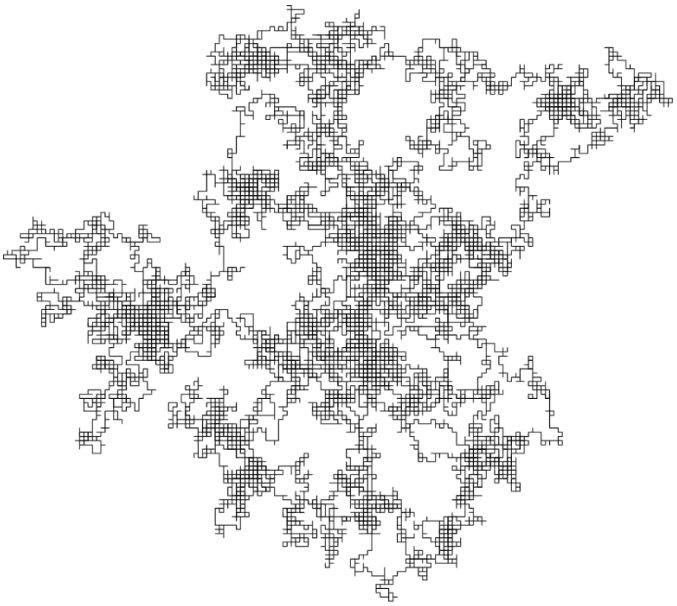
ECE 4122/6122 Lab 4: CUDA-based 2D Random Walk Simulation

(100 pts)

Category: CUDA
Due: Tuesday November 7th, 2023 by 11:59 PM



Objective:

Implement a CUDA program to simulate a 2D random walk. A random walk is a mathematical process that describes a path consisting of a sequence of random steps. In this assignment, you will simulate a large number of walkers taking steps either north, south, east, or west on a grid, and calculate the average distance they travel from the origin.

Description:

Create a C++ application using CUDA that takes as program input arguments the Number of Walkers, and the number of steps each walker needs to take on a 2D integer grid. Use command line flags to distinguish Number Walkers (-W) and (-I) for number of steps. All the walkers start at the origin (0, 0). Your application should implement three functions that internally use different memory models to perform the calculations.

1. (30 pts) cudaMalloc
2. (30 pts) cudaMallocHost
3. (30 pts) cudaMallocManaged

Your application should output to the console screen the total time each function took to perform the calculations and the average distance of the walkers from the origin and then exit.

(10 pts) Minimizing calculation times for all three functions.

Sample Program Flow:

```
>Lab4 -W 1000 -I 10000
> Normal CUDA memory Allocation:
>(4 spaces) Time to calculate(microsec): ???
>(4 spaces) Average distance from origin: ???.???
> Pinned CUDA memory Allocation:
>(4 spaces) Time to calculate(microsec): ???
>(4 spaces) Average distance from origin: ???.???
> Managed CUDA memory Allocation:
>(4 spaces) Time to calculate(microsec): ???
>(4 spaces) Average distance from origin: ???.???
>Bye
```

Turn-In Instructions

Zip up your file(s) into **Lab4.zip** and upload this zip file on the assignment section of Canvas.

Grading Rubric:

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will look through your code for other elements needed to meet the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	Up to 25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

Appendix A: Coding Standards

***Indentation*:**

When using *if/for/while* statements, make sure you indent **4** spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.