Title page

Project title: Ant simulation

Module code: CMP202 2023/24 Term 2

Mini project 1, CPU

Student name: Cassie Parsons

Student ID:2201287

Table of contents

Contents

Contents

Introduction

I have created a program that simulates a number of ants trying to find food and bringing it back to the colony. The ants move largely at random, but will leave trails of pheromone, which will influence which way they turn, as more ants begin to find food and make their way home, ideally, they will follow the existing trail of pheromones and leave more pheromones, making that route even more appealing.

Application overview

My application simulates a large number of ants. On each tick, it will move each ant by one step, and then evaporate any pheromones. To move the ant, it get a random angle to turn, and samples an area around it for pheromones, the type of pheromone, distance from the pheromone and pheromone strength all affect the final pheromone strength it has, which is added to the random angle (the sign of pheromone strength determines if the ant turns left or right). If the pheromone strength is larger than an arbitrary value, the ant instead turns in whatever direction the pheromone is as far as it can. Ants also leave a trail of pheromones as they walk.

There are 2 types of pheromones within the simulation. Food pheromones and home pheromones. Ants that are looking for food will only listen to food pheromones and will leave trails of home pheromone, and ants with food looking for home will only listen to home pheromones and leave trails of food pheromones, this is so the path ants took to find food can then guide them home and vice versa.

If an ant comes within 5 units of a piece of food, it will walk towards it and pick it up. They then won't pick up any more food and start looking to go home. Currently they always know what direction home is, which overrides the pheromones. This was because the number of ants leaving home pheromones was overwhelming the ants. This would be fixed by carefully tweaking the numbers, but that would require a lot of time to carefully balance all the values.

The program runs through all the ants using a task farm to run several ants in parallel, and then once it has simulated every ant, it passes through both evaporation arrays and evaporates pheromones. This is a single "tick" of the program, and it repeats this continually.

My application makes use of 2 task farms, one for the ants, and one for the pheromone evaporation. These each have their own "worker" function and run a total of 16 threads.

The program simulates the number of ants using a task farm, this has multiple queues of tasks, one for each thread. Which prevents the mutexes on getting tasks becoming a large bottleneck as threads wait their turn to get a task. There is a similar task farm set up for the pheromone evaporation. When new tasks are added, they are split evenly between the queues.

The program makes use of lock guards within the pheromone manager where the ants are writing to the pheromone array. This prevents 2 ants from adding their pheromones to the same cell if they were to overlap.

Parallelisation Strategy and implementation

The program uses 2 different task farms to simulate the ants and the pheromones. To avoid the overhead of continually creating and destroying threads, the worker threads run continually and

are detached at the start of the program, since I don't need to join them to wait for a thread to finish execution.

To prevent the worker threads starting early, the task farms wait on a condition variable, this isn't notified until all the parts of the simulator are set up. I initially had issues with the condition variables, where only one would continue after the notify_all function was called, I solved this by manually unlocking the unique_lock the condition variables after the condition variable so all the threads could execute.

The ant task farm creates a vector of queues, one queue for each thread. Each thread takes tasks from its own queue. This reduces the amount of mutex bottle necks that exist within the program. Each thread gets a task, which is an integer index of which ant it's supposed to calculate, then performs that ant's AI calculations. When the ant is taking food, or leaving pheromones, I've used mutexes to prevent 2 ants from trying to take the same piece of food or leaving pheromone on the same pixel. These practically rarely happen, since for both ants to write to the same pixel of food or pheromone, they'd need to be on the same pixel.

Another thing I did to reduce the bottlenecks mutexes could pose was in the random number generation, instead of generating random numbers continually, at the start of the program running, it generates several arrays of random numbers, one array for each thread, and cycles through those. Each thread worker is given it's own index, which is how it knows which array to take from, when it reaches the end of the array, it cycles back to the start. This means getting random numbers also doesn't need to be protected, since the Mersenne twister (the random number generation method I use) isn't thread safe.

To evaporate the pheromones, I made use of a second task farm, this was because I had to evaporate the pheromone on every single pixel within the screen. Which was a lot of pixels. The task farm for evaporating the pheromone worked nearly identically to the ant task farm, it just had a different task that it performed. The task of evaporating the pheromone is a lot simpler than the ants AI though, all it must do is subtract a value from the value at the index it is working on.

Since each tick of the simulator had to calculate the ants, then the pheromone, once each. Each task farm was pausable, this was done with a check at the start of the worker thread loop, if that task farm was supposed to be paused, it would reset to the top of the loop. This could've been better achieved with a condition variable, since it would mean the threads would wait until they were unpaused, but since I had issues with condition variables previously, I went with an easier approach. The loop for each tick would unpause the ant farm, then once the ant farm was done, pause it, and unpause the pheromone farm.

Since all the threads were detached, when the application was closed, I couldn't just call join on each of the threads. To prevent the threads running when the main thread had ended, each of the task farms has a while loop that continually runs based on a boolean variable. In the destructor of the task farm, it sets that boolean to false, so each of the task farms exit the while loop and return, which ends the function and frees up that thread. This prevents thread leaks.

Performance evaluation

My CPU is an Intel Core i7 – 10750H CPU. It has a base speed of 2.6 Ghz with 6 cores and 12 logical processors.

To test the performance of the application, I'm letting it run for a period of time, and the application is calculating how long it takes to simulate one tick.

I cannot test how the application would run on a single thread, since it has 2 separate task farms, it would not be possible to run this using one thread. Instead, I've varied the number of threads each task farm works with.

10'000 ants

Ant threads	Pheremone threads	Average ms
1	1	81
2	2	61
4	4	62
6	6	51
8	8	52
16	16	53
32	32	56
8	4	50
10	2	63
11	1	63

100'000 ants

Ant threads	Pheremone threads	Average ms
1	1	396
2	2	282
4	4	218
6	6	191
8	8	181
16	16	175
32	32	165
8	4	193
10	2	175
11	1	178

Conclusion

Working with multiple threads has been an interesting and new challenge. It has some powerful benefits, such as significant speedups in processing large volumes of data, as well as several programming techniques that wouldn't be possible otherwise, such as doing game calculations whilst rendering to the screen. It also poses new problems and challenges, such as data atomicity, and the execution of a program needing to be protected more since one thread could run faster or slower than expected. There are techniques that allow the programmer to ensure threads are used safely, some of which I made use of within my program.

The ant simulation has been a very interesting project to work on and is something I want to continue after the project is done. I've made some mistakes in the way the ants read

pheromones and let it affect the way they turn, and that has resulted in ants following the trail of pheromones the wrong way. I would like to do more research into how existing ant simulations do this and try to make a more effective ant simulation in the future.