GPU Biometric Matching

Simon Barras

School of Computer and Communication Sciences

Semester Project

February 26, 2026

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Abstract

Abastract....

Contents

1 Introduction

2 Analysis

This project is about optimizing the Miura's pipepline [4] with a GPU.

2.1 The Miura's Pipeline

The Miura's pipeline from the publication[4] is a method that takes as input a single image of a finger, extracts the veins and then computes the distance between the extracted veins and a reference vein pattern. It shows good performance and accuracy and show a possibility of using this biometric attribute for authentication purposes.

2.1.1 Previous Work

The Miura's pipeline is a method that was proposed in 2004 and since then, several works have been done inside the EPFL lab's Lasec to use it as auhtentication for patient identification in hospitals.

Optimized Pipeline

In 2024, Lara Sofie Lenz, created an optimized implementation of Miura's pipeline [3]. This C++ optimized implementation runs arround 10 times faster than the original implementation in Python for a **1 to 1** matching. The figure 1 shows the flow of the optimized pipeline.

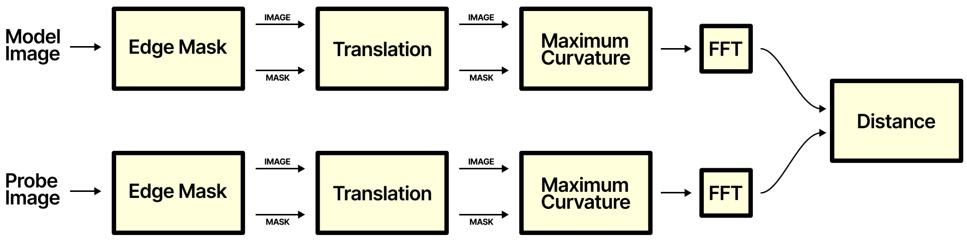

Figure 1: The flow of the optimized pipeline from Lara Sofie Lenz' report [3]

Mathematical Background

We see that the Ooptimized pipeline (figure 1) uses a Fast Fourier Transform (FFT) to compute the cross-correlation between the extracted veins and the reference vein pattern. The mathematical explanation of using a FFT to compute the cross-correlation between two signals is explained in detail in the semester project of Vincent Luke Ventura about Biometric Homomorphic Matching [6].

2.1.2 1 to N Matching

All the previous works in the Lasec lab about Biometric identification explained in the previous section 2.1.1 show good performance (less than 1 second) for a **1 to 1** matching. The 1:1 is obviously quicker than a 1:N matching, but it requires the user to select his id before the authentication. The 1:N matching will be used at sign up phase, to ensure that the user is not already in the database.

The idea is to compute the cross-correlation between the extracted veins and all the reference vein patterns in the database at the same time. The strategy is a shifting of architectural design from a CPU which is the classical processor unit to a GPU which is a specialized hardware that is designed to perform parallel computations efficiently.

The scope of this project is to uses the optimized pipeline from Lara Sofie Lenz [3] and don't change the part of the pipeline related to the veins extraction. The matching part of the pipeline is modified to compute the cross-correlation between the extracted veins and all the reference vein patterns in the database at the same time using a GPU.

2.2 Graphics Processing Units

Graphics Processing Unit (GPU) are specialized hardware that are designed to perform parallel computations efficiently. Most of the time, they are used as a co-processor to a CPU to accelerate the performance of certain applications.

2.2.1 GPU Architecture

The classical architecture and the one of the main processor in a computer is the Central Procesing Unit (CPU). This processor is designed to perform a wide range of tasks and is optimized for single-threaded performance. We generally says that it is a Single Instruction Single Data (SISD) even if now some CPU have multiple cores and can perform some parallel computations.

The GPU is a specialized hardware that is designed to perform parallel computations efficiently. We generally says that it is a Single Instruction Multiple Data (SIMD) because it can perform, physically, the same instruction on multiple data at the same time.

In general, a GPU is composed of several streaming multiprocessors (SM) that are composed of several cores. Each core can execute a thread and the threads are grouped in blocks that are executed on the SMs. The particularity between the core of a CPU and the core of a GPU (SM) is that all the threads of a block execute **the same instruction** at the same time.

Multiple SMs together form a Graphics Processor Cluster (GPC) and multiple GPCs together form a GPU. As shown in the figure 2, the GPU has a different memory hierarchy than the CPU and it is optimized for high throughput rather than low latency. The L2 cache is shared between all the SMs and the global memory is accessible by all the threads but has a high latency compared to the shared memory that is only accessible by the threads of a block and has a low latency.

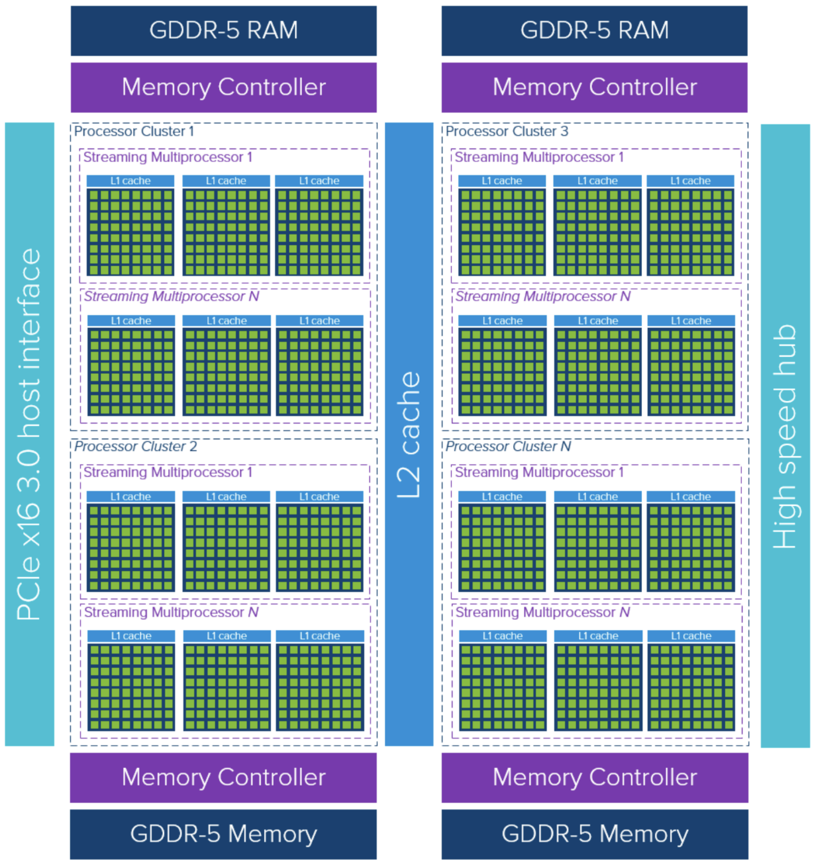

Figure 2: A GPU architecture schema from the PDF document [2]

Every constructors of GPU have their own architecture and their own way to organize the threads and the memory hierarchy but the general idea is the same and the main difference is the number of cores and the memory bandwidth. Other architecture, like Apple Silicon

2.2.2 GPU Constructors And Programming Models

In 2026, the two main architectures designers of GPU are Nvidia and AMD. There are also emerging architectures like Intel with their Intel Arc series and Apple which include a GPU in their Apple Silicon chips.

To use a device like this, we need to use a programming model that allows us to offload the computations and the data. There is CUDA which is a proprietary programming model developed by Nvidia and that only works on Nvidia GPU. AMD has developed HIP which is a programming model that allows to write code that can be executed on both Nvidia and AMD. There is also OpenCL which is an open standard that allows to write code that can be executed on a wide range of devices, including GPU from different manufacturers.

The choice of the programming model is important because it will determine the performance and the portability of the code. SYCL [5] is an abastraction layer that allows to write code that can be executed with different GPU constructors. This is very useful because it allows to write code that isn't tied to a specific architecture and that can be executed on a wide range of devices. Which is great as the architecture of the end device is not know yet and it facilitates the development phases for computer wihtout a specific GPU constructor. Also, the portability of the code don't affect the performance according to the study "Comparing Performance and Portability Between CUDA and SYCL for Protein Database Search on NVIDIA, AMD, and Intel GPUs" (reference [1]).

2.2.3 GPU Programmation

Now, as the GPU is a co-processor, it is used to accelerate the performance of certain tasks. The code executed on a computer is by default executed on the CPU and when we want to use the GPU, we need to explicitly offload the computations and the data to the GPU using a programming model. The figure 3 shows the memory organization of a GPU.

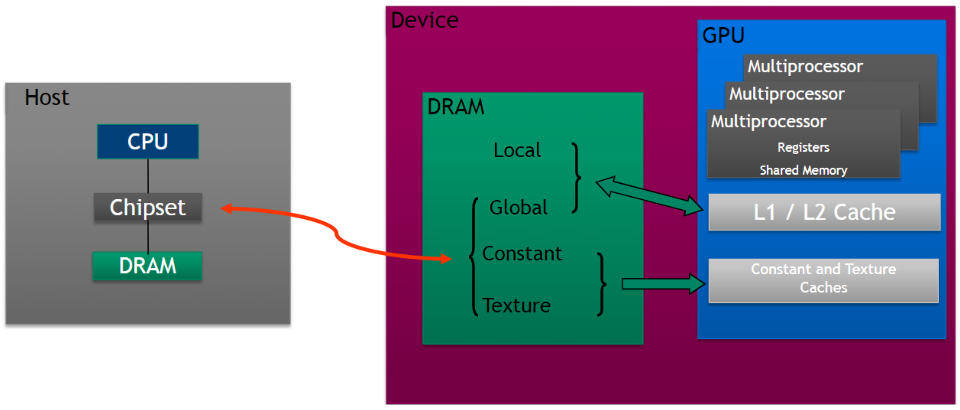

Figure 3: Memory hierarchy in a PC with a GPU

The general steps to use a GPU, regardless of the programming model, are the following:

Step 1: Allocate the parameters and the return values on the host memory.

Step 2: Allocate the parameters and the return values on the device memory.

Step 3: Copy the parameters from the host memory to the device memory.

Step 4: Launch the kernel on the device.

Step 5: Copy the return values from the device memory to the host memory.

Step 6: Free the device memory.

3 Conception

4 Realisation

5 Evaluation

6 Discussion & Future Work

7 Conclusion

8 Example:

Example of latex section

Previous Work

...

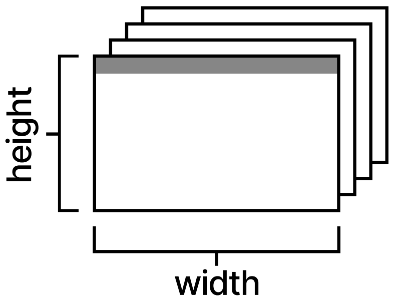

Figure 4: A visualization of how the four 240 × 376 matrices are aligned in the optimized implementation. The gray area denotes the values that we intend to access.

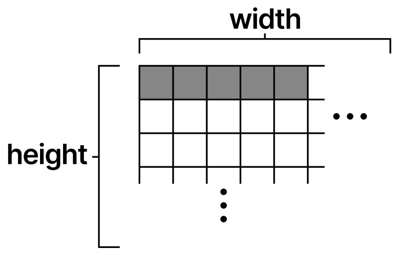

Figure 5: A visualization of the memory access pattern produced by Fig. 4. The gray boxes indicate the memory regions that need to be accessed when wanting to read the first row of the first matrix.

		V1 (in s)	*V2* (in s)	*V3* (in s)
Complete Pipeline	μ	1.969198	0.387614	0.170519
	σ	0.424002	0.017806	0.009598
Edge Mask	μ	0.072014	0.017158	0.003849
	σ	0.019911	0.002081	0.000473
Prealignment	μ	0.006043	0.004284	0.004659
	σ	0.002194	0.001518	0.00058
Maximum Curvature	μ	1.715232	0.363249	0.153078
	σ	0.384366	0.015857	0.006914
Postalignment	μ	0.022223	0.007586	
	σ	0.003242	0.000427	
Plain Distance	μ	0.001574	0.000298	
	σ	0.000329	$6.0327 \cdot 10^{-5}$	
Corrected Distance	μ	0.023797	0.007884	0.001254
	σ	0.003467	0.000433	0.000103

Table 1: Time measurement results (measured in seconds) for the entire pipeline and each pipeline step while Turbo Boost was enabled.

References

[1] Manuel Costanzo, Enzo Rucci, Carlos García-Sánchez, Marcelo Naiouf, and Manuel Prieto-Matías. Comparing performance and portability between cuda and sycl for protein database search on nvidia, amd, and intel gpus. In *2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 141–148, 2023. doi:10.1109/SBAC-PAD59825.2023.00023.

[2] Niels Hagoort. Exploring the gpu architecture. Accessed: 2024-02-26. URL: https://www.vmware.com/docs/exploring-the-gpu-architecture.

[3] Lara Sofie Lenz. Finger vein biometric matching. Semester project thesis, EPFL, June 2024.

[4] Naoto Miura, Akio Nagasaka, and Takafumi Miyatake. Feature extraction of finger vein patterns based on iterative line tracking and its application to personal identification. *Syst. Comput. Japan*, 35(7):61–71, jun 2004.

[5] The Khronos SYCL Working Group. SYCL 2020 Specification (revision 10). https://www.khronos.org/registry/SYCL/specs/sycl-2020/html/sycl-2020.html, 2024. Accessed: 2026-02-25.

[6] Vincent Luke Ventura. Biometric homomorphic matching. Semester project thesis, EPFL, January 2026.

Acronyms

CPU Central Procesing Unit.

CUDA Computaional Unified Device Architecture.

FFT Fast Fourier Transform.

GPU Graphics Processing Unit.

HIP Heterogeneous-computing Interface for Portability.

OpenCL Open Computing Language.

SIMD Single Instruction Multiple Data.

SISD Single Instruction Single Data.

SYCL System-wide Compute Language.

A The Maximum Curvature Algorithm

...

Step 1: ...

  Step 1-1: ...
  Step 1-2: ...

Step 2: ...