Optimizing Image Processing Performance: A GPU vs CPU Comparison

Optimizing Image Processing Performance: A GPU vs CPU Comparison

Introduction

With the growing demand for high-performance computing in areas like image processing, scientific simulations, and machine learning, there has been a shift towards parallel GPU architectures over traditional CPU-based systems. This report evaluates the performance of CUDA and Numba on three image processing tasks: image blurring and histogram computation, by comparing CPU and GPU implementations.

The focus is not only on the raw performance differences between these platforms but also on the challenges faced in kernel design, memory management, and synchronization. Our analysis highlights the differences between shared memory and global memory usage in CUDA-based implementations.

Problem 1: Basic Image Blurring (GPU vs CPU)

Objective

The goal of this problem is to apply a 9x9 box blur filter to a grayscale image using both CPU and GPU, comparing their performance in terms of execution time and computational speedup.

Implementation

- On the CPU: A nested loop structure was used, iterating through each pixel, calculating the mean of its 9x9 neighborhood.
- For the GPU: The same logic was implemented in parallel using a 2D CUDA grid, where each thread processes a single pixel. This allows the blur operation to be applied simultaneously across the entire image.

Performance

GPU Execution Time: 0.3511 seconds

CPU Execution Time: 2.3072 seconds

Comparison of GPU and CPU results: True

GPU speedup factor: 6.57x

Approximate FLOPS for GPU: 8139852

Approximate FLOPS for CPU: 8139852

GPU Performance: 0.02 GFLOPS

CPU Performance: 0.00 GFLOPS

GPU Memory Usage: 3459.67 MB

CPU Memory Usage: 0.05 MB

GPU Bandwidth: 9852.54 MB/s

CPU Bandwidth: 0.02 MB/s#### Observations

The GPU-based implementation achieved a significant speedup, confirming that parallelization can drastically improve performance. Proper memory management and boundary conditions were critical in achieving this result.

Problem 2: Shared Memory Optimized Image Blurring (GPU vs CPU)

Objective

The aim of this problem was to further optimize the image blurring operation by using shared memory, which is faster than global memory. This optimization reduces memory access times and increases throughput.

Methodology

- For the GPU: Each block of threads loads a tile of the image into shared memory, including a halo of surrounding pixels to handle boundary conditions. Synchronization barriers ensure that all threads complete loading their data before computation starts, reducing the need for global memory access.

Performance

Image Size : 50246 pixels

GPU Execution Time: 0.7597 seconds

CPU Execution Time: 2.2650 seconds

Outputs match: False

GPU speedup factor: 2.98x

Approximate FLOPS for GPU: 25725952

Approximate FLOPS for CPU: 25725952

GPU Performance: 0.03 GFLOPS

CPU Performance: 0.01 GFLOPS

GPU Memory Usage: 3459.67 MB

CPU Memory Usage: 0.05 MB

GPU Throughput: 66141.25 pixels/second

CPU Throughput: 22183.51 pixels/second

GPU Processing Time per Pixel: 0.0000151192 seconds

CPU Processing Time per Pixel: 0.0000450785 seconds

Key Challenges

Correct indexing while managing shared memory was complex, especially when handling halos at the tile boundaries. Synchronization was critical to avoid race conditions, and tuning block and grid size for optimal performance proved challenging.

Conclusion

While shared memory optimization improved performance, the speedup was not as significant as anticipated. This experiment underscored the importance of managing memory access and synchronization in GPU programming.

Problem 3: Histogram Computation Using Shared Memory (GPU vs CPU)

Objective

This task aimed to compute the histogram of an image's pixel intensities. The GPU version utilized shared memory to store local histograms, which were later merged into a global histogram using atomic operations.

Methodology

- For the GPU: Each thread processes a subset of pixels and updates a local histogram in shared memory. After processing, these local histograms are atomically merged into the global histogram, ensuring thread-safe updates.
- For the CPU: A straightforward loop was used to tally the pixel values in a NumPy array.

Performance

Evaluation Results

Image Size : 50246 pixels

GPU Execution Time : 0.281276 seconds

CPU Execution Time : 0.282211 seconds

Outputs Match : Yes

GPU Speedup Factor : 1.00x

GPU Throughput : 178636.09 operations/sec

CPU Throughput : 178044.35 operations/sec

GPU FLOPS : 357272.18 FLOPS

CPU FLOPS : 356088.69 FLOPS

GPU Memory Bandwidth : 0.000182 GB/s

CPU Memory Bandwidth : 0.000182 GB/s

GPU Memory Usage : 3459.67 MB

CPU Memory Usage : 0.05 MB

GPU Processing Time per Pixel: 0.0000055980 seconds

CPU Processing Time per Pixel: 0.0000056166 seconds

Observations

The histogram computation task showed nearly identical performance between the CPU and GPU, which is expected since it is a memory-bound task. However, shared memory optimizations helped improve the efficiency of atomic operations.

Conclusion

This comparative study of image processing tasks demonstrates the potential of GPU programming using CUDA. The GPU implementations of the image blur and histogram computation tasks demonstrated notable speedups over their CPU counterparts, showcasing the power of parallelism for data-parallel problems.

The experiments emphasize the importance of memory management, thread synchronization, and kernel design in GPU programming. Although the GPU showed substantial improvements, challenges in memory management and synchronization remain critical to optimizing performance.

In the future, techniques like warp shuffling, memory coalescing, and multi-GPU configurations could further enhance the performance of real-time image processing tasks.