Elliott Thorley

12/6/21

CS5260

**Parallel Maze Generation**

**Description**

The goal of this project is to take an existing maze generation algorithm that runs in serial and converting it to run in parallel, then comparing the results. Some of the reasons for doing this are: creating a better visual for how code runs in parallel vs serial because you will be able to see which thread generated which part of the maze, comparing the timings of serial vs parallel, and discussing the steps in the conversion process from serial to parallel.

**Research & Design**

There are many maze generation algorithms, here is a list of some of them:

• Randomized depth-first search

• Randomized Kruskal’s algorithm

• Randomized Prim’s algorithm

• Wilson’s algorithm

• Aldous-boarder algorithm

• Recursive division method

• Recursive backtracking method

Because I will not be creating the maze generation code, the method of maze generation does not matter all that much to me. The one I ended up using was recursive backtracking because the C implementation of it that I found was very user friendly and so the converting of the program into parallel will be easier to do.

The way the program will run in parallel will be through the use of OpenMP which is a C library designed for parallel computation. The reason for choosing this over Cuda is because Cuda is less intuitive to work with, so someone reading the code will have a harder time understanding the parallel portions, compared with if they were reading OpenMP code.

**Implementation**

Making the maze generate in parallel has several approaches. The first would be to have multiple processes working towards the generation of 1 maze, or to have multiple processes making their own maze, that then gets connected to create 1 cohesive maze.

In my program, I chose option 2. This option has certain benefits that make it to better option for this project. By having each process make its own maze that then connect to make one large maze, we can set up the entry and exit points to alternate sides each process, and then stack the mazes. This allows for a very clear understanding of where each processes maze starts and ends. This is a very visual element, and is better explained in the slides.

The timing of the maze generation is different from the timing of any program I have done in the past. This is because I want to only part of the parallel section, of multiple processes. I then add it up on thread 0 in order to get the total time taken. This does have a drawback however: when using seconds, if the total time is .5 seconds, it will round to 1 second. This issue becomes amplified when using lots of processes as that 0.4 second difference gets applied to each individual process. The way around this is to not do any of the timings in seconds and instead only use microseconds. This will allow for much more accurate timings and overall a better understanding of the data.

**Testing/Samples**

Raw Data:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N size | Threads | 1 | 5 | 10 | 20 | 50 | 100 | 200 |
| 10000 | Test 1 | 411 | 9396 | 19603 |  |  |  |  |
|  | Test 2 | 403 | 10780 | 24964 |  |  |  |  |
|  | Test 3 | 409 | 8661 | 22108 |  |  |  |  |
| 20000 | Test 1 | 781 | 17393 | 49687 | 60234 |  |  |  |
|  | Test 2 | 764 | 16374 | 50097 | 55090 |  |  |  |
|  | Test 3 | 768 | 12254 | 46852 | 67081 |  |  |  |
| 50000 | Test 1 | 1806 | 48089 | 114602 | 161788 | 113083 |  |  |
|  | Test 2 | 1801 | 45999 | 113403 | 157226 | 112197 |  |  |
|  | Test 3 | 1810 | 45503 | 114491 | 126771 | 122446 |  |  |
| 100000 | Test 1 | 3563 | 127235 | 205486 | 276354 | 263765 | 262480 |  |
|  | Test 2 | 3555 | 134913 | 220433 | 306236 | 261170 | 315973 |  |
|  | Test 3 | 3513 | 130352 | 215883 | 283289 | 330963 | 223589 |  |
| 500000 | Test 1 | 17258 | 695447 | 1356693 | 1795794 | 1571214 | 1415254 | 1103605 |
|  | Test 2 | 17268 | 689555 | 1277202 | 1770560 | 733613 | 1824070 | 694697 |
|  | Test 3 | 17232 | 667530 | 1374768 | 1748918 | 1843546 | 1642150 | 853363 |
| 10000000 | Test 1 | 34408 | 1363047 | 2951913 | 3712689 | 3737632 | 3328506 | 2164343 |
|  | Test 2 | 34634 | 1365523 | 2658286 | 3258293 | 3599048 | 2874710 | 2620223 |
|  | Test 3 | 34326 | 1347777 | 2580397 | 3324736 | 3585532 | 3108673 | 2386798 |

Averaged Data:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Threads | 1 | 5 | 10 | 20 | 50 | 100 | 200 |
| NSize |  |  |  |  |  |  |  |
| 10000 | 407.6666667 | 9612.333333 | 22225 |  |  |  |  |
| 20000 | 771 | 15340.33333 | 48878.66667 | 60801.66667 |  |  |  |
| 50000 | 1805.666667 | 46530.33333 | 114165.3333 | 148595 | 115908.6667 |  |  |
| 100000 | 3543.666667 | 130833.3333 | 213934 | 288626.3333 | 285299.3333 | 267347.3333 |  |
| 500000 | 17252.66667 | 684177.3333 | 1336221 | 1771757.333 | 1382791 | 1627158 | 883888.3333 |
| 10000000 | 34456 | 1358782.333 | 2730198.667 | 3431906 | 3640737.333 | 3103963 | 2390454.667 |

Speedup Data:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Threads | 5 | 10 | 20 | 50 | 100 | 200 |
| NSize |  |  |  |  |  |  |
| 10000 | 0.04241079169 | 0.01834270716 |  |  |  |  |
| 20000 | 0.05025966407 | 0.01577375269 | 0.01268057345 |  |  |  |
| 50000 | 0.03880622676 | 0.01581624311 | 0.01215159774 | 0.01557835767 |  |  |
| 100000 | 0.02708535032 | 0.01656429865 | 0.012277697 | 0.01242087258 | 0.01325491682 |  |
| 500000 | 0.02521665923 | 0.01291153684 | 0.009737601387 | 0.0124766987 | 0.01060294493 | 0.01951905689 |
| 10000000 | 0.02535799823 | 0.01262032702 | 0.01003990203 | 0.009464016996 | 0.01110064778 | 0.01441399433 |

Efficiency Data:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Threads | 5 | 10 | 20 | 50 | 100 | 200 |
| NSize |  |  |  |  |  |  |
| 10000 | 0.008482158338 | 0.001834270716 |  |  |  |  |
| 20000 | 0.01005193281 | 0.001577375269 | 0.0006340286725 |  |  |  |
| 50000 | 0.007761245352 | 0.001581624311 | 0.0006075798872 | 0.0003115671534 |  |  |
| 100000 | 0.005417070064 | 0.001656429865 | 0.00061388485 | 0.0002484174516 | 0.0001325491682 |  |
| 500000 | 0.005043331846 | 0.001291153684 | 0.0004868800694 | 0.0002495339739 | 0.0001060294493 | 0.00009759528447 |
| 10000000 | 0.005071599645 | 0.001262032702 | 0.0005019951013 | 0.0001892803399 | 0.0001110064778 | 0.00007206997163 |

Averaged Data Line Graph:

Chart, line chart

Description automatically generated

**Goals Reached**

Achieved:

* Serial Maze Generation
* Parallel Maze Generation
* Timing the Parallel Maze Generation
* Get User Input for Threads

Not Achieved:

* Get User Input for Schedule Types

**User Guide**

There is a readme file in the project folder that explains how to compile and run the code. Other things of note: the height and width of the maze will change in order to fit the maze generators limitations. This means that you only have loose control over the height and width of the maze, but it also prevents crashing. Also, do to a time constraint, I am unable to polish the program to the extent that I would like, because of this, there are a few known bugs:

* Program gets stuck while displaying the mazes and has to be stopped
* The time displayed overflows and becomes a negative number
* The size of the maze scaling incorrectly

**References**

Joe Wingbermuehle 1999 08 05 maze.c v1 source code <https://github.com/joewing/maze/blob/master/maze.c>

An Introduction to Parallel Programming. Peter S. Pacheco, Elsevier 2011 (ISBN: 978-0-12-374260-5)