TÉCNICO LISBOA

Parallel and Distributed Computing

Project Assignment

GAME OF LIFE 3D

Version 1.0 (10/03/2017)

2016/2017
2nd Semester

Contents

Revisions

Version 1.0 (March 10, 2017)	Initial Version

1 Introduction

The purpose of this class project is to gain experience in parallel programming on UMA and multicomputer systems, using OpenMP and MPI, respectively. For this assignment you are to write a sequential and two parallel implementations of the game of life in 3D.

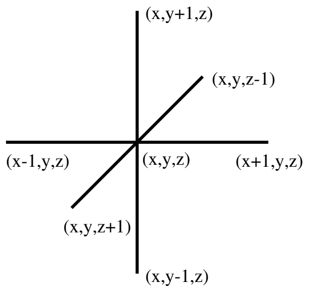
2 Problem Description

The life game is a zero-player game, that starts from an initial distribution of individuals in the game space. The game evolves by computing new generations of individuals based on simple rules.

For this assignment we consider a three-dimensional space, more specifically a cube. Each cell of this cube may be occupied (live) or empty (dead). In order to determine the status of a cell in the next generation, the following rules should be used:

- a live cell with fewer than two live neighbors dies.
- a live cell with two to four live neighbors lives on to the next generation.
- a live cell with more than four live neighbors dies.
- a dead cell with two or three live neighbors becomes a live cell.

We consider as neighbors only the cells on the sides of the a cell, not cells in the diagonal, as illustrated in the figure below:



Note that it is considered that the sides wrap around, that is, a cell with a coordenate 0 is a neighbor of the cell with coordinate *size-1*, if the other two coordinates are the same.

From an initial population in the cube, the objective is to compute the population after a given number of generations.

3 Implementation Details

3.1 Input Data

The initial population is contained in a file (e.g., `ex1.in`) that starts with one line with a single positive integer (below 10,000) indicating the size of the side of the cube that represents the space where the population will evolve.

This is followed by a random number of lines, each of which will contain three integers, representing the coordinates (x, y, z) of a cell where a live individual should be placed initially.

Your program should allow exactly two command-line parameters, the first used to specify the name of this input file, and the second an integer defining the desired number of generations.

3.2 Output Data

The output of this problem should be the coordinates of the cells with live individuals at the end of the specified number of generations. This output must be **ordered** with respect to the coordinates (with x having the higher weight, then y and then z), so that it can be validated against the correct solution.

Your program should send these output lines (and **nothing else!**) to the standard output.

The project **cannot be graded** unless you follow these input and output rules!

3.3 Implementation Notes

The number of individuals will be in the order of $O(\text{size}^2)$, hence the cube will be very sparse. Your implementation should be designed to allocate an amount of space proportional to this value. If a dense data structure to the matrix is used (in the order of $O(\text{size}^3)$), your program will not be able to handle the largest problem instances.

Note also that the computation of the next generation should be based on a stable representation of the current generation (i.e., you cannot modify the current population as you compute the next generation).

3.4 Sample Problem

If file `simple.in` contains:

```
4
0 0 0
0 2 0
1 1 1
```

it defines a cube with four cells on the side and an initial population with three individuals at coordinates (0,0,0), (0,2,0) and (1,1,1).

To evolve for a single generation, the command and respective output should be:

```
$ life3d simple.in 1
0 1 0
0 3 0
$
```

For two generations:

```
$ life3d simple.in 2
0 0 0
0 2 0
$
```

4 Part 1 - Serial implementation

Write a serial implementation of the algorithm in C (or C++). Name the source file of this implementation `life3d.c`. As stated, your program should expect exactly two input parameters.

This will be your base for comparisons and it is expected that it should be as efficient as possible.

5 Part 2 - OpenMP implementation

Write an OpenMP implementation of the algorithm, with the same rules and input/output descriptions. Name this source code `life3d-omp.c`. You can start by simply adding OpenMP directives, but you are free, and encouraged, to modify the code in order to make the parallelization more effective and more scalable. Be careful about synchronization and load balancing!

6 Part 3 - MPI implementation

Write an MPI implementation of the algorithm as for OpenMP, and address the same issues. Name this source code `life3d-mpi.c`.

For MPI, you will need to modify your code substantially. Besides synchronization and load balancing, you will need to take into account the minimization of the impact of communication costs. You are encouraged to explore different approaches for the problem decomposition.

Extra credits will be given to groups that present a combined MPI+OpenMP implementation.

7 What to Turn in, and When

You must eventually submit the sequential and both parallel versions of your program (**please use the filenames indicated above**) and the times to run the parallel versions on input data that will be made available (for 1, 2, 4 and 8 parallel tasks for both openMP and MPI, and additionally 16, 32 and 64 for MPI). Note that we will **not** be using any level of compiler optimizations to evaluate the performance of your programs, so you shouldn't also.

You must also submit a short report about the results (1-2 pages) that discusses:

- the approach used for parallelization
- what decomposition was used
- what were the synchronization concerns and why
- how was load balancing addressed
- what are the performance results, and are they what you expected

You will turn in the serial version and OpenMP parallel version at the first due date, with the short report, and then the serial version again (hopefully the same) and the MPI parallel version at the second due date, with an updated report. Both the code and the report will be uploaded to the Fenix system in a zip file. **Name these files** as `g<n>omp.zip` and `g<n>mpi.zip`, where `<n>` is your group number.

1st due date (serial + OMP): **April 7th**, until 5pm.
Note: your project will be tested in the practical class just after the due date.

2nd due data (serial + MPI): **May 19th**, until 5pm.
Note: your project will be tested in the practical class just after the due date.