This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3005476, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Asynchronous Processing for Latent Fingerprint Identification on Heterogeneous CPU-GPU Systems

ANDRES J. SANCHEZ[1], LUIS F. ROMERO[1], DANIEL PERALTA[2,3], MIGUEL ANGEL MEDINA-PÉREZ[4], SIHAM TABIK[5], YVAN SAEYS[2,3], AND FRANCISCO HERRERA[5]

[1]Department of Computer Architecture, University of Malaga, 29071 Malaga, Spain
[2]Applied Mathematics, Computer Science and Statistics, Ghent University, 9000 Ghent, Belgium
[3]Data Mining and Modelling for Biomedicine Group, VIB Center for Inflammation Research, 9000 Ghent, Belgium
[4]Tecnologico de Monterrey, Carr. al Lago de Guadalupe Km. 3.5, Atizapán, 52926 Estado de México, México
[5]Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071 Granada, Spain

Corresponding author: Andres J. Sanchez (ajsanchez@ac.uma.es).

This work was supported in part by the Spanish Ministry of Science and Technology through the TIN2016-80920-R project, the National Council of Science and Technology of Mexico (CONACYT) under the grant PN-720, and the University of Malaga (International Campus of Excellence (ICE) Andalucia TECH) through the U-Smart-Drive project from the I Plan Propio Smart-Campus. Daniel Peralta is a Postdoctoral Fellow of the Research Foundation of Flanders (FWO, 170303/12X1619N). Yvan Saeys is an ISAC Marylou Ingram Scholar.

ABSTRACT Latent fingerprint identification is one of the most essential identification procedures in criminal investigations. Addressing this task is challenging as (i) it requires analyzing massive databases in reasonable periods and (ii) it is commonly solved by combining different methods with very complex data-dependencies, which make fully exploiting heterogeneous CPU-GPU systems very complex. Most efforts in this context focus on improving the accuracy of the approaches and neglect reducing the processing time. Indeed, the most accurate approach was designed for one single thread. This work introduces ALFI (Asynchronous processing for Latent Fingerprint Identification), the fastest methodology for latent fingerprint identification maintaining high accuracy. ALFI fully exploits all the resources of CPU-GPU systems using asynchronous processing and fine-coarse parallelism for analyzing massive databases. Our approach reduces idle times in processing and fully exploits the inherent parallelism of comparing latent fingerprints to fingerprint impressions. We analyzed the performance of ALFI on Linux and Windows operating systems using the well-known NIST/FVC databases. Experimental results reveal that ALFI is in average 22x faster than the state-of-the-art algorithm, reaching a value of 44.7x for the best-studied case.

INDEX TERMS Asynchronous processing, accelerator architectures, CUDA, fine-grained parallelism, fingerprint recognition, heterogeneous computing, latent fingerprint identification, parallel processing.

I. INTRODUCTION

THE identification of suspects based on fingerprints acquired from crime scenes is an essential procedure for forensics and law enforcement agencies all around the world [1]–[3]. These biometric features are thoroughly used in daily identification systems because of their uniqueness and easiness of use. The problem of fingerprint recognition can be addressed by using two different approaches [4]: verification and identification. The first approach only performs one comparison to check if the particular fingerprint matches with another stored previously; this is a 1:1 comparison. The second approach is related to the problem of identifying a person among those whose data are included in a specific database; this is a 1:N comparison where N is the number of samples of the database. It also coincides with the number of comparisons to be performed in a procedure commonly known as the matching process. This is the most challenging one in terms of computational cost and complexity [5], [6].

The type of fingerprint is another aspect to consider when developing a fingerprint matching algorithm. As shown in Figure 1, fingerprints can be classified into three different classes depending on the conditions under which they are acquired [4]: rolled, plain, and latent. Rolled fingerprints are obtained by rolling the finger from one side to the other, hence getting more information, but also introducing deformations in the resulting image. Plain fingerprints are

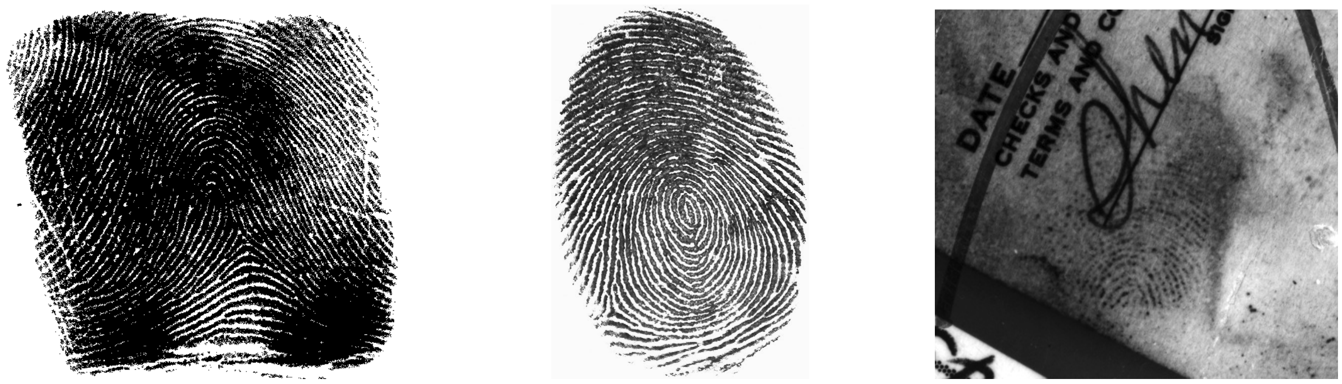

(a) Rolled (b) Plain (c) Latent

FIGURE 1: Types of fingerprints depending on the acquisition process.

produced just by pressing the finger onto a surface. Both types of fingerprints are characterized by having a good image quality due to a voluntary acquisition process performed under controlled conditions. On the contrary, latent fingerprints are those unintentionally left on a surface by deposits of sweat and/or oil from the fingertip. This type of fingerprint is usually not visible to the naked eye and requires additional processing in order to be detected. Most common acquisition techniques include dusting with fine powder and the use of chemicals. Fingerprints obtained by any of these procedures may result in incomplete and inaccurate information per fingerprint, which introduces errors to the matching process [7]. However, their utility in criminal investigations and the inherent challenge of processing lower quality and deformed images [8] are just a few of the compelling reasons to process them.

The difficulty in processing latent fingerprints remains very high nowadays. The current trend seems to be in the direction of developing specific algorithms for latent fingerprint matching so that they are suited to their particular processing needs [7], [9]. Since there is very little information available per latent fingerprint, the focus is on finding and assessing relationships among the fingerprint descriptors. This fact creates data dependencies between different stages of processing and complex methodologies are required to manage them, making the use of parallel techniques difficult. Another main disadvantage related to these algorithms lies in their inability to handle massive databases, in the order of millions of fingerprints, in the time required by law enforcement authorities. The latent fingerprint identification algorithm that provided the best trade-off between computational cost and precision is based on the Deformable Minutiae Clustering (DMC) method using Minutia Cylinder-Codes (CC) [10]. However, this algorithm was designed for one single thread.

This paper introduces a new methodology, called ALFI (Asynchronous processing for Latent Fingerprint Identification), for latent fingerprint identification specifically designed to fully exploit all the resources of heterogeneous CPU-GPU systems. ALFI is able to analyze large databases faster than the state-of-the-art algorithm [10] while providing very similar precision results. This can be of great help to local authorities as processing times get even closer to real-time systems using the processing units available on almost any computer today.

The main contributions of this work are:

- We design a new methodology named ALFI (Asynchronous processing for Latent Fingerprint Identification) for a faster and accurate latent fingerprint identification.
- We propose a fine-grained parallelism at fingerprint descriptor level as a basis for achieving an effective CPU-GPU processing pipeline.

ALFI performance is tested on Linux and Windows operating systems (OSs) using three CPU-GPU pair systems. Well-known identification databases such as NIST SD27, SD14, and SD4 are used to test the accuracy of the proposed algorithm. Additionally, FVC 2002, 2004, and 2006 verification databases are also used. Computational performance results prove that ALFI is in average 22x faster than the state-of-the-art algorithm maintaining similar accuracy. In particular, for the best-studied case, it yields a speed-up of 44.7x.

This paper is organized as follows: Section II presents the state-of-the-art regarding fingerprint identification. Section III explains current GPU architecture with particular focus on the CUDA parallel computing platform and programming model. Section IV describes the ALFI methodology for CPU-GPU heterogeneous systems. Section V evaluates the proposed algorithm in terms of accuracy and computational performance compared to the state-of-the-art algorithms. Section VI presents the conclusions. The Appendix section contains the essential functions used throughout this paper.

II. RELATED WORK: FINGERPRINT IDENTIFICATION

Relevant research in the field of fingerprint recognition can be divided into five categories: (i) fingerprint representation, (ii) fingerprint data enhancement, (iii) fingerprint data preprocessing, (iv) accelerating fingerprint matching, and (v) latent fingerprint identification.

There exists a large body of works in fingerprint representation. Early works analyze fingerprints considering core and delta parameters or ridge flow methods [11], [12], whereas current approaches consider minutiae [13]. Minutiae are local structures related to specific points in the discontinuities of the fingerprint ridges, such as endings and bifurcations. These structures are the basis of the well-known Minutia Cylinder-Codes descriptors (MCC) [14], widely used in the recent literature because of its high accuracy at a relatively low computational cost [13], [15].

Most works in fingerprint data enhancement focus on designing new preprocessing techniques to improve the data acquired from a fingerprint or verify its authenticity. For instance, the orientation of the sample can produce bad accuracy results, so most relevant approaches focus on finding a correct orientation field model [16]. This parameter can be built even in the presence of noise and distortion [17] or using a trained Convolutional Neural Network (CNN) [18]. On the other hand, security and fault tolerance in current identification systems are very important issues in our society. Therefore, finding solutions to prevent attacks in the fingerprint identification procedure is essential. This problem is usually addressed by analyzing whether a particular fingerprint sample stems from a live subject or an artificial replica [19]. Although this problem remains difficult in terms of robustness, effectiveness, and efficiency, several studies are still proposing hardware and software-based approaches [20], [21].

Real-world fingerprint databases contain in the order of millions of fingerprints. Several studies reduce the computational cost by using approaches such as classification, indexing, hardware improvement, and/or parallel computing. The most studied one is classification, which filters large-scale databases by separating fingerprints into different categories based on their shapes. Only those belonging to the same class as the input sample will be processed in the following steps. This method increases the speed of processing and allows to handle massive databases [22], [23]. Nevertheless, latent fingerprints usually correspond to partially or poorly acquired data making these preprocessing tasks almost impossible.

With the emergence of new hardware, the use of Graphics Processing Units (GPUs) in biometric recognition algorithms has increased in recent years. Several studies focus on this particular approach for databases with good quality fingerprints. For instance, the authors in [24] proposed an optimized GPU fingerprint matching system based on MCC, which accelerates the comparison method up to 100.8x over the sequential CPU implementation. The proposal presented in [25] yields a speed-up of 1946x and 207x, considering the ratio between the thousand match per second (KMPS) values and compared to the non-optimized baseline and the one optimized with SIMD sequential CPU implementations, respectively. The work described in [26] accelerates a well-known fingerprint matching algorithm [27], achieving superior performance results in contrast to multi-threaded CPU implementations [6]. The proposal in [28] speeds up the comparison method and implements a novel strategy in the consolidation stage that is shown to enhance accuracy. All mentioned works that are specifically developed to be executed on GPUs share a common object: to speed-up the evaluation of massive databases by increasing the number of fingerprints processed per second (throughput). However, these implementations need to be developed considering the underlying architecture and must be relatively simple to run effectively on GPUs [29], thereby reducing accuracy in most cases. Besides, GPU-based algorithms do not exploit the power of the CPU in processing, which would lead to better run-time results.

Many studies analyze the performance of general identification algorithms in processing latent fingerprint databases. The achieved results revealed a poor performance owing to the low quality of the input data [30] and thus, opening the way to the development of new algorithms specifically designed to this aim. In latent fingerprint identification, early works proposed several solutions for handling typical deformations which affect the matching procedure. For instance, regarding the minutiae matching process, several approaches were considered: the use of a minutia-based descriptor [31] or a combination of this structure and an orientation field descriptor of the fingerprint [32]. In practice, a global matching operation is performed by selecting the five best minutiae pairs to find new sets. For each found cluster, a matching score is computed, and after that, the maximum value is chosen as the similarity score between latent and rolled fingerprints. On the other hand, the proposal presented in [33] uses a different approach that combines local minutiae descriptors and fingerprint alignment through the Hough Transform to improve fingerprint matching performance. One main characteristic of latent fingerprints lies in the presence of noise after the feature extraction. For this reason, researchers in [34] developed a method to improve the latent matching accuracy by incorporating feedback from exemplars (rolled or plain fingerprints) to refine the feature extraction. The most accurate latent fingerprint identification algorithm among those which are based solely on minutiae structures finds deformable clusters of matching minutiae pairs in local regions by performing multiple alignments [10]. Overlapped clusters are merged to find consolidated matching minutiae pairs that are thereafter used to build a Thin-Plate Spline (TPS) model [35]. New minutiae pairs, which might not have been found due to deformations in previous steps, can be obtained through this methodology.

This work addresses a new solution to the latent fingerprint identification problem in order to fully exploit the capabilities of heterogeneous CPU-GPU systems. Our approach brings the identification procedure even closer to a real-time task.

We evaluate and compare in terms of accuracy and speed-up our methodology to the state-of-the-art algorithms [10]. The latter was specifically designed for CPU processing and neglects the potential of GPUs. To the best of our knowledge, there are no related algorithms in the literature developed for latent fingerprint identification for heterogeneous systems.

III. GENERAL-PURPOSE COMPUTING ON GPUS

In the last decade, the role of Graphics Processing Units (GPUs) has evolved from managing tasks only related to visual processing (e.g., rendering 3D graphics in video games and visual applications) to general data processing, commonly known as General-Purpose Computing on Graphics Processing Units (GPGPU). The areas in which GPU processing is widely used are usually related to emerging scientific and technological fields such as molecular analysis, weather prediction, and biometric recognition [24]–[26]. These scientific fields have in common the need to manage and process a massive amount of data, a task which can be remarkably accelerated by using graphical processing units. The first developers using GPUs for general-purpose computing needed to represent their mathematical problems by using vertices and pixels so that they could be executed on these devices. Nevertheless, it was not until the year 2006 when NVIDIA [36] launched a hardware and software architecture to use NVIDIA GPUs for general-purpose computing, allowing researchers and developers to take advantage of the parallel nature of GPUs with less effort and more efficiently than before. This framework called Compute Unified Device Architecture (CUDA) provides a high-level abstraction for C/C++ programming and enables applications running on the CPU (host) to perform data processing on the GPU (device). A model of this framework is given in Figure 2 for a better understanding of the following sections.

A. HARDWARE AND SOFTWARE ARCHITECTURE

The hardware side of the NVIDIA CUDA framework [36] is formed by a set of Stream Multiprocessors (SMs), whose number depends on the GPU architecture. Each SM is composed of usually 32 cores, which can run many threads in parallel responsible for executing the functions, commonly known as kernels, specifically designed for the device. Likewise, threads are grouped into processing structures called warps (typically containing 32 threads each). Every thread from a particular warp should be performing Single Instruction Multiple Data (SIMD) operations inside the kernels to achieve maximum performance. The cause of this fact lies in avoiding the thread divergence problem, which occurs when threads from the same warp take different paths after processing a branch instruction such as if-else and switch statements. Threads are also grouped at a higher level into thread blocks, which run on the same SM sharing its resources. Finally, thread blocks are gathered together inside a grid and must be able to be executed independently, since communication is not possible between blocks unless the global memory is used.

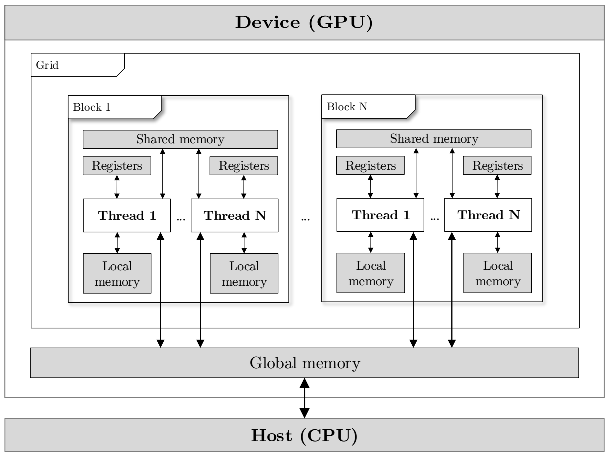


FIGURE 2: CUDA platform model formed by grids, blocks, threads, and the different sorts of memories in the device (GPU) and host (CPU) processing units.

B. MEMORY HIERARCHY

Regarding the device memory hierarchy, the smallest and fastest memory units are registers, followed by local memory, which is much slower. Both types of memory are private for every thread, and the data stored cannot be shared between them. The next level of memory is the shared memory space whose data is accessible for all threads within the same block, provided that the block is being executed. Finally, the largest and slowest storage space is the global memory, which can be accessed by all thread blocks and therefore allowing sharing data between threads, even those that belong to different blocks. This last memory unit is also used for communication with the host unit. Data allocations in the host memory are pageable by default, and the device cannot access this data directly. Therefore, when a data transfer from pageable host memory to device memory is invoked, the CUDA driver comes into play. This must first allocate a temporary page-locked (generally known as pinned memory), copy the host data to this pinned memory and, finally, transfer the data from the pinned memory to device memory. To avoid this expensive process, data in the host can be directly allocated in pinned memory, improving transfer speeds by preventing the memory from being swapped out.

C. CONCURRENT MODEL

In CUDA programming, concurrent execution is possible by using structures called streams which are a series of queued commands that are executed sequentially. Developers can create and utilize non-default streams, performing multiple operations such as the execution of multiple kernels and memory transfers concurrently in different streams. For this reason, using multiple streams can add an additional layer of parallelization to particular applications. This also offers many more opportunities for optimization, e.g., overlapping data transfers with (i) computation on the host, (ii) computation on the device, and (iii) other data transfers between the host and device. Synchronization between the different operations is necessary, and events can be used to perform this particular task. They can block the device or the host execution until some operations inside a particular stream are completed.

IV. ALFI METHODOLOGY FOR LATENT FINGERPRINT IDENTIFICATION

This section describes the new methodology specifically designed to address the latent fingerprint identification problem. Our proposal is called ALFI (Asynchronous processing for Latent Fingerprint Identification) and fully exploits the intrinsic parallelism of the latent fingerprint identification procedure, which has not been addressed in recent literature. This methodology is developed considering the technical features of CPU (host) and GPU (device) to take the maximum advantage of these high-performance devices.

First, an analysis in detail of the state-of-the-art latent identification algorithm is carried out in Section IV-A. Afterwards, we describe the fundamentals of the ALFI methodology regarding asynchronous processing (Section IV-B) and fine-grained parallelism (Section IV-C). The necessary data structures are described in Section IV-D. In Section IV-E, the different pseudo-codes are presented related to (i) the host function in control of the device in Section IV-E1, (ii) the different kernels running on the device in Sections IV-E2-IV-E5, and (iii) the host function in charge of the final evaluation stage in Section IV-E6.

A. REVIEW OF THE STATE-OF-THE-ART IDENTIFICATION ALGORITHM

The Deformable Minutiae Clustering algorithm using Minutia Cylinder-Codes representation (DMC-CC) [10] was developed by merging four well-known independent methods and a final similarity computation stage to obtain the similarity values between latent and fingerprint impressions. First, this algorithm uses the *Minutia Cylinder-Codes* descriptors as input data of the local matching processing to find the first group of minutiae pairs. In addition, these descriptors are based on 3D data structures built from minutiae positions and angles, after merging local structures [27]. The *Minutiae Discrimination* method [37] calculates the quality value of each minutia in the latent fingerprint and the fingerprint impression based on the direction consistency around it. The *Deformable Minutiae Clustering* method [10], [38] finds clusters of minutiae pairs, along with a weight value for each one, from the initial set of matching minutiae pairs. After merging the clusters, a final set of minutiae pairs is used for calculating an initial similarity score between fingerprints. Then, the *Thin Plate Spline* method (TPS) [35] is applied to avoid data loss due to fingerprint deformation and find new matching minutiae pairs. These pairs could have been discarded in previous steps owing to the deformation effects and may improve the previously calculated similarity value. The last step is called *Similarity Computation*, where the different statistical outcomes are obtained depending on the type of experiment. Given the above, the DMC-CC algorithm can be described as follows:

1) *Minutia Cylinder-Codes*. Let L and T be the minutiae sets of the latent fingerprint and a particular fingerprint impression from a database, respectively. Each minutia $q \in L$ is compared to all minutiae $p \in T$ based on their minutiae descriptors. Similar minutiae are selected as matched minutiae pairs (q, p) and included inside a new set A, which is after that, sorted in descending order according to their similarity values. Then, a new array M is filled with no more than $max\{|L|, |T|\}$ local matching minutiae pairs from A so that the repetition of minutiae within different pairs is reduced.
2) *Minutiae Discrimination*. Quality value is computed for every minutia $q \in L$ and $p \in T$, relying on the minutiae direction consistency of all minutiae inside its respective neighbourhood. After that, two sets containing all minutiae quality values from both fingerprints are obtained.
3) *Deformable Minutiae Clustering*.
    a) Every minutiae pair $(q, p) \in M$ is used to align fingerprints and find a cluster of matching minutiae pairs. Let $\mathcal{C}_s$ be the set of found clusters of matching minutiae pairs. Every $(q_h, p_h) \in M, h = 1...|M|$ is used in this step to work as the centroid of its cluster, denoted as B_h. For each $(q_g, p_g) \in M, g = 1...|M|$ compute if q_g matches with p_g when aligning using current (q_h, p_h) and, if this condition is fulfilled, update $B_h = B_h \cup (q_g, p_g)$.
    b) Sort B_h in descending order according to their new similarity values obtained in the previous step. Let C_h be defined as the cluster which will contain a reduced number of minutiae pairs from sorted B_h to decrease the repetition of minutiae within different pairs. A weight $w_{p_h}^{q_h}$ for every minutiae pair is computed depending on the number of minutiae pairs inside its respective cluster C_h and the number of minutiae in the latent and fingerprint impressions. Every admissible cluster is then added to the actual set $\mathcal{C}_s = \mathcal{C}_s \cup \{C_h\}$ which will be used in the following steps.
    c) The final weight w_{C_h} for each cluster $C_h \in \mathcal{C}_s, h = 1...|\mathcal{C}_s|$ is obtained by accumulating every weight $w_{p_k}^{q_k}$ of the minutiae pairs $(q_k, p_k) \in C_h, k = 1...|C_h|$. Then, $\mathcal{C}_s$ is sorted in descending order based on their cluster weights and, thereafter, all clusters are merged according to several design parameters to find a preliminary set of global matching minutiae pairs (M').
4) *Thin Plate Spline*. From the previous set of minutiae pairs, a TPS model is built in order to correct any deformations the fingerprint image may have. By using this method, new minutiae pairs are found and included

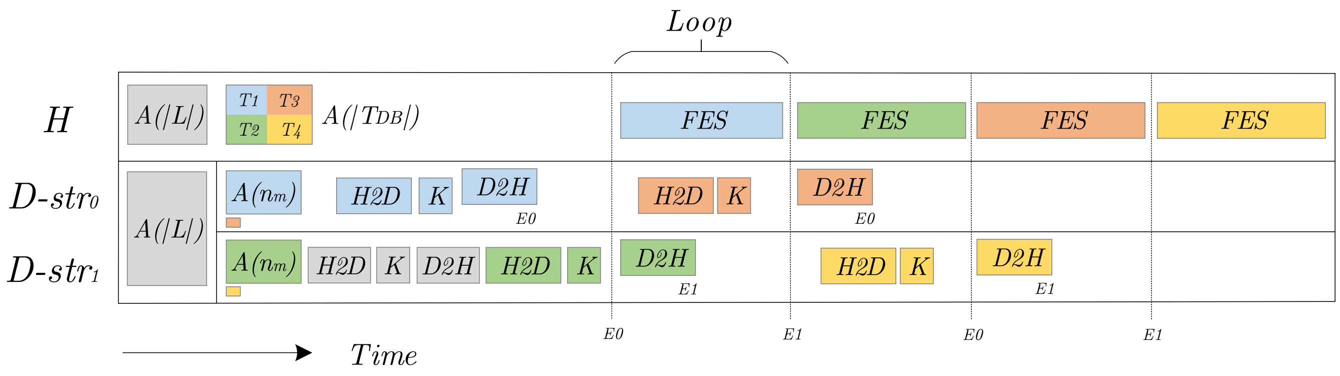


FIGURE 3: Particular case of the proposed methodology considering one latent fingerprint L and four batches T of fingerprints resulting from splitting the database T_{DB}. These batches are allocated (A) in both device D and host H memories. Data transfers ($H2D$, $D2H$) and device computation carried out in kernels K are overlapped with the multi-threaded final evaluation stage FES performed on the host. CUDA streams str_0 and str_1 and their corresponding synchronization events E_0 and E_1 are used to coordinate the requested operations.

in a set called M^*. The weights of these minutiae pairs found with this method are calculated in a similar way as the one presented in Step 3a-3c.

5) *Similarity Computation*. The matching score between the latent fingerprint and the fingerprint impression is obtained by accumulating the weights of every minutiae pair inside both M' and M^* sets.

The above-mentioned steps of the DMC-CC algorithm present several unavoidable and complex dependencies which force to execute them sequentially. This fact causes a significant loss of performance when computing on multi-core and heterogeneous systems.

B. ASYNCHRONOUS DATA PROCESSING

ALFI methodology is inspired by the state-of-the-art DMC-CC algorithm but based on a complete redesign to achieve faster processing and correct performance on heterogeneous systems. We decided to change and develop new methods for Step 1-3a to be executed through different kernels K because they are suitable to be processed on the device. Step 3b-5 are modified to take the device outcomes as input and process them on the host to balance the computational load between both processing units. The computation of these last steps performed on the host will be referred to as the multi-threaded final evaluation stage FES from now on. In addition, the host unit also coordinates the launch of all further operations to be performed on the device.

Let L and T_{DB} be the latent fingerprint used as a case study and the large-scale fingerprint database of impressions, respectively. Since all the information cannot be entirely stored in the device memory at one stroke, the large-scale database must be divided into several batches. The comparison between the latent fingerprint L and every impression fingerprint from a particular batch $T \in T_{DB}$ can be processed on the host after several steps are completed on the device. These steps include host to device data transfer $H2D$, processing kernels K, and device to host data transfer $D2H$ operations. ALFI methodology efficiently overlaps and synchronizes these operations and the operations performed on the host through the use of synchronization events in a effective CPU-GPU processing pipeline avoiding idle times.

The behaviour of ALFI is shown in Figure 3 for the particular case of the T_{DB} divided into four batches of fingerprints $T_i \in T_{DB}, i = 1...4$, for the sake of simplicity. First, the allocation of the latent fingerprint $A(|L|)$ and the entire database $A(|T_{DB}|)$ are performed in the host H, particularly in pinned memory. The allocation of the entire database is possible in the host but not in the device since typically the memory space available in the host is far larger than the available space in the device memory. Besides, pinned memory is used in the host memory since this method prevents these memory spaces from being swapped out, improving the speed of memory transfers between host and device units. Regarding the memory management in the device D, the allocation of the latent fingerprint $A(|L|)$ is carried out at start-up. The rest of the available memory space is divided into two large spaces. Both areas, denoted as $A(2 \cdot n_m)$, will be filled in with two different batches of fingerprints so that memory transfers and computation can be overlapped. Likewise, each area and the batch included within it is managed by one of the two non-default streams str_0 and str_1 from the device. In particular, these two memory areas will be filled in with T_1 and T_2 batches at start-up and managed by str_0 and str_1, respectively. In the following iterations, T_3 will be stored in the first memory space for the stream str_0 while the processing of T_2 is taking place in the stream str_1 and after the $D2H$ operation containing the results of processing T_1 is finished (E_0 event). Similarly, T_4 will be stored in the second memory space for str_1 while the processing of T_3 is taking place in str_0 and after the $D2H$ operation containing T_2 results is finished (E_1 event). This way the data is always stored in the device before starting the processing and thus reducing idle times.

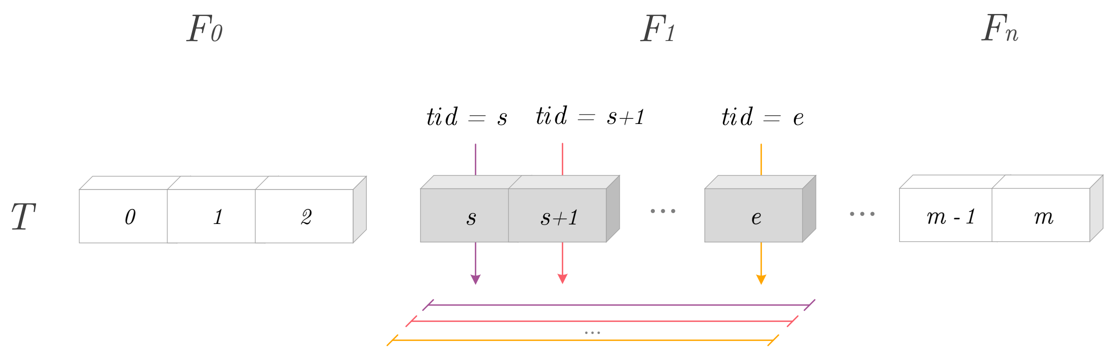


FIGURE 4: The proposed processing of fingerprint impressions by one CUDA stream on the device. Each thread tid performs all the required operations in a kernel K over its corresponding minutia from a fingerprint F_i inside the T batch of fingerprints. Parameter descriptions are shown in Table 1.

C. FINE-GRAINED PARALLELISM IN PROCESSING

Once the data is correctly allocated in the device memory, four different kernels K are launched to process batches of fingerprints on the device. Local minutiae matching is performed in $K_{1,2}$ with a fixed number of found matching minutiae pairs. The quality score for every minutia inside the latent fingerprint and the batch of fingerprints is related to two executions of K_3, with slight variations for the latent fingerprint. Finally, the alignment of the minutiae pairs to find clusters of these structures is carried out in K_4 for a specific batch of fingerprints. The computation performed in the previously mentioned kernels follows a similar pattern, except for K_1, since it implements a modified version of the algorithm described in [25]. Detailed descriptions of these kernels will be presented in Section IV-E.

Inside each kernel, our methodology (Figure 4) states that every thread is in charge of processing a certain minutia from a batch of fingerprints, according to its thread identification number tid. For instance, considering a batch of fingerprints T containing m minutiae and a kernel K, the thread with $tid = s$ will pick and analyze the similarity of the minutia with index s with the ones in the latent fingerprint. This thread will carry out all the requested operations considering the fingerprint limits (starting s and ending e minutia indexes) to which the chosen minutia belongs. After processing the four kernels on the device, a set of partial outcomes vT will be generated for every processed batch of fingerprints. This result is then transferred to the host and used as input to perform the multi-threaded FES step, obtaining similarity scores between the latent and the fingerprint impressions.

D. DATA CONFIGURATION

To bring the aforementioned methodology to reality, a number of data structures are needed to efficiently handle fingerprint processing. All different sorts of structures and parameters required by the ALFI methodology are shown in Tables 1-2, and their descriptions follow:

- $ClusterCount$ is a vector which contains the number of minutiae pairs inside the corresponding cluster from the $ClusterMtiaK$ matrix.
- $ClusterMtiaK$ is a matrix which contains the latent minutiae indexes found while performing alignments for each minutiae pair, which is formed by the one in T and its partner stored in $MaxMtiaL$, working as the centroid of the clusters.
- L is a SoA which includes the information related to the latent fingerprint. It is built in a similar way as the previously described one, but without the use of the index parameter since only one fingerprint is stored.

TABLE 1: Parameter descriptions and values. The values replaced by hyphen symbols indicate that they are dependent on the database used in the experiments or design choices specified in the results section.

Parameter	Description	Value
α	The highest number of minutiae in a fingerprint from the database	-
λ	Max. angular difference between minutiae	$\pi/4$
ξ	Special value used to point the end of an array	-1
b	Bit-vector length of each minutia cylinder	1280
C_b	The number of blocks in the device unit	$32 \cdot C_s$
C_s	The number of SMs available in the device unit	-
C_t	The number of threads per block in the device unit	1024
H_θ	Threshold for angular minutiae similarity	$\pi/6$
H_e	Threshold for distance minutiae similarity	16
H_m	The number of minutiae inside the neighbourhood	3
$H_{q1,q2}$	Thresholds used for computing minutiae quality	18, 42
l	The number of minutiae in the latent fingerprint	-
m_d	The number of minutiae in the fingerprint database	-
m	The number of minutiae in the i-th batch of fps.	-
$N_{D,S}$	Sections and cells in every minutia cylinder	5,16
n_d	Total number of fps. in the fingerprint database	-
n_m	The number of fps. per stream in the device memory	-
n	The number of fps. in the batch of fingerprints	-
tid	Thread identification number	-
z	Total number of quantized angles	256

TABLE 2: Data structures used in the ALFI methodology for host and device units. Parameter descriptions and their values are shown in Table 1.

Name	Layout	Memory Transfer	Device Access
$ClusterCount$	Array$[m]$	D2H	W
$ClusterMtiaK$	Matrix$[m \cdot \alpha]$	D2H	W
L	SoA$[l]$	H2D	R
LUT_D	Matrix$[z \cdot (l+1)]$	-	R/W
LUT_S	Array$[n+1]$	H2D	R
$MatchingValue$	Array$[m]$	D2H	R/W
$MaxMtiaL$	Array$[m]$	D2H	R/W
$QualityL$	Array$[l]$	D2H	W
$QualityT$	Array$[m]$	D2H	W
$Similarity$	Array$[n_d]$	-	-
T_{DB}	SoA$[m_d]$	-	-
T	SoA$[m]$	H2D	R

- LUT_D is a look-up table which includes all allowed latent minutiae indexes for any quantized angle $\hat{\gamma} = 0...z$ so that $LUT_D[\hat{\gamma}] = \{q_h \in L[d_\theta(\hat{\theta}_h, \hat{\gamma}) < \hat{\lambda}]\}$, where d_θ, in this particular case, represents the minimum angular difference between two quantized angles (Equation 1 in Appendix).
- LUT_S is a look-up table which contains the first minutia index for every fingerprint in the database.
- $MatchingValue$ is a vector which includes the similarity value between every minutia in T and its found partner stored in $MaxMtiaL$.
- $MaxMtiaL$ is a vector used to store the most similar latent minutiae indexes from L for each minutia in T.
- $QualityL$ and $QualityT$ are vectors which include the quality value for each minutia in L and T, respectively.
- $Similarity$ is a vector which includes the final matching score between latent and fingerprint impressions.
- T_{DB} is a structure of arrays (SoA) which contains the data of the fingerprint database in an optimal way for processing. In particular, every minutia data inside the fingerprint database is distributed across several vectors according to its different attributes, along with the k index of the fingerprint to which it belongs (Definitions 1 and 2 in the Appendix). In addition, T_{DB} will be split into several batches T for processing and the content of every one can be accessed on the device just by indexing with pointers.

E. PSEUDO-CODES

1) The work of the host: Controlling the device

The host unit controls all further operations to be performed on the device, as presented in Algorithm 1. The parameter r represents the ratio between the number of fingerprints in the database and the size of the fingerprint batches. It is used to indicate how many times the loop is performed. Moreover, i and k are auxiliary variables used as indexes for the execution of the different operations. These operations are queued and will be dispatched sequentially inside each stream, but operations running in different streams can be overlapped. Once the data is successfully transferred to the device, the processing is carried out in the following sequential kernels. Kernel execution configurations are selected after carrying out several tests to obtain the optimal combination that allows the hardware to reach its full performance potential.

Algorithm 1: Host function that controls the device.

```
1  $r = n_d/n$, $i = 2$ and $k = 1$
2  $A(|L|)$ and $A(|T_{DB}|)$ in pinned host memory
3  Split $T_{DB}$ into $T_h, h = 1...r$
4  $A(|L|)$ and $A(2 \cdot n_m)$ in device memory
5  $str_1 \leftarrow$ do $H2D(L)$
6  $str_1 \leftarrow$ launch $K_1$($z$ threads per block, 1 block)
7  $str_1 \leftarrow$ launch $K_3$(128 threads per block, 1 block)
8  $str_1 \leftarrow$ do $D2H(QualityL)$
9  $str_0 \leftarrow$ do $H2D(T_1)$
10 $str_0 \leftarrow$ launch $K_2$($C_t/2$ threads per block, $C_b$ blocks)
11 $str_0 \leftarrow$ launch $K_{3,4}$($C_t$ threads per block, $C_b$ blocks)
12 $str_0 \leftarrow$ do $D2H(vT_1)$
13 $str_1 \leftarrow$ do $H2D(T_2)$
14 $str_1 \leftarrow$ launch $K_2$($C_t/2$ threads per block, $C_b$ blocks)
15 $str_1 \leftarrow$ launch $K_{3,4}$($C_t$ threads per block, $C_b$ blocks)
16 for $iter = 1$ to $r - 2$ do
17     $str_k \leftarrow$ do $D2H(vT_i)$
18     Update $i = i + 1$ and $k = 1 - k$
19     $str_k \leftarrow$ do $H2D(T_i)$ and launch $K_{2-4}$
20     Perform $FES(T_{i-2}, vT_{i-2})$
21 $str_k \leftarrow$ do $D2H(vT_i)$
22 Perform $FES(T_{i-1}, vT_{i-1})$ and $FES(T_i, vT_i)$
```

Algorithm 2: LUT_D computation (Kernel-1).

```
1  $\hat{\lambda} = (z \cdot \lambda)/(2\pi)$ and $\hat{\gamma} = tid$
2  for each $q_h \in L, h = 0...|L|$ and $i = 0$ do
3      $\hat{\theta}_h = (z \cdot \theta_h)/(2\pi)$
4      if $d_\theta(\hat{\theta}_h, \hat{\gamma}) < \hat{\lambda}$ then
5          $LUT_D[\hat{\gamma}][i] = h$ and $i = i + 1$
6  $LUT_D[\hat{\gamma}][i] = \xi$
```

2) Preprocessing angular differences (Kernel-1)

This kernel filters less similar minutiae based on the angular direction similarity as presented in Algorithm 2. This preprocessing technique makes Kernel-2 run faster by avoiding checking the condition in processing. In particular, the LUT_D look-up table will contain all the minutiae indexes from the latent fingerprint that meet the condition for every quantized angle $\hat{\gamma} = 0...z$ [25]. The condition is fulfilled if the minimal angular differences (see Equation 1 in the Appendix) between the minutia direction and the corresponding $\hat{\gamma} = tid$ are below the quantized λ threshold. Regarding the execution of this kernel, only one block of threads with z threads is launched, i.e., one thread per quantized angle.

Algorithm 3: Local matching process (Kernel-2).

```
for each $q_h \in L, h = 0...|L|$ do
    Store $\nu_h$ in shared memory
while $tid < m$ do
    Set $maxSim$ to $max\{Float\}$
    $maxIx = \xi$
    $T[tid] \leftarrow p_t$
    Store $\nu_t$ in local memory
    Set $\hat{\theta}_t = (z \cdot \theta_t)/(2\pi)$ and $i = 0$
    while $LUT_D[\hat{\theta}_t][i] \neq \xi$ do
        $k = LUT_D[\hat{\theta}_t][i]$
        $sim = \sigma(q_k, p_t)$
        if $sim > maxSim$ then
            $maxSim = sim$
            $maxIx = k$
            $i = i + 1$
    $MatchingValue[tid] = maxSim$
    $MaxMtiaL[tid] = maxIx$
    $tid = tid + C_t \cdot C_b$
```

3) Matching minutiae descriptors (Kernel-2)

This kernel aims to find a first set of matching minutiae pairs using the operations shown in Algorithm 3. Every thread manages a particular minutia $p_t \in T, t = tid$ (Definitions 1 and 2 in Appendix) and compares it to every allowed minutia $q_h \in LUT_D[\hat{\theta}_t]$ resulting from the execution of Kernel-1. In the end, the most similar minutia from the latent fingerprint is stored for every impression minutia in the database. This selection is based on the function described in Equation 4 in the Appendix. This kernel is launched using $C_t/2$ threads per block so as not to exceed the maximum register size and optimizing available resources.

4) Minutia quality computation (Kernel-3)

The object of this kernel lies in the calculation of a quality value for every processed minutia as given in Algorithm 4. Every thread takes a particular minutia $p_t \in T, t = tid$ and obtains the quality value depending on the direction consistency between this one and the surrounding minutiae, which form its neighbourhood. The computation of the Euclidean distance is carried out between all minutiae inside a specific circumference to find the H_m closest minutiae inside every fingerprint. The mean distance value is then used to obtain the final quality score for each minutia depending on H_{q1} and H_{q2} thresholds (See Equations 3 and 5 in the Appendix). Likewise, this kernel is also used to obtain the quality value of every minutia in the latent fingerprint. In this case, the kernel is launched using C_t threads per block to optimize available resources.

Algorithm 4: Minutia quality calculation (Kernel-3).

```
while $tid < m$ do
    Set array $distance = \{0\}$
    $T[tid] \leftarrow p_t$
    $s = LUT_S[t]$ and $e = LUT_S[t+1] - 1$
    for each $p_h \in T, h = s...e$ do
        $d = d_e(p_h, p_t)$
        if $(h \neq t) \wedge (d < distance)$ then
            update $distance$ with $d$
    Compute $\bar{d}$ from $H_m$ first values in $distance$
    $QualityT[tid] = \rho\left(\bar{d}\right)$
    $tid = tid + C_t \cdot C_b$
```

Algorithm 5: Finding clusters of matching minutiae pairs (Kernel-4).

```
while $tid < m$ do
    $s = LUT_S[t], e = LUT_S[t+1] - 1$
    $T[tid] \leftarrow p_t$
    $h = maxMtiaL[t]$
    if $h \neq \xi$ then $f_1 = 1$ else $f_1 = 0$ and $h = 0$
    $ClusterMtiaK[t][0] = t$
    for each $p_k \in T, k = s...e$ and $i = 1$ do
        $r = maxMtiaL[k]$
        if $r \neq \xi$ then $f_2 = 1$ else $f_2 = 0$ and $r = 0$
        $q'_r = \psi(q_r, q_h, p_t)$
        $sim = \sigma_e(q'_r, p_k) \cdot \sigma_\theta(q'_r, p_k) \cdot \sigma_t(q'_r, p_k)$
        if $(f_1 \cdot f_2 \cdot sim) > 0$ then
            $ClusterMtiaK[t][i] = k$
            $i = i + 1$
    $ClusterCount[tid] = i$
    $tid = tid + C_t \cdot C_b$
```

5) Finding clusters (Kernel-4)

This kernel, shown in Algorithm 5, finds clusters of similar minutiae pairs after checking for alignments using the initial set obtained in Kernel-2. More specifically, each thread takes a minutia $p_t \in T, t = tid$ and looks for the most similar one in the latent fingerprint. Therefore, two scenarios are possible: (i) all the minutiae in T have a match in L, or (ii) some minutia in T does not have a similar one in L. In the first case, the workload will be balanced between all the threads, so there is no degradation of performance. However, the second case suffers from the thread divergence problem as a few threads will carry on with the processing whereas others will be idle. To address this problem, the first minutia from the latent fingerprint is selected as a dummy structure for those minutiae in T that does not have a similar one. Using this approach, the thread divergence problem is minimized since broad if-else statements are avoided. Once

Algorithm 6: Host function in charge of the final evaluation stage (*FES*).

```
1  for each $fp \in T$ in parallel do
2      $s = LUT_S[fp]$ and $e = LUT_S[fp+1]-1$
3      for each $p_k \in T, k = s...e$ do
4          $h = maxMtiaL[k]$
5          if $h \neq \xi$ then
6              $M = M \cup (q_h, p_k)$
7      for each $p_k \in T, k = s...e$ do
8          for $i = 0$ to $ClusterCount[k]$ do
9              $r = ClusterMtiaK[k][i]$
10             $h = maxMtiaL[r]$
11             $\forall (q_h, p_r) \in M : B_h = B_h \cup (q_h, p_r)$
12         Perform Step 3b from Section IV-A
13     Perform Step 3c-5 and update $Similarity[fp]$
```

this problem has been solved, clusters of minutiae pairs are obtained by performing several alignments following the expressions from Equations 6-9 in the Appendix. This procedure obtains a set of corresponding minutiae indexes from the impression fingerprints, which are stored in $ClusterMtiaK$, and their matched minutiae from the latent fingerprint. This way new matching minutiae pairs are included to the initial group of matches. Regarding the execution of this kernel, it is launched using the same configuration as Kernel-3.

6) Final evaluation stage executed on the host

The results obtained after processing a particular batch of impressions on the device are used as input to the FES function executed on the host (Algorithm 6). This function carries out the final part of the fingerprint matching process, which is performed in parallel at fingerprint matching level. In every fingerprint comparison, found minutiae pairs formed by each impression minutia and its most similar one from the latent fingerprint, stored in the $MaxMtiaL$ vector, will be placed as the centroid of the corresponding cluster. These clusters are formed by reading the impression minutiae indexes previously stored in the $ClusterMtiaK$ vector and considering the number of minutiae for the corresponding cluster in $ClusterCount$ for each minutia inside the impression fingerprint. Finally, consolidation and TPS methods are carried out, obtaining the final similarity values. It should be pointed out that while the FES function is being executed on the host over a particular batch of impressions, the device will finish processing the next one and deliver the results to the host so that idle times are removed.

V. EXPERIMENTS AND RESULTS

This section analyzes and compares our proposal with respect to the state-of-the-art in terms of accuracy and computational performance on widely used databases.

Section V-A explains the experimental setup. Section V-B describes the databases used in the different experiments. Section V-C1 evaluates the accuracy of the ALFI proposal in the latent fingerprint identification task. Additionally, Section V-C2 evaluates the accuracy of ALFI in the verification task. Section V-D assess the computational performance of ALFI in terms of execution time and speed-up for Linux and Windows operating systems. Finally, Section V-E discusses the results accomplished regarding accuracy and computational performance.

A. EXPERIMENTAL SETUP

This research focuses on the design of a new methodology for latent fingerprint identification specifically designed for heterogeneous CPU-GPU systems. With this in mind and after considering the published works in this area so far, we can conclude that:

- DMC is the latent fingerprint identification algorithm that has demonstrated the best results when working with all considered combinations of databases or even with a background database of more than 1.1 million impressions [10]. Apart from its excellent performance in identifying fingerprints, it is the algorithm with the second-best performance in the field of fingerprint verification in the FMISO-HARD-1.0 competition of the FVC-onGoing platform [39], [40], among those developed by academic groups. The algorithm with the best performance in this competition is the MntModel [37]; however, it cannot be replicated since several steps of the development of the algorithm were omitted in the article. Also, this algorithm was not tested on latent fingerprint identification, and its performance carrying out this particular task is unknown.
- Only the works described in [14] and [10] provide the source code or program which allows researchers to replicate the results with different databases, and therefore, we can only compare ourselves against the numbers they report in their articles.
- The DMC-CC version uses the Cylinder-Codes descriptors, which were shown in a recent study to be the best minutiae descriptors for identifying latent fingerprints [13].

Regarding the implementation, ALFI has been developed using C++ and CUDA C++ programming languages. C++ is a compiled language so that it is translated into machine language before being executed. It allows us to test the performance of ALFI on Linux and Windows operating systems (OSs) using the same implementation. We have proven that its use significantly improves the computational performance of the latent identification task according to a previous research presented in [41].

Host codes are compiled with -O2 optimization flag using g++ 5.4 and MSVC 14.16.27023 for Linux (Ubuntu 16.04.5 LTS) and Windows 10, respectively. Devices codes make use of the NVIDIA NVCC compiler from the CUDA compilation tools V10.0.130. The OpenMP C/C++ version 2.0 is used inside the final evaluation stage function to enable the multi-threaded execution at the fingerprint matching level. The Armadillo C++ library version 7.800.2 [42], [43] with OpenBLAS 0.2.14.1 is also used to carry out the necessary linear algebra operations.

B. DATABASES

To test the performance of the ALFI methodology in the identification task, the popular NIST SD27 [44] database is used in the experiments, which includes fingerprints and minutiae. This database holds 258 latent fingerprints collected from real cases, along with their images available at 500 dpi. Every case includes the image of the latent fingerprint and its rolled fingerprint mate, where experts have validated all minutiae. Moreover, we have designed six background databases according to different combinations of fingerprints, as shown in Table 3. Indeed, the NIST SD27 database is further extended with rolled fingerprints from the NIST SD4 [45] and NIST SD14 [46] databases to obtain small (B_{1-3}) and medium (B_{4-5}) sized databases. In order to obtain a more extensive background database, synthetic plain fingerprints generated using the *SFinGe Version 4.1 (build 1746) Demo* were included in B_6. The fingerprints generated with this last software have been used in several fingerprint verification competitions proving that the results achieved with these features are similar to the ones achieved on real databases [47]. However, as plain fingerprints contain less information than rolled ones, this can affect the experiments in terms of accuracy and computational performance. Regarding minutiae per fingerprint, they are extracted using the VeriFinger SDK [48] for the impression fingerprints.

Although ALFI is designed for identification, we choose to test its accuracy on fingerprint verification databases as well to check whether it is suitable for this particular task. The FVC 2002 [49], FVC 2004 [50], and FVC 2006 [51] databases are used to carry out the fingerprint verification experiment. The DB1_A section from FVC 2006 database was discarded due to the low resolution of the images.

On the other hand, the computational performance of ALFI is analyzed under conditions that are as close to a real case as possible. In particular, several of the six background databases (from Table 3) have a similar number of fingerprints and hence, some of them can be dismissed to eliminate redundancy. Therefore, medium and large-sized background databases related to B_3, B_5, and B_6 are considered since they have a representative number of fingerprints.

TABLE 3: Background databases used in the experiments with their number of fingerprints included. The two rightmost columns show the total number of fingerprints and the average number of minutiae extracted per fingerprint, respectively.

Database	NIST SD27	NIST SD4	NIST SD14	Synthetic	N^o fps.	N^o mtiae./fp.
B_1	258	-	-	-	258	21
B_2	258	-	2,000	-	2,258	149
B_3	258	2,000	-	-	2,258	101
B_4	258	-	27,000	-	27,258	163
B_5	258	2,000	27,000	-	29,258	159
B_6	258	2,000	27,000	357,985	387,243	38

C. ACCURACY ANALYSIS

This experiment presents the accuracy results of ALFI using identification and, additionally, verification databases. We compare the ALFI methodology to the state-of-the-art DMC algorithm using the following descriptors: Cylinder-Codes (CC), M-Triplets (MT) and Neighboring Minutiae-based Descriptor (NMD).

1) Identification databases

Cumulative Matching Characteristic (CMC) curves, described in [32], are widely used in the literature to assess the accuracy of identification algorithms that produce an ordered list of possible matches. This type of result plots the probability that a correct identification occurs (rank-k identification rate) within a group of k returned candidates, where k = 1...20. In practice, latent fingerprint examiners may request (i) all returned candidates with a match score above a certain threshold, or (ii) a specific number of highest-ranked candidates instead. In any case, examiners normally begin the analysis with the candidate which has the highest rank (related to rank-1), and continue through the remaining ones if they do not succeed [52]. Therefore, not only the rank-1 value is important for the identification evaluation, but also the rank-20 and the complete CMC curve in order to make it as close to a real case as possible.

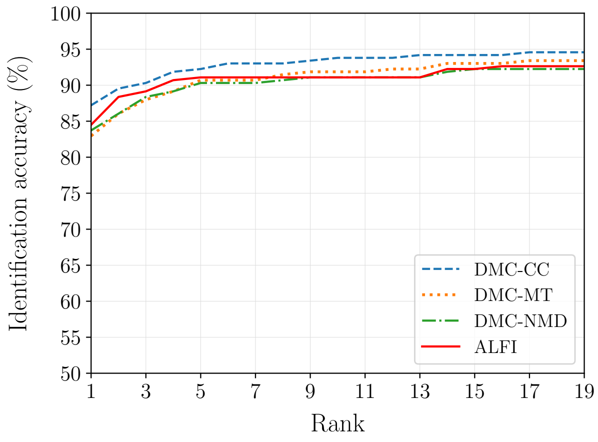
In this experiment, each latent fingerprint in the NIST SD27 is compared to every impression fingerprint in the background database generating the Cumulative Matching Characteristic (CMC) curves shown in Figure 5. These results are complemented by the corresponding rank-1 and

TABLE 4: Rank-1 values from the CMC curves shown in Figure 5. Values are given in percentages.

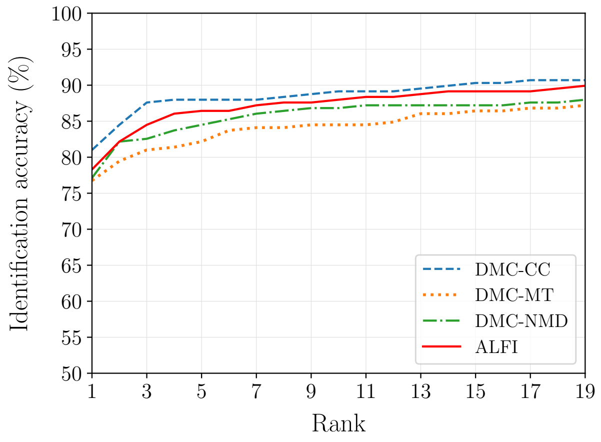
Algorithm	B_1	B_2	B_3	B_4	B_5	B_6
DMC-CC	87.21	81.01	80.62	70.16	69.77	62.79
DMC-MT	82.95	76.74	76.36	69.38	69.38	66.67
DMC-NMD	83.72	77.13	79.46	66.67	66.67	62.79
ALFI	84.50	78.29	77.52	67.83	67.44	61.24

TABLE 5: Rank-20 values from the CMC curves shown in Figure 5. Values are given in percentages.

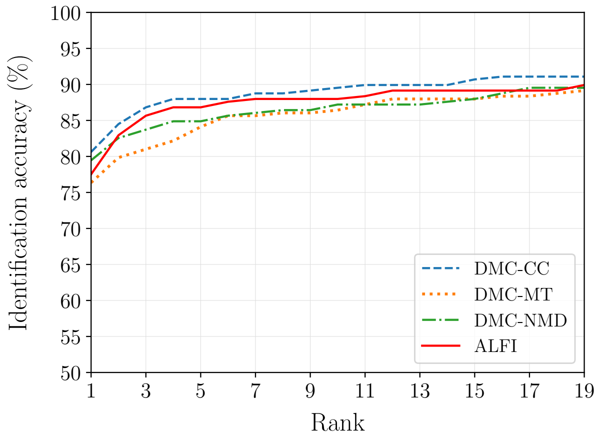
Algorithm	B_1	B_2	B_3	B_4	B_5	B_6
DMC-CC	94.57	91.47	91.09	85.27	84.50	78.68
DMC-MT	93.41	87.60	89.15	79.84	79.84	77.13
DMC-NMD	92.25	87.98	89.53	81.78	81.40	78.29
ALFI	92.64	90.31	89.92	83.72	83.72	75.58



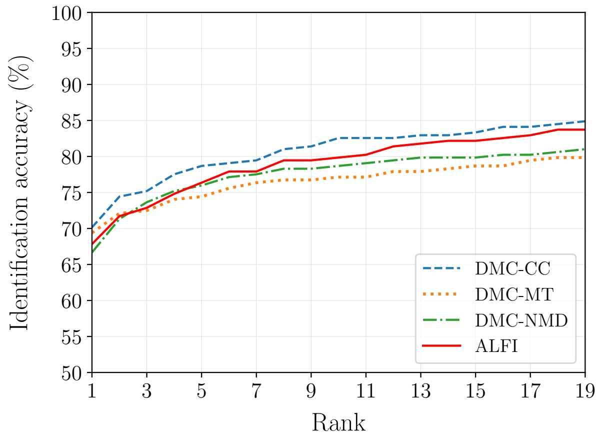
(a) B_1



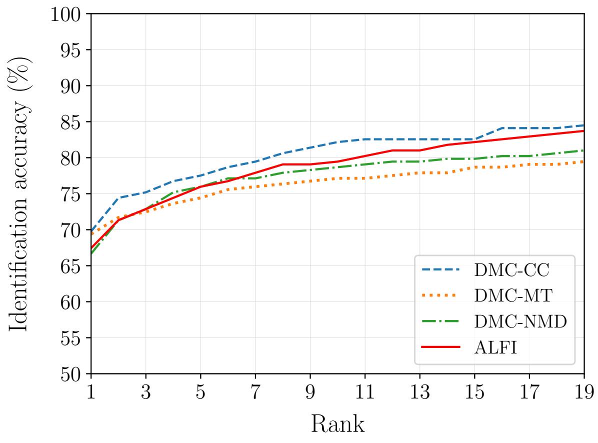
(b) B_2



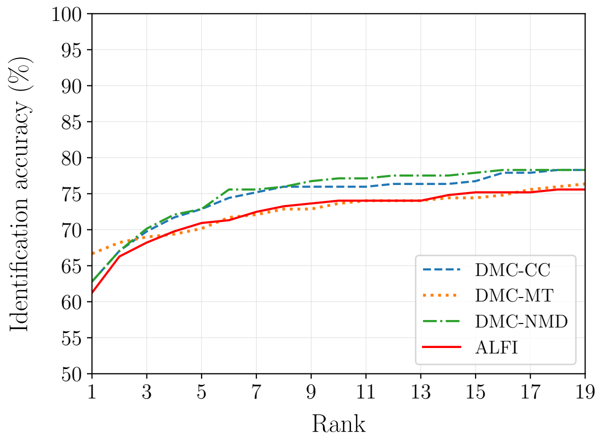
(c) B_3



(d) B_4



(e) B_5



(f) B_6

FIGURE 5: Cumulative Match Curves (CMC) of the DMC algorithms and the ALFI proposal using the NIST SD27 database as reference and six different background databases B_{1-6} described in Table 3.

rank-20 values presented in Tables 4-5 according to different background databases. From these results, the following observations may be made:

- In most cases, the DMC-CC algorithm is the best ranked; however, the difference in accuracy between this version and the ALFI proposal is negligible.
- Compared to the DMC-MT algorithm, ALFI outperforms it by approximately 1.6%, 1.6%, and 1.2% on databases B_1, B_2, and B_3, respectively, considering rank-1 values. For rank-20 values, ALFI outperforms the same algorithm by approximately 2.7%, 0.8%, 3.9%, and 3.9% on databases B_2, B_3, B_4, and B_5, respectively.
- Compared to the DMC-NMD algorithm, ALFI outperforms it by approximately 0.8%, 1.2%, 1.2%, and 0.8% on databases B_1, B_2, B_4, and B_5, respectively, considering rank-1 values. For rank-20 values, ALFI outperforms the same algorithm by approximately 0.4%, 2.3%, 0.4%, 1.9%, and 2.3% on databases B_1, B_2, B_3, B_4, and B_5, respectively.

TABLE 6: Accuracy results for the DMC algorithm and the ALFI proposal on FVC 2002 databases [49].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB1_A	DMC-CC	0.55	0.64	**0.79**	1.18
	DMC-MT	0.65	0.79	1.11	1.25
	DMC-NMD	**0.50**	**0.61**	**0.79**	**1.14**
	ALFI	0.55	0.79	1.00	1.54
DB2_A	DMC-CC	0.50	**0.50**	**0.71**	1.00
	DMC-MT	**0.43**	0.61	0.75	**0.79**
	DMC-NMD	0.60	0.61	0.86	1.04
	ALFI	0.59	0.68	1.00	1.36
DB3_A	DMC-CC	**2.27**	**2.43**	**3.71**	4.82
	DMC-MT	2.54	3.18	4.07	5.18
	DMC-NMD	2.39	3.11	4.32	**4.64**
	ALFI	2.67	3.21	4.14	6.00
DB4_A	DMC-CC	**1.08**	**1.18**	**1.89**	**2.18**
	DMC-MT	1.51	1.86	2.68	3.79
	DMC-NMD	1.58	1.79	2.50	2.75
	ALFI	1.28	1.43	2.36	3.21

TABLE 7: Accuracy results for the DMC algorithm and the ALFI proposal on FVC 2004 databases [50].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB1_A	DMC-CC	**3.24**	5.39	**9.79**	17.39
	DMC-MT	3.76	6.36	10.25	**15.46**
	DMC-NMD	3.62	6.04	12.14	17.75
	ALFI	3.41	**5.18**	11.39	16.21
DB2_A	DMC-CC	**4.18**	5.96	9.00	10.68
	DMC-MT	4.23	**5.68**	**8.21**	**10.04**
	DMC-NMD	4.52	6.11	8.96	13.21
	ALFI	4.52	6.86	9.89	11.46
DB3_A	DMC-CC	**2.74**	4.07	**5.96**	9.54
	DMC-MT	3.38	4.79	8.32	12.46
	DMC-NMD	2.78	4.79	10.14	15.89
	ALFI	2.77	**3.93**	6.36	**7.82**
DB4_A	DMC-CC	**2.15**	**2.82**	**3.89**	**4.46**
	DMC-MT	2.91	3.25	3.96	4.89
	DMC-NMD	2.80	3.32	4.36	4.86
	ALFI	2.91	3.50	4.46	5.75

TABLE 8: Accuracy results for the DMC algorithm and the ALFI proposal on FVC 2006 databases [51].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB2_A	DMC-CC	0.42	**0.35**	**0.50**	**1.18**
	DMC-MT	**0.36**	0.37	**0.50**	1.31
	DMC-NMD	0.51	0.42	0.78	1.88
	ALFI	0.48	0.42	0.60	1.39
DB3_A	DMC-CC	**3.36**	**4.46**	**6.76**	**10.69**
	DMC-MT	3.51	4.95	7.80	12.44
	DMC-NMD	3.39	4.64	8.12	14.13
	ALFI	3.70	5.25	7.96	11.76
DB4_A	DMC-CC	**2.52**	**3.13**	5.91	**8.17**
	DMC-MT	2.75	3.51	**5.07**	11.13
	DMC-NMD	2.57	3.32	6.30	8.83
	ALFI	3.10	3.79	7.66	10.43

2) Verification databases

Although ALFI is a methodology developed specifically for latent identification, its accuracy on fingerprint verification databases is also analyzed. The FVC 2002, FVC 2004, and FVC 2006 databases are used for this purpose, along with the performance evaluation proposed by Cappelli *et al.* [47] based on EER, FMR100, FMR1000, and ZeroFMR indicators where lower values are related to better performance.

The results of this experiment are given in Tables 6-8. From them, the following observations may be made:

- Compared to the DMC-CC algorithm, ALFI performs equal to or better than it for 5 accuracy measurements.
- Compared to the DMC-MT algorithm, ALFI performs equal to or better than it for 15 accuracy measurements.
- Compared to the DMC-NMD algorithm, ALFI performs equal to or better than it for 21 accuracy measurements.

Considering all the accuracy measurements and the high variability on the results, it could be pointed out that the accuracy values of ALFI are in the same range as the ones obtained by the DMC algorithm, even though ALFI is intended for latent identification.

D. COMPUTATIONAL PERFORMANCE ANALYSIS

In this section, we aim at comparing the computational performance of ALFI with the results obtained by the state-of-the-art algorithm. The DMC-CC algorithm is chosen for the comparison due to its better performance compared to DMC-MT and DMC-NMD approaches, as stated in [10].

In the experiment, a random latent fingerprint from the NIST SD27 database is matched against the B_3, B_5, and B_6 background databases (described in Section V-B). We measure the time required to complete this task and the average throughput in processing. This last parameter is measured in KMPS which stands for thousand matches per second. Also, we use three CPU-GPU pair systems (S_{1-3}) whose characteristics are presented in Tables 9-10 with the aim of carrying out a thorough analysis. The outcomes of this experiment are presented according to the operating system (OS), either Linux or Windows, on which the computational performance is measured.

TABLE 9: Characteristics of the host units.

Parameter	S_1 (Linux)	S_2 (Both)	S_3 (Windows)
Processor Type	Intel Xeon	Intel Core	AMD Ryzen
Processor Model	E5-2698 v3	i5-8600K	7-1700x
Number of cores	16	6	8
Number of threads	32	6	16
Frequency (GHz)	2.3	3.6	3.4
Memory RAM (GB)	256	8	16
Cache L1 (kB)	8x64	6x64	8x96
Cache L2 (kB)	8x256	6x256	8x512
Cache L3 (MB)	1x40	1x9	2x8

TABLE 10: Characteristics of the device units.

Parameter	S_1 (Linux)	S_2 (Both)	S_3 (Windows)
Model	GTX 980	GT 1030	GTX 1050-Ti
Architecture	Maxwell	Pascal	Pascal
Number of CUDA cores	2048	384	768
Number of SMs	16	3	6
Base clock (MHz)	1.12	1.22	1.12
Global memory (GB)	4	2	4
Memory per block (kB)	48	48	48
Max. threads per block	1024	1024	1024
Threads per warp	32	32	32
Memory bandwidth (GB/s)	224	48	112
Performance (TFLOPs)	4.6	1.13	2.14

1) Linux OS

The results of this experiment are shown in Table 11 and Figure 6 in terms of execution time and throughput, respectively. The baseline DMC-CC algorithm has been port from C# to C++ to ensure a fair comparison on Linux. The reason for this choice is that the original C# code could not run efficiently on Linux, making it impossible to compare the ALFI methodology with the one presented by the authors in [10]. Best-studied cases show that:

- S_1: ALFI is up to 28.9 times faster than the DMC-CC algorithm on database B_5. The maximum throughput value is 31.13 KMPS and it is achieved by ALFI on database B_6.
- S_2: ALFI is up to 20.6 times faster than the DMC-CC algorithm on database B_6. The maximum throughput value is 43.66 KMPS and it is achieved by ALFI on the same database.

2) Windows OS

The outcomes of this study are shown in Table 12 and Figure 7. The reference DMC-CC algorithm for Windows is the C# implementation presented by their authors. Best-studied cases for every system show that:

- S_2: ALFI is up to 29.2 times faster than the DMC-CC algorithm on database B_6. The maximum throughput value is 23.89 KMPS and it is achieved by ALFI on the same database.
- S_3: ALFI is up to 44.7 times faster than the DMC-CC algorithm on database B_6. The maximum throughput value is 24.29 KMPS and it is achieved by ALFI on the same database.

TABLE 11: Average run-time and speed-up results of the ALFI proposal and DMC-CC on Linux OS.

		S_1		S_2	
Database	Algorithm	Time (s)	Speed-up	Time (s)	Speed-up
B_3	DMC-CC	4.36	1.0	2.63	1.0
	ALFI	0.66	6.6	0.36	7.3
B_5	DMC-CC	91.69	1.0	57.37	1.0
	ALFI	3.17	28.9	3.57	16.1
B_6	DMC-CC	292.85	1.0	183.08	1.0
	ALFI	12.44	23.5	8.87	20.6

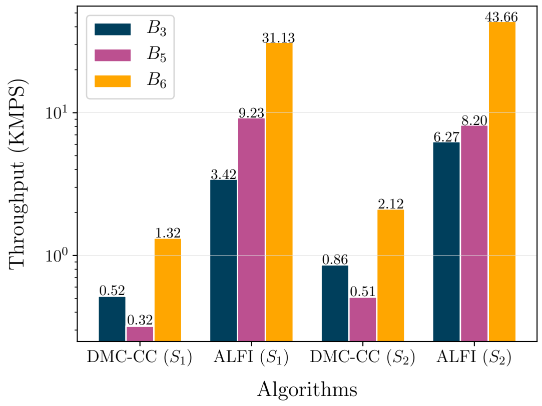


FIGURE 6: Throughput results of the ALFI proposal and DMC-CC on Linux OS.

TABLE 12: Average run-time and speed-up results of the ALFI proposal and DMC-CC on Windows OS.

		S_2		S_3	
Database	Algorithm	Time (s)	Speed-up	Time (s)	Speed-up
B_3	DMC-CC	6.61	1.0	9.48	1.0
	ALFI	0.61	10.8	0.66	14.4
B_5	DMC-CC	146.98	1.0	209.22	1.0
	ALFI	6.71	21.9	5.46	38.3
B_6	DMC-CC	472.66	1.0	712.78	1.0
	ALFI	16.21	29.2	15.94	44.7

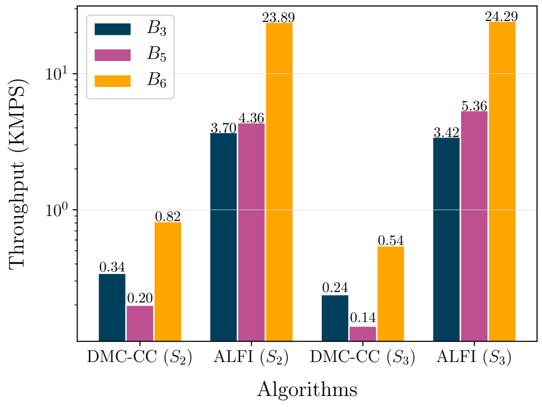


FIGURE 7: Throughput results of the ALFI proposal and DMC-CC on Windows OS.

TABLE 13: Accuracy differences for the identification experiment obtained from analyzing the data in Tables 4-5. The lowest accuracy value of the four algorithms is taken as a reference for every background database.

Algorithm	B_1	B_2	B_3	B_4	B_5	B_6	B_1	B_2	B_3	B_4	B_5	B_6
DMC-CC	4.3%	4.3%	4.3%	3.5%	3.1%	1.6%	2.3%	3.9%	1.9%	5.4%	4.7%	3.1%
DMC-MT	0.0%	0.0%	0.0%	2.7%	2.7%	5.4%	1.2%	0.0%	0.0%	0.0%	0.0%	1.6%
DMC-NMD	0.8%	0.4%	3.1%	0.0%	0.0%	1.6%	0.0%	0.4%	0.4%	1.9%	1.6%	2.7%
ALFI	1.6%	1.6%	1.2%	1.2%	0.8%	0.0%	0.4%	2.7%	0.8%	3.9%	3.9%	0.0%
	rank-1						rank-20					

E. ANALYSIS OF THE RESULTS

The primary goal of the ALFI development lies in obtaining the best possible computational performance in the latent identification procedure without compromising accuracy. Considering this, ALFI has accomplished significant results in both terms.

1) Accuracy

In latent fingerprint identification, the accuracy values of ALFI are within the same range as the ones obtained by the state-of-the-art algorithms, as shown in Table 13. However, the inclusion of graphical processing units in processing results in a slight accuracy reduction for some background databases and k-ranks compared to the reference algorithm in latent identification. The reason for this lies in the impossibility of developing a dynamic algorithm for GPU processing, which profoundly affects the early stage of the processing where matching minutiae pairs must be found. Using the host code, it is possible to obtain very different numbers of minutiae pairs (dynamic allocation in memory) from one fingerprint comparison to another without compromising performance. However, using the device code, a maximum number of minutiae pairs must be imposed to improve performance (forced fixed allocation in memory) resulting in the loss of some possible matching minutiae pairs. This drawback is balanced with the significant improvement achieved in computational performance.

2) Computational performance

ALFI has proven to outperform the state-of-the-art algorithm in execution time for every studied database and operating system in the latent identification task. This achievement is based on the fact that the workload is balanced between the CPU and GPU using asynchronous processing and fine-grained parallelism so that idle times are drastically reduced. On the contrary, the state-of-the-art algorithm is designed for single-thread execution and neglects the use of GPUs to accelerate the processing.

The throughput experiment also revealed that this parameter increases with the size of the database for the ALFI methodology. On the contrary, the throughput of the DMC-CC algorithm decreases between databases B_3 and B_5, but increases with B_6. The explanation for this lies in the difference in the nature of the fingerprints included in the databases and the nonlinearity of their processing. In particular, B_6 contains 7.6% of rolled fingerprints and the rest are plain fingerprints. This latter type includes less information per fingerprint and they are therefore processed faster compared to rolled ones. Indeed, database B_6 has in average 38 minutiae per fingerprint; whereas databases B_3 and B_5 have 101 and 159, respectively, as given in Table 3.

VI. CONCLUSIONS

In this paper, we present a novel methodology called ALFI based on Asynchronous processing for latent fingerprint identification on heterogeneous CPU-GPU systems. ALFI efficiently overlaps and synchronizes two tasks related to (i) the data processing on the device which involves finding matching minutiae pairs, computing minutia quality and obtaining clusters, and (ii) the multi-threaded final evaluation stage performed on the host that evaluates clusters and returns the possible matched fingerprints. This methodology reduces idle times in host and device units, obtaining faster similarity results between latent and fingerprint impressions. Besides, the novel strategy applied to the data processing on the device takes advantage of the intrinsic parallelism of the latent identification process. It makes each thread from the device to process a particular minutia from a batch of fingerprints, and compares it with every minutia from the latent fingerprint.

ALFI has been tested on Linux and Windows operating systems using three different CPU-GPU pair systems. Well-known identification databases such as NIST SD27, NIST SD14, and NIST SD4 were used to test the accuracy of the proposed algorithm in latent fingerprint identification. Additionally, the FVC 2002, FVC 2004, and FVC 2006 verification databases were also used to test the verification performance. Experiments have proven that ALFI outperforms the state-of-the-art DMC algorithm in computational performance up to 22x in average maintaining the accuracy results within the same range. In particular, considering the best-studied case ALFI yields a speed-up of 44.7x. To the best of our knowledge, ALFI is the first methodology in the literature for latent fingerprint identification that is specifically designed to fully exploit the capabilities of heterogeneous CPU-GPU systems.

APPENDIX A

Definition 1. Minutiae are points located in the ridge discontinuities of a fingerprint. Every minutia structure is characterized by x and y positions, θ direction, cylinder $c = (\nu, \eta)$ containing its value and norm, respectively, and also type $t \in [0, 1, 2] : t \rightarrow$ [unknown, end, bifurcation].
Definition 2. The set of all possible minutiae is defined as $A = \{(x, y, \theta, \nu, \eta, t) : x, y, \theta, \eta \in \mathbb{R} \,|\, \nu, t \in \mathbb{N}\}$, where $\mathbb{R}$

and $\mathbb{N}$ represent the sets of real and natural numbers.

The function d_θ computes the minimal difference between two quantized angles $\hat{\alpha}$ and $\hat{\beta}$:

$d_\theta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$(\hat{\alpha}, \hat{\beta}) \to min\,(\,|\hat{\alpha} - \hat{\beta}|, z - |\hat{\alpha} - \hat{\beta}|\,) \qquad (1)$$

where z is the total number of quantized angles. Furthermore, this function can be also used with two given minutiae a and b as inputs:

$d_\theta : A \times A \to \mathbb{R}$

$$(a, b) \to min\,(\,|\theta_a - \theta_b|, 2\pi - |\theta_a - \theta_b|\,) \qquad (2)$$

The function d_e computes the Euclidean distance given two minutiae a and b:

$d_e : A \times A \to \mathbb{R}$

$$(a, b) \to \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \qquad (3)$$

The function σ computes the similarity score between two given minutiae a and b by using their minutiae descriptors:

$\sigma : A \times A \to \mathbb{R}$

$$(a, b) \to 1 - \frac{\sqrt{pop(\nu_a \oplus \nu_b)}}{\eta_a + \eta_b} \qquad (4)$$

where ν is related to the minutia cylinder value, η is the cylinder norm, the XOR operator is denoted as $\oplus$ and the function pop is the bit population count operation.

The function ρ returns the corresponding quality value for an specific minutia depending on H_{q1} and H_{q2} thresholds and a given distance d:

$\rho : \mathbb{R} \to \mathbb{R}$

$$d \to \begin{cases} 1 & if\;\; d > H_{q2} \\ 0 & if\;\; d < H_{q1} \\ (d - H_{q1})/(H_{q2} - H_{q1}) & otherwise \end{cases} \qquad (5)$$

The function ψ maps the minutia a into another by using a minutiae pair (b, c) as reference:

$\psi : A \times A \times A \to A$

$$(a, b, c) \to \left[\begin{pmatrix} c(\Delta\theta) & -s(\Delta\theta) & 0 \\ s(\Delta\theta) & c(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_a - x_b \\ y_a - y_b \\ \theta_a - \theta_b \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \\ \theta_c \end{pmatrix}\right]^T \qquad (6)$$

where c and s denote the trigonometric sine and cosine functions, respectively, and $\Delta\theta = \theta_c - \theta_b$.

The function σ_e computes the similarity score between two given minutiae a and b according to their Euclidean distance:

$\sigma_e : A \times A \to [0, 1]$

$$(a, b) \to \begin{cases} 0 & if\;\; u\;\; or\;\; v\;\; or\;\; w \\ 1 - \frac{d_e(a,b)}{H_e} & otherwise \end{cases} \qquad (7)$$

where $\Delta x = x_b - x_a$, $\Delta y = y_b - y_a$, the empirical threshold value is denoted as H_e, and d_e is the Euclidean distance function described in Equation 3. Also, u denotes $|\Delta x| > H_e$, v denotes $|\Delta y| > H_e$ and w denotes $d_e(a, b)^2 > {H_e}^2$.

The function σ_θ computes the similarity score between two given minutiae a and b according to their direction difference:

$\sigma_\theta : A \times A \to [0, 1]$

$$(a, b) \to \begin{cases} 0 & if\;\; d_\theta(a, b) > H_\theta \\ 1 - \frac{d_\theta(a,b)}{H_\theta} & otherwise \end{cases} \qquad (8)$$

where H_θ is an empirical threshold value and d_θ is the angular difference function described in Equation 2.

The function σ_t computes the similarity score between two given minutiae a and b based on their types:

$\sigma_t : A \times A \to [0.5, 0.75, 1]$

$$(a, b) \to \begin{cases} 0.75 & if\;\; t_a = 0\;\; or\;\; t_b = 0 \\ g & otherwise \end{cases} \qquad (9)$$

$g : [0, 1, 2] \times [0, 1, 2] \to [0.5, 1]$

$$(t_a, t_b) \to \begin{cases} 1 & if\;\; t_a = t_b \\ 0.5 & otherwise \end{cases}$$

REFERENCES

[1] K. Cao and A. K. Jain, “Automated latent fingerprint recognition,” IEEE transactions on pattern analysis and machine intelligence, vol. 41, no. 4, pp. 788–800, 2018.

[2] K. Cao, D.-L. Nguyen, C. Tymoszek, and A. K. Jain, “End-to-end latent fingerprint search,” IEEE Transactions on Information Forensics and Security, vol. 15, pp. 880–894, 2019.

[3] A. B. Kanbar, “Fingerprint identification for forensic crime scene investigation,” International Journal of Computer Science and Mobile Computing, vol. 5, no. 8, pp. 60–65, 2016.

[4] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, Handbook of fingerprint recognition. Springer Science & Business Media, 2009.

[5] S. L. Cooper, “Challenges to fingerprint identification evidence: Why the courts need a new approach to finality,” Mitchell Hamline L. Rev., vol. 42, p. 756, 2016.

[6] D. Peralta, I. Triguero, R. Sanchez-Reillo, F. Herrera, and J. M. Benitez, “Fast fingerprint identification for large databases,” Pattern Recognition, vol. 47, no. 2, pp. 588–602, 2014.

[7] A. Sankaran, M. Vatsa, and R. Singh, “Latent fingerprint matching: A survey.” IEEE Access, vol. 2, pp. 982–1004, 2014.

[8] S. Kiltz, M. Hildebrandt, J. Dittmann, and C. Vielhauer, “Challenges in contact-less latent fingerprint processing in crime scenes: Review of sensors and image processing investigations.” in 2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO). IEEE, 2012, pp. 1504–1508.

[9] B. DeCann and A. Ross, “Can a “poor” verification system be a “good” identification system? a preliminary study.” in 2012 IEEE International Workshop on Information Forensics and Security (WIFS). IEEE, 2012, pp. 31–36.

[10] M. A. Medina-Pérez, A. M. Moreno, M. A. F. Ballester, M. García-Borroto, O. Loyola-González, and L. Altamirano-Robles, “Latent fingerprint identification using deformable minutiae clustering,” Neurocomputing, vol. 175, pp. 851–865, 2016.

[11] K. Karu and A. K. Jain, “Fingerprint classification,” Pattern recognition, vol. 29, no. 3, pp. 389–404, 1996.

[12] A. Ross, A. Jain, and J. Reisman, “A hybrid fingerprint matcher,” Pattern Recognition, vol. 36, no. 7, pp. 1661–1673, 2003.

[13] D. Valdes-Ramirez, M. A. Medina-Pérez, R. Monroy, O. Loyola-González, J. Rodríguez, A. Morales, and F. Herrera, “A review of fingerprint feature representations and their applications for latent fingerprint identification: Trends and evaluation.” IEEE Access, vol. 7, pp. 48 484–48 499, 2019.

[14] R. Cappelli, M. Ferrara, and D. Maltoni, “Minutia cylinder-code: A new representation and matching technique for fingerprint recognition.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 12, pp. 2128–2141, 2010.

[15] D. Peralta, M. Galar, I. Triguero, D. Paternain, S. García, E. Barrenechea, J. M. Benítez, H. Bustince, and F. Herrera, “A survey on fingerprint minutiae-based local matching for verification and identification: Taxonomy and experimental evaluation,” Information Sciences, vol. 315, pp. 67–87, 2015.

[16] W. Bian, D. Xu, Q. Li, Y. Cheng, B. Jie, and X. Ding, “A survey of the methods on fingerprint orientation field estimation,” IEEE Access, vol. 7, pp. 32 644–32 663, 2019.

[17] S. Yoon, J. Feng, and A. K. Jain, “Latent fingerprint enhancement via robust orientation field estimation.” in 2011 international joint conference on biometrics (IJCB). IEEE, 2011, pp. 1–8.

[18] J. Li, J. Feng, and C.-C. J. Kuo, “Deep convolutional neural network for latent fingerprint enhancement.” Signal Processing: Image Communication, vol. 60, pp. 52–63, 2018.

[19] L. J. Gonzalez-Soler, M. Gomez-Barrero, L. Chang, A. Perez-Suarez, and C. Busch, “On the impact of different fabrication materials on fingerprint presentation attack detection,” in 2019 International Conference on Biometrics, ICB 2019, 2019, pp. 1–8.

[20] H. Liu, W. Zhang, G. Liu, and F. Liu, “A zero-shot based fingerprint presentation attack detection system,” arXiv preprint arXiv:2002.04908, 2020.

[21] R. Agarwal, A. S. Jalal, and K. V. Arya, “A review on presentation attack detection system for fake fingerprint,” Modern Physics Letters B, vol. 34, no. 05, p. 2030001, 2020.

[22] K. N. Win, K. Li, J. Chen, P. F. Viger, and K. Li, “Fingerprint classification and identification algorithms for criminal investigation: A survey,” Future Generation Computer Systems, 2019.

[23] D. Peralta, I. Triguero, S. García, Y. Saeys, J. M. Benitez, and F. Herrera, “On the use of convolutional neural networks for robust classification of multiple fingerprint captures,” International Journal of Intelligent Systems, vol. 33, no. 1, pp. 213–230, 2018.

[24] P. D. Gutierrez, M. Lastra, F. Herrera, and J. M. Benitez, “A high performance fingerprint matching system for large databases based on gpu,” IEEE Transactions on Information Forensics and Security, vol. 9, no. 1, pp. 62–71, 2014.

[25] R. Cappelli, M. Ferrara, and D. Maltoni, “Large-scale fingerprint identification on gpu,” Information Sciences, vol. 306, pp. 1–20, 2015.

[26] M. Lastra, J. Carabaño, P. D. Gutiérrez, J. M. Benítez, and F. Herrera, “Fast fingerprint identification using gpus,” Information Sciences, vol. 301, pp. 195–214, 2015.

[27] X. Jiang and W.-Y. Yau, “Fingerprint minutiae matching based on the local and global structures,” in Proceedings 15th international conference on pattern recognition. ICPR-2000, vol. 2. IEEE, 2000, pp. 1038–1041.

[28] H. H. Le, N. H. Nguyen, and T.-T. Nguyen, “Speeding up and enhancing a large-scale fingerprint identification system on gpu,” Journal of Information and Telecommunication, vol. 2, no. 2, pp. 147–162, 2018.

[29] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, “A survey on parallel computing and its applications in data-parallel problems using gpu architectures,” Communications in Computational Physics, vol. 15, no. 2, pp. 285–329, 2014.

[30] A. Mikaelyan and J. Bigun, “Ground truth and evaluation for latent fingerprint matching.” in 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2012, pp. 83–88.

[31] A. K. Jain, J. Feng, A. Nagar, and K. Nandakumar, “On matching latent fingerprints,” in 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2008, pp. 1–8.

[32] A. K. Jain and J. Feng, “Latent fingerprint matching,” IEEE Transactions on pattern analysis and machine intelligence, vol. 33, no. 1, pp. 88–100, 2011.

[33] A. A. Paulino, J. Feng, and A. K. Jain, “Latent fingerprint matching using descriptor-based hough transform,” IEEE Transactions on Information Forensics and Security, vol. 8, no. 1, pp. 31–45, 2013.

[34] S. S. Arora, E. Liu, K. Cao, and A. K. Jain, “Latent fingerprint matching: performance gain via feedback from exemplar prints,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 12, pp. 2452–2465, 2014.

[35] A. M. Bazen and S. H. Gerez, “Fingerprint matching by thin-plate spline modelling of elastic deformations,” Pattern Recognition, vol. 36, no. 8, pp. 1859–1867, 2003.

[36] “Nvidia corporation,” Accessed January 12, 2020, https://www.nvidia.com.

[37] K. Cao, E. Liu, L. Pang, J. Liang, and J. Tian, “Fingerprint matching by incorporating minutiae discriminability,” in 2011 International Joint Conference on Biometrics (IJCB). IEEE, 2011, pp. 1–6.

[38] M. A. Medina-Pérez, M. García-Borroto, A. E. Gutierrez-Rodríguez, and L. Altamirano-Robles, “Improving fingerprint verification using minutiae triplets,” Sensors, vol. 12, no. 3, pp. 3418–3437, 2012.

[39] “Fvc-ongoing,” Accessed November 13, 2019, https://biolab.csr.unibo.it/FVCOnGoing.

[40] B. Dorizzi, R. Cappelli, M. Ferrara, D. Maio, D. Maltoni, N. Houmani, S. Garcia-Salicetti, and A. Mayoue, “Fingerprint and on-line signature verification competitions at icb 2009,” in International Conference on Biometrics. Springer, 2009, pp. 725–732.

[41] A. J. Sanchez, L. F. Romero, S. Tabik, M. A. Medina-Pérez, and F. Herrera, “A first step to accelerating fingerprint matching based on deformable minutiae clustering,” in Conference of the Spanish Association for Artificial Intelligence. Springer, 2018, pp. 361–371.

[42] C. Sanderson and R. Curtin, “Armadillo: a template-based c++ library for linear algebra,” Journal of Open Source Software, vol. 1, no. 2, p. 26, 2016.

[43] ——, “A user-friendly hybrid sparse matrix class in c++,” in International Congress on Mathematical Software. Springer, 2018, pp. 422–430.

[44] M. D. Garris and M. D. Garris, NIST special database 27: Fingerprint minutiae from latent and matching tenprint images. US Department of Commerce, National Institute of Standards and Technology, 2000.

[45] C. I. Watson and C. Wilson, “Nist special database 4,” Fingerprint Database, National Institute of Standards and Technology, vol. 17, no. 77, 1992.

[46] C. I. Watson, “Nist special database 14: Mated fingerprint cards pairs 2 version 2,” tech. rep., Citeseer, 2001.

[47] R. Cappelli, D. Maio, D. Maltoni, J. L. Wayman, and A. K. Jain, “Performance evaluation of fingerprint verification systems,” IEEE transactions on pattern analysis and machine intelligence, vol. 28, no. 1, pp. 3–18, 2006.

[48] “Verifinger sdk,” Accessed January 20, 2020, http://www.neurotechnology.com/verifinger.html.

[49] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, “Fvc2002: Second fingerprint verification competition,” in Object recognition supported by user interaction for service robots, vol. 3. IEEE, 2002, pp. 811–814.

[50] ——, “Fvc2004: Third fingerprint verification competition,” in International Conference on Biometric Authentication. Springer, 2004, pp. 1–7.

[51] R. Cappelli, M. Ferrara, A. Franco, and D. Maltoni, “Fingerprint verification competition 2006,” Biometric Technology Today, vol. 15, no. 7-8, pp. 7–9, 2007.

[52] N. Ratha and R. Bolle, Automatic fingerprint recognition systems. Springer Science & Business Media, 2003.

ANDRES J. SANCHEZ received the B.S. degree in Industrial Technology Engineering with specialization in Automatic Systems and the M.S. degree in Mechatronic Engineering from the University of Malaga, Malaga, Spain, in 2016 and 2017, respectively. In 2018, he joined the Department of Computer Architecture, University of Malaga, where he is currently pursuing the Ph.D. degree in Mechatronic Engineering in the field of parallel programming on heterogeneous systems. His research interests include algorithm optimization, parallel programming and large-scale data processing on heterogeneous GPU-CPU systems.

LUIS F. ROMERO received the M.Sc. degree in Physics from the Complutense University of Madrid, Madrid, Spain, in 1988 and the Ph.D. degree in Computer Science from the University of Malaga, Malaga, Spain, in 1996. He is a Full Professor in the Department of Computer Architecture at the University of Malaga, where he has been since 1989. His research interests span both heterogeneous parallel computing and computer physics. Much of his work has been on improving the understanding, design, and performance of parallel and networked computer systems, mainly through physical system modelling, GIS algorithmics and numerical integration.

DANIEL PERALTA received the M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2011 and 2016 respectively. He is currently a post-doctoral researcher at Ghent University and the Vlaams Instituut voor Biotechnologie (Ghent, Belgium) within the Data Mining and Modeling for Biomedicine research group. He has published 15 papers in international journals, and received the BBVA Award for Young Computer Science Researchers in 2018. His research interests include data mining, biological imaging data analysis, biometrics and large-scale parallel and distributed computing.

PROF. MIGUEL ANGEL MEDINA-PÉREZ received the Ph.D. degree in Computer Science from the National Institute of Astrophysics, Optics, and Electronics, Mexico, in 2014. He is currently a Research Professor with the Tecnologico de Monterrey, Campus Estado de Mexico, where he is also a member of the GIEE-ML (Machine Learning) Research Group. Prof. Medina-Pérez has rank 1 in the Mexican Research System. His research interests include Pattern Recognition, Data Visualization, Explainable Artificial Intelligence, Fingerprint Recognition, and Palm Print Recognition. Prof. Medina-Pérez has published tens of papers in referenced journals such as "IEEE Transactions on Information Forensics and Security," "Pattern Recognition," "Information Fusion," "Knowledge-Based Systems," "Information Sciences," and "Neurocomputing." Prof. Medina-Pérez has an extensive experience developing software to solve pattern recognition problems. A successful example is a fingerprint and palm print recognition framework which has more than 1.2 million visits and 132 thousand downloads.

SIHAM TABIK received the B.Sc. degree in Physics from University Mohammed V, Rabat, Morocco, in 1998 and the Ph.D. degree in Computer Science from the University of Almeria, Almeria, Spain, in 2006. She is currently Ramón y Cajal researcher at the University of Granada, Granada, Spain. Her research interests include machine learning and high performance computing.

YVAN SAEYS received the M.Sc. and Ph.D. degrees in Computer Science from Ghent University, Ghent, Belgium. After spending time abroad at the University of the Basque Country and the University of Lyon, he established the DAMBI research group at VIB and Ghent University, where he is currently an Associate Professor in the Department of Applied Mathematics, Computer Science and Statistics. His research focuses on the development and application of data mining and machine learning techniques for biological and medical applications. He has published more than 100 papers in international journals and high-profile conferences and has been involved in the organization of many international workshops and conferences on machine learning and bioinformatics. His current research interests include instance and feature selection, large-scale machine learning, and machine learning for single-cell biology.

FRANCISCO HERRERA received the M.Sc. and Ph.D. degrees Mathematics from the University of Granada, Granada, Spain, in 1988 and 1991, respectively. He is currently a Full Professor in the Department of Computer Science and Artificial Intelligence from the University of Granada, Granada, Spain. He has been the Supervisor of 42 Ph.D. students. He has published more than 370 journal papers, receiving more than 71,000 citations and an h-index of 135.

• • •