Image Convolution: Scalar vs. OpenCL Implementation Analysis

Overview and Approach

My code implements image convolution for edge detection using two approaches: a scalar CPU-based implementation and an accelerated GPU-based implementation using OpenCL with SIMD optimizations.

Introduction to Convolution

Convolution is a fundamental mathematical operation in image processing that involves sliding a kernel (small matrix) over an input image to produce an output image. Each pixel in the output is calculated by multiplying elements of the kernel with corresponding input image pixels in the neighborhood and summing the results.

For an image I and kernel K, the convolution operation is defined as:

$$O(x,y) = \sum_{i,j} I(x+i, y+j) * K(i,j)$$

Where O is the output image, and the summation is performed over all positions (i,j) in the kernel.

Convolution Type Clarification

Both implementations focus on edge detection, specifically using vertical edge detection kernels:

1. The scalar implementation (q1.cpp) uses a vertical edge detection kernel:

   ```
   { 1,  0, -1 }
   { 1,  0, -1 }
   { 1,  0, -1 }
   ```

2. The OpenCL implementation (p1.cpp) uses a Sobel operator for vertical edge detection:

   ```
   { 1, 0, -1 }
   { 2, 0, -2 }
   { 1, 0, -1 }
   ```

Both kernels highlight vertical edges by calculating horizontal gradients, but the Sobel operator provides better edge detection with smoothing properties due to its weighted center row.

SIMD Optimization Details

Vectorization in OpenCL

The OpenCL implementation leverages SIMD (Single Instruction, Multiple Data) computation through:

1. **Vectorized Data Types**: The float4 data type processes four floating-point values in parallel:

   ```
   float4 sum = (float4)(0.0f);
   float4 pixel = vload4(0, &input[imgIndex]);
   float4 kernelVal = (float4)(filter[filterIndex]);
   sum += pixel * kernelVal;
   ```

2. **Work-Item Distribution**: The implementation uses a 2D global work size matching the image dimensions:

   ```
   cl::NDRange globalSize(width, height);
   ```

   This ensures each pixel is processed by an individual work-item, allowing massive parallelism across the GPU's compute units.

Memory Access Optimization

1. **Memory Type Selection**:

   Input image and output buffer use global memory (__global)
   Kernel/filter uses constant memory (__constant), which is optimized for read-only broadcast access
2. **Coalescing Techniques**:

   The vload4 function loads four adjacent pixels at once, enabling coalesced memory access patterns
   Work-items in the same work-group access adjacent memory locations, maximizing memory bandwidth utilization
3. **Boundary Handling**:

   The OpenCL kernel includes explicit boundary checking to prevent out-of-bounds memory access
   This ensures correctness while maintaining efficient memory access patterns

Performance Analysis

While specific execution times for various image sizes aren't provided in the code snippets, we can analyze the expected performance characteristics:

Execution Time Analysis

From the code structure, we can see both implementations measure execution time:

```
// OpenCL version timing
```

```
auto start = high_resolution_clock::now();
queue.enqueueNDRangeKernel(convKernel, cl::NullRange, globalSize, cl::NullRange);
queue.finish();
auto end = high_resolution_clock::now();
duration<double> elapsed = end - start;
cout << "Execution time (OpenCL): " << elapsed.count() << " seconds" << endl;
```

Q1(PART A SCREEN SHOTS)

```
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_Scalar$ g++ -o c q2_multiFiles.cpp `pkg-config --cflags --libs opencv4`
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_Scalar$ ./c
Processing: flickr_wild_000010.jpg (512x512)
Execution time: 0.012501 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000010.jpg
Processing: flickr_wild_000005.jpg (512x512)
Execution time: 0.013078 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000005.jpg
Processing: flickr_wild_000008.jpg (512x512)
Execution time: 0.012227 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000008.jpg
Processing: flickr_wild_000009.jpg (512x512)
Execution time: 0.011194 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000009.jpg
Processing: flickr_wild_000007.jpg (512x512)
Execution time: 0.011660 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000007.jpg
Processing: flickr_wild_000011.jpg (512x512)
Execution time: 0.011645 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000011.jpg
Processing: flickr_wild_000003.jpg (512x512)
Execution time: 0.012111 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000003.jpg
Processing: flickr_wild_000002.jpg (512x512)
Execution time: 0.012233 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000002.jpg
Processing: flickr_wild_000006.jpg (512x512)
Execution time: 0.012438 seconds
Saved: /home/hasnain-akhtar/Documents/A3/outputprocessed_flickr_wild_000006.jpg

Processing complete! Total images processed: 9
```

```
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_Scalar$ g++ -o a q1.cpp `pkg-config --cflags --libs opencv4`
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_Scalar$ ./a
Image loaded successfully: 512x512
Execution time: 0.020089 seconds
Processed image saved as output.jpg
```

Q1 PART B SCREENSHOTS

```
hasnain-akhtar@HasnainThinkbook: ~/Documents/A3/Q1_OPENCL
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_OPENCL$ g++ main.cpp -o convolution -lOpenCL `pkg-config --cflags --libs opencv4`

In file included from main.cpp:1:
/usr/include/CL/opencl.hpp:437:112: note: '#pragma message: opencl.hpp: CL_HPP_TARGET_OPENCL_VERSION is not defined. It will default to 300 (OpenCL 3.0)'
  437 | ET_OPENCL_VERSION is not defined. It will default to 300 (OpenCL 3.0)")
      |                                                                       ^

hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_OPENCL$ ./convolution
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000010.jpg in 0.00251112 seconds.
Saved: ./processed/flickr_wild_000010_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000005.jpg in 0.00122936 seconds.
Saved: ./processed/flickr_wild_000005_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000008.jpg in 0.00123985 seconds.
Saved: ./processed/flickr_wild_000008_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000009.jpg in 0.00124545 seconds.
Saved: ./processed/flickr_wild_000009_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000007.jpg in 0.00130034 seconds.
Saved: ./processed/flickr_wild_000007_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000011.jpg in 0.00118443 seconds.
Saved: ./processed/flickr_wild_000011_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000003.jpg in 0.00115441 seconds.
Saved: ./processed/flickr_wild_000003_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000002.jpg in 0.00124117 seconds.
Saved: ./processed/flickr_wild_000002_processed.jpg
Processed /home/hasnain-akhtar/Documents/A3/wild/flickr_wild_000006.jpg in 0.00148976 seconds.
Saved: ./processed/flickr_wild_000006_processed.jpg
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_OPENCL$
```

```
hasnain-akhtar@HasnainThinkbook:~/Documents/A3/Q1_OPENCL$ ./convolution
Execution time (OpenCL): 0.00199419 seconds
Processed image saved as output1.jpg
```

Expected Speedup Factors

The OpenCL implementation should provide significant speedup over the scalar version due to:

1. **Massive Parallelism**: Processing thousands of pixels simultaneously
2. **SIMD Execution**: Processing multiple data elements with a single instruction
3. **Specialized Hardware**: Utilizing GPU compute units optimized for parallel math operations

We would expect speedup factors ranging from 10x to 100x or more, depending on:

Image size (larger images typically show better GPU utilization)
GPU hardware specifications
Kernel complexity
Memory transfer overhead

Challenges and Solutions

1. Boundary Handling

Challenge: Convolution kernels require accessing pixels outside image boundaries when processing edge pixels.

Solutions:

The scalar implementation uses copyMakeBorder to pad the input image with replicated border pixels
The OpenCL implementation uses explicit boundary checking within the kernel

2. Memory Transfer Overhead

Challenge: Data transfer between CPU and GPU can be a bottleneck.

Solution: The OpenCL implementation minimizes transfers by:

Sending input image data once
Computing entirely on the GPU
Retrieving only the final result

3. Normalization and Output Processing

Challenge: Convolution output needs normalization for proper visualization.

Solutions:

Both implementations use normalization to rescale values to the 0-255 range
The OpenCL kernel applies fabs() to enhance edge visibility
Post-processing handles min/max value detection for proper scaling

4. Vectorization Limitations

Challenge: The vectorized approach using float4 requires special handling for dimensions not divisible by 4.

Solution: The current implementation uses a reduction step to combine vector elements, but additional optimization could include proper work-group sizing and boundary handling for optimal performance.

Conclusion

The OpenCL implementation successfully leverages SIMD parallelism and GPU acceleration to outperform the scalar CPU implementation. Key optimizations include:

1. Utilizing vectorized data types (float4)
2. Optimizing memory access patterns
3. Distributing computation across many work-items
4. Using specialized memory types (constant memory for the kernel)

For future improvements, you might consider:

1. Using local memory for collaborative filtering within work-groups
2. Implementing non-uniform work-group sizes optimized for the specific GPU architecture
3. Further optimizing boundary handling to reduce conditional branches
4. Exploring additional kernel optimizations like loop unrolling