Sawyer Cowan
CSCE 435

Anthill Problem - Homework 3

Instructions

- Attached to the submission is a .zip file containing my version of the anthill.cpp
- To compile my code:
  module load intel/2017A
  lcpc -o anthill.exe anthill.cpp -qopenmp
- To execute my code:
  export OMP_NUM_THREADS=<number of threads desired>
  ./anthill.exe <length of lawn side> <anthill x coord> <anthill y coord> <# steps>

The Problem

To begin, we know a couple factors about the anthill: in the main function, we know the size and we can get access to each thread through OpenMP, and we also know that it is better to query the number of ants at a location rather than query whether the location is the anthill source. One more fundamental point to note is that the area we are working with is a **square**.

The Approach

The first method that was provided was the brute-force method. Essentially, tell every thread to start at square (0,0) and guess every location until it's right. This obviously results in the easiest algorithm, but it has a worst case of O(n^2) (where n is the size of one side) and the penalty is much worse.

The second method I tried was to query first whether the number of ants at a location is greater than or equal to one. If the number of ants was less, then it won't bother guessing that location. **We know for a fact that the source of the anthill has to be at least one after any number of steps.** Therefore, we can reduce the number of guess-penalties by a very large factor.

The number of guesses reduced is correlated with the number of steps/ants in the lawn. The second algorithm is less effective if the number of ants gets very large.

The third method is a slight modification of the second method: we optimize our run-time by splitting up the work amongst multiple threads. For example, assume we have a Lawn size of 16x16 and the number of threads we give the program is 4. Since we know there are 196

squares total in the lawn, then each thread would only have to search through 49 squares.We can use the following algorithm in the for-loops to calculate where each thread will start.

```
271  #pragma omp parallel for default(none) shared(MyLawn, found, size)
272      for (int i = (omp_get_thread_num()*(size*size/omp_get_num_threads())) % size; i < MyLawn.m; i++) {
273      for (int j = (omp_get_thread_num()*(size*size/omp_get_num_threads())) / size; j < MyLawn.m; j++) {
274          // double local_max = 0.0;
275          if (found == 0) {
276              // local_max = MyLawn.number_of_ants_in_cell(i,j);
277
278
279          if (MyLawn.number_of_ants_in_cell(i,j) >= 1) {
280              // omp_set_lock(lck);
281              // shared_max = local_max;
282              if (MyLawn.guess_anthill_location(i,j) == 1)
283              {
284                  found = 1;
285                  // omp_unset_lock(lck);
286                  // omp_destroy_lock(lck);
287                  #pragma omp flush(found)
288
289              }
290              // omp_unset_lock(lck);
291          }
292          }
293      }
294      }
```

Each thread in OpenMP knows its thread number as well as the number of threads given to the program. Therefore, we can use the modulo (%) expression to calculate the x-coordinate (or i in this case). Likewise, the division (/) expression can calculate the y-coordinate (j in this case). Once each thread knows its starting position, each thread will implement the second method's algorithm of asking the value of the cell before guessing the anthill's location. The runtime of the third method is much faster than the single-threaded method because as lawn size increases linearly, the runtime increases in a squared-fashion. However, dividing up the lawn into subsections for each thread, we can find the anthill much faster.

The "Better" Approach

One other method was attempted to be implemented. While thinking more about the anthill problem, I thought there could be one more improvement done. Splitting up the work amongst multiple threads is better than not, but the queries for guess_anthill_location() could be reduced further. OpenMP has the ability to include a mutex-type object specific for OMP threads. I attempted to instead change Line 279 of anthill.cpp to instead compare whether a specific number of ants in a location is larger than what the thread has seen already. After visually checking my Lawn_Data, I found that the number of ants increases the closer it gets to the center. Instead, I wanted to have a global maximum variable that was shared amongst all threads. Then, each thread would check the number of ants at their spot and compare it to the global max. If any of them found their ant value to be larger than the global max, it would **lock**

the global value and assign the new maximum. This way, we don't run into repeated values being guessed over and over. Eventually one of the threads will find the absolute maximum, which happens to be the anthill location.

Issues with OpenMP Locks

A major issue I ran into with locking my "better" approach is that runtime would output a segmentation fault. I did some research, and it seems that OpenMP will often incur **stack overflow** when locking and unlocking thread sections. I made sure that my code did not overreach the boundaries of the lawn arrays, so I ended up giving up on the better approach. Hypothetically, I should have seen even less guess penalties as I tested the maneuvering of the anthill location.

Relations Found from Testing

- For the first method
    - The worst case for linear search is O(n^2)
- The second method
    - The worst case is O(n^2) and occurs when the anthill is in the last square with only **one ant** (one step)
    - Average case is most likely O(nlog(n)) or O(n) depending on the number of ants present
- The third method
    - The worst case is O(n^2/k), where k is the number of threads
    - The average case is most likely O(n) as the number of threads is more than one but not exceedingly large
- Hypothetical - "Better" Method
    - Not sure what O( ) notation would exist for this method but the runtime if guessed would be around O(log(n)) since there are multiple threads and assuming there are plenty of ants available to search for in the Lawn object.

- Overall, the penalties improved drastically from checking the number of ants to be greater than 1 before guessing. While we guessed more than what may have been than technically necessary, the penalty value improved vastly.