LAB 6: Heterogeneous Programming in OpenCL

Universidad Zaragoza

DAVID PADILLA ORENGA, NIA: 946874
IGNACIO PASTORE BENAIM, NIA: 920576

Programming and Architecture of Computing Systems
(2024/2025)

Comparative Analysis of GPU-Only vs Heterogeneous Processing

1. Introduction

The performance of image processing tasks has become increasingly critical in modern applications, necessitating the use of parallelized computation. In this study, we analyze **two approaches** to processing a large set of simulated images using OpenCL: the first approach involves utilizing **only the GPU (non-heterogeneous)** to process all the data, while the second employs a **heterogeneous** strategy where the workload is distributed between the **CPU and GPU**. The aim is to measure the efficiency of each method, identify potential bottlenecks, and derive insights into the optimal configuration for such tasks.

The problem involves processing 5000 simulated images derived from a base image, blacklotus.jpg. Each simulated image is treated as a fragment, requiring kernel computation and subsequent **communication** between the **host** and the **devices involved**. These fragments are processed either entirely on the GPU or distributed between the CPU and GPU, with various workload distributions tested to assess performance.

2. Methodology

2.1 Hardware Configuration

The experiments were conducted on the system provided by the lab for measurements equiped with:

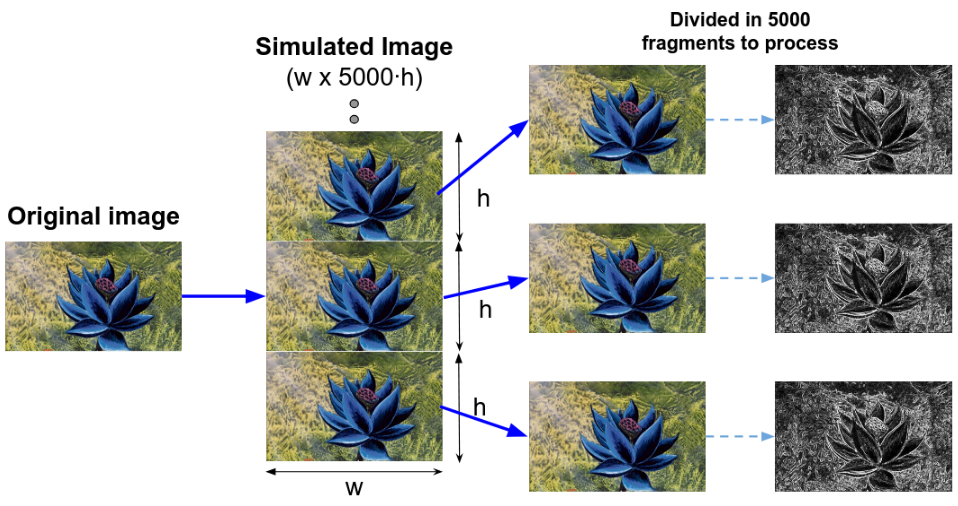
- CPU:12th Gen Intel(R) Core(TM) i7-12700.
- GPU: Intel(R) Graphics [0x4680].

These devices were utilized through OpenCL to ensure portability and direct control over parallel computation.

2.2 Image Simulation

Simulating the processing of 5000 images was necessary to evaluate the scalability and efficiency of the proposed methods under a realistic workload. Instead of using 5000 unique images, we replicated a base image ***(blacklotus.jpg)*** to create a large synthetic dataset. This approach avoids the need for extensive storage and ensures that the computational tasks are representative of real-world scenarios where similar operations are performed repeatedly on a batch of images.

The simulation involved constructing a single large image by vertically stacking 5000 replicas of the base image. This large image was then divided into 5000 equally sized fragments, each corresponding to one simulated image. The decision to divide the simulated image into complete fragments **simplifies the reconstruction process**, as each fragment is self-contained and independent. This approach reduces complexity and ensures that the **focus remains on the computational and communication aspects** of the processing pipeline.



The fragmentation also allows for straightforward distribution across devices (CPU and GPU), making it **easier to analyze the workload** balance and measure the performance of each approach. WE just stored one image of the 5000 processed since all of them would have the same result.

2.3 Implemented Approaches

Two approaches were implemented for comparison:

1. **GPU-Only Processing:** In this approach, all fragments were processed exclusively on the GPU.
2. **Heterogeneous Processing:** The workload was distributed between the CPU and GPU with different configurations:
   - **Configuration 1:** 50% CPU and 50% GPU.
   - **Configuration 2:** 25% CPU and 75% GPU.

These configurations were designed to evaluate how shifting the workload impacts performance and resource utilization.

2.4 Metrics Measured

Several metrics were collected to analyze the performance of each approach:

- **Computation Time:** Time spent executing the kernel on each device.
- **Communication Time:** Time spent transferring data between the host and devices.
- **Workload Balance:** Distribution of computational workload between devices.

$$Workload\ Balance\ (Device) = \frac{Computation\ Time\ (Device)}{Total\ Computation\ Time\ (CPU + GPU)} \times 100$$

- **Thread Time:** Time of execution of each of the threads.

- **Total Execution Time.** Containing the creation of images simulated, the separation of the images, the creation of buffer, contexts and all the OpenCL configuration.

3. Results and Observations

3.1 GPU-Only Processing

Metric	**Value**
Total Images Processed	5000
GPU Computation Time	1119.46 ms
GPU Communication Time	163.497 ms
Total Execution Time	**4404.72 ms**

The GPU-only configuration, where all 5000 images are processed exclusively by the GPU, demonstrates a **total execution time of 4404.72 ms**. This time consists of **1119.46 ms of computation time** and **163.497 ms of communication time**. The GPU achieves efficient computation, with the computation time representing approximately **25%** of the total execution time. The communication time, though smaller in magnitude, contributes to **3.7%** of the total execution demonstrating the GPU's capacity to handle a significant workload with minimal overhead.

3.2 Heterogeneous Processing

Configuration 1: 50% CPU, 50% GPU of percentage of the images

Metric	**CPU**	**GPU**	**Total**
Total Images Processed	2500	2500	5000
Computation Time	1309.13 ms	520.43 ms	**1829.56 ms**
Communication Time	43.92 ms	83.80 ms	**127.72 ms**
Thread Time	3591.45 ms	2652.91 ms	**-**
Workload Balance (%)	71.55%	28.44%	**-**
Total Execution Time	**5728.04 ms**		

In Configuration 1, the workload is evenly distributed between the CPU and GPU, each processing **2500 images**. The **total execution time is 5728.04 ms**, significantly higher than the GPU-only approach. This is due to handling two devices, which has more complexity than the GPU-only case and the significantly **low efficiency computation capacity of the**

CPU, which required **1309.13 ms**, compared to only **520.43 ms** for the GPU for the same amount of images.

The **communication time** for this configuration totals **127.72 ms**, with the GPU contributing **83.80 ms** and the CPU **43.92 ms,** being the CPU faster in communication as expected since it is directly connected to the host. While communication times are relatively small compared to computation, their combined overhead becomes non-negligible, particularly when adding the setup and buffer management times.

The workload balance shows that **71.55%** of the computation time is consumed by the CPU, while the GPU accounts for the remaining **28.44%**, indicating an imbalance that limits performance.

Configuration 2: 25% CPU, 75% GPU of percentage of the images

Metric	CPU	GPU	Total
Total Images Processed	1250	3750	5000
Computation Time	668.162 ms	778.658 ms	**1446.82 ms**
Communication Time	23.276 ms	123.164 ms	**146.44 ms**
Thread Time	1863.93 ms	3209.55 ms	**-**
Workload Balance (%)	46.69%	53.30%	**-**
Total Execution Time	**5366.4 ms**		

In Configuration 2, the workload is shifted towards the GPU, with the CPU processing **1250 images** and the GPU handling **3750 images**. The **total execution time is 5366.4 ms**, which is an improvement over Configuration 1 but still higher than the GPU-only configuration.

The **computation time** totals **1446.82 ms**, with the CPU taking **668.162 ms** and the GPU taking **778.658 ms**. The communication overhead is slightly higher than Configuration 1, totaling **146.44 ms**, split between the CPU (**23.276 ms**) and GPU (**123.164 ms**).

The workload balance improves, with the GPU contributing **53.30%** of the computation time, compared to **46.69%** by the CPU. This shift leads to better utilization of the GPU's computational power, though the CPU still imposes a significant performance bottleneck.

4. Analysis

When referring to the two main actions that were studied here we can conclude:

1. **Computation Time:** The GPU-only configuration highlights the efficiency of the GPU, achieving the lowest computation time of **1119.46 ms** for all 5000 images. In contrast, the heterogeneous configurations suffer from the CPU's slower computation speed,

especially in Configuration 1, where the CPU requires over double the GPU's computation time for an equal number of images.

2. **Communication Time:** Although communication times are smaller than computation times,we can extract some conclusions. In the GPU-only configuration, communication represents **3.7% of the total execution time**, whereas in heterogeneous configurations, the total communication overhead increases slightly due to the involvement of both devices.

Referring to the **total execution time** is the most comprehensive metric, encompassing all factors, including computation, communication, and setup overhead:

- GPU-only: **4404.72 ms**
- Configuration 1: **5728.04 ms** (+30% compared to GPU-only)
- Configuration 2: **5366.4 ms** (+21.8% compared to GPU-only)

The bottlenecks identified include:

1. **CPU Performance:** The CPU's slower computation speed significantly impacts heterogeneous configurations, especially Configuration 1, where the CPU handles 50% of the workload.
2. **Communication Overhead:** While smaller in magnitude, communication times become more pronounced in heterogeneous setups due to additional data transfers between host and devices.

4. Conclusions

This comparison demonstrates that even the best heterogeneous configuration (Configuration 2) incurs a **21.8% overhead** compared to GPU-only processing. This result highlights the challenges of coordinating between devices, particularly in managing buffers and communication. In the heterogeneous setup, separate buffers are created for each device, requiring additional time for data transfers and synchronization. By contrast, GPU-only processing benefits from streamlined buffer management within a single device context, minimizing these overheads and simplifying execution.

However, non-heterogeneous setups are not always inferior. Optimizations such as buffer reuse, leveraging multiple GPUs, or utilizing unified memory architectures can dramatically improve performance. For example, systems with multiple GPUs on the same platform can share resources efficiently, reducing communication delays. Unified memory can further eliminate the need for explicit data transfers, cutting down on latency. While this experiment shows GPU-only processing as the most efficient for the given workload, it underscores that non-heterogeneous approaches, when optimized, have significant potential to outperform, particularly in scalable or specialized scenarios.