CSCE 435 Summer 2025

HW 3: Find the Anthill

I have an ant problem. There is an anthill in an unknown location in my back yard. Ants come out of the anthill at the rate of one per hour and wander off. Over time, they have proliferated across the yard, and are now the cause of innumerable ant bites. The anthill is very well hidden under grass but if I can find the anthill I will be able to fix this problem once and for all, or at least until next summer.

I observed that if I divide my backyard into small square tiles or cells, I can model the behavior of the ants very accurately. An ant moves from its cell to one of the four neighbors with equal probability, i.e., a probability of 0.25, once every hour. If it happens to walk off my lawn into my neighbor's lawn, it never comes back.

I have developed a code to model my lawn as a 2D array where each element contains the expected number of ants in the corresponding cell. I start the simulation with a lawn that has a single ant at the anthill. Every hour a new ant is added to the anthill cell. The array is also updated as ants move to neighboring tiles. If the simulation is allowed to run for a sufficient number of steps, the ant distribution over the lawn reaches an equilibrium.

I am sure that by estimating the number of ants at a few tiles I can figure out where the anthill is likely to be. My code takes as input the number of cells along each dimension of the lawn, the coordinates of the anthill, and the number of steps to simulate. It then simulates the ant distribution to calculate the final distribution of ants in the lawn. The code has the following routines that can be used:

- `number_of_ants_in_cell(i,j)` – returns the expected number of ants in cell (i,j);
- `guess_anthill_location(i,j)` – returns true if the anthill is at cell (i,j), false otherwise;
- `initialize_Lawn(m,x,y,k)` – simulates a lawn with m x m cells and an anthill at (x,y) for k steps starting from the beginning.

By using these calls selectively, one should be able to home in on the location of the anthill. The catch is that each of these calls has a cost associated with it which makes simplistic approaches such as exhaustively guessing for each cell very expensive. The code keeps track of how many times each routine was called in the course of finding the anthill. The final cost depends on the cost of these calls as well as on the execution time. (More details will follow.)

You have to develop an algorithm to determine the location of the anthill using the simulation code. You are allowed to modify the code appropriately, without altering the core functionality, to implement your strategy. You must use OpenMP to parallelize your routines to ensure that the execution time is minimized.

The code provided to you can be compiled on Grace as shown below:

```
icpx -o anthill.exe anthill.cpp -qopenmp
```

and executed are shown below:

```
export OMP_NUM_THREADS=4
anthill.exe 128 64 64 12800
```

The code creates a file with lawn data that can be used in MATLAB to visualize the ant distribution via the following commands.

```
load Lawn_Data;
m = sqrt(length(Lawn_Data));
X = reshape(Lawn_Data(:,1),m, m);
Y = reshape(Lawn_Data(:,2),m, m);
Z = reshape(Lawn_Data(:,3),m, m);
figure(1); clf; surf(X, Y, Z);
```

MATLAB can be invoked on Grace via the HPRC portal at https://portal.hprc.tamu.edu using the pull down menu for Interactive Apps.

1. (80 points) Develop an OpenMP-based implementation to determine the location of the anthill using the code provided with the assignment.
2. (20 points) Your code should compute the solution quickly and efficiently, utilizing all available processors on a single compute node of Grace. 20 points are reserved for an efficient parallel implementation. Testing will include problems with a variety of lawn sizes and anthill locations.

Submission: Upload **two** files to Canvas:

1. Submit the code files as a **single zip file.**
2. Submit a PDF/Word document with a brief description of your strategy. Also include details on how to compile and execute the code.

Helpful Information:

1. Load the Intel software stack prior to compiling and executing the code. Use:

   ```
   module load intel
   ```