One page report on

GPU-Accelerated Command-Line Tool for Image Enhancement Using OpenCL

Submitted by:

Biswajit Gorai

Roll-MA24M005

1. Technical Approach & Kernel Design

OpenCL Pipeline Overview:

The project uses PyOpenCL to execute two OpenCL kernels sequentially:

- **Kernel 1:** Applies a 3x3 Gaussian blur on the RGB channels.
- **Kernel 2:** Applies logarithmic tone mapping based on pixel luminance.

The host code loads an input image using Pillow, converts it to a `numpy.uint8` buffer, and sends it to the GPU via `cl.Buffer`. The final output is copied back and saved as a PNG.

Gaussian Blur & Boundary Handling:

To handle boundaries, I used `clamp()` to prevent out-of-bounds memory access—this replicates the nearest edge pixel, avoiding artifacts.

Logarithmic Tone Mapping:

Luminance `Y = 0.2126R + 0.7152G + 0.0722B` is computed and tone-mapped using: Y_out = log(1 + Y) / log(1 + max_luminance)

Then, each RGB channel is rescaled proportionally using the ratio `Y_out / Y`. Alpha values are preserved

---

2. Challenges, Learning & Problem-Solving

- **OpenCL Setup on ARM:** Installing OpenCL on an ARM CPU with POCL was challenging due to architecture-specific limitations and missing Visual Studio tools. Solved by using Conda-forge and WSL on Ubuntu.
- **Kernel Compilation Errors:** Early build errors in `.cl` were resolved by line-by-line isolation and use of PyOpenCL's `program.build()` error messages.
- **Data Type Conversion:** Careful float-to-uchar casting was needed to avoid visual artifacts.
- **Debugging Strategy:** I tested each stage independently (first blur, then tone map), used print-based debugging on CPU fallbacks, and validated results with test vectors.

Skills Gained:

- End-to-end OpenCL programming.
- GPU buffer management.
- Writing efficient image kernels.

---

3. Parallelism & Performance Considerations

Parallel Execution:

Each OpenCL work-item corresponds to a unique pixel (x, y), allowing both the blur and tone mapping kernels to process the image in parallel. This massive data parallelism makes GPU execution efficient.

Performance Factors (Theoretical):

- **Memory Access Pattern:** Gaussian blur accesses a 3x3 neighborhood—non-coalesced reads can impact cache usage.
- **Computational Intensity:** Tone mapping is more compute-heavy but memory-light.
- **Work-group Size:** Could be tuned to improve shared memory usage (not done in this version). Using local memory for filter weights may also help on real GPUs.

4.Project Closure & Future Work:

The project was brought to completion by dividing it into distinct phases: OpenCL setup, kernel development, host integration, image testing, and final report writing.For real-world extension, the core functionality can be enhanced by:

- **Use shared (local) memory** to optimize Gaussian blur by reducing global memory access.
- **Extend to advanced filters** like bilateral or guided filters for better visual quality.
- **Switch to perceptual color spaces** (e.g., LAB, HSV) for more accurate tone mapping.