

Hill Climbing with Montecarlo

EduHPC'22 Peachy Assignment

1 Introduction

You are provided with a sequential code that uses a probabilistic approach to find the highest point of a two dimensional function (representing for example the height of terrain) in a given rectangular area. The selected area is modelled as a set of discrete and evenly spaced points or positions. A bidimensional array is used to store the value of height provided by the function for the points. As the bidimensional array can be huge and the cost of computing the function for all the points is high, we use a probabilistic method to obtain a solution without computing all the values. The hill-climbing approach is based on selecting a starting random point, and iteratively move the selected point to one of the four neighbors: The one with the highest value of the function. Thus, the search is climbing in the direction that looks more promising to find a higher value. When the selected-point value is higher or equal than the neighbor values, a local maximum has been reached and the search stops. The program uses a probabilistic approach to find the global maximum by starting a given number of hill-climbing searches, with the hope that the highest local maximum found will be the global maximum in the selected area. Launching more searchers implies more computation but probabilities to find the correct global maximum.

In order to avoid computing the costly function for all the control points in advance, a matrix is initialized to non-valid values, and a cell is updated with the corresponding function value only when a searcher checks that point. Also, when a searcher moves to a cell already travelled by a previous searcher, it would follow exactly the same path to the same local maximum, and the searcher can stop. Ancillary structures are used to store which searcher visits each cell for the first time, and which searcher has stopped because it is following the trail of another searcher. After all searches have finished, apart from the global maximum found, the program computes the number of trail steps of all the searchers leading to the same local maximum, the total number of travelled points, and the accumulated values of height in local maximums.

Example Figure 1 shows the output of the program in debug mode for an example with an array of 40x30 cells. The first plot represents the identifiers of the first searchers that travel a cell. The second plot represents the heights computed.

2 Sequential code description

Program arguments The program arguments describe the simulation scenario:

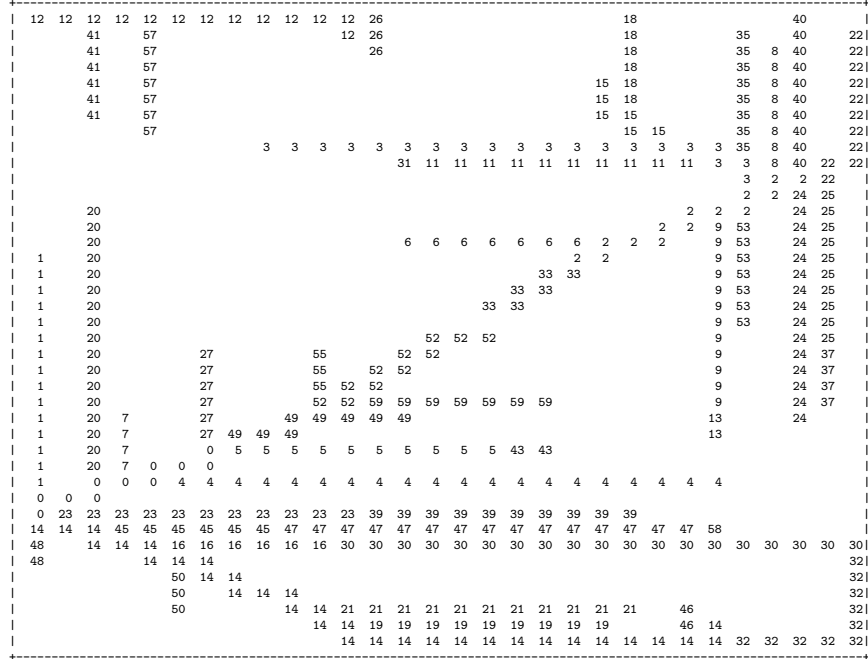
1. `rows, columns`: Number of array positions
2. `x_min, x_max, y_min, y_max`: Limits of the rectangular area explored.
3. `searchers_density`: Ratio of searchers per lattice positions.
4. `short_rnd1, 2 and 3`: Three unsigned short numbers that are used as seeds to initialize random generators to initialize searchers starting positions.

Students are encouraged to create their own scenarios, to test different situations and problems. Several examples are provided along with the code.

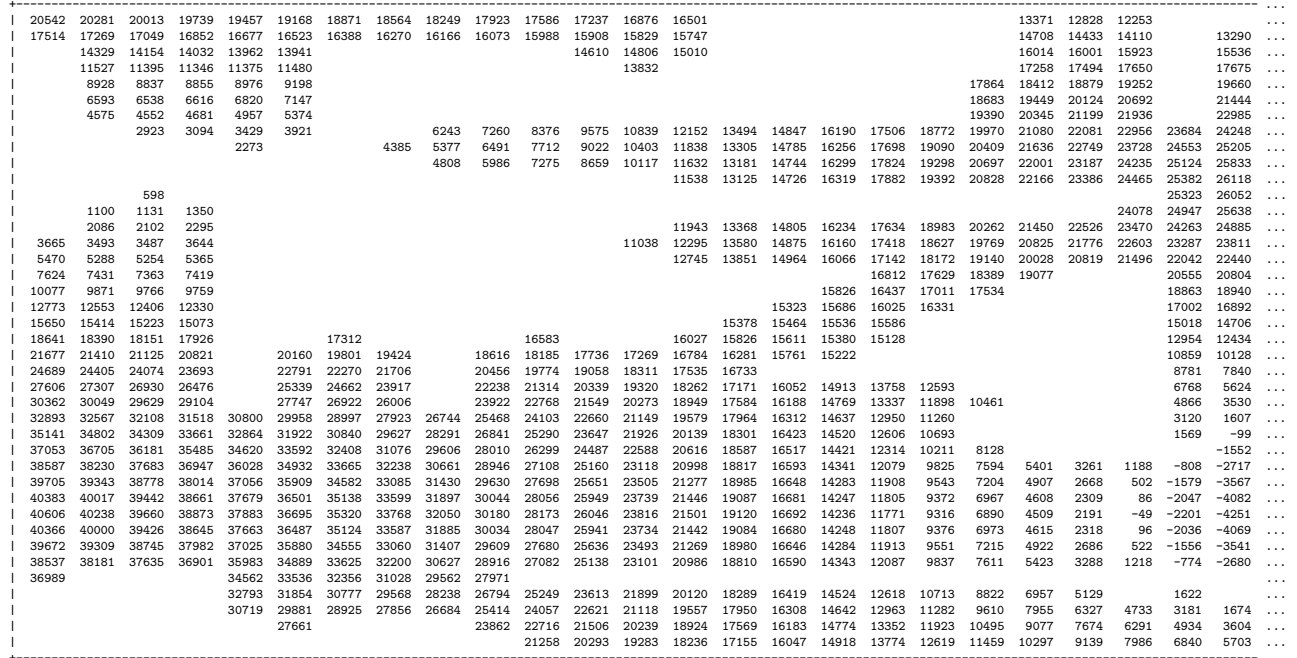
```
./climb_seq 40 30 -6.28 -0.2 -6.28 -0.2 0.05 9342 234 73323
```

```
Arguments, Rows: 40, Columns: 30
Arguments, x_range: ( 0, 0 ), y_range( 32, 0 )
Arguments, searchers_density: 0.050000
Arguments, Init Random Sequence: 9342,234,7787
```

Trails:



Heights:



Time: 0.109113
Result: 3, 40606, 285, 438, 14012463

Figure 1: Example of the program output in debug mode for the indicated argument values. Only the left part of the second matrix is shown in the figure due to limited space.

Results The program shows at the end of the simulation the following results. The execution time of the simulation (without initialization times) and a list of values that is used to verify the correctness of the simulation.

Debug mode If the program is compiled with the flag `-DDEBUG` (option included in the provided makefile) a piece of code is activated to print with ASCII art a representation of the results at the end of the program (as in the previous figure). In the sequential program it prints the results after each step of a searcher, to better understand the sequential program behaviour. The arguments and starting data are also shown to have a record of the arguments used to create the scenario.

3 Project goal

Use the parallel programming model proposed by the teacher to parallelize this program without changing the algorithm. Optimize the code and obtain the best possible performance. Always check that the results are correct and equivalent to the sequential execution with the same input arguments.

A note about random numbers and parallelism The generation of a pseudo-random sequence of numbers with the classical C library functions is inherently sequential. Our program uses library functions that compute the next random value using a small input array that stores the state of a random sequence after each call. Take care to avoid changing the order in which random numbers from a given sequence are obtained, or the program results will change and will be non correct.

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.