Midterm Part II of III - MPI IO & Grid Distribution

Adam Ross
adro4510@colorado.edu

Abstract

This project report addresses the implementation and correctness of my grid implementation of Conway's game of Life for both synchronous and asynchronous communication and parallel file output via MPI IO. To prove the correctness of my implementations I will be comparing the living bug counts for various processor counts and iteration counts to known values. These include 1000 and 10000 iteration counts and 4, 9, 25, and 36 processor counts, and the compared data will come from part one of this midterm. In addition I will be using code snippets to prove both the grid implementation and the MPI IO implementation. I will then detail scripts used to speed up repetitive development processes when developing on the computing cluster Comet.

1. Introduction

When processing large chunks of data it is useful to split this data inter smaller sets for processing. Building on the last midterm portion, which implemented row decomposition, we now look at a more scaleable solution of grid decomposition. We find value in the method with data sets that may expand as n^2, as a row decomposition would maintain the full width of the data.

Additionally, when moving and generating enormous sets of data, as is done on large computing clusters, it would be an extreme bottleneck to be doing serial IO to disk. Given this problem there is motivation for parallel file systems and IO operations. MPI IO provides a library to accomplish this much, and it is worthwhile to investigate these methods.

I first examine code snippets showing the respective methodologies, then I examine the output and bug count across a plethora of input permutations.

2. Overview

To implement a grid distribution we are more or less adding a second dimension to our parallel processing and data decomposition. First I will provide pseudo code for my row and column process assignment. This is detailed below in figure 2.1.

```
nrows = (int)sqrt(np);
ncols = (int)sqrt(np);
my_row = rank / nrows;
my_col = rank - my_row * nrows;
```

Figure 2.1 - Row and column process assignment.

Here we are assuming, and checking later, that we will have a square processor count where the data distributes evenly across all processes. This is checked in later code and returned if false.

It is then useful to define a column data type to pass out relevant column data to the vertical padding on surrounding each process's data set. This is done via the MPI vector datatype, where we pass only our local height as to implement two stage grid passing. Two stage passing details passing column data with local height padding horizontally and field width padding vertically allowing the passage of corner data. This allows only four communications per block as opposed to eight; one for each side opposed to one for each side and corner. This is outlined in figure 2.2 and used in figure 2.3.

```
          // 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);
```

Figure 2.2 - Column data type detail.

```
// Send to right or recv from left
MPI_Sendrecv(&env_a[1 * field_width + 1], 1, column, left_dest, 0,
             &env_a[2 * field_width - 1], 1, column, left_source, 0,
             MPI_COMM_WORLD, &status);
// Send to left or recv from right
MPI_Sendrecv(&env_a[2 * field_width - 2], 1, column, right_dest, 0,
             &env_a[1 * field_width + 0], 1, column, right_source, 0,
             MPI_COMM_WORLD, &status);

// 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 2.3 - Horizontal - vertical two-stage communication - synchronous.

Similarly to the synchronous code the asynchronous code also uses two stage communication. This, not unlike the asynchronous code from midterm part I, was also spread around various other pieces of computational work to allow processing in between and during communication. The process is outlined below in pseudo code.

```
<Start algorithm>

<calculate destinations and sources for padding exchanges>
// Initial exchange for horizontal communication
MPI_ISend(env_a data, row_length, MPI_CHAR, left_dest, tag, MPI_COMM_WORLD, &request);
MPI_ISend(env_a data, row_length, MPI_CHAR, right_dest, tag, MPI_COMM_WORLD,
&request);
```

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

       MPI_IRecv(env_a data, row_length, MPI_UNSIGNED_CHAR, right_source, tag,
       MPI_COMM_WORLD, &request);
       MPI_IRecv(env_a data, row_length, MPI_UNSIGNED_CHAR, left_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, 1, column, left_dest, tag, MPI_COMM_WORLD, &lr);
       MPI_Isend(env_b data, 1, column, right_dest, tag, MPI_COMM_WORLD, &rr);


       <print to file if need be>
       <count if need be>
       <any other work>

           MPI_Irecv(env_b data, 1, column, left_source, tag, MPI_COMM_WORLD,
       &request);
           MPI_Irecv(env_b data, 1, column, right_source, tag, MPI_COMM_WORLD,
       &request);
           // Need the horizontal data before we send vertically
           MPI_Wait(&lr, &status);
           MPI_Wait(&rr, &status);

           MPI_Isend(env_b data, field_width, MPI_UNSIGNED_CHAR, top_dest, tag,
       MPI_COMM_WORLD, &request);
           MPI_Isend(env_b data, field_width, MPI_UNSIGNED_CHAR, bot_dest, tag,
       MPI_COMM_WORLD, &request);
```

Figure 2.4 - Asynchronous ghost row exchange pseudocode.

To accomplish the parallel IO required we first need to build two custom MPI datatypes; a distributed array and an array-padding extraction vector. Both are outlined below in figure 2.5.

```
// Create darray and commit
MPI_Type_create_darray(np, rank, 2, gsizes, distribs, dargs, psizes, MPI_ORDER_C,
        MPI_UNSIGNED_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);
```

Figure 2.5 - MPI datatypes to distribute an array and to extract the data we want from our sub arrays.

In order to write only the data we care about to file we create a vector data type where the stride is the whole data width, the data chunk size is the useful data width and that we repeat for the row count of the relevant data. The distributed array knows how many processes we have, this process's rank, the global array size, and how we distribute this data. This is used when telling each process exactly where to write its chunk of the data into the current file. These two data types are then combined with the MPI_File_set_view and MPI_File_write calls to write our data out onto the parallel file system available on Comet. The important file writing code is included below in figure 2.6. Notice we write the pgm file header before writing the Conway data.

```
	MPI_File_open(MPI_COMM_WORLD, frame, MPI_MODE_CREATE|MPI_MODE_WRONLY,
MPI_INFO_NULL, &out_file);

	char header[15];
	sprintf(header, "P5\n%d %d\n%d\n", global_width, global_height, 255);
	int header_len = strlen(header);

	//write header
	MPI_File_set_view(out_file, 0,  MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "native",
MPI_INFO_NULL);
	MPI_File_write(out_file, &header, 13, MPI_UNSIGNED_CHAR, MPI_STATUS_IGNORE);

	// write data
	//MPI_File_set_view(out_file, 15 + rank * local_width + local_width,
MPI_UNSIGNED_CHAR, darray, "native", MPI_INFO_NULL);
	MPI_File_set_view(out_file, 13, MPI_UNSIGNED_CHAR, darray, "native",
MPI_INFO_NULL);

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

Figure 2.6 - MPI IO set view and write calls.

In order to debug and develop faster several scripts were developed. The first I shall mention is a comet remake script, which made developing MPI IO many times less tedious. This script is detailed below in figure 2.7.

```
git pull && make clean && make && cp RossAdam_MT2 ../../bin && rm -f
/oasis/scratch/comet/adamross/temp_project/* && sbatch batch_files/RossAdam_testing.sh
&& watch squeue -u adamross && cat "dev/comet_out/$(ls -1rt dev/comet_out/ | tail
-n1)"
```

Figure 2.7 - Comet remake script. Pulls down the latest code, re-makes it, submits a testing batch file. Watches the queue, and cats the output when we exit the watch command.

Additionally to support the development and debugging on the grid distribution an animation script was created seen below in figure 2.8. These scripts reduced redundant keystroke operations many times over.

This is something I have experienced at work as well. If you need to perform a task over and over it will be well worth the time to script it up.

```
cp /oasis/scratch/comet/adamross/temp_project/* dev/data/. && for f in `ls -1
dev/data`; do xxd -p -c 16 ${f}; sleep 0.5; done
```

Figure 2.8 - Comet anim script. This script copies the parallel output into the local directory and pops out a terminal visualization of the frames produced.

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-Row	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953
Async-Row	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953
Sync-Grid	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953
Async-Grid	25301	18340	16512	16001	15449	14953	14953	14953	14953	14953	14953

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

Row Distribution

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	18340	18340	18340

Figure 3.2 - Asynchronous and Synchronous Counts for Row distribution varying np.

Grid Distribution

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	18340	18340	18340

Figure 3.3 - Asynchronous and Synchronous Counts for Grid distribution varying np.

In my last report I had seen some strange numbers coming back from my async communication. I had also seen this when debugging this time, which I came to find was that I had not completely initialized my whole array to 0s initially, which in some cases was producing strange numbers.

4. Conclusion/Learned

Given the nature of asynchronous communication, which allows us to do computation while running communication through the pipeline I would expect a reasonable performance increase. The limitation here being that we cannot be sending while actually iterating over the array. We need the current data to do the array calculations and can only send them once we have done them, hence there cannot be any asynchronous communication per process while we do the main chunk of our work.

```
// 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                     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)
int                     awidth;             // width of global array + padding
int                     aheight;            // height of global array + padding
unsigned char           *env_a;             // 1D character array to represent our 1st 2D en
vironment
unsigned char           *env_b;             // 1D character array to represent our 2nd 2D en
vironment
unsigned char           *out_buffer;        // 1D character array to represent our global 2D
 environment + padding

#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              global_width;
extern int              global_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
```

```c
/*
 * 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);
```

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

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

C_FILES = RossAdam_MT2.c pgm.c pprintf.c helper.c
O_FILES = RossAdam_MT2.o pgm.o pprintf.o helper.o

all: RossAdam_MT2

RossAdam_MT2: $(O_FILES)
	$(CC) -o RossAdam_MT2 $(O_FILES) $(LDFLAGS)

.PHONY: clean
clean:
	/bin/rm -f core $(O_FILES) RossAdam_MT2

RossAdam_MT2: 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_MT2.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 tur
n 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");
}

/*
 * 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;
}
```

```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;
    char            header[10];
    int             depth;
    int             rv;

    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 (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;

    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 < 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 + 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;
}
```

```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;
}

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,
                        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;
    dist                dist_type;
    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;


    // Parse commandline
    while ((option = getopt(argc, argv, "d:sn:c:i:w")) != -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;
             case 'w' :
                 writing = true;
                 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 < 2) {
        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 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(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) {
        if (rank == 0) {
            pprintf("Asynchronous communication starting\n");
        }
        if (dist_type == 1) {
            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;
            }
        } else if (dist_type == 2) {
        // 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);
        }

        // 2 step communication methodology as detailed on the moodle and by Michael
        if (dist_type == 2) {
            // Send horizontal communication first of height: local_height
            MPI_Isend(&env_a[1 * field_width + 1], 1, column, left_dest, 0, MPI_COMM_WORLD,
&lr);
            MPI_Isend(&env_a[2 * field_width - 1], 1, column, right_dest, 0, MPI_COMM_WORLD,
 &rr);

            MPI_Irecv(&env_a[2 * field_width - 2], 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);
```

```
            // Need the horizontal data before we send vertically
            MPI_Wait(&lr, &status);
            MPI_Wait(&rr, &status);
        }
        // Send vertical communication of width: field_width
        // This is applicable for both row and block distrobutions
        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_UNSIGNED_CH
AR, bot_dest, 0, MPI_COMM_WORLD, &br);
    }

    while(n < iter_num) {
        // sync or a async here MPI_PROC_NULs
        if (dist_type > 0) {
            // calculate pairings
            if (dist_type == 1) { // 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 == 2) {
            // 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_des
t, left_dest, right_dest, MPI_PROC_NULL);
            }

            if (!async) {
                // If we choose block decomposition send horizontally first
                if (dist_type == 2) {
                    // Send to right or recv from left
                    MPI_Sendrecv(&env_a[1 * field_width + 1], 1, column, left_dest, 0,
                                 &env_a[2 * field_width - 1], 1, column, left_source, 0, MPI
_COMM_WORLD, &status);
                    // Send to left or recv from right
                    MPI_Sendrecv(&env_a[2 * field_width - 2], 1, column, right_dest, 0,
                                 &env_a[1 * field_width + 0], 1, column, right_source, 0, MP
I_COMM_WORLD, &status);
                }
                // 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_UNS
IGNED_CHAR, top_source, 0, MPI_COMM_WORLD, &ar);
                MPI_Irecv(&env_a[0 * field_width + 0], field_width, MPI_UNSIGNED_CHAR, bot_s
ource, 0, MPI_COMM_WORLD, &br);
                // 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 (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 we are in row distrobution send vertically - thats all we need to do
        // If we are in block distrobution send horizontally first
        if (async && dist_type == 1) {
            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 == 2) {
            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);
        }
```

```c
        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[15];
            sprintf(header, "P5\n%d %d\n%d\n", global_width, global_height, 255);
            int header_len = strlen(header);

            //write header
            MPI_File_set_view(out_file, 0,  MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, "native",
MPI_INFO_NULL);
            MPI_File_write(out_file, &header, 13, 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, 13, MPI_UNSIGNED_CHAR, darray, "native", MPI_INFO_NU
LL);

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

        // Uncomment to produce pgm files per frame
        /*MPI_Gather(env_b, field_width * field_height, MPI_UNSIGNED_CHAR, out_buffer, field
_width * field_height, MPI_UNSIGNED_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);
       }*/

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

            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);
            }
        }

        // Receive our horizontal communication and send the vertical
        if (async && dist_type == 2) {
            MPI_Irecv(&env_b[2 * field_width - 1], 1, column, left_source, 0, MPI_COMM_WORLD
, &lr);
            MPI_Irecv(&env_b[1 * field_width + 0], 1, column, right_source, 0, MPI_COMM_WORL
D, &rr);
            // Need the horizontal data before we send vertically
            MPI_Wait(&lr, &status);
            MPI_Wait(&rr, &status);

            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);
        }

        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 */
```