Cuda bonus question – Adam Levi

Preface: In this question I was asked to build a code that will calculate the mean square displacement for the random walk problem. The code had to be parallel. Meaning all the walkers had to be calculated in parallel.
I chose to use Cuda, which is Nvidia's parallel GPU computing tool. My environment was Matlab. The server I used was Tamnun.
In this file I shall explain my solution, the first part will be the technical part of using the server, and the second will explain the code itself.

Technical part: We want to connect the Tamnun server and use the node that the nice people in the faculty gave us.
First we login into Tamnun (we use -X to be able to use graphics):

ssh -X course03@tamnun

After logging into Tamnun we login into the the node that was allocated to us using ssh:

ssh gn002

When we are logged in we simply run Matlab:

matlab

Now we have our work environment, we simply add a new script inside Matlab and start to write our code, which will be explained in the following section.

Code explanation:

Our code is:

```
%% 1. Clearing
   clear all;
   close all;
   clc;

%% 2. Parameters:
   p = 0.5;                % probability to go right.
   q = 1-p;                % probability to go left.
   N_steps = 1000;         % Number of steps for each walker.
   N_walkers = 100;        % Number of walkers.

%% 3. Cuda Walk:
   mean_square = 0;
   parfor cnt=1:N_walkers                    % The parfor runs all the for iterations in parallel.
      prob_gpu = parallel.gpu.GPUArray.rand(1,N_steps);     % Created random vector in the GPU.
      Right_steps = sum(prob_gpu > p);                      % Calculates the number of right steps.
      final_pos = 2*gather(Right_steps) - N_steps;          % Returns the final pos from the GPU.
      mean_square = mean_square + final_pos^2/N_walkers;    % Calculates the mean square.
   end
```

I divided the code into three simple section, so it will be easy to follow. Those who do not know Matlab, notice that everything after a % symbol is a comment and is ignored by the compiler. I also painted it in green for simplicity. Following is the code explanation:

1. The first part is the easiest, I clear all the variables in the environment so there will be no collision with old variables, and close all open graphs, the third command is simply to clear the command line.

2. The second part of the code is also simple, we simply define the parameters for our code, so we will have easy access to change them. The first and second commands in this section define the probability for our random walkers, whether to go right or left, in our case, we used 0.5, so we have equal probability. The next two commands simply define the number of steps that each walker takes and how many walkers we want to calculate.

3. The third part is the core of our code, the first command is simply putting zero into our mean square variable since we always add the new value to the old one.
We have a loop here to calculate each of the walkers contribution to the mean square, notice that instead of *for* command we use a *parfor* this tells our compiler that we want the inside of the loop to be computed in parallel instead of in serial as loops usually computed.
Inside the loop, for each walker we randomize all of his steps. Instead of using the regular *rand* command we use *parallel.gpu.GPUArray.rand* , this makes Matlab created a randomized vector in the GPU instead of the CPU.
In the next command we calculate the number of right steps for our current walker, this is simple, we just take the random vector and sum the number of times that we got a number which is larger then *P* which is the probability to go right.
The line before last inside the loop simply calculates our final position, which is twice the number of right steps minus the number of total steps. Notice that we have used the *gather* command, this will retrieve the result from the GPU memory back into our regular memory.
The last command inside the loop is simply using our result to calculate the mean square using its definition, meaning summing the square final position and dividing by the number of walkers. Here we sum by iterations, meaning in each step adding to our last result (this is the correct way of thinking though it is calculated in parallel).

This is it, simple and working, we use *F5* to run our code, when we are finish our result is inside the *mean_square* variable, we can see it in the right upper window in our environment or simply right it in the command line and we will get it's value.
The theoretical value is 4*P*Q*N, where N is the number of steps. In our parameter this yields 1000. The more walkers we will use our numeric result will be closer to the theoretical one.