Challenge #17 – Systolic Array Bubble Sort

Course: ECE 410/510

Student: Balu Anudeep Kanipogu

Introduction

A systolic array is a highly parallel, regular network of processing elements (PEs), where data flows rhythmically through the array, resembling the pulsing action of a heart (hence "systolic"). Each PE performs simple operations and passes data to its neighbor. Systolic arrays are ideal for tasks with high data regularity and predictable data paths, such as matrix multiplication or sorting.

In this challenge, I explored bubble sort mapped onto a 1D systolic array. Each PE compares its value with its neighbor's and swaps if necessary. Multiple passes push larger values toward the end, effectively sorting the data. The goal was to implement, test, and analyze the performance of systolic bubble sort and compare it to Python's built-in sort.

Methodology

1. Algorithm Design:
I simulated a 1D systolic array using software, where:
- Each PE holds an element of the array.
- Each PE compares with its neighbor.
- After (n-1) passes, the array becomes sorted.

2. Software Implementation:
- I implemented the algorithm in Google Colab using Python.
- I tested the algorithm on random arrays of increasing sizes: 10, 100, 1000, 5000, 10000 elements.
- Execution time was measured using Python's time module.
- A performance comparison was plotted between systolic bubble sort and Python's built-in sorted() function.

Results

Correctness Verification:
Before: [54, 0, 54, 91, 74, 13, 0, 6, 60, 100]
After: [0, 0, 6, 13, 54, 54, 60, 74, 91, 100]

Execution Time Results:

Array Size	Systolic Bubble Sort Time
10	0.00001 seconds
100	0.00060 seconds

1000	0.08330 seconds
5000	3.34942 seconds
10000	8.31389 seconds

Discussion

The systolic array bubble sort worked as expected, providing a clear example of neighbor-to-neighbor data exchange. However, as expected, the performance degraded rapidly as array size increased, due to the $O(n^2)$ time complexity of bubble sort. Python's built-in sorted() dramatically outperformed it due to its O(n log n) complexity and heavy optimization in C.

Observations:
- Systolic bubble sort is ideal for educational and hardware modeling due to its simplicity and localized communication.
- It is not practical for large datasets in software due to poor scaling.
- The experiment visually demonstrates why algorithm selection is critical for large data sizes.

Conclusion

This experiment helped me understand both the theoretical and practical aspects of systolic arrays and their application to sorting.

Key learnings:
- Implementing systolic architectures in software can model real-world hardware behavior.
- Systolic bubble sort is an excellent teaching tool but not suited for high-performance software applications.
- Data size strongly influences execution time in quadratic algorithms.

This hands-on challenge reinforced the importance of algorithmic complexity analysis and gave insights into how systolic arrays can be used in custom hardware designs.