Solving a maze using OpenMP

Francesca Lanzillotta
E-mail address
francesca.lanzillotta@edu.unifi.it

Abstract

The ABSTRACT is to be in fully-justified italicized text, at the top of the left-hand column, below the author and affiliation information. Use the word "Abstract" as the title, in 12-point Times, boldface type, centered relative to the column, initially capitalized. The abstract is to be in 10-point, single-spaced type. Leave two blank lines after the Abstract, then begin the main text.

1. Introduction

The goal of this project is to compare the performances of a maze solver using both a sequential and a parallel approach using the OpenMP library. The maze, generated using a recursive algorithm, is solved by randomly moving a certain amount of particles until one reaches the exit.

1.1. Maze

The maze is generated using the recursive backtracker algorithm, a randomized version of the depth-first search algorithm. This method is one of the simplest way to generate a maze and it's easily implemented with a recursive approach. The maze is firstly initialized as a grid resembling a chess board, alternating walls and free space cells. On this object we manually set a starting point, roughly in the middle of the maze, where all particles will be initially spawned. We also randomly set the exit on one of the sides of the maze. The following recursive algorithm is invoked to create the pathways:

1. Pick a starting cell.
2. Mark the cell as visited.
3. While the cell has unvisited neighbours.
   (a) Randomly pick an unvisited neighbouring cell.
   (b) Remove the wall between the current cell and the chosen cell.
   (c) Repeat recursively steps 2-3 on the chosen cell.

The algorithm stops once it successfully backtracks all the way to the starting cell, ensuring all cells have been visited. Figure **??** depicts the maze before and after the pathways have been generated.
The mazes created with this approach have a single solution i.e. there exist a single path connecting the starting point and the exit. Moreover, the mazes tend to have many long corridors, since the algorithm fully explores every branch before backtracking. All these functionalities are implemented in the `Maze` class.

1.2. Particle

Particles are objects that move through the maze at random, until some particle reaches the exit: once the solution is found, all particles follow its path to the exit.
Each particle is spawned from the same starting point. At each step, the particles randomly pick a free neighbouring cell. In this context, a cell is free i.e. a particle can move to it, if it's not a wall. This means that more than one particle can reside on a single cell: this ensures the particles don't clog the pathways and can move freely within the maze.
As already mentioned, the mazes used in this project tend to have many long corridors: to make the particles move less erratically along the pathways, we implemented the function picking the random moves so that the option of going backward is inhibited with a certain probability.

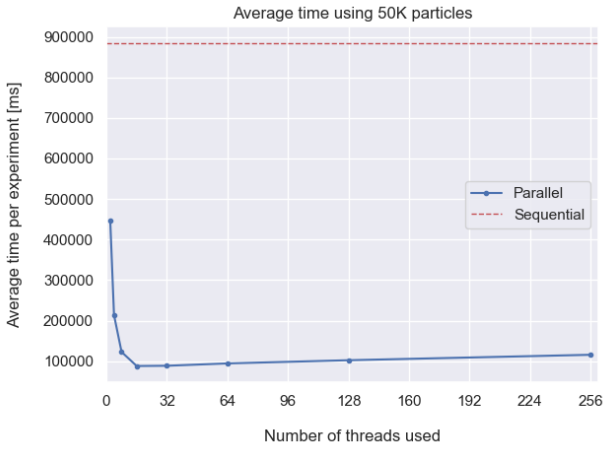
2. Solving the maze

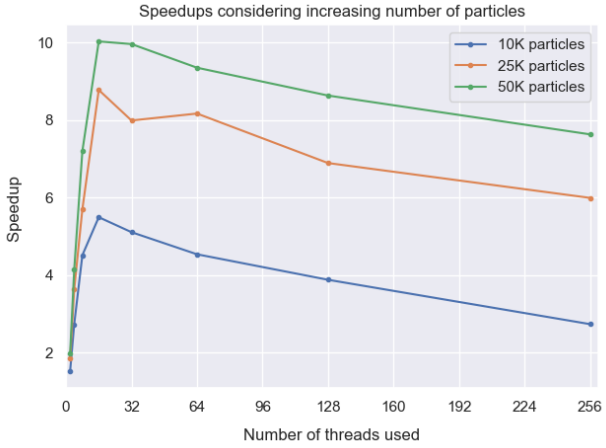
The strategy to solve the maze is divided in two main steps:

1. Move randomly all particles until one finds the exit
2. Have all the particles follow the solution's path until they exit the maze

The goal of this project is to compare

Threads	Number of particles		
	10K	25K	50K
Sequential	6.1 s	213.698 s	884.825 s
2	4 s	115.9 s	447.5 s
4	2.2 s	58.8 s	213.8 s
8	1.3 s	37.5 s	122.8 s
16	1.1 s	24.3 s	88.2 s
32	1.2 s	26.7 s	88.8 s
64	1.3 s	26.2 s	94.6 s
128	1.6 s	31 s	102.4 s
256	2.3 s	35.6 s	115.9 s





2.1. Sequential approach

2.2. Parallel approach

3. Experimental Results

References