Real-time Image Processing on Embedded Hardware using OpenCL

Muhammad Arief Kurniawan

December 2024

SAXION HOGESCHOOL

Contents

Chapter 1

Introduction

1.1 Background

Applications of real-time image processing span across industries such as security surveillance, autonomous vehicles, robotics, and augmented reality. Performing real-time image processing often requires significant computational power as each frame is processed within milliseconds.

Embedded systems are frequently used as the solution to real-time image processing for its portability and cost-effectiveness. Nevertheless, their limited computational resource pose challenges in achieving real-time performance, particularly when relying on traditional CPU-based processing.

Open Computing Language (OpenCL) provides a solution to this challenge by enabling parallel execution of intensive computational tasks on a variety of devices, including GPUs, FPGAs, and other embedded processors. This makes OpenCL an ideal choice for accelerating image processing operations on resource-constrained hardware.[2]

1.2 Problem Statement

Achieving real-time image processing on embedded systems is challenging due to the limited processing power and memory bandwidth. While CPUs on these devices are sufficient in completing computational tasks, they often lack the performance required for computationally intensive operations, such as edge detection or convolution, when applied to high-frame-rate video streams.

1.3 Objective

The goal of this project is to develop a real-time image processing application that utilises OpenCL to accelerate image operations, such as edge-detection and blurring on embedded hardware. This project demonstrates the performance improvements achieved by utilising OpenCL for parallel computation, compared to a CPU-only implementation.

Chapter 2

Scope and Deliverables

2.1 Scope

This project focuses on implementing a real-time image processing application on embedded hardware, whilst utilising OpenCL to accelerate computationally intensive tasks.

The scope includes:

1. Designing and implementing core image processing algorithms, such as edge detection and Gaussian blur, using OpenCL kernels
2. Integrating algorithms with a video capture and display pipeline to enable real-time processing of live camera feeds or pre-recorded video.
3. Optimising performance to achieve smooth processing for standard resolutions and frame-rates (e.g., 30 FPS).

This project does not cover advanced image processing such as object detection or machine learning enhancements. This project is limited to the OpenCL compatible devices, specifically for RaspberryPi or equivalent hardware.

2.2 Deliverables

Number	Deliverable	Description
D.1	Functional Application	A working real-time image processing application that captures video, processes frames using OpenCL, and displays or saves the output.
D.2	Source code and Documentation	Well-documented source code for the application, including OpenCL kernel implementations. A user guide explaining how to install, configure, and run the application on the target hardware.
D.2.1	Project Plan	Report that elaborates on the problem statement, objective, methodology, and planning of the project
D.2.2	Final Report	Report that showcases the findings, conclusion, and recommendations of the project.
D.3	Performance Benchmark Report	A report comparing the performance of CPU-only and OpenCL-accelerated implementations, including metrics like frame rate and processing time per frame.

Table 2.1: Table of Deliverables

Chapter 3

Methodology

The development of the project will follow an iterative agile inspired approach, combining aspects of SCRUM with technical prototyping. This methodology is chosen to ensure adaptability to challenges and frequent testing at every development phase.

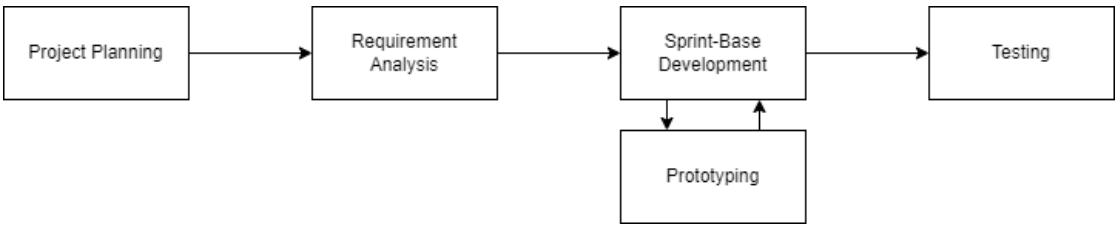

Figure 3.1: Iterative Agile Inspired Methodology

Rationale

The rationale for using this methodology are elaborated in four factors:

1. **Iterative Deployment**: OpenCL projects often require significant testing and optimisation. Iterative sprints ensure continuous progress while allowing flexibility to address performance issues
2. **Early and Frequent Prototyping**: Prototyping enables testing kernels and application features early in the development cycle, reducing the risk of last-minute issues
3. **Stakeholder Feedback**: Regular reviews ensure alignment with the project goals and help identify issues early
4. **Flexibility**: The hybrid approach accommodates the need to switch priorities based on the testing results

3.1 Project Planning and Requirement Analysis

This phase of the project is set to establish clear goals, milestones, and deliverables. To aid in managing the work done, Trello is used as the preferred project management tool for this project. Using Trello, project tasks are broken down into manageable work items. This planning phase and use of Trello helps structure the project, keeping it focused on and flexible as the project progresses.

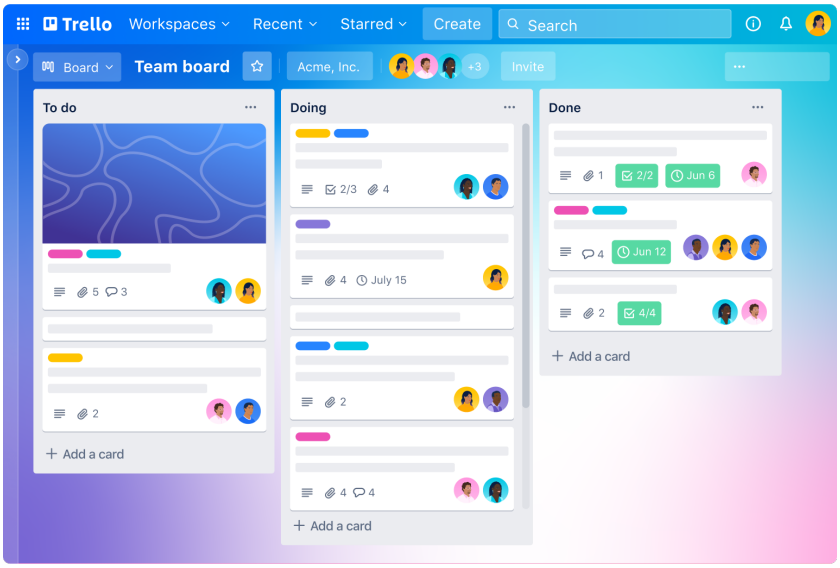

Figure 3.2: Preview of Trello Board

3.2 Sprint-Base Development

Instead of adhering to formal SCRUM cycles with daily team checks, this methodology adapts the principles of SCRUM for individual use. The project is divided into smaller focused phases each corresponding to a particular milestone such as kernel development or performance optimisation.

Phase Structure

Each phase can span **1-2 weeks** with a review at the end of each period to evaluate progress. The reviews are based on three factors; What was accomplished in the current (ending) phase, what will the focus be on the next phase, and what are the expected challenges. This approach encourages reflection and continual adjustment which provides the project with flexibility to make real-time adjustments without the formalities in SCRUM.

3.2.1 Sprints Overview

Number	Sprint	Duration (Weeks)	Description
OpenCL Fundamentals			
S.1	**Introduction to OpenCL**	**2**	- What is OpenCL - What are Heterogeneous Platforms - Conceptual foundations of OpenCL - Platform model - Execution model - Memory model
S.2	**OpenCL Programs and Kernels**	**1**	- OpenCL Platforms - OpenCL Devices - OpenCL Contexts - Kernel fundamentals
S.3	**Buffers and Sub-buffers**	**1**	- Memory objects - Creating buffers and sub-buffers - Mapping buffers and sub-buffers
S.4	**Images and Samplers**	**1**	- Creating image objects - Working with images in OpenCL
Real-time Image Processing Project			
S.5	**Environment Setup**	**1**	- Configure OpenCL and OpenCV on the Raspberry Pi - Verify hardware and software functionality by running test programs
S.6	**Kernel Development (Phase 1)**	**1**	- Develop the grayscale conversion kernel - Validate performance with test images
S.7	**Kernel Development (Phase 2)**	**1**	- Implement the edge detection and Gaussian blur kernels - Test kernels individually for accuracy and performance
S.8	**Application Integration**	**2**	- Integrate kernels with OpenCV for video capture, processing, and display - Implement a basic UI or command-line interface for switching filters
S.9	**Optimisation and Testing**	**1**	- Optimize memory usage and kernel execution - Test real-time performance with varying input resolutions and lighting conditions

Table 3.1: Table of Sprint Backlog [3]

3.3 Prototyping

As the project progresses, prototypes of the finished application and system are built after completing each major component. The prototypes allow the testing of individual parts of the system resulting in continuous troubleshooting and integration. The primary source of feedback will be the developer's individual observations, however, feedback from mentors and peers are considered as well. With continuous prototyping the project can be iteratively refined and avoids significant issues at final phases of development.

3.4 Testing

The testing phase is initiated only when the core development phase of the application is complete. This mainly involves **unit** and **integration** testing. The subsections below elaborate more on the testing types. For both tests, tools such as timers and logs are used to track execution time and ensure that performance benchmarks are met.

Unit Testing

Unit testing ensures that each component works as expected. Each component is measured to a metric defined at the each sprint phase.

Integration Testing

Integration testing ensures that each component implemented works well with other components. For this project, this mainly consist of integrating three factors; OpenCL, OpenCV, and any embedded hardware constrictions.

Documentation

Throughout the project, thorough documentation is performed to ensure that the results of the project are reproducible for potential future development.

Chapter 4

Results and Discussion

This section will focus on the anticipated outcomes of the project and how they will be evaluated and analysed. It will include both performance metrics and qualitative observations about the implementation process.

4.1 Expected Results

The primary result of the project will be a functional real-time image processing application running on embedded hardware. Key deliverables include the successful implementation of OpenCL-accelerated image processing kernels and their integration into a video pipeline capable of processing live or recorded footage.

Quantitative performance improvements are expected when comparing the OpenCL-accelerated implementation to a CPU-only baseline. Specifically, the following outcomes are anticipated:

1. **Frame Rate**: The application should sustain a frame rate of at least 30 FPS for standard video resolutions (e.g., 640x480)
2. **Processing Time**: Individual image processing operations, such as edge detection or blurring, exhibits significantly reduced execution times when accelerated with OpenCL
3. **Resource Utilisation**: The project will aim to achieve an efficient balance of CPU, GPU, and memory usage to maximize performance on the embedded platform

4.2 Evaluation Metrics

To assess the success of the project, the following metrics will be measured and compared:

1. **Execution Time per Frame**: The time taken to process each video frame, with and without OpenCL acceleration
2. **Frame Rate Stability**: The consistency of frame rates under varying input resolutions and video complexities
3. **Power Consumption**: An optional analysis of energy efficiency during runtime, comparing OpenCL acceleration to CPU-only processing
4. **Accuracy of Image Processing Results**: Validation of edge detection and blurring results against expected outputs to ensure functional correctness

4.3 Discussion Plan

The discussion will analyse the observed results, focusing on the following aspects:

1. **Performance Gains**: An evaluation of how OpenCL acceleration improves performance, supported by data such as frame rate graphs and execution time comparisons
2. **Challenges Encountered**: A reflection on technical and implementation challenges, such as memory bottlenecks or kernel optimization issues, and how they were resolved
3. **Scalability and Practicality**: An assessment of the scalability of the application to higher resolutions or more complex image processing tasks and its practical usability in real-world scenarios
4. **Future Work**: Insights into potential extensions of the project, such as incorporating advanced image processing techniques or deploying on alternative embedded platforms

Chapter 5

Technical Details

This section outlines the technical components of the project, including the hardware, software stack, and implementation methods. It also highlights the technical considerations made during development.

5.1 Hardware Components

Number	Component	Description	Quantity
H.1	**RaspberryPi 4**	- Acts as the primary embedded system - Quad-core Cortex-A72CPU - VideoCore VI GPU - Sufficient computational power for OpenCL tasks	**1**
H.2	**Camera Module**	- Captures video streams for real-time processing - HQ camera is the preferred camera module	**1**

Table 5.1: Table of Hardware Components [1]

5.2 Software Stack

Component	Description
OpenCL	- Used for writing parallel processing kernels to accelerate intensive image processing tasks
OpenCV	- Handles video capture, frame handling, and post processing tasks. - Provides utilities for testing and validating results
CMake and GCC	- Build and compilation framework
RaspberryPi OS	- Provides necessary drivers and libraries to support OpenCL and OpenCV on hardware
C++	- Primary programming language

Table 5.2: Table of Software Components [4]

5.3 Implementation Specifics

Number	Implementation	Description
IS.1	**Kernel Development**	- Core image processing as OpenCL kernels - Grayscale conversion - Edge detection - Gaussian blur
IS.2	**Data Flow**	- Video frames captured from the camera module will be sent to OpenCL kernels - Processed frames then sent back to the application for real-time display or storage
IS.3	**Performance Optimisation**	- Minimising host-device memory transfers - Tuning work-group sizes
IS.4	**Testing Tools**	- OpenCL profiling tools to measure execution time of kernels - OpenCV visualisation function used to ensure correctness of images

Table 5.3: Table of Implementation Specifics

5.4 Technical Considerations

Number	Factors	Description
TC.1	**Memory limitation**	Application is optimised to handle moderate resolutions
TC.2	**Real-time constraints**	Algorithm and pipeline will be tested to maintain at least 30 FPS
TC.3	**Compatibility**	Project will be designed to ensure portability across other OpenCL platforms

Table 5.4: Table of Technical Considerations

Chapter 6

Timeline

			Week											
Number	Sprint		W1	W2	W3	W4	W5		W6	W7	W8	W9	W10	W11
S.1	Introduction to OpenCL	OpenCL Fundamentals						Real-time Image Processing Project						
S.2	OpenCL Programs and Kernels													
S.3	Buffes and Sub-buffers													
S.4	Images and Samplers													
S.5	Environment Setup													
S.6	Kernel Development (Phase 1)													
S.7	Kernel Development (Phase 2)													
S.8	Application Integration													
S.9	Optimisation and Testing													

Figure 6.1: Sprint Timeline Chart

Chapter 7

Conclusion

This project aims to develop a real-time image processing application utilising OpenCL for hardware acceleration on embedded platforms. With implementing key image processing tasks such as edge detection and blurring, the project addresses the challenge of achieving high performance within the computational constraints of devices like the Raspberry Pi. Through the use of OpenCL, the application is expected to demonstrate improvements in processing speed and efficiency, validating the potential of parallel computation in embedded systems.

The project also serves as a practical application of theoretical OpenCL concepts, highlighting their adaptability to real-world problems. The anticipated outcomes include not only a functional application but also valuable insights into the trade-offs and challenges of optimizing image processing on embedded hardware.

Bibliography

[1] Raspberry Pi Foundation. *Raspberry Pi 4 Model B Specifications*. Accessed: 2024-12-05. n.d. URL: https://www.raspberrypi.org.

[2] Khronos Group. *OpenCL - The open standard for parallel programming of heterogeneous systems*. Accessed: 2024-12-05. n.d. URL: https://www.khronos.org/opencl.

[3] A. Munshi et al. *OpenCL Programming Guide*. Boston, MA: Addison-Wesley, 2011.

[4] OpenCV. *Open Source Computer Vision Library*. Accessed: 2024-12-05. n.d. URL: https://opencv.org.