**Sorting Algorithms in CUDA: A Visual Analysis**

Anthony Delcimmuto, Chris Tatlonghari, Joe Snider

This document provides a deep dive in the analysis of sorting algorithms using Nvidia’s CUDA parallel software. In this analysis we will compare the same sorting algorithms ran on a CPU with ones on a GPU. We will also provide a sorting algorithm visualizer to see how these are sorted in parallel.

I. **INTRODUCTION**

In this analysis, we wanted to study different sorting algorithms and how they match up with one another. Specifically, we want to compare sorting algorithms between CPUs and GPUs to demonstrate the benefits of Parallel Computing. We also want to provide a visualizer that will visually indicate how the sorting algorithm processes an unsorted dataset.

For this project, we want to focus on 3 main sorting algorithms: Merge Sort, Radix Sort, and Bitonic sort.

To do this, we will have a script that will generate a dataset that is full of random integers or float values. When running this script, we will offer many parameters such as TEST\_CASE\_SIZE which will allow the user to generate a test case to their liking. This will be needed to give the user the ability to better visualize and have more control over the data going into the sort. We can then represent each specific float value as an individual color, or size of a line. At the start of the program, we render these individual colors and lines in the order of the randomly generated dataset. Then for every step in the sort operation, we update the rendering with the newly sorted set. One thing we need to be sure of is that we do not want to increase the work or step complexity of the sorting process specifically because of the render operation. We want to make sure we keep these as separate processes to maximize efficiency.

II. **Design and Optimization Approach**

* **Dataset generator script**: this script will be a single file that will randomly generate a dataset of floats that will be used as input for our sorting algorithms. It will generate an *input.raw* file to which our sorting algorithms will read from. When running a sorting algorithm, this file will be the first input i.e. *./a.out input.raw*
* **Sorting algorithms**: each algorithm will have its own specific file. It will include functions that are specific to both a CPU sort and a GPU sort of that algorithm. The program will have input options to enable the user to properly visualize the sort. We will also add a verbose option to generate more insight to the user on the sorting process.
* **Sorting visualizer**: I have no fucking clue.

III. **Division of work**