Midterm Part I of III

Adam Ross
adro4510@colorado.edu

Abstract

This report implements are proves the correctness of Conway's Game of life automata algorithm. First the single core serial version is implemented with the ability to write out to pgm files to form animation frames. The alive count is then compared to known values and proved to be the correct calculation. Then synchronous and asynchronous MPI parallel implementation are added and compared against the known output for correctness. These algorithms are then run of one the super computing platforms we have access on various iteration sizes and processor counts. These include 1000 and 10000 iteration counts and 4, 9, 25, and 36 processor counts.

1. Introduction

Conway's game of life is a cellular automata requiring many iterations on possibly large data sets that requires each iteration to look at every cell in the 2d environment. The algorithm is O(n^2) and is a prime algorithm to benefit from parallelism, hence it is worth investigating. These initial implementation steps are a precursor in this three part exam to more timing and analysis work. I will first be discussing my implementations of Conway's and the correctness of each respectively. I will then provide the various processor and iteration count data.

2. Overview

When developing this program most choices were made in favor of basic C optimization such as cache hit optimization or instruction optimization. I explain in detail below. I shall start by discussing basic data type usage. In favor of loading more data in cache and also the limited requirements to represent our data the two dimensional contiguous environment arrays used in my code are char array. This is advantageous to limit the amount of cache misses when iterating over our 3x3 stamp. I used Comet to run the various iteration counts and processor counts and in the case of Comet we have a 30MB shared L3 cache, individual 256KB L2 cache and 32KB L1 cache. Given that a char is a single Byte and that our sample matrix is 900x900 we would have to have a 810KB cache to store the entire thing. We can however store 35 rows at a time in the 32KB L1 cache using chars, whereas using integers would only get us 8 rows and more inter cache movement.

To actually calculate and do Conway's algorithm no loops were used, i.e. loop unrolling, and operations that used similar spots in memory were grouped together as much as possible. The code detailing this is outline in Figure 2.1 below.

```
for (i = 1; i < local_height + 1; i++) {
    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];

        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
        }
    }
}
```

Figure 2.1 - Conway’s game of life algorithm code.

Next I will outline the ghost rows I used for exchanging information across memory chunks and processors. For the purpose of simplicity and the ability to reuse my code for later midterm parts I put a row and column around every side of each memory-array chunk to buffer interactions between processes. In the case of this midterm part we are require to implement row block data decomposition, meaning we will be exchanging data only vertically. With that in mind my implementation for passing the relevant information into each process’s ghost row is outlined below in Figure 2.2.

```
// Send to below or recv from above
MPI_Sendrecv(&env_a[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, top_dest, 0,
             &env_a[(field_height - 1) * field_width + 0], field_width,
MPI_UNSIGNED_CHAR, top_source, 0, MPI_COMM_WORLD, &status);

// Send to above or recv from below
MPI_Sendrecv(&env_a[(field_height - 2) * field_width + 0], field_width,
MPI_UNSIGNED_CHAR, bot_dest, 0,
             &env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_source,
0, MPI_COMM_WORLD, &status);
```

Figure 2.2 - Ghost row exchange synchronous code.

Additionally we are required to use asynchronous MPI_Send and MPI_Recv calls to make things easier and allow processing of data during message passing. In my implementation I send the updated environment B information to the appropriate ghost rows as soon as it is calculated, meanwhile our processor if free to count the living if called to do so, output a file or any other sort of work. Mixed pseudo code below outlines this process.

```
<Start algorithm>

<calculate destinations and sources for ghost row exchange>
```

```
// Initial exchange for N = 0 to N = 1
MPI_ISend(env_a data, row_length, MPI_CHAR, top_dest, tag, MPI_COMM_WORLD, &request);
MPI_ISend(env_a data, row_length, MPI_CHAR, bottom_dest, tag, MPI_COMM_WORLD,
&request);

<begin algorithm>
    <calculate destinations and sources for ghost row exchange>

    MPI_IRecv(env_a data, row_length, MPI_CHAR, top_source, tag, MPI_COMM_WORLD,
    &request);
    MPI_IRecv(env_a data, row_length, MPI_CHAR, bottom_source, tag, MPI_COMM_WORLD,
    &request);

    <calculate N + 1 state>

    // send the data we just calculated as soon as we know it
    MPI_ISend(env_b data, row_length, MPI_CHAR, top_dest, tag, MPI_COMM_WORLD,
    &request);
    MPI_ISend(env_b data, row_length, MPI_CHAR, bottom_dest, tag, MPI_COMM_WORLD,
    &request);

    <print to file if need be>
    <count if need be>
```

Figure 2.3 - Asynchronous ghost row exchange pseudocode.

My pgm file reader was just the implementation provided to use with minor tweaks. Why reinvent the wheel?

For testing purposes I used gimp to create a glider in the top left corner of a 16x16 pgm image. This was simple enough to debug using a process count of 2 during synchronous and asynchronous debugging. Additionally I used gimp to make a slightly larger file at 80x80 that contained a glider gun that was useful to debug for process counts > 2.

To debug I used the above images and a bit of counting and printf statements. I would like to have used valgrind, but once again it is not supported on MacOS 10.12. This can be solved by locating another machine or loading debian onto a spare hard drive.

3. Verification

Below is a table containing the various output information from the serial, synchronous and asynchronous implementations.

N	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Serial	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953

Sync	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953

Figure 3.1 - Serial and Synchronous to 10000, np = 9 counts.

Method - np	Sync - 4	Sync - 9	Sync - 25	Sync - 36	Async - 4	Async - 9	Async - 25	Async - 36
Alive at 1000	18340	18340	18340	18340	18340	18245	18289	18342

Figure 3.2 - Asynchronous and Synchronous Counts for varying np.

Given that my Asynchronous numbers were off there must be a miscounting or communication issue. This will be a problem I must fix before Part II.

4. Conclusion

It clear from my timing information that my asynchronous implementation is lacking somewhere. It will be interesting to time these implementations given that I fix my asynchronous implementation, though I can figure what they will look like. Programming conclusions for this exercise are that I should be more careful checking my array/loop boundaries and to be sure of exactly what you are sending in scatter or gather code.

```
// 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"

#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                     local_width;    // Width and height of game on this processor
int                     local_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)
int                     awidth;
int                     aheight;
unsigned char           *env_a;
unsigned char           *env_b;
unsigned char           *out_buffer;

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

extern int              local_width;
extern int              local_height;
extern int              N;

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

#endif
```

helper.h

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

void print_usage();
void print_matrix(unsigned char *matrix);
void swap(unsigned char **a, unsigned char **b);
unsigned char *Allocate_Square_Matrix();
int count_alive(unsigned char *matrix);
```

```
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 *, ...);
```

```
CC = mpicc
CCFLAGS = -g -Wall -std=c99
ifeq ($(DEBUG),on)
        CCFLAGS += -DDEBUG
endif

C_FILES = RossAdam_MT1.c pgm.c pprintf.c helper.c
O_FILES = RossAdam_MT1.o pgm.o pprintf.o helper.o

all: RossAdam_MT1

RossAdam_MT1: $(O_FILES)
        $(CC) -o RossAdam_MT1 $(O_FILES) $(LDFLAGS)

.PHONY: clean
clean:
        /bin/rm -f core $(O_FILES) RossAdam_MT1

RossAdam_MT1: pgm.o pprintf.o helper.o

.c.o:
        $(CC) $(CCFLAGS) -c -o $*.o $*.c


# All of the object files depend on the globals, so rebuild everything if they
# change!
*.o: globals.h

# Nothing really depends on the pprintf prototypes, but just be safe
*.o: pprintf.h

*.o: helper.h

# Conway depends on PGM utilities
RossAdam_MT1.o: pgm.h pprintf.h helper.h
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "globals.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 * awidth + 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 *) malloc(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;
}
```

```c
/*
 * 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"

// 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, y;
    int             start_x, start_y;
    int             b, lx, ly, ll;

    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
    char header[10];
    int width, height, depth;
    int rv = fscanf( fp, "%6s\n%i %i\n%i\n", header, &width, &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 (rank == 0)
        pprintf( "%s: %s %i %i %i\n", filename, header, width, 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 (width % ncols) {
        if (rank == 0)
            pprintf( "Error: %i pixel width cannot be divided into %i cols\n", width, ncols
);
        return false;
    }
    if (height % nrows) {
        if (rank == 0)
            pprintf( "Error: %i pixel height cannot be divided into %i rows\n", height, nrow
s );
        return false;
    }

    // Divide the total image among the local processors
    local_width = width / ncols;
    local_height = height / nrows;

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

    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;

    // Total width for pgm animation and iterating
    awidth = ncols * field_width;
    aheight = nrows * field_height;
    pprintf( "Gather matrix x:%d y:%d\n", awidth, aheight);

    // 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);

    // Read the data from the file. Save the local data to the local array.
    for (y = 0; y < height; y++) {
        for (x = 0; x < 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 + local_width && y >= start_y && y < start_y + l
ocal_height) {
                // Calculate the local pixels (+1 for ghost row,col)
                lx = x - start_x + 1;
                ly = y - start_y + 1;
                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;
}
```

```
/* $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;
}
```

```
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;
}
```

```
/* 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"

typedef enum { SERIAL, ROW, BLOCK } dist;

int main(int argc, char* argv[]) {
    unsigned short      i, j;
    unsigned short      neighbors =         0;
    int                 top_dest =          5280;
    int                 top_source =        5280;
    int                 bot_dest =          5280;
    int                 bot_source =        5280;
    MPI_Status          status;
    MPI_Request         rq, qr;
    int                 counting =          -1;
    int                 count =             0;
    int                 total =             0;
    int                 n =                 0;
    int                 option =            -1;
    dist                dist_type;
    bool                async =             false;
    int                 iter_num =          1000;
    char                *filename;
    char                frame[47];

    // Parse commandline
    while ((option = getopt(argc, argv, "d:sn:c:i:")) != -1) {
        switch (option) {
             case ’d’ :
                 dist_type = atoi(optarg);
                 break;
             case ’s’ :
                 async = true;
                 break;
             case ’n’ :
                 iter_num = atoi(optarg);
                 break;
             case ’c’ :
                 counting = atoi(optarg);
                 break;
             case ’i’ :
                 filename = 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");

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

    //
    // Determine the partitioning
    //
    if (!dist_type || dist_type == 1) {
        if (!rank)
            pprintf("Row distribution selected.\n");
        ncols = 1;
        nrows = np;
        my_col = 0;
        my_row = rank;
    }

    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;
    }

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

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

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

    // Perform initial exhange to calculate 0 and 1 states
    if (async && dist_type == 1) {
        top_dest = bot_source = rank - 1;
        top_source = bot_dest = rank + 1;
        if (!rank) {
            top_dest = MPI_PROC_NULL;
            bot_source = MPI_PROC_NULL;
        } else if (rank == (np - 1)) {
            top_source = MPI_PROC_NULL;
            bot_dest = MPI_PROC_NULL;
        }

        MPI_Isend(&env_a[1 * field_width + 0], field_width, MPI_CHAR, top_dest, 0, MPI_COMM_
WORLD, &rq);
        MPI_Isend(&env_a[(field_height - 2) * field_width + 0], field_width, MPI_CHAR, bot_d
est, 0, MPI_COMM_WORLD, &qr);
    }

    while(n < iter_num) {
        // sync or a async here MPI_PROC_NULs
        if (dist_type == 1) { // row distro
            // calculate pairings
            top_dest = bot_source = rank - 1;
            top_source = bot_dest = rank + 1;
            if (!rank) { // 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;
            }

            if (!async) {
                // Send to below or recv from above
                MPI_Sendrecv(&env_a[1 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, to
p_dest, 0,
                             &env_a[(field_height - 1) * field_width + 0], field_width, MPI_
UNSIGNED_CHAR, top_source, 0, MPI_COMM_WORLD, &status);
                // Send to above or recv from below
                MPI_Sendrecv(&env_a[(field_height - 2) * field_width + 0], field_width, MPI_
UNSIGNED_CHAR, bot_dest, 0,
                             &env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bo
t_source, 0, MPI_COMM_WORLD, &status);

            } else { // Aschrnous enabled, receive from the last iteration or inital setup
                MPI_Irecv(&env_a[(field_height - 1) * field_width + 0], field_width, MPI_CHA
R, top_source, 0, MPI_COMM_WORLD, &rq);
                MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_CHAR, bot_source, 0,
 MPI_COMM_WORLD, &qr);
                // To avoid getting data mixed up wait for it to come through
                MPI_Wait(&rq, &status);
                MPI_Wait(&qr, &status);
            }
        } // else block distro

        // Uncomment to produce pgm files per frame
        /*MPI_Gather(env_a, field_width * field_height, MPI_CHAR, out_buffer, field_width *
field_height, MPI_CHAR, 0, MPI_COMM_WORLD);

        if (rank == 0) {
           for (int k = 0; k < aheight; k++) {
               for (int a = 0; a < awidth; a++) {
                   if (!out_buffer[k * awidth + a]) {
                       out_buffer[k * awidth + a] = 255;
                   } else {
                       out_buffer[k * awidth + a] = 0;
                   }
               }
           }

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

        // calulate neighbors and form state + 1
        for (i = 1; i < local_height + 1; i++) {
            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_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 we are doing async we now have the data we need for the next iter, send it
        if (async && dist_type == 1) {
            MPI_Isend(&env_b[1 * field_width + 0], field_width, MPI_CHAR, top_dest, 0, MPI_C
OMM_WORLD, &rq);
            MPI_Isend(&env_b[(field_height - 2) * field_width + 0], field_width, MPI_CHAR, b
ot_dest, 0, MPI_COMM_WORLD, &qr);
        }

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

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

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

    // 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_Allreduce(&count, &total, 1, MPI_INT, MPI_SUM, 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 );

    MPI_Finalize();


} /* end main */
```