Agent-based Simulation of Fire Extinguishing

EduHPC'19 Peachy Assignment

1 Introduction

You are provided with a sequential code that simulates the evolution of a fire scenario with several focal points that is extinguished by multiple teams of firefighters.

The code provided computes a simplified simulation of fire emergencies, like for example in industrial facilities like oil refineries or chemical plants. Only a rectangular surface is considered in the simulation. This surface is modelled as a set of discrete and evenly spaced control points. A bidimensional array is used to represent the heat of each point in a given time step. Another one-dimensional array is used to store the information of a set of fire focal points. These points will be activated in fixed time instants during the simulation. Each focal point has a constant amount of heat until a team arrives to suffocate it. The program computes the evolution of the whole fire, taking into account the focal points and the behaviour of the teams of firefighters. This behaviour is simulated in a very simplified way, always commanding each team to move in the direction of its nearest focal point to suffocate it.

Figure 1 shows the output of the program for a surface of 30x30 points, after 14 iterations simulating the operations of two extinguishing teams. The following symbols are used to represent the state of each control point:

`0`: The point is at ambient temperature.

`-`: The heat in the point is between 25 and 50 degrees.

`+`: The heat in the point is between 50 and 100 degrees.

`1-9`: The heat in the point is between the number multiplied by 100, and that number plus 100 degrees.

`*`: The heat in the point is greater or equal to 1000 degrees.

`( )`: The point is an active focal point.

`[ ]`: One or more teams are visiting the point, or extinguishing the fire, if it is an active focal point.

The example represents two teams that have just extinguished a focal point at the bottom right of the surface (the residual heat is still dissipating) and now they are moving to the active focal point at the top left.

2 Sequential code description

Program arguments The program receives a scenario description, in a file or by the following arguments:

1. size1, size2: Sizes of the bidimensional arrays of control points that stores the heat information of the surface.

2. maxIter: Maximum number of simulation iterations allowed.

```
Iteration: 14
+------------------------------------------------------------------------------------------+
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  1  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  2  3  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  2  4 (4) 4  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  2  3  4  3  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  2  2  2  2  1  1  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  1  2  1  1  1  1  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  +  1  1  1  1  1  1  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  1  1  1  1  +  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  +  +  +  +  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  .  +  +  +  +  +  +  +  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  .  .  .  .  +  .  .  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  .  .  .  .  .  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  .  .  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 [0] 0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 [.] 0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  .  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  .  .  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
+------------------------------------------------------------------------------------------+
Global residual: 5.503904
```

Figure 1: Example of output of the program in debug mode for an array of 30x30 positions and two extinguishing teams

3. numTeams: Number of extinguishing teams.
4. teamsDataList: Tuples of three numbers separated by spaces that indicate the initial position of each team and its extinguishing capacity (valid types are 1, 2, or 3).
5. numFocalPoints: Number of fire focal points.
6. focalPointsDataList: Tuples of four numbers separated by spaces that indicate for each focal point: position (x,y), activation time step, and constant amount of heat when it is active, before it is extinguished.

Surface file format The program arguments introduced in the previous section can be also passed to the program by means of a text file. The first line of the file must contain the grid sizes *size1* and *size2*, and the maximum number of iterations *maxIter*. The second line contains the number of extinguishing teams. The following lines contain the information of one team per line, its initial position (x,y), and its type (1, 2, or 3). Then, it appears a line with the number of focal points. The following lines contain the information of one focal point per line, the position (x,y), time step of activation, and the amount of heat. Students are encouraged to manually create or automatically generate their own test files. Several examples are provided along with the code.

Functional description The program initializes all the control points of the surface with temperature `0` (ambient temperature), and then it starts the simulation loop. On each iteration there are several stages. First, the focal points array is examined in order to check if any of

them must be activated at that time step. Then, 10 steps of heat propagation are computed. The surface data is copied into an ancillary array, and the heat value of each position is updated using the copied values of its neighbours. The maximum difference between old and new heat values on any position of the surface is found.

Each team calculates its distance to all the active focal points in order to decide in which direction should it move. Movement rules are slightly different depending on the type of the team. After that, the result of the actions performed by each time is computed. If a team reaches an active focal point, it extinguishes it. It also reduces the amount of heat in the surroundings of its location according to an *action radius* that depends on the type of the team. Type-1 teams move fast (diagonally) but they have a short action radius. Type-2 and -3 teams move slower (steps can be just horizontal or vertical), but they carry extinguishing tools that allow them the reduction of heat in a longer range.

The simulation ends whether a fixed number of iterations is reached or the (residual) heat variation is under a given stability threshold.

The program shows the elapsed time of the simulation and some results that are useful to check its correctness (final positions of the teams, and residual heat in the focal points).

Debug mode If the source code is compiled with the -DDEBUG flag, a graphical representation of the results is presented, as discussed before. This representation can be useful to detect bugs, or just to know how the simulation evolves.

3 Project goal

Use the proposed parallel programming model to parallelize the program, optimize the code, and obtain the best possible performance.

Code modifications allowed Students can modify the sequential code provided as long as they observe the following restrictions:

- Exploit parallelism using only the parallel programming model proposed.
- The argument processing section, array memory allocations, time measurement points, and output of results, should not be changed. The section of code that the students should target and modify is clearly identified in the main function. This section is found between the points where the time measurement is started and ended. Functions defined in the program that are called from the target section of the code can also be modified, substituted, or eliminated.
- Any change in the algorithm or data structures must be discussed with the teachers in advance, in order to avoid modifications that significantly alter the parallelism exploitation with the proposed parallel programming model, which is the purpose of the assignment.

To measure execution times, compile the program with the maximum optimization level of the compiler (for example gcc -O3). We want to focus on program changes that do not interfere, and even facilitate, compiler optimizations.