RANDOM MAZE SOLVER – K-MEANS

Parallel Programming - Final Presentation

Plator Rama

Florence 21/10/2024

- **Random Maze Solver**
  - Introduction
  - The algorithm(with OpenMP)
  - Results
- **K – Means**
  - Introduction
  - Data structure
  - The algorithm(with CUDA)
  - Results

Introduction

1. Random Maze Solver

The program generates maze in the form:

```
# S # # # # # # # # # # # # # # # # # # #
# * * * * * # . . . . . . . . . # . . . #
# * # # # . # . # # # # # # # . # . # . #
# * # . # . # . . . . . . . # . # . # . #
# * # . # . # # # # # # # . # . # # # . #
# * # . . . # . . . . . . . # . . . . . #
# * # . # # # . # # # # # # # # # # # . #
# * # . . . # . # * * * . . . . . # . #
# * # # # # # # # * # # # # # # # . # . #
# * * * * * * * * * # * * * * * # . # . #
# * # # # * # # # # # * # # # * # . # . #
# * # * * * # * * * * * . . # * # . # . #
# * # * # * # * # # # # # # # * # . # * #
# * # * # * # * . . # * * * * * # . # * #
# # # * # * # * # # # * # # # # # . # * #
# * * * # * # * # * * * # . . . . . # * #
# * # # # * # * # * # # # . # # # # # * #
# * # * * * # * # * * * # . . . # * * * #
# * # # # # # * # # # * # # # # # * # # #
# * * * * * * * . . # * * * * * * * * * #
# # # # # # # # # # # # # # # # # # # E #
```

- S symbol represent starting position
- E symbol represent ending position
- # represents wall
- . (dots) represents corridor
- * represents solution

In order to find the exit, the algorithm generates N particles (initialized on the starting position) and for each of them, at each step, it randomly chooses an adjacent tile among the allowed ones (the ones with dot inside).

The algorithm

1. Random Maze Solver

1. Initialize partition size to divide particles among threads.
2. Initialize a pointer to store the '*winning particle*' and a flag '*exit_found*' to signal if the exit is found.

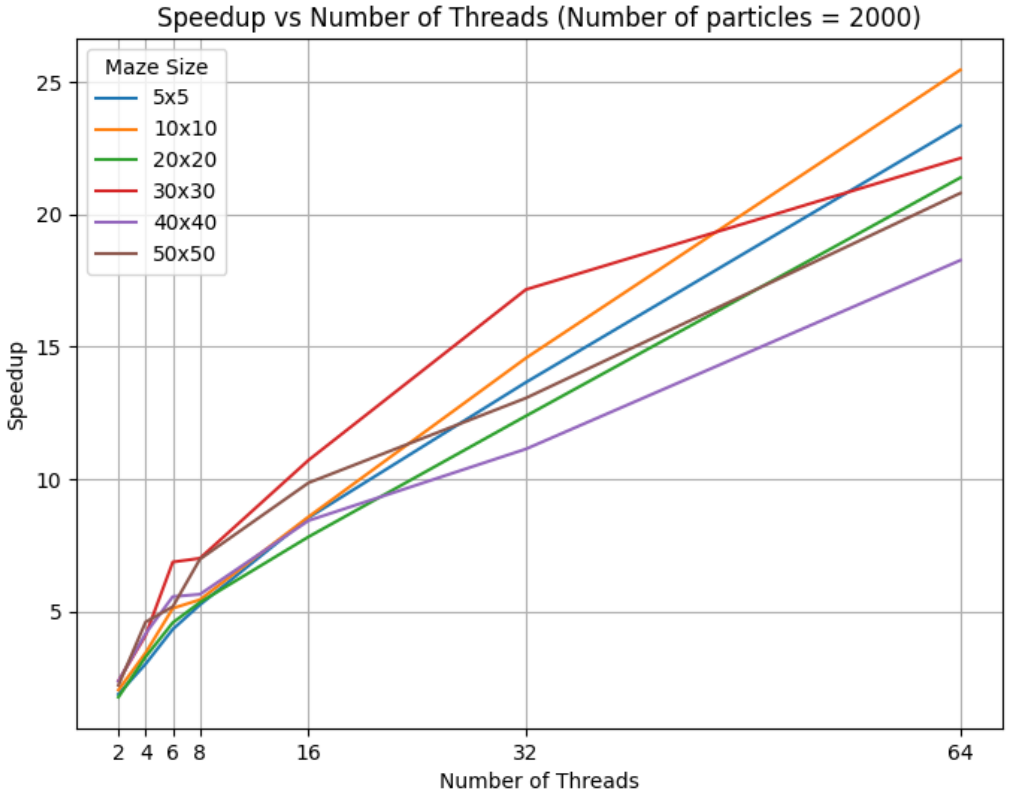
#pragma omp parallel num_threads(threads_number)

3. For each thread:
   a) Calculate the start and end indices for the partition of particles assigned to the thread.
   b) While the exit is not found:
      a) For each particle in the partition:
         - Move the particle in a random direction.
         - Check if it has reached the exit.
      b) If a particle finds the exit:
         - Enter **critical section** to avoid conflicts between threads.
         - If the exit has not been flagged, update `*winning_particle*` and `*exit_found*`.

4. If a winning particle is found, trace the path in the maze.

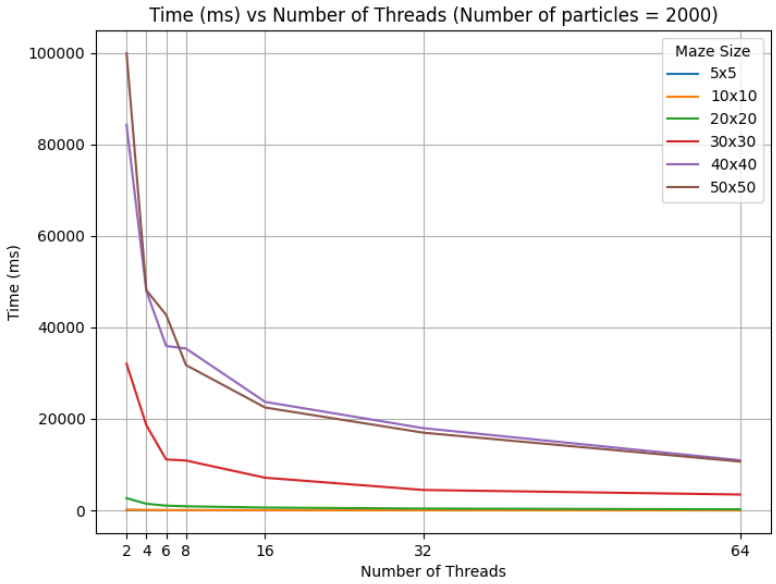
Results

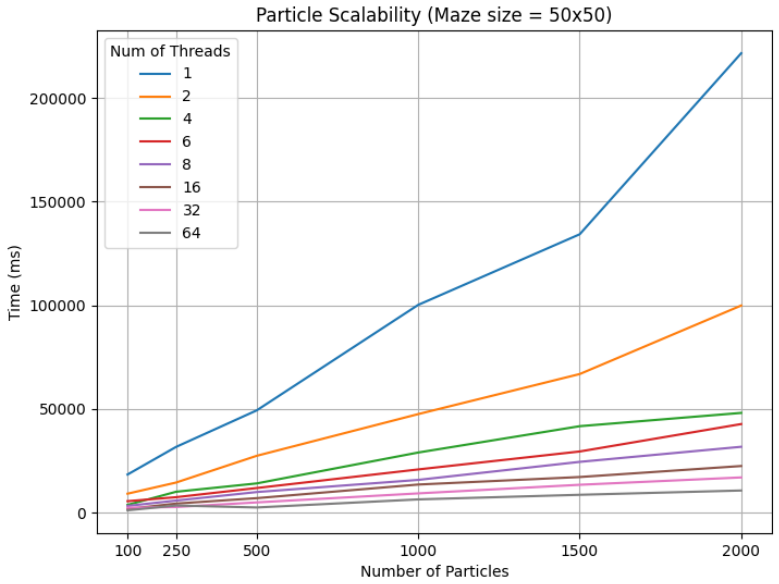
1. Random Maze Solver



Results

1. Random Maze Solver





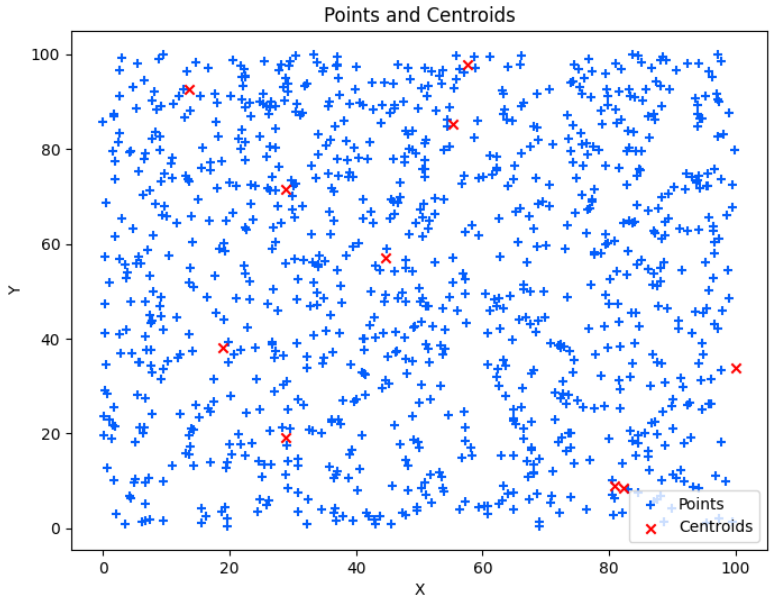
Introduction

2. K – Means 2D

K-Means is an unsupervised learning method that allows you to find subgroups in a dataset. In this work K-Means 2D is studied.

We start by generating n random points after which we generate k random centroids.

The goal is running k-means algorithm for i iteration.



Data structure

2. K – Means 2D

Generic points and centroids follow the ”Array of Structures” (AoS) approach

```
struct Points {
    float *x;
    float *y;
    int *closestCentroid;     // Array storing the index of the closest centroid for each point
};

struct Centroids {
    float *x;
    float *y;
    int *pointCount;         // Array storing the number of points assigned to each centroid
    float *sumX;             // Sum of x coordinates of points assigned to each centroid (for recalculating positions)
    float *sumY;             // Sum of y coordinates of points assigned to each centroid (for recalculating positions)
};
```

Algorithm

2. K – Means 2D

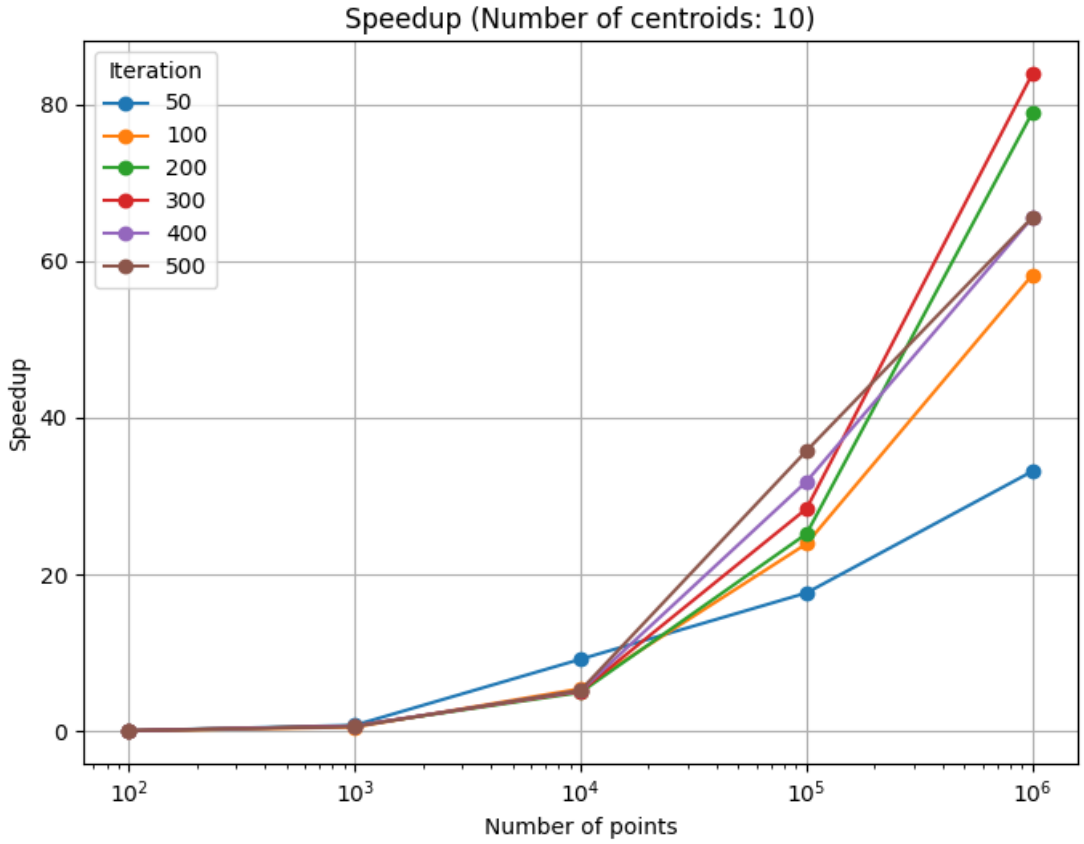
The parallel implementation of k-means on the GPU utilizes two main kernels to optimize performance:

1. **Kernel 1: Assigning Points to Centroids**
   - Computes the Euclidean distance between each point and all centroids.
   - Determines the nearest centroid for each point.
   - Uses atomic operations to update accumulators storing the sum of point coordinates and the count of points assigned, avoiding race conditions.

2. **Kernel 2: Updating Centroids**
   - Calculates the new positions of the centroids by averaging the coordinates of the assigned points.
   - Resets the accumulators in preparation for the next iteration.

Results

2. K – Means 2D



Results

2. K – Means 2D