&env_b[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_dest,
 0, MPI_COMM_WORLD, &ar);
            MPI_Isend(&env_b[(field_height - 2) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, bot_dest, 0, MPI_COMM_WORLD, &br);
        } else if (async && dist_type == GRID && n < iter_num - 1) {
            MPI_Irecv(&env_a[2 * field_width - 1], 1, column, left_source, 0, MPI_COMM_WORLD
, &lr);
            MPI_Irecv(&env_a[1 * field_width + 0], 1, column, right_source, 0, MPI_COMM_WORL
D, &rr);

            MPI_Isend(&env_b[1 * field_width + 1], 1, column, left_dest, 0, MPI_COMM_WORLD,
&lr);
            MPI_Isend(&env_b[2 * field_width - 2], 1, column, right_dest, 0, MPI_COMM_WORLD,
 &rr);
        }

        ////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////

        // calulate neighbors and form state + 1 for upper half - edges
        for (i = 2; i < half_height; i++) {
            for (j = 2; j < local_width; j++) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
           env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        ////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////

        // Receive our horizontal communication and send the vertical
        if (async && dist_type == GRID && n > 0) {
            // Need the horizontal data before we send vertically
            MPI_Wait(&lr, &status);
            MPI_Wait(&rr, &status);

            // Aschrnous enabled, receive from the last iteration or inital setup
            MPI_Irecv(&env_a[(field_height - 1) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, top_source, 0, MPI_COMM_WORLD, &ar);
            MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_sourc
e, 0, MPI_COMM_WORLD, &br);

            MPI_Isend(&env_a[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_dest,
 0, MPI_COMM_WORLD, &ar);
            MPI_Isend(&env_a[(field_height - 2) * field_width + 0], field_width, MPI_UNSIGNE
D_CHAR, bot_dest, 0, MPI_COMM_WORLD, &br);
        }

        ////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////

        // calulate neighbors and form state + 1 for lower half - edges
        for (i = half_height; i < local_height; i++) {
            for (j = 2; j < local_width; j++) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
           env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        ////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////
```

```

        if (async && n > 0) {
            // To avoid getting data mixed up wait for it to come through
            MPI_Wait(&ar, &status);
            MPI_Wait(&br, &status);
        }

        ////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////

        // calulate neighbors and form state + 1 for edges
        i = 1;
        for (j = 1; j < local_width + 1; j++) {
            neighbors = 0;
            // loop unroll neighbor checking - access row dominant
            neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_width
+ j] + env_a[(i - 1) * field_width + j + 1];
            neighbors += env_a[i * field_width + j - 1] +
       env_a[i * field_width + j + 1];
            neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_width
+ j] + env_a[(i + 1) * field_width + j + 1];

            // Determine env_b based on neighbors in env_a
            if (neighbors == 2) {
                env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 spawn
            } else if (neighbors == 3) {
                env_b[i * field_width + j] = 1; // exactly 3 spawn
            } else {
                env_b[i * field_width + j] = 0; // zero or one or 4 or more die

            }
        }

        // calulate neighbors and form state + 1 for edges
        for (i = 1; i < local_height; i++) {
            // need i = 1 and local_width + 1
            for (j = 1; j < local_width + 1; j += local_width - 1) {
                neighbors = 0;
                // loop unroll neighbor checking - access row dominant
                neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_wi
dth + j] + env_a[(i - 1) * field_width + j + 1];
                neighbors += env_a[i * field_width + j - 1] +
           env_a[i * field_width + j + 1];
                neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_wi
dth + j] + env_a[(i + 1) * field_width + j + 1];

                // Determine env_b based on neighbors in env_a
                if (neighbors == 2) {
                    env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 sp
awn
                } else if (neighbors == 3) {
                    env_b[i * field_width + j] = 1; // exactly 3 spawn
                } else {
                    env_b[i * field_width + j] = 0; // zero or one or 4 or more die

                }
            }
        }

        // calulate neighbors and form state + 1 for edges
        i = local_height;
        for (j = 1; j < local_width + 1; j++) {
            neighbors = 0;
            // loop unroll neighbor checking - access row dominant
            neighbors += env_a[(i - 1) * field_width + j - 1] + env_a[(i - 1) * field_width
+ j] + env_a[(i - 1) * field_width + j + 1];
            neighbors += env_a[i * field_width + j - 1] +
       env_a[i * field_width + j + 1];
            neighbors += env_a[(i + 1) * field_width + j - 1] + env_a[(i + 1) * field_width
+ j] + env_a[(i + 1) * field_width + j + 1];

            // Determine env_b based on neighbors in env_a
            if (neighbors == 2) {
                env_b[i * field_width + j] = env_a[i * field_width + j]; // exactly 2 spawn
            } else if (neighbors == 3) {
                env_b[i * field_width + j] = 1; // exactly 3 spawn
            } else {
                env_b[i * field_width + j] = 0; // zero or one or 4 or more die

            }
        }

        ////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////

        // If we are doing async we now have the data we need for the next iter, send it
        // If we are in row distrobution send vertically - thats all we need to do
        // If we are in block distrobution send horizontally first

        // sync or a async here MPI_PROC_NULs
        if (dist_type > SERIAL && !async) {
            // If we choose block decomposition send horizontally first
            if (dist_type == GRID) {
                // Send to right or recv from left
                MPI_Sendrecv(&env_b[1 * field_width + 1], 1, column, left_dest, 0,
                             &env_b[2 * field_width - 1], 1, column, left_source, 0, MPI_COM
M_WORLD, &status);
                // Send to left or recv from right
                MPI_Sendrecv(&env_b[2 * field_width - 2], 1, column, right_dest, 0,
                             &env_b[1 * field_width + 0], 1, column, right_source, 0, MPI_CO
MM_WORLD, &status);
            }
            // Send to below or recv from above
            MPI_Sendrecv(&env_b[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_de
st, 0,
                         &env_b[(field_height - 1) * field_width + 0], field_width, MPI_UNSI
GNED_CHAR, top_source, 0, MPI_COMM_WORLD, &status);
            // Send to above or recv from below
            MPI_Sendrecv(&env_b[(field_height - 2) * field_width + 0], field_width, MPI_UNSI
GNED_CHAR, bot_dest, 0,
                         &env_b[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_so
urce, 0, MPI_COMM_WORLD, &status);
        }

        finish = MPI_Wtime();
        if (rank == 0 && n > 0) {
            timing_data[n] = finish - start;
        }

        // If counting is turned on print living bugs this iteration
        if (n != 0 && (n % counting) == 0) {
            count = count_alive(env_b);

            MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
            if (rank == 0) {
                pprintf("%i total bugs alive at iteraion %d\n", total, n);
```

```
            }
        }

        n++;
        swap(&env_b, &env_a);
    }

    if (rank == 0) {
        for (i = 1; i < n; i++) {
            avg += timing_data[i];
        }

        avg = avg / (n - 1);

        pprintf("avg: %1.20f\n", avg);
    }

    // Final living count
    if (counting != -1 && n != counting) {
        count = count_alive(env_a);
        pprintf("Per process bugs alive at the end: %d\n", count);

        MPI_Reduce(&count, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        if (rank == 0) {
            pprintf("%i total bugs alive at the end.\n", total);
        }
    }

    // Free the fields
    MPI_Barrier(MPI_COMM_WORLD);
    if (env_a != NULL) free( env_a );
    if (env_b != NULL) free( env_b );
    if (timing_data != NULL) free( timing_data );


    MPI_Finalize();


} /* end main */
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "globals.h"
#include <math.h>

// Self explanitory
void print_usage() {
    printf("Usage: -i filename, -d distribution type <0 - serial, 1 - row, 2 - grid>, -s turn on asynchronous MPI functions, -c <#> if and when to count living\n");
}

/*
 * Helper method to print a square matrix
 * Input: a matrix and the order of that matrix
 */
void print_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 1; i < local_height + 1; i++) {
        for (j = 1; j < local_width + 1; j++) {
            printf("%u ", matrix[i * field_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

void print_padded_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 0; i < field_height; i++) {
        for (j = 0; j < field_width; j++) {
            printf("%u ", matrix[i * field_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

void print_global_matrix(unsigned char *matrix) {
    unsigned char          i;
    unsigned char          j;

    //printf("local_width is: %d, local_height is: %d\n", local_width, local_height);

    for (i = 0; i < global_height; i++) {
        for (j = 0; j < global_width; j++) {
            printf("%u ", matrix[i * global_width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

/*
 * Helper function to swap array pointers
 * Input: array a and Array b
 */
void swap(unsigned char **a, unsigned char **b) {
    unsigned char          *tmp = *a;
    *a = *b;
    *b = tmp;
}

/*
 * Helper function to allocate 2D array of ints
 * Input: Order of the array
 */
unsigned char *Allocate_Square_Matrix(int width, int height) {
    unsigned char          *matrix;

    matrix = (unsigned char *) calloc(width * height, sizeof(unsigned char));

    return matrix;
}

/*
 * Helper function to clean up code duplication
 * Input: pointer to array
 */
int count_alive(unsigned char *matrix) {
    int                   count =           0;
    int                   i, j;

    for (i = 1; i < local_height + 1; i++) {
        for (j = 1; j < local_width + 1; j++) {
            if (matrix[i * field_width + j]) {
                count ++;
            }
        }
    }

    return count;
}

/* Helper function calculate the confidence interval, error margins and determine
 * if we should keep looping.
 * Returns 1 or 0 for conintue or stop.
*/
int Calc_Confidence_Interval_stop(double *timing_data, int n) {
    double              sum =               0.0;
    double              mean =              0.0;
    double              std_dev =           0.0;
    double              marg_err =          0.0;
    double              marg_perc =         100.0;
    int                 i;

    if (n > 2) {
        for (i = 0; i < n; i++) {
            sum += timing_data[i];
        }
        mean = sum / n;
        sum = 0.0;
        for (i = 0; i < n; i++) {
            sum += pow(timing_data[i] - mean, 2);
        }
        std_dev = sqrt(sum / n);
        marg_err = 1.96 * (std_dev / sqrt(n));
        marg_perc = (marg_err / mean) * 100;
    } else {
        return 0;
    }
    if (marg_perc > 5.0  && n < 20) {
        return 0;
    } else {
        printf("%d\t%1.20f\t%1.10f\t%1.10f\t%f\t", n, mean, std_dev, marg_err, marg_perc);

        return 1;
    }
}
```

```
/*
 * HPGM helper functions to be included in main
 * Provided by Michael Oberg, Modified by Adam Ross
 *
 */

// System includes
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mpi.h"
#include <math.h>

// User includes
#include "globals.h"
#include "pprintf.h"
#include "helper.h"

typedef enum { false, true } bool; // Provide C++ style ’bool’ type in C

bool readpgm( char *filename ){
    // Read a PGM file into the local task
    //
    // Input: char *filename, name of file to read
    // Returns: True if file read successfully, False otherwise
    //
    // Preconditions:
    //  * global variables nrows, ncols, my_row, my_col must be set
    //
    // Side effects:
    //  * sets global variables local_width, local_height to local game size
    //  * sets global variables field_width, field_height to local field size
    //  * allocates global variables env_a and env_b
    int             x = 0;
    int             y = 0;
    int             start_x, start_y;
    int             b, lx, ly, ll;
    char            header[10];
    int             depth;
    int             rv;
    int             grab_width;
    int             grab_height;
    int             x_add = 1;
    int             y_add = 1;

    pp_set_banner( "pgm:readpgm" );

    // Open the file
    if (rank == 0)
        pprintf( "Opening file %s\n", filename );
    FILE *fp = fopen( filename, "r" );
    if (!fp) {
        pprintf( "Error: The file ’%s’ could not be opened.\n", filename );
        return false;
    }

    // Read the PGM header, which looks like this:
    //  |P5        magic version number
    //  |900 900       width height
    //  |255         depth
    rv = fscanf( fp, "%6s\n%i %i\n%i\n", header, &global_width, &global_height, &depth );
    if (rv != 4){
        if (rank == 0)
            pprintf( "Error: The file ’%s’ did not have a valid PGM header\n", filename );
        return false;
    }

    if (fake_data_size != 0) {
        global_width = global_height = fake_data_size;
    }

    if (rank == 0)
        pprintf( "%s: %s %i %i %i\n", filename, header, global_width, global_height, depth )
;

    // Make sure the header is valid
    if (strcmp( header, "P5")) {
        if(rank==0)
            pprintf( "Error: PGM file is not a valid P5 pixmap.\n" );
        return false;
    }
    if (depth != 255) {
        if (rank == 0)
            pprintf( "Error: PGM file has depth=%i, require depth=255 \n", depth );
        return false;
    }

    // Make sure that the width and height are divisible by the number of
    // processors in x and y directions

    if (global_width % ncols) {
        if (rank == 0)
            pprintf( "Error: %i pixel width cannot be divided into %i cols\n", global_width,
 ncols );
        return false;
    }
    if (global_height % nrows) {
        if (rank == 0)
            pprintf( "Error: %i pixel height cannot be divided into %i rows\n", global_heigh
t, nrows );
        return false;
    }

    // Divide the total image among the local processors
    local_width = global_width / ncols;
    local_height = global_height / nrows;

    // Find out where my starting range is
    start_x = local_width * my_col;
    start_y = local_height * my_row;

    grab_width = local_width;
    grab_height = local_height;

    pprintf( "Hosting data for x:%03i-%03i y:%03i-%03i\n",
        start_x, start_x + local_width,
        start_y, start_y + local_height );

    // Create the array!
    field_width = local_width + 2;
    field_height = local_height + 2;

    // allocate contiguous memory - returns a pointer to the memory
    env_a = Allocate_Square_Matrix(field_width, field_height);
    env_b = Allocate_Square_Matrix(field_width, field_height);
    // Need to handle edge cases to not grab extras
     if (dist_type == ROW ){
        grab_height = field_height;

        if (rank == 0) {
            grab_height--;
        } else if (rank == np - 1) {
            grab_height--;
            start_y--;
            y_add = 0;
        } else {
            start_y--;
            y_add = 0;
        }
    } else if (dist_type == GRID) {
        grab_width = field_width;
        grab_height = field_height;

        if (my_row == 0) {
            grab_height--;
        } else if (my_row == sqrt(np) - 1) {
            grab_height--;
            start_y--;
            y_add = 0;
        } else {
            start_y--;
            y_add = 0;
        }

        if (my_col == 0) {
            grab_width--;
        } else if (my_col == sqrt(np) - 1) {
            grab_width--;
            start_x--;
            x_add = 0;
        } else {
            start_x--;
            x_add = 0;
        }
    }

    //pprintf("start_x: %d\tstart_y: %d\tx_add: %d\ty_add: %d\t\n", start_x, start_y, x_add,
y_add);
    //pprintf("grab_width: %d\tgrab_height: %d\t\n", grab_width, grab_height);

    // Read the data from the file. Save the local data to the local array.
    if (fake_data_size == 0) {
        for (y = 0; y < global_height; y++) {
            for (x = 0; x < global_width; x++) {
                // Read the next character
                b = fgetc(fp);
                if (b == EOF){
                    pprintf( "Error: Encountered EOF at [%i,%i]\n", y,x );
                    return false;
                }

                // From the PGM, black cells (b=0) are bugs, all other
                // cells are background
                if (b == 0) {
                    b = 1;
                } else {
                    b = 0;
                }

                // If the character is local, then save it!
                if (x >= start_x &&
                    x < start_x + grab_width &&
                    y >= start_y &&
                    y < start_y + grab_height) {

                    // Calculate the local pixels (+1 for ghost row,col)
                    lx = x - start_x + x_add;
                    ly = y - start_y + y_add;
                    ll = (ly * field_width + lx);
                    env_a[ll] = b;
                    env_b[ll] = b;
                } // save local point


            } // for x
        } // for y
    }



    fclose(fp);

    pp_reset_banner();
    return true;
}
```

```c
/* $Id: pprintf.c,v 1.5 2006/02/09 20:42:25 mccreary Exp $ */

/*
 * Copyright (c) 2006 Sean McCreary <mccreary@mcwest.org>. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * 3. The name of the author may not be used to endorse or promote products
 * derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* Pretty printf() wrapper for MPI processes */

#include <stdio.h>
#include <stdarg.h>
#include <string.h>

#define PP_MAX_BANNER_LEN       14
#define PP_MAX_LINE_LEN         81
#define PP_PREFIX_LEN           27
#define PP_FORMAT               "[%3d:%03d] %-14s : "

static int pid = -1;
static int msgcount = 0;
static char banner[PP_MAX_BANNER_LEN] = "";
static char oldbanner[PP_MAX_BANNER_LEN] = "";

int init_pprintf(int);
int pp_set_banner(char *);
int pp_reset_banner();
int pprintf(char *, ...);

int init_pprintf( int my_rank )
{
    pp_set_banner("init_pprintf");
    pid = my_rank;
/*
    pprintf("PID is %d\n", pid);
*/
    return 0;
}
```

```c
int pp_set_banner( char *newbanner )
{
    strncpy(oldbanner, banner, PP_MAX_BANNER_LEN);
    strncpy(banner, newbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pp_reset_banner()
{
    strncpy(banner, oldbanner, PP_MAX_BANNER_LEN);
    return 0;
}

int pprintf( char *format, ... )
{
    va_list ap;
    char output_line[PP_MAX_LINE_LEN];

    /* Construct prefix */
    snprintf(output_line, PP_PREFIX_LEN+1, PP_FORMAT, pid, msgcount, banner);

    va_start(ap, format);
    vsnprintf(output_line + PP_PREFIX_LEN,
                PP_MAX_LINE_LEN - PP_PREFIX_LEN, format, ap);
    va_end(ap);

    printf("%s", output_line);
    fflush(stdout);
    msgcount++;
    return 0;
}
```