Dante Welch and Jonah Bukowsky[1]

[1] *Grand Valley State University*

The purpose of this paper is to discuss the construction of a CUDA-based massively multi-threaded application that computes and models effluent dispersion caused by point-source pollution. Various experiments will be discussed regarding the model, including scaling the problem from 1 dimension to multiple dimensions. Speedup of the parallelized version will also be discussed, as well as some visualizations of the final result.

I. INTRODUCTION

As the spec for the projects states, dispersion, or diffusion, is the movement of particles across concentration gradient, which continues until a density equilibrium has been reached. The diffusion equation is modeled via a partial differential equation that describes density dynamics. In this project, we have modeled diffusion as it would occur as pollutants entering a body of water. We used the finite difference equation as our mathematical model we used to model diffusion, which says the density at point x in the next time step is equal to the average density of its two neighbors at the current time step. When simplified, the equation looks like the following:

$$f_t(x,y) = \frac{f_{t-1}(x+\Delta x, y) + f_{t-1}(x-\Delta x, y) + f_{t-1}(x, y+\Delta y) + f_{t-1}(x, y-\Delta y)}{4}$$

Which essentially computes the average of the four points surrounding the current point. Computing this over very small differences in time and very small distances along the cylinder will allow for convergence to an accurate approximation of the concentration of pollutants in the body of water.

II. APPROACH

Two cylinders, and later grids when extended to two dimensions, but we will refer to it as a cylinder for the sake of theory, are created on the device. These cylinders are called ‘old’ and ‘new.’ The old cylinder is initialized with the starting values. We initialize the starting values on the device and the initializeValues() function acts as follows:

- Each thread initializes it’s own array index to 0 if it’s index is not an impulse.
- In the case where it’s index is an impulse, it sets that cell to the concentration of the impulse.

We were sure to initialize the values on the device so we can have a time complexity of O(K) (constant) rather than O(N) (linear), as copying back and forth from the host and the device is expensive.

To continue, the host loops through every time step and calls the device function to calculate the next timestep of the simulation. The calculateNext() function acts as follows:

- Each thread sets its cell in the array to the average of its neighboring cells in the old cylinder.
- For cells which are edges, we replace the value of the cell itself for each out of bounds value.

We iterate over each timestep on the host because the calls on the device are synchronized, which means that all threads finish calculating their values before the next time slice can begin. To check that our parallelized version works properly, we use CudaMemcpy to copy the desired cell value out of the final cylinder from the device to the host and compare it to the answer for the sequential version.

Using this logic for our one dimensional simulation, we were able to successfully extend our simulation to two dimensions and due to this, the visualizations seen through the paper will be multidimensional.

III. SPEEDUP ANALYSIS

For our one-dimensional solution, the max speedup achieved was 6.41. For our two-dimensional solution, the max speedup achieved was 140.45. The speedups most likely get even larger with larger execution time, however the sequential executions would take too long to measure. The one-dimensional speedup is not nearly as impressive as the speedup was for the two-dimensional grid. The reason for this is because the one-dimensional sequential solution is O(N), whereas the two-dimensional solution is O(N^2). This means that the sequential version for the two-dimensional grows at a much faster rate than the one-dimensional. The parallel versions for both are close to being O(K) (if the entire grid is smaller than the number of CUDA cores), so the speedup is much greater with the two-dimensional version. The following figures visualize our speedup for the 1 dimensional cylinder and 2 dimensional grid, respectively:

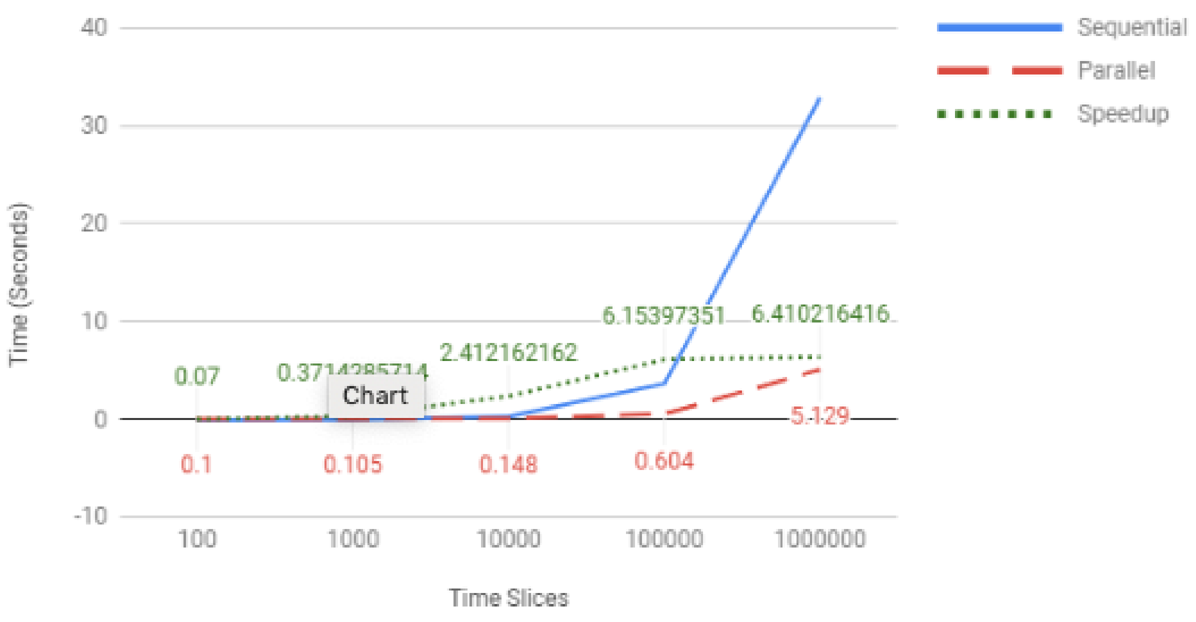

FIG. 1. Speedup for a 1-D Cylinder of 10000 Slices with a Single Impulse of Concentration 347092249.0

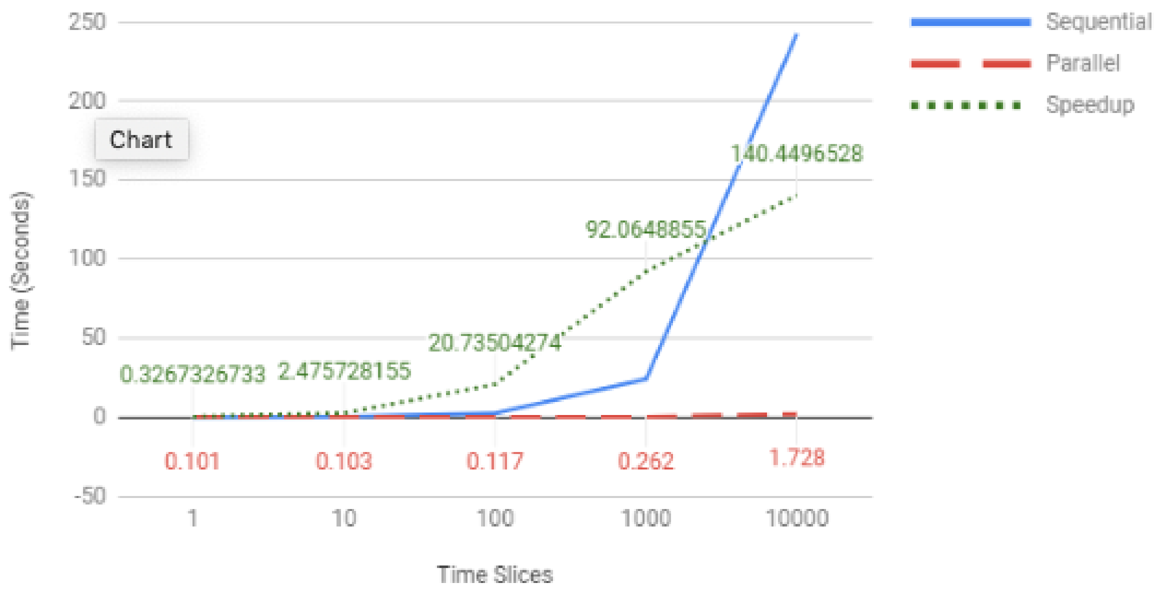

FIG. 2. Speedup for a 1000x1000 2-D Grid with One Impulse at Index (50,50) of Concentration 473174830.0

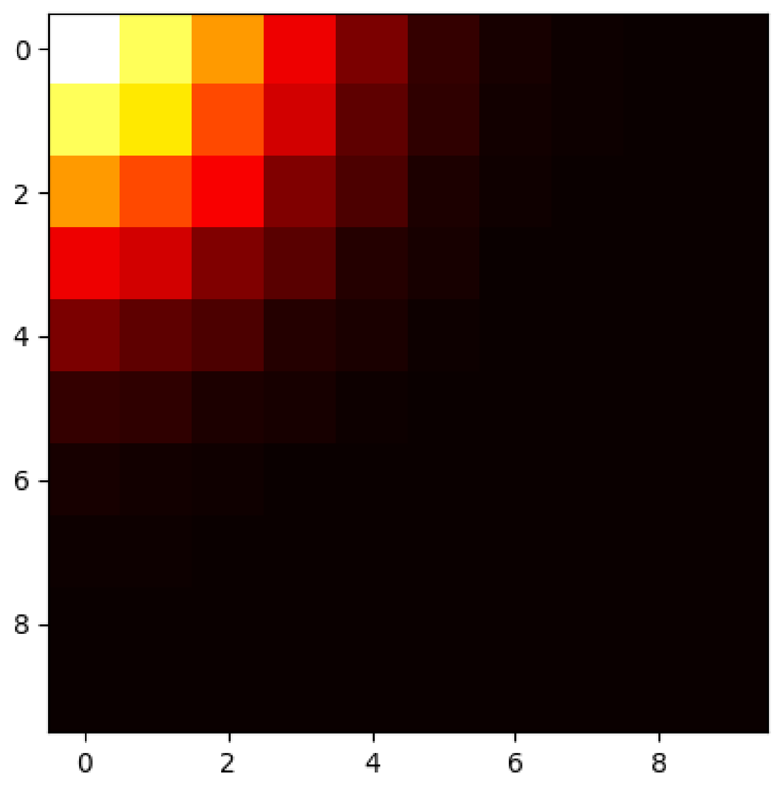

FIG. 3. Heatmap of a 10x10 2-D Grid with a Single Impulse of Concentration 1000.0

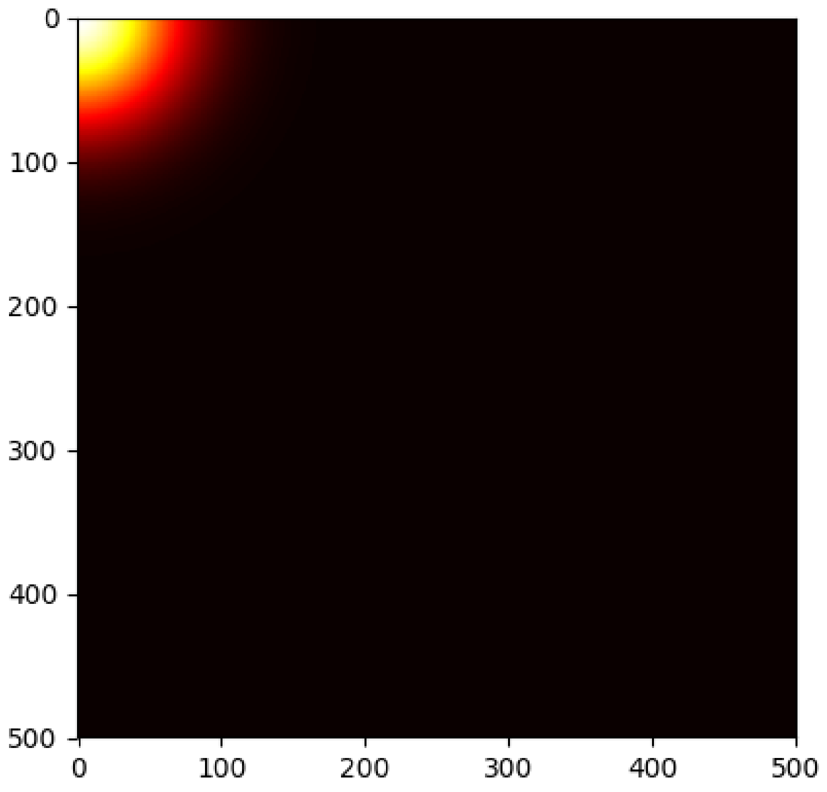

FIG. 4. Heatmap of a 500x500 2-D Grid with a Single Impulse of Concentration 10000.0

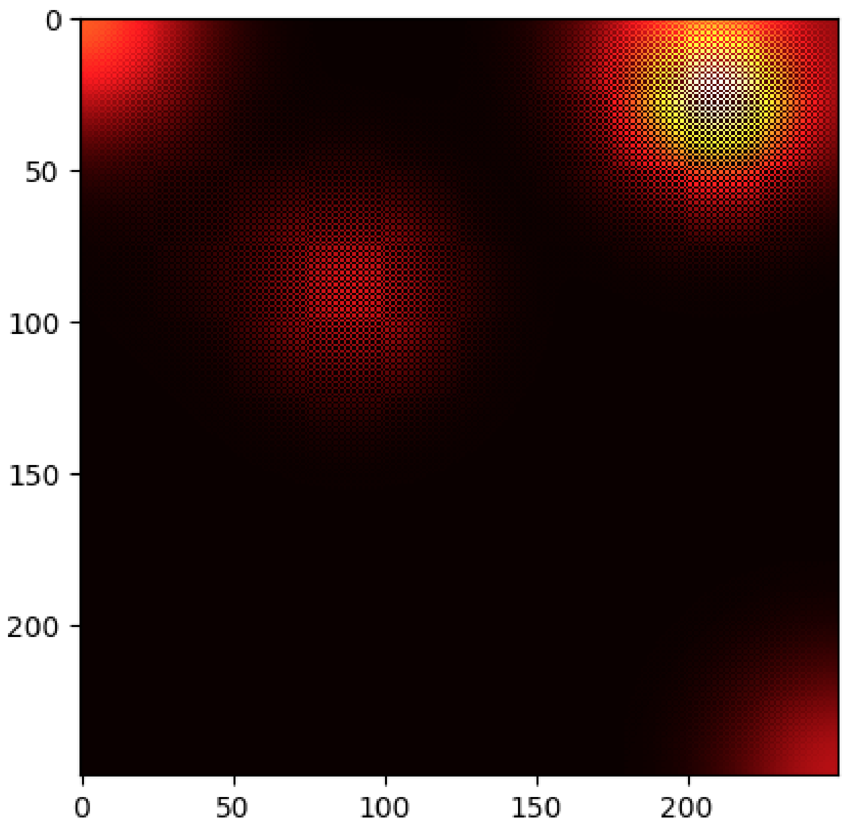

FIG. 5. Heatmap of a 250x250 2-D Grid with a Multiple Impulses of Varying Concentration

IV. CONCLUSION

If given more time, we would have wanted to extend our logic to three dimensions and create a visualization for it. In addition, it would be interesting to try to optimize how the visualizations for the heatmaps were done, as the method in which we chose to visualize them was quite slow and also involved writing and reading many files. Optimizing this process would allow for visualizations that take place over a longer period of time.