Assignment 3: Ocean Parallel Fishing Problem

HPC
Autumn 2014

Students:
Hannes Pétur Eggertsson
Klemenz Hrafn Kristjánsson
Jeffrey Robert Hair

Teacher:
Morris Riedel

1 Ocean Parallel Fishing Problem

We decided to use a 3×3 cartesian grid for the ocean and let 9 cores each take care of one square. At the start of the simulation we put two boats on the grid at ranks 0 and 5 (i.e. where `mod`($rank$,5) is 0) and at every square there is a random number of fish, ranging between 0 and 20 per square. Each square stores the following information:

- If the square has a boat.
- If the square has a net.
- How much room for fish is in the net?
- How many fish are in my square?
- Which rank am I?
- x and y coordinates on the grid.
- Who is above, below, to the left, and to the right of me?

At each iteration the nets are filled until they are filled with fish and then the fish can swim around to other squares randomly. If a fish is caught it is removed from the number of fish in the square and added to the number of fishes in the net. When the fishes are swimming around we use non-blocking communication (`MPI_Isend` and `MPI_Irecv`) between squares but the nets can start filling up we need to make sure to wait for all communication to finish by using `MPI_Waitall`. While the nets are filling up the captain of the boats can talk to each other, and once all nets are full the captains stop talking. Logging IO is implemented in the code and could be used using the `writeLine` function defined in the `logging.c` file but it would often cause some strange file system errors on Jotunn, so instead we're only using `printf` statements.

2 Example run

Here we specify the fishing net capacity as 5 fishes per net. Except for the starting conditions only messages from ranks 0 and 5 (the ones with nets) are shown until they've filled their net.

```
(0, 0) Starting with 5 fish
(0, 1) Starting with 20 fish
(0, 2) Starting with 19 fish
(1, 0) Starting with 17 fish
(1, 1) Starting with 19 fish
(1, 2) Starting with 14 fish
(2, 0) Starting with 3 fish
(2, 1) Starting with 15 fish
(2, 2) Starting with 2 fish
5 (1, 2) received 19 fish, sent 14 fish, now the fish are 19
I'm the rank 5, I'm catching fish with my net. Left to fill net 5
5 (1, 2) I'm finishing!!! Caught all 5 fish
0 (0, 0) received 0 fish, sent 5 fish, now the fish are 0
0 (0, 0) My net is not full! Still room for 5 more fish
|Dance with me!||              ||              ||              ||              ||              |
0 (0, 0) received 0 fish, sent 0 fish, now the fish are 0
0 (0, 0) My net is not full! Still room for 5 more fish
|Mambo number 5||              ||              ||              ||              ||              |
0 (0, 0) received 27 fish, sent 0 fish, now the fish are 27
I'm the rank 0, I'm catching fish with my net. Left to fill net 5
0 (0, 0) I'm finishing!!! Caught all 5 fish
|              ||              ||              ||              ||              ||              |
```

As we can see rank 5 can fill their net in the very first timestep but for rank 0 all fish swim away so there are no fish to fill the net. At the 3rd timestep rank 0 receives fish and fills its net. Then the captains have all stoped talking and its time to empty the net onto the boat.

3 Full code

The code is available here: https://github.com/ProjectMoon/ocean but will also be pasted here below:

ocean.c

```
#include <mpi.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include "ocean.h"

//Defines: event types for various oceanic communication
#define EVENT_FISH 0
#define EVENT_BOAT 1

//Other helpful macros
#define ARR_LEN(arr) ( sizeof(arr) / sizeof(arr[0]) )

int main (int argc, char** argv) {
    int rank, size;
        int x_size = 3;
        int y_size = 3;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm grid;

    create_communicator(MPI_COMM_WORLD, &grid, x_size, y_size);
    MPI_Comm_rank(grid, &rank);

    work(rank, size, grid);

    MPI_Finalize();
    return 0;
}

void create_communicator(MPI_Comm input, MPI_Comm *comm, int x, int y) {
    //number of processors required is x * y
    int num_dims = 2;
    int dims[2] = { x, y };
    int periods[2] = { 0, 0 };
    int reorder = 0;

    MPI_Cart_create(input, num_dims, dims, periods, reorder, comm);
}

int get_adjacent(grid_square square, direction_t dir) {
    switch (dir) {
    case DIR_LEFT:
        return square.left;
    case DIR_RIGHT:
        return square.right;
    case DIR_UP:
        return square.up;
```

```c
    case DIR_DOWN:
        return square.down;
    default:
        return MPI_PROC_NULL;
    }
}

int get_opposite(grid_square square, direction_t dir) {
    switch (dir) {
    case DIR_LEFT:
        return square.right;
    case DIR_RIGHT:
        return square.left;
    case DIR_UP:
        return square.down;
    case DIR_DOWN:
        return square.up;
    default:
        return MPI_PROC_NULL;
    }
}

void work(int rank, int size, MPI_Comm grid) {
    //Where am I?
    int coords[2];
    MPI_Cart_coords(grid, rank, 2, coords);

    //Get neighbours. dim 0 = columns, dim 1 = rows
    int left, right, up, down;
    MPI_Cart_shift(grid, 0, 1, &up, &down);
    MPI_Cart_shift(grid, 1, 1, &left, &right);

    //Randomly generate a number of fish (0 to 20). Modulus slightly
    //reduces randomness, but whatever.
    srand(time(NULL) + rank);
    int fish = rand() % 21;

    grid_square square = {
                .has_boat = (rank % 5 == 0),
        .has_net = (rank % 5 == 0), //every 5th square has a net
                .fish_leftToFillNet = 1,
        .fish = fish,
        .rank = rank,
        .x = coords[0],
        .y = coords[1],
        .left = left,
        .right = right,
        .up = up,
        .down = down
    };

        printf("(%d, %d) Starting with %d fish\n", square.x, square.y, square.fish);
        sleep(1);
        int finish = 0;
    for (int i = 0; i < 10; i++) {
                //printf("%d %d I'm in the for loop, iteration %d \n", square.rank,
                    rank, i);
        finish = simulation_step(grid, &square, size);
        sleep(1);
                // if (finish == 1) {
                        // printf("---I'm %d and I want to stop the
                            iteration.---\n",square.rank);
                // }
    }
}

int simulation_step(MPI_Comm grid, struct grid_square* square, int size) {
    //send fish swimming and receive incoming fish from neighbours
    direction_t dir = rand() % 5;
    int fish_dest = get_adjacent(*square, dir);

    MPI_Request reqs[8]; //4 send, 4 receive.
    MPI_Status statuses[8];

    int fish_arrived[4] = { 0 }; //for each incoming direction.
        int total_fish_sent = 0;

    for (int c = 0; c < 4; c++) {
        //if we are currently sending to the chosen node of the fish,
        //then we send them off. otherwise no fish go there.
        int fish_swimming, destination;
        destination = get_adjacent(*square, c);

        if (destination == fish_dest && fish_dest != MPI_PROC_NULL) {
            fish_swimming = square->fish;
                        total_fish_sent += fish_swimming;
        }
        else {
            fish_swimming = 0;
        }

        MPI_Isend(&fish_swimming, 1, MPI_INT, destination, EVENT_FISH, grid, &reqs[c]);

        //get fish arrivals from the opposite direction.
        int opposite;
        opposite = get_opposite(*square, c);
        MPI_Irecv(&fish_arrived[c], 1, MPI_INT, opposite, EVENT_FISH, grid, &reqs[c + 4]);
    }

    MPI_Waitall(8, reqs, statuses);

    if (fish_arrived > 0) {
        int total_fish_arrived = 0;

        for (int c = 0; c < ARR_LEN(fish_arrived); c++) {
                        total_fish_arrived += fish_arrived[c];
                }
                square->fish += total_fish_arrived;
                square->fish -= total_fish_sent;
                printf("%d (%d, %d) received %d fish, sent %d fish, now the fish are %d\n", square->rank, square->x, square->y, total_fish_arrived, total_fish_sent, square->fish);
    }

        //Put fish in square into the net.  If net is full, remove boat and net
        if ((square->has_net == true) && (square->fish > 0)) {
                printf("I'm the rank %d, I'm catching fish with my net. Left to fill net %d \n", square->rank, square->fish_leftToFillNet);
                if (square->fish_leftToFillNet > square->fish) {
                        square->fish_leftToFillNet -= square->fish;
                        square->fish = 0;
                }
                else {
                        printf("%d (%d, %d) I'm finishing!!! Caught all %d fish\n", square->rank, square->x, square->y, square->fish_leftToFillNet);
                        square->fish -= square->fish_leftToFillNet;
                        square->fish_leftToFillNet = 0;
                        //printf("%d Fish in square at end of timestep: %d\n", square->rank, square->fish);
                        square->has_net = false;
                        square->has_boat = false;
                        }
                }

                if (square->has_net == true) {
                        if (square->fish_leftToFillNet == 0) {
                                printf("%d (%d, %d) My net is full! \n", square->rank, square->x, square->y);
                        }
                        else {
                                printf("%d (%d, %d) My net is not full! Still room for %d more fish \n", square->rank, square->x, square->y, square->fish_leftToFillNet);
                        }
                }

                int msgLength = 16;
                char *messages[6] = {"|LOUD NOISES!!!|", "|I'm on a boat!|", "|Dance with me!|", "|Mambo number 5|", "|I like flowers|", "|Sail with me..|"};
                char *my_message = "|              |";
                int randmsg;
                if(square->has_boat) {
                        randmsg = rand() % 5;
                        my_message = messages[randmsg];
                }
                char *r_msgs = (char *)malloc(sizeof(char) * size*msgLength);
                assert(r_msgs != NULL);
                MPI_Allgather(my_message, msgLength, MPI_CHAR, r_msgs, msgLength, MPI_CHAR, MPI_COMM_WORLD);
                if(square->rank==0) {
                        printf("%s \n",  r_msgs);
                }
                int stillBoats = 1;
                if((strcmp(r_msgs,"|              ||              ||              ||              ||              ||              ||              ||              ||              |") == 0)) {
                        printf("All messages are empty (all nets full)! \n");
                        stillBoats = 0;
                }
                free(r_msgs);
                if (stillBoats == 1) {
                        return 0;
                }
                else {
                        return 1;
                }
}
```

logging.c

```
#include <stdio.h>
#include "logging.h"

/* Declarations */
MPI_Info info;
/* MPI_Status status; */
int size, rank;
MPI_File fh;
MPI_Status status;

void initLogFile(int rank,int size) {
    char *file_name = "logFile.txt";
    char buf[80];
    MPI_Info_create(&info);
    //printf("Rank %d of %d has initialized their log file.\n", rank, size);
    MPI_File_open( MPI_COMM_WORLD, file_name, MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh);
    sprintf(buf,"I’m %d of %d and I’m writing to the log\n",rank,size);
    MPI_File_write_ordered(fh, buf, strlen(buf), MPI_CHAR, &status);
}

void closeLogFile() {
    MPI_File_close(&fh);
        // printf("%d/%d closed their chunk of the log file.\n", rank, size);
}

void writeLine(char *line) {
    MPI_File_write_ordered(fh, line, strlen(line), MPI_CHAR, &status);
}
```