**Challenge #17**

1. Design a systolic array that can do Bubble sort. What dimension does the array need to have?

A systolic array is a network of processing elements (PEs) that rhythmically compute and pass data through the system. For Bubble Sort, the basic idea is:

* Each PE holds one number.
* In each clock cycle, adjacent PEs compare and possibly swap their values.
* To fully sort an array of n elements, we need to perform n-1 such passes.

**Systolic Array Dimensions**

* **1D Linear Array** of n Processing Elements is sufficient.
* **Array size**: n × 1 (i.e., 1D with n PEs).
* Time complexity on systolic array: still O(n²) cycles (one full pass per stage), but hardware-parallel.

1. (Vibe) Code a software version in your favourite language and test it.

Below is the GPU-compatible version using PyTorch in a Google Colab environment:

import torch

# Enable GPU

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

def bubble\_sort\_gpu(arr):

    n = len(arr)

    data = torch.tensor(arr, dtype=torch.float32, device=device)

    for i in range(n):

        for j in range(0, n - i - 1):

            if data[j].item() > data[j+1].item():

                temp = data[j].item()

                data[j] = data[j+1].item()

                data[j+1] = temp

    return data.cpu().numpy()

# Take user input

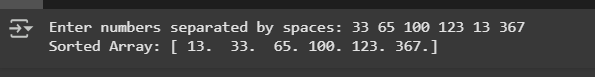
user\_input = input("Enter numbers separated by spaces: ")

arr = list(map(float, user\_input.strip().split()))

# Run bubble sort

sorted\_arr = bubble\_sort\_gpu(arr)

print("Sorted Array:", sorted\_arr)



Testbench: -

import unittest

import torch

# Bubble sort function (from your code)

def bubble\_sort\_gpu(arr):

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    n = len(arr)

    data = torch.tensor(arr, dtype=torch.float32, device=device)

    for i in range(n):

        for j in range(0, n - i - 1):

            if data[j].item() > data[j + 1].item():

                temp = data[j].item()

                data[j] = data[j + 1].item()

                data[j + 1] = temp

    return data.cpu().numpy()

# Unit test class

class TestBubbleSortGPU(unittest.TestCase):

    def test\_sorted\_array(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([1, 2, 3]), [1, 2, 3]))

    def test\_reverse\_array(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([3, 2, 1]), [1, 2, 3]))

    def test\_unsorted\_array(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([64, 34, 25, 12, 22, 11, 90]), sorted([64, 34, 25, 12, 22, 11, 90])))

    def test\_duplicates(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([5, 1, 2, 2, 3]), sorted([5, 1, 2, 2, 3])))

    def test\_empty\_array(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([]), []))

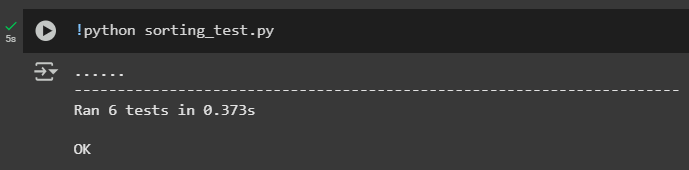
    def test\_single\_element(self):

        self.assertTrue(np.allclose(bubble\_sort\_gpu([42]), [42]))

if \_\_name\_\_ == '\_\_main\_\_':

    import numpy as np

    unittest.main(argv=[''], exit=False)



1. Visualize the execution times for various sorting sizes. E.g., 10, 100, 1000, 10000, etc.

import torch

import time

import matplotlib.pyplot as plt

import numpy as np

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

def bubble\_sort\_gpu(arr):

n = len(arr)

data = torch.tensor(arr, dtype=torch.float32, device=device)

for i in range(n):

for j in range(0, n - i - 1):

if data[j].item() > data[j+1].item():

temp = data[j].item()

data[j] = data[j+1].item()

data[j+1] = temp

return data.cpu().numpy()

sizes = [10, 100, 500, 1000]

times = []

for size in sizes:

arr = np.random.rand(size) \* 1000

start = time.time()

bubble\_sort\_gpu(arr)

torch.cuda.synchronize() if device.type == 'cuda' else None

end = time.time()

times.append((end - start) \* 1000)

plt.plot(sizes, times, marker='o')

plt.title("Bubble Sort on GPU - Execution Time vs Input Size")

plt.xlabel("Array Size")

plt.ylabel("Execution Time (ms)")

plt.grid(True)

plt.show()

