Project 2: Organism Simulator

Due: Friday April 12

In this project, you will be creating a simulation of organisms in a petri dish as they grow and evolve following the simulations done by Avida-ED. The organisms themselves each have a cyclic genome composed of CPU instructions for a hypothetical CPU. During each timestep of the simulation, the organisms execute the next CPU instruction in their genome (if they have sufficient energy). Over time the organisms reproduce, evolve, and die off.

You can play around with the graphical version of Avida-ED at avida-ed.msu.edu/app/AvidaED.html to get an understanding of the mechanics of a single organism and the population of organisms as a whole.

This simulation will be written using MPI and OpenMP. Each MPI process will be responsible for a subset of the organisms and OpenMP is used within a single process for parallelization. The main/root/primary process will be responsible for coordinating the activities of the population as a whole (along with its own subset of the organisms).

An Organism

An individual organism is like a process on a computer: it has a sequence of basic CPU instructions, memory, and registers. The virtual CPU is defined in the AvidaED manual and includes instructions for replication. The copy instruction sometimes mutates the data allowing for evolution. Organisms die if they fail to replicate after a certain amount of time or if they are replaced by another organism. If an organism "learns" how to perform bitwise logic tasks (like NOT, NAND, XOR, …) then it gains extra energy allowing it to reproduce faster and take over.

An individual organism is already defined since it is essentially a serial program on its own (and cannot be parallelized in any significant way). See organism.h and organism.c for the definition of an organism. You do not need to make any changes in organism.c. You only need to make changes in simulation-distributed.c (and possibly a few minor ones in organism.h).

The Environment

The environment is a square grid. Each location on the grid can have no or one organism. In our program, an organism with a size 0 genome is a non-existent organism. When an organism reproduces it takes one of the neighboring cells in the grid, possibly replacing any organism currently there. The size of the grid has a large influence on the time required to run simulations since it controls the number of organisms allowed in the environment.

The environment needs to be divided up amongst the MPI processes. It is up to you to decide how to divide up the environment (could be consecutive rows of the grid, blocks of the grid, or even constantly alternating). You may add additional fields to the `environment` struct if you need to store information there. You cannot change the `organism` or other related structures.

Within a single process, there will likely be multiple organisms since the grids will be large. You will use OpenMP to make good use of parallelization within a single process. Between processes, you may have slightly incomplete environments, but the organisms require accurate values for `world_size` and `mutation_prob`. Importantly, the `grid` does not need to be filled out at all. However, the `env` field within an individual organism needs to be correct along with its `location_i` and `location_j` fields. The function `environment_alloc_grid()` ensures that all organisms within the grid have these fields set properly.

The Simulation Programs

The simulation programs takes 3 or 4 optional arguments:

- `-n` the number of iterations to simulate for; default is 4,194,304 (2^{22})
- `-r` the random seed to use, default it the current time, set this to get consistent results
- `-m` probability of mutation for each byte of memory copied, default is 0.02 (2%)
- `-t` number of threads to use (in distributed it is the number of threads *per process*)

After the optional arguments, there are 2 required arguments for the input file and output file.

The program prints out the time it took to run the entire simulation (not including argument parsing, loading, saving, printing, or cleanup) along with the number of each task completed.

Input and Output Files

The input and output files are plain text files. The first line has one number for the size of the world. All lines after that start with the position of an organism followed by the genome of the organism. No other information is saved about an organism (no registers, heads, stacks, …). Like a shutdown computer, the programs are saved but that is it. Nothing in active memory is saved. These files are read and written to with `environment_load_from_path()` and `environment_save_to_path()`.

Versions

There are 3 versions of the simulation program:

- Serial - done for you
- Shared - only OpenMP, done for you
- Distributed - both MPI and OpenMP

Changes

You need to make several changes to simulation-distributed.c. Currently, it is a copy of simulation-shared.c with several TODO statements where additions need to be made. Several of them are complicated enough to warrant separate functions.

Benchmarks

Coming soon. In the meantime make sure the results you are producing are correct (the number of completed tasks are the same and the output file matches when providing the same random seed along with a matching number of iterations and mutation probabilities). Expected outputs can be generated with the provided programs.

Analysis

Perform a detailed analysis of both of the parallelized programs. Make sure to discover how well the programs work as the number of threads and/or processes changes and the size of the world changes. Report the speedup and efficiency. Find the "crossover" points (when programs switch ranks). There are a total of at least 3 crossover points.

When doing your analysis, you should use large, pre-populated "worlds" (instead of just starting with just one of two in the middle). They should express a variety of functions (i.e. not be genetically identical). It is much better to start with a populated world as the programs all go at very odd speeds when there are numerous empty cells.

Additional pre-populated worlds can be created by starting from basic.txt (or similar) and running it for numerous iterations and then using the output file as the input file for other runs.

You will include your written-up, nice-to-read, fully explained, analysis with plots in the provided markdown file.

Rubric

35 pts for simulation-distributed
25 pts for sufficient speed in all cases (limits will be published)
25 pts analysis
15 pts code quality