Pattern Recognition 47 (2014) 588–602

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Fast fingerprint identification for large databases

D. Peralta [a,*], I. Triguero [a], R. Sanchez-Reillo [b], F. Herrera [a], J.M. Benitez [a]

[a] Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada 18071, Spain
[b] University Group for Identification Technologies (GUTI) at the Electronics Technology Department, Carlos III University of Madrid, Avda. Universidad, 30, 28911 Leganes, Madrid, Spain

ARTICLE INFO

Article history:
Received 13 March 2013
Received in revised form
6 June 2013
Accepted 1 August 2013
Available online 14 August 2013

Keywords:
Distributed computing
Large databases
Minutiae matching
Parallel computing
Real-time fingerprint identification

ABSTRACT

Fingerprint matching has emerged as an effective tool for human recognition due to the uniqueness, universality and invariability of fingerprints. Many different approaches have been proposed in the literature to determine faithfully if two fingerprint images belong to the same person. Among them, minutiae-based matchers highlight as the most relevant techniques because of their discriminative capabilities, providing precise results. However, performing a fingerprint identification over a large database can be an inefficient task due to the lack of scalability and high computing times of fingerprint matching algorithms.

In this paper, we propose a distributed framework for fingerprint matching to tackle large databases in a reasonable time. It provides a general scheme for any kind of matcher, so that its precision is preserved and its time of response can be reduced.

To test the proposed system, we conduct an extensive study that involves both synthetic and captured fingerprint databases, which have different characteristics, analyzing the performance of three well-known minutiae-based matchers within the designed framework. With the available hardware resources, our distributed model is able to address up to 400 000 fingerprints in approximately half a second. Additional details are provided at http://sci2s.ugr.es/ParallelMatching.

1. Introduction

Personal identification is one of the largest problems in the society today in a wide variety of fields: from access control to criminology and forensic identifications, payments and identification in computer systems [1]. Among all the biometric features that can be used for identification, such as voice, iris, DNA, fingerprints are the most widely used [2]. They are very suitable for human recognition because of their uniqueness, universality, invariability and extraction facilities.

A fingerprint is basically a pattern of ridges and valleys captured from a finger by inked press, capacitive or optical sensors, etc. Fingerprint recognition has been studied for many years and a great number of fingerprint matching algorithms have been proposed in the specialized literature [3,4]. Minutiae-based matching algorithms highlight as the most relevant approaches because minutiae are considered the most discriminating and reliable features [5,6]. The design of Automatic Fingerprint Identification Systems (AFISs) [7] is an important task in pattern recognition. Although very effective solutions are currently available, many problems still remain [8]. Among them, the performance and speed of AFISs for large databases need to be improved.

Fingerprint recognition can be categorized into two different problems: verification [9] and identification [10]. The former consists of determining whether two images belong to the same fingerprint, that is, a one-to-one comparison. The latter is devoted to search for the matching of an input fingerprint in a template database, so that the owner of this fingerprint can be identified. Thus, identification can be seen as a generalization of the verification problem that conducts one-to-many comparisons. In this paper, we will focus on identification.

In general, matching algorithms are designed to carry out a fingerprint verification and their generalization to address identification is straightforward. Most of them are focused on achieving very accurate matchings, what usually negatively affects the time consumption. This factor is determinant in most real time systems where a high response time is equivalent to a system failure. Furthermore, this weakness is especially harmful when the number of templates in the database is increased. Although some approaches have been designed to be as fast as possible [6], they are not suitable to tackle large databases maintaining their precision.

High Performance Computing (HPC) is one of the tools that support the modern Science, allowing the execution of multiple calculations in a reasonable time [11] by using an adequate massive computational structure [12]. HPC has been successfully used in many different pattern recognition problems [13–15], and more

* Corresponding author. Tel.: +34 958244019; fax: +34 958243317.
E-mail addresses: dperalta@decsai.ugr.es (D. Peralta), triguero@decsai.ugr.es (I. Triguero), rsreillo@ing.uc3m.es (R. Sanchez-Reillo), herrera@decsai.ugr.es (F. Herrera), J.M.Benitez@decsai.ugr.es (J.M. Benitez).

0031-3203/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2013.08.002

concretely in real-time image comparison [16] and other artificial intelligence systems [17]. Given the complexity order of an AFIS, HPC is a promising resource that has already been proven to reduce the identification time [18,19]. However, the proposals in the current scientific literature focus on objectives other than performance, such as high availability or database distribution. Real-time response times can only be obtained through a correct algorithm design and implementation in order to exploit the available resources as flexibly and efficiently as possible.

In this paper, we design a two-level distributed framework to provide matching algorithms the capacity of dealing with arbitrarily large databases by adapting the underlying hardware. According to the so far presented reasons, three objectives are defined for this paper:

- To analyze the behavior of matching algorithms when dealing with large databases.
- To verify the scalability of the proposed system.
- To provide a real-time answer.

To check the performance of the proposed system, we will conduct experiments involving up to 400 000 fingerprints. Because of the absence of large captured fingerprint databases, we use the SFinGe software tool [20,2] to generate a large synthetic database. This database is used for experiments both with the ground-truth minutia provided by SFinGe and using the NIGOS *mindtct* [21] minutiae extractor in a seek of a more realistic framework. Furthermore, in order to validate the results we also include experiments with captured databases: NIST DB4 [22] and DB14 [23].

Due to the space constraints not every experiment could be included in the paper. Complementary material about the work done for this paper can be found at the URL http://sci2s.ugr.es/ParallelMatching.

The rest of this paper is organized as follows: Section 2 provides a description of the fingerprint recognition process, defining in detail the most important steps. In Section 3, the HPC paradigm is presented, showing its hardware and software requirements, theoretical benefits and current applications to AFISs. Section 4 explains the proposed distributed system for tackling the fingerprint identification problem in a reasonable time. Section 5 describes the experimental framework. Section 6 examines the results obtained, presenting a discussion of them. Finally, Section 7 concludes the paper.

2. Background

A considerable research effort has been carried out in the fingerprint recognition field over the last decades. This section sums up the state-of-the-art in that field, starting with the fingerprint recognition problem (Section 2.1), and explaining the generalities of feature extraction (Section 2.2) and fingerprint matching (Section 2.3).

2.1. Fingerprint recognition

Because of its different application fields, most authors divide the fingerprint recognition problem into two variants that constitute by themselves different problems [2]:

- *Verification* consists of determining whether two fingerprint images P_1 and P_2 belong to the same person, performing a 1:1 comparison [9]. The system output is an acceptation or a refusal of the claimed identity depending on the similarity level (called score) of both fingerprints.
- *Identification* aims to find the fingerprint that matches with the input fingerprint in a database, so that its owner can be identified [10]. A fingerprint database is a set T of N template fingerprints $T=\{T_1,T_2,\ldots,T_N\}$ that are used as reference for the identification. Thus, identification is a problem of 1:N comparison as the input fingerprint I needs to be compared with all T_i template fingerprints (with $i\in\{1,2,\ldots,N\}$) to find the matching that provides the highest score. This score is called m_{best}. It is defined in Eq. (1), where $Q(I,T_i)$ is the matching function (see Section 2.3). If m_{best} is lower than a certain threshold ϕ, then the system may consider that the input fingerprint has no corresponding template in the database. Hence, the system output can be the matched identity, a "not found" notification, or a set of candidate identities. This paper focuses on a system that considers only the maximum score, so the last case is not detailed, as shown in Eq. (2). A description of the system behavior in the case with a set of candidates can be found in the web site:

$$m_{best}=\max\{Q(I,T_i)|i\in\{1,2,\ldots,N\}\} \tag{1}$$

$$Id(I)=\begin{cases}\text{not found}, & \text{if } m_{best}>\phi\\ \underset{i}{\operatorname{argmax}}Q(I,T_i)i\in\{1,\ldots,N\}, & \text{otherwise}\end{cases} \tag{2}$$

The identification problem can be seen as a verification performed once per each fingerprint in the database. The main difference between these problems is therefore a matter of complexity order. The objective in a verification problem is to obtain a very precise result, reducing the error rates as much as possible. However, complex verification methods are not useful for identification because the overall response time would be excessive.

So far, the general characteristics of the identification problem have been defined. The requirements needed by an AFIS to deal with large databases can be fixed:

- *Precision*: error rates have to be as low as possible in order to get an accurate system. Additional information about error rates can be found in the web site associated with this paper.
- *Efficiency*: the time that is needed to locate a fingerprint in the database should be as small as possible. In a real-time system, for example, a high delay can be equivalent to a system failure [24]. The delay threshold depends on the specific system but it is very often within the order of a few seconds.
- *Scalability*: it reveals the system capabilities to deal with databases of almost arbitrary size, in a reasonable amount of time, maintaining the precision requirement. This can be done by guaranteeing that a large database can be explored in the same time than a smaller one by increasing correspondingly the underlying hardware resources.
- *Flexibility*: the system has to fit easily and efficiently any database size, any database features (such as noisy fingerprints or rollings), as well as any hardware configuration (different architectures, varying cluster size, different processors).

Although there are several solutions to the fingerprint identification problem, the general search process structure is composed of the following steps [2]:

(1) Input fingerprint fetching
(2) Feature extraction
(3) Search of a similar fingerprint in the database
(4) Returning the result

2.2. Feature extraction

A fingerprint is basically formed by ridges and valleys. They can be easily appreciated in a good quality image (Fig. 1a), or on the

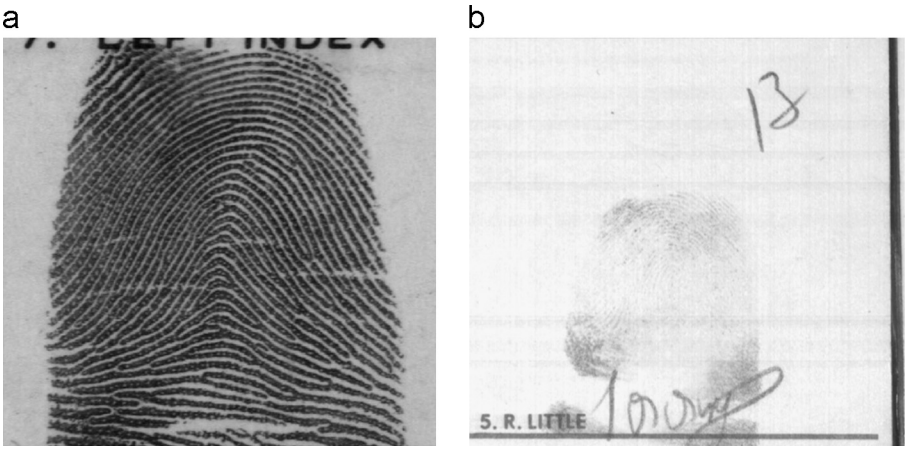

Fig. 1. Good and bad quality images. (a) Good quality image and (b) bad quality image.

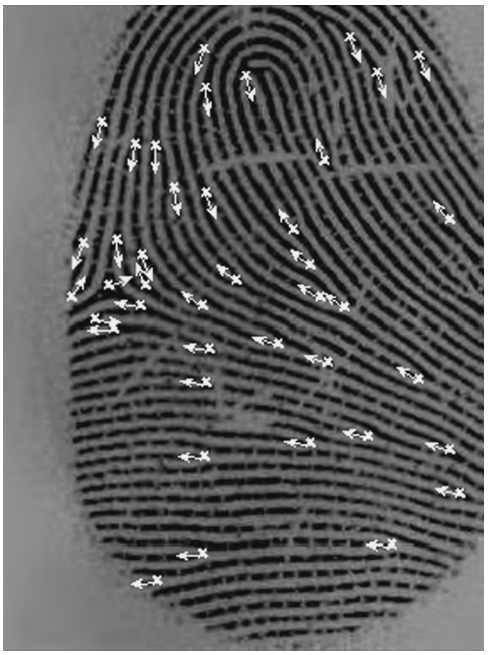

Fig. 2. Fingerprint minutiae with their orientation.

contrary they can be blurred or even indistinguishable (Fig. 1b), difficulting the knowledge extraction process.

As they are analyzed at different degrees theses ridges and valleys present some patterns that can be used to perform the fingerprint comparison. The most relevant features, ordered from the most global to the most local, are the following [2]:

- *Singular points*: they are detected at the most global level. They are points around which the ridge patterns are wrapped. There are two kinds of them: loops and deltas, and a fingerprint can have between zero and five singular points.
- *Orientation map*: it belongs to the same level as singular points and contains the direction of the fingerprint lines for each coordinate in the image.
- *Minutiae*: they are the ridge bifurcations and endings, which are detected at a more detailed level (Fig. 2).

Among these kinds of patterns, minutiae are the most used features for fingerprint recognition [2]. Some studies state that they are the most reliable features for these purposes [25,26], and that twelve perfectly matching minutiae between two fingerprints can ensure that they are the same [27]. However, in bad quality images their extraction can be troublesome [28].

A minutia M_i is typically described with five parameters $(x_i,\ y_i,\ \theta_i,\ t_i,\ q_i)$:

- $(x_i,\ y_i)$: coordinates in the picture
- θ_i: orientation or minutia angle
- t_i: type (ridge ending or bifurcation)
- q_i: quality

Therefore, a fingerprint F with r minutiae can be represented as a minutiae vector $\{M_1, M_2, \ldots, M_r\}$.

The number of minutiae r is typically between 30 and 100. Thus, minutiae can be efficiently stored and easily handled in a computing environment, and fingerprint comparison can be treated as a similarity calculation between minutiae sets.

There are two main types of minutiae extractors [2]:

- *Binarization-based methods*: most of the methods require a binary fingerprint image. The image usually passes through a thinning process that reduces the line thickness to one pixel, resulting in a skeleton image. Although these steps are time-consuming and may cause some information loss, they allow the minutiae detection with a simple image scan and they greatly benefit from previous enhancement processes such as the approaches presented in [29–31]. Some methods of this type are NIGOS *mindtct* [21], and an approach based on peak detection along sections orthogonal to the ridge orientation [32]. Additionally, other methods improve the image quality before the thinning step, for example by using adaptive windows to follow the ridges and find the gaps and holes [33].
- *Direct gray-scale extractors*: some methods do not use binarization or thinning. Therefore, there is no information loss and the time spent on binarization and thinning steps is avoided, but these methods do not benefit from a priori enhancements. One of the most used methods uses the orientation map to follow the ridges [5], and is used as a basis by further proposals [34–38]. Other methods use alternatives to ridge-line tracking, such as neural networks [39] or spatial filtering [40].

2.3. Matching

A matching algorithm compares the features of two fingerprints and returns a similarity score. The algorithm and the data

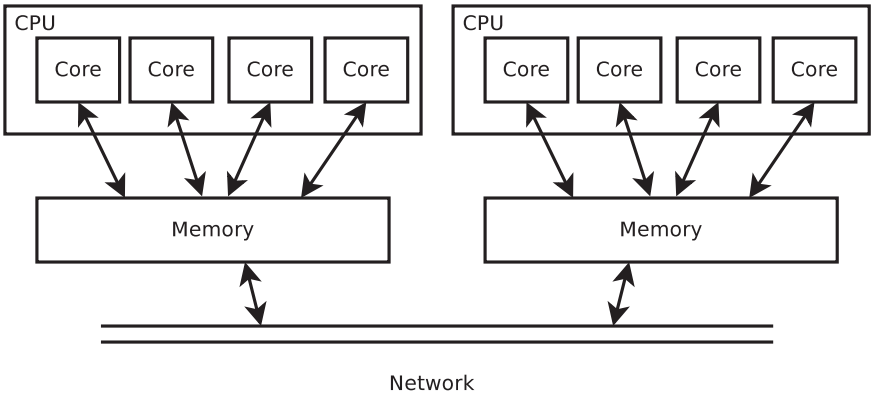

Fig. 3. Typical hybrid architecture of the cluster used for the experiments in Section 6.

structures it uses depend on the specific features that are extracted from the fingerprint image, allowing the following classification of matchers [2]:

- Correlation-based [41,42].
- Minutiae-based [43,28,6,44].
- Non-minutiae feature-based [33,4].

This paper focuses on minutiae-based matchers, whose usual data structures are the following:

- Distance between minutiae.
- Minutiae neighborhood.
- Number of ridges between minutiae (ridge count).

A matching algorithm performs some calculations from these structures and the fingerprint features themselves and returns a score (typically a real number) that describes the similarity level ranging from completely different fingerprints to the totally identical pictures.

The minutiae-based matching process can be performed at three different levels [2,6]:

- *Global*: The minutiae of the whole image are compared. This matching type is more sensitive to image distortions, rotations and translations, although the usage of information of the whole image at the same time provides a complete view of the fingerprint. Some proposals are presented in [45,46].
- *Local*: Small groups of minutiae close to each other are compared. Problems due to rotations and translations are softened because the use of relative angles and coordinates makes the method rotation and translation invariant. The distortion problem is also reduced because close minutiae are less affected by distortions. However, not considering the fingerprint as a whole implies a loss of information that can affect the precision of the algorithm. Some approaches are described in [47,28].
- *Hybrid*: Most reliable algorithms use a hybrid approach, combining both philosophies. First, a local matching extracts the most similar minutiae groups of both fingerprints. These minutiae are considered to be the same, and then a global matching based on this correspondence is executed. Some of the most relevant proposals are [43,6].

3. High performance computing

HPC systems are normally used for distributed and parallel computing, providing several advantages:

- *Efficiency*: the parallel processing in several cores and computers can be used to get results faster.
- *Robustness*: the use of several machines allows the system to be fault-tolerant, because if one machine fails, the rest can assume its work and the system still provides a correct response.
- *Scalability*: hardware evolves towards a higher number of cores and collaborating processors. Thus, an algorithm that is able to solve bigger problems just by using more computers could solve arbitrarily big problems without being modified.

In Sections 3.1 and 3.2, the hardware and software that give support to an HPC system are described. Section 3.3 presents the theoretical expectation of improvement in the execution times of a generic system that uses HPC. Finally, the state-of-the-art about distributed AFISs is studied in Section 3.4.

3.1. Hardware support

Hardware has evolved in two ways to support HPC. On one hand, several computers can be integrated with a high-speed network to form a cluster. This provides a great flexibility when the processing capacity has to be increased, but the performance can become limited by the network speed.

On the other hand, a single computer can have several processors, a single processor can have several cores, and a single core can handle several execution threads (for example, with the Intel Hyperthreading technology [48]). All these processors and cores can communicate using shared memory, which is very fast, as long as the synchronization is efficiently performed. This is not always easy and may imply great design and implementation efforts. However, the number of cores in a single computer is still quite limited, and nowadays is not higher than about 12 or 24.

A typical computing cluster is formed by a bunch of computers, and each one of them has one or several multicore processors, where all the cores share the main memory and some cache. This kind of clusters are called *hybrid* clusters (Fig. 3).

3.2. Software support

According to the current evolution of technology, the parallel paradigm for software development is bound to be increasingly necessary in the next years (and most likely in a longer term too).

Within a hybrid cluster, the computing program is typically divided into several processes and each process is run in a different node. These processes communicate using Message Passing Interface (MPI).[1] Again, each process can be divided into several execution threads that can communicate using shared memory, which is faster than MPI. The maximum performance is

[1] http://www.mpi-forum.org/

usually reached when each computing node executes a single process that contains one or two threads per node core. Thus, the adequate implementation of a system in a computing cluster is a complex task.

The execution of these processes and threads can be tackled by the operating system (for example when using C++), or by a virtual machine (for instance with Java, Scala or Erlang).

3.3. Theoretical expectations

There are several formulas to measure the performance of a parallel system. The most widely used is the speedup ($S = t_s/t_p$), which measures the relation between the execution times of the sequential (t_s) and parallel (t_p) versions of a same calculation.

If a calculation is executed in n processing cores, and a portion f of the calculation is performed in parallel, the maximum attainable speedup would be S^*, according to the Amdahl's Law [49], which is shown in

$$S^* = \frac{1}{(1-f) + \frac{f}{n}} \tag{3}$$

Therefore, if the calculation is fully parallelizable ($f=1$) the maximum speedup would be equal to the number of cores (n). However, in practice the attained speedup is lower than this maximum due to several factors:

- There is always some part of the calculation that is not parallelizable ($1-f$). Even if this part is very small it can represent a very big speedup loss when the number of parallel cores is high, as it can be seen in Eq. (4), which shows the maximum speedup for a certain f even if the number of processors is arbitrarily high. Therefore, it is crucial to reduce as much as possible the fraction of non-parallelizable calculation

$$\lim_{n \to +\infty} \frac{1}{(1-f) + \frac{f}{n}} = \frac{1}{1-f} \tag{4}$$

- A parallel application includes extra communication and synchronization workloads that are not necessary in sequential programs.
- When some threads or processes finish their workload before others, the hardware does not work at full capacity any more because some of the processing cores remain idle, waiting for new tasks to be assigned.

However, there are some cases where a superlinear speedup can be attained. One of them is when the amount of processed information does not fit in the main memory of a single computer. If several computers collaborate, the total amount of available memory is higher and then the necessity of slow hard-disk accesses can be removed.

Finally, the relationship between processing (t_{pr}) and communication (t_c) workload as the problem size increases is also important ($R_{pc} = t_{pr}/t_c$). If the processing workload is higher, a bigger cluster would be useful in order to improve the performance. However, if there is more communication as the problem size increases, there would be a bottleneck and the use of more machines would not imply faster results.

3.4. Distributed AFISs: proposals in the specialized literature

According to the preceding sections, the features provided by HPCs are very similar to the AFIS objectives described in Section 2. HPC is a promising tool for the design of a flexible and scalable AFIS because it would allow a parallel search through the fingerprint database, providing an increased system performance [18,50].

At the time of writing this paper there are several AFISs in the specialized literature and also in the commercial market. Most of these systems have an acceptable performance when they deal with small databases. Nevertheless, in most real world problems there is a need of finding a person among databases whose sizes can range from tens of thousands to tens of millions. These identifications must be performed in a reasonable time, often shorter than a threshold of a few seconds. Furthermore, as explained in Section 2, the larger the size of the fingerprint database, the harder it is to obtain a good identification accuracy.

Within this context, the bottleneck step in the identification process is the matching algorithm, because it must be performed once per each database fingerprint to determine which one is the most similar to the input.

The proposals in the specialized literature can be classified into different categories:

- *Client-server systems*: in [51], the authors propose a server-like AFIS where the fingerprint database is distributed among several servers. When a client requests an identification from a server, it searches the input fingerprint in its database portion. If it succeeds, it sends the response to the client. However, if the fingerprint is not found in the server, the request is forwarded to other servers and the server acts as a client. Therefore, this system does not process the information in parallel. The distribution only affects the database and not the processing, and the overall processing time is higher than in a sequential AFIS. This makes this architecture unsuitable for very large databases with hundreds of thousands of fingerprints. A similar system is described in [19], that additionally includes a GPS-based system for an increased security.
  The objective of these systems is to provide an AFIS for distributed databases, whereas this paper focuses on attaining low identification times in large databases. Thus, no comparison can be performed between these systems.
- *Agent-based systems*: in [52] an agent-based system is presented, mostly oriented to heterogeneous hardware architectures. The novelty of this work is that it uses the idle times of a bunch of computers that are mainly used for other purposes, especially desktop machines. The main part of this proposal is therefore a load-balancing algorithm. The system has a master-slave structure where a set of slave agents compare fingerprints and a master agent distributes and organizes the computing workload. The proposed architecture is divided into layers that isolate the resource monitoring, the agent manager and the matching algorithm. A similar, less centralized approach is presented in [53], where slave agents are able to communicate and share their found scores. Several processes are dynamically created when an input fingerprint is received to better distribute the database exploration. Although this may improve the system flexibility, there is a negative impact on the identification time.
  Again, the objective in these systems is not performance, but load-balancing between shared machines. The execution times shown in [52] are of 3 min and 14 s for performing 700 matchings in a set of 20 Pentium IV machines. This result shows that this approach is not able to handle identifications through hundreds of thousands of fingerprints in no more than a few seconds, as is the requirement for most real-time biometric systems.

To sum up, there are some solutions and ideas to improve the efficiency and the availability of AFISs; however, there is no really scalable AFIS available in the current scientific literature.

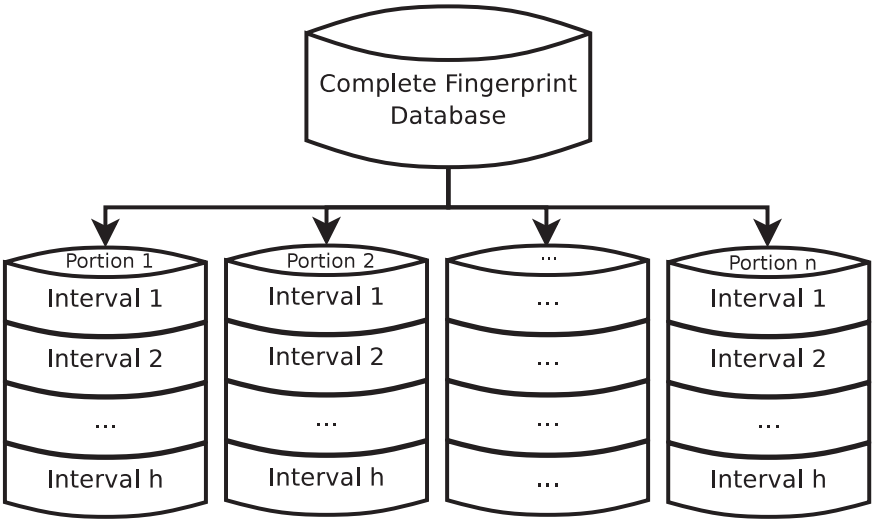

Fig. 4. Database partition for nodes and threads.

4. Distributed and scalable AFIS framework

As it has been explained in the previous sections, the bottleneck in a traditional AFIS is the matching process, that has to be executed once for each fingerprint in the database. This makes the system less usable when it comes to deal with large or very large databases (from tens of thousands of fingerprints onwards) as the response time becomes too high. However, the fingerprint identification problem is naturally parallelizable, because the comparisons of the input fingerprint I with each one of the N fingerprints T_i in the database are entirely independent. This feature can be exploited by designing a flexible and efficient parallel identification system based on the HPC paradigm, which eliminates the bottleneck.

The proposed system is described as follows: Section 4.1 details its parallel structure, Section 4.2 describes the database distribution and Section 4.3 explains the distributed search process.

4.1. Two-level parallelization

As described in Section 3.1, a typical computer cluster has two parallelism levels. Both nodes and cores contribute to the system performance and can execute processes by themselves; however, they must be handled by the software in a different way if a maximum performance must be attained.

The proposed software system (which is implemented in C++) consequently has a two-level parallelization:

- *Processes*: typically one per node, they are handled with MPI (see footnote 1).
- *Threads*: one or several per process, they are handled with OpenMP.[2]

There is a single process (called "master") which reads the input fingerprint and gathers the results at the end of a search. All the other processes are called "slaves", and perform parts of the search executing the matching algorithm. Each slave loads its corresponding fraction of the database and searches the input fingerprint in it. Additionally, each slave process is itself formed by one or more threads, therefore its database fraction can be divided over again and the threads perform a parallel search within each process.

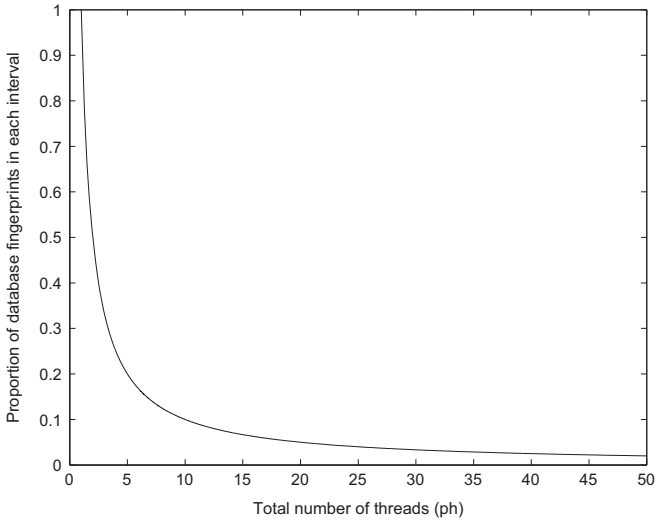

Fig. 5. Relative interval size for each thread.

4.2. Database distribution

Suppose a generic system with N fingerprints, p nodes and h threads per node. The database would be divided into one portion per node, so that each process searches in its corresponding portion of N/p fingerprints (Fig. 4). This distribution can be physical, if the fingerprints are stored in their corresponding nodes in order to improve the access time and avoid the bottlenecks of a centralized database, or merely logical if the database is centralized.

Inside each node, the process performs a logical partition of its database portion. Hence, each thread searches through only $N/(ph)$ fingerprints. This scheme allows N, p and h to be modified in a totally flexible way, so they can be adjusted to any hardware (from single-core computers to hybrid clusters), any environment conditions and any database to obtain a maximum performance gain.

Fig. 5 represents the interval size for each thread as a function of the total number of threads ph.

[2] http://openmp.org/wp/

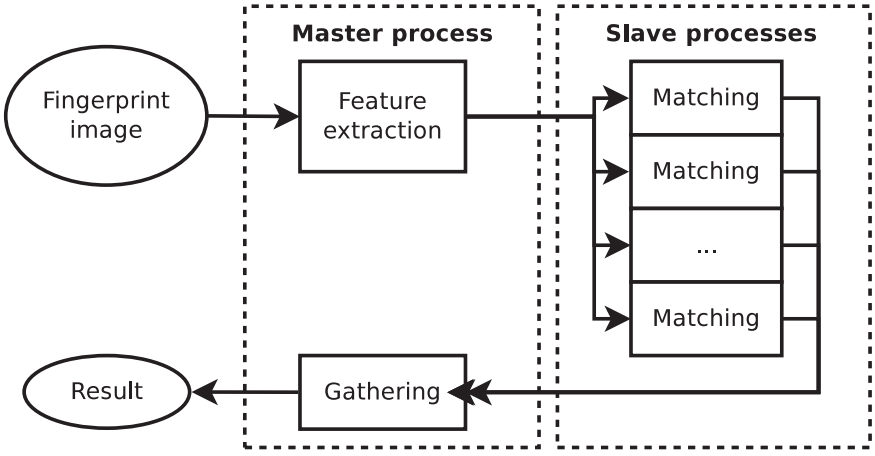

Fig. 6. Processing in the proposed distributed model.

4.3. Distributed search process

The logic of the system remains the same independent of how many nodes or threads are used, and is depicted in Fig.6.

(1) *Initialization*: this step is executed only once, and then the system can perform as many identifications as required. The database partitions are established, each slave loads its part of the database, preprocesses it if necessary, and the master waits for input fingerprints.
(2) *Identification loop*:
   (a) The master receives an input fingerprint. As the feature extraction is performed only once, it can be computed either in the master process or in each of the slaves independently, depending on the system features.
   (b) When a slave gets the fingerprint features, each one of its threads performs N/ph matchings to compare it with the template fingerprints in its database portion.
   (c) Each slave sends its successful matches to the master process.
   (d) The master computes the results and gives a response to the user.

This scheme ensures that the bottleneck step (number 2.b) is executed with two levels of parallelism, in order to accelerate the execution as much as possible and eliminate the bottleneck. Moreover, as the matching algorithm remains the same as in a sequential search, there is no loss of precision and the system is guaranteed to find exactly the same solution as the sequential model in much less time. This also makes the system independent of the matching algorithm, which can be easily replaced.

According with these data and Amdahl's Law (Section 3.3), this system can obtain a maximum speedup of $1/ph$. In that case the identification time plot would have the same hyperbolic shape as the partition size in Fig. 5.

The proposed distributed system shows several important advantages:

- Very high speedup because of several factors:
  - Independent processing among slave processes.
  - Independent processing among threads within each slave.
  - Minimum communication overhead.
  - Optimal exploitation of the hardware structure.
- Adaptability to multiple sequential or distributed platforms and architectures.
- Flexibility for centralized or distributed databases.
- Flexibility for any matcher or feature extractor.
- Same precision as the sequential model.

5. Experimental setup

This section describes the experimental framework for this paper. The aim of this experimental study is to check the system scalability — along with its adaptability to the underlying hardware system — in several aspects:

- The number of computing nodes.
- The number of threads in each node.
- The size of the database.

All the performed experiments have the same structure: first, a fingerprint database is loaded in the system and all its fingerprints are preprocessed according to the corresponding matching algorithm; then, a set of input fingerprints are searched throughout the database for their identification. All the presented results are averages of the identification times obtained for 1000 input fingerprints. For the sake of readability and reasons of space, the standard deviations are not included, but they can be found in the web site associated with this paper. The penetration rate is 100% for this setup, as there is no stopping criterion for the search. As explained in Section 4.3, the proposed system provides the same identifications as a traditional sequential AFIS. A large description of the possible stopping criterion is presented in the associated web site, as well as the precision results for all the databases used in this paper.

Firstly, the hardware and software support are defined and detailed in Section 5.1. Then, Section 5.2 describes the large synthetic databases created with SFinGe. Finally, Section 5.3 details the captured databases that are used in the experiments.

5.1. Hardware and software environment

The experiments have been carried out on up to twelve nodes in a cluster. Each of these nodes has the following features:

- *Processors*: 2 × Intel Xeon CPU E5-2620
- *Cores*: 6 per processor (12 threads)
- *Clock Speed*: 2.00 GHz
- *Cache*: 15 MB
- *Network*: Gigabit Ethernet (1 Gbps)
- *RAM*: 64 GB

Table 1
Parameters for the methods used in the experimentation.

Algorithm	Parameters	Reference
Mindtct	Output format = ANSI INCITS 378-2004 Image enhancement = enabled	[21]
Jiang	$w_d = 1,\ w_\theta = 54\pi,\ w_\phi = 54\pi,\ w_n = 0,\ w_t = 0$ Consolidation step iterations = 5, minutiae neighborhood size = 2 $BG_1 = 8,\ BG_2 = \frac{\pi}{6},\ BG_3 = \frac{\pi}{6}$	[43]
Chen	$Thr_L = 55,\ Thr_H = 80,\ R = 80,\ RS = 100,\ \theta_L = 0.25,\ \theta_H = 0.4$ $\text{len}_L = 5,\ \text{len}_H = 20,\ Thr_{topo} = 0.7$	[28]
MCC16	$R = 70,\ N_s = 16,\ N_d = 6,\ \sigma_s = \frac{28}{3},\ \sigma_d = \frac{2\pi}{9},\ \mu_\Psi = 0.01,\ \tau_\Psi = 400$ $\omega = 50,\ \min_{VC} = 0.75,\ \min_M = 2,\ \min_{ME} = 0.60,\ \sigma_\theta = \frac{\pi}{2},\ \max_{n_p} = 12$ Floating-point-based version: enabled, $\mu_P = 20$ $w_R = 0.5,\ \mu_1^\rho = 5,\ \tau_P = 0.6,\ \min_{n_p} = 4$ $\mu_2^\rho = \frac{\pi}{12},\ \mu_3^\rho = \frac{\pi}{12},\ \tau_1^\rho = -1.6,\ \tau_2^\rho = -30,\ \tau_3^\rho = -30,\ n_{rel} = 5$	[6]

One of the nodes acts as the interface with the user and hosts the master process. However, as this process does not perform any major processing tasks, a slave process can also be executed on the same node without compromising the performance and thus the hardware is more efficiently exploited.

The proposed distributed model has been implemented in C++, using the OpenMPI 1.6 library[3] for the communication and synchronization of processes. Similarly, the OpenMP library (see footnote 2) has been used for handling the threads within each process. In all databases where the fingerprint features had to be extracted, the NIGOS *mindtct* [21] algorithm was used.

Three different matching algorithms of the state-of-the-art literature have been used within the framework:

- *Jiang* is a classical hybrid matching algorithm [43]. Each minutia is described with a feature vector that depends on its neighboring minutiae, and the feature vectors of both fingerprints are compared in pairs. The algorithm assumes that the most similar pair corresponds to the same minutia in both fingerprints and compares the rest of the minutiae using relative coordinates and angles (avoiding the translation and rotation problems).
- *Chen* focuses on getting robustness despite of the fingerprint distortion [28]. The algorithm is mostly local, as it calculates the local topology for each minutiae given a fixed radius. Then, it compares the local topologies of both fingerprints, and if they are similar enough, it repeats the comparison with a modified radius to avoid problems with the image distortion.
- *Minutia-Cylinder-Code (MCC)* uses both local and global information to perform the matching [6]. For each minutia, a tridimensional cylinder is built and discretized in cells. Each cell is given a value that depends on its position and the relative position of neighboring minutiae. According to this number, the cell can be declared either valid or invalid, so that only cylinders with a minimum number of valid cells are taken into account for the matching process. This process compares the cylinders of both fingerprints cell by cell and merges the results (global matching) to get the score.
  This algorithm has a binary and a real version. In this paper, we focus on the latter, which is more precise and more suited for general purpose machines. We also fix 16 cells as the cylinder side size in order to get the most accurate configuration, which is also the most computationally complex. Results for the version with $Ns = 8$ are included in the web site associated to this paper.

[3] http://www.open-mpi.org/

Table 2
Parameter specification used with the SFinGe tool.

Scanner parameters
Acquisition area: 0.58" × 0.77" (14.6 mm × 19.6 mm)
Resolution: 500 dpi
Image size: 288 × 384
Background type: optical
Background noise: default
Crop borders: 0 × 0
Generation parameters
Impression per finger: 25
Class distribution: natural
Set all distributions as: "varying quality and perturbations"
Generate pores: enabled
Save ISO templates: enabled
Output settings
Output file type: WSQ

All three algorithms have been implemented by the authors of this paper, with the only help of the information shown at each of the referred papers. All the used methods parameters are common for all databases, and they were selected according to the recommendation of the corresponding authors (Table 1).

5.2. SFinGe databases: ground-truth minutiae and NIGOS mindtct extraction

A correct scalability study requires a very large database. However, there is no public captured fingerprint database big enough to cover this need, so we used SFinGe [20,2] to generate a database with 400 000 synthetic fingerprints, using the parameters described in Table 2 to ensure the generation of realistic fingerprints.

SFinGe randomly generates the fingerprint minutiae and calculate a fingerprint image from them, following patterns so that the resulting synthetic fingerprints behave as natural captures. As SFinGe is able to provide the generated minutiae as an additional output, this paper has used both the returned ground-truth minutiae and the extracted minutiae (using *mindtct*), obtaining two databases with the same fingerprints but slightly different characteristics.

For each fingerprint 25 impressions have been generated. One of the impressions is selected as template, and the rest are considered input fingerprints. Then, several subsets of the whole database (each of them of increasing sizes) have been selected, respecting the natural class distribution, in such a way that each

Table 3
SFinGe databases size and average number of minutiae.

DB size	Ground-truth		Extracted	
	Template	Input	Template	Input
1000	40.79	36.84	55.35	49.60
2000	40.84	36.81	55.47	49.61
5000	40.97	36.98	55.64	49.87
10 000	40.79	36.77	55.48	49.61
50 000	40.72	36.70	55.44	49.58
100 000	40.73	36.71	55.46	49.62
200 000	40.74	36.71	55.50	49.63
400 000	40.70	36.68	55.47	49.66

database contains the immediately smaller one. The whole enrollment process is described in the associated web site.

The size of all database subsets are presented in Table 3, along with the average number of minutiae for both template and input fingerprints. As it can be seen, the number of average minutiae is higher for the extracted feature vectors due to the noise introduced by the image generation and the processing steps. This implies that *mindtct* extracts an average of 15 spurious minutiae per fingerprint.

Finally, we have selected one random input impression for each fingerprint in the smallest database, obtaining a test set of 1000 input fingerprints that is valid for the experiments with all the generated databases.

The execution parameters take the following values for the experiments with these database subsets, producing a total of 480 executions of 1000 identifications each:

- *Number of nodes*: 1, 2, 4, 8 and 12
- *Number of threads per node*: 1, 4, 12 and 24
- *Matchers*: Jiang, Chen and MCC16

For the extracted minutiae databases not all parameter combinations are necessary, and the experiments are limited to the sequential and fully parallel cases to compare if the system behavior when using the extracted minutiae is the same as when using the ground-truth database. Considering the 3 matching algorithms and the 8 database sizes, this produces a total of 48 experiments. It is important to note that the increase in the number of minutiae shown in Table 3 implies a slow down in the identification process.

5.3. NIST DB4 and DB14 databases

In addition to the above mentioned experiments, NIST DB4 and DB14 databases have also been used. These databases are provided by the National Institute of Standards and Technology (NIST). They contain 2000 and 27 000 rolled fingerprint pairs, respectively, and thus the number of minutiae extracted by NIGOS *mindtct* is very high (135.87 in DB4 and 206.90 in DB14). The aim of using DB4 and DB14 is to test if the proposed system can deal with captured databases, and also with rolled fingerprints. The used parameters for the minutiae extraction and the matching algorithms are the same as for the SFinGe extracted database.

6. Experimental study

This section describes the results of the performed experiments. Sections 6.1, 6.2, and 6.3 present the results for SFinGe ground-truth, SFinGe extracted and NIST databases, respectively.

6.1. Speedup with SFinGe ground-truth minutiae

The obtained results are presented in Tables 4 and 5. Note that the experiments with the 400 000 fingerprints database combined with the MCC16 algorithm could not be performed within a single machine because the preprocessed database is bigger than the whole RAM space in the used computers (64 GB), and thus no speedups can be calculated. The sequential tests could be run using virtual memory, but the performance loss would be dramatic and the speedup when using several machines would be superlinear, as described in Section 3.3. This is a very clear case of a problem that cannot be solved in a sequential manner, but can be successfully tackled using a distributed approach.

In the rest of the results, the decrease in the execution time and the corresponding increase in the speedup as the amounts of threads and processes are augmented can be seen. It is also clear that the speedup is generally almost linear with the total number of threads that perform the distributed search.

However, there are some exceptions to this statement:

- When the number of threads per computing node is 24, the performance gain is not proportional to this number of threads, but lower. Nevertheless, this behavior is normal because each node in the used cluster only has 12 cores. The Intel Hyper-threading technology is able to handle two threads in each core, but its performance is not as high as when these threads are executed in parallel within different cores, and strongly depends on the specific instructions executed by the threads.
- When the database size is small and the computing resources are high, the performance is not optimal. This behavior is especially clear with the Jiang algorithm, which is the fastest method tested in this paper. For example, the execution time for 10 000 fingerprints is lower than for 1000 fingerprints when the full cluster (12 nodes and 24 threads) is used. This is due to the ratio between processing and communication times (R_{pc}), as explained in Section 3.3. With small databases and big resources, the processing time t_{pr} is much smaller than the communication time t_c and thus no gain is obtained. Furthermore, t_c is increased due to the synchronization between threads and processes. The response time of the overall system depends on the response time of the slowest thread; when the database is small, the chunks assigned to each thread are very small and the difference are high. This problem disappears as the database grows, and it explains why bigger databases can be explored faster than smaller ones when large resources are employed.
  Since we are mainly concerned with large databases, this is no issue. Anyway, it can be easily solved by using a specific configuration for databases with a size smaller than a given threshold.

Table 4
Execution times and speedup with the MCC16 algorithm.

Slaves	*DBsize*	1 Thread *Time* (*s*)	*Speedup*	4 Threads *Time* (*s*)	Speedup	12 Threads *Time* (*s*)	*Speedup*	24 Threads *Time* (*s*)	*Speedup*
1	1000	6.5377	1.0000	1.6867	3.8759	0.6444	10.1447	0.5043	12.9646
	2000	12.9968	1.0000	3.2963	3.9429	1.2041	10.7936	0.9059	14.3463
	5000	32.5242	1.0000	8.1634	3.9841	2.9056	11.1937	2.1228	15.3212
	10 000	64.7076	1.0000	16.2010	3.9940	5.7484	11.2566	4.1308	15.6645
	50 000	321.8924	1.0000	81.9340	3.9287	28.2669	11.3876	20.1745	15.9554
	100 000	624.3297	1.0000	160.9759	3.8784	56.5323	11.0438	40.2844	15.4980
	200 000	1250.2594	1.0000	314.9397	3.9698	113.1789	11.0468	80.4265	15.5454
	400 000	–	–	–	–	–	–	–	–
2	1000	3.3728	1.9384	0.9062	7.2142	0.3633	17.9976	0.3081	21.2203
	2000	6.5398	1.9873	1.6969	7.6592	0.6457	20.1289	0.5144	25.2667
	5000	16.3442	1.9900	4.1567	7.8246	1.4826	21.9367	1.1156	29.1537
	10 000	32.5416	1.9885	8.1786	7.9118	2.9220	22.1446	2.1294	30.3882
	50 000	161.8845	1.9884	40.3953	7.9686	14.1720	22.7132	10.1626	31.6743
	100 000	324.4350	1.9244	80.6912	7.7373	28.2606	22.0919	20.1649	30.9612
	200 000	625.5980	1.9985	161.2985	7.7512	56.5520	22.1081	40.2440	31.0669
	400 000	1248.7902	–	322.3448	–	109.9070	–	80.4963	–
4	1000	1.8610	3.5129	0.5247	12.4603	0.2329	28.0741	0.2133	30.6491
	2000	3.3892	3.8347	0.9121	14.2499	0.3680	35.3140	0.3172	40.9691
	5000	8.2977	3.9197	2.1465	15.1521	0.7898	41.1816	0.6155	52.8417
	10 000	16.3510	3.9574	4.1657	15.5334	1.4917	43.3775	1.1096	58.3179
	50 000	80.8244	3.9826	20.3091	15.8497	7.1278	45.1600	5.1219	62.8463
	100 000	161.5434	3.8648	40.4924	15.4185	14.2047	43.9523	10.1475	61.5253
	200 000	312.9654	3.9949	80.7448	15.4841	28.2520	44.2538	20.1662	61.9977
	400 000	621.7593	–	158.2815	–	56.4944	–	40.2569	–
8	1000	0.9849	6.6382	0.3098	21.1005	0.1681	38.8978	0.1750	37.3486
	2000	1.8604	6.9859	0.5302	24.5143	0.2335	55.6547	0.2195	59.2200
	5000	4.2340	7.6816	1.1434	28.4443	0.4392	74.0569	0.3691	88.1285
	10 000	8.3241	7.7735	2.1611	29.9425	0.7880	82.1115	0.5932	109.0777
	50 000	40.4281	7.9621	10.2904	31.2810	3.5670	90.2412	2.5289	127.2860
	100 000	80.9415	7.7133	20.2252	30.8689	7.0044	89.1343	4.9369	126.4614
	200 000	161.5014	7.7415	40.3357	30.9963	14.1784	88.1803	10.1298	123.4243
	400 000	315.6620	–	80.5549	–	28.2313	–	20.1449	–
12	1000	0.6955	9.4000	0.2366	27.6279	0.1384	47.2470	0.1616	40.4603
	2000	1.2825	10.1340	0.3841	33.8357	0.1885	68.9394	0.1909	68.0741
	5000	2.9076	11.1858	0.8120	40.0537	0.3320	97.9537	0.2866	113.4908
	10 000	5.5884	11.5789	1.4935	43.3259	0.5632	114.8979	0.4399	147.0949
	50 000	27.1921	11.8377	7.0135	45.8961	2.4538	131.1832	1.7928	179.5466
	100 000	54.1348	11.5329	13.5321	46.1370	4.8029	129.9911	3.4581	180.5417
	200 000	104.8093	11.9289	26.9958	46.3132	9.4815	131.8623	6.7807	184.3857
	400 000	213.1696	–	51.5151	–	18.8523	–	13.4805	–

Additionally, Fig. 7 shows the speedup when the number of threads and processes is varied, with one line per database size and number of slaves. For the sake of readability and to avoid the irregular behavior described in the preceding paragraph, only databases from 10 000 fingerprints onwards have been drawn in these plots.

These figures clearly state that when there are more threads than physical cores the speedup does not increase linearly. It can also be seen that the lines corresponding to different database sizes are grouped. Thus, the database size does not affect the speedup when it is reasonably large. This result, along with the fact that the flexible proposed system allows the identification in arbitrarily large databases, ensures full scalability regarding the database size. In other words, if the database size is doubled, the identification time can be kept constant by simply doubling the computing resources.

Fig. 8 shows the speedup as a function of the total number of threads that are performing the parallel search. The theoretical limit imposed by the Amdahl's Law is also displayed, and it can be seen that the results are close to the line. Moreover, the larger the database size is, the closer to the line the results are, showing that the system scalability increases along with the database size. This is the best possible situation, as it ensures a maximum scalability and performance when it is most needed. As before, there are exceptions where the speedup is much lower than the limit, when the database size is small and when the number of used threads is higher than the number of physical computing cores. The right side of the plots in Fig. 8 shows this behavior very clearly: then the full cluster is used, the speedup increases when the database grows.

6.2. SFinGe databases: extracted minutiae

Once the speedup behavior when changing the computing resources has been studied, more experiments have been performed in order to validate the results with more realistic databases. For this purpose, we have compared the sequential execution times with the times obtained when using the best configuration (12 nodes and 24 threads per node). Table 6 presents the sequential and parallel times, along with their quotient (speedup).

It becomes clear that the good speedups obtained with the ground-truth database are also obtained with extracted minutiae. This is due to the flexibility of the database partitioning scheme, which distributes the database statically among nodes and dynamically among threads. Another fact that can be seen in the table is that the execution times are higher than with the ground-truth minutiae. This is a consequence of the higher number of obtained minutiae when using NIGOS *mindtct*.

Table 5
Execution times and speedup with Jiang and Chen algorithms.

Slaves	*DBsize*	Jiang 1 Thread *Time* (*s*)	*Speedup*	4 Threads *Time* (*s*)	Speedup	12 Threads *Time* (*s*)	*Speedup*	24 Threads *Time* (*s*)	*Speedup*	Chen 1 Thread *Time* (*s*)	*Speedup*	4 Threads *Time* (*s*)	Speedup	12 Threads *Time* (*s*)	*Speedup*	24 Threads *Time* (*s*)	*Speedup*
1	1000	0.2223	1.0000	0.0572	3.8865	0.0210	10.6091	0.0322	6.9010	2.6166	1.0000	0.6707	3.9014	0.2631	9.9439	0.2158	12.1275
	2000	0.4491	1.0000	0.1143	3.9292	0.0418	10.7486	0.0469	9.5809	5.2484	1.0000	1.3442	3.9046	0.5200	10.0937	0.4094	12.8186
	5000	1.1262	1.0000	0.2871	3.9232	0.1048	10.7434	0.0898	12.5358	13.2122	1.0000	3.3910	3.8963	1.3043	10.1294	1.0083	13.1038
	10 000	2.2432	1.0000	0.5710	3.9286	0.2072	10.8245	0.1670	13.4285	26.3182	1.0000	6.7189	3.9170	2.6000	10.1224	1.9823	13.2764
	50 000	11.2532	1.0000	2.8536	3.9434	1.0443	10.7753	0.7408	15.1909	130.0427	1.0000	33.3673	3.8973	12.9129	10.0708	9.7984	13.2719
	100 000	22.3477	1.0000	5.6989	3.9214	2.0743	10.7737	1.4444	15.4715	261.2312	1.0000	66.7052	3.9162	25.8390	10.1099	19.5891	13.3355
	200 000	44.7627	1.0000	11.4571	3.9070	4.1789	10.7115	2.8860	15.5104	520.8561	1.0000	133.8296	3.8919	51.7934	10.0564	39.2616	13.2663
	400 000	89.4130	1.0000	22.8261	3.9171	8.3137	10.7549	5.7485	15.5541	1041.2164	1.0000	268.2014	3.8822	103.6681	10.0437	78.6457	13.2393
2	1000	0.1142	1.9461	0.0294	7.5499	0.0108	20.6412	0.0292	7.6174	1.3740	1.9044	0.3538	7.3956	0.1360	19.2343	0.1180	22.1809
	2000	0.2233	2.0118	0.0575	7.8125	0.0213	21.1162	0.0364	12.3264	2.6151	2.0069	0.6745	7.7813	0.2643	19.8543	0.2176	24.1182
	5000	0.5647	1.9942	0.1455	7.7383	0.0522	21.5809	0.0579	19.4507	6.6614	1.9834	1.7315	7.6305	0.6522	20.2588	0.5073	26.0449
	10 000	1.1266	1.9911	0.2917	7.6913	0.1041	21.5529	0.0940	23.8630	13.2512	1.9861	3.4134	7.7102	1.3075	20.1283	1.0074	26.1243
	50 000	5.5878	2.0139	1.4407	7.8108	0.5222	21.5507	0.3782	29.7538	65.3116	1.9911	16.8256	7.7289	6.4966	20.0171	4.9242	26.4089
	100 000	11.2132	1.9930	2.8599	7.8141	1.0442	21.4017	0.7306	30.5885	130.0469	2.0087	33.5380	7.7891	12.9157	20.2258	9.7965	26.6656
	200 000	22.3400	2.0037	5.7152	7.8323	2.0770	21.5513	1.4477	30.9198	263.4494	1.9771	67.1486	7.7568	25.8201	20.1725	19.5915	26.5858
	400 000	44.7918	1.9962	11.4334	7.8203	4.1682	21.4514	2.8645	31.2138	494.2013	2.1069	134.0177	7.7692	51.7896	20.1047	39.2861	26.5034
4	1000	0.0636	3.4950	0.0160	13.8813	0.0060	36.8029	0.0237	9.3672	0.7832	3.3409	0.2000	13.0831	0.0773	33.8613	0.0773	33.8581
	2000	0.1154	3.8922	0.0297	15.1395	0.0111	40.4738	0.0294	15.2556	1.3838	3.7928	0.3590	14.6187	0.1362	38.5442	0.1155	45.4296
	5000	0.2848	3.9547	0.0731	15.4160	0.0268	42.0524	0.0382	29.4869	3.3920	3.8951	0.8811	14.9944	0.3295	48.0000	0.2660	49.6665
	10 000	0.5649	3.9707	0.1456	15.4113	0.0526	42.6104	0.0582	38.5593	6.6707	3.9454	1.7267	15.2417	0.6550	40.1810	0.5116	51.4412
	50 000	2.8063	4.0099	0.7184	15.6639	0.2631	42.7751	0.1996	56.3777	33.6899	3.8600	8.4567	15.3775	3.2448	40.0772	2.4605	52.8523
	100 000	5.5953	3.9940	1.4330	15.5947	0.5248	42.5867	0.3764	59.3691	65.3531	3.9972	16.8876	15.4688	6.4745	40.3479	4.9155	53.1444
	200 000	11.2069	3.9942	2.8843	15.5196	1.0416	42.9750	0.7293	61.3813	130.2397	3.9992	33.7397	15.4375	12.9011	40.3730	9.7735	53.2930
	400 000	22.3816	3.9949	5.7571	15.5309	2.0919	42.7426	1.4346	62.3251	244.8766	4.2520	67.2316	15.4870	25.8130	40.3369	19.5542	53.2478
8	1000	0.0316	7.0401	0.0083	26.7494	0.0035	64.2974	0.0290	7.6733	0.4037	6.4812	0.1054	24.8257	0.0428	61.1117	0.0515	50.8454
	2000	0.0620	7.2497	0.0161	27.9362	0.0062	72.6729	0.0212	21.1652	0.7753	6.7691	0.2062	25.4555	0.0775	67.7110	0.0758	69.2445
	5000	0.1451	7.7641	0.0378	29.7743	0.0137	82.4991	0.0132	85.4633	1.7007	7.7686	0.4545	29.0717	0.1694	77.9951	0.1482	89.1464
	10 000	0.2842	7.8943	0.0736	30.4730	0.0269	83.4649	0.0189	118.5300	3.3823	7.7811	0.9019	29.1795	0.3319	79.2880	0.2626	100.2375
	50 000	1.3982	8.0483	0.3657	30.7747	0.1338	84.0739	0.0865	130.1263	16.2990	7.9786	4.3068	30.1949	1.6056	80.9917	1.2014	108.2427
	100 000	2.8018	7.9761	0.7160	31.2100	0.2617	85.4097	0.1676	133.3246	32.6356	8.0045	8.3803	31.1721	3.2778	79.6972	2.3699	110.2273
	200 000	5.5918	8.0050	1.4278	31.3501	0.5252	85.2298	0.3771	118.6958	65.4597	7.9569	16.7699	31.0590	6.4707	80.4948	4.9169	105.9329
	400 000	11.2094	7.9766	2.8535	31.3340	1.0465	85.4367	0.7311	122.2957	130.3891	7.9855	33.5594	31.0261	12.9884	80.1652	9.8312	105.9092
12	1000	0.0217	10.2596	0.0059	37.8051	0.0026	84.7512	0.0355	6.2588	0.2860	9.1492	0.0750	34.8743	0.0319	82.0586	0.0462	56.6480
	2000	0.0420	10.7056	0.0111	40.5654	0.0045	100.4881	0.0305	14.7041	0.5355	9.8015	0.1402	37.4315	0.0555	94.5845	0.0605	86.7839
	5000	0.0987	11.4113	0.0257	43.7860	0.0099	113.8505	0.0245	46.0084	1.1946	11.0598	0.3207	41.1937	0.1207	109.4223	0.1097	120.3981
	10 000	0.1913	11.7289	0.0496	45.2401	0.0180	124.5452	0.0294	76.2192	2.2545	11.6735	0.6144	42.8372	0.2249	117.0009	0.1826	144.0929
	50 000	0.9391	11.9824	0.2479	45.3899	0.0886	127.0723	0.0769	146.2598	11.6401	11.1720	2.9209	44.5208	1.0921	119.0792	0.8393	154.9394
	100 000	1.8793	11.8913	0.4780	46.7535	0.1757	127.1641	0.1298	172.2133	21.8480	11.9568	5.6096	46.5684	2.1689	120.4465	1.6707	156.3570
	200 000	3.7270	12.0104	0.9538	46.9313	0.3514	127.3876	0.2573	173.9799	43.5101	11.9709	11.1898	46.5475	4.3222	120.5079	3.2824	158.6822
	400 000	7.4420	12.0146	1.9056	46.9214	0.7015	127.4627	0.4922	181.6637	83.1721	12.5188	22.3146	46.6609	8.6148	120.8634	6.5342	159.3485

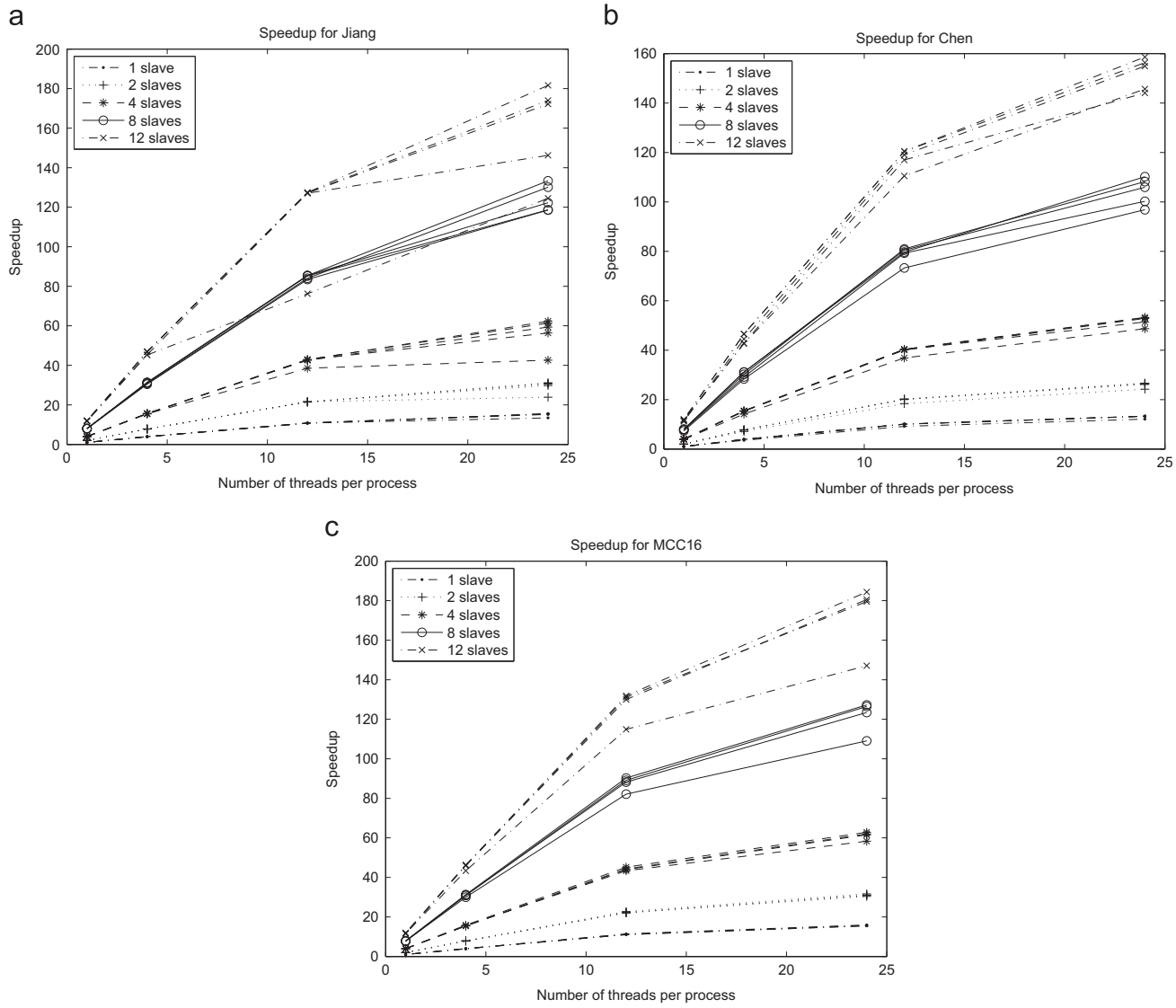

Fig. 7. Speedup when varying the number of slave processes and threads, for databases with 10 000 or more fingerprints. (a) Jiang, (b) Chen, and (c) MCC16.

Fig. 9 shows the obtained speedups depending on the database size (note the logarithmic scale on the horizontal axis). It can be seen that the larger the database, the higher the speedup. It is due to the same reasons explained in the preceding section. Thus, the system behavior remains the same even when the database is changed. The plot also shows how the obtained speedup when using two threads per core is far from the theoretical maximum with the Intel Hyperthreading technology, but it is also considerably higher than the maximum of 144 (12 nodes with 12 cores) that would be attainable if this technology were not implemented in the microprocessors. This proves again that we are in an optimal case for the application of a distributed system and that the proposed system has been optimally designed and implemented.

6.3. NIST DB4 and DB14 databases

Finally, the NIST DB4 and DB14 databases have been used to test the proposed system in the same conditions as the SFinGe extracted fingerprints. The results are presented in Table 6 and Fig. 10.

Again, the speedup values are similar to those obtained with the SFinGe ground-truth database, proving that the proposed system is database-independent and can achieve very good results both with plain and rolled fingerprints, whose matching times are totally different.

The plot also shows that the speedups are higher when the search is performed in DB14, which is by far the biggest database. This result is in the same line as those obtained with the SFinGe databases, where bigger databases allow better scalability. If the figure is compared with Fig. 9, it can be seen that both NIST databases reach a better speedup than SFinGe databases of the same size. This is due to the higher number of minutiae of the rolled fingerprints: the matching process is more computationally complex, and thus HPC is able to improve the time results even further because the impact of the sequential preprocessing is reduced.

The different matching algorithms show the same behavior with the NIST and SFinGe databases, as it can be seen when comparing Figs. 9 and 10: the Jiang algorithm has less speedup because its processing workload is very small, and thus the communication time has a bigger impact on the overall time. On the other extreme, the most computationally expensive algorithm (MCC) obtains a superlinear speedup when inserted in the proposed framework, although in theory this situation should not be possible. In this case, as mentioned in Section 3.3, a higher

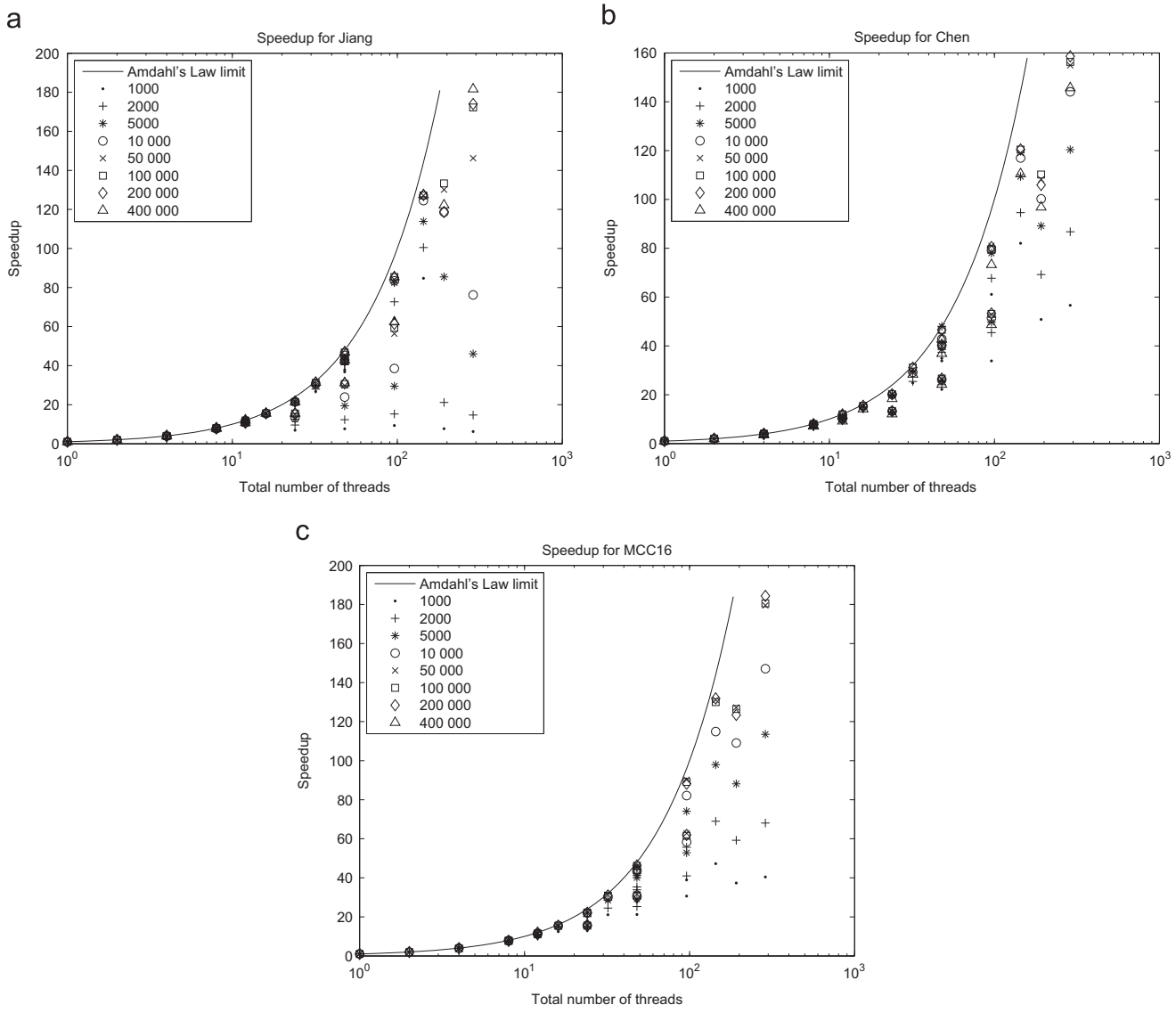

Fig. 8. Speedup when varying the database size. (a) Jiang, (b) Chen, and (c) MCC16.

Table 6
Sequential (t_s) and parallel (t_p) execution times and speedup with the SFinGe extracted and NIST databases.

DB size	**Jiang**			**Chen**			**MCC16**		
	t_s (*s*)	t_p (*s*)	***Speedup***	t_s (*s*)	t_p (*s*)	***Speedup***	t_s (*s*)	t_p (*s*)	***Speedup***
1000	0.4735	0.0253	18.7070	5.9325	0.0661	89.6952	10.4988	0.2023	51.9038
2000	0.9330	0.0255	36.5825	11.4148	0.1004	113.6886	19.7435	0.2519	78.3921
5000	2.3099	0.0325	71.0363	28.6567	0.2061	139.0185	47.5580	0.3912	121.5540
10 000	4.6722	0.0446	104.8213	56.6954	0.3477	163.0649	93.2347	0.6227	149.7332
50 000	22.9674	0.1475	155.6925	283.3022	1.6784	168.7886	459.9514	2.5605	179.6316
100 000	46.3680	0.2734	169.6040	569.8500	3.3252	171.3708	922.1267	4.9613	185.8622
200 000	92.3980	0.5288	174.7348	1144.3038	6.6053	173.2413	1851.2036	9.7530	189.8089
400 000	183.2521	1.0368	176.7501	2303.3817	13.2144	174.3086	–	19.3620	–
DB4	7.7601	0.0795	97.5911	81.7127	0.5893	138.6534	192.5607	1.6131	119.3743
DB14	307.4337	2.0310	151.3740	2454.5287	10.4387	235.1381	6460.3050	20.6486	312.8684

number of computers also means more main memory and more caches. As the database chunks explored by each node are also smaller, they can fit more easily in the cache memory and thus can be explored even faster.

7. Conclusions

In this paper, we have introduced a novel two-level parallelized automatic fingerprint identification system. The proposed framework

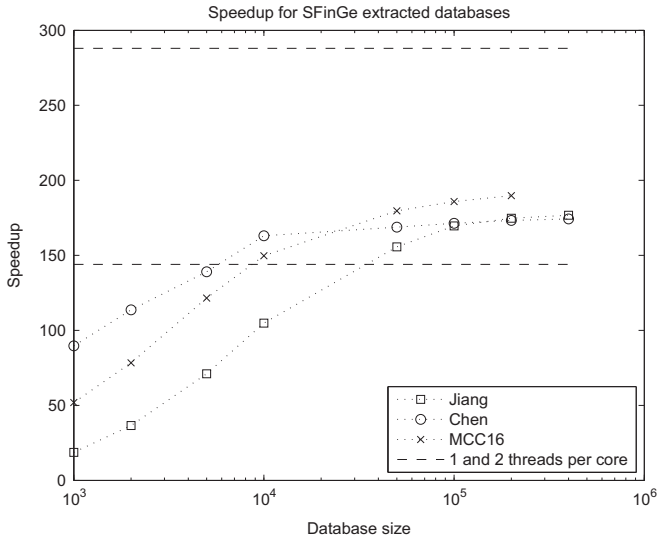

Fig. 9. Speedup with the SFinGe extracted database. The dashed lines represent theoretical maximum speedups considering one or two threads per core.

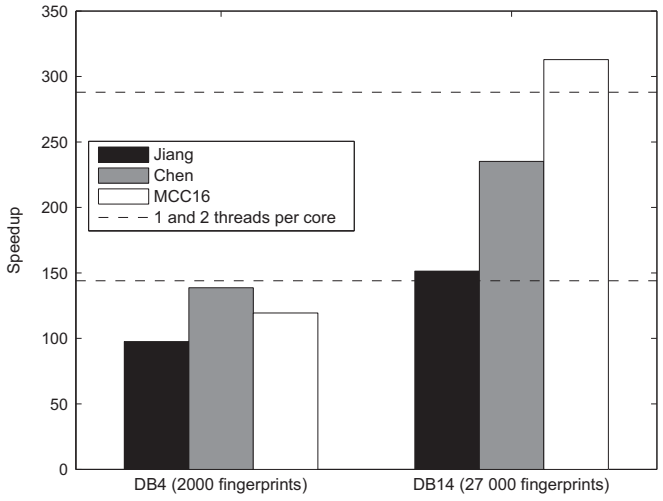

Fig. 10. Speedup with the NIST databases.

combines process-level and thread-level parallelism in order to obtain a maximum speedup for any kind of underlying hardware architecture from monocore processors to large hybrid clusters. It also abstracts the fingerprint matching algorithm, in such a way that the inclusion of a new algorithm is straightforward and does not affect either the algorithm or the global framework.

In order to verify the capabilities of the system, we have used the SFinGe software [20,2] to generate a database of 400 000 fingerprints that has been used for identification in a set of experiments on a hybrid cluster, ranging from sequential to massively parallel runs. In a search for more realistic fingerprints, we have applied the NIGOS *mindtct* minutiae extractor on the database and performed more experiments. Finally, another set of experiments has been executed using two large real-world databases from the NIST. All these experiments have been run with three well-known fingerprint matching algorithms [43,28,6].

After detailing the obtained results, we can conclude that the proposed framework fulfills the expectations. It has a linear scalability regarding the fingerprint database, as well as an optimal adaptability to the underlying hardware. In theory, this allows the identification in databases of arbitrary size as long as there is enough computing power. In practice, the identification time can be kept constant against the database growth just by augmenting the computing resources in the same proportion. The framework has also proven to maintain its good behavior independent of the underlying matching algorithms and fingerprint features.

Conflict of interest

None declared.

Acknowledgements

This work was supported by the research projects CAB(CDTI), TIN2011-28488 and TIN2009-14575. D. Peralta holds an FPU scholarship from the Spanish Ministry of Education and Science (FPU12/04902).

References

[1] A.K. Jain, R.M. Bolle, S. Pankanti, Biometrics: Personal Identification in Networked Society, Springer, 2005.
[2] D. Maltoni, D. Maio, A. Jain, S. Prabhakar, Handbook of Fingerprint Recognition, Springer-Verlag New York Inc., 2009.
[3] A. Jain, S. Prabhakar, L. Hong, S. Pankanti, Filterbank-based fingerprint matching, IEEE Transactions on Image Processing 9 (2000) 846–859.
[4] F. Liu, Q. Zhao, D. Zhang, A novel hierarchical fingerprint matching approach, Pattern Recognition 44 (2011) 1604–1613.
[5] D. Maio, D. Maltoni, Direct gray-scale minutiae detection in fingerprints, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997) 27–40.
[6] R. Cappelli, M. Ferrara, D. Maltoni, Minutia cylinder-code: a new representation and matching technique for fingerprint recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (2010) 2128–2141.
[7] N. Ratha, R. Bolle, Automatic Fingerprint Recognition Systems, Springer, New York, 2004.
[8] S. Pankanti, S. Prabhakar, A. Jain, On the individuality of fingerprints, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 1010–1025.
[9] A. Jain, L. Hong, R. Bolle, On-line fingerprint verification, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997) 302–314.
[10] A. Jain, L. Hong, S. Pankanti, R. Bolle, An identity-authentication system using fingerprints, Proceedings of IEEE 85 (1997) 1365–1388.
[11] H. Stone, High-Performance Computer Architecture, Addison-Wesley Longman Publishing Co., Inc., 1992.
[12] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, B. Smolinski, Toward a common component architecture for high-performance scientific computing, in: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999, pp. 115–124.
[13] M.S. Seidenberg, J.L. McClelland, A distributed, developmental model of word recognition and naming, Psychological review 96 (1989) 523–568.
[14] A. Datta, S. Soundaralakshmi, Fast parallel algorithm for distance transform, IEEE Transactions on System, Man, Cybernetics A, System, Humans 33 (2003) 429–434.
[15] A. Stamatakis, M. Ott, Exploiting fine-grained parallelism in the phylogenetic likelihood function with mpi, pthreads, and openmp: a performance study, in: Proceedings of the 3rd International Conference on Pattern Recognition in Bioinformatics, Springer-Verlag, 2008, pp. 424–435.
[16] M. Gong, Y. Zhang, Y.H. Yang, Near-real-time stereo matching with slanted surface modeling and sub-pixel accuracy, Pattern Recognition 44 (2011) 2701–2710.
[17] T.Y. Ho, P.M. Lam, C.S. Leung, Parallelization of cellular neural networks on GPU, Pattern Recognition 41 (2008) 2684–2692.
[18] G. Danese, M. Giachero, F. Leporati, N. Nazzicari, An embedded multi-core biometric identification system, Microprocessors and Microsystems 35 (2011) 510–521.
[19] M. Hulea, A. Astilean, T. Letia, R. Miron, S. Folea, Fingerprint recognition distributed system, in: Proceedings of the 16th IEEE International Conference on Automation, Quality and Testing, Robotics, vol. 3, 2008, pp. 423–428.
[20] R. Cappelli, D. Maio, D. Maltoni, Synthetic fingerprint-database generation, in: Proceedings of the 16th International Conference on Pattern Recognition, vol. 3, 2002, pp. 744–747.
[21] C.I. Watson, M.D. Garris, E. Tabassi, C.L. Wilson, R.M. Mccabe, S. Janet, K. Ko, User's Guide to NISTBiometric Image Software (NBIS), Technical Report, NIST, 2010.
[22] C. Watson, C. Wilson, NISTSpecial Database 4, Technical Report, NIST, 1992.
[23] C. Watson, NISTSpecial Database 14, Technical Report, NIST, 1993.
[24] G.K. Manacher, Production and stabilization of real-time task schedules, Journal of ACM 14 (1967) 439–465.
[25] F.B. of Investigation (Ed.), The Science of Fingerprints: Classification and Uses, U.S. Government Printing Office, 1984.

[26] H. Lee, R. Gaensslen, Advances in Fingerprint Technology, CRC, 2001.
[27] Q. Fang, N. Bhattacharjee, Incremental fingerprint recognition model for distributed authentication, in: Proceedings of the International Conference on Security and Management, 2008, pp. 41–47.
[28] X. Chen, J. Tian, X. Yang, A new algorithm for distorted fingerprints matching based on normalized fuzzy similarity measure, IEEE Transactions on Image Processing 15 (2006) 767–776.
[29] C. Gottschlich, C. Schönlieb, Oriented diffusion filtering for enhancing low-quality fingerprint images, IET Biometrics 1 (2012) 105–113.
[30] J. Liu-Jimenez, R. Sanchez-Reillo, L. Mengibar-Pozo, O. Miguel-Hurtado, Optimisation of biometric id tokens by using hardware/software co-design, IET Biometrics 1 (2012) 168–177.
[31] P. Sutthiwichaiporn, V. Areekul, Adaptive boosted spectral filtering for progressive fingerprint enhancement, Pattern Recognition 46 (2013) 2465–2486.
[32] N.K. Ratha, S. Chen, A.K. Jain, Adaptive flow orientation-based feature extraction in fingerprint images, Pattern Recognition 28 (1995) 1657–1672.
[33] L. Coetzee, E.C. Botha, Fingerprint recognition in low quality images, Pattern Recognition 26 (1993) 1441–1460.
[34] X. Jiang, W.Y. Yau, W. Ser, Minutiae extraction by adaptive tracing the gray level ridge of the fingerprint image, in: IEEE International Conference on Image Processing, vol. 2, 1999, pp. 852–856.
[35] X. Jiang, W. Yau, W. Ser, Detecting the fingerprint minutiae by adaptive tracing the gray-level ridge, Pattern Recognition 34 (2001) 999–1013.
[36] J. Liu, Z. Huang, K.L. Chan, Direct minutiae extraction from gray-level fingerprint image by relationship examination, in: IEEE International Conference on Image Processing, vol. 2, 2000, pp. 427–430.
[37] J. Chang, K. Fan, Fingerprint ridge allocation in direct gray-scale domain, Pattern Recognition 34 (2001) 1907–1925.
[38] M. Fons, F. Fons, N. Canyellas, E. Cantó, M. López, Hardware-software co-design of an automatic fingerprint acquisition system, in: IEEE International Symposium on Industrial Electronics, vol. III, 2005, pp. 1123–1128.
[39] M. Leung, W. Engeler, P. Frank, Fingerprint image processing using neural networks, in: International Conference on Computer and Communication Systems, IEEE, 1990, pp. 582–586.
[40] K. Nilsson, J. Bigun, Using linear symmetry features as a pre-processing step for fingerprint images, in: Audio and Video-Based Biometric Person Authentication, Springer, 2001, pp. 247–252.
[41] S.H. Lee, H.B. Chae, S.Y. Yi, E.S. Kim, Optical fingerprint identification based on binary phase extraction joint transform correlator, in: Proceedings of the International Society for Optical Engineering, vol. 2752, 1996, pp. 224–232.
[42] B.V.K.V. Kumar, M. Savvides, C. Xie, K. Venkataramani, J. Thornton, A. Mahalanobis, Biometric verification with correlation filters, Applied Optics 43 (2004) 391–402.
[43] X. Jiang, W. Yau, Fingerprint minutiae matching based on the local and global structures, in: Proceedings of the 15th International Conference on Pattern Recognition, vol. 2, IEEE, 2000, pp. 1038–1041.
[44] K. Cao, X. Yang, X. Chen, Y. Zang, J. Liang, J. Tian, A novel ant colony optimization algorithm for large-distorted fingerprint matching, Pattern Recognition 45 (2012) 151–161.
[45] N. Ratha, K. Karu, S. Chen, A. Jain, A real-time matching system for large fingerprint databases, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (1996) 799–813.
[46] S. Chang, F. Cheng, W. Hsu, G. Wu, Fast algorithm for point pattern matching: invariant to translations, rotations and scale changes, Pattern Recognition 30 (1997) 311–320.
[47] A. Hrechak, J. McHugh, Automated fingerprint recognition using structural matching, Pattern Recognition 23 (1990) 893–904.
[48] D. Koufaty, D. Marr, Hyperthreading technology in the netburst microarchitecture, IEEE Micro 23 (2003) 56–65.
[49] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the Spring Joint Computing Conference, ACM, 1967, pp. 483–485.
[50] G. Indrawan, B. Sitohang, S. Akbar, Parallel processing for fingerprint feature extraction, in: International Conference on Electrical Engineering and Information, 2011, pp. 1–6.
[51] R.F. Miron, T.S. Letia, M. Hulea, Two server topologies for a distributed fingerprint-based recognition system, in: 15th International Conference on System Theory, Control and Computing, 2011, pp. 1–6.
[52] K. Beghdad Bey, Z. Guessoum, A. Mokhtari, F. Benhammadi, Agent based approach for distribution of fingerprint matching in a metacomputing environment, in: Proceedings of the 8th International Conference on New Technologies in Distributed Systems, 2008, pp. 1–7.
[53] K. Nagaty, E. Hattab, An approach to a fingerprints multi-agent parallel matching system, in: IEEE International Conference on Systems, Man and Cybernetics, volume 5, 2004, pp. 4750–4756.

D. Peralta received the M.Sc. degree in Computer Science in 2011 from the University of Granada, Granada, Spain. He is currently a Ph.D. student in the Department of Computer Science and Artificial Intelligence, University of Granada. His research interests include data mining, biometrics and parallel and distributed computing.

I. Triguero received the M.Sc. degree in Computer Science from the University of Granada, Granada, Spain, in 2009. He is currently a Ph.D. student in the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain. His research interests include data mining, data reduction, biometrics, evolutionary algorithms and semi-supervised learning.

R. Sanchez-Reillo graduated as Telecommunication Engineer by the Polytechnic University of Madrid, obtaining his Ph.D. at the same University in 2000, based on Biometric Authentication in Smart Cards. From 1994 he has been researching at the University Group for Identification Technologies, managing the group since 2000. He is currently an Associate Professor at Carlos III University of Madrid. He is also member of ISO/IEC JTC1 SC17, SC27 and SC37 standardization bodies, holding some management position in national and international standardization bodies. His interests in R&D cover all Personal Identification Technologies, including Smart Cards, Biometrics and Secure Authentication Systems. He is a founder of IDTestingLab, an evaluation facility for identification products.

F. Herrera received his M.Sc. in Mathematics in 1988 and Ph.D. in Mathematics in 1991, both from the University of Granada, Spain. He is currently a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada. He has published more than 230 papers in international journals. He is a coauthor of the book "Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases" (World Scientific, 2001). He currently acts as an Editor in Chief of the international journal "Progress in Artificial Intelligence" (Springer). He acts as an Area Editor of the International Journal of Computational Intelligence Systems and Associated Editor of the journals: IEEE Transactions on Fuzzy Systems, Information Sciences, Knowledge and Information Systems, Advances in Fuzzy Systems, and International Journal of Applied Metaheuristics Computing; and he serves as a member of several journal editorial boards, among others: Fuzzy Sets and Systems, Applied Intelligence, Information Fusion, Evolutionary Intelligence, International Journal of Hybrid Intelligent Systems, Memetic Computation, and Swarm and Evolutionary Computation. He received the following honors and awards: ECCAI Fellow 2009, 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science", International Cajastur "Mamdani" Prize for Soft Computing (Fourth Edition, 2010), IEEE Transactions on Fuzzy System Outstanding 2008 Paper Award (bestowed in 2011), and 2011 Lotfi A. Zadeh Prize Best paper Award of the International Fuzzy Systems Association. His current research interests include computing with words and decision making, bibliometrics, data mining, biometrics, data preparation, instance selection, fuzzy rule based systems, genetic fuzzy systems, knowledge extraction based on evolutionary algorithms, memetic algorithms and genetic algorithms.

J.M. Benitez is an Associate Professor at the Department Computer Science and Artificial Intelligence (http://decsai.ugr.es), Universidad de Granada, Granada, Spain. Dr. Benítez holds an M.S. Degree and a Ph.D. in Computer Science, both from the Universidad de Granada. He is a member of the Research Group "Soft Computing and Intelligent Information Systems" (SCI2S, http://sci2s.ugr.es) and head of the "Distributed Computational Intelligence and Time Series" research lab (DICITS, http://sci2s.ugr.es/DiCITS). He is an active researcher in the Computational Intelligence field where his work covers the whole spectrum from foundations to applications in a number of engineering and scientific areas. In particular, his current fields of interest are time series analysis and modeling, distributed/parallel computational intelligence, high performance computing, cloud computing, data mining, biometrics, and statistical learning theory. He is a member of a number of scientific associations, including IEEE, IEEE Computational Intelligence Society, and EUSFLAT.