Input Array: [41 78 3 ... 17 77 32]

[GPU] Output Array (Left Rotated): [78 3 52 ... 77 32 41]

[CPU] Output Array (Left Rotated): [78 3 52 ... 77 32 41]

Validation Passed: GPU and CPU results match.

GPU Execution Time: 0.000169 seconds

CPU Execution Time: 0.000118 seconds

**Analysis of GPU and CPU Performance for Left Rotation Operation**

The performance comparison between the GPU and CPU implementations for the left rotation operation revealed that the GPU is slower than the CPU for the given dataset size of 1024 elements. This result might initially appear counterintuitive, given the parallel processing capabilities of GPUs.

One of the primary factors contributing to the slower GPU performance is the overhead involved in transferring data between the CPU and the GPU. For the dataset of 1024 elements, the time taken to move data from the host (CPU) to the device (GPU) and back dominates the computation time. This overhead becomes less significant only when the dataset size is large enough to allow the computational gains of parallel processing to overshadow the transfer costs.

Another factor is the fixed overhead associated with launching a kernel on the GPU. While this overhead is negligible for large workloads, it becomes a considerable fraction of the execution time for small datasets. In this scenario, the size of the input array is insufficient to justify the overhead of kernel initialization and execution.

Modern GPUs are designed for massively parallel workloads, with the capability to handle thousands of threads simultaneously across multiple streaming multiprocessors. In this implementation, only a single thread block with 1024 threads is used, which underutilizes the GPU's computational power. The GPU architecture thrives on higher concurrency levels, which are not achieved in this test due to the small dataset size.

Furthermore, the CPU implementation of the left rotation operation uses the highly optimized NumPy library, specifically the np.roll function. NumPy is optimized for vectorized operations on small to medium-sized datasets, making it extremely efficient for this specific task. Unlike the GPU implementation, the CPU version does not suffer from data transfer overhead or kernel launch delays, further contributing to its superior performance in this scenario.

Increasing the size of the input dataset to millions of elements would help distribute the overheads more evenly and highlight the computational efficiency of the GPU. Additionally, performing multiple operations on the data while keeping it resident on the GPU would reduce data transfer overhead and further leverage the GPU's strengths. However, the assignment states that we assume that we only have one thread block with N threads and the size of the array is also N (N <= 1024).