Parallel Programming - Final term assignment
Random Maze Solver (GPU Version)

Angelo Caponnetto

June 18, 2024

Abstract

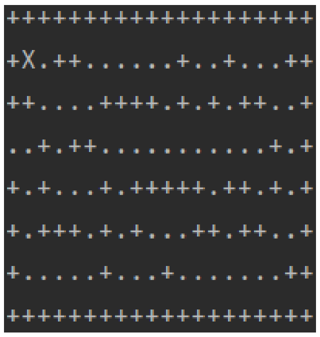
In this work, I will study the CUDA Random Maze Solver. I will compare the sequential version (running on CPU) of the program with the parallel version (running on GPU). I will describe the code I've used and the obtained speed up.

1 Introduction

The assignment consists in the study of the parallel version of a Random Maze Solver. The program generates a maze in the form:

$$maze[y_{MAX}][x_{MAX}]$$

where y_{MAX} and x_{MAX} are the dimensions of the maze.



```
++++++++++++++++++++
+X.++......+..+...++
++....++++.+.+.++..+
..+.++...........+.+
+.+...+.+++++.++.+.+
+.+++.+.+...++.++..+
+.....+...+.......++
++++++++++++++++++++
```

Figure 1: Maze visualization: exit in $(0, 3)$

Dot represents corridor, plus symbol represents wall and X is the starting position.

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 program finishes when a particle finds the exit.

2 Data structure

If the algorithm used the maze structure described in previous section, in order to find all the adjacent corridors, at each iteration, it should use if conditions.

Algorithm 1 FindCorridors(maze, (x_s, y_s))

```
1:  $(x_s, y_s)$ // particle position
2:  $corArr$ // empty array of corridor tiles
3:
4:  //Right tile
5:  if $x_s + 1 < x_{MAX}$ then
6:      if $maze[y_s][x_s + 1] = CorrSymb$ then
7:          Add $(x_s + 1, y_s)$ to corArr
8:      end if
9:  end if
10: //Same for Left, Up and Down tile
```

These conditionals may generate divergence when they are used in an algorithm running on GPU, and because of that we may have a loss in performance. Due to this consideration, the data structure the algorithm will use will be an array organized in this way:

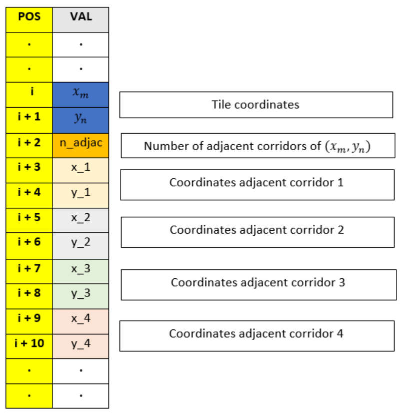

POS
VAL
i
x_m
i + 1
y_n
i + 2
n_adjac
i + 3
x_1
i + 4
y_1
i + 5
x_2
i + 6
y_2
i + 7
x_3
i + 8
y_3
i + 9
x_4
i + 10
y_4
Tile coordinates
Number of adjacent corridors of (x_m, y_n)
Coordinates adjacent corridor 1
Coordinates adjacent corridor 2
Coordinates adjacent corridor 3
Coordinates adjacent corridor 4

Figure 2: Data structure

If a tile doesn't have four adjacent corridors, the excess positions in the array will be filled with zeroes. Tiles in data structure are sorted by rows ((0,0), (1,0), (2,0)...).

3 The Algorithm

Each thread of the kernel follows the evolution of one particle. The pseudocode is reported in Algorithm 2. Note that RandSolver is a __global__function while *d_lin_maze* is the data structure loaded on the device. Starting from $(x, y) = (x_{start}, y_{start})$ the algorithm generates a random number $n \in [0, n_{adjac} - 1]$ and updates (x, y) with the selected neighbour. It checks if the exit is reached and if so it changes the value of the flag (it is a device variable that each thread of the grid can see). Furthermore, x_{array} and y_{arrey} are the arrays in witch the algorithm saves the positions of the particles. The algorithm doesn't save the path of the particles (number of particles $\approx 10^6$); it focuses on carrying out as many calculations in the shortest time.

4 Utilized GPU

Following results were collected using the GPU NVIDIA GTX 1660 Ti with max-Q. Here we report same characteristics:

Algorithm 2 __global__RandSolver(x_{array}, y_{array}, (x_{ext}, y_{ext}), flag, N, d_lin _maze)

```
int  $idx \leftarrow$ thread id
if $idx < N$ then
    short  $x \leftarrow x_{array}[idx]$,   $y \leftarrow y_{array}[idx]$
    short  $n\_rand$
    short  $firstNeigPos \leftarrow$ first initialization
    int $n\_steps \leftarrow 0$

    while $flag \neq 1$ and $n\_steps < max\_steps$ do
        $n\_rand \leftarrow n \in [0, n_{adjac} - 1]$
        $x \leftarrow d\_lin\_maze[firstNeigPos + 2n\_rand]$
        $y \leftarrow d\_lin\_maze[firstNeigPos +$
                                      $2n\_rand + 1]$
        Update $firstNeigPos$
        $n\_steps \ += 1$

        if $(x, y) = (x_{exit}, y_{exit})$ then $flag = 1$
        end if
    end while
    $x_{array}[idx] \leftarrow x$,   $y_{array}[idx] \leftarrow y$
end if
```

- Architecture: Turing
- Memory Size: 6 GB
- Bandwidth: 288.0 GB/s
- CUDA: 7.5
- Shading Units: 1536
- SM Count: 24
- L1 Cache: 64 KB (per SM)
- L2 Cache: 1536 KB

5 Results

In this section time results will be reported. They were taken by changing the number of particles N and the number of active thread N_{THR} for each block. The dimension of the grid is always evaluated as $(N+N_{THR}-1)/N_{THR}$. Speed up curves are evaluated using times of the sequential version (it uses same logic and same data structure) as reference.

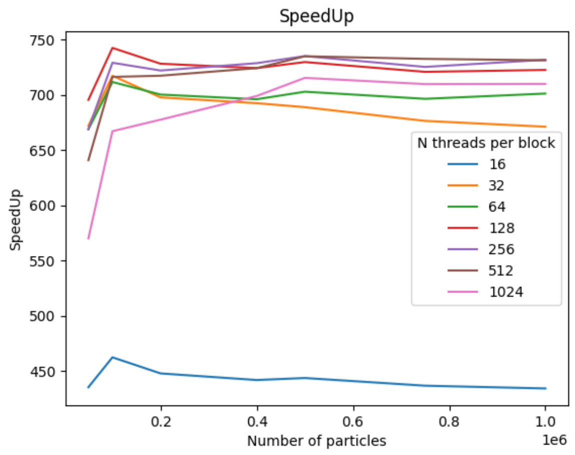


Figure 3: Speed Up curves

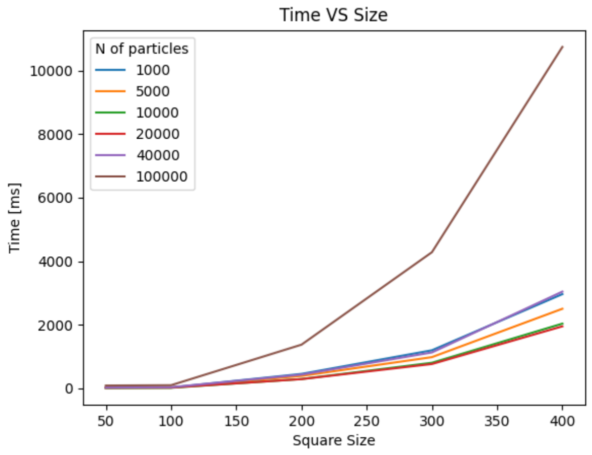


Figure 4: For every setup $BlockDim = 256$ and $d(P_{start}, P_{exit}) = 1.5L$, where d is the Manhattan distance between the start and exit points and L is the size of the maze

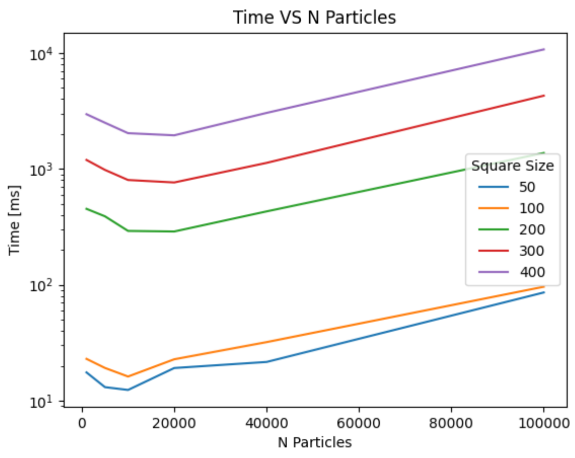


Figure 5: Same setup explained in Figure4.

6 Conclusions

Speed-Up plot shows the performances of the algorithm, when the number of particles and the number of active thread per block change. As we can see, GPU version of Maze Random Solver performs much better than the sequential version; it is ≈ 700 times faster than the CPU version and this result appears to be independent of the number of particles (if this number is large enough). Furthermore, the program shows better performances when the number of active threads is in the range $128 - 512$. Note that when $N_{THR} = 16$ (value below the warp dimension), half of the computational resources are unused and we have loss in performance, as shown in Figure3.

Figure4 shows the performances when the size of the maze changes: the greater the size, the longer the time to reach the exit. Furthermore, an high number of particles (threads), does not guarantee the best performance: this is because there is a trade-off between speed and steps to reach the exit. In fact, when the number of particles increases, the workload increases too (higher time execution) while the number of steps to reach di exit decreases.

Figure5 shows this trade-off: there are plotted time

VS N Particles curves (for each maze size). Every curve has got a minimum, where the best performance is reached.