Parallel Programming

C Lab - Practice assignment

Analyzing a C program code

Introduction

We will make use of an initiative born in the EduPar workshop of the IPDPS conference [1] and the EduHPC workshop of the SuperComputing conference [2] (probably two of the most prestigious conferences in the world in High Performance Computing - HPC). This initiative provides a set of problems that can be used in labs of subjects of High Performance Computing or Distributed Systems and that have been included in a more ambitious project of the NSF/IEEE_TCPP for the curriculum design and the elaboration of course contents in the field [3].

We have chosen, from all the existing possibilities in this project, a simulation system based on fire extinguishing agents [4]. We will provide you with a sequential C code that simulates the extinguishing of a fire, with different sources of ignition, by one or more firefighting teams.

This code will be used during the course for two purposes: 1.) to learn C programming and 2.) to generate a parallel version using different paradigms we will study. Thus, for this first lab, the idea is to study the source code of the sequential version of this program to identify the different phases of this program, how it depends on the inputs of the simulation, what is its computational cost and what is its potential for parallelism.

Below we describe the problem and propose a set of tests and questions to guide the analysis of the code at the functional level and its parallelization potential.

Program description

The code provided for this lab makes a simplified simulation of a fire and its extinguishing considering one or more focal points (fire points). We will have a matrix that will represent the rectangular surface where the simulation will be carried out and a set of structures to store the parameters of the firefighting teams and the focal points.

Figure 1 shows a possible state of the simulation for a surface of 30x30 points after a certain number of iterations. The following symbols are used to represent the state of each point:

0 : The point is at ambient temperature.

. : The point temperature is between 25 and 50 degrees.

1-9 : The point temperature is between the number multiplied by 100 and (number + 1) multiplied by 100 degrees.

+ : The temperature of the point is greater than 1000 degrees.

() : The point is an active focal point.

[] : One or more teams are visiting the point or extinguishing the fire (if it is a focal point).

The example shown on Figure 1 represents two firefighting teams that have just extinguished a fire in the lower right part of the simulated surface (residual heat is dissipating) and are now moving towards the active fire in the upper left part.

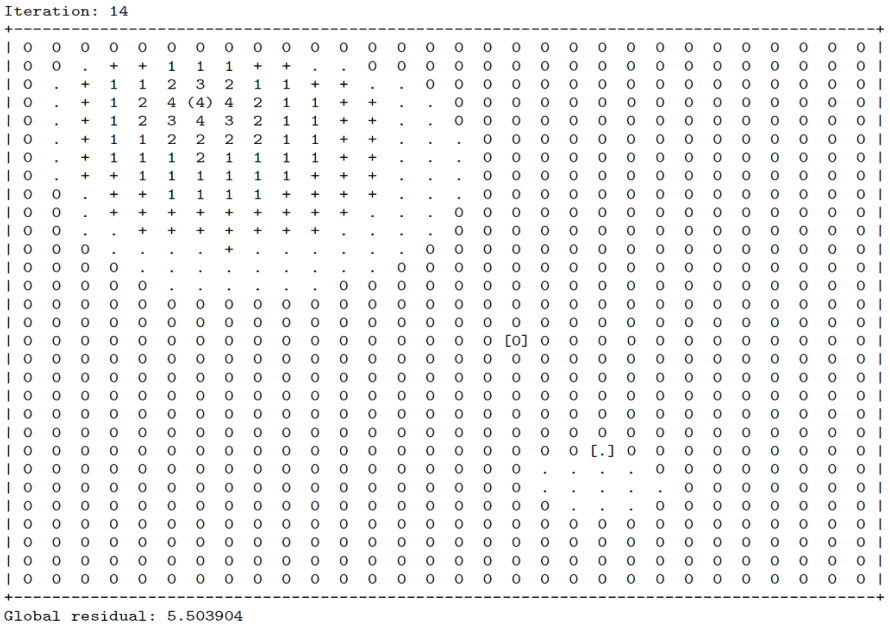

Figura 1. Example of simulation on a 30x30 surface with 2 focal points and 2 firefighting teams.

The code provided includes, apart from the simulation, reading the input and checking for errors, defining structures, creating dynamic structures, etc. You don't need to modify this accessory code, so keep in mind that when you develop a parallel version, you will only need to work on the section of the code marked by the comments "START HERE: DO NOT CHANGE THE CODE ABOVE THIS POINT" and "STOP HERE: DO NOT CHANGE THE CODE BELOW THIS POINT.

Code description

Program arguments: The program receives a description of the simulation scenario as a text file or via the following arguments:

size1, size2: Size of the two-dimensional array that stores the temperature information of the surface.

maxIter: Maximum number of iterations the simulation will perform.

numTeams: Number of firefighting teams present in the simulation.

teamDataList: Tuple of three numbers separated by spaces indicating the initial position of each team (x, y) and their firefighting capacity (1, 2 i 3).

numFocalPoints: Number of focal points (fire points).

FocalPointsDataList: Tuple of 4 numbers separated by spaces indicating for each focal point: position (x, y), step in which it will be activated and amount of constant heat that the fire will emit once active.

Format of surface files. Program arguments (introduced in the previous section) can be passed via a text file. The first line of the file tells us the values of size1, size2 and maxIter. The second contains the number of firefighting teams present in the simulation. Then we will have a line per team where its initial position (x, y) and its type - capacity (1,2,3) will be indicated. Once we have finished describing the firefighting teams, we will have a line that will indicate the number of focal pointsin the fire. Then, we will have one line per focal point indicating its position (x, y), the step of the fire where it will be activated and the amount of heat it will emit once active. You can see examples of input files for the simulation in the ./test_files directory inside the practice folder (example command: *./exec_name -f test_files/test1*). The cases included in this folder are the ones we will use to analyze the performance for both versions developed during the semester: a serial and a parallel one. Read the README file included in this folder to understand the different cases provided.

Functional description. At the beginning the program initializes all points of the surface at ambient temperature (0). Once initialized, the fire simulation will begin, and different phases will be processed (numbered in the same way as in the source code comments):

4.1. **Activate focal points.** All the focal points that turn on for this step of the simulation are activated. The number of focal points that have been disabled during the simulation are counted.

4.2. **Propagate heat.** This phase takes 10 steps of spreading the heat over the surface before the firefighters start to act.

4.3. **Move teams.** Once the heat has spread, the firefighting teams move and go into action.

4.4. **Team actions.** Once the firefighters have moved, they will carry out the relevant actions.

The simulation will end once all the focal points are turned off or when the residual heat is below the stability threshold or when the maximum set of iterations is reached. At the end of the program, it shows the time that has passed during the simulation and some values to verify that it has been executed correctly (final position of the firefighting teams and residual heat of the focal points).

Debug mode. If we compile the code with the –DDEBUG flag, a graphical representation of the simulation will be displayed (as the one we can see on Figure 1). This can be useful for detecting bugs or seeing how the simulation evolves.

Material

With this assignment we provide you with the sequential code of the program extinguishing.c. You will find these files in Campus Virtual.

The recommendation is that you create a copy of the provided directory into your account and make the modifications you deem appropriate.

To compile the sequential version of the program you have to put the following command:

gcc -Ofast -o **exec_name** extinguishing.c -lm

where **exec_name** is the name of the executable you want to have.

If you want to see the output of each simulation step, you must add the option -DDEBUG to the previous command line.

Deliverable

This is the second deliverable for the C programming part. It allows you to obtain the rest of the grade (4 points) and hence, reach the maximum grade (10). You will have to deliver your answers to the questionnaire indicated below. The answers must be included in a PDF file and delivered via Virtual Campus before the indicated deadline.

You will work with this program for different parts of the course (OpenMP, MPI and GPU programming), and hence (for the next deliverables) you will have to parallelize it using different programming paradigms: shared or distributed memory.

Bibliography

[1] https://www.ipdps.org/ipdps2022/

[2] https://sc21.supercomputing.org/

[3] https://tcpp.cs.gsu.edu/curriculum/?q=peachy

[4] https://tcpp.cs.gsu.edu/curriculum/?q=system/files/Peachy_Eduhpc_19.pdf

Questionnaire

1. Describe the following data structures, indicating the meaning, use, and value range of each of its fields:
   a. Team
   b. FocalPoint

2. The simulated surface is two-dimensional, but the data structure representing it is a dynamically created one-dimensional array (a macro has been included that makes it easy to convert two-dimensional coordinates to a position in the array). What reasons do you think the programmers might have had for making this decision, instead of dynamically creating a two-dimensional structure?

3. Using the cp_Wtime function, measure the time of the following iterative structures (*for* loop) for Tests 2, 3, and 4. In each case, calculate the percentage of it regarding the total time for each simulation iteration. In which phase of the program should you focus your optimization efforts?
   a. Line 308:
   ```
   for( i=0; i<num_focal; i++ )
   ```
   b. Line 320:
   ```
   for( step=0; step<10; step++ )
   ```
   c. Line 356:
   ```
   for( t=0; t<num_teams; t++ )
   ```
   d. Line 401:
   ```
   for( t=0; t<num_teams; t++ )
   ```

4. The iterations of the simulation are controlled by the *for* loop of the line 304:
   ```
   for( iter=0; iter<max_iter && ! flag_stability; iter++ )
   ```
   Use the cp_Wtime function included in the code to measure the simulation time. Do you think it's valuable to measure the time for each iteration? Justify your answer. Would it be beneficial to use a different metric other than the total execution time?