Universidad de Granada

Departamento de Ciencias de la Computación
e Inteligencia Artificial

Programa de Doctorado en Tecnologías de la Información y la Comunicación

Identificación y Clasificación Biométrica a Gran Escala Basada en Huellas Dactilares y GPU

Tesis Doctoral

Pablo David Gutiérrez Pérez

Granada, Marzo de 2017

Editor: Universidad de Granada. Tesis Doctorales
Autor: Pablo David Gutiérrez Pérez
ISBN: 978-84-9163-579-6
URI: http://hdl.handle.net/10481/48506

Universidad de Granada

Identificación y Clasificación Biométrica a Gran Escala Basada en Huellas Dactilares y GPU

MEMORIA PRESENTADA POR

Pablo David Gutiérrez Pérez

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

Marzo de 2017

DIRECTORES

Francisco Herrera Triguero y Miguel Lastra Leidinger

Departamento de Ciencias de la Computación
e Inteligencia Artificial

La memoria titulada "*Identificación y Clasificación Biométrica a Gran Escala Basada en Huellas Dactilares y GPU*", que presenta D. Pablo David Gutiérrez Pérez para optar al grado de doctor, ha sido realizada dentro del Programa Oficial de Doctorado en "*Tecnologías de la Información y la Comunicación*", en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección de los doctores D. Francisco Herrera Triguero y D. Miguel Lastra Leidinger.

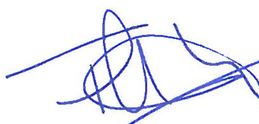
El doctorando, D. Pablo David Gutiérrez Pérez, y los directores de la tesis, D. Francisco Herrera Triguero y D. Miguel Lastra Leidinger, garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis, y hasta donde nuestro conocimiento alcanza, en la realización del trabajo se han respetado los derechos de otros autores a ser citados cuando se han utilizado sus resultados o publicaciones.

Granada, Marzo de 2017

El Doctorando

Fdo: Pablo David Gutiérrez Pérez

Los directores



Fdo: Francisco Herrera Triguero

Fdo: Miguel Lastra Leidinger

Esta tesis doctoral ha sido desarrollada con la financiación del Programa de Becas de Formación de Personal Investigador del Ministerio de Economía y Comptetitividad, en su Resolución del 28 de Noviembre de 2012, bajo la referencia BES-2012-060450.

Un paso a delante y dos hacia atrás,
seguimos andando pero sin avanzar.
091, Nada es real

Agradecimientos

Es imposible agradecer a toda la gente que tiene parte de culpa de que hoy esté escribiendo estas líneas en una sola página. Sé que voy a olvidar incluir muchos nombres y os pido disculpas por ello. Espero que luego no me lo echéis en cara.

Antes que a nadie, quiero dar las gracias a mi familia. A mis padres, Juan y Antoñita, además de por hacerme ser la persona que soy, por todo su trabajo y esfuerzo para que yo pudiera conseguir mis metas. A mis hermanos, Loli, Conchi y Juanba, por todos los buenos ratos y por todo lo que me han enseñado, han sido el mejor ejemplo que he podido tener. A mis cuñados, Miguel, Antonio y Yesi, por su apoyo y especialmente por mis sobrinos, Ana, Elvira, Miguel, David y Daniel, capaces de sacarme una sonrisa por muy duro que haya sido el día de trabajo.

Mis directores, Francisco Herrera y Miguel Lastra, también tienen que ocupar un lugar destacado ya que, sin su esfuerzo y dedicación constantes, habría sido imposible llegar hasta aquí. Su guía y sus consejos han sido mis compañeros de viaje durante estos años. Gracias Paco por enseñarme lo que es la investigación y todo lo que hay que sacrificar por ella. Gracias Miguel por todo tu apoyo, tu ayuda y por hacer las cosas sencillas cuando yo ya no veía soluciones posibles.

También quiero agradecer a José Manuel Benítez que, aunque no aparezca como director de esta tesis, ha estado directamente implicado en la misma. Sin sus consejos y recursos, esta tesis no habría llegado a buen puerto.

Después tengo que agradecer a todos mis compañeros, a los *senior*: Salva, Isaac, Joaquín, Victoria, Fran, Julián, Alberto..., a los de mi promoción: Dani, Sergio, Juanan, Rafa, Sara, Manu, Manuel, Pablo, Jorge, José Antonio, Pepe, Almudena, Lala..., y a los que aún les queda una buena parte de su camino: Jesús, Sergio, Paco, Elena, Diego... Compartir todo este tiempo, experiencias, partidos de padel y alguna cerveza con vosotros ha sido un apoyo fundamental para conseguir terminar esta tesis.

I would like to thank Jaume Bacardit, the ICOS research group and the School of Computing Sciences at Newcastle University for all their support in my research stay. They made me feel at home being so far. I still miss the Friday night boardgames at the common room.

A mis amigos, Dani, José Angel, Guillermo y Juanmi, no puedo olvidarlos porque, a pesar de que a veces pasen los meses sin que podamos hablar o vernos, siempre estáis ahí y siempre es como si nos hubiéramos visto ayer, os debo mucho. También a todos con los que comparto *Superbowls*, películas, juegos de mesa, baloncesto y kárate, no os puedo nombrar a todos pero quiero daros las gracias.

Last but not least, because actually she is the one that deserves more credit, the last words of this section are to thank you, Abi, not only for proofreading this dissertation, but also and more importantly for all your support and help. I do not think that there is anything I can do to repay you. You always see the best in me. Salamat Abi!

Table of Contents

Page

I PhD dissertation **1**

1 Introduction . . . 1

Introducción . . . 5

2 Preliminaries . . . 9

2.1 Graphics Processing Units . . . 9

2.1.1 GPU device architecture . . . 9

2.1.2 Programming GPU devices . . . 10

2.1.3 Measuring the performance . . . 11

2.2 Fingerprint matching . . . 12

2.2.1 Nearest neighbor Algorithm . . . 13

2.2.2 Fixed radius algorithm . . . 14

2.3 Classification and imbalanced data . . . 16

3 Justification . . . 19

4 Objectives . . . 20

5 Methodology . . . 21

6 Summary . . . 22

6.1 Nearest Neighbor Minutiae Matching on GPU devices . . . 22

6.2 Fixed Radius Minutiae Matching on GPU devices . . . 23

6.3 *k*NN classification algorithms on GPU devices . . . 23

6.4 Big Data Preprocessing on GPU devices . . . 24

7 Discussion of results . . . 26

7.1 Nearest Neighbor Minutiae Matching on GPU devices . . . 26

7.2 Fixed Radius Minutiae Matching on GPU devices . . . 26

7.3 *k*NN classification algorithms on GPU devices . . . 27

7.4 Big Data Preprocessing on GPU devices . . . 28

8 Concluding Remarks . . . 29

Conclusiones . . . 29

9 Future Work 31

II Publications: Published Papers **33**

1 A High Performance Fingerprint Matching System for Large Databases Based on GPU 34

2 Fast fingerprint identification using GPUs 45

3 GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs . 70

4 SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification 109

Bibliografía **119**

Chapter I

PhD dissertation

1 Introduction

The identification of people is an important issue in a large number of fields, from access control to rooms and buildings to criminology and forensics identification, passing through payments and identification in computer systems. With the advancement of the information society, the number of people that must be identified has greatly increased; big companies and public administration institutions reach hundreds of millions of individuals [iaf, uid].

Traditionally, this identification has been performed using different objects for each user: ID cards, passports, intelligent cards, etc. However, objects can be lost, stolen or falsified, which is a potential security risk. Another solution is the use of passwords. However, passwords can be forgotten or stolen too. Thus, the use of passwords is also a risk. There are hybrid solutions, such as credit cards, that combine objects and passwords, but this solution is still subject to the problems of the two previous solutions. Therefore finding identification systems that are not based on *what we have* or *what we know* but on *who we are* has been a great interest for the scientific community [JFR07].

Biometric-based identification provides a solution to these problems since it is based on intrinsic features, such as fingerprints, iris, face or DNA, of each person. In order to be suitable for an identification system, a feature needs to be: universal, unique, invariable and easy to use. Fingerprints are the most used features since they fulfill these properties [JFR07]: everybody has fingerprints, except in rare cases of severe amputations; the fingerprints of each finger of each person are different; fingerprints do not change along a person's life; fingerprints are fast, cheap and non-invasive to collect.

Fingerprints have been historically used for identification purposes. There are references that fingerprints of criminals were taken in Babylon around 2000 BC and there are scientific studies about fingerprints in the XVII century [Gre84, Bid85]. The first modern approach to fingerprint extraction for identification did not appear until the XIX century [Hen00] but their study and use continues nowadays.

A fingerprint is the pattern that the skin forms on a fingertip. This pattern is composed of ridges and valleys that form different designs that can be used for recognition. Fingerprints have different types of features that can be used to compare them. These features are usually divided into three different levels: global (singular points, orientation maps and pseudoridges), local (minutiae) and detail (pores and intra-ridge features).

A comparison between two fingerprints in order to determine if they come from the same person is usually called matching. Manual matching processes are tedious and time-consuming. The goal of an automatic fingerprint identification system is to avoid the need of manual matching or even human supervision in order to verify the identity of a person. A computer-based system can perform the matching operations in a systematic and efficient way, that speeds up the identification process.

An automatic matching algorithm receives two images of fingerprints and provides a score, within a range, which defines how similar both fingerprints are. The comparison could be done directly at image level but two images of the same fingerprint can exhibit significant differences due to rotations, translations and deformation on the skin at the moment the images were taken. This type of transformations may lead to low scores in an image-to-image comparison. Thus, a feature extraction process is performed to obtain the information referring to some of the different features previously commented and this information is used for the comparison. Among the fingerprint's features, the minutiae are by far the most used features for fingerprint recognition [PGT+15] because they can be easily described and their number allows an efficient and reliable comparison.

Minutiae are the bifurcations and the ends of the ridges. They are characterized by their position and angle in the image and by their type. Minutiae-based algorithms use these four values to compare two sets of minutiae and produce a score. Most algorithms perform two different steps: the first step builds a local structure per minutiae with information about its surrounding minutiae and compares the structures of each pair of minutiae from the two fingerprints; the second step produces the final score by combining the information obtained from the best matching pairs of minutiae. The accuracy of these methods depends on the operations performed on each step, the more complex they are, the more accurate and slower the algorithm is.

There are two different approaches to the fingerprint recognition problem and each of them is a problem on its own [MMJP09]:

- **Verification** [JHB97]: verification is the problem that determines whether two fingerprints belong to the same person or not. This is a 1:1 comparison, which requires only one comparison.
- **Identification** [JHPB97]: identification is the problem that determines which person, among a list of candidates, an input fingerprint belongs to. This is a 1:n comparison, which requires n comparisons.

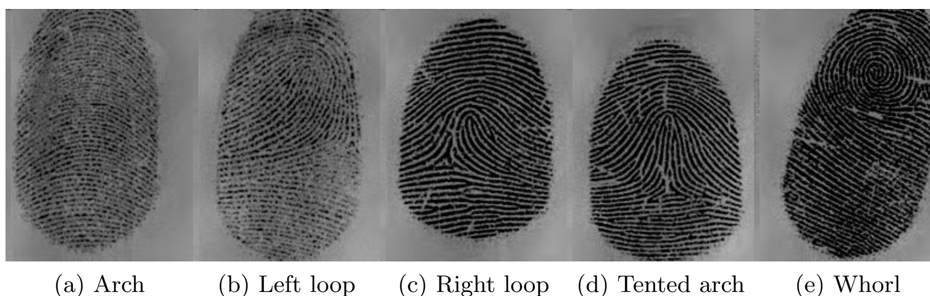
This thesis focuses on the identification problem which is the more challenging problem from a computer engineering point of view, given the increasing amounts of data to manage, and from a computer sciences point of view, since knowledge extraction techniques can be applied to improve the performance. An Automatic Fingerprint Identification System (AFIS) has to be:

- **Accurate**: The identification error rate should be as low as possible in order to not accept impostors and to not reject genuine users.
- **Efficient**: The time required to identify the user should be as little as possible since many applications require a response in a few seconds.
- **Scalable**: The number of users can grow over time, but the expected time for response should not, so the system needs to be able to cope with it by increasing hardware resources.
- **Flexible**: Fingerprints can have different characteristics, such as quality and size, but the system should be able to deal with them.

An AFIS usually has two different use cases: introduction of a new user into the database and identification of a user against the database. The first case is commonly called enrollment. In order to build a database of fingerprints, it is mandatory to scan the fingerprints of every user of the system. Before storing the fingerprint, it can be preprocessed, extracting its features and pre-computing data required by the matching algorithm. The second case occurs when a user tries to identify himself against the system. An input fingerprint is provided, preprocessed in the same way as the fingerprints in the database and compared against them in order to return the best match found. The most common approach consists of using a verification algorithm that compares the input fingerprint to each fingerprint in the database [MMJP09].

AFIS usually suffer from two difficulties, both related to the size of the database: high identification time and accuracy loss. An identification process takes at least n times longer than the underlying verification algorithm used, where n is the number of fingerprints in the database. In the same way, there is usually only one matching fingerprint in the database (it could be none if the input fingerprint belongs to a non-enrolled user), the probability of making an identification mistake increases as the database increases. Actually, these problems are so serious that a direct brute-force approach is not possible if the database is larger than a few thousands of users [PTSR+14]. However, current societal necessities are reaching the order of hundreds of millions of people [iaf, uid]. There is a strong need for systems that tackle these problems and provide reliable and scalable solutions.

The use of classification techniques can improve the performance of an AFIS both in time and accuracy [DHS12]. The classification proposed by Henry [Hen00] is the most widely used. It presents five types of fingerprints: left loop, right loop, whorl, arch and tented arch, as Figure 1 shows. Using a classification method to identify the type of each fingerprint can reduce the number of matching operations required in the identification process since the input fingerprint would be compared first against the fingerprints of its own type [MMJP09]. If it were not matched, it would be compared against the rest of the database. This can happen for two reasons: the input fingerprint is not present in the database or it was misclassified.



(a) Arch (b) Left loop (c) Right loop (d) Tented arch (e) Whorl

Figure 1: Five fingerprint classes defined by Henry [Hen00]

Misclassifying a fingerprint introduces a high cost for the identification process so the classifier used has to be reliable. One of the most powerful techniques to improve the accuracy of multiple class classification is the use of a One-vs-One ensemble [GFB+11]. This technique involves training a classifier that focuses on distinguishing between each pair of classes and classifying the input combining the outputs of each classifier. However, the characteristics of the Henry classification make this task more difficult because the fingerprints are not evenly distributed among the classes. More than 90% of fingerprints are left loop (33.8%), right loop (31.7%) and whorl (27.9%), while the rest belong to arch (3.7%) and tented arch (2.7%).

The difference in the number of samples of each class makes the process of training the

classifier more difficult. This situation is usually known as learning from imbalanced data [HG09]. Imbalanced classification has been studied from a machine learning point of view [LFG+13, PBS15, Kra16] but there is still the necessity for new methods that can deal with this situation when the number of instances is large. Currently AFIS should handle hundreds of millions of fingerprints, which means the same quantity of training samples for the classification method is available. In order to apply these techniques to fingerprint classification, it is necessary to solve this problem related to the size of the database.

Graphics Processing Units (GPUs) have proven to be a very useful tool in the acceleration of computationally intensive algorithms. These devices introduce massive parallelism in the calculations reducing run times in several orders of magnitude. Applications of this technology can be found in different fields as molecular modelling [FEV+09], bioinformatics [STDV07] or shallow-water simulation [LMUn+09]. Matching algorithms usually require a large number of computations whose time performance should improve if they are redesigned to work on GPU devices while keeping the same accuracy. GPU devices should also be useful in classification since different methods have already been redesigned to work on them [GDNB10].

In this thesis we are going to study how GPU devices can be used to improve the performance of fingerprint matching methods and imbalanced classification strategies that are suitable to be introduced in a large scale AFIS. In order to do that, we first studied the characteristics of GPU devices and matching methods. Then, we proposed GPU-based designs for two of the most representative matching algorithms. After that, we studied how to solve the imbalanced learning problem in large databases by providing a scalable solution for one of the most well-known techniques to deal with imbalanced classification and the underlying classification algorithm that this technique uses.

Section 2 describes in detail the background of the main topics covered by the thesis: GPU devices and their characteristics (Section 2.1), fingerprint matching (Section 2.2) and classification and imbalanced data (Section 2.3).

Section 3 presents the justification of this doctoral dissertation, describing the problems addressed throughout the thesis. Section 4 marks the objectives pursued and Section 5 introduces the methodology used to achieve them. Section 6 summarizes the works that comprise this thesis, while Section 7 analyses their results in relation to the objectives. Section 8 presents the conclusions obtained during the work collected in this thesis and Section 9 points out future lines of work derived from the results obtained.

Introducción

La identificación de personas es un problema importante en un gran número de campos, desde el control de acceso a habitaciones y edificios a la criminología, pasando por pagos e identificación en sistemas informáticos. Con el avance de la sociedad de la información, el número de personas que deben identificarse también se ha incrementado de forma considerable; grandes compañias y administraciones públicas pueden alcanzar cientos de millones de individuos [iaf, uid].

Tradicionalmente, estas identificaciones se realizaban mediante el uso de distintos objetos para cada usuario: tarjetas de identificación, pasaportes, tarjetas inteligentes, etc. Sin embargo, estos objetos pueden perderse, robarse o falsificarse, lo que supone un potencial riesgo de seguridad. Otra solución habitual es el uso de contraseñas, pero esta solución también presenta problemas ya que estas contraseñas se pueden olvidar y, también, robar. Siendo, de este modo, un riesgo. Hay soluciones híbridas, que combinan objetos con contraseñas, como las tarjetas de crédito, pero esta solución tiene los problemas combinados de ambas propuestas. Por tanto, encontrar sistemas de identificación que no se basen en *qué tenemos* o *qué sabemos* si no en *quíen somos* ha sido de grán interés en la comunidad científica [JFR07].

La identificación biométrica proporciona una solución a estos problemas ya que se basa en marcadores intrínsecas de cada persona, como pueden ser las huellas dactilares, el iris, la cara o el ADN. Para ser adecuada para sistemas de identificación, un marcador necesita ser: universal, única, invariable y fácil de usar. Las huellas dactilares son los marcadores biométricos más utilizados ya que cumplen estas propiedades [JFR07]: todo el mundo tiene huellas dactilares, excepto en raros casos de amputaciones severas, las huellas dactilares de cada persona son diferentes y su captura es sencilla, económica y no invasiva.

Las huellas dactilares se han usado historicamente para propósitos de identificación. Hay referencias de que se tomaban las huellas dactilares de criminales en Babilonia en torno al año 2000 aC. Hay estudios científicos sobre huellas dactilares en el siglo XVII [Gre84, Bid85], aunque la primera técnica moderna para extracción e identificación de huellas dactilares no aparece hasta el siglo XIX [Hen00], continuándose su estudio hasta nuestros días.

Una huella dactilar e el patrón que forma la piel de la punta del dedo. Este patrón se compone de crestas y valles que forman distintos diseños que se pueden utilizar para reconocer a una persona. Las huellas dactilares tienen distintos tipos de características que se pueden utilizar en su comparación. Dichas características se dividen habitualmente en tres niveles: global (puntos singulares, mapas de orientación y pseudocrestas), local (minucias) y detalle (poros y características intracresta).

Una comparación entre dos huellas dactilares para determinar si corresponden o no a la misma persona se denomina habitualmente emparejamiento o *matching*. El proceso manual de matching es tedioso y requiere bastante tiempo. El objetivo de un sistema de identificación automático consiste en evitar la necesidad de realizar un proceso de *matching* de forma manual o requerir supervisión humana a la hora de verificar la identidad de una persona. Un sistema informático de este tipo puede realizar las operaciones de *matching* de forma sistemática y eficiente, lo que acelera el proceso de identificación.

Un sistema de *matching* automático recibe dos imágenes y proporciona una puntuación, dentro de un rango, que define como de similares son ambas huellas. La comparación podría hacerse directamente a nivel de imágen, pero dos imágenes de la misma huella pueden mostrar diferencias significativas debido a rotaciones, translaciones y deformaciones en la piel en el momento en que las imágenes se tomaron. Este tipo de transformaciones puede llevar a la obtención de puntuaciones

bajas en una comparación de imagen a imagen. Para evitarlo, se realiza una extracción de características en la que se obtiene información referente a algunas de las distintas características de las huellas previamente comentadas y es esta información la que se utiliza en las comparaciones. Entre las características de las huellas dactilares las minucias son las más populares para tareas de reconocimiento [PGT+15] ya que se pueden describir fácilmente y su número permite una comparación fiable y eficiente.

Las minucias son las bifurcaciones y los finales de las crestas. Se caracterizan por su posición y ángulo en la imagen y por su tipo. Los algoritmos basados en minucias utilizan estos cuatro valores para comparar dos conjuntos de minucias y producir una puntuación. La mayor parte de los algoritmos realizan dos pasos: el primero de ellos construye una estructura local por minucia con información sobre las minucias que la rodean y compara las estructuras de cada parea de minucias de las dos huellas. El segundo paso computa la puntuación final combinando la información obtenida de las mejores parejas de minucias. El nivel de acierto de estos métodos depende de las operaciones calculadas en cada paso, cuanto mas complejas sean, más fiable y lento será el algoritmo.

Existen dos formas de aproximarse al problema del reconocimiento de huellas dactilares y cada una de ellas es un problema en sí misma [MMJP09]:

- **Verificación** [JHB97]: verificación es el problema que consiste en determinar si dos huellas pertenecen a la misma persona o no. Esto es, una comparación 1:1, que solamente require una comparación.
- **Identificación** [JHPB97]: identificación es el problema que consiste en determinar a que persona, entre una lista de candidatos, pertenece una huella de entrada. Esto es una comparación 1:n, que requiere n comparaciones.

Esta tesis se centra en el problema de identificación que es el problema más exigente desde el punto de vista de la ingeniería informática, dada la creciente cantidad de información a manejar, y desde el punto de vista de la ciencia de datos, ya que se pueden aplicar técnicas de extracción de conocimiento para mejorar el rendimiento. Un sistema automático de identificación de huellas dactilares (*Automatic Fingerprint Identification System*, AFIS) tiene que ser:

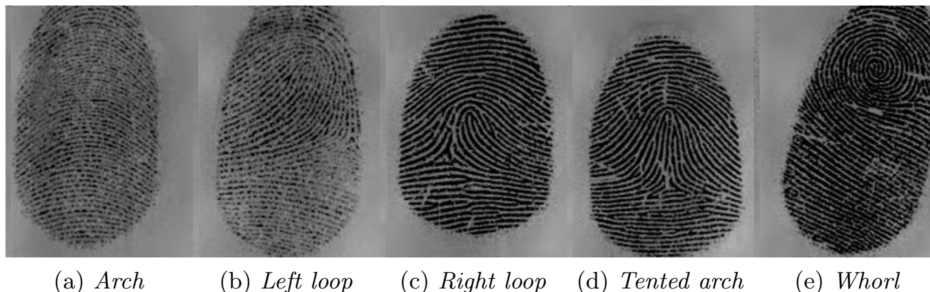
- **Preciso**: El ratio de error de identifación debe ser tan bajo como sea posible para no aceptar impostores y no rechazar usuarios genuinos.
- **Eficiente**: El tiempo requerido para identificar al usuario debe ser tan bajo como sea posible ya que muchas aplicaciones requieren una respuesta en pocos segundos.
- **Escalable**: El número de usuarios puede crecer a lo largo del tiempo, pero el tiempo esperado de respuesta no debería, por tanto, el sistema tiene que ser capaz de afrontar estos cambios incrementando los recursos hardware necesarios.
- **Flexible**: Las huellas dactilares pueden tener distintas características, como su calidad o tamaño, pero el sistema debe ser capaz de trabajar con ellas y producir una respuesta.

Un AFIS normalmente tiene dos casos de uso: la introducción de un nuevo usuario en la base de datos y la identificación de un usuario contra la base de datos. El primer caso se denomina habitualmente registro. Para poder construir la base de datos de usuarios, es necesario escanear las huellas dactilares de cada usuario del sistema. Antes de almacenar estas huellas, es posible preprocesarlas, extrayendo sus características y precomputando los datos necesarios para

el algoritmo de *matching*. El segundo caso ocurre cuando un usuario trata de identificarse contra el sistema. Se obtiene una huella de entrada que se preprocesa de la misma manera que las huellas de la base de datos y, posteriormente, es comparada con esas mismas huellas para devolver la que más se le parezca. El enfoque más habitual consiste en un proceso de verificación que compara la huella de entrada con cada huella de la base de datos [MMJP09].

Los AFIS suelen padecer dos problemas, ambos relacionados con el tamaño de la base de datos: un tiempo de identifación alto y perdida de precisión. Un proceso de identificación requiere a menos n veces el tiempo del algoritmo de identificación que se utilice, donde n es el número de huellas dactilares en la base de datos. Del mismo modo, solo hay una huella que se corresponde con cada usuario de la base de datos (aunque podría no haber ninguna si el usuario no se ha registrado), la probabilidad de cometer un erro de identificación se incrementa conforme se incrementa el tamaño de la base de datos. De hecho, estos problemas afectan de una forma tan seria a este tipo de sistemas que una aproximación por fuerza bruta no es posible si la base de datos tiene más de algunos miles de usuarios [PTSR+14]. Sin embargo, las necesidad es de la sociedad actual están alcanzando el orden de los cientos de millones de personas [iaf, uid]. Hay una gran necesidad de sistemas que puedan afrontar estos problemas y ofrezcan soluciones fiables y escalables.

El uso de técnicas de clasificación puede mejorar el rendimiento de un AFIS tanto en tiempo como en precisión [DHS12]. La clasificación propuesta por Henry [Hen00] es la más habitual. En ella se presentan cinco tipos de huellas: *left loop*, *right loop*, *whorl*, *arch* and *tented arch*, como muestra la Figura 2 shows. Utilizando un algoritmo de clasificación para identificar el tipo de cada huella se puede reducir el número de operaciones de *matching* necesarias para una identificación ya que la huella de entrada se compara primero con las huellas de si propio tipo [MMJP09]. Si no se encuentra al usuario en ese tipo, la búsqueda de extiende al resto de la base de datos. Este caso puede ocurrir por dos razones: la huella de entrada no está en la base de datos o fue clasificada erróneamente.



(a) *Arch* (b) *Left loop* (c) *Right loop* (d) *Tented arch* (e) *Whorl*

Figura 2: Cinco tipos de huellas dactilares definidos por Henry [Hen00]

Clasificar erróneamente una huella dactilar introduce un coste muy elevado para el proceso de clasificación, por tanto el clasificador utilizado tiene que ser confiable. Una de las técnicas más potentes para mejorar la precisón de clasificación con varias clases es el uso de un *ensemble One-vs-One* [GFB+11]. Esta técnica entrena un clasificador que se centra en distinguir entre cada pareja de clases y produce la clasificación final combinando los resultados de cada clasificador. Sin embargo, las características de la clasificación de Henry hacen que este problema sea más complejo ya que el número de huellas de cada clase no esta distribuido de forma equilibrada. Más del 90 % de las huellas son de tipo *left loop* (33,8 %), *right loop* (31,7 %) y *whorl* (27,9 %), mientras que el resto se reparte entre *arch* (3,7 %) y *tented arch* (2,7 %).

La diferencia en el número de muestras de cada clase complica el proceso de entrenamiento

de un clasificador. Esta situación se conoce como aprendizaje sobre datos desbalanceados [HG09]. La clasificación desbalanceada se ha estudiado desde el punto de vista del aprendizaje automático [LFG+13, PBS15, Kra16] pero aún son necesarios nuevos métodos que puedan afrontar problemas con un gran número de instancias. Actualmente , un AFIS debería manejar cientos de millones de huellas, lo que implica la misma cantidad de muestras para el clasificador. Para poder aplicar estas técnicas a la clasificación de huellas dactilares, es necesario resolver este problema relacionado con el tamaño de la base de datos.

Las unidades de tarjeta gráfica (*Graphics Processing Units*, GPUs) han demostrado ser una herramienta útila para acelerar algoritmos computacionalmente intensivos. Estos dispositvos introducen paralelismo masivo en los cálculos reduciendo los tiempos de ejecución en varios órdenes de magnitud. Se pueden encontrar aplicaciones de esta tecnología en campos tan distintos como el modelado molecular [FEV+09], la bioinformática [STDV07] o la simulación de aguas poco profundas [LMUn+09]. Los algoritmos de *matching* habitualmente requieren un número elevado de cálculos cuyo rendimiento, en tiempo, debería mejorar si se rediseñan para trabajar en dispositivos GPU sin modificar su precisión. Los dispositivos GPU también pueden ser útiles en clasificación ya que existen métodos que ya han sido rediseñados para trabajar sobre ellos [GDNB10].

En esta tesis se va a estudiar como los dispositivos GPU se pueden utilizar para mejorar el rendimiento de métodos de *matching* de huellas dactilares y de estrategias de clasificación desbalanceada que sean apropiados para utilizarse en un AFIS a gran escala. Para poder llevar esto a cabo, primero se estudiaron las caracteristicas de los dispositivos GPU y de los métodos de *matching*. Después se han propuesto diseños basados en GPU para dos de los algoritmos de *matching* más representativos. Posteriormente, se ha estudiado como resolver el problema de aprendizaje desbalanceado en grandes bases de datos proponiendo una solución escalable para una de las técnicas más conocidas en este campo y para el método de clasificación en el que está basada.

La Sección 2 describe en detalle los antecedentes de los principales temas que se cubren en esta tesis: dispositivos GPU y sus características (Sección 2.1), *matching* de huellas dactilares (Sección 2.2) y clasificación y datos desbalanceados (Sección 2.3).

La Sección 3 presenta la justificación de esta tesis doctoral, describiendo los problemas afrontados a lo largo de la misma. La Sección 4 marca los objetivos que se persiguen y la Sección 5introduce la metodología empleada para alcanzarlos. La Sección 6 resume los trabajos que componen esta tesis, mientras que la Sección 7 analiza sus resultados en relación a los objetivos. La Sección 8 presenta las conclusiones obtenidas durante el trabajo realizado en la tesis y la Sección 9 señala líneas de trabajo futuro que derivan de los resultados obtenidos.

2 Preliminaries

In this section we are going to describe the required background for the main topics covered by this thesis. First, we present the GPU devices architecture and characteristics (Section 2.1). After that, we are going to study the main characteristics of matching algorithms (Section 2.2). Finally, we are going to introduce the classification problem, focusing on imbalanced data scenarios (Section 2.3).

2.1 Graphics Processing Units

In the last decade, Graphics Processing Units (GPU) have emerged as a powerful parallel computing piece of hardware. These devices group thousands of processing cores, providing large-scale parallelism, even on desktop computing platforms. GPU devices were designed to render 3D graphics in games and design applications. These devices are responsible for the floating point computations involved in rendering in a very parallel and efficient way, using a Single Instruction Multiple Data (SIMD) architecture and offloading the computational cost from the CPU device.

The use of GPU devices to run general purpose programs started with the first devices. However, developers had to map scientific calculations onto problems that could be represented by triangles and polygons, until NVIDIA presented CUDA [CUD] in 2006. NVIDIA CUDA is the hardware/software architecture that allows the use of NVIDIA GPU devices for general purpose computation, exposing their parallel processing nature to non-graphics-specialized developers. NVIDIA CUDA presents the GPU as a parallel co-processor. A CUDA program alternates sections of sequential code running on the CPU device, with parallel sections that run on the GPU device. These parallel sections are introduced through function calls, which are called kernels.

Since GPU devices have a different architecture, we are going to describe the characteristics of these devices from a hardware point of view, in Section 2.1.1. Once we are familiar with their structure, we can present the characteristics and restrictions from the software design point of view, in Section 2.1.2. An algorithm that is going to be implemented using CUDA needs to be designed taking into account the characteristics and restrictions of GPU devices to achieve the best possible performance. Finally, Section 2.1.3 introduces usual measures of performance for GPU devices.

2.1.1 GPU device architecture

GPU devices have a large number of cores that reaches up to thousands for some devices, but these cores are different from the cores we can find in modern CPU devices, which are more similar to arithmetic logic units. GPU cores are grouped into streaming multiprocessors (SMX). Each SMX has a certain number of cores, registers and cache memories, so it can work independently from the others. All the SMX of a device are equal, but there are differences if different devices are compared.

As mentioned previously, GPU devices have a SIMD architecture. This means that the same instruction is applied to different data at the same time, using different threads and cores. However, not all the cores of a GPU device run the same instruction at the same time. A warp is a group of 32 threads that run the same instruction simultaneously. If a SMX has more than 32 cores, which is something not unusual in the devices released in the last years, it can run several warps at once.

The number of registers of a GPU device is much larger than the number of registers of a CPU device. The reason for this is that the registers are assigned to a thread from the start

until it finishes its computations. This allows GPU devices to perform extremely fast context switching between warps. A SMX handles several warps switching among them to maximize the time they perform useful computation. This way the delays related to memory access are *hidden* with computation.

For this reason, the number of warps handled by a SMX is larger than the number of warps that its cores can run at once. However, there are limits for the threads a SMX can handle, the number of warps and the number of registers per thread. These maximum values vary depending on the specific device.

The first level of cache memory, usually called L1, is also inside the SMX. This memory has a particularity: it is programmable. CUDA programmers can specify information to be stored in this cache memory. This is useful because that memory is shared among the threads, although there are some restrictions at software level regarding which threads can share information with others, as we will see in the next section.

GPU devices, as a separate piece of hardware, have their own memory and cannot access the computer's main memory. This means that all the information required to perform the computation on GPU devices needs to be copied from one memory to another through the PCI Express port. In the same way, the results required from the CPU device point of view also need to be copied. These copies can be made asynchronously so the GPU device can perform computations with other data during the transfer.

There are two other types of memory that have their own caches inside the SMX:

Texture memory The GPU is designed to render graphics so it is optimized to work with textures. CUDA provides functions to use textures to store data. This memory is optimized for accesses that have 2-dimensional locality.

Constant memory This memory, as its name suggests, stores data that is not modified during the execution of the kernels but can be read by them.

2.1.2 Programming GPU devices

A kernel is a function that is applied to different data in parallel on a GPU device. Each kernel function is run on the GPU device through a set of threads. These threads are grouped into blocks and the set of all blocks that run a kernel is called grid. Every thread in the grid runs a copy of the same kernel, on different data. The data to be processed by a thread is determined by two three-dimensional indexes, one to identify the block and the other one to identify the thread within the block. When a kernel is called, the number of blocks and the number of threads per block need to be specified.

The previous section stated that a SMX handles several warps. From this point of view, each SMX handles several blocks. Handling complete blocks in a SMX allows CUDA to provide synchronizing operations at block level, but since each SMX works independently, it is not efficient to synchronize the whole grid.

In the same way, only the threads of the same block can share information through shared memory. The amount of shared memory is limited to a maximum of 64 kB per SMX in current devices. The amount of shared memory per block needs to be specified in the kernel function call, in the same way as the dimensions of the kernel grid.

Regarding the programming style, there are factors that can also harm the performance

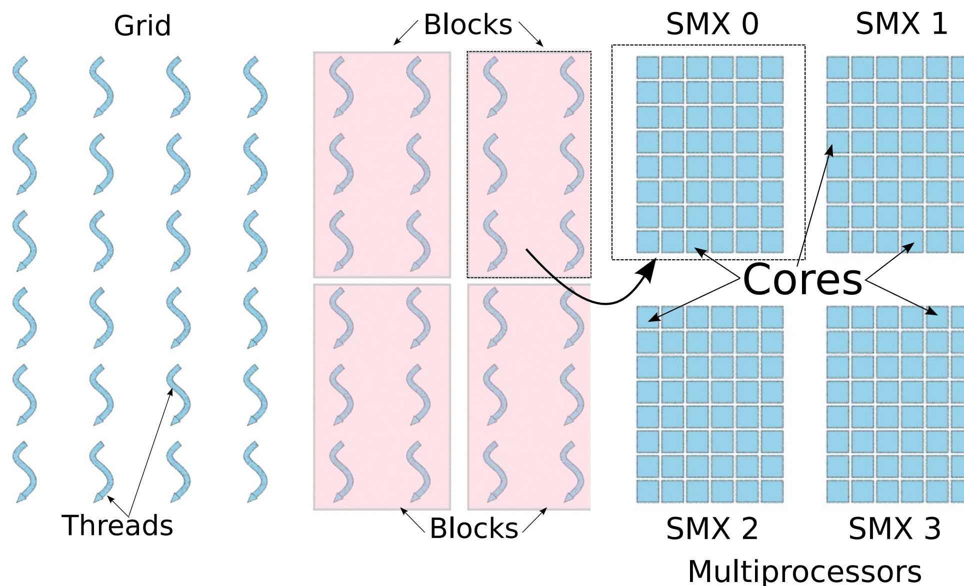


Figure 3: Grid, threads, blocks and multiprocessors. Each block is run on the same multiprocessor

obtained. It was mentioned in the previous section that the threads of a warp run the same instruction simultaneously. The evaluation of conditional predicates that produce different results for the threads within a warp reduces the performance since the evaluation of the next instruction needs to be serialized.

Another important aspect is the memory access pattern. The whole memory and cache system of GPU devices is optimized for coalescent access, which means that consecutive threads are expected to access consecutive positions of memory. If the memory access pattern is not coalescent, the performance obtained drops significantly.

All these hardware and software considerations need to be taken into account in order to take advantage of the resources that GPU devices offer.

2.1.3 Measuring the performance

There are two main measures of the performance in this field:

Speed Up This measure expresses the ratio between sequential and parallel running time of an application that performs the same computing (Equation I.1). It is also used in parallel and distributed computation.

$$S = \frac{t_{sequential}}{t_{parallel}} \tag{I.1}$$

Occupancy This measure computes a percentage value per kernel that represents how close it is to the maximum number of warps that can be handle by a SMX. There is a theoretical value and a real value. The theoretical one can be determined by the characteristics of the device, the parameters of the kernel and the number of registers that it needs. The real value is measured by the actual use of the GPU device while running the kernel.

The speed up provides a real measure of how the performance of a process has been improved by using a parallel scheme. If the speed up is greater than 1, this means that the parallel

implementation outperforms the sequential one. On the other hand, if it is lower than 1, the sequential implementation is more efficient in terms of runtime. The theoretical maximum speed up is the number of parallel processing units although this value is difficult to reach as there are dependencies between parallel sections (which require synchronization points). Furthermore, the additional communication overhead and modules that cannot be parallelized decrease the actual speed up value.

In general purpose GPU computing, it is difficult to establish a maximum speed-up value, as GPU and CPU devices have different architectures and their performance not only depends on the number of cores of the device but also on the occupancy of the multiprocessors and several parameters associated to each kernel launch, such as the number of thread blocks and their size. Some of these factors, for example the number of blocks in the kernel call, might depend on the data being processed. The number of cores, on the other hand, differs depending on the device. Moreover, the occupancy can give more information about how the resources of the GPU are being used.

The theoretical occupancy is computed based on the grid dimensions, the number of registers required per thread and the amount of shared memory required per block. These values are used to compute the maximum number of warps that can be handled at the same time by a SMX. If this number of warps is equal to or greater than the physical limit of warps and blocks that can be handled by a SMX, the kernel can achieve a theoretical 100% occupancy. If this number is lower, the occupancy obtained is equal to the ratio between the maximum obtained and the physical limit.

On the other hand, the real occupancy depends on other values like the efficiency of the read and write operations or the divergence caused by conditional orders in a warp. The maximum value that the real occupancy can achieve is equal to the theoretical occupancy. If it is much lower in comparison to the theoretical, it usually means that the kernels have not been efficiently designed and that latencies are not hidden. Therefore, a suboptimal performance level is achieved. Although a high theoretical and real occupancy is desirable, it is not critical in order to achieve good performance. In fact, sometimes it is possible to achieve a better performance with a scheme that has an associated lower theoretical occupancy, if this allows the GPU device to have a better real occupancy.

2.2 Fingerprint matching

The matching algorithm is the core of an AFIS since it is in charge of determining how similar every pair of fingerprint is. As mentioned before, most of the matching algorithms in the literature are based on the minutiae of the fingerprint [PTSR+14]. This is mainly due to its reliability and the manageable amount of data it involves. These algorithms build a local structure based on a neighborhood for each minutia and can be divided into two main categories depending on how they define it [MMJP09, PGT+15]:

- **Nearest neighbor** The neighborhood of a given minutia is defined by the k closest minutiae.
- **Fixed radius** The neighborhood of a given minutia is defined by all the minutiae inside an imaginary circle of radius R centered on the minutia.

In the first case, the neighborhood has the same size for all the minutiae. This makes nearest neighbor algorithms very efficient although usually very sensitive to missing and spurious minutiae. The neighborhood size of the fixed radius algorithms depends on the density of minutiae and can

be different for each minutia. This makes these kind of algorithms more complex than nearest neighbor ones but more tolerant with respect to missing minutiae.

In this section we present a very representative algorithm of each type to clarify their characteristics. These algorithms have also been redesigned to work on GPU devices.

2.2.1 Nearest neighbor Algorithm

The algorithm of Jiang [JY00] is one of the classic algorithms in the literature. It is a nearest neighbor algorithm and was one of the first algorithms to introduce the combination of global and local matching.

Jiang's algorithm builds two data structures associated to each minutia, M_k, of the fingerprint: an array with the characteristics of the minutia and another array, F_k, with characteristics of the neighborhood, Fl_k. The F_k array stores the characteristics of the minutia and depends on the feature extractor employed, it stores the minutia coordinates within the fingerprint, (x_k, y_k), the orientation of the minutia, φ_k, and its type, t_k:

$$F_k = (x_k \; y_k \; \varphi_k \; t_k)^T \tag{I.2}$$

The Fl_k array stores information that relates the minutia M_k to the l minutiae of its neighborhood. This information is composed of the distances between minutiae (Equation I.3), the relative direction between them (Equation I.4), the difference of their angles (Equation I.5), the ridge count between them and their type. These are stored in the format shown in Equation I.6.

$$d_{ki} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{I.3}$$

$$\theta_{ki} = d\phi\left(\tan^{-1}\left(\frac{y_k - y_i}{x_k - x_i}\right), \varphi_k\right) \tag{I.4}$$

$$\varphi_{ki} = d\phi\left(\varphi_k, \varphi_i\right) \tag{I.5}$$

$$Fl_k = (d_{ki_1} \; d_{ki_2} \; ... \; d_{ki_l} \; \theta_{ki_1} \; \theta_{ki_2} \; ... \; \theta_{ki_l} \; \varphi_{ki_1} \; \varphi_{ki_2} \; ... \; \varphi_{ki_l} \; n_{ki_1} \; n_{ki_2} \; ... \; n_{ki_l} \; t_k \; t_{i_1} \; t_{i_2} \; ... \; t_{i_l})^T \tag{I.6}$$

Where d_{ki} is the distance between minutiae k and i; θ_{ki} is the relative direction between minutiae k and i; φ_{ki} is the difference of angles between minutiae k and i; $d\phi\left(\alpha, \beta\right)$ is a function that computes the difference between angles α and β; and n_{ki} is the ridge count between minutiae k and i.

These two arrays, F_k and Fl_k can be precomputed and stored for each fingerprint in the database, so they are only computed from the input fingerprint. Once the arrays have been computed, the algorithm performs a two-step matching between each fingerprint in the database and the input fingerprint.

The first step is the local matching that tries to find the pair of minutiae (one from each fingerprint) which are the most similar, according to their neighborhoods. The algorithm defines a similitude matrix, sl, where each position $sl(i, j)$ stores the score achieved by the local matching of the minutia M_i of one fingerprint and the minutia M_j of the other. This score is calculated by a weighted sum of the elements of the Fl_k array.

The second step performs a global matching. The algorithm fixes the coordinates of the pair of minutiae that achieved the maximum value in the previous step and compares the relative position and orientation of the rest of the minutiae. Using relative positions and orientations minimizes the influence of the rotation and the displacement of the pictures of the fingerprints but it is still sensitive to image distortion. Another matrix ml is built using the relative coordinates and the sl matrix value of each pair of minutiae.

The final score is calculating by adding the maximum values of the ml matrix ensuring that every minutia is used only once. To comply with this restriction, the algorithm looks for the maximum iteratively, setting the row and column of the previous maximum to zero.

Jiang's algorithm is very efficient because it only involves two operations with matrices whose size depends on the number of minutiae of each fingerprint, which is usually not very high. In addition, since it is a nearest neighbor method, Fl_k arrays are always the same size, which enables efficient implementation of the algorithm.

2.2.2 Fixed radius algorithm

The Minutia Cylinder-Code matching algorithm [CFM10], called MCC, is the most complex algorithm in this field. MCC uses a combination of local and global matching and tries to combine the high efficiency that the nearest neighbor algorithms reach, because all minutiae have the same number of neighbors, with better tolerance to deformations achieved by the fixed radius algorithms. In addition, the authors designed the algorithm to reach different goals:

- Improve the accuracy of the algorithm when the fingerprints are deformed.
- Achieve interoperability with other algorithms by using standard characteristics (in this case, minutiae).
- Get an efficient algorithm adapted for implementation in embedded systems.

As indicated by one of the objectives, the algorithm uses only minutiae in the calculations. In fact, it only uses the position and orientation of minutiae and ignores the type. The authors base this decision on the assumption that feature extractors can easily mistake this parameter thereby making it unreliable.

To ensure that the neighborhood characteristics of a minutia, m are stored in a structure that is always the same size, as in the nearest neighbor methods, the MCC algorithm builds a cylinder associated to each minutia. This cylinder is centered on the minutia, has a fixed radius, R, and has a height of 2π. The cylinder is discretized into cells as shown in Figure 4, where N_s and N_d represent the number of cells used for the diameter and the height of the cylinder respectively. Each cell has an associated two-coordinate position, $p^m_{i,j}$, that represents the center of the cell projected on the cylinder base, and an angle, $d\varphi_k$, defined by the height of the cell.

A numerical value $C_m(i,j,k)$ is calculated for each cell. This value stores the sum of the contributions of every minutia m_t different from m in the neighborhood of $p^m_{i,j}$, $N_{p^m_{i,j}}$. The radius of this neighborhood is $3\sigma_S$ where σ_S is the standard deviation of the spatial contribution. The neighborhood can consider minutiae that are outside the cylinder but are closer than $3\sigma_S$ to $p^m_{i,j}$.

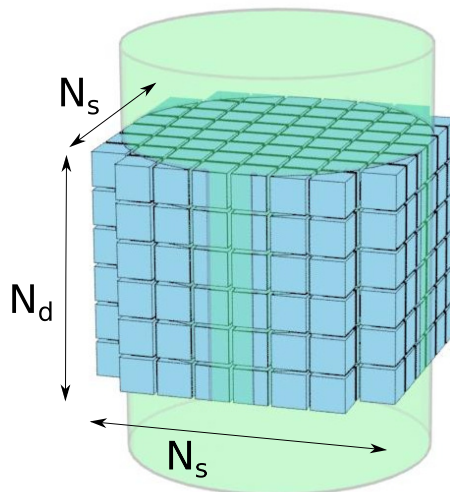


Figure 4: Structure of a cylinder

The function C_m is defined as follows:

$$C_m(i,j,k) = \begin{cases} \Psi\left(\sum_{m_t \in N_{p^m_{i,j}}} \left(C^S_m\left(m_t, p^m_{i,j}\right) \cdot C^D_m\left(m_t, d\varphi_k\right)\right)\right) \\ \text{if } \xi_m\left(p^m_{i,j}\right) = valid \\ invalid \text{ otherwise} \end{cases} \tag{I.7}$$

where $C^S_m\left(m_t, p^m_{i,j}\right)$ is the spatial contribution of the minutia, which is higher when the minutia location is closer to $p^m_{i,j}$, $C^D_m\left(m_t, d\varphi_k\right)$ is the directional contribution of the minutia, which is higher when the minutia direction is closer to the defined angle $d\varphi_k$, and $\xi_m\left(p^m_{i,j}\right)$ is a function that checks if the Euclidean distance between m and $p^m_{i,j}$ is equal to or lower than R and if $p^m_{i,j}$ is inside the convex hull [PS85] of the minutiae that form the fingerprint with a certain offset. Ψ is a sigmoid function controlled by two parameters that limit the contribution of dense minutiae clusters and ensure that the final value is in the $\mathbb{R}[0,1]$ range. This function is defined as:

$$\Psi(v) = Z(v, \mu_\Psi, \tau_\Psi) = \frac{1}{1 + e^{-\tau_\Psi(v-\mu_\Psi)}} \tag{I.8}$$

Once the cylinder computation has finished, a cylinder can be discarded if it does not contain enough useful information (valid cells) or if the number of minutiae that were used to make the computation is too low. The discarded cylinders are usually associated to minutiae on the edges of the fingerprint. These minutiae are more sensitive to errors and deformations so they are less reliable.

When the algorithm has computed the cylinders of both fingerprints, a local matching process can be started. This computation is made on every pair of cylinders storing the resulting local similarity scores in a matrix. Two cylinders C_a and C_b are considered matchable if the directional difference between the two minutiae is not greater than a certain value, δ_θ, and if the intersection of valid cells of both cylinders is big enough. If the cylinders are matchable, their

similarity is defined as:

$$\gamma(a,b) = \begin{cases} 1 - \frac{\|c_{a|b} - c_{b|a}\|}{\|c_{a|b}\| + \|c_{b|a}\|} & \text{if } C_a \text{ and } C_b \text{ are } \textit{matchable} \\ 0 & \text{otherwise} \end{cases} \tag{I.9}$$

Where $c_{a|b}$ is a vector that stores the value of the corresponding cell of C_a if the cell is valid in C_a and C_b; $c_{b|a}$ is defined in the same way but storing the value of the cell of C_b.

After the local matching step, a global matching process, called consolidation, is performed. The authors propose several methods of consolidation that are basically different ways to add some of the values obtained by the local matching, usually the highest values.

To achieve the last goal, which is to build an algorithm adapted for implementation in embedded systems, the authors explain a binary version of the algorithm in the paper. However, this version loses accuracy compared to the full algorithm.

2.3 Classification and imbalanced data

In a classification problem a model is built in order to predict the label of a data sample from its characteristics. In the case of fingerprints, such model can be used to obtain the class of a fingerprint using characteristics extracted from the image [GDP+15b, GDP+15a]. Being able to reliably classify fingerprints reduces the number of candidate fingerprints for an AFIS as many candidate fingerprints from the database can be efficiently discarded in a step performed before the matching process.

To build such a model, the data samples, usually called instances, are divided into two sets. The first set is used to train the model while the second set is used to test its accuracy, hence both sets are usually known as training and test set. Using this approach the model is not trained with all the data, which means there is information that could be missing. In order to avoid this situation it is common to split the datasets in several folds of the same size. The model is trained as many times as folds, using a different fold as test set every iteration and the rest as train set.

The simplest classification algorithm consist of measuring the Euclidean distance between a test instance and every training instance and then assigning the label, also known as class, of the closest training instance to the test instance. This algorithm is known as the Nearest Neighbor (NN) method [FHJ51]. There is an extension of this algorithm which considers a certain value, k, of neighbors [CH67]. This kNN algorithm assigns the class to each instance in the test set using the majority class among its neighbors. A simple algorithm like this is one of the most important data mining techniques [WK10] and is used as te basis for other classification algorithms [GW07, WNC07, NSB03] and data preprocessing methods [CBHK02, LGH12].

A trained classifier defines decision boundaries for the different classes. However, it is harder to build accurate classifiers for multiclass problems, like fingerprint classification, than for only two classes. The decision boundaries in the latter case can be simpler. This idea has been exploited in order to improve the performance in multiclass classification.

The most common strategies that exploit this idea are called “one-vs-one” (OVO) [KPD90] and “one-vs-all” (OVA) [AMMR95]. The first one creates as many binary problems as possible combinations between pairs of classes. Each classifier learns how to distinguish between each pair. The second one creates a binary problem per class. Each classifier learns how to distinguish each class from the rest. In both cases the results of the classifiers are aggregated to produce the global prediction.

The distribution of instances among the classes can also have a large impact on the performance of the classification [HG09]. If a class has a significantly larger number of instances than the other, the decision boundaries are harder to define due to the lack about information of the minority class. This problem is known as imbalanced classification [HG09, Kra16, PBS15], since the data of the different classes is not balanced. In an OVA approach every classifier has to face this situation, but it can also affect an OVO approach, depending on the distribution of the instances among the classes. Imbalanced data can lead to having small classes ignored by the ensembles, since it is unlikely that one of these classes reaches enough positive predictions.

There are different methods that tackle the problem of imbalanced data [LFG$^+$13]:

- **Algorithmic modification** Modifying classification algorithms in order to tackle the problem by design [ZE01].
- **Cost-sensitive learning** Introducing costs for misclassification of the minority class at data or algorithmic level [Dom99] [ZLA03].
- **Data sampling** Preprocessing the data in order to reduce the difference between the number of instances of each class [BPM04] [CBHK02].

Data sampling strategies have the advantage of being completely independent of the classification algorithm used. Once the dataset has been preprocessed it can be used with different classification methods in order to test their performance without making any changes. Preprocessing algorithms try to reduce the difference between the number of instances of both classes either by increasing the number of instances of the minority class (oversampling methods) or reducing the number of instances of the majority class (undersampling methods).

The number of instances of the minority class can be increased by duplicating existing instances or by creating new instances based on the information of the actual instances of the minority class. The Random Over-Sampling (ROS) method [BPM04] is an example of the first case. This algorithm randomly duplicates instances of the minority class. The SMOTE technique [CBHK02] is an example of the second case. This algorithm is based on the kNN algorithm, as it computes the neighborhood of each minority instance within the minority class. After that, it creates new instances using random interpolation between an instance and one of its neighbors.

In the same way, the reduction of instances of the majority class can be done randomly [LFG$^+$13] by selecting random instances of the majority class. Other methods try to select the most representative instances within the majority class[MFV02, GCH08].

Fingerprint classification suffers both problems, since it has 5 different classes and the distribution of instances among them is extremely imbalanced. In addition, misclassification has a high impact on the AFIS performance since it directs the search of the matching fingerprint in the wrong section of the database. Fingerprint classification also introduces another issue for classification algorithms: the volume of data.

As previously mentioned, there is a necessity for AFIS that can handle hundreds of millions of fingerprints [iaf, uid]. This means that a system with that number of fingerprints has the same number of instances that can be used to train the classification algorithm. However, considering the large amount of computations that classification algorithms require, they need an extremely long time to build their models. In recent years, this problem has been tackled using what has been called Big Data approaches[Mad12, ZEd$^+$11]. These solutions are based on the MapReduce paradigm [DG08] and rely on tools like Apache Hadoop [Whi15] or Apache Spark [ZCD$^+$12]. However, it would be interesting if preprocessing methods could be applied in advance and without

the costly equipment that these platforms require, since it is common to try several preprocessing settings in order to obtain the best results [TdRL+15]. In this scenario, oversampling methods have the advantage of only using the instances of the minority class.

3 Justification

The previous sections have stated the need for AFIS that can handle millions of fingerprints. The main bottleneck of these systems is located at the comparison of the input fingerprint with the fingerprints stored in the database. Different techniques, such as the use of GPU devices and fingerprint classification, have the potential to improve the performance of the identification process. However, these techniques present issues that need to be addressed in order to integrate them as part of an AFIS.

Minutiae matching algorithms are, in general, not designed to work on GPU devices. The computation of their data structures and the comparison operations need to be redesigned in order to be suitable for the characteristics of these devices, which is a challenging and key task to achieve a high performance level. Two different types of matching algorithms have been studied:

Nearest neighbor methods : these methods use regular structures since only the information of a fixed number of minutiae is considered. Jiang's method belongs to this family of techniques.

Fixed radius methods : these techniques use information of a variable number of minutiae. MCC is an important representative of this family of methods.

The data access patterns and the amount of computation required differ from one type to the other. This can lead to different performance levels on GPU devices, but there is no a priori information about which type may better suit this type of hardware platforms. At the time this thesis was started, no matching algorithm had been adapted to work on GPU devices, although there was an approach based on FPGA [JC08], which is another parallel architecture.

Fingerprint is a multiclass classification problem with an imbalanced distribution of instances. These issues have been studied in data mining and can be improved with OVO ensembles and preprocessing techniques. However, classification of large amounts of data, as is the case of fingerprints, is still a challenging problem, especially for preprocessing algorithms. There is a need for scalable and reliable preprocessing methods.

GPU devices can also be useful to improve preprocessing algorithms without using MapReduce platforms. The SMOTE technique is based on the kNN algorithm that has been successfully adapted to GPU devices [GDNB10, ARBM12, KDD14]. The different characteristics of these approaches need to be studied in order to check if they present scalability issues for extremely large databases. After this study, we can propose a new approach that can be used to build a scalable SMOTE algorithm over it.

The aforementioned issues can be encompassed within the scope of this thesis: the study of the capabilities of GPU devices to improve the performance of fingerprint identification and classification.

4 Objectives

After reviewing the state of the art and the potential open problems in the previous sections, it is possible to focus on the objectives of this thesis. These objectives are:

- **To study the suitability of GPU devices to implement fingerprint minutiae matching algorithms.** There are two main families of minutiae matching methods which have their own characteristics and requirements. In order to prove the suitability of GPU devices a significant algorithm of each type should be redesigned. Jiang's algorithm and MCC algorithm are significant representatives of the nearest neighbor and the fixed radius algorithm families, respectively.

  The proposed GPU-based designs have to keep the original results and consider GPU devices of different characteristics. Furthermore, if several GPU devices are available, the design should be able to scale and take advantage of them.

- **To analyze the scalability of current GPU-based approaches of the kNN classification algorithms and the development of new scalable approaches** Different GPU-based designs have been proposed over the years but there are still open scalability issues. It is required to study the different characteristics of these approaches in order to identify the weak and strong points of each of them. Taking advantage of all these knowledge, we can propose a new design that can provide high scalability and performance for large datasets.

- **To build a scalable preprocessing algorithm for large databases that does not require the use of MapReduce platforms**. Oversampling methods usually only require instances from the minority class. A design scheme that allows these methods to run on large datasets within a reasonable amount of time without using MapReduce platforms can be applied to the fingerprint classification problem.

5 Methodology

The methodology used in this thesis follows the guidelines of the scientific method. Nevertheless, it is important to adapt the general guidelines to the specific problem studied. In this section that adaptation is presented:

1. **Observation:** Detailed study of the fingerprint identification problem, its characteristics and the main methods that tackle it. The study of the possibilities and characteristics of GPU devices to be used in this problem and the study of the data mining techniques that can be applied are also required.

2. **Hypothesis formulation:** The design of new methods that make use of GPU devices to improve the performance on the open problems in fingerprint identification and classification. The new methods should fulfill the previously mentioned objectives.

3. **Observation gathering:** Obtaining results for the new methods and measuring the performance achieved.

4. **Contrasting hypothesis:** Comparison of the obtained results with the previously observed and studied methods. The comparison has to be as fair as possible, using the same hardware and databases.

5. **Hypothesis proof or rejection:** Acceptance, rejection and modification, according to case, of the developed techniques as a consequence of the two previous steps. If necessary, formulate a new hypothesis and repeat the process.

6. **Scientific thesis:** Extraction, redaction and acceptance of the conclusions obtained throughout the research process. All the approaches and results gathered along the entire process should be synthesized into the thesis dissertation.

6 Summary

This section presents a summary of the proposals described in the publications associated to this thesis. Afterwards, Section 7 will show an overview of the obtained results. The research carried out for this thesis and the results obtained in each case are collected into the following published papers:

- P.D. Gutiérrez, M. Lastra, F. Herrera, J.M. Benítez. A High Performance Fingerprint Matching System for Large Databases Based on GPU. IEEE Transactions on Information Forensics and Security 9:1 (2014) 62–71, doi: 10.1109/TIFS.2013.2291220.
- M. Lastra, J. Carabaño, P.D. Gutiérrez, J.M. Benítez, F. Herrera. Fast fingerprint identification using GPUs. InformationSciences, 301 (2015) 195–214. doi: 10.1016/j.ins.2014.12.052
- P.D. Gutiérrez, M. Lastra, J. Bacardit, J.M. Benítez, F. Herrera. GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs. Information Sciences 373 (2016) 165–182. doi: 10.1016/j.ins.2016.08.089
- P.D. Gutiérrez, M. Lastra, J.M. Benítez, F. Herrera. SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification. Submitted to Progress in Artificial Intelligence.

This section presents a summary of the different proposals presented in this dissertation related to the objectives defined in Section 4. Sections 6.1 and 7.2 present GPU-based approaches for different minutiae matching methods. Then, Section 6.3 shows the study of the different GPU-based kNN methods and presents our approach. Finally, Section 6.4 details our design for large database preprocessing on commodity hardware.

6.1 Nearest Neighbor Minutiae Matching on GPU devices

Nearest neighbor matching algorithms use information about the closest minutiae of each minutia to perform the matching operation. These algorithms usually create a local structure to encode these relationships, and Jiang's algorithm is no exception. The number of neighbors is usually small, which keeps the computational requirements low in identity verification environments. However, when dealing with large-scale identification scenarios, the total number of comparisons becomes computationally expensive.

GPU devices provide a massive source of parallelism that can be used to improve the performance of Jiang's algorithm in identification problems. In order to achieve a high performance level, GPU devices need to be provided with enough workload. Jiang's algorithm's one-to-one comparison is relatively simple and it does not provide the required workload level. It is therefore mandatory to perform one-to-many comparisons.

A similar workload scheme is used in the kernels that compute both parts of the algorithm: local and global matrices are computed column-like in one thread. This scheme ensures a coalescent access to the data leading to a high performance improvement. The final step of the algorithm that transforms the global matrix to obtain the matching score is also performed on GPU-device.

The database is processed chunk-wise. Our design takes advantage of asynchronous memory transfers to copy the next chunk of the database while one chunk is still being computed. This

scheme avoids idle periods on GPU device. Several GPU devices, if available, can work in parallel over different chunks of the database improving the performance of the algorithm.

Our design has been tested on two different fingerprint databases of real and synthetic fingerprint samples, with sizes ranging from 54 thousand to 800 thousand. Three different types of GPU devices have been used to measure the performance obtained and up to 4 GPU devices have been used in parallel.

The journal paper associated to this part is:

- M. Lastra, J. Carabaño, P.D. Gutiérrez, J.M. Benítez, F. Herrera. Fast fingerprint identification using GPUs. InformationSciences, 301 (2015) 195–214. doi: 10.1016/j.ins.2014.12.052

6.2 Fixed Radius Minutiae Matching on GPU devices

The minutia cylinder code (MCC) algorithm is one of the most reliable fixed radius methods. It creates a structure, called cylinder, for each minutia that stores information related to the surrounding minutiae. Then, it compares the similarity of pairs of minutiae from different fingerprints using those cylinders and, finally, produces a score combining and refining the best similarity results. This last step is called consolidation.

The reliability of the algorithm comes at a cost, being computationally expensive. GPU devices can improve the performance of different steps of the algorithm. Our proposal uses different kernels to compute the data structures involved: cylinders (to encode the information about the neighborhood of each minutiae), the similarity matrix, and the operations related to the final score combination process. These kernels group the computations in convenient ways in order to provide a high performance level.

The massive parallelism provided by GPU devices allows the search of the best matching fingerprint in the database as a set of one-to-many comparison steps instead of just many one-to-one similarity tests, making the identification process more efficient. Scores referring to several fingerprints are computed at once in the similarity computation and in the relaxation step of the consolidation.

Since a small fraction of the algorithm is computed on CPU device, our design uses two CPU threads to avoid idle times on the GPU device. This way, using asynchronous copies, the data for the next matching process is copied while another matching operation is being computed.

The experimental set up uses different GPU types and also presents results of two GPU devices collaborating. The performance of the algorithms was measured on different fingerprint databases of real and synthetic fingerprint samples, with sizes ranging from 1000 to 100 000 fingerprints.

The journal paper associated to this part is:

- P.D. Gutiérrez, M. Lastra, F. Herrera, J.M. Benítez. A High Performance Fingerprint Matching System for Large Databases Based on GPU. IEEE Transactions on Information Forensics and Security 9:1 (2014) 62–71, doi: 10.1109/TIFS.2013.2291220.

6.3 kNN classification algorithms on GPU devices

The kNN classification algorithm is one of the most important classification algorithms due to its simplicity and high accuracy. It is also a computationally demanding algorithm, as the number

of computations is linearly dependent on the size of the dataset. There are several GPU-based approaches that tackle these computational problems for large datasets.

These approaches classify several instances at once, building a distances matrix. Subsequently, these techniques use different methods to perform the selection of the neighborhood of each instance in parallel. Each step is computed on a separate kernel function and the kernel set is processed consecutively. All approaches present issues related to one of these steps, or both.

Regarding the computation of the distance matrix, almost every existing approach requires storing the whole matrix on GPU device memory. When this is not possible the dataset is split into several chunks and computed iteratively. However the performance drops significantly since the achieved GPU device occupancy level also lessens.

Regarding the neighborhood selection, most approaches are based on sorting methods. The memory access patterns that these sorting methods require are not especially suited for GPU devices and usually require a large number of synchronization operations. The performance of this step lowers as the problem size increases.

We propose a scalable and memory efficient approach for the kNN rule, called GPU-SME-kNN, which tackles the problems related to the dependence between dataset size and the memory footprint of the distance matrix. The distance matrix is computed in portions of a user defined size.

The neighborhood is computed incrementally as each portion of the matrix becomes available, allowing the reuse of the data structures. The selection method is based on the quicksort algorithm and it is designed in a way that avoids synchronization inside blocks and performs global memory access operations in a coalescent way.

This design is also suitable for other lazy learning algorithms based on kNN. Three of these algorithms: center kNN, kNN adaptive and symmetric kNN are also studied.

Our approach is compared to several other GPU-based kNN approaches and to the Keel [AFSG$^+$09] modules for lazy learning algorithms. Two different large datasets where used in the experiments up to a size of more than 4.5 million instances. These datasets were subsampled to different sizes to study how the different algorithms scale.

The journal paper associated to this part is:

- P.D. Gutiérrez, M. Lastra, J. Bacardit, J.M. Benítez, F. Herrera. GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs. Information Sciences 373 (2016) 165–182. doi: 10.1016/j.ins.2016.08.089

6.4 Big Data Preprocessing on GPU devices

As stated before, it would be interesting to perform preprocessing to deal with imbalanced data over large datasets without the use of MapReduce platforms. Among the different solutions to deal with this type of data, preprocessing algorithms seem to be the most interesting option since they are independent from the classification algorithm used. Preprocessing algorithms can be divided into over-sampling methods, if they increase the number of instances of the minority class, and under-sampling methods, if they reduce the number of instances of the majority class.

Over-sampling methods have the advantage that they only need the data from the minority class. With the proper design, these data can be handled on commodity hardware without the need for MapReduce platforms, even in Big Data scenarios. GPU devices can be used to compute

the most resource intensive operations in an acceptable time.

The SMOTE technique for over-sampling is one of the most popular oversampling algorithms and it is based on the kNN algorithm. In the previous section we presented an approach for the kNN algorithm that can handle large datasets efficiently. This approach is combined with a proper memory handling scheme that reduces the memory footprint to the space required to store the minority class and the neighborhood of each instance. Our design is also suitable for other methods, like Random Over Sampling (ROS), which was also tested.

Our approach for sampling algorithms has been tested with different versions of four datasets, ranging from 0.5 million instances to 10 million instances, and imbalanced ratios up to 49. These experiments have been performed on three different hardware platforms: a server node, a desktop PC and a laptop.

The journal paper associated to this part is:

- P.D. Gutiérrez, M. Lastra, J.M. Benítez, F. Herrera. SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification. Submitted to Progress in Artificial Intelligence.

7 Discussion of results

The following subsections summarize and discuss the results obtained in each specific stage of the thesis.

7.1 Nearest Neighbor Minutiae Matching on GPU devices

Our GPU-based design for Jiang's algorithm introduces a one-to-many comparison and an efficient workload for every step that leads to a high runtime performance. The accuracy results of the algorithm are not affected by the device used for the computations, since the operations performed are the same.

Two different fingerprint databases have been used: one with real fingerprints (DB14 [Wat93]) and 54 thousand samples and another one with synthetic fingerprints, created by the SFinGe software [CMM02], with 800 thousand samples. A total of five GPU devices have been used in the experiments: four server range GPU devices (two Tesla M2090 and two Tesla K20), installed on the same server node, and a desktop device (GeForce GTX 680). The desktop device cannot handle any whole database due to the lack of the required main memory size, 10 thousand and 100 thousand samples, of the respective databases, were used in those tests.

The careful design provides a huge performance speed-up ranging from 42X to 83X on a single GPU device when compared to a single thread CPU-based version. These values increase when sets of two and four GPU devices are working together on the same matching process reaching an improvement of 287X.

If we translate these speed-up values into matching processes per second, we obtain that a single GPU device can process more than 400 thousand fingerprints per second and the combination of four devices reaches 1.5 million fingerprints per second. However, the accuracy of this algorithm in the identification problem is not high enough to base a complete identification system on it, but it can be used as a first step in a two-step fingerprint identification system that uses a more reliable algorithm on the second step.

7.2 Fixed Radius Minutiae Matching on GPU devices

The proposed GPU-based design for the MCC algorithm introduces different kernel functions to compute the main steps of the algorithm and combines several matching processes in a one-to-many process suitable for identification tasks. The accuracy results of the algorithm do not change independently of the platform used, since the operations performed are the same.

The experimental set up of this work uses two different GPU devices: a server range device and a high-end desktop device. These results are compared to a single threaded CPU implementation of the algorithm on different databases, two from real fingerprints (DB4 [WW92] and DB14 [Wat93]) and one created synthetically with SFinGe [CMM02]. The sizes of these databases range from 1000 to 100 000 fingerprints. The SFinGe database has been subsampled at different values to show the performance of the algorithm as the number of fingerprints increases. Four different sets of parameters have been used in the experiments, two different cylinder sizes and two alternative consolidation steps.

The results obtained in the experiments prove that GPU-devices are a useful tool to improve the time performance of the MCC algorithm since the speed-up obtained ranges from 20X to 52.4X for the server node and from 28.2X to 56.6X for the desktop device. The experiment using increasing

database sizes shows that these speed-up ratios remain stable if enough workload is supplied to the GPU device, a point that is reached at database size of 5000 fingerprints. The number of fingerprints per second that a single GPU device can handle was also computed, reaching a rate of more than 55 000 matching operations per second.

The largest speed-up values for each device are reached with different configurations of the algorithm: the server range device obtains a better performance with the largest cylinder size and the desktop devices works better with the smallest size. Each device also seems to adapt better to a different consolidation step. These differences are related not only to the different architectures of the devices, but also to the parameters of the kernel that control the one-to-many comparison scheme. These parameters have been chosen in order to provide the best possible result on each device and configuration.

The synthetic fingerprint database was also used to measure the performance obtained with two server range devices. Two different setups were tested: in the first one, both GPU devices collaborate in the same matching process; in the second one, each GPU device works independently on different sections of the database. The results obtained show that the latter case achieves a better performance than the former. The speed-up level doubles when the GPU devices are used separately. This version is also easier to implement since it does not require the different GPU devices to exchange information and it could be easily extended to more GPU devices if they were available.

7.3 *k*NN classification algorithms on GPU devices

The scalable and memory efficient *k*NN classification algorithm (GPU-SME-*k*NN) that we presented tackles the main issues that appear in the existing approaches available in the literature. The distance matrix is computed in portions to avoid the linear dependence with the dataset size and the neighborhood is computed in an incremental and efficient way using a quicksort-based algorithm.

Our design is compared with one of the first approaches to the GPU-based *k*NN technique, which is used as a baseline in most of the papers in the literature, and with two other methods that use similar ideas to the ones we presented. The first one computes the distance matrix chunk-wise too and the second one uses a different approach to adapt the quicksort algorithm to the selection problem.

These methods were tested on two large datasets that were subsampled at different instance sizes, ranging from 250 000 to more than 4.5 million, and different values for the k parameter: 5, 100 and 1000. The results obtained show how our design scales better in both dimensions, problem size and k value. For some experiments our version is two times faster than the next best technique, which is a significant speed-up considering that all the methods compared are accelerated by a GPU device.

For the lazy learning algorithms included in our study the comparison was performed against the Keel machine learning software [AFSG+09], which does not use multi-thread acceleration. It was only possible to obtain results up to 650 thousand instances. The results of our design become several orders of magnitude faster than the ones obtained using a traditional CPU-based software like Keel as the size of the dataset increases.

7.4 Big Data Preprocessing on GPU devices

We presented a design for over-sampling methods that combines GPU-based parallelism and an efficient memory handling scheme to be able to perform preprocessing on Big Data datasets without the use of a MapReduce platform. Our design has been applied to the SMOTE algorithm and the Random Over-sampling (ROS) technique.

Two different settings for both methods have been tested with different versions of four datasets, ranging from 0.5 million instances to 10 million instances, and imbalanced ratios up to 49. These experiments have been performed on three different hardware platforms: a server node, a desktop PC and a laptop.

The results obtained show an improvement on the identification of the minority class using both algorithms. However, SMOTE outperforms ROS in most scenarios of our study. Considering the time required for the most time demanding experiment, server and desktop configurations took around 25 minutes while the laptop took almost two hours to process a problem of 4.5 million instances with more than 1 million instances of the minority class. Traditional implementations of these algorithms were not able to produce results for those experiments after 8 hours of runtime.

8 Concluding Remarks

In this thesis, we have addressed the problem of fingerprint identification and classification in large databases. We have analyzed, designed and implemented different approaches based on GPU devices to improve the performance of identification systems.

The initial objective was to study the suitability of GPU devices for implementing minutiae matching algorithms, which constitute the main computational bottleneck of identification systems. We have proposed two different designs for significant algorithms of the two main families of minutiae matching methods: nearest neighbor, with Jiang's algorithm, and fixed radius, with MCC algorithm. The designs were tested on different devices and the obtained results show a large improvement on the performance of both methods on all devices. The design also maintained the accuracy results since the same operations are performed on GPU and CPU devices.

The use of several GPU devices has also been studied, combining up to four different devices working simultaneously. Two different approaches were proposed, one where the different devices collaborate on the same matching process and another one where they worked on different sections of the database. The latter approach has obtained a better performance reaching 1.5 million fingerprint matching operations per second when using four GPU devices with Jiang's algorithm.

The second objective was to analyze the scalability of current GPU-based designs for the kNN classification method and the development of new scalable approaches that could be used for large databases in preprocessing algorithms. We have studied the characteristics of the different techniques in the literature identifying their weaknesses and strong points. That knowledge has been used to build a scalable and memory efficient approach that can tackle large datasets without losing performance and can be adapted to a broad variety of GPU devices.

The last objective was to build a scalable preprocessing algorithm for large datasets that does not require the use of MapReduce platforms. We have designed a new GPU-based approach for oversampling methods that has been applied to the well known SMOTE technique. This design has been able to efficiently handle datasets with several millions of instances in a reasonable time on commodity hardware. The experimental setup included a laptop computer, which was able to perform all the tests successfully.

Conclusiones

En esta tesis, hemos abordado el problema de la identificación y clasificación de huellas digitales en grandes bases de datos. Hemos analizado, diseñado e implementado diferentes enfoques basados en dispositivos GPU para mejorar el rendimiento de los sistemas de identificación.

El objetivo inicial era el estudio de la capacidad de los dispositivos GPU para implementar algoritmos de *matching* de minucias, que constituyen el principal cuello de botella de los sistemas de identificación. Se han propuesto dos diseños distintos para algoritmos significativos de las dos principales familias de algoritmos de *matching* de minucias: vecino más cercano, con el algoritmo de Jiang, y radio fijo, con el algoritmo MCC. Estos diseños han sido probados en distintos dispositivos y los resultados obtenidos muestran una mejora considerable en el rendimiento de ambos métodos en todos los dispositivos. Además, el diseño mantiene los resultados a nivel de acierto ya que tanto los dispositivos GPU como los CPU realizan las mismas operaciones.

El uso de varios dispositivos GPU también ha sido estudiado, combinando hasta cuatro dispositivos diferentes trabajando simultáneamente. Se han propuesto dos soluciones distintas, una en la que los diferentes dispositivos colaboran en el mismo proceso de *matching* y otra en la que cada dispositivo trabaja sobre distintas secciones de la base de datos. Esta última opción ha obtenido mejores resultados, alcanzando 1,5 millones operaciones de *matching* por segundo, con cuatro dispositivos GPU en el algoritmo de Jiang.

El segundo objetivo consistía en analizar la escalabilidad de los diseños existentes para le algoritmo de clasificación basado en kNN que utilizan dispositivos GPU y desarrollar nuevas propuestas escalables que pudieran utilizarse en tareas de preprocesamiento de grandes bases de datos. Se han estudiado las características de las distintas técnicas disponibles en la literatura, identificando sus ventajas y debilidades. En base a este conocimiento, se ha construido un método escalable y eficiente en memoria que permite abordar grandes conjuntos de datos sin perder rendimiento y que puede adaptarse a una gran variedad de dispositivos GPU.

El último objetivo era construir un algoritmo de preprocesamiento escalable para grandes bases de datos que no requiera el uso de plataformas MapReduce. Se ha diseñado un método basado en dispositivos GPU para algoritmos de sobremuestreo que se ha aplicado a la bien conocida técnica SMOTE. Este diseño ha sido capaz de manejar eficientemente conjuntos de datos con varios millones de instancias en un tiempo razonable utilizando hardware de nivel usuario. El marco experimental incluía el uso de un ordenador portátil, que fue capaz de realizar todas las pruebas de forma satisfactoria.

9 Future Work

The work carried out during this thesis has highlighted new promising research lines, either to further enhance the performance of the models proposed, or to apply them to new challenging problems.

Information fusion approaches In order to improve the robustness of an identification system, information from different sources can be combined. These sources can be different matching algorithms [JPC99, NM06] or different fingerprints of the same person [JFR07]. A sample in the database needs to achieve a high score in both matching operations to be considered a positive matching. However, the identification time doubles since each different matching operation is solved separately. If the first matching step is used to select a group of candidates and the second matching process is only applied to those candidates, the time does not increase in such a significant way [PTG+16].

Big data approaches Big data is expected to be one of the main challenges for data mining in the near future [FdRL+14]. MapReduce platforms are the main tool to deal with these scenarios. These platforms not only provide huge computational capabilities but also transparent scalability and failure tolerance. It would be really interesting if a MapReduce platform could be installed in a cluster with GPU devices. This way the GPU-based matching algorithms could scale easily and take advantage of resources from different machines, allowing them to tackle larger databases.

Our on-going research tries to combine these two ideas: information fusion and big data. We have developed a structure that allows Apache Spark [ZCD+12] to use our GPU-based methods. This structure controls how many GPU devices are on each node and how many threads can access them simultaneously. Then a wrapper transforms the data from sections of a Spark RDD to the structures that the CUDA implementation uses.

Two fingerprints per individual and both GPU-based designs are combined in this proposal. The first matching method is applied to one of the fingerprints first and then to the other, and later the second matching method is applied in the same way. The number of candidate individuals is reduced after performing each matching operation. The scores of the same candidate are aggregated as soon as they are computed and this aggregated score is the one used to reduce the number of candidates.

In distributed systems, the amount of information transferred over the network is an important factor. For this reason, the designs of the matching algorithms have been modified. For this system, neither the global neighborhoods of Jiang's algorhtm nor the cylinders of MCC algorithm are precomputed for the database samples and are computed on demand. This change lowers the performance of the GPU-based matching algorithms on a single machine, but the general performance improves since different nodes work in parallel.

Our experimental set up covers a database with up to 4 million individuals (8 million fingerprint samples) and the preliminary results have required an average identification time of less tan 20 seconds on a 5 node cluster with 2 GPU devices per node. The accuracy of the system, without using any threshold for the identification reaches 99% of positive matchings.

New classification approaches The imbalanced data problem associated to one-vs-one (OVO) approaches is not the only approach to improve the classification accuracy. There are new and

more sophisticated techniques based on OVO that could be interesting to try for the fingerprint classification problem.

In the last few years, deep neural networks [LBH15] have also arisen as a very powerful tool to solve many complex classification problems, especially those based on images [KSH+12]. It would be interesting to test the performance of these methods on fingerprint classification.

Both ideas are not exclusive and could be combined using a deep neural network as base learner for the OVO ensemble.

Multi-modal biometrics The use of different biometric features is a natural way of improving the accuracy and reliability of the identification [RNJ06]. By using different features it is possible to provide systems that can keep identifying even in cases of injuries or amputations. However, the time required for the identification will increase since a separate identification is performed per feature used.

Hybrid architectures: The number of computational cores available on CPU devices has increased over the last years. Even when using several threads to provide workload to GPU devices, CPU devices were not used at their full potential. An interesting approach would be to use these extra CPU cores to perform matching operations working on chunks of the database different from the ones computed on GPU devices.

Chapter II

Publications: Published Papers

1 A High Performance Fingerprint Matching System for Large Databases Based on GPU

- P.D. Gutiérrez, M. Lastra, F. Herrera, J.M. Benítez. A High Performance Fingerprint Matching System for Large Databases Based on GPU. IEEE Transactions on Information Forensics and Security 9:1 (2014) 62–71, doi: 10.1109/TIFS.2013.2291220.
  - Status: **Published**.
  - Impact Factor (JCR 2014): 2.408
  - Subject Category: Computer Science, Theory & Methods. Ranking 9 / 102 (**Q1**).
  - Subject Category: Engineering, Electrical & Electronic. Ranking 38 / 249 (**Q1**).

The published paper can be found here:
`http://ieeexplore.ieee.org/abstract/document/6665046/`
A draft is provided for copyright compliance.

A high performance fingerprint matching system for large databases based on GPU

Pablo D. Gutiérrez*, Miguel Lastra, Francisco Herrera and José M. Benítez

***Abstract*—Fingerprints are the biometric features most used for identification. They can be characterized through some particular elements called minutiae. The identification of a given fingerprint requires the matching of its minutiae against the minutiae of other fingerprints. Hence fingerprint matching is a key process. The efficiency of current matching algorithms does not allow their use in large fingerprint databases, to apply them, a breakthrough in running performance is necessary. Nowadays the Minutia Cylinder-Code (MCC) is the best performing algorithm in terms of accuracy. Notwithstanding a weak point of this algorithm is its computational requirements. In this paper we present a GPU fingerprint matching system based on MCC. The many-core computing framework provided by CUDA on NVIDIA Tesla and GeForce hardware platforms offers an opportunity to enhance fingerprint matching. Through a thorough and careful data structure, computation and memory transfer design we have developed a system that keeps its accuracy and reaches a speed-up up to 100.8× compared to a reference sequential CPU implementation. A rigorous empirical study over captured and synthetic fingerprint databases show the efficiency of our proposal. These results open up a whole new field of possibilities for reliable real time fingerprint identification in large databases. Additional details are provided at http://sci2s.ugr.es/fingerprint-GPU.**

***Index Terms*—Fingerprint identification, minutiae, matching, MCC, GPU, CUDA.**

EDICS Categories: BIO-PEVA, BIO-MODA-FIN, BIO-UNIM

I. INTRODUCTION

FINGERPRINTS are the most widely used biometric features in identification tasks thanks to the usability and reliability of systems based on them [1]. Fingerprints are used in a large number of applications, for example, forensic identifications, ID cards, access control, etc. Furthermore, fingerprints are also one of the most studied biometric features. Proposals addressing their acquisition [2], processing [3], classification [4] and matching [5] can be found over the last years.

There are two different kind of issues in this field: verification and identification. Verification systems try to determine if two fingerprints were produced by the same finger with the highest possible reliability. On the other hand, identification systems try to find which fingerprint in a database matches the input fingerprint. Since the identification complexity is much higher than verification, identification systems usually accept an accuracy loss in order to achieve a faster matching process.

The time required to find a fingerprint increases linearly with the size of the fingerprint database. One of the state of the art algorithms for fingerprint identification, the Minutia Cylinder Code (MCC) algorithm [5], takes about 45 milliseconds to perform a single comparison between two fingerprints (matching) [5]. Extrapolating this result, it would take 45 seconds to identify a fingerprint in a database of 1000 individuals. Therefore, the processing time becomes unacceptably long when the size of the database reaches the order of tens or hundreds of thousands. The usual way to improve the performance in these cases is using a threshold to reduce the rate of penetration in the database during the search process. This does not improve the performance of the worst case and it can cause accuracy loss. Our work does address this scalability problem.

Graphics Processing Units (GPUs) have proven to be a very useful tool in the acceleration of computationally intensive algorithms. These devices introduce massive parallelism in the calculations reducing run times in several orders of magnitude. Applications of this technology can be found in different fields as molecular modelling [6], bioinformatics [7] or shallow-water simulation [8].

The goal of this paper is to propose an accurate fingerprint system that is also efficient. To achieve this, we propose the use of GPUs to introduce parallelism in the fingerprint matching process. This permits addressing the scalability problem of the MCC algorithm without losing accuracy, taking advantage of the shorter running times provided by the GPUs. A careful redesign of the algorithm is required so that the maximum performance can be attained out of this architecture. The shorter matching times expected with this new design will make the MCC algorithm more usable in real-world problems, where the systems need to provide a result in a certain short time and the data size increases exponentially. These constraints require to process a high number of fingerprints per second, something that cannot be achieved with traditional techniques.

The robustness, effectiveness and performance of the designed system have been thoroughly tested through a rigorous study over large synthetic and captured fingerprint databases and diverse hardware. The captured fingerprint databases, DB4 [9] and DB14 [10], have been provided by the NIST (National Institute of Standards and Technology). The synthetic fingerprint database was created using SFinGe [11]. This database

P.D. Gutiérrez, F. Herrera and J.M. Benítez are with the Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada, Spain, 18071 (e-mails: pdgp@decsai.ugr.es, herrera@decsai.ugr.es, J.M.Benitez@decsai.ugr.es).

M. Lastra is with the Department of Software Engineering of the University of Granada, CITIC-UGR, Granada, Spain, 18071 (e-mail: mlastral@ugr.es).

This work was supported by the research projects CAB(CDTI), TIN2009-14575 and TIN2011-28488. P.D. Gutiérrez holds an FPI scholarship from the Spanish Ministry of Economy and Competitiveness (BES-2012-060450).

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

has a larger size, up to 100 000 fingerprints, which allowed the scalability study. The performance of the proposal has been studied comparing it to a reference single-thread CPU implementation.

To the best of our knowledge there is no published work where a high performance matching system has been proposed. That is there is no previous proposal alike the one presented in this paper.

Due to the space constraints not every experiment could be included in the paper. Complementary material about the work done for this paper (including the databases) can be found at the URL http://sci2s.ugr.es/fingerprint-GPU.

The rest of the paper is organized as follows: Section II explains the fingerprint recognition problem in detail and analyzes the MCC algorithm. Section III discusses the characteristics of GPUs and GPU programming. Section IV explains the proposed technique to improve the performance using GPUs. In Section V the experiments and results obtained are shown. Finally, Section VI draws the conclusions and future work.

II. Fingerprint identification

The matching process is the main bottleneck of identification systems, as the input fingerprint is compared with every fingerprint in the database. Most of the fingerprint matching algorithms in the literature are based on minutiae, Section II-A introduces the main common characteristics of this kind of algorithms. Section II-B explains the MCC algorithm [5] in detail, which is the basis of the proposed GPU-based system.

A. Minutiae matching

A minutia is a change on the ridges of the fingerprint, usually ridge endings and bifurcations. A minutia is defined by its position, angle and type although other representations like a spectral representation have also been proposed [12]. Minutiae-based algorithms are the most used mainly due to their reliability and the amount of data involved. There are two types of minutiae based matching algorithms: global and local, but most of the algorithms use a combination of both models. Local algorithms define a neighborhood and try to match minutiae from two fingerprints with similar neighbors, while global algorithms use the information of all the minutiae at once.

Algorithms that focus on local matching processes can be divided into two categories depending on how they define the neighborhood of the minutiae:

- Nearest neighbor: The neighborhood of a given minutia is defined by the K closest minutiae.
- Fixed radius: The neighborhood of a given minutia is defined by the minutiae inside an imaginary circle of radius R centered at the minutia.

In the first case, the neighborhood has the same size for every minutia. This makes nearest neighbor algorithms very efficient although, usually, very sensitive to missing and spurious minutiae. The neighborhood size of the fixed radius algorithms depends on the minutiae density and can vary for each minutia. This makes this kind of algorithms more complex than nearest neighbor but more tolerant with respect to missing minutiae.

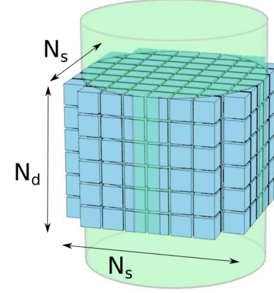


Fig. 1. Structure of a cylinder.

B. Minutia Cylinder-Code algorithm

The Minutia Cylinder-Code algorithm (MCC) [5], is one of the most elaborated algorithms in the fingerprint matching field. MCC uses a combination of local and global matching and also combines the high efficiency of nearest neighbor algorithms with a higher tolerance to deformations achieved by the fixed radius algorithms. Some of the goals of the authors of the MCC algorithms were accuracy, even when processing deformed fingerprints, and interoperability with other algorithms by using standard characteristics (minutiae).

The MCC algorithm uses only the position and orientation of minutiae and ignores the type. The authors base this decision on the assumption that feature extractors can easily mistake this parameter thereby making it unreliable.

A 3-dimensional structure, called cylinder, associated to each minutia is built to store the neighborhood characteristics of each minutia m. This cylinder is centered at the minutia, has a fixed radius, R, and has a height of 2π. As these dimensions are fixed all the cylinders have the same size, as in the nearest neighbor methods.

Each cylinder is discretized into $N_s \times N_s \times N_d$ cells as shown in Figure 1. N_s defines the resolution of the discretized 2D space around minutia m ($N_s \times N_s$) and N_d represents the number divisions applied to the height of the cylinder (2π) which represents angular distance. Each cell has an associated 2D position $p^m_{i,j}$, which represents the center of the cell projected on the cylinder base, and an angle $d\varphi_k$ defined by the height of the cell.

A numerical value $C_m(i,j,k)$ is computed for each cell which stores the sum of the contributions of every minutia m_t in the neighborhood of $p^m_{i,j}$, $N_{p^m_{i,j}}$. The contribution of each minutia in $N_{p^m_{i,j}}$ is based on its location and direction. The spatial contribution is higher when the location is closer to $p^m_{i,j}$ and the directional contribution is higher when its direction is closer to the defined angle $d\varphi_k$. A cell is considered as valid only if $N_{p^m_{i,j}}$ contains enough minutiae inside the convex hull [13] of the fingerprint.

Once the cylinder computation has been completed, a cylinder can be discarded if it does not contain enough useful information (valid cells) or if the number of neighbor minutiae contained in the cell structure is too low. The discarded cylinders are usually associated to minutiae on the edges of the fingerprint as these minutiae are more sensitive to errors and deformations and therefore less reliable.

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

With the cylinder set of the two fingerprints (A and B) to be matched, a local matching process is started. This computation is performed on every pair of cylinders and the results are stored in a matrix. Two cylinders C_a and C_b are considered matchable if the directional difference between the two minutiae is not greater than a certain value, δ_θ, and if the intersection of valid cells of both cylinders is big enough. If the cylinders are matchable, their similarity is defined as:

$$\gamma(a,b) = \begin{cases} 1 - \frac{\|c_{a|b} - c_{b|a}\|}{\|c_{a|b}\| + \|c_{b|a}\|}, & \text{if } C_a \text{ and } C_b \text{ are } matchable \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Where $c_{a|b}$ is a vector that stores the value of the corresponding cell of C_a if the cell is valid in C_a and C_b; $c_{b|a}$ is defined in the same way but storing the value of the cell of C_b.

After the local matching process, a global matching process, called consolidation, is run. This process combines some pairs of the similarity matrix, usually the best, to obtain the final score achieved by the fingerprints.

III. Graphics Processing Units

Recently, graphics processing units (GPUs) have emerged as a parallel computing resource offering hundreds or thousands of processing cores and providing large-scale parallelism on computing platforms. In this section, the characteristics of GPUs and GPU programming are exposed. Section III-A provides a background about these devices and their programming model. Section III-B explains the high and low level structures used by GPUs to introduce parallelism. Finally, Section III-C shows the memory hierarchy present in these devices.

A. Background

GPUs were initially designed to produce 3D graphics in games and CAD applications. Its hardware is responsible for the floating point computations involved in rendering in a highly parallel and efficient way, offloading the computational cost from the CPU. A Single Instruction Multiple Data (SIMD) architecture is used in GPU devices to introduce parallelism.

The use of GPUs to run general purpose programs started in an early stage but developers had to map scientific calculations onto problems that could be represented by vertices and pixels, until NVIDIA [14] launched CUDA [15] in 2006. NVIDIA CUDA is the hardware/software architecture that allows the use of NVIDIA GPUs as general purpose computation devices, exposing their parallel processing nature to non-graphics-specialized developers.

NVIDIA CUDA provides high level abstraction interfaces that make GPUs more easily programmable from the numeric software developer's point of view without the need for specialized graphics terminology.

The hardware side of the NVIDIA CUDA architecture presents the GPU as an array of streaming multiprocessors and the software side is an extension of the C programming language (CUDA C) that exposes the GPU as a parallel co-processor.

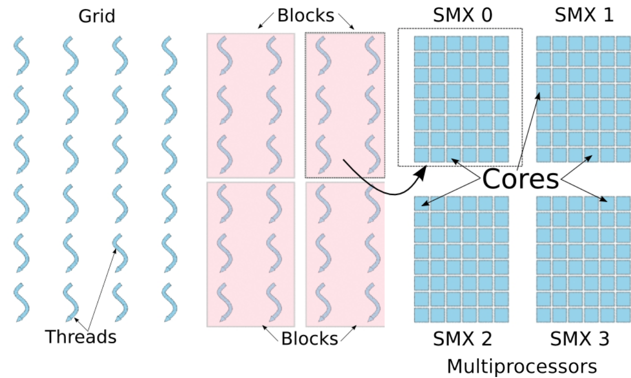


Fig. 2. Threads, blocks and multiprocessors. Each block is run on the same multiprocessor.

B. Grids, blocks, threads and warps

Programs that are run on a GPU are called kernels and contain the set of instructions that will be run across a set of computing cores. GPU cores are grouped at the hardware level into stream multiprocessors (SMX).

Threads are instances of a kernel and share therefore the same code but each thread can work on a different dataset. Threads are grouped into blocks (see Figure 2); the threads of the same block:

- are always run on the same multiprocessor (see Figure 2)
- share a small high speed memory area called *shared memory*.

All the thread blocks of the same kernel form a grid which contains all the threads run to complete a certain task.

Each thread is identified by its thread index and the block index that contains the thread. These indices can have up to three dimensions.

At a lower level, threads are run in sets called warps. Threads of the same warp should be running the same instruction (in parallel) at the same time because in case of code divergence serialization occurs producing an important performance penalty. Warps are run in parallel and warp switching is very fast because all the register data of each thread of each warp is kept and not overwritten because of the context switch. Memory latencies are hidden by switching to a different warp when the current one requires any memory access.

C. Memory hierarchy

As on many architectures, registers are the fastest type of memory and the number of registers needed by a kernel has an impact on the level of parallelism achieved.

On the next level, each multiprocessor has a 64KB area which is used as shared memory for threads of the same block and also as L1 cache. Developers might choose from several shared memory/cache predefined distributions.

Global memory is the slowest memory type and it is accessible by all threads of all blocks. Its contents are automatically cached in the L1 cache. This can also be done by the developer by explicitly loading data from global memory to the shared

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

data area. In general, when data is transferred to the GPU it is transferred to global memory.

The less resources (registers and shared memory) required by the threads the more threads can be kept ready to be run or active (high occupancy). When a kernel function achieves a high occupancy of the device, the memory latencies can be hidden more effectively improving the performance obtained.

IV. A GPU-BASED ALGORITHM FOR FINGERPRINT IDENTIFICATION

This section presents the GPU-based design of both the required data structures and computational steps of the MCC algorithm. Several performance enhancements and scalability considerations are also treated in this section. Section IV-A focuses on the adaptation of the different data structures to the GPU. Section IV-B details the distribution of each calculation of the algorithm on the GPU. Section IV-C shows specific enhancements for identification systems. Finally, Section IV-D focuses on the scalability problem.

A. Data Structures

Data structures are one of the key issues when designing GPU based programs because data organization has a big impact on the resulting performance. Coalesced memory accesses is one of the factors which reduces memory access times by allowing several memory operations issued by different cores or GPU threads to be combined into one access.

The use of coalesced memory accesses does also have an impact on the design of the computational structure of the GPU program but suitable data structures are the basis to achieve this goal.

As stated in Section II-B, the MCC algorithm requires the storage of data related to every minutia (position (x, y), orientation θ and validity v) and the cylinder cells associated to each minutia (each cell stores one floating point value). For each fingerprint, minutiae data is stored as float4 elements. Float4 data types are native in GPU programs and represent floating point 4-tuples that offer the optimum memory alignment. This representation allows retrieving all the information of a minutia with a single memory access. It also reduces the amount of memory transfer operations between the host and the GPU and increases the throughput achieved.

Cell data is also packed in 4-tuples and stored in a lineal array. A fingerprint database is therefore constructed using two lineal arrays. The first one contains the minutiae data of each fingerprint and the second one the values of the cells associated to each minutia, in the same order as they are stored in the first array.

B. Computation

The computational model offered by CUDA requires the distribution of the input fingerprint data structures computation and the subsequent matching process into a set of threads. These threads have to be grouped into blocks that share a small common memory area and that are run on the same GPU multiprocessor. The mapping chosen for each computation step will be detailed in the following sections.

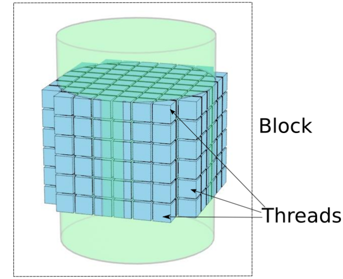


Fig. 3. Computation mapping $N_s = 8$.

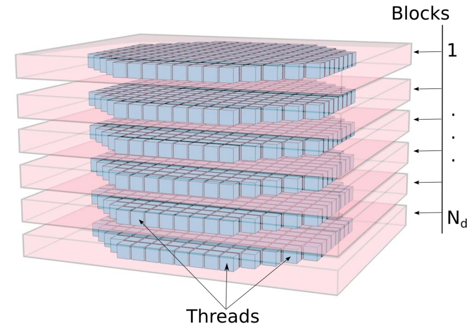


Fig. 4. Computation mapping $N_s = 16$.

1) Cylinder generation: The fingerprint to be compared to the fingerprint database has to be represented in terms of cylinders representing minutiae and cylinder cells. The number of cells per cylinder ($N_s \times N_s \times N_d$) is one of the MCC algorithm parameters. For this work the two configurations proposed in [5] have been chosen: $(N_s = 8, N_d = 6)$ and $(N_s = 16, N_d = 6)$.

For the first configuration, the total number of cells per cylinder is below the maximum number of threads per block (which is limited by the device to 1024 since the Fermi GPU generation) and allows the computation of one cylinder per thread block. All the computations associated to a cell (Section II-B) are assigned to one thread. Figure 3 shows the proposed scheme.

The main advantage of mapping the computations related to one cylinder to the same thread block is that the process of determining the validity of a cylinder, which is a reduction process, will be run on the same multiprocessor and can be fully implemented using the block shared memory. The reduction process computes the count of the valid cells of the cylinder.

When the number of cells per cylinder is increased by setting $N_s = 16$, keeping the *cell computation*-thread mapping requires the process to be split over several blocks. In our system, cell layers were assigned to thread blocks. As each cylinder has N_d layers, the same number of threads blocks is created. Figure 4 shows a graphic scheme of the computational structure.

Dividing this process into several blocks requires and additional reduction step to compute the validity of a cylinder in

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

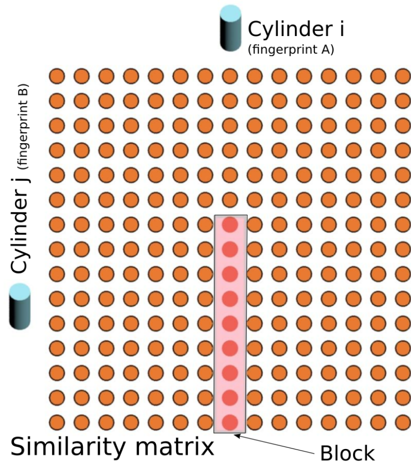


Fig. 5. Similarity matrix computation.

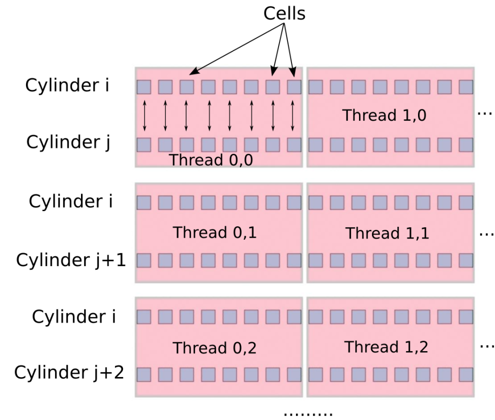


Fig. 6. Similarity matrix computational scheme.

terms of the partial validity values computed for each layer.

2) Fingerprint matching: The fingerprint matching process requires comparing an input fingerprint to a set of fingerprints stored in a database. This process can be split into a set of one to one matching processes. Each of these comparisons or matching processes has been adapted to the parallel GPU architecture.

In order to decide whether a fingerprint A matches another fingerprint B, first a local similarity approach is applied by computing a similarity value for each cylinder pair (i, j) where i is the index of a cylinder of fingerprint A and j the index of a cylinder of fingerprint B. This leads to the computation of a similarity matrix where each thread block performs a set of (i, j) similarity computations (see Figure 5), following the Equation 1.

Each thread of a block will perform the comparison of a *bucket* of cells of each cylinder. The threads are organized into two dimensions inside each block. The first index is used to identify which cells are computed by the thread while the second dimension is used to identify the cylinder inside the block, as it is shown in Figure 6. Depending on the value of parameter N_s and on the type of consolidation different numbers of threads per cylinder and cylinders per block have been used in order to achieve the maximum performance.

After the local similarity computation, as a second step, a global similarity is carried out. We have selected two techniques among the ones proposed in [5]:

- *Local Similarity Sort (LSS)*. This technique sorts the local similarity values of the (i, j) pairs and computes the average of the best n values. The value of n is defined by the number of valid cylinders of each fingerprint.
- *Local Similarity Sort with Relaxation (LSSR)*. This technique first carries out an LSS process, then selects the n_R positions with the highest values and performs an iterative process with them. n_R is greater than n, so, when the iterative process ends, the algorithm selects the best n elements out of the n_R elements and computes their average.

This average is the final result (score) of the matching process between two fingerprints.

The iterative relaxation process compares pairs of minutiae from both fingerprints, as shown in Equation 2.

$$\lambda_t^i = w_R \cdot \lambda_t^{i-1} + (1 - w_R) \cdot \left(\sum_{\substack{k=1 \\ k \neq t}}^{n_R} \rho(t, k) \cdot \lambda_t^{i-1} \right) / (n_R - 1) \quad (2)$$

where λ_t^i is the value of item t of the set of n_R elements selected at iteration i, w_R is a parameter of the algorithm and $\rho(t, k)$ is a function that measures the compatibility of two pairs of minutiae. This function uses the positions of the minutiae, its orientation and the angles that they form to measure how likely it is for two minutiae from a fingerprint to be the same as the other two from another fingerprint. The initial value of λ_t is the similarity value of the element.

Considering the usual values of n and n_R, GPU sorting methods do not offer any significant speed-up as this stage. However, the computation of each iteration of the relaxation process can be parallelized to improve its performance.

The relaxation kernel is set up with n_R thread blocks to compute a whole iteration of this process at once. Each thread of the kernel computes a set of compatibility tests of one element with another and then performs a reduction operation on shared memory that enables the last thread to obtain the result of the whole iteration. This way the number of threads per block can be fixed independently of parameter n_R that changes depending on the number of minutiae of the fingerprint.

Once the iterative process has finished, the results are returned to the CPU to perform the selection of the n best values and compute the score.

C. Performance enhancements for identification systems

A fingerprint identification system's goal is not to perform one to one fingerprint matches but to find the matching fingerprint in a database to match an input fingerprint. In this section, the reduction of GPU idle periods and the packaging of several matching processes into one will be treated. Both

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

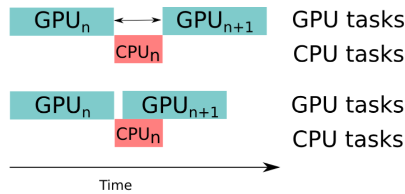


Fig. 7. Reducing the GPU idling periods.

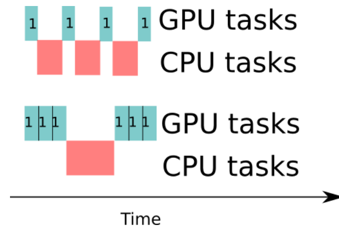


Fig. 8. Packaging of several matching processes.

optimizations allow processing more input fingerprints per time unit with respect to the initial system design.

As the CPU performs its part of the fingerprint matching process, the GPU is idling and during that period it could already start the computations for the next matching process. To achieve this in our system, the GPU is supplied with requests from two CPU threads. The two threads run the GPU matching process of the input fingerprint first with database fingerprints i and $i+1$. At the next iteration step with database fingerprints $i+2$ and $i+3$ and so on. This reduces GPU idle time. It is important to state that this enhancement does not mean running two fingerprint matching processes in parallel at the GPU level because the GPU will still dispatch sequentially the tasks (see Figure 7).

Providing enough workload to the GPU is also essential to obtain the highest performance. Moreover, replacing many small workload packages by fewer bigger ones does provide a performance boost in many cases. This fact led to a redesign of part of the matching process which replaced the *one to one* matching processes by sets of *one to many* matching processes. This results in each kernel call processing the input fingerprint and a set of the fingerprints stored in the database (see Figure 8). It also allows the relaxation kernel to process different fingerprints in the same block organizing the threads into two dimensions. In an analogous way to the similarity computation, the first index is used to identify the compatibility test to compute and the second to identify the fingerprint.

Grouping the matching processes and reducing the GPU idling time are complementary enhancements which have been used in our system providing a speed-up of over $2\times$ with respect to the ad-hoc GPU algorithm.

D. Scalability

1) Multi-GPU: Multi-GPU configurations are becoming more mainstream due to the increasing availability of several full speed PCI Express sockets on computer motherboards. In our work we have also addressed these platforms and the solution that has been designed offers almost a lineal speed-up increase with respect to the number of GPUs.

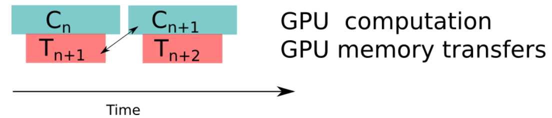


Fig. 9. Asynchronous memory transfers.

The two strategies that can be chosen are to use the GPUs to collaborate on the same task or to perform several isolated tasks in parallel. In the fingerprint matching process, the tasks are of two types: input fingerprint processing and the matching process.

By using several GPUs in a collaborative way it is possible to get the full acceleration of the computational task at which they are applied but memory transfers from one GPU to the rest and also, the reduced amount of workload available to each GPU can be and important burden to reach the maximum speed-up. By assigning different tasks to each GPU it is easier to achieve good scalability but the time required for a single task is not reduced.

In our system, a setup with two GPUs, both GPUs are used in a collaborative way to process the input fingerprint. This allows the reduction of the time required for this process. Each GPU is assigned the computation of half of the minutiae to be converted into cylinders. On the other hand, for the matching process both approaches were tested: each GPU may perform half of the matching process or each GPU may compute the matching score of the input fingerprint with different fingerprints of the database.

2) Fingerprint database size: When an input fingerprint is to be matched to a fingerprint database, if the database can be stored in the GPU's memory the process of transferring this database from main memory (or disk) can be considered part of the system initialization process and is not critical. If, on the other hand, really large fingerprint databases that do not fit in the GPU global memory are to be considered, the transfer mentioned above can become a bottleneck.

To overcome this problem we have used asynchronous memory transfers (from the host computer) that can be performed in parallel with the matching process. The database is divided into chunks and therefore processed chunk-wise. While the input fingerprint is matched to chunk i (computation associated to chunk i), in parallel, chunk $i+1$ is loaded into the GPU memory (transfer of chunk $i+1$). This allows processing arbitrary sized fingerprint databases without adding any memory transfer overheads (see Figure 9).

Synchronization events are used to prevent the computation process overtaking the memory transfer process and causing a race condition, although the chunk size has been carefully chosen to avoid delays caused by this synchronization requirement.

V. Experimental Results

An exhaustive empirical evaluation has been carried out in this work to study the performance of the proposed GPU-

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

based algorithm. This evaluation has been performed on different hardware devices and using different kinds of databases in order to test the robustness and effectiveness of the system. Section V-A introduces the performed experiments. Section V-B describes the hardware where the experiments has been run and SectionV-C describes the databases that were used. Section V-D shows the obtained results. Finally, Section V-E concludes with an analysis of the results.

A. Experiments

The accuracy obtained by the GPU based algorithm is the same as the one obtained by the CPU implementation. The accuracy of the later has been tested on the FVC-onGoing benchmark [16]. As a reference indicator, with our implementation, we have achieved an EER of 0.62% against a 0.57% of MCC-baseline. These are certainly very minor differences (The complete benchmark results can be consulted in the complementary material website). The GPU-based algorithm and the CPU implementation have been tested on databases that include plain and rolled as well as captured and synthetic fingerprints. These databases have different sizes that range from 1000 to 100 000 fingerprints and different average number of minutiae per fingerprint. This allows the study of the algorithms behavior with large databases.

A set of random fingerprints of each database has been selected and identified against the database, measuring the average time to perform an identification. The speed-up factor has been used to represent the improvement obtained. This factor is computed as the ratio of the single-thread CPU implementation running time over the GPU implementation running time.

The accuracy obtained by the GPU based algorithm is the same as the obtained by the CPU implementation. The different organization of the operations and the limited resolution of both CPU and GPU floating point representation introduce rounding differences in the matching score. Nevertheless, these differences do not affect the identification process.

B. Hardware

Two different types of GPUs have been used in the experiments:

1) Tesla GPU, an NVIDIA Tesla M2090 with 512 CUDA cores, Fermi architecture and 6GB of memory.
2) GTX GPU, an NVIDIA GeForce GTX 680 with 1536 CUDA cores, Kepler Architecture and 2GB of memory.

Two Tesla GPU are installed in a server with an Intel Xeon E5-2630 processor, while the GTX device is a desktop device coupled with a commodity PC. Both computers have 24GB of RAM memory with similar characteristics. On the other hand, the reference CPU running times of the results were computed on a server with an Intel Core i7 930 processor.

C. Datasets

The GPU-based algorithm has been tested on different databases from different sources, including captured and synthetic fingerprints. These databases have different sizes and types of fingerprints.

TABLE I
KERNEL THREADS CONFIGURATION ON TESLA DEVICE

	Similarity		Relaxation	
	Threads per cylinder	**Cylinders per block**	**Threads per compatibility**	**Fingerprints per block**
Ns 8; LSS	4	64	-	-
Ns 16; LSS	32	10	-	-
Ns 8; LSSR	4	32	8	24
Ns 16; LSSR	32	10	8	24

TABLE II
KERNEL THREADS CONFIGURATION ON GTX DEVICE

	Similarity		Relaxation	
	Threads per cylinder	**Cylinders per block**	**Threads per compatibility**	**Fingerprints per block**
Ns 8; LSS	4	32	-	-
Ns 16; LSS	8	32	-	-
Ns 8; LSSR	4	32	8	16
Ns 16; LSSR	8	32	16	8

- The DB4 database has been provided by the NIST (National Institute of Standards and Technology) [9] and contains 2 000 pairs of rolled fingerprints. The average number of minutiae is 135.84 and the maximum 275. It contains two captures of each fingerprint.
- The DB14 database has also been provided by the NIST [10] and contains 27 000 pairs of rolled fingerprints and we used the first 19 000. The fingerprints of this database have the highest number of minutiae of all the databases that we have tested: an average of 206.9 and a maximum of 610. It also contains two captures of each fingerprint.
- A SFinGe [11] based database was synthetically generated using the SFinGe software with 100 000 plain fingerprints which were generated following the Galton-Henry classification [1], [17]. The average number of minutiae is 40.69 and the maximum 89. Two captures of each fingerprint were created. This database and more information about the parameters used in its creation are available in the website associated to this paper.

D. Empirical results

This section provides the experiments and results carried out with the datasets mentioned above. The results include running times using LSS and LSSR global similarity processes and two different values for the N_s parameter ($N_s = 8$ and $N_s = 16$). The value of the kernel threads configurations are shown in Tables I and II. All the running time values are expressed in milliseconds. These results and some extra figures can be found at the complementary material website.

1) Single GPU results: Tables III to VIII show the results obtained by the GPU-based method using only one GPU on the different databases.

- DB4 database: as it contains two captures of each fingerprint, the first one is used to build the database while the second is used as input for the algorithm. The experiment performed consists of a series of one hundred identifications of randomly selected fingerprints. 200 fingerprints have been grouped in the matching process with $N_s = 8$, 250 with $N_s = 16$ on the Tesla GPU and 50 on the GTX

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

TABLE III
DB4 RESULTS USING A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	23 914.0	692.0	34.6
Ns 16; LSS	93 539.0	1756.0	53.3
Ns 8; LSSR	102 474.1	2581.5	39.7
Ns 16; LSSR	174 963.7	3530.3	49.6

TABLE IV
DB4 RESULTS USING A GTX DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	23 914.0	603.6	39.6
Ns 16; LSS	93 539.0	2004.5	46.7
Ns 8; LSSR	102 474.1	1810.7	56.6
Ns 16; LSSR	174 963.7	3187.8	54.9

TABLE V
DB14 RESULTS USING A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	633 546.0	25 415.9	24.9
Ns 16; LSS	2 205 491.2	42 103.0	52.4
Ns 8; LSSR	2 269 426.8	113 308.4	20.0
Ns 16; LSSR	3 749 775.2	105 424.8	35.6

TABLE VI
DB14 RESULTS USING A GTX DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	633 546.0	21 854.6	29.0
Ns 16; LSS	2 205 491.2	49 868.2	44.2
Ns 8; LSSR	2 269 426.8	80 559.8	28.2
Ns 16; LSSR	3 749 775.2	91 140.5	41.1

TABLE VII
SFINGE RESULTS WITH A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1998.8	30.2
Ns 16; LSS	239 241.4	4763.0	50.2
Ns 8; LSSR	141 631.1	3568.6	39.7
Ns 16; LSSR	331 559.1	6486.2	51.1

TABLE VIII
SFINGE RESULTS WITH A GTX DEVICE

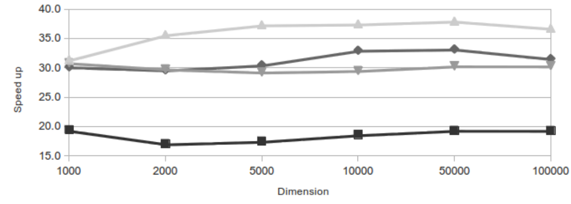
	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1796.3	33.6
Ns 16; LSS	239 241.4	5888.1	40.6
Ns 8; LSSR	141 631.1	2775.4	51.0
Ns 16; LSSR	331 559.1	6994.6	47.4

GPU. Tables III and IV show the average result of one identification.

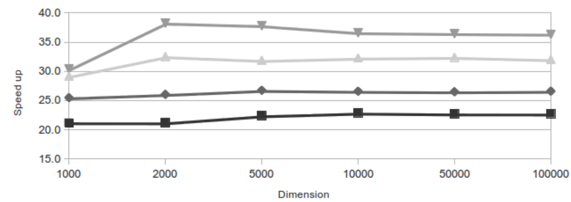
- DB14 database: as it was done with the DB4 dataset, the first scan of each fingerprint is stored in the database and the second is used as input. One hundred identifications have been performed in this case. 50 fingerprints have been grouped in the matching process with $N_s = 8$ and 25 with $N_s = 16$. Tables V and VI show the average result of one identification.

It can be observed that the higher the average minutiae per fingerprint the less fingerprints need to be grouped to achieve the optimum performance.

- SFinGe based database: one hundred input fingerprints were randomly selected to be identified but in this experiment we used different fingerprint database sizes (ranging from 1000 to 100 000) to study how the GPU based algorithm scaled with the database size. 600 fingerprints have been grouped in the matching process with $N_s = 8$ and 150 with $N_s = 16$. Tables VII and VIII show the average result of one identification in the complete database. Figure 10 shows the evolution of the speed-up as the size of the database increases.



(a) Tesla device



(b) GTX device

Fig. 10. Speed-up evolution with the SFinGe database.

TABLE IX
SFINGE RESULTS USING TWO TESLA DEVICES COLLABORATING

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1374.8	43.9
Ns 16; LSS	239 241.4	3250.5	73.6
Ns 8; LSSR	141 631.1	2932.7	48.3
Ns 16; LSSR	331 559.1	4689.5	70.7

TABLE X
SFINGE RESULTS USING TWO TESLA DEVICES WORKING IN PARALLEL

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1024.0	59.0
Ns 16; LSS	239 241.4	2393.9	99.9
Ns 8; LSSR	141 631.1	1839.5	77.0
Ns 16; LSSR	331 559.1	3288.6	100.8

2) MultiGPU results: The experiments using two Tesla GPUs were performed with the SFinGe database. As it was mentioned in Section IV-D1, there are two ways of using several GPUs: collaborating in the same matching process (Table IX) and performing different matching processes independently (Table X). 1200 fingerprints have been grouped in the matching process with $N_s = 8$ and 300 with $N_s = 16$ when

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

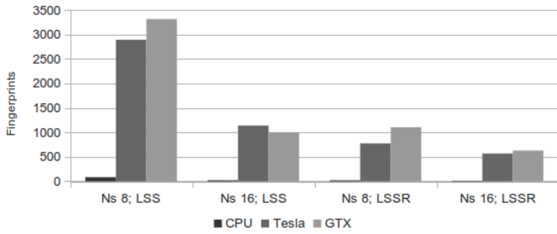


Fig. 11. Fingerprints per second with the DB4 database.

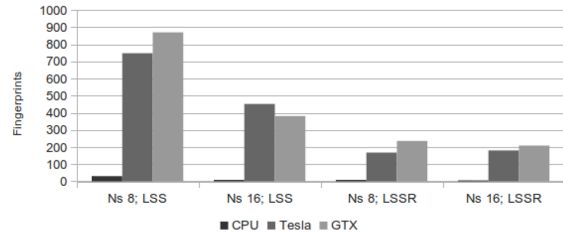


Fig. 12. Fingerprints per second with the DB14 database.

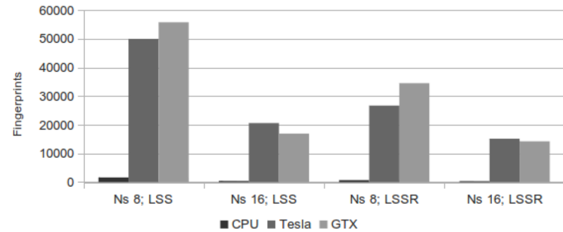


Fig. 13. Fingerprints per second with the SFinGe database.

the two GPUs collaborate and 100, in both configurations, when they work in parallel.

E. Analysis of the results

The proposed GPU algorithm achieves a speed-up between 20× and 53.3× for the Tesla GPU and between 28.2× and 56.6× for the GTX GPU. This improvement in the performance enables to increase the number of fingerprints that can be evaluated in a predefined time, which is a critical factor in any identification system (especially in large scale ones). Figures 11 to 13 show the estimated number of fingerprints that can be processed per second for each database. It is possible to process up to 55 700 fingerprints per second on a single GPU while the CPU algorithm only allows processing up to 1600. In the worse case the CPU algorithm only processes 5 fingerprints per second while the GPU algorithm achieves 170 fingerprints in the same time.

The speed-up increases when several GPUs are used and the results show that the use of different GPUs performing different matching processes is more efficient than the use of different GPUs collaborating on the same one. When the GPUs are collaborating they have to exchange data to perform different operations and this makes the calculation slower than performing different matching operations on each GPU. When the GPUs work in parallel the speed-up is almost doubled, reaching up to 100.8×.

The speed-up achieved using the LSSR consolidation is higher than using the LSS consolidation because the LSSR consolidation is a very slow process in the CPU as it depends quadratically on parameter n_R. In the proposed algorithm this parameter only defines the number of blocks in the kernel call and as the kernel runs the blocks in parallel, the influence of this value is smaller than in the CPU implementation.

As the experiments with the SFinGe database show, the speed-up remains almost constant independently of the database size. The use of asynchronous transfers when copying the database to the GPU (introduced in Section IV-D2) makes this possible.

Grouping several fingerprints depending on the value of the N_s parameter allows in turn achieving a higher level of occupancy in the GPU which increases the performance and reduces the difference (in terms of speed-up) between the two algorithm configurations that were tested: $N_s = 8$ and $N_s = 16$.

The differences between the results of the different GPU devices, Tesla and GTX, are explained by their different architecture. The Fermi architecture of Tesla devices uses faster cores than the Kepler architecture of the GTX device which has more but slower cores (in terms of clock rate) and has more registers per multiprocessor. Furthermore, Kepler architecture allows to have more blocks active per multiprocessor than the Fermi architecture. These differences lead to a higher occupancy value and make the GTX device outperform the Tesla device for the $N_s = 8$ configurations. However, this increase of the occupancy it is not enough to beat the higher clock rate of the Tesla for the $N_s = 16$ configurations.

It is also important to state, as it was commented in Section V-A, that the GPU algorithm achieves the same accuracy rate than the CPU implementation but the different organization of some floating point operations and also the differences between the computing architecture of each device causes minor score differences. However, these differences are negligible and do not affect to the identification process.

VI. Conclusions

We have presented an efficient GPU based fingerprint method using the MCC algorithm [5]. Our proposal implies an effective design of the parallel algorithm with the inclusion of smart techniques to overlap memory transfers with computation as well as packaging sets of independent identifications.

We obtained speed-up ratios up to 100.8× with respect to a single-thread CPU implementation. We also showed that our system has no scaling issues when the fingerprint database size increases and that the speed-up ratios are highly independent of the fingerprint type and the mean number of minutiae per fingerprint. Furthermore, our proposal is able perform an identification in a reasonable time for large databases, processing up to 55 700 fingerprints per second with a single GPU, maintaining the accuracy of the CPU implementation and making the MCC algorithm usable in real-world situations.

As future work, we plan to study other aspects of the fingerprint identification, as the reduction of database fingerprint

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

candidates by using fast preselection methods [18]. We also plan to study the use of several fingerprints from the same person in the identification process to improve the accuracy [19].

REFERENCES

[1] D. Maltoni, D. Maio, A. Jain, and S. Prabhakar, *Handbook of fingerprint recognition*. Springer-Verlag New York Inc, 2009.

[2] H. Choi, K. Choi, and J. Kim, "Mosaicing touchless and mirror-reflected fingerprint images," *Information Forensics and Security, IEEE Transactions on*, vol. 5, no. 1, pp. 52 –61, march 2010.

[3] F. Turroni, D. Maltoni, R. Cappelli, and D. Maio, "Improving fingerprint orientation extraction," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3 PART 2, pp. 1002–1013, 2011.

[4] K. C. Leung and C. H. Leung, "Improvement of fingerprint retrieval by a statistical classifier," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 59–69, 2011.

[5] 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.

[6] M. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. Beberg, D. Ensign, C. Bruns, and V. Pande, "Accelerating molecular dynamic simulation on graphics processing units," *Journal of Computational Chemistry*, 2009.

[7] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, "High-throughput sequence alignment using graphics processing units," *BMC Bioinformatics*, 2007.

[8] M. Lastra, J. M. Mantas, C. Ureña, M. J. Castro, and J. A. García-Rodríguez, "Simulation of shallow-water systems using graphics processing units," *Math. Comput. Simul.*, vol. 80, no. 3, pp. 598–618, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.matcom.2009.09.012

[9] C. Watson and C. Wilson, "Nist special database 4," National Institute of Standards and Technology, Tech. Rep., 1992.

[10] C. Watson, "Nist special database 14," National Institute of Standards and Technology, Tech. Rep., 1993.

[11] R. Cappelli, D. Maio, and D. Maltoni, "Synthetic fingerprint-database generation," in *Proc. 16th Int'l Conf. Pattern Recognition*, vol. 3, 2002, pp. 744 – 747.

[12] H. Xu, R. Veldhuis, A. Bazen, T. Kevenaar, T. Akkermans, and B. Gokberk, "Fingerprint verification using spectral minutiae representations," *Information Forensics and Security, IEEE Transactions on*, vol. 4, no. 3, pp. 397 –409, sept. 2009.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Springer, 1985.

[14] "Nvidia," http://www.nvidia.com/.

[15] "Cuda," http://www.nvidia.com/object/cuda_home_new.html.

[16] "Fvc-ongoing benchmark," http://biolab.csr.unibo.it/fvcongoing/UI/Form/Home.aspx.

[17] C. Wilson, G. Candela, and C. Watson, "Neural network fingerprint classification," *Journal of Artificial Neural Networks*, vol. 1, no. 2, pp. 203–228, 1994.

[18] R. Cappelli, M. Ferrara, and D. Maio, "Candidate list reduction based on the analysis of fingerprint indexing scores," *Information Forensics and Security, IEEE Transactions on*, vol. 6, no. 3, pp. 1160 –1164, sept. 2011.

[19] S. Prabhakar and A. Jain, "Decision-level fusion in fingerprint verification," *Pattern Recognition*, vol. 35, no. 4, pp. 861–874, 2002.

Pablo David Gutiérrez 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 high performance and general purpose GPU computing.

Miguel Lastra received his M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Spain. He is an Assistant Professor with the Software Engineering Department at the University of Granada since 2000.

He is a former researcher of the Computer Graphics group and is currently a member of the "Soft Computing and Intelligent Information Systems" research group. His current research interests include high performance computing using GPUs, biometrics and data analysis.

Francisco 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 been the supervisor of 28 Ph.D. students. He has published more than 240 papers in international journals. He is coauthor of the book Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases" (World Scientific, 2001).

He currently acts as 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, IFSA 2013 Fellow, 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, big data, cloud computing, data preparation, instance selection and generation, imperfect data, fuzzy rule based systems, genetic fuzzy systems, imbalanced classification, knowledge extraction based on evolutionary algorithms, memetic algorithms and genetic algorithms, biometrics.

José Manuel Benítez is an Associate Professor at Dept. Computer Science and Artificial Intelligence (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) <sci2s.ugr.es> and head of the "Distributed Computational Intelligence and Time Series" research lab (DICITS)<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 modelling, distributed/parallel computational intelligence, high performance computing, cloud computing, big data, 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.

The published paper can be found here:
http://ieeexplore.ieee.org/abstract/document/6665046/
This draft is provided for copyright compliance

2 Fast fingerprint identification using GPUs

- M. Lastra, J. Carabaño, P.D. Gutiérrez, J.M. Benítez, F. Herrera. Fast fingerprint identification using GPUs. InformationSciences, 301 (2015) 195–214. doi: 10.1016/j.ins.2014.12.052
  - Status: **Published**.
  - Impact Factor (JCR 2015): 3.364
  - Subject Category: Computer Science, Information Systems. Ranking 8 / 144 (**Q1**).

The published paper can be found here:
`http://www.sciencedirect.com/science/article/pii/S0020025515000043`
A draft is provided for copyright compliance.

Fast fingerprint identification using GPUs

Miguel Lastra[a,*], Jesús Carabaño[b,c], Pablo D. Gutiérrez[c], José M. Benítez[c], F. Herrera[c]

[a] *Depto. Lenguajes y Sistemas Informáticos. E.T.S. Ingeniería Informática y Telecomunicación. CITIC-UGR. Universidad de Granada.*
[b] *Department of Information Technologies. Åbo Akademi University. Finland*
[c] *Depto de Ciencias de la Computación e Inteligencia Artificial. E.T.S. Ingeniería Informática y Telecomunicación. CITIC-UGR. Universidad de Granada.*

Abstract

Fingerprints are widely used in a variety of biometric identification systems. The fingerprint matching process is a processing step whose computational requirements limit the size of the fingerprint database that can be dealt with.

Fingerprint matching algorithms based on minutiae are one of the most relevant families of biometric identification techniques. The scalability of these models is determined not only by the number of fingerprints but also the number of minutiae per fingerprint. Therefore, processing millions of fingerprints per second requires being able to process hundreds of millions of minutiae per second.

In this paper we present a new design of the minutiae based fingerprint matching algorithm presented by Jiang *et al.* specifically created for GPU based massively parallel architectures. The parallel design allows speed-up ratios of up to 15 with one GPU compared to multi-threaded CPU implementations, and up to 54 using several GPUs in parallel and fingerprint processing rates of between 300 000 and 1 500 000 fingerprints per second.

Keywords: Fingerprint matching, identification, GPU, CUDA

1. Introduction

The fingerprint matching process is the keystone of many biometric identification environments [24]. Different aspects of the fingerprint identification systems such as acquisition [2], classification [32] and matching [5] have been widely studied, but designing systems able to produce reliable real-time results when handling large databases with several million fingerprints is still an open problem.

There is a wide range of biometric features related to fingerprints, such as minutiae and orientation fields [23], and other hand areas that are used in identification systems: finger veins [35], finger knuckles [20], palmprint [7] and many more. Many identification techniques focus on extracted features but others use image based operations [17]. This is of course an immense area of research and only a few samples of recent research work in this field are provided.

Minutiae based fingerprint matching algorithms represent each fingerprint as a set of elements called minutiae which are extracted from the fingerprint ridges. These minutiae are recorded at singular points such as ridge endings, bifurcations, sharp direction changes, etc. Each minutia is represented by its 2D position, direction and its type. The fingerprint matching process for this kind of algorithm consists of deciding whether the minutiae set of the input fingerprint matches the minutiae set of any of the template fingerprints stored in a database. The main challenge in this process is being able to handle the deformations, rotations and translations which occur as a result of different conditions, or the different capturing devices

*Corresponding author. Tel: +34 958246144
Email addresses: mlastral@ugr.es (Miguel Lastra), jcaraban@abo.fi (Jesús Carabaño), pdgp@decsai.ugr.es (Pablo D. Gutiérrez), J.M.Benitez@decsai.ugr.es (José M. Benítez), herrera@decsai.ugr.es (F. Herrera)

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

used when fingerprints are captured while providing very reliable results. Minutiae based algorithms are one of the most widely used techniques in biometric identification systems because of the quality of the results and also because their associated acquisition process is less intrusive than those associated with other biometric features.

This work focuses on creating a highly efficient fingerprint matching technique able to tackle millions of fingerprints in a reasonable time (ideally in tenths of a second). The goal is to overcome the limits, in terms of efficiency and cost, imposed by existing CPU based solutions [29].

The minutia based fingerprint matching technique presented by Jiang *et al.* [19] is composed of a local structure matching phase, which accounts for rotations and translations, and a global matching phase to reduce the number of false positive results and to increase the accuracy of the algorithm.

The use of parallel architectures has enabled us to process large amounts of data in a reasonable time. Graphics Processing Units (GPUs) provide massive parallelism and are universally used as almost every computer has one. These devices have been applied to several problems with intense floating point calculations such as bioinformatics [31], shallow-water simulation [21] and also fingerprint identification [3][16]. Some approaches have explored the idea of using FPGAs (Field Programmable Gate Arrays) for fingerprint matching tasks [30] [18] but without state-of-the-art matching techniques and hardware or focusing on low cost proposals that could be used in embedded systems for small-scale scenarios more suited to verification systems [11][12].

In this work, we propose a massively parallel redesign of the algorithm created by Jiang *et al.* suited to GPU based architectures. The process for creating the new fingerprint matching system requires dealing with different non trivial tasks such as correctly identifying the sources of parallelism, creating an efficient workload mapping between computational tasks and parallel computing elements to fully utilize the computational power offered by GPUs, avoiding any GPU idling periods by using asynchronous memory transfers and overlapping the processing of different tasks (and memory transfers) and avoiding bottlenecks such as those produced by memory allocations.

The speed-up factor of the many-core approach with respect to traditional multi-core systems is obtained while maintaining the same accuracy of the original algorithm. The rates of over 300 000 fingerprint matching operations per second obtained by our proposal with one GPU and up to 1 500 000 matching operations per second using four GPUs, matching the performance of a cluster with 12 dual processor nodes with 12 physical cores per node, allows the presented system to be used as part of a hybrid model to achieve the right balance of accuracy and efficiency by combining it with other, slower but more accurate techniques such as the MCC fingerprint matching algorithm [5]. These ideas will be discussed in Section 6.6.

The paper is structured as follows: Section 2 describes fingerprints as biometric characteristics and their importance in the identification systems domain, Section 3 describes the original fingerprint matching algorithm on which this work is based, Section 4 presents an introduction to GPU based general purpose programming and its application to the fingerprint identification process, in Section 5 a detailed description of the GPU based algorithm redesign is provided and Section 6 shows the results of the different experiments that have been carried out, together with a comparison with parallel CPU implementations and finally, a hybrid fingerprint matching model is discussed with a view to obtaining a balance between performance and accuracy. The conclusions are presented in Section 7.

2. Fingerprint based biometrics

Biometric systems are designed to perform the recognition of people. The need to verify that a person corresponds to the individual it is claiming to be (verification) or to determine which person is trying to access a certain piece of information, restricted area or device is an issue that has been tackled for over a century. The idea of identifying criminals by the fingerprints collected from crime scenes started in the 19th century although evidences exist that some cultures used fingerprints many centuries B.C. As an example a Chinese clay seal dated 300 B.C was found with a finger imprint and it is believed that in the Chinese culture they were to some degree aware of the uniqueness of fingerprints 5000 years ago [22, 24].

In the modern era fingerprints started being studied scientifically in the 17th century [4, 15] and its uniqueness was established in the 18th century [25]. The identification of criminals in the forensics field

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

using fingerprints was the main use of this biometric feature and this includes the use of fingerprints as part of criminal record databases.

Some of the reasons fingerprints are the most used biometric trait:

- They are assumed to be unique.
- They were introduced as identification method many centuries ago.
- They are inalterable unless they get scarred or affected by a burn.
- Fingerprints can be acquired using non intrusive methods .
- They could be captured and compared without the need of electronic devices. Ink impressions and manual comparison were the techniques used in the pre-electronic era.
- A vast amount of research has been done to create efficient systems for capturing and comparing fingerprints automatically.

Multimodal systems use more than just a single biometric characteristic to verify the identity of a person or to identify an individual. Some of these additional characteristics are: iris texture, face, hand/finger veins, voice, etc. The use of technology has allowed managing these traits but it has not decreased the importance of fingerprints in the biometrics field. Many countries have fingerprints records of an important part of the population acquired in the process of issuing the national ID card or passport.

Designing and implementing efficient and robust Automatic Fingerprint Identification (AFI) systems has been a topic of interest not only to process crime scene latent fingerprints (left involuntarily on surfaces being touched) but also because of the increasing national security related issues such as fast and automatic passenger identification or even access control in private enterprises. Moreover, the introduction of commodity hardware capable of fingerprint scanning is increasing the use this biometric feature as identification tool in personal computers (mainly in laptops) and mobile devices. On some mobile phones the identification using a fingerprint can already replace less secure and practical techniques such as line patterns, passwords, etc.

Any AFI requires a database of template fingerprints of the population that should be potentially identified. The second key stone of such a system is an efficient matching technique that can provide a response in a reasonable time. Currently the main issue is still the matching process as it is performed on site and in real time. The acquisition and processing of input fingerprints to be added to the database is only performed once per individual and it not as demanding in terms of processing power as matching a fingerprint against a big template database in real time.

An excellent compendium of the state of the art of fingerprint processing and matching can be found in the Handbook of Fingerprint Recognition [24].

2.1. Feature extraction and acquisition

The acquisition of fingerprints using ink impressions on paper has been replaced by techniques based on electronic devices. These devices scan fingerprints by either posing or rolling a finger on the device. When fingers are rolled more information (area) is captured which may facilitate the process of identifying latent fingerprints.

The ridges and valleys that are part of the structure of each fingerprint enable to extract several types of features:

- Singular points: these are the points ridges are arranged around (usually a number between 0 and 5).
- Orientation maps: composed of the direction of the fingerprint lines at any point.
- Intra-ridge details such as skin pores can be detected using very high-resolution images.
- Minutiae: these are by far the most used fingerprint feature as they can be extracted even from ink impressions and because they are considered the most reliable feature. Minutiae are ridge points recorded where a change is detected: bifurcations, ridge ends, relevant direction changes, etc.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

Figure 1: Fingerprint minutiae set

A minutia is mainly defined by its coordinates on the fingerprint picture, orientation or angle and type label, although additional characteristics can be recorded. A typical number of extracted minutiae would be between 50 and 200 and a fingerprint can be fully represented using only the minutiae set if minutiae based fingerprint matching techniques are used. Figure 1 depicts a fingerprint and the minutiae set extracted from it.

2.2. Matching process

Minutiae based fingerprint matching techniques compute similarity scores between the minutiae sets of two fingerprints. Calculating this score usually involves computing distances and differences between neighboring minutiae. The similarity can be evaluated at a global scope, considering the relations of all the minutiae of the fingerprints and/or at a local level where local relative relations with neighboring minutiae are considered. State of the art matching techniques usually include both types of similarity evaluation to compute a global score.

The rest of this work focuses on such a minutiae matching technique and its adaptation to GPU hardware architectures.

3. Fingerprint matching process using local and global structures

Jiang *et al.* published a work [19] in which they presented a new minutiae based fingerprint matching algorithm which allowed a large amount of fingerprints to be processed per second. This algorithm is based on a two-step strategy in which, first a local matching phase is applied to find the best matching minutiae pair using only local relations. Afterwards, a second phase performs another matching pass using global relations to produce a matching score. This scheme addresses the potential distortions, rotations and translations that any fingerprint might have suffered during the acquisition process.

3.1. Local matching step

Let n be the number of minutiae of a fingerprint $(k_0 \ldots k_{n-1})$. The local structure matching step is based on the comparison of local neighborhoods and for each minutia k of each fingerprint, the l-nearest neighboring minutiae $(lr_0 \ldots lr_{l-1})$ are considered to compute the local structure feature vector based on:

- the relative distance between k and each neighboring minutia lr_j

$$d_{klr_j} = \sqrt{\left(x_k - x_{lr_j}\right)^2 + \left(y_k - y_{lr_j}\right)^2}$$

- the radial angle between k and each lr_j

$$\theta_{klr_j} = d\phi\left(\tan^{-1}\left(\frac{y_k - y_{lr_j}}{x_k - x_{lr_j}}\right), \varphi_k\right)$$

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

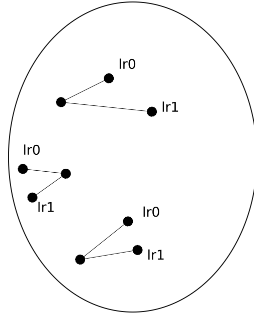


Figure 2: Local structures using the 2 nearest neighbors

- the difference between the angles of k and each lr_j

$$\varphi_{klr_j} = d\phi\left(\varphi_k, \varphi_{lr_j}\right)$$

where φ is the local ridge direction and $d\phi\left(\alpha, \beta\right)$ is a function that computes the difference between two angles.

Because of the relative nature of these measures, local structures are independent from rotation and translations produced during the capturing process. Finally, the number of ridges and the minutiae types are added to the local structures defining local structure feature vector Fl_k for each minutia.

Let the input fingerprint be represented by n_i minutiae and let each template fingerprint stored in the database be composed of n_t minutiae. The value of n_t will be different for each template fingerprint but for simplicity, n_t will be used to represent the number of minutiae of any template fingerprint. In order to compute a local similarity score (SL) between two minutiae k_a and k_b, a weighted difference of their local feature vectors Fl_{k_a} and Fl_{k_b} is computed. Taking into account the minutiae sets of both the input and the template fingerprint, an SL $n_i \times n_t$ sized similarity matrix is obtained which stores the local similarity level of any pair of minutiae of both fingerprints.

Matrix SL is defined as follows:

$$SL(k_a, k_b) = \begin{cases} \frac{bl - W|Fl_{k_a} - Fl_{k_b}|}{bl}, & \text{if } |Fl_{k_a} - Fl_{k_b}| < bl \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where bl and W are empirically defined constants provided in the original paper.

Figure 3 shows the pseudocode of the local matching process of a minutia of an input fingerprint and a minutia of a template. Although this function contains a loop over the l-nearest neighbors, for $l = 2$ (or any low value for l), the complexity of this function can be considered as $\mathrm{O}\big(1\big)$. In that pseudocode each minutiae k_a is represented by a set of two local relationships (lnb.localRelation[j], j={1,2}) where each of these relationships store the relative distance, radial angle,... between k_a and its two neighbors. The feature vector Fl_{k_b} is formed by the values that represent each neighboring relationship.

The local matching process of a pair of fingerprints requires the computation of all the possible one-to-one minutiae matching processes between their minutiae (SL matrix). This process is shown in the pseudocode of Figure 4 and has a complexity $\mathrm{O}\big(n^2\big)$ with respect to the number of minutiae per fingerprint.

Finally, performing the local matching step with an input fingerprint against all the template fingerprints of a database would have a complexity of $\mathrm{O}\big(n^2\, m\big)$ where m represents the fingerprint database size and n represents the number of minutiae per fingerprint.

3.2. Global matching step

After the local matching step, the two best matching minutiae of the previous phase are selected as reference elements for the global matching process. During this stage, the global structure for each of these two minutiae is computed using the same relative Euclidean and angular differences of the local phase with

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

```
 1 float minutiaeLocalMatching(localNBH lnbi,localNBH lnbt) {
 2 //lnbi: local neighborhood of an input minutia
 3 //lnbt: local neighborhood of a template minutia
 4
 5   float sum = 0
 6   //Piecewise computation of the weighted difference of feature vectors
 7   for j = 1 to l { //l-nearest minutiae
 8     lri = lnbi.localRelation[j] //neighboring relationship j of lnbi
 9     lrt = lnbt.localRelation[j] //neighboring relationship j of lnbt
10     sum += substractFeatureVectors(lri,lrt)
11   }
12   return (sum < bl)?  (bl-sum)/bl : 0.0
13 }
```

Figure 3: Local matching process of a pair of minutiae

```
 1 float fingerprintLocalMatching(localNBH[] lnbiArray,localNBH[] lnbtArray) {
 2 //lnbiArray: all local neighborhoods of the input fingerprint
 3 //lnbtArray: all local neighborhoods of a template fingerprint
 4
 5   maxSL = 0
 6   for t1 = 1 to lnbiArray.size() {
 7     for t2 = 1 to lnbtArray.size() {
 8       SL[t1,t2] = minutiaeLocalMatching(lnbiArray[t1],lnbtArray[t2])
 9       maxSL = max(maxSL,SL[t1,t2])
10     }
11   }
12   return maxSL
13 }
```

Figure 4: Local matching process of a pair of fingerprints

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

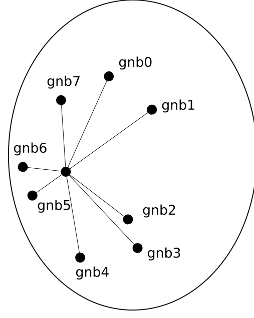


Figure 5: Global structure

respect to the rest of the minutiae of each corresponding fingerprint $(d_{kl_j}, \theta_{kl_j}, \varphi_{kl_j})$ to create global structure vectors Fg_k. The minutiae type and ridge count values are not used in this step. It is important to state that the global structure of the input fingerprint encodes the relation of the best matching minutiae of the input fingerprint with the rest of the n_i minutiae. The same applies to the template fingerprint.

During the global matching phase a similarity matrix ML is computed which stores the similarity level of any pair of minutiae based on the global structure matching and the local similarity score of the previous phase. The computation of the matrix ML requires that each global structure of the input fingerprint be compared to all the global structures of the template fingerprint.

Matrix ML is defined as follows:

$$ML(k_a, k_b) = \begin{cases} \frac{1}{2} + \frac{1}{2}\, SL(k_a, k_b), & \text{if} \, |Fg_{k_a} - Fg_{k_b}| < Bg \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where Bg is a vector with constant values defined in the original paper.

The final score M_s is computed as an average of the ML values imposing an important constraint, $ML(k_a, k_b)$ is set to 0 if there is any minutia c so that:

$$ML(k_a, k_c) > ML(k_a, k_b)$$

or

$$ML(k_c, k_b) > ML(k_a, k_b)$$

to ensure minutiae are not being used several times for different matches. Then, M_s is computed as follows:

$$M_s = 100\, \frac{\sum_{i,t} ML(i,t)}{\max\{n_i, n_t\}}$$

The complexity of the global matching process is the same as the local matching process: $\mathrm{O}(n^2)$. The pseudocode scheme is also equivalent and is shown in Figure 6. At the minutia level, the difference of global feature vectors Fg_k for every pair of minutiae is computed.

4. Computing with GPUs and Fingerprint identification: A short snapshot

In this section general purpose computation using GPUs and its application to fingerprint identification systems is presented. First, the general GPU based computing paradigm will be introduced and then how it has been used successfully in the complex task of fingerprint identification in large environments will be discussed.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

```
void fingerprintGlobalMatching(globalNBH[] gnbiArray,globalNBH[] gnbtArray) {
//gnbiArray: all global neighborhoods of the input fingerprint
//gnbtArray: all global neighborhoods of a template fingerprint

  for t1 = 1 to gnbiArray.size()  {
    for t2 = 1 to gnbtArray.size() {
      ML[t1,t2] = minutiaeGlobalMatching(gnbiArray[t1],gnbtArray[t2])
    }
  }
}
```

Figure 6: Global matching process of a pair of fingerprints

4.1. General purpose computing with GPUs

GPUs have become very common in a wide range of computational environments. These devices were initially designed to efficiently process the computer graphics pipeline by providing programmable stages. Nowadays, they are becoming a mainstream computational option in a growing number of areas.

GPUs are currently programmable using frameworks such as openCL [28] and CUDA [8] although there are other higher level options available (at the expense of efficiency). CUDA is provided by Nvidia [27] for its GPUs and includes a compiler which adds some extensions to C/C++ to allow parallel applications to be run on Nvidia GPUs. These GPUs are composed of several Streaming Multiprocessors (SMXs) which are, in turn composed of several processing units (CUDA cores).The Nvidia Tesla K20m model offers 2496 CUDA cores grouped into 13 SMXs (streaming multiprocessors) which yields 192 processors per multiprocessor. These cores are able to run a large amount of threads in parallel using an SIMD or SIMT scheme. Threads are run in groups of 32 (warp) in such a way that any thread in the warp should be running the same instruction (although generally over different data elements) at the same time. Threads are also grouped (at a higher level) into blocks. Threads in the same block can synchronize and use a high speed shared memory. This is achieved by running all the threads belonging to the same block on the same SMX.

The latest GPU hardware developments aim at increasing energy efficiency and the number of computing cores. The Fermi architecture of Nvidia provided a maximum of 512 cores per GPU whereas current GPUs with the Kepler architecture offer up to 2880 CUDA cores. The forthcoming Maxwell architecture will also focus on increasing performance per watt. Without going into the greatest level of detail, the main difference between the latest Nvidia GPU generations is the total number of CUDA cores, the number of cores per multiprocessor and the clock speed of each core. In this regard, GPU design trends are following the same pattern as in the CPU domain: increase in the parallel computation capabilities, reduction (or only low increase) of the clock rate and die shrink (reduction of the transistor size). This is the result of having reached certain power consumption levels which limit performance increases. An increase in the number of cores helps accelerating compute-bound problems but not memory-bound ones where accessing the data is the bottleneck. Moreover, if the next generation cores operate at a lower clock speed, memory-bound problems can actually show no performance increase despite of the larger number of computing elements.

Problems which exhibit a high degree of data parallelism that may be divided into many instances (threads) of the same code (kernel) running in parallel are candidates to be highly optimized by the use of GPUs. In order to obtain a good performance it is important to obey certain rules:

- There should be no code divergence inside each warp, in order to avoid code execution serialization.
- Threads should be assigned an adequate workload level in order to achieve a sufficient instruction level parallelism and keep the pipelines of the CUDA cores as full as possible.
- Threads should not require too many synchronization points between each other.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

- Memory access latencies should be avoided by providing a sufficient level of thread level parallelism. When a warp needs to perform a memory access operation it is stopped and another warp is run while memory is accessed. GPUs perform extremely low cost context switches by providing a very large number of registers that avoid the need to restore the state of a warp after being stopped and restarted. This means that enough threads have to be supplied in order to keep a large number of threads ready to be run in case a memory operation needs to be performed.
- Memory access operations should be fully coalesced to allow memory accesses for different threads to be performed with only one operation.

The problem to be solved using GPUs has therefore to be partitioned into large sets of threads. These threads have to be grouped into thread blocks depending on the synchronization and shared memory requirements. Both threads and blocks can be organized in up to three dimensions to better map the problem being solved. The whole set of thread blocks created is called a grid.

According to the details provided above, boosting the efficiency of an algorithm through the use of GPUs is not a simple matter of hardware improvement due to a higher number of cores, i.e. processing elements. On the contrary, a great effort is necessary since the algorithms often require undergoing a thorough redesign in order to obtain peak performance by using GPUs. In this work we have efficiently mapped a fingerprint matching algorithm to the GPU architecture as described in the following sections.

Finally, to provide a full picture of the GPU based computing world, gaming oriented graphics cards can also be used for general purpose high performance computing. The Nvidia GTX GPU family corresponds to gaming oriented GPUs which nevertheless can provide, in some scenarios, performance similar to Nvidia Tesla GPUs but that were not designed for high performance computing environments. GTX GPUs lack ECC (Error Correcting Code) memory and the size of this memory is usually notably smaller, have a lower double precision computing power and are clocked at higher frequencies theoretically offering less stability guarantees. GTX GPUs also include only one memory transfer engine, which means only a memory transfer operation can be performed at a time, while Tesla cards include two. Nonetheless, in applications were GPU memory size and transfers are not an issue and double precision computations are not required, these type of GPUs can provide performance similar to Tesla GPUs at a fraction of the cost.

AMD [1] is the other main high-performance GPU producing company in the market. The latest architecture designed by AMD GPUs is called GCN (Graphics Core Next [14]) and it is of course different from the one developed by Nvidia. This does not mean that GPU software developments are completely GPU vendor specific because both architectures are equivalent up to a certain level. The GCN architecture provides computing units (CUs) composed of several SIMD units and each GPU is composed of a set of CUs. This is equivalent to a GPU being composed of multiprocessors which group a set of CUDA cores. In the GCN architecture, threads are run in groups of 64 called wavefronts which are also equivalent to Nvidia's thread warps. Moreover, both Nvidia and AMD GPUs can be programed with the cross-platform framework openCL [28], which is an alternative to CUDA.

The GPU-based fingerprint matching system presented in this work is not intrinsically tied to only one GPU vendor (Nvidia) and could be re-implemented on openCL and run on AMD hardware. The adaptation would require fine-tuning the workload and work packaging parameters but the main structure could be maintained.

4.2. Recent approaches on the use of GPUs for fingerprint identification

The problem of adapting the fingerprint identification process to GPU based architectures has not been dealt with in many works. An image based proposal using GPUs was presented in [3] but the quality of the extracted fingerprint features was not addressed by that work. The comparison of the GPU and CPU based software presents some problems as the number of extracted features differs depending on the software that is chosen and also because the hardware that was used is far from the current state-of-the-art of GPU devices. According to the authors, the GPU implementation of the superior technique presented in that paper is slower than the CPU version during the matching phase due to excessive memory transfers.

In [16] a GPU adapted redesign of the well-known MCC [5] fingerprint matching technique is presented. This work provides very important speed-up factors with respect to CPU based systems and offers a high

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

accuracy level but, due to its inherent computational intensity, still does not allow real-time fingerprint matching for databases of millions of fingerprints.

5. Proposal: fast fingerprint identification using GPUs

As mentioned in the introduction, in order to design a GPU version of the fingerprint matching algorithm presented by Jiang et al. [19], the main sources of parallelism have to be studied. In this case, considering a one-to-one fingerprint matching process, the number of minutiae of both fingerprints and the quadratic nature of the matrices that have to be computed supply a natural source of parallelism.

On the other hand, considering that this work focuses on matching using big fingerprint databases, the database size also provides a very important source of parallelism.

Another important factor is the low arithmetic intensity of the operations required for both the local and global matching steps. This fact represents an additional challenge as it requires a careful workload distribution to fully use the performance offered by GPU hardware.

The proposed fingerprint matching system has been implemented in C++ using the C language extensions provided by CUDA. The debugging and profiling modules of the Nvidia Nsight Eclipse Edition IDE [26] were used in the development process.

5.1. Local matching step

The local matching step computes the SL matrix and the best matching minutiae pair from the input fingerprint and a template fingerprint from the database. Moreover, the SL matrix computation requires that all the local structures of the input and template fingerprints be processed and afterwards a reduction operation has to be performed to obtain the maximum of that matrix.

We have chosen to use the two nearest neighbors for each minutia ($l = 2$) as indicated in [19]. This means that each local structure is composed of two elements: lr_0 and lr_1 which are used to compute the local feature vectors for each minutia. These vectors will be referred to as $lnbi$ (local neighborhood structure of minutiae i).

A naïve task assignment scheme would map each thread to the *minutiaeLocalMatching* function (Figure 3). The parallelism would be obtained by replacing the nested loops of each fingerprint local matching step (Figure 4) by the parallel launch of n^2 threads. Finally, due to the parallel computation of the SL matrix the computation of the maximum value of this matrix would have to be moved to a separate process after the whole SL matrix has been computed. This scheme is simple and direct but the performance obtained in early tests shows that it does not provide enough thread level parallelism.

The improved mapping between threads and computational tasks chosen is shown in Figure 7. In our GPU oriented design each thread computes the similarity between a local structure from the input fingerprint $lnbi$ and all the local structures from the template fingerprint $lnbt0 \dots lnbtn_t$ (each local structure is composed of two elements). Figure 7 shows the matrix of all the possible comparison operations that could be performed and the marked area represents the portion of work assigned to a single thread which, as mentioned, implies pairwise processing one local structure from the input fingerprint and all the local structures of the template. This scheme provides sufficient workloads for each thread to maintain an adequate level of instruction level parallelism (loops are partially unrolled) and also allows a partial result of the reduction operation to be obtained in order to compute the global maximum of the SL matrix. Each thread performs a local computation of the maximum of the SL matrix, thereby avoiding the need for a dedicated GPU kernel for this task. The global maximum of the set of local maxima computed by each thread is computed by the CPU after the GPU completes the local matching computation.

As each thread processes a column of the SL matrix, memory access operations are guaranteed to be fully coalesced as each thread of a warp accesses input local structures which are stored in consecutive memory positions.

Figure 8 shows the function that is assigned to each thread, which has an increased instruction level parallelism and allows a part of the maximum computation to be included in the same kernel. Figure 9 shows the structure of the local matching process of a pair of fingerprints using the new scheme.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

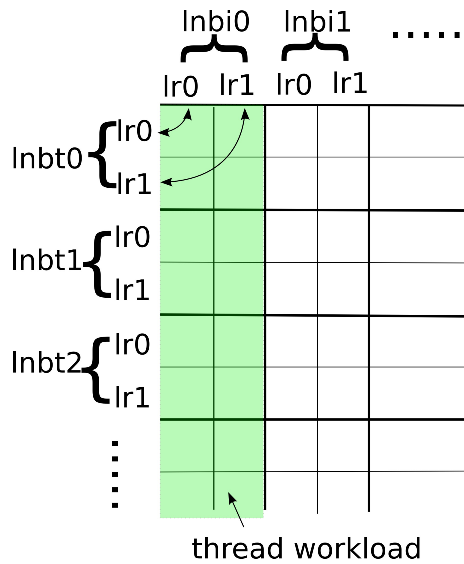


Figure 7: Local matching process

```
 1 float optimizedMinutiaeLocalMatching(localNBH lnbi,localNBH[] lnbtArray) {
 2   //Process the ith local neighborhood of the input fingerprint
 3   //and all the local neighborhoods of the template fingerprint
 4   lmax = 0.0
 5   i=lnbi.index
 6   for t = 1 to lnbtArray.size() { //This loop is marked to be unrolled
 7     float sum = 0
 8     for j = 1 to 2 { //l=2
 9       lri = lnbi.localRelation[j]
10       lrt = lnbtArray[t].localRelation[j]
11       sum += substractFeatureVectors(lri,lrt)
12     }
13     SL[i,t] = (sum < bl) ? (bl-sum)/bl : 0.0
14     lmax = max(lmax,SL[i,t])
15    }
16    return lmax
17 }
```

Figure 8: Optimized local matching process of a pair of minutiae

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

```
1 float optimizedFingerprintLocalMatching(localNBH[] lnbiArray,localNBH[] lnbtArray) {
2   maxSL = 0
3   //This loop is parallelized using lnbiArray.size() GPU threads
4   for t1 = 1 to lnbiArray.size() {
5     localMax[t1] = optimizedMinutiaeLocalMatching(lnbiArray[t1],lnbtArray)
6   }
7   for t1 = 1 to lnbiArray.size() {
8     maxSL = max(maxSL,localMax[t1])
9   }
10  return maxSL
11 }
```

Figure 9: Optimized local matching process of a pair of fingerprints

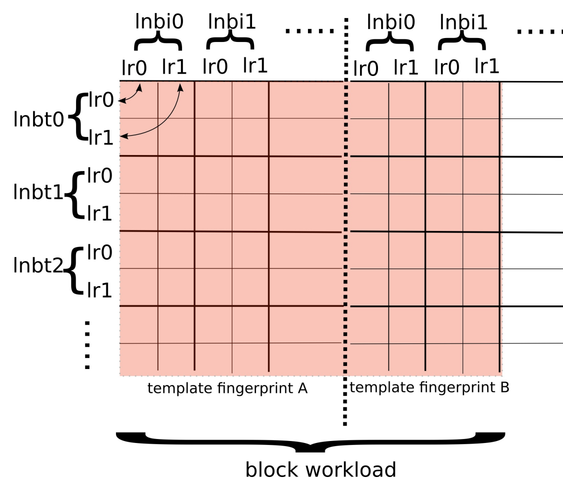


Figure 10: Template fingerprints workload packaging

Nevertheless, the aforementioned computing scheme requires n_i threads to compute matrix SL and, taking into account the usual values of n_i, the thread level parallelism would be quite low. To increase the thread level parallelism several template fingerprints are processed by each CUDA grid. This means that the matching process between the input fingerprint and the fingerprint database is packaged into chunks whereby, at each step the input fingerprint is matched against a bucket of template fingerprints. Considering that we are focused on large fingerprint databases, this mechanism provides an adequate workload level to achieve a good thread level parallelism.

In order to set the number of threads per block and to prevent blocks being too small (input fingerprints with a low number of minutiae) which could make the maximum number of blocks per SMX a limiting factor, a variable number of template fingerprints are processed per block and the block dimension has been set to 128. The number of template fingerprints processed by each block depends on the number of minutiae per fingerprint and fingerprints with a number of minutiae larger than 128 would require more than one thread-block to be processed. Figure 10 shows an example of how one template fingerprint and part of another one are assigned to one thread block. In that figure, the marked (and joined by a horizontal brace) area represents the workload assigned to a whole thread block. Each thread compares an $lnbi$ local structure with all the local structures of one template fingerprint A. As the input fingerprint has less than 128 (block size) local structures, the two last threads of the block compare $lnb0$ and $lnb1$ of the same input fingerprint with the all the local structures of the next template fingerprint B.

To summarize the task assignment: each thread processes one local structure of the input fingerprint and

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

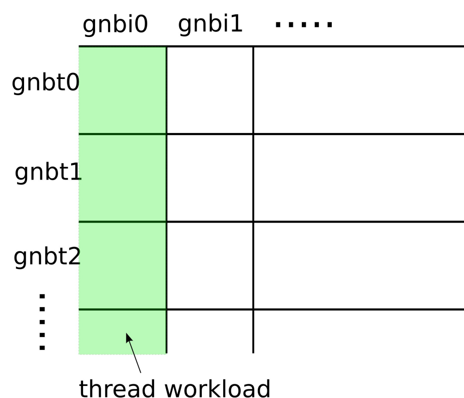


Figure 11: Global matching process

all the local structures of one template fingerprint. Each thread block includes the computation associated with 128 input fingerprint local structures.

5.2. Global matching step

The global matching step computes the ML matrix which has a different conceptual definition to SL, but the computations required have a similar structure for both matrices. Each thread is assigned a global structure from the input fingerprint *gnbi* and all the global structures from the template fingerprint $gnbt0 \ldots gnbtn_t$. Figure 11 shows how each thread is assigned the similarity computation of the global structure of one fingerprint and all the global structures of the template one. In this case, as the computation of the average of ML has to ensure that a minutia is only paired with another minutia from the other fingerprint, no local reduction process is performed during the computation of ML. This fact reduces the arithmetical intensity with respect to the number of memory access operations.

As in the previous stage, each thread block has a fixed number of threads, and depending on the number of minutiae of the fingerprint, a variable number of template fingerprints is processed by each thread block.

After matrix ML has been fully computed by iteratively using a parallel reduction kernel (which sets the row and column corresponding to a minutiae pair to 0 once it has been used in the average computation) the final score is obtained. At each iteration the local maximum associated with each input global structure (one matrix column) of matrix ML is computed by each thread. The second step uses the high speed thread block shared memory to iteratively perform a parallel reduction operation to compute the global maximum using the previous local results. After an iteration finishes, the values of ML which are in the same row and column as the maximum value that has just been computed are set to 0 (this operation is also performed in parallel). The maximum values obtained at each step are the values required to compute the final score M_s.

5.3. Fingerprint matching against a database

As has been mentioned above, the use case we are considering is a scenario in which each input fingerprint has to be matched against a fingerprint database. This process is not divided into several one-to-one matching processes because that scheme would not produce sufficient thread level parallelism. As has been stated in the previous sections, the whole process is packaged into chunks where the input fingerprint and a set of template fingerprints are processed each time. Moreover, depending on the number of minutiae of the input fingerprint, each thread block may include the processing of several template fingerprints.

When an adequate value for the chunk size is used, this scheme provides the required thread level parallelism. Nevertheless, during the whole matching process of a database chunk, there are certain moments when the CPU computes its part of the process, for example when it is finishing the reduction of the SL matrix, that the GPU is idle. These periods are relatively short but we have tried to avoid them. In order to keep the GPU from idling, several workload packages are issued to the GPU in parallel. This allows the

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

GPU to process the workload corresponding to one fingerprint database chunk while the CPU does its part of the computation of a different chunk.

Each chunk or work package has to be transferred to the GPU to be processed and memory transfers using the PCI Express bus could become a bottleneck if they are not handled properly. In our system asynchronous memory transfers are scheduled in parallel to the matching processes. At the same time a chunk i of the database is processed on the GPU, chunk $i+1$ is transferred from the main memory to the GPU. Due to the lack of dependencies between work packages and the capability of GPUs to overlap computing phases and memory transfers, this scheme allows the cost of memory transfers to be completely hidden.

The chunk size needs to be large enough to make the latency related to memory transfers and kernel launch operations negligible. Once a certain chunk size is reached, further increasing it does not provide any additional benefits in terms of run time and only increases the memory footprint because each kernel needs to keep twice the chunk size allocated during the whole process.

5.4. MultiGPU matching process

As multi-GPU computing devices are becoming common due to the availability of several high-speed PCI Express slots on many PC motherboards, we have extended our GPU fingerprint matching to support an arbitrary number of GPUs. Each GPU requests its workload package and processes it as has been described in previous sections. When the work has been finished, another workload package is requested from the pending work queue.

This scheme allows multiple GPUs to collaborate efficiently in the fingerprint matching process of each input fingerprint. As the workload is assigned dynamically, the performance differences between the GPUs has a lower impact on the final performance than if it was assigned statically at the beginning of the process. The workload assignment strategy follows a classic manager-workers scheme. The host computer acts as manager and supplies work to the GPU workers upon request. As the processing task is divided into chunks (see Section 5.3) faster GPUs will naturally request and process more chunks than slower ones.

In practice each GPU is used as an independent high performance co-processor (each with its own memory). Memory transfers are performed from/to the host computer memory to/from each GPU using DMA operations over the shared PCI-E bus. As no communication is required between GPUs their processing tasks are kept independent from each other which reduces the system complexity.

6. Experiments and analysis of results

To assess the effectiveness and efficiency of our proposal we have designed and carried out a thorough empirical analysis. It is described in this section, in which we detail the hardware used (Section 6.1), fingerprint databases (Section 6.2) and experiments (Section 6.3). After that, the results obtained are presented (Section 6.4). The section concludes with the analysis of the obtained results (Section 6.5) and the proposal of a GPU based multi-technique identification pipeline (Section 6.6).

6.1. Hardware

Our proposal has been tested on a variety of hardware platforms including different CPUs and GPUs. These platforms are composed of a cluster node and a desktop computer.

The cluster node is equipped with two Intel Xeon E5-2630 processors at 2.30GHz and 128 GB of RAM memory. Each of these processors has 6 cores that allow the running of up to 12 threads per processor using hyperthreading (24 threads using both processors). This node is equipped with four Nvidia Tesla GPUs where Tesla denotes the brand of Nvidia GPUs for high performance computing servers. These GPUs are characterized by a higher amount of memory and stability (at the expense of a lower clock rate) compared to their desktop GTX counterparts. Two different models of GPUs were used:

- Two Nvidia Tesla K20m: Kepler architecture with 2496 CUDA cores at 0.7 GHz and 5 GB of memory.
- Two Nvidia Tesla M2090: Fermi architecture with 512 CUDA cores at 1.3 GHz and 6 GB of memory.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

Scanner parameters
Acquisition area: 0.58” x 0.77” (14.6mm x 19.6mm).
Resolution: 500 dpi, Image size: 288 x 384.
Background type: Optical, Background noise: Default.
Crop borders: 0 x 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.

Table 1: Parameter specification used with SFinGe tool

These four GPUs are connected to the same host computer by using four PCI Express slots.

The Fermi architecture provides higher clock frequencies but a lower number of cores. On the other hand, the Kepler architecture, which is the latest with high-end products released by Nvidia, provides a higher number of cores but at a lower clock rate than Fermi devices.

The desktop computer is equipped with an Intel Core i7-3820 at 3.6GHz processor and 24 GB of RAM. This processor has 4 cores that allow the running of up to 8 threads using hyperthreading. An Nvidia GeForce GTX 680 GPU is attached to this computer. This device belongs to the Kepler architecture family and has 1536 CUDA cores at 1.06 GHz and 2 GB of memory. This GPU is a computer graphics oriented device.

6.2. Datasets

Two different fingerprint databases have been used to test the proposed algorithm. These databases have different sizes, sources, number of minutiae and fingerprint types.

- The DB14 database, provided by the NIST (National Institute of Standards and Technology) [33], includes two samples of 27 000 rolled fingerprints, making a total of 54 000 fingerprints. The fingerprints in this database generate a large number of minutiae with an average of 206.9 and a maximum of 610. Minutiae were extracted using the mindtct [34] software.

- A synthetic fingerprint database generated using the SFinGe software [6]. Table 1 shows the parameters used for SFinGe. Although this database contains 25 impressions of each fingerprint, only two samples of 400 000 plain fingerprints were used in this work, making a total of 800 000 fingerprints. The selection of impressions was configured to discard samples with less than 40 minutiae. The minutiae of each fingerprint were extracted using the mindtct [34] software generating an average number of 51.8 minutiae per fingerprint and a maximum of 141.

6.3. Experimental framework

To measure the performance of our proposal, different experiments have been performed using the databases and hardware introduced in the precedent sections. The experiments consisted of performing a set of identifications of a fingerprint against the database and measuring the average time employed. One hundred fingerprints were randomly selected from each database as input fingerprints.

The speed-up factor measures the improvement obtained by a parallel implementation of a system against a sequential reference implementation. We have used this factor to measure the improvement achieved by our GPU based system compared to a parallel reference CPU implementation. The CPU implementation of the algorithm uses 24 threads when running on the server node and 8 threads when running on the desktop computer. The experiments performed also show the performance using several GPUs processing fingerprint matching tasks in parallel on different parts of the database.

To emulate a deployed system, the local structures of each template fingerprint have been precalculated and preloaded before the identification process starts. This consumes a high amount of RAM memory

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

	CPU time (s)	GPU time (s)	Speed-up
K20 x1	26.576	3.548	7.49
M2090 x1	26.576	3.764	7.06
K20 x2	26.576	1.780	14.93
M2090 x2	26.576	1.888	14.07
4 GPU	26.576	0.943	28.20

Table 2: DB14 results with respect to the server node (54 000 template fingerprints)

	CPU time (s)	GPU time (s)	Speed-up
K20 x1	10.802	0.688	15.69
M2090 x1	10.802	0.731	14.79
GTX 680	10.802	1.030	10.48
K20 x2	10.802	0.354	30.49
M2090 x2	10.802	0.376	28.71
4 GPU	10.802	0.199	54.20

Table 3: DB14 results with respect to the desktop computer (10 000 template fingerprints)

making it impossible to perform the same tests on the desktop computer and on the cluster node. Two experiments have been carried out with each database, one on each computer, to compare the run times obtained by the different CPUs. Each experiment uses a different amount of template fingerprints, limited by the amount of memory available on each computer.

6.4. Empirical results

This section presents the results obtained from the set of experiments that have been carried out. It also describes the parameter values that have been used and the number of template fingerprints used in each experiment.

The run times included in the tables correspond to the average time of a single identification process computed by performing 100 identifications using 100 different and randomly selected fingerprints from each database as the input. This section also describes the parameter values that have been used and the number of template fingerprints used in each experiment.

- DB14 database: The experiment performed on the server node uses all the fingerprints of the database as templates, a total of 54 000 fingerprints. By contrast, the experiment performed on the desktop computer uses only the first 10 000 fingerprints of the database as templates. The GPU algorithm packs 350 template fingerprints per kernel launch for Tesla devices and 400 when the GTX 680 device was used. Three kernels are run in parallel on every device.

  Tables 2 and 3 present the results of each experiment with this database. The speed-up factors range from 7 using one GPU to 28 using four GPUs with respect to the server node and from 15 to 54 with respect to the desktop computer. It is important to highlight that the reference CPU time was obtained on a dual processor with 12 physical cores (usable as 24 cores using hyperthreading) in the case of the server node and a Core-i7 CPU (8 cores using hyperthreading) using a multithreaded code in both cases.

- SFinGe database: The experiment performed on the server node uses the whole database, a total of 800 000 fingerprints. On the desktop computer, only the first 150 000 fingerprints were used. Independently of device type, 1250 template fingerprints were processed per kernel run. However, on Tesla devices five parallel kernels were run while on the GTX 680 the best results were obtained with four parallel kernels.

  Tables 4 and 5 show the results of the experiments using the SFinGe database. The speed-up factors with respect to the multithreaded CPU systems range from 5 and 47 depending on the type of CPU and the number of GPUs used.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

	CPU time (s)	**GPU time** (s)	**Speed-up**
K20 x1	11.580	2.188	5.29
M2090 x1	11.580	1.938	5.98
K20 x2	11.580	1.101	10.51
M2090 x2	11.580	0.972	11.91
4 GPU	11.580	0.533	21.71

Table 4: SFinGe results with respect to the server node (800 000 template fingerprints)

	CPU time (s)	**GPU time** (s)	**Speed-up**
K20 x1	5.505	0.448	12.28
M2090 x1	5.505	0.393	14.02
GTX 680	5.505	0.617	8.92
K20 x2	5.505	0.230	23.97
M2090 x2	5.505	0.202	27.30
4 GPU	5.505	0.115	47.89

Table 5: SFinGe results with respect to the desktop computer (150 000 template fingerprints)

	CPU time (s)	**GPU time** (s)	**Speed-up**
K20 x1	57.320	0.688	83.27
M2090 x1	57.320	0.731	78.46
GTX 680	57.320	1.030	55.63
K20 x2	57.320	0.354	161.81
M2090 x2	57.320	0.376	152.34
4 GPU	57.320	0.199	287.58

Table 6: DB14 results with respect to a single-threaded implementation on the desktop computer (10 000 template fingerprints)

	CPU time (s)	**GPU time** (s)	**Speed-up**
K20 x1	26.395	0.448	58.89
M2090 x1	26.395	0.393	67.20
GTX 680	26.395	0.617	42.75
K20 x2	26.395	0.230	114.93
M2090 x2	26.395	0.202	130.91
4 GPU	26.395	0.115	229.59

Table 7: SFinGe results with respect to a single-threaded implementation on the desktop computer (150 000 template fingerprints)

Since this section focuses on showing the performance and scalability of the proposed system, in order to use as many fingerprints as possible, both samples of each fingerprint were stored in the databases and used for the experiments. Using both samples as templates for each identification allowed duplicating the size of the databases without having to generate more fingerprints.

In order to provide another perspective on the additional performance improvements GPUs provide, Tables 6 and 7 show the speed-up with respect to a single-threaded implementation of the algorithm on the desktop computer.

Finally, we provide some additional data about the experimentation process. The total number of different threads per identification process run using the DB14 and SFinGe databases on each Tesla device was $2E+7$ and $8E+7$ respectively. On the GTX device, as a reduced version of each database was used, the total number of threads run on this device was $4E+6$ and $1.5E+7$, respectively. The average percentage of time at least one warp was active on a multiprocessor [9] ranged from 84% to 90.5% during the computation steps and from 74% to 86% in the reduction steps. The ratio of the average active threads per warp with respect to the maximum number of threads per warp [9] ranged from 86.1% to 96.5% for the computation steps and from 80.6% to 90.7% for the reduction steps.

The GPU memory requirements (on each GPU) depend on the size of the work packages or chunks as it is necessary to keep stored the set of template fingerprints used in the current matching task and the

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

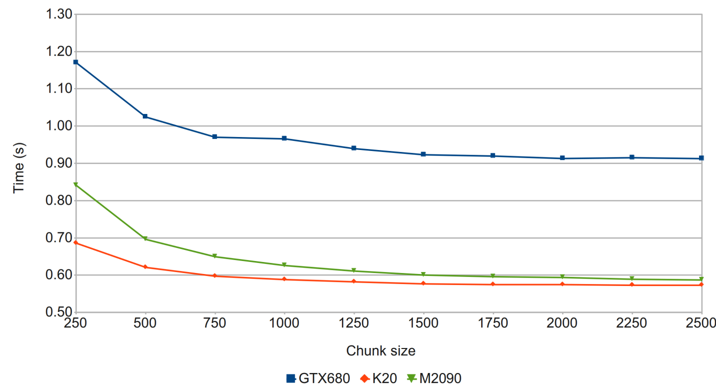


Figure 12: Influence of the chunk size for each GPU type (SFinGe database)

next set that will be processed. The latter is transfered asynchronously while the current chunk is processed (see Section 5.3). The other two factors that influence the memory requirements are the average number of minutiae per fingerprint and the number of parallel matching kernels run on each GPU. The average GPU memory footprint of the experiments ranged from 650 MB to 1.8 GB depending on the values of the aforementioned experimental parameters.

The effect of different chunk size values is shown in Figure 12. Once a certain value is reached the run time remains almost fixed but the memory requirements increase as the space for the chunk being processed and the next chunk remains allocated during the whole matching process.

6.5. Analysis of the results

The results presented in the previous section show that very important speed-up ratios of up to two orders of magnitude are obtained by applying GPUs to the fingerprint matching process (see Tables 2, 3, 4 and 5). It is important to highlight that the CPU implementations are multi-threaded ones that use all the cores and hyperthreading capabilities of each CPU. GPU speed-up values are often presented compared only to single-threaded designs but this is not the case here. It is also important to state that the server node provides two physical CPUs.

The experiments also show how the performance scales almost linearly as more GPUs are added. This fact allows for an efficient use of the full computational power of the four GPUs installed on the same host computer. It also illustrates an effective way of building cost efficient and high density fingerprint matching nodes.

These results are only possible by assigning an adequate workload amount per thread, an efficient mapping of the problem to the computational model provided by CUDA, avoiding GPU idle times and employing an efficient asynchronous memory transfer design. The importance of the memory transfer scheme is highlighted by the fact that the results obtained using a consumer grade graphics card (GTX 680) showed a lower performance despite the higher clock frequency due to the reduced parallel memory transfer capabilities compared to the Tesla counterparts.

As an additional performance measure we provide the number of fingerprint matching tasks than can be run per second. This information is extracted from the result tables of the previous section. Tables 8 and 9 show how, depending on the fingerprint database and device, the results range from less than one thousand when using one GPU to more than a million.

Finally, in order to offer a full picture of the performance level obtained, Figures 13 and 14 show the number of matching operations that can be performed per second with each device and database.

Comparing the results with those presented in [29], by using only two GPUs, the work presented in this paper is roughly on a par with a cluster of 12 dual processor nodes with 12 cores per node. In [29], using a 400 000 sized fingerprint database, the system requires 0.4922 seconds for a matching operation against the

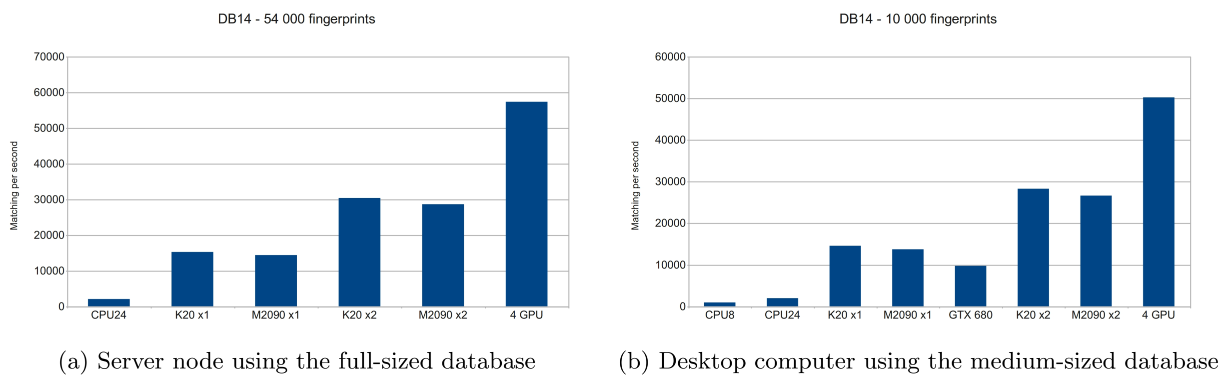
The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

	DB14	SFinGe
Xeon CPU	2032	69 086
K20 x1	15219	365 702
M2090 x1	14 345	412 834
K20 x2	30 342	726 382
M2090 x2	28 598	822 639
4 GPU	57 294	1 499 996

Table 8: Fingerprints per second using the full-sized databases

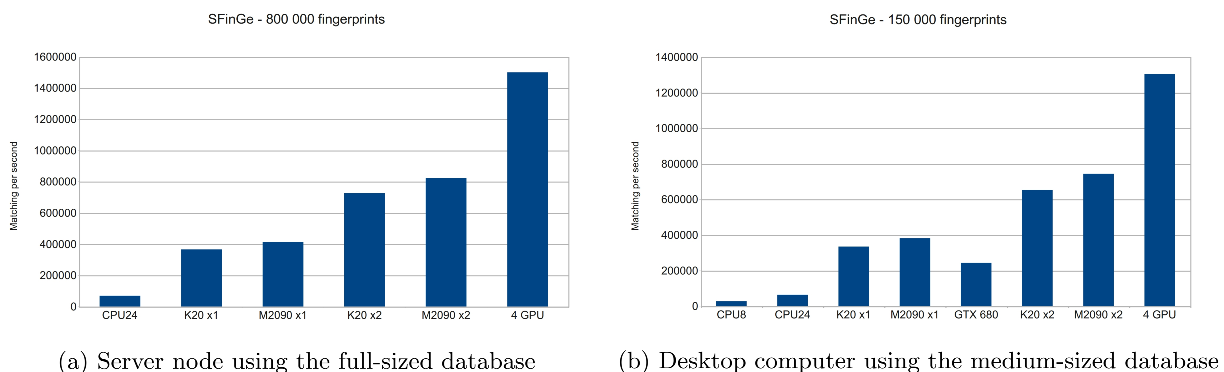
	DB14	SFinGe
Core i7 CPU	926	27 247
K20 x1	14 528	334 692
M2090 x1	13 687	381 892
GTX 680	9706	242 969
K20 x2	28 229	653 118
M2090 x2	26 578	743 932
4 GPU	50 171	1 304 752

Table 9: Fingerprints per second using the medium-sized databases for the desktop computer



(a) Server node using the full-sized database (b) Desktop computer using the medium-sized database

Figure 13: Matching operations per second using the DB14 database



(a) Server node using the full-sized database (b) Desktop computer using the medium-sized database

Figure 14: Matching operations per second using the SFinGe database

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

database (812 678 matching operations per second) and in our GPU based system, using only two GPUs and a fingerprint database with 800 000 fingerprints the performance reaches up to 822 669 matching operations per second (see tables 4 and 8). These results show that the hardware cost per matching operation and power consumption per matching operation ratios are clearly favorable to the GPU based system. It is important to highlight that the fingerprint databases mentioned in this comparison were the same. The experiment using the cluster used the first 400 000 fingerprints and the GPU system the first 800 000 fingerprints of the same fingerprint database.

6.6. Managing the performance vs. accuracy trade-offs with two level matching systems

In order to provide the reader with an insight of the accuracy of the presented fingerprint matching technique, the information obtained as a result of its submission to the FVC onGoing project [10, 13] are presented. This project provides an automated evaluation system for fingerprint recognition algorithms which resulted as an evolution of several fingerprint verification competitions. The FVC onGoing web platform serves as a fair testing environment which allows to compare different algorithms under the same conditions (it uses sequestered datasets). Its main drawback is that it only allows the submission of CPU based algorithms which excludes, for example, GPU based solutions. This means the efficiency indicators obtained can not be taken into account for high performance computing platforms which require specific pieces of hardware. This section will therefore only consider the accuracy related results of the aforementioned competition. Section 6.4 provides detailed information about the performance results on different kind of GPU devices and datasets.

The GPU-based fingerprint matching technique presented in this paper offers a high performance in terms of millions of fingerprints processed per second but its error rates are too high (2% Equal Error Rate in the standard test of the FVC onGoing project) in applicability domains where high accuracy is required. Other models exist that offer lower performance levels but provide better error rates such as the MCC technique (0.4% ERR in the same test) for which a GPU based version exists [16]. A detailed comparison based on the FVC onGoing tests is shown in Table 10 and Table 11. Those tables show the false non-matching rates (FMNR) obtained allowing a certain false matching rate (FMR):

- FMR100: the lowest FNMR for FMR$\leq$ 1%
- FMR1000: the lowest FNMR for FMR$\leq$ 0.1%
- FMR10000: the lowest FNMR for FMR$\leq$ 0.01%
- ZeroFMR: the lowest FNMR for FMR= 0%
- ZeroFNMR: the lowest FMR for FNMR= 0%

Algorithm	ERR	FMR100	FMR1000	FMR10000	ZeroFMR	ZeroFMNR
MCC (Baseline)	0,411%	0,285%	0,602%	0,999%	1,840%	95,151%
Jiang	2,039%	2,608%	4,856%	7,648%	9,614%	100,000%

Table 10: FVC onGoing results (Standard test)

Algorithm	ERR	FMR100	FMR1000	FMR10000	ZeroFMR	ZeroFMNR
MCC (Baseline)	1,765%	2,050%	3,618%	5,704%	6,071%	99,986%
Jiang	6,214%	11,175%	18,794%	24,441%	26,542%	100,000%

Table 11: FVC onGoing results (Hard test)

In identification scenarios there are equivalent rates as the ones used in verification tests.

- FNIR (False Negative Identification Rate). Under certain circumstances this rate can be considered equal to the FNMR [24].

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

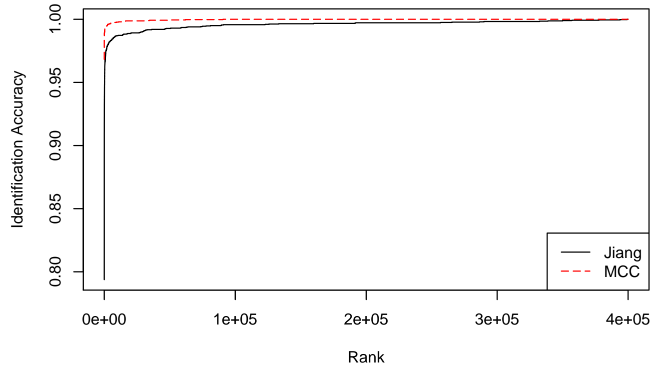


Figure 15: Cumulative match characteristic (CMC) curve computed identifying 1% of the fingerprints of the SFinGe database

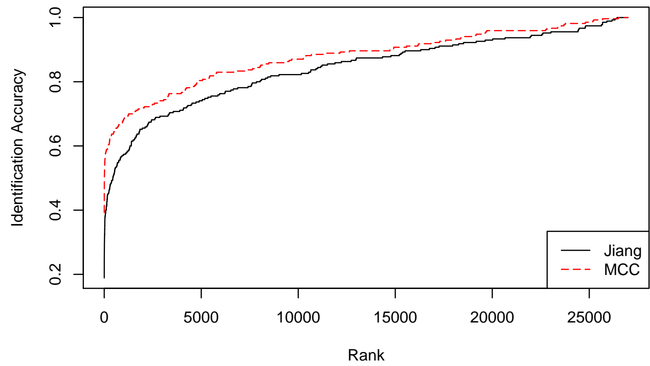


Figure 16: Cumulative match characteristic (CMC) curve computed identifying 1% of the fingerprints of the DB14 database

- FPIR (False Positive Identification Rate). Under certain circumstances this rate can be considered equal to:

$$1 - (1 - FMR)^N$$

  where N is the number of template fingerprints. If the FMR is very small $FPIR \approx N \cdot FMR$ [24].

Nonetheless, in closed-set identification tests (where fingerprints of individuals not enrolled in the system are not used) the accuracy of identification systems can also be represented using the Cumulative Match Characteristic (CMC) curve. This curve shows the probability of the correct identity associated to the input fingerprint being in the candidate set of t elements produced by the identification system (t values are represented on the x-axis). Figure 15 and 16 represent the CMC curves of the Jiang and the MCC fingerprint matching techniques using the DB14 and SFinGe databases. For this test only one sample of each fingerprint was used as template and 1% of the second samples of each fingerprint where used as input.

Regarding the Jiang matching technique, Figure 15 shows that by setting the length of the candidate list output by the identification process to less than 1% of the database size a 98% accuracy rate is reached. Using a candidate list length of less than 8% of the database size a 99% accuracy rate is obtained. By contrast, the MCC algorithm reaches a 98% accuracy rate using only the best 8 candidates and a 99% accuracy level is achieved using a candidate list of length 0.07% of the database size.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

On the other hand, Figure 16 shows that on a database with fingerprints of lower quality a candidate list with a size of the same order as the database size is required to achieve a high identification rate. Figure 16 also shows that MCC again provides the higher accuracy rates (at the expense of a higher computational cost).

At the performance level, comparing the results of [16] with those presented in this work, the GPU-based Jiang matching module processes over 7 times more fingerprints per second than the fastest GPU-based MCC version (using the SFinGe database, a Tesla M2009 GPU and MCC-8-LSS). This performance difference grows when MCC-16 is used (more cylinder cells per minutia) or a more computationally demanding global similarity technique is used for MCC (LSSR).

These facts could lead to the design of a hybrid matching system to produce a complete high speed fingerprint matching system for extremely large fingerprint databases. The faster and less accurate model would be used as a first stage filter to provide a set of candidate template fingerprints that could match the input fingerprint. In a second step, the more accurate but computationally demanding technique would be applied to the candidate set to obtain the final matching score. This scheme reduces the database penetration rate required for the second stage.

This coupling does of course require a careful binding system design to avoid the introduction of any bottlenecks produced by additional memory transfers. It would also require carefully tuning the parameters of each algorithm to find the right balance between the false match rate and false non-match rate at each stage. The false non-match rate should be minimized at the first stage at the cost of a higher false match rate taking into account the fact that only a preselection of the matching fingerprints will be created.

The proposed system would also require a highly optimized minutiae extraction system on a par with the matching system. GPU based hardware is also a candidate architecture type to be used as a base for the fingerprint capturing module but its connection to the rest of the system and the use of fingerprints of different fingers simultaneously are points that need to be addressed.

Figure 17 shows a block diagram of the proposed hybrid matching system. The fact that several matching modules are run in parallel reflects both the fact that several matching processes are run per GPU and that the whole process can be distributed over several independent GPUs or even several independent GPU nodes.

In summary, creating an efficient fully GPU based fingerprint processing pipeline presents some open questions that are not within the scope of this work and which deserve to be analyzed in a future study.

7. Conclusions

The main objective of this work was to obtain an extremely fast fingerprint matching algorithm that could be used on a two stage fingerprint matching system. The GPU based algorithm presented in this work is a perfect candidate for the first stage building block of such a system, taking into account that up to 1 500 000 fingerprint matching operations can be performed per second when four GPUs are used.

An in-depth and careful study of the sources of parallelism has been performed. This has enabled the design of an effective algorithm with an adequate task granularization and mapping between computational elements and algorithmic stages, efficient memory transfer operations and a task overlapping scheme.

The performance that was obtained matches that offered by a CPU based cluster composed of 12 dual processor nodes. This performance is obtained without losing any accuracy with respect to the reference CPU system and providing excellent scalability ratios as the number of GPU devices increases.

As future work we present an analysis for the design of a hybrid fingerprint matching system based on different techniques. The model presented in this paper would be used as a first processing step (by adjusting the matching threshold) to reduce the penetration rate in the fingerprint database required by another more accurate model such as MCC.

8. Acknowledgements

This work was supported by the research projects CAB(CDTI), TIN2011-28488 and TIN2013-4720-P. P.D. Gutiérrez holds an FPI scholarship from the Spanish Ministry of Economy and Competitiveness (BES-

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

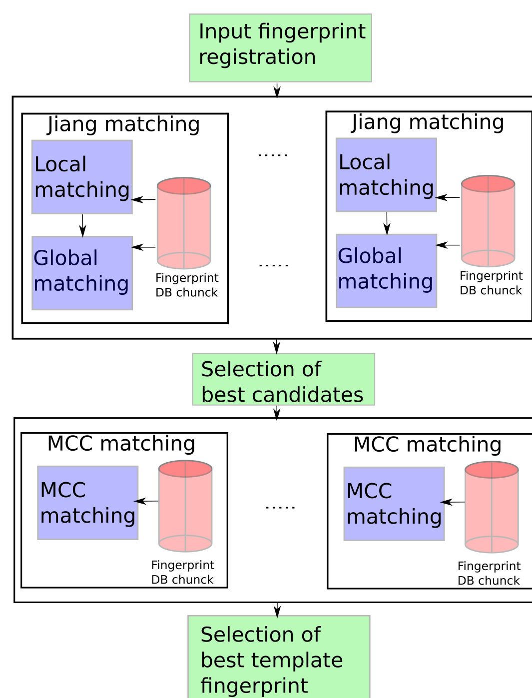


Figure 17: 2 level fingerprint matching system

2012-060450).

References

[1] AMD. `http://http://www.amd.com/`.

[2] R. Arjona and I. Baturone. A hardware solution for real-time intelligent fingerprint acquisition. *Journal of Real-Time Image Processing*, pages 1–15, 2012.

[3] A.I. Awad. Fingerprint local invariant feature extraction on GPU with CUDA. *Informatica (Slovenia)*, 37(3):279–284, 2013.

[4] G. Bidloo. *Anatomia Humani Corporis*. Tot Amsterdam, 1685.

[5] 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*, 32(12):2128–2141, 2010.

[6] R. Cappelli, D. Maio, and D. Maltoni. Synthetic fingerprint-database generation. In *Proc. 16th Int'l Conf. Pattern Recognition*, volume 3, pages 744 – 747, 2002.

[7] F. Chen, X. Huang, and J. Zhou. Hierarchical minutiae matching for fingerprint and palmprint identification. *IEEE Transactions on Image Processing*, 22(12):4964–4971, 2013.

[8] CUDA. `http://www.nvidia.com/object/cuda_home_new.html`.

[9] Nvidia Cuda Profiler User Guide. `http://docs.nvidia.com/cuda/profiler-users-guide/`.

[10] Bernadette Dorizzi, Raffaele Cappelli, Matteo Ferrara, Dario Maio, Davide Maltoni, Nesma Houmani, Sonia Garcia-Salicetti, and Aurlien Mayoue. Fingerprint and on-line signature verification competitions at ICB 2009. In Massimo Tistarelli and MarkS. Nixon, editors, *Advances in Biometrics*, volume 5558 of *Lecture Notes in Computer Science*, pages 725–732. Springer Berlin Heidelberg, 2009.

[11] M. Fons, F. Fons, and E. Cantó. Biometrics-based consumer applications driven by reconfigurable hardware architectures. *Future Generation Computer Systems*, 28(1):268–286, 2012.

[12] M. Fons, F. Fons, E. Cantó, and M. López. FPGA-based personal authentication using fingerprints. *Journal of Signal Processing Systems*, 66(2):153–189, 2012.

[13] FVC-onGoing: on-line evaluation of fingerprint recognition algorithms. `https://biolab.csr.unibo.it/fvcongoing`.

[14] GCN (Graphics Core Next). `http://developer.amd.com/wordpress/media/2013/06/2620_final.pdf`.

[15] N. Grew. The description and use of the pores in the skin of the hands and feet. *Philosophical Transactions of the Royal Society of London*, 14:566567, 1684.

[16] 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*, In Press, 2014.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

[17] H. Hasan and S. Abdul-Kareem. Fingerprint image enhancement and recognition algorithms: A survey. *Neural Computing and Applications*, 23(6):1605–1610, 2013.
[18] R.M. Jiang and D. Crookes. FPGA-based minutia matching for biometric fingerprint image database retrieval. *Journal of Real-Time Image Processing*, 3(3):177–182, 2008.
[19] X. Jiang and W.Y. Yau. Fingerprint minutiae matching based on the local and global structures. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 2, pages 1038–1041. IEEE, 2000.
[20] A. Kumar and C. Ravikanth. Personal authentication using finger knuckle surface. *Information Forensics and Security, IEEE Transactions on*, 4(1):98 –110, march 2009.
[21] M. Lastra, J.M. Mantas, C. Ureña, M.J. Castro, and J.A. García-Rodríguez. Simulation of shallow-water systems using graphics processing units. *Math. Comput. Simul.*, 80(3):598–618, November 2009.
[22] H.C. Lee, R. Ramotowski, and R.E. Gaensslen. *Advances in Fingerprint Technology, Second Edition*. Forensic and Police Science Series. Taylor & Francis, 2001.
[23] M. Liu, S. Liu, and Q. Zhao. Fingerprint orientation field reconstruction by weighted discrete cosine transform. *Information Sciences*, (in Press), 2013.
[24] D. Maltoni, D. Maio, A.K. Jain, and S. Prabhakar. *Handbook of fingerprint recognition*. Springer-Verlag New York Inc, 2009.
[25] J.C.A. Mayer. *Anatomische Kupfertafeln nebst dazugehrigen Erklrungen [Anatomical Illustrations (etchings) with Accompanying Explanations]*. Georg Jacob Decker, 1783.
[26] Nvidia Nsight Eclipse Edition. `https://developer.nvidia.com/nsight-eclipse-edition`.
[27] Nvidia. `http://www.nvidia.com/`.
[28] openCL. `http://www.khronos.org/opencl/`.
[29] D. Peralta, I. Triguero, R. Sanchez-Reillo, F. Herrera, and J.M. Benitez. Fast fingerprint identification for large databases. *Pattern Recognition*, 47(2):588 – 602, 2014.
[30] N. K. Ratha. A real-time matching system for large fingerprint databases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):799–813, 1996.
[31] M.C. Schatz, C. Trapnell, A.L. Delcher, and A. Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 2007.
[32] F. Turroni, D. Maltoni, R. Cappelli, and D. Maio. Improving fingerprint orientation extraction. *IEEE Transactions on Information Forensics and Security*, 6(3 PART 2):1002–1013, 2011.
[33] C.I. Watson. NIST special database 14. Technical report, National Institute of Standards and Technology, 1993.
[34] Craig I. Watson, Michael D. Garris, Elham Tabassi, Charles L. Wilson, R. Michael Mccabe, Stanley Janet, and Kenneth Ko. User's guide to NIST biometric image software (NBIS).
[35] J. Yang and Y. Shi. Towards finger-vein image restoration and enhancement for finger-vein recognition. *Information Sciences*, (in Press), 2013.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025515000043
This draft is provided for copyright compliance

3 GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs

- P.D. Gutiérrez, M. Lastra, J. Bacardit, J.M. Benítez, F. Herrera. GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs. Information Sciences 373 (2016) 165–182. doi: 10.1016/j.ins.2016.08.089
  - Status: **Published**.
  - Impact Factor (JCR 2015): 3.364
  - Subject Category: Computer Science, Information Systems. Ranking 8 / 144 (**Q1**).

The published paper can be found here:
`http://www.sciencedirect.com/science/article/pii/S0020025516306739`
A draft is provided for copyright compliance.

GPU-SME-kNN: Scalable and Memory Efficient kNN and Lazy Learning using GPUs

Pablo D. Gutiérrez[a,*], Miguel Lastra[b], Jaume Bacardit[c], José M. Benítez[a], Francisco Herrera[a]

[a] *Depto de Ciencias de la Computación e Inteligencia Artificial. E.T.S. Ingeniería Informática y Telecomunicación. CITIC-UGR. Universidad de Granada. Granada. Spain*
[b] *Depto. Lenguajes y Sistemas Informáticos. E.T.S. Ingeniería Informática y Telecomunicación. CITIC-UGR. Universidad de Granada. Granada. Spain*
[c] *Interdisciplinary Computing and Complex BioSystems (ICOS) Research Group, School of Computing Science , Newcastle University, Newcastle upon Tyne, Tyne and Wear, United Kingdom*

Abstract

The k nearest neighbor (kNN) rule is one of the most used techniques in data mining and pattern recognition due to its simplicity and low identification error. However, the computational effort it requires is directly related to the dataset sizes, hence delivering a poor performance on large datasets. The use of graphics processing units (GPU) has improved the run-time performance of the kNN rule but the computational requirements of current approaches limit this performance as the dataset size increases.

In this paper, we propose a new scalable and memory efficient design for a GPU-based kNN rule, called GPU-SME-kNN, that breaks the dependency between dataset size and memory footprint while delivering high performance. An experimental study of GPU-SME-kNN is presented showing a high performance, even in cases that other methods cannot address, while the computational requirements are suitable for most commercial GPU devices. Our design has also been applied to kNN-based lazy learning algorithms reducing run-times in a significant way.

Keywords: kNN, GPU, CUDA.

*Corresponding author
Email address: `pdgp@decsai.ugr.es` (Pablo D. Gutiérrez)

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

1. Introduction

The k nearest neighbor rule (kNN) [11] [30] is one of the most used data mining and pattern recognition techniques. The simplicity and low identification error of this rule makes it the reference tool to test classifiers and datasets [13]. It has been considered one of the top 10 algorithms of data mining [35]. The kNN rule is also the base of several classifiers that belong to the lazy learning family of classifiers [3].

The kNN rule is based on the idea that an unknown instance will be similar to other instances that are close to it in the space of characteristics. Although this idea is simple its computation requires a large amount of operations that increases with the dataset sizes, in terms of both attributes and instances. When addressing large problems, the time required to compute the results makes the kNN rule virtually unusable. The lazy learning algorithms that are based on the kNN rule suffer the same issues.

Currently, several real-world applications introduce scalability challenges which must be overcome by data mining techniques [28]. Given that many real world applications routinely produce massive amounts of data it is absolutely necessary to tackle the scalability challenges of the kNN rule, if this technique is going to be applied to such datasets.

Graphics processing units (GPU) have proven to be useful in managing large amounts of data efficiently in different situations like fingerprint identification [23], continuous optimization [24] bioinformatics [29] and data mining [10] [9] [8]. Moreover, the kNN rule has been successfully adapted to run on GPU devices to improve its run-time performance [16] [5] [21].

GPU devices provide massive parallelism that can potentially reduce the run-time of computationally intensive tasks. On the other hand, these devices have a limited amount of memory. Most approaches that tackle large datasets reduce their performance when there is not enough memory to allocate the complete datasets and kNN structures on the GPU device. Furthermore, some

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

30 of these methods just do not work if the dataset is too large. Most approaches also require sorting all the distance values or use suboptimal methods in order to locate the neighborhood, not taking advantages of all the possibilities that the GPU devices offer.

In the literature, some authors have tried to overcome these limitations. Are-
35 fin et al. [5] reduce the usage of memory dividing the computations in square-shaped portions, but the data structures required still limit the run-time performance. Komarov et al. [21] propose a quicksort-based selection method in order to improve the performance of that part of the kNN rule, but their design requires a high amount of synchronization operations that hinders the run-time
40 performance obtained.

In this paper, we propose a design of the kNN rule, called GPU-based scalable and memory efficient kNN (GPU-SME-kNN), which addresses the aforementioned issues. To do so, we introduce two novel approaches:

- An incremental neighborhood computation scheme that eliminates the
45 dependencies between dataset size and memory footprint. This scheme allows fully customization of its parameters and takes advantage of asynchronous memory transfers, making the required structures fit into the available memory for a broad range of GPU devices while delivering high run-time performance independently of the number of instances of the
50 dataset. This is detailed in Sections 4.1 and 4.2.1.
- An efficient quicksort-based selection design that avoids synchronization operations and has an enhanced pivot-selection method to provide a high performance and scalable solution. This is detailed in Section 4.2.2.

GPU-SME-kNN has been tested with two large datasets from the UCI [1]
55 repository [6]: Poker, with 1 025 009 instances and KDDCup 1999 with 4 898 431 instances. Increasing datasets sizes (up to the full size) and different values of the k parameter have been used in order to thoroughly study the behavior of

[1] http://archive.ics.uci.edu/ml

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

the GPU-based kNN rule in terms of scalability. The results focus on the run-time performance and the memory requirements of the method. Given that our algorithm is designed to improve the efficiency of the kNN rule, but it does not change its behavior, it is not necessary to evaluate its predictive capacity. GPU-SME-kNN is compared with well-known GPU-based kNN approaches showing a good performance.

The rest of the paper is organized as follows: Section 2 presents the kNN rule and the lazy learning family algorithms. Section 3 introduces how GPU devices work and summarizes the previous approaches of GPU-based kNN rule. Section 4 explains our design for the kNN rule. Section 5 shows the results obtained on the experiments performed. Section 6 studies design modifications that our algorithm requires to be applied to lazy learning algorithms and the results obtained. Finally, Section 7 presents the conclusions.

2. The k Nearest Neighbor rule and Lazy Learning

This section summarizes the main aspects of the k Nearest Neighbor rule and family of lazy learning algorithms that will be referred to in the design of the GPU-based method. Section 2.1 explains the kNN rule and Section 2.2 presents the lazy learning algorithms characteristics.

2.1. The k Nearest Neighbor rule

The k Nearest Neighbor rule (kNN) [11] predicts the class of a test instance as the majority class of the k training instances that have the smallest distances to that test instance [12]. This means that each test instance is compared with every training instance, measuring the distance between them. These distances are checked in order to find the k smallest values and the training instances that correspond to the selected distances are used to predict the class of the test instance.

Although different distance measures can be used [34] [26] [32], the well-

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

known Euclidean distance is typically used:

$$d(x, y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2} \qquad (1)$$

where $d(x, y)$ is the distance between instances x and y and D is the number of attributes of the problem.

The kNN rule is usually applied to a set of test instances. If M is the number of test instances and N the number of training instances, the algorithm requires $M \times N$ distance computations and M selections of k instances from an array of N elements. When training and test set sizes increase, the distance computations increase quadratically. The number of selection operations increases linearly with the value of M and the computational cost of each operation increases with the size of N in a way that depends on the specific selection method used but which is, at least, linear. This increase of the number of operations makes the application of this rule really difficult for large datasets.

2.2. Lazy learning

Typically, the process of learning from data involves the generation of some kind of model, from the training data. This model is used to classify the instances of the test set. The family of lazy learners [3] skips this general model creation. When a test instance is evaluated, these algorithms compute a specific model that relates that test instance with the training set and use that model to classify it.

Lazy learning algorithms usually have greater storage requirements and high computational cost when evaluating a test instance than other algorithms. Algorithms that build a model can discard the training instances once the model is built reducing the memory requirements. Moreover, the evaluation of a model is almost inexpensive compared to the cost of building that model, reducing the computational cost for non lazy learning algorithms. The impact of these two issues increases with the dataset size.

This behavior can be observed on the kNN rule because it belongs to the

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

family of lazy learners [15]. There are several lazy learning algorithms that are based on the kNN rule and have the same computational issues.

3. Graphics Processing Units and NVIDIA CUDA

115 Graphics processing units (GPU) were originally created to offload the computations related to 3D graphics on games and design applications from the CPU device into specialized hardware. Modern GPU devices provide a specific processor with a Single Instruction Multiple Data (SIMD) architecture to handle these computations efficiently. NVIDIA CUDA [1] is a hardware/software 120 architecture that allows the use of NVIDIA [2] GPU devices for general purpose programming.

CUDA presents GPU devices as parallel coprocessors with their own memory, caches and registers that can cooperate with one or several CPU cores. In order to take advantage of the characteristics of GPU devices, it is required to 125 redesign the algorithms, determine which operations of the algorithm match the characteristics of each device and the amount of data required to be transferred between devices. To produce efficient GPU based programs or systems, it is mandatory to know some technical aspects of GPU devices (Section 3.1). Once these aspects have been studied we will briefly review the approaches to the 130 kNN rule that can be found in the literature (Section 3.2).

3.1. Technical aspects of GPU devices

Functions are run on GPU devices dividing the workload into a set of threads which share the same code but operate on different data. These functions are called kernels and the set of threads of each kernel is called grid. Threads within 135 a grid are grouped into blocks of threads.

At the hardware level, a GPU device has a set of computing cores that are grouped into stream multiprocessors (SMX). When a grid is run on the GPU, each block is assigned to one SMX, as shown in Figure 1. It is possible to synchronize all threads that belong to a block. However, there is no efficient 140 synchronization method for threads in different blocks.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

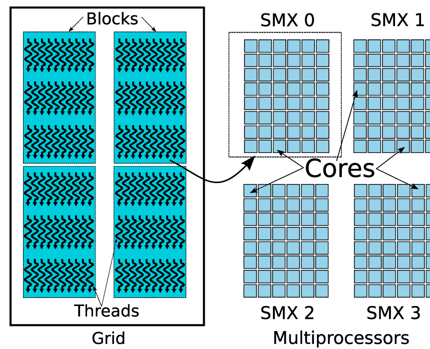


Figure 1: Threads, blocks and multiprocessors. Each block is run on the same multiprocessor.

Each block is divided into groups of 32 threads called warps, which are run synchronously on a SMX. All threads within a warp execute the same instruction (in a parallel way) at the same moment. In case of code divergence within the warp, like a conditional instruction with different results, the execution is serialized penalizing the run-time performance.

Regarding data storage and access, as in CPU devices, a GPU device has a memory hierarchy from large and slow memory banks to small and fast registers including several cache levels. The fastest cache (L1) is available to developers on demand. Each SMX has its own L1 cache but its size is limited. The L1 cache is commonly known as shared memory.

Global memory is the last level of the hierarchy, and it is the largest memory area of a GPU device but also the slowest. The most efficient way for a large number of threads to simultaneously request data from global memory is to perform these parallel requests in a coalescent way: consecutive threads in a block have to request consecutive memory positions at the same time.

In order to start the computation, the input data used by the kernels needs to be copied from the computer main memory to the GPU device global memory. This memory transfer can be done either synchronously or asynchronously. By using asynchronous copies it is possible to run a kernel while copying data that will be required in subsequent kernel calls.

When a kernel function is called, the programmer has to set the number of

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

threads per block, blocks per grid and shared memory required by the kernel. The maximum number of simultaneous blocks per SMX and warps per SMX is device dependent. The resources required by each block limit the number 165 blocks that can be run in parallel.

3.2. GPU-based approaches to kNN

This section presents a brief summary of the most relevant proposed GPU-based methods that tackle the computational issues of the kNN rule. All these approaches divide the kNN rule into two parts: the computation of the distances 170 and the identification of the nearest neighbors. The differences among them rely on how these steps are solved.

As the distances are computed in a separate stage from the selection, a distance matrix is built grouping the distance array related to each test instance. In the second step, several selections are performed in parallel on the different 175 rows of the matrix.

Kuang et al.[22] propose to compute one distance per thread for the distance matrix calculation and sort the distances array of each test instance to get the k nearest neighbors. The distance matrix is split into blocks of a predefined number of threads that compute a distance operation. Each matrix row is 180 sorted using a radix sort method that is computed by a block of threads.

Garcia et al.[16] use the previous distance matrix calculation scheme but they use an insertion sort method instead of radix sort. Both approaches also differ in the way the sorting is done. Garcia et al. compute one sort operation per thread instead of one per block. The code of this implementation of the 185 kNN rule is available on-line and is used as reference for comparisons in other work [19] [31].

Kato et al.[20] propose a design that is also suitable for several GPU devices. The distance matrix is split into blocks of rows where each thread computes the distances for a matrix row. The selection method is performed with one block 190 per test instance. The neighborhood is built in shared memory using an insertion approach.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

GPU-FS-kNN, presented by Arefin et al.[5], divides the computation of the distance matrix into squared chunks in both dimensions. Each chunk is computed using a different kernel call, reusing the allocated GPU-memory. The
195 distance computation kernel divides the chunk into smaller square subsets, one per block. Training and test data of each square subset are copied to shared memory and then each thread computes a Pearson distance. A selection step is performed after each chunk is processed with a modified version of the insertion sort technique. This process is performed using one thread per chunk row. The
200 neighborhood computed in a previous chunk is reused for the next chunks that correspond to the same test instances. The code of this algorithm of the kNN rule is available on-line.

Jian et al.[19] use the same approach as Kuang et al. and Garcia et al. to compute the distance matrix. For the selection step, they propose a method
205 that uses several blocks per test instance. Each block selects k distances and then the results of all blocks are combined iteratively.

Komarov et al.[21] modify the selection step with a quicksort-based selection. Each block performs a selection operation with a large number of threads per block. The matrix computation uses the Kuang et al. and Garcia et al. scheme.
210 The authors of this method have not made its code available. In order to compare our method to it, we have re-implemented this approach. The source code is available with the rest of the code related to this work at: `http://sci2s.ugr.es/GPU-SME-kNN/`

There are other approaches in the literature related to the kNN rule issues.
215 Some of them use FPGA devices [27] but they do not outperform GPU-based techniques in run-time. Only when power consumption is considered these devices become an interesting option. There are also kNN versions for specific problems, like text classification [18], but the optimizations performed are focused and dependent on the specific problem or the distance measure used and
220 cannot be applied to other situations. Some pieces of work consider the resolution of only one test query at a time[7] but this approach is not suitable for cases where a large number of test queries is available, like in Lazy Learning

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

algorithms.

The majority of these approaches assume that the distance matrix and the rest of the data structures fit on GPU memory but this is not possible for large datasets, like the KDDCup 1999 dataset. The solution provided by Garcia et al. for this issue is to divide the test set into parts and compute these parts iteratively but without considering the use of asynchronous memory copies to avoid idle times. Furthermore, this approach affects the run-time performance of the different steps of the kNN rule and it is not suitable for the constantly increasing sizes of the datasets.

Arefin et al.'s algorithm is the exception in terms of memory requirements assumptions, as the computation is performed chunk-wise. As the authors expose in their paper, the memory footprint is reduced because some structures are reused during the computation of the algorithm. However, this method is still limited by the memory requirements because the complete dataset (training and test sets) is copied to GPU memory. In that case, a smaller chunk size can be used but this reduces the run-time performance.

Our design (Section 4) tackles the memory related issues present in the literature, overcoming the dependence between dataset size and memory required. Nevertheless, efficient kernels have been designed for each step of the computation which improves the run-time performance.

4. A GPU-based kNN rule for large datasets

There are two main issues inherent to large datasets processing: the computational complexity, in terms of amount of operations, and the memory required to store the structures of the algorithm. Our method introduces an incremental neighborhood scheme to reduce the memory requirements. This scheme has to be combined with an efficient memory transfer design between devices. These points are addressed in Section 4.1.

The design of the kernel functions, Section 4.2, has a high impact on the run-time performance achieved. We split the distance computation into two

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

different kernel functions: the first one computes the square of the distance and the second one performs the square root operation, but only on the selected neighbors. For the selection step, we propose a quicksort [17] based selection
255 that takes advantage of the iterative neighborhood computation characteristics.

Section 4.3 presents how memory requirements of GPU-SME-kNN are determined by the different algorithm parameters regardless of the dataset size.

4.1. CPU-GPU interaction model

The main steps of the kNN rule are the ones related to the computation of
260 the distance matrix and the selection of the k nearest neighbors.

GPU devices have a small amount of memory compared to desktop and server computers. Some structures, like the distance matrix, do not fit into device memory. The solution to this problem is to split the computations in steps and take advantage of the asynchronous memory copy operations to avoid
265 idle times.

Our method splits the computations of the distance matrix and neighborhood. The method of Arefin et al. [5] also splits these computations, however, this technique requires to compute square-shaped portions of the matrix and to copy the complete training and test sets to device memory. Depending on the
270 GPU device and the dataset used, the size of the matrix portion may need to be reduced, leading to a loss of run-time performance. Our method overcomes these limitations providing a more customizable scheme that can be adapted to almost every GPU device. The details are discussed in the next section.

4.1.1. Incremental neighborhood computation

275 The distance matrix is the most memory demanding data structure needed in the kNN rule. The size of this matrix is $M \times N$, where M is the number of test instances and N the number of training instances.

The most common solution in order to make the distance matrix fit into memory is to divide it into strips. The distance matrix is split into M/m
280 matrices of size $m \times N$, where m is a portion of the test set small enough to

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

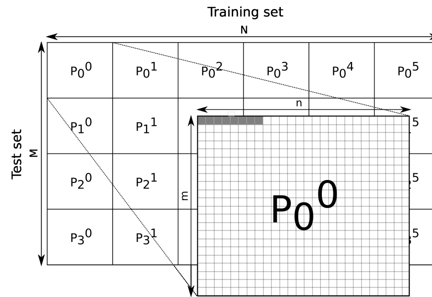


Figure 2: The distance matrix of size $M \times N$ is partitioned in pieces of $m \times n$ elements.

make the matrix fit into device memory. The algorithm iterates through the test set to complete the computation. This is the solution used by Garcia et al. [16].

However, this solution is limited by the size of the training set because the N instances of the training set and m instances of the test set need to be kept on device memory in order to perform the distance computation. In some scenarios, the GPU device could not have enough memory even when setting m to 1.

Our design is based on the algorithm of Arefin et al. [5] where the matrix is split in both dimensions. As shown in Figure 2, a portion, P_i^j, of the matrix of size $m \times n$ is computed on each step, where m and n are portions of the test and training sets, respectively. The algorithm iterates in both dimensions performing $N/n \times M/m$ iterations, covering the whole training set for each test chunk before moving to the next one. On Arefin et al's method n and m have the same value, in order to split the chunk easily into square-shaped blocks. Our distance computation model, explained in detail in Section 4.2.1, allows arbitrary values of n and m that can be tuned to offer good run-time performance and fit into the available memory for a broad range of GPU devices.

All the components in a strip of the matrix, P_i^0 to $P_i^{N/n}$, are needed to find the neighborhood of the chunk i of the test set. To reduce the amount of memory needed, a local selection of the neighbors has to be performed. A selection operation can be computed for each chunk of the matrix, as shown in Figure 3, and then a global selection is performed on the local solutions. This

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

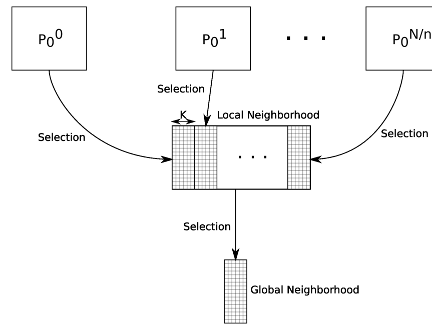


Figure 3: Local neighborhood selection scheme.

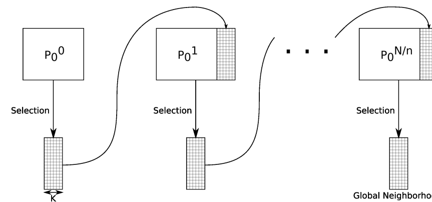


Figure 4: Incremental neighborhood selection scheme.

2-step computation scheme reduces the amount of memory needed. It keeps in memory $m \times k$ values instead of $m \times n$, taking into account that k is typically smaller than n. However, this strategy might not be enough for very large values of N.

Our algorithm computes the neighborhood in an incremental way, combining each chunk of the matrix with the resulting k neighbors of the previous chunks, as Figure 4 shows. This approach does not need a global selection step because the last local selection includes all the results obtained.

Furthermore, the amount of memory required is fixed regardless of the sizes of training and test sets, depending only on the values of m, n, k and the number of attributes of the dataset. These four values define the sizes of all data structures required for the computations and all of them are independent of the dataset size. The size of m and n can be decided in an arbitrary way in order to maximize the run-time performance and make the data fit into device memory. Section 4.3 details how to compute the exact memory footprint.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

4.1.2. CPU-GPU memory transfers

CPU and GPU devices have different and exclusive memory banks. The
320 input data and the results have to be copied between devices. The pseudocode presented on Algorithm 1 includes these interactions.

Algorithm 1: Proposed algorithm pseudocode.

```
input  : Training and test sets, k
output: k nearest neighbors of each test instance in the training set.
1  copyTestPieceAsync_1
2  copyTrainPieceAsync_1
3  for i ← 1 to M/m do
4      checkTestCopy_i
5      for j ← 1 to N/n do
6          checkTrainCopy_j
7          computeDistanceMatrix_{i,j}
8          if j = N/n then
9              copyTestPieceAsync_{i+1}
10             copyTrainPieceAsync_1
11         else
12             copyTrainPieceAsync_{j+1}
13         end
14         computeSelection_j
15     end
16     copyNeighborhood_i
17     checkResultCopyAsync_i
18     copyResultPiece_i
19 end
```

copyTestPieceAsync and *copyTrainPieceAsync* represent the asynchronous copy of the instances required to compute a chunk of the matrix. When using asynchronous copies it is required to check if the copy has finished before using

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

325 the data. In Algorithm 1, $checkTestCopy$ and $checkTrainCopy$ represent this check. Except for the initial copy to start the algorithm, these copies (lines 8 to 13) are made in parallel during the selection process (line 14).

$checkResultCopy$ and $copyResultPieceAsync$ are, respectively, the safety check and the asynchronous copy of the neighborhood for a certain piece of the 330 test set. The data structures used to keep the results in memory are the same for all the iterations. In order to avoid checking if the copy has finished on every iteration of the training set related loop, the final neighborhood is copied to a different structure, this operation is represented by $copyNeighborhood$ in Algorithm 1.

335 *4.2. Kernel design*

Two different steps have been defined: the computation of a chunk of the distance matrix and the incremental selection of the k nearest neighbors. However, the square root calculation of the distance measure is applied only to the selected neighbors, in order to improve the run-time performance of the algo- 340 rithm.

Therefore, three different kernel functions are used: the first kernel computes a chunk of the distance matrix, line 7 on Algorithm 1, the second kernel performs the selection of the k nearest neighbors, line 14 on Algorithm 1, and the third kernel computes the square root operation on the selected neighbors while the 345 neighborhood is copied, line 16 on Algorithm 1. The following sections explain each kernel details.

4.2.1. Distance matrix computation

The distance matrix kernel computes the distances of one piece of the matrix of size $m \times n$, as commented on Section 4.1.1. Our method introduces a com- 350 pletely new distribution of the kernel threads that allows the customization of the parameters while delivering a high run-time performance. The kernel grid has m blocks, one per test instance, and d threads. In Figure 2 zoomed-in area, these d threads are represented as filled cells for the first block. Each thread of

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

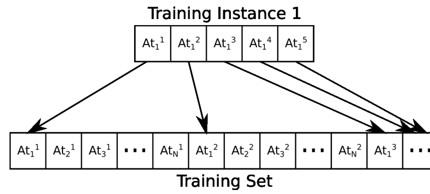


Figure 5: Dataset stored on memory. At_j^i represents attribute i of instance j.

the kernel computes several distances between 1 instance from the test set and n/d instances from the training set. The number of threads per block, d, is set to a value that delivers good performance on the GPU device regardless of the value of n that only defines the number of distance computations per thread.

None of the existing approaches uses such a thread distribution scheme. Most of them compute one distance per thread, introducing a high level of parallelism with extremely light threads. Kato et al. [20] is an exception, as all distances in a row are computed by one thread. This approach provides a higher workload per thread but reduces the degree of parallelism. Our design tries to find the right balance between both strategies in order deliver a better performance.

The input data is stored in a coalescent way to provide an efficient memory access. The distances are computed in parallel so all threads will request first the first attribute of the instance, then the second and so on. The dataset is stored in memory as an array of floating numbers as shown in Figure 5. As each block is related to one test instance, a thread computes several distances of the same test instance. Copying the values of the attributes of the test instance to shared memory provides a more efficient access rather than each thread requesting these values independently.

4.2.2. Neighborhood selection

The quicksort algorithm [17] is a well known algorithm to sort an array. The algorithm selects one element of the array (pivot) and divides the array into two parts: the left part stores the elements smaller than the pivot while the right part stores the elements greater than the pivot. Repeating the process in

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

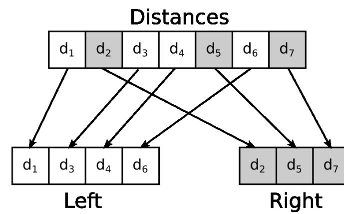


Figure 6: Non coalescent memory writings when dividing the vector.

a recursive way for each part finally gets the array sorted.

The majority of approaches in the literature use a sorting method to compute the neighborhood of the instances. This requires a high number of operations in order to completely sort the array. Our algorithm relies on a selection method to reduce the number of operations. The previous sorting technique can be adapted to select the k smallest elements of an array, repeating the process only for one of the parts of the vector: on the left part if this part has more than k elements or on the right part if the left part has less than k elements. In the second case, the left part of the array and the pivot are part of the selected elements.

This selection method is also used by Komarov et al.[21] in their method. However, this computational step has been improved in our method to avoid synchronization operations hence providing better run-time performance.

The ad-hoc design of this algorithm for GPU devices is to use a kernel to create the left and right parts of the array in a parallel way and then call the same kernel in a recursive way as it is done on CPU devices. This solution would be suitable only for the latest NVIDIA devices, which allow recursive kernel calls. However, the recursive step algorithm can be easily transformed into an iterative solution that works for most GPU devices.

Different structures are used for the distances array, the left part and the right part. At the end of each iteration, the left part or the right part, depending on their sizes, becomes the distance array and the former distance array is used as one of the parts.

Aside from this, the creation of the left and right parts of the array does not

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

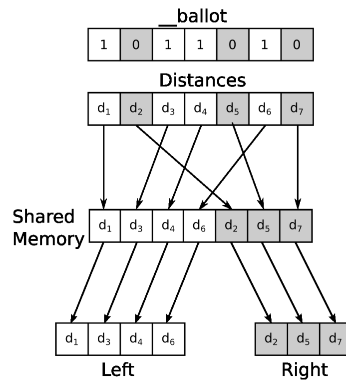


Figure 7: Local neighborhood selection

suit the characteristics of the GPU devices because it involves non coalescent memory accesses. With a grid configuration of m blocks (one per test instance), several comparisons can be done in parallel but, depending on the result of the comparison, each thread will need to write on a different part of the memory penalizing the run-time performance, see Figure 6. To solve this problem, the results are written in two steps. The first step writes the element to a shared memory array in a non coalescent way grouping the elements smaller than the pivot on one side and the elements bigger on the other side. The second step writes the element to its final position in global memory using two coalescent memory accesses, one for the left part and another for the right part of the array, as shown in Figure 7. The right part memory access can be skipped once the left part has more than k elements, improving the run-time performance.

However, to find the position where each element has to be written is a problem in itself. Each thread needs to know how many elements from the threads with a lower index are greater or smaller than the pivot. CUDA provides functions, *__ballot* and *__popc*, to solve this problem but only within a warp. The *__ballot* function creates a 32-bit integer where each bit is the result of a predicate evaluated by each thread of the warp, see Figure 7, while the *__popc* function counts the number of bits set to 1 in a 32-bit integer. Using the proper bit mask for each thread, it is possible to get the writing position within the warp.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

A grid of m blocks and 32 threads per block can select the neighborhood but it might not provide enough workload for the GPU device. Generally, a higher number of threads per block provides a better performance. There are two ways of increasing the number or threads: using more than 32 threads to build the array parts or processing different arrays on each warp.

The first solution is used by Komarov et al.[21]. It requires sharing the values obtained with *__ballot* on each warp and to set synchronization points to ensure the values are correct. These requirements would penalize the run-time performance so we decided to use the second solution. The neighborhood of each test instance is computed by a single warp and each block computes several neighborhoods. However, if the number of test instances is small, Komarov's et al. algorithm obtains better GPU occupancy ratios. Our design uses fewer threads per selection step requiring a larger number of test instances to fully occupy the resources of the GPU device.

The value of the pivot used in the quicksort algorithm has an impact on the run-time performance. The pivot value is usually selected as the median of the first, last and center values of the array as an approximation of the median of the array. Using the median value of the array as pivot produces equally sized array parts, halving the size in each iteration, but we can use a more aggressive strategy thanks to the incremental neighborhood scheme. If a distance is larger than the farthest neighbor from the last chunk, that instance is not going to be a part of the neighborhood because there are already k smaller values. Setting the pivot to the farthest neighbor distance of the previous iteration of the algorithm focuses the effort on the interesting values providing a better performance. For the first matrix chunk of each test chunk, this solution cannot be applied, since we do not have a previous neighborhood, so the usual heuristic is used.

When the left part of the array is smaller than k, those distances and the pivot belong to the final neighborhood of that step. However, as the selection algorithms continues and needs to reuse that memory area, these values are copied to a different array of $m \times k$ elements.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

4.2.3. Square root calculation

The square root operation is a costly operation and it is not required to select the neighborhood of an instance. In our classification scenario, this operation can be skipped. However, it is performed for two reasons: to provide 455 the accurate distance information in case any future application needs it and to be able to establish a fair comparison with other approaches that perform this operation.

Other algorithms, like Garcia et al. [16], also use a specific kernel for the square root computation, although it is only briefly commented in the corre- 460 sponding papers, as a minor optimization. In our design, this kernel has been combined with one of the required memory transfer operations, as commented in Section 4.1.2. This provides better run-time performance than performing two separate operations.

All the selected distances are located on coalescent memory, it can be con- 465 sidered as a long array. This array has a size of $m \times k$ elements. In order to get the highest possible occupancy of the GPU device, we split the array to create $\frac{m}{128}$ blocks of 128 threads. Each thread performs a square root operation on k distances and copies their respective indexes in the training set to the separate structures.

470 4.3. Total memory required

As mentioned before, the distance matrix size can be defined in an arbitrary way depending on the values of the parameters k, m and n. The amount of memory required for the rest of the operations also depends on these parameters and on the number of attributes of the dataset, D.

475 This way, it is possible to define an expression that represents the amount of elements required to keep on memory for each configuration of the parameters:

$$D \times m + D \times n + 6(m \times (n + k)) + 4(m \times k) \quad (2)$$

The exact amount of memory can be obtained weighing each part of the equation with the size, in bytes, of the type of elements (floating point numbers, integers)

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

that are stored. The memory requirements of the experiments performed are
480 shown in Section 5.3.

5. Experimental results of GPU-SME-kNN

Different experiments have been carried out and the results obtained are presented here. Section 5.2 presents the hardware and datasets used and the experiments performed, Section 5.3 shows the results obtained and Section 5.4
485 analyses the results.

5.1. Experiments

Although the design of the algorithm for GPU and CPU devices changes significantly, the same computations are performed in both devices. This means that the same algorithm obtains the same results regardless of the device. Tak-
490 ing this into account, the experiments have been designed to highlight the efficiency differences between GPU-SME-kNN and the reference implementations.

Two large datasets have been selected for the experiments. These datasets have been subsampled at different sizes to show how the behavior of the algorithm changes as the size of the dataset increases. A 5-fold cross validation
495 scheme has been used with all sizes of the dataset. This scheme reduces the impact that the relative order of the instances within the dataset can have on the performance. Different values of k have also been used. The results shown in Section5.3 are computed as the average times of the values obtained.

The performance of GPU-SME-kNN has been compared with several existing
500 techniques:

1. The technique of Garcia et al. [16], denoted as GPU-Garcia-kNN, available on Github.
2. The technique of Arefin et al. [5], GPU-FS-kNN, is also available on-line, but modified to use the Euclidean distance instead of the Pearson distance
505 in order to make a fair comparison.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

3. The technique of Komarov et al. [21], denoted as GPU-Komarov-kNN, is not available on-line, but we have implemented their design of the quicksort selection with our incremental neighborhood calculation scheme. Using the same scheme for the distance calculation the differences rely on the design of the selection algorithm. In addition, our scheme allows this method to scale to problem sizes that the original one cannot address. The parameters n and m have been set to 65536 and 2048, respectively, in order to provide a matrix chunk of a size similar to the one used in [21].

A website associated to this paper has been created that includes the datasets, results and code of GPU-SME-kNN and GPU-Komarov-kNN used in this work. The URL for this website is: `http://sci2s.ugr.es/GPU-SME-kNN/`

5.2. Hardware and datasets

The experiments have been performed on a server-class computer equipped with a high-end GPU device. This computer has an Intel Xeon E5-2630 processor at 2.30 GHz. The GPU device is a NVIDIA Tesla K20m with 5GB of RAM memory and 2496 CUDA cores. Nevertheless, the proposed design can be run on a computer with lower specifications.

Two large datasets from the UCI repository [6] have been used in the experiments:

- The poker dataset has 1 025 009 instances, 10 attributes and 10 classes. The dataset has been subsampled at sizes ranging from 50 000 to 1 000 000 instances in steps of 50000 instances.
- The KDDCup 1999 dataset has 4 898 431 instances, 41 attributes and 5 classes. This dataset has been subsampled in steps of 250 000 instances from 250 000 to 1 500 000 instances and in steps of 500 000 instances for larger sizes.

Different experiments have been performed with these datasets using a 5-fold cross validation scheme for all sizes. The experiments use k values of 5, 100

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

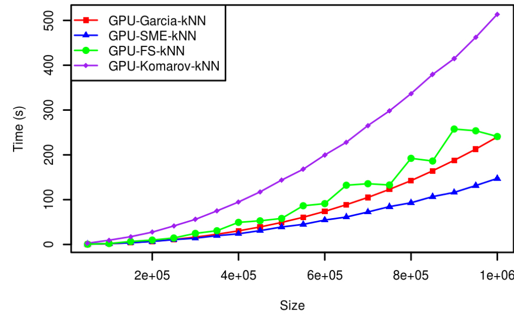


Figure 8: Poker dataset results with $k = 5$.

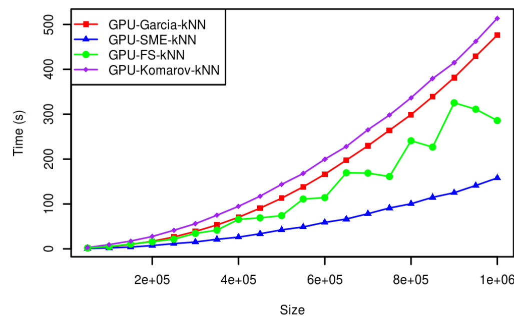


Figure 9: Poker dataset results with $k = 100$.

and 1000 for both datasets. Some k values might be too large to offer accurate classification ranges but the objective is to assess the scalability of the evaluated methods in relation to k.

5.3. Empirical results

This section presents the experiments and the results obtained on the previously mentioned hardware and datasets. For all the experiments the value of m was set to 16384 and the value of n to 2048, the number of threads per block for the distance matrix kernel, d is 256.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

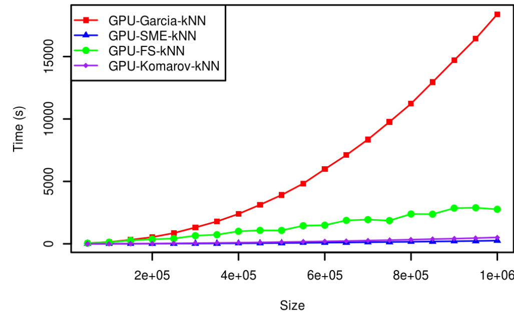


Figure 10: Poker dataset results with $k = 1000$.

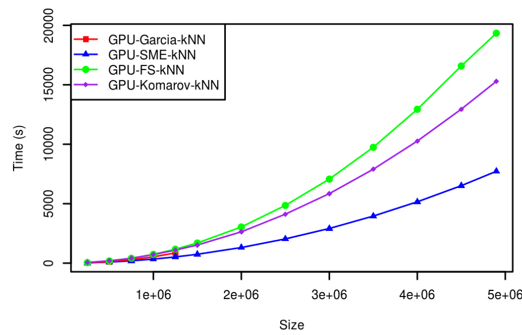


Figure 11: KDDCup 1999 dataset results with $k = 5$.

Figures 8 to 10 present the results for the poker dataset. Three different values of k have been used with this dataset: 5, 100 and 1000.

Figures 11 to 13 present the results for the KDDCup 1999 dataset. The
545 algorithm of Garcia et al. was able to complete the experiments successfully for this dataset only up to 1 250 000 instances. For bigger sizes the experiments failed due to memory requirement problems. Tables 1 to 3 compare the results of all approaches for these values. GPU-FS-kNN also presents some issues with this dataset and the largest value of $k(1000)$. In this case, the software provided
550 by the authors behaves in an abnormal way for sizes larger than 1.5 million of

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

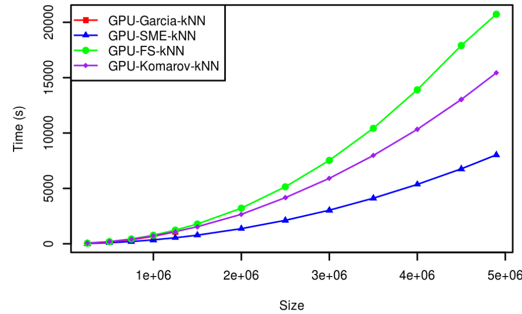


Figure 12: KDDCup 1999 dataset results with $k = 100$.

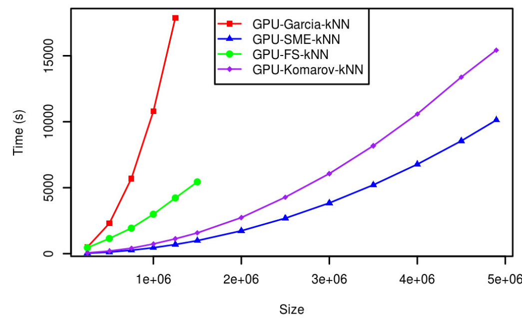


Figure 13: KDDCup 1999 dataset results with $k = 1000$.

instances and it does not provide accurate results.

5.4. Analysis of the results

As the results of the previous section presents, GPU-SME-kNN outperforms the results of the other approaches. The strategy for dealing with large distance matrices causes high run-times performance differences. GPU-Garcia-kNN stores full distance matrix strips on GPU memory to compute the neighborhood. The matrix strip width becomes smaller as the size of the training set increases and this reduces the performance of the design. GPU-SME-kNN keeps the width of the matrix, through the incremental neighborhood computation,

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

Table 1: KDDCup 1999 dataset time results, in seconds, with $k = 5$

Size	GPU-Garcia-kNN	GPU-SME-kNN	GPU-FS-kNN	GPU-Komarov-kNN
250 000	29.052	**25.192**	45.127	60.837
500 000	121.281	**90.243**	180.955	196.098
750 000	284.577	**194.467**	409.278	407.561
1 000 000	528.100	**338.979**	732.452	719.672
1 250 000	862.306	**522.74**5	1170.310	1093.420

Table 2: KDDCup 1999 dataset time results, in seconds, with $k = 100$

Size	GPU-Garcia-kNN	GPU-SME-kNN	GPU-FS-kNN	GPU-Komarov-kNN
250 000	37.931	**26.182**	50.388	59.856
500 000	158.213	**93.964**	193.651	194.744
750 000	366.198	**202.579**	432.981	405.127
1 000 000	667.465	**352.820**	772.101	725.162
1 250 000	1083.842	**544.529**	1233.264	1097.123

providing the desired performance regardless the training set size.

The run-time performance differences with GPU-Garcia-kNN also rise when value of k is increased. A higher value of k requires more space to store the neighborhood, reducing the amount of memory available for the distance matrix which produces the same effect that happens when the training set size increases. However, the difference is also introduced by the selection method. GPU-Garcia-kNN uses an insertion method to select the neighborhood of a test instance on each thread of the kernel. By increasing the value of k the probability of code divergence also increases. The divergence happens when some threads in a warp find a neighbor candidate at some position but not all threads find it. The threads that did not find a neighbor have to wait until the threads that find one make the required computations.

The code divergence problem also affects GPU-FS-kNN. However, the modifications of the insertion scheme introduced in its selection method lower the impact of the divergence on the run-time performance. Increasing the value of k

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

Table 3: KDDCup 1999 dataset time results, in seconds, with $k = 1000$

Size	GPU-Garcia-kNN	GPU-SME-kNN	GPU-FS-kNN	GPU-Komarov-kNN
250 000	496.624	**33.890**	449.668	61.665
500 000	2297.120	**119.163**	1150.744	199.355
750 000	5691.280	**259.552**	1932.540	415.234
1 000 000	10 788.800	**448.624**	2992.348	736.441
1 250 000	17 885.280	**694.892**	4221.566	1125.497

Table 4: GPU-SME-kNN GPU memory usage in MB

k	Poker dataset	KDDCup 1999 dataset
5	1158	1162
100	1247	1251
1000	2091	2159

575 also reduces the performance because this method does not uses asynchronous memory copies. This introduces computation idle times while the results are copied from GPU to CPU memory. The distance computation scheme of GPU-FS-kNN, which is similar to our incremental neighborhood computation, allows this method to complete almost all the experiments but the performance is 580 affected by the issues mentioned above.

GPU-SME-kNN does not suffer from this kind of code divergence problems, as a warp collaborates to select the neighborhood. Furthermore, this selection method reduces the impact of the value of k: each iteration the same steps are followed, the value of k only changes the number of iterations made. However, 585 the most important factor on the run-time performance of the selection step is the incremental neighborhood computation.

Our selection method requires a large number of memory accesses to read the array and store the left and right parts. Although these accesses are programmed on a coalescent way and are efficient, a large number of iterations could 590 be required depending on the quality of the pivots provided by the heuristic, especially when k is set to a small value. The incremental neighborhood tech-

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

nique provides the maximum distance to be considered a neighbor candidate, reducing the number of iterations.

Our implementation of GPU-Komarov-kNN also uses this idea. However, we cannot see the effect in the results because GPU-Komarov-kNN uses 512 threads per selection step [21] whereas GPU-SME-kNN uses 32. As we commented in Section 4.2.2, by using 32 threads it is possible to avoid synchronization operations, improving the run-time performance of GPU-SME-kNN. In addition, the use of a small number of threads improves the use of the resources of the GPU. If the array size is lower than the number of threads per selection some threads do not perform any computation, but their resources cannot be released until the selection process is finished. This situation is reached in a faster way when a large number of threads per selection is used, taking into account that the size of the array is approximately halved in each iteration of the algorithm. Small values of k highlight this fact. On the other hand, when using large values of k GPU-Komarov-kNN outperforms other approaches. However, as we can see in Figure 13, the differences against GPU-SME-kNN are significant. The situation is similar in Figure 10 but, in this case, the results of the GPU-Garcia-kNN technique affect the scale of the plot and it cannot be observed.

For scenarios with a small number of test instances the run-time performance differences between GPU-SME-kNN and GPU-Komarov-kNN would decrease. The higher number of threads used in the selection step makes GPU-Komarov-kNN reach the maximum occupancy of the GPU device faster than GPU-SME-kNN. Therefore, under some circumstances, GPU-Komarov-kNN could outperform GPU-SME-kNN. However, in the presented 5-fold cross validation results, GPU-SME-kNN exhibits a better performance, especially as the number of test instances increases.

In cases with similar number of training and test instances the run-time performance differences between GPU-SME-kNN and GPU-Komarov-kNN would be even larger. This situation can be found in different scenarios, for instance, some steps of lazy learning algorithms require computing the neighborhood of the training test. Taking into account that the differences are already significant

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

in the results presented in this section, only GPU-SME-kNN can compute these steps in a reasonable time.

625 Table 4 shows that the amount of memory used by GPU-SME-kNN is similar for both datasets regardless of the difference of more than 3 500 000 instances. The memory requirements difference is actually caused by the different number of attributes of both datasets. In all cases, the amount of memory required is relatively small making GPU-SME-kNN suitable for most current GPU devices.

630 6. GPU-based Lazy learning

The design patterns used on the GPU-based method for the kNN rule should also be suitable for lazy learning algorithms. The following sections detail the algorithms that have been adapted (Section 6.1), the design modification that they required (Section 6.2) and the results obtained (Section 6.3).

635 *6.1. Algorithms*

We have selected three different algorithms from the ones available on the KEEL software tool [4] in order to test our design. These algorithms are described in this section.

6.1.1. Center$k$NN

640 The Center$k$NN technique [14] is a lazy learning algorithm based on the kNN rule that modifies the distance computation. The algorithm computes the center of each class, as the average of the instances that belong to that class and uses this value to modify the reference point used to measure the distance.

To compute the distance between a test instance, y, and a training instance, 645 x, the algorithm computes first the line that passes through x and the center of its class, c_x, then, projects y onto that line and uses the resulting point p_y to measure the distance. Figure 14 shows these values graphically.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

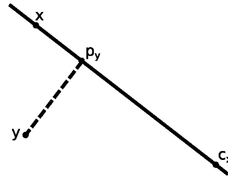


Figure 14: Center$k$NN distance measure, the dashed line is the distance measured

6.1.2. kNN adaptive

The kNN adaptive algorithm [33] weighs the measured distance of the kNN rule. The weight used is specific for each training instance. As a previous step before using the kNN rule, each training instance computes the distance to the closest instance that does not belong to its own class within the training set.

That distance is used as weight when the kNN rule is applied. The distance between a training and a test instance is divided by this weight. This weighting scheme considers training instances that are close to the frontier of two classes less reliable as neighbors, than the ones in the center of clusters of the same class.

6.1.3. Symmetric kNN

The Symmetric kNN algorithm [25] computes the kNN rule in both directions. In particular, the algorithm computes the kNN rule of each training instance compared to the training set, as a first step. When the distance between a training instance and a test instance is computed, it is compared to the distance of the farthest neighbor of the training instance. The test instance would be part of the neighborhood of the training instance if the distance between them is smaller than the distance from the training instance to the farthest neighbor of the training instance. A training instance is considered part of the symmetric neighborhood when this happens.

This symmetric neighborhood is joined with the regular neighborhood, obtained with the usual application of the kNN rule, to obtain the final neighborhood that classifies the test instance. The join operation is made in a way that avoids double voting if there is an instance in both symmetric and regular

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

neighborhoods.

6.2. GPU design modifications

The lazy learning algorithms can use the same distance matrix and incremental neighborhood calculation scheme as the kNN rule. However, these algorithms require a previous step where part of the information they need is calculated. The specific modifications for each method are presented in this section.

6.2.1. Center$k$NN

The Center$k$NN algorithm requires the computation of the centers of each class as a previous step to the kNN rule. The center of a class is computed as the average value for each attribute on a training instance that belongs to that class. This step can be performed on the CPU device when the dataset is loaded into RAM memory. These center values are copied to device memory only once, before the first piece of the matrix is computed.

The projection of the test instance is computed in the distance kernel. The kernel keeps the same structure but it performs more operations to compute the projected instance.

6.2.2. kNN adaptive

The weight values of the kNN adaptive technique can be computed as the kNN rule with $k = 1$ ignoring the training instances that belong to the same class of the instance whose weight is being computed. A modified version of the kNN rule that introduces a void value in the matrix when both instances have the same class is used to compute the weight for each instance as a previous step.

The original kNN rule is also modified in order to include the weighting of the distance. In this algorithm, the square root of the distance needs to be computed for every training instance because the weight is different for each one and can modify the selected instances.

The weights of the training instances are copied chunk-wise to a dedicated array alongside the copy of the training chunk they correspond to.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

6.2.3. Symmetric kNN

To apply the kNN rule in both directions, it is required to know the farthest distance of the neighborhood of each training instance. A modified version of the kNN rule that introduces a void value in the matrix when the training and
705 test instances have the same index is used to compute the neighborhood of the training set as a previous step.

The distance matrix kernel of the original kNN rule is modified to compare the distance obtained with the farthest distance of the neighborhood. The symmetric part of the rule is satisfied when the distance computed by the kernel
710 is smaller than the one stored before. When that happens, the training instances vote, increasing the value of its class in a voting structure specific to this algorithm.

The voting structure has a size of $m \times C$ where C is the number of classes. This structure is set to 0 at the beginning of each strip of the matrix and it is
715 copied with the final neighborhood. The process that assigns the class to the test instance uses these votes as base and adds the votes of the final neighborhood, checking if these instances have already voted to avoid double voting.

6.3. Empirical Results

The lazy learning algorithms have been tested against the CPU implemen-
720 tation available on the KEEL software[4]. These algorithms have been tested with the poker dataset up to 650 000 instances using a 5-fold cross validation scheme. Figure 15 shows the results for the Center$k$NN. The implementation available on KEEL does not allow changing the k value for this algorithm which is set to 1.

725 Figures 16 and 17 present the results for the kNN adaptive and the symmetric kNN algorithms. In both algorithms, k can be set to different values, and for these experiments, the value selected was 5.

The results of the lazy learning algorithms show a similar behavior on the three cases: our approaches reduce the time from hours to minutes. Center$k$NN
730 shows the highest differences, as the computation of the projection introduces

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

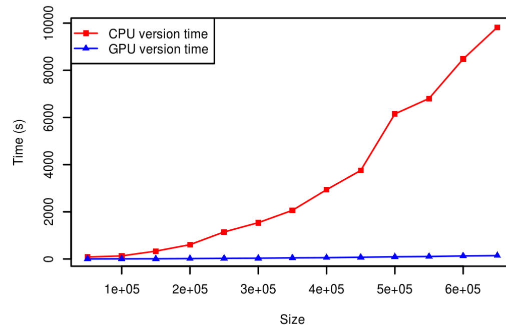


Figure 15: Center$k$NN results on poker dataset.

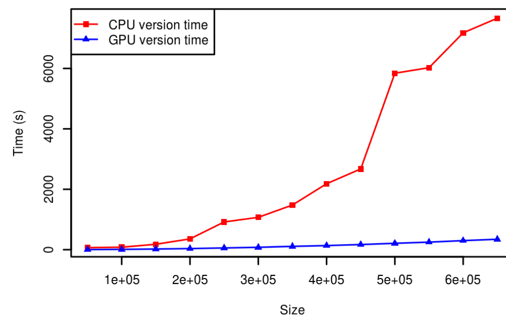


Figure 16: kNN adaptive results on poker dataset.

more floating point computations that can be addressed efficiently by GPU devices. On the other hand, Symmetric kNN requires more comparisons and data accesses, which are addressed less efficiently than floating point operations on GPU devices, making the differences smaller. The performance of kNN
735 adaptive is similar to the symmetric kNN.

7. Conclusions

We have presented a new GPU-based approach for the kNN rule that outperforms the approaches in the literature and provides a high scalability in terms

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

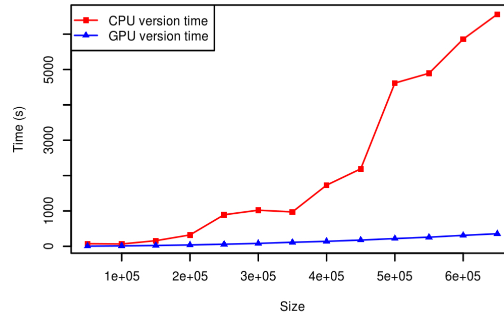


Figure 17: Symmetric kNN results on poker dataset.

of dataset size and k value. GPU-SME-kNN keeps a stable level of memory
740 usage that allows to address any dataset regardless of its size, which was not possible by any of the previous GPU kNN methods. Furthermore, given that (1) the memory footprint of the method can be totally controlled by user-defined parameters and that (2) we do not use capabilities only present in the most recent GPU cards, our method can be efficiently used across a very broad range
745 of GPU devices with varying amount of card memory and CUDA capabilities.

We have also proven that our design is suitable for lazy learning algorithms based on the kNN rule. The run-time performance of the three algorithms presented, Center$k$NN, kNN adaptive and Symmetric kNN, has been improved in a significant way reducing the run-time from hours to minutes.

750 All own code is available as open source, along with the datasets and results, on the website: `http://sci2s.ugr.es/GPU-SME-kNN/`

Acknowledgements

This work was supported by the research Projects TIN2014-57251-P, TIN2013-4720-P and P12-TIC-2958. P.D. Gutirrez holds an FPI scholarship from the
755 Spanish Ministry of Economy and Competitiveness (BES-2012-060450) and a short stay in foreign institutions scholarship (EEBB-I-14-08977).

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

References

[1] CUDA, http://www.nvidia.com/object/cuda_home_new.html.

[2] NVIDIA, http://www.nvidia.com/.

760 [3] D. W. Aha, Lazy Learning, Springer, 1997.

[4] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, F. Herrera, KEEL: A software tool to assess evolutionary algorithms for data mining problems, Soft Computing 13 (3) (2009) 307–318.

765 [5] A. S. Arefin, C. Riveros, R. Berretta, P. Moscato, GPU-FS-kNN: A Software Tool for Fast and Scalable kNN Computation Using GPUs, PLoS ONE 7 (8) (2012) e44000.
URL http://dx.doi.org/10.1371%2Fjournal.pone.0044000

[6] K. Bache, M. Lichman, UCI machine learning repository (2013).
770 URL http://archive.ics.uci.edu/ml

[7] G. Beliakov, G. Li, Improving the speed and stability of the k-nearest neighbors method, Pattern Recognition Letters 33 (10) (2012) 1296 – 1301.

[8] A. Cano, J. Luna, S. Ventura, High performance evaluation of evolutionary-mined association rules on gpus, Journal of Supercomputing 66 (3) (2013)
775 1438–1461.

[9] A. Cano, A. Zafra, S. Ventura, Speeding up the evaluation phase of gp classification algorithms on gpus, Soft Computing - A Fusion of Foundations, Methodologies and Applications 16 (2) (2012) 187–202.

[10] B. Catanzaro, N. Sundaram, K. Keutzer, Fast support vector machine
780 training and classification on graphics processors, 2008, pp. 104–111.

[11] T. Cover, P. Hart, Nearest neighbor pattern classification, Information Theory, IEEE Transactions on 13 (1) (1967) 21–27.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

[12] A. Dhurandhar, A. Dobra, Probabilistic characterization of nearest neighbor classifier, International Journal of Machine Learning and Cybernetics 4 (4) (2013) 259–272.

[13] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification (2Nd Edition), Wiley-Interscience, 2000.

[14] Q.-B. Gao, Z.-Z. Wang, Center-based nearest neighbor classifier, Pattern Recognition 40 (1) (2007) 346–349.

[15] E. Garcia, S. Feldman, M. Gupta, S. Srivastava, Completely Lazy Learning, Knowledge and Data Engineering, IEEE Transactions on 22 (9) (2010) 1274–1285.

[16] V. Garcia, E. Debreuve, F. Nielsen, M. Barlaud, K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching, in: Proceedings - International Conference on Image Processing, ICIP, 2010, pp. 3757–3760.

[17] C. A. R. Hoare, Algorithm 64: Quicksort, Commun. ACM 4 (7) (1961) 321–.

[18] L. Huang, Z. Li, A novel method of parallel gpu implementation of knn used in text classification, in: Networking and Distributed Computing (ICNDC), 2013 Fourth International Conference on, 2013, pp. 6–8.

[19] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, Y. Shi, Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA), Journal of Supercomputing 64 (3) (2013) 942–967.

[20] K. Kato, T. Hosino, Multi-GPU algorithm for k-nearest neighbor problem, Concurrency and Computation: Practice and Experience 24 (1) (2012) 45–53.

[21] I. Komarov, A. Dashti, R. M. D'Souza, Fast k-NNG Construction with GPU-Based Quick Multi-Select, PLoS ONE 9 (5) (2014) e92409.
URL http://dx.doi.org/10.1371%2Fjournal.pone.0092409

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

[22] Q. Kuang, L. Zhao, A practical GPU based KNN algorithm, in: In Proceedings of the Second Symposium on International Computer Science and Computational Technology (ISCSCT 09), 2009.

[23] M. Lastra, J. Carabao, P. D. Gutiérrez, J. M. Benítez, F. Herrera, Fast fingerprint identification using GPUs, Information Sciences 301 (0) (2015) 195 – 214.

[24] L. Mussi, F. Daolio, S. Cagnoni, Evaluation of parallel particle swarm optimization algorithms within the CUDA^{TM} architecture, Information Sciences 181 (20) (2011) 4642 – 4657, special Issue on Interpretable Fuzzy Systems.

[25] R. Nock, M. Sebban, D. Bernard, A simple locally adaptive nearest neighbor rule with application to pollution forecasting, International Journal of Pattern Recognition and Artificial Intelligence 17 (8) (2003) 1369–1382.

[26] J. R. Prasad, U. Kulkarni, Gujrati character recognition using weighted k-nn and mean χ 2 distance measure, International Journal of Machine Learning and Cybernetics 6 (1) (2015) 69–82.

[27] Y. Pu, J. Peng, L. Huang, J. Chen, An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl, in: Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on, 2015, pp. 167–170.

[28] A. Rajaraman, J. Ullman, Mining of Massive Datasets, Cambridge University Press, 2011.

[29] M. Schatz, C. Trapnell, A. Delcher, A. Varshney, High-throughput sequence alignment using Graphics Processing Units, BMC Bioinformatics.

[30] G. Shakhnarovich, T. Darrell, P. Indyk, Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing), MIT Press, 2006.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

[31] I. Stamoulias, E. Manolakos, Parallel architectures for the kNN classifier - Design of soft IP cores and FPGA implementations, Transactions on Embedded Computing Systems 13 (2).

[32] N. Tomašev, M. Radovanović, D. Mladenić, M. Ivanović, Hubness-based fuzzy measures for high-dimensional k-nearest neighbor classification, International Journal of Machine Learning and Cybernetics 5 (3) (2014) 445–458.

[33] J. Wang, P. Neskovic, L. Cooper, Improving nearest neighbor rule with a simple adaptive distance measure, Pattern Recognition Letters 28 (2) (2007) 207–213.

[34] D. Wilson, T. Martinez, Improved heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1997) 1–34.

[35] X. Wu, V. Kumar, The top ten algorithms in data mining, CRC Press, 2010.

The published paper can be found here:
http://www.sciencedirect.com/science/article/pii/S0020025516306739
This draft is provided for copyright compliance

4 SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification

- P.D. Gutiérrez, M. Lastra, J.M. Benítez, F. Herrera. SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification. Submitted to Progress in Artificial Intelligence
  - Status: **Submitted**.

anuscript Click here to download Manuscript smote.tex

Progress in Artificial Intelligence manuscript No.
(will be inserted by the editor)

SMOTE-GPU: Big Data Preprocessing on Commodity Hardware for Imbalanced Classification

Pablo D. Gutiérrez[1] · Miguel Lastra[2] · José M. Benítez[1] · and Francisco Herrera[1]

Received: date / Accepted: date

Abstract Nowadays, it is usual to work with large amounts of data since our capacity of collecting and storing information has increased significantly. The extraction of knowledge from these scenarios is commonly known as Big Data. MapReduce platforms have been created to deal with Big Data problems and there are machine learning libraries available for these platforms. These libraries do not include tools to deal with imbalanced classification, like data sampling. Oversampling techniques only use a small part of the data, even in Big Data problems, that can be addressed on commodity hardware taking advantage of the parallel computation capabilities of Graphics Processing Units. SMOTE is one of the most popular oversampling methods which is based on the nearest neighbor rule. The proposed GPU-based SMOTE can efficiently handle large datasets (several millions of instances) on a wide variety of commodity hardware, including a laptop computer.

Keywords Imbalanced Classification · SMOTE · CUDA · Big Data

P.D. Gutiérrez
E-mail: pdgp@decsai.ugr.es
M. Lastra
E-mail: mlastral@ugr.es
J.M. Benítez
E-mail: J.M.Benitez@decsai.ugr.es
F. Herrera
E-mail: herrera@decsai.ugr.es
[1] Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada, Spain, 18071
[2] Department of Software Engineering of the University of Granada, CITIC-UGR, Granada, Spain, 18071

1 Introduction

Nowadays, it is usual to work with large amounts of data since our capacity of collecting and storing information has increased significantly. The extraction of knowledge from these scenarios is commonly known under the term ”Big Data” [17,27]. This term applies to situations that traditional Knowledge Discovery methods are unable to deal with. MapReduce [8] platforms, such as Apache Hadoop [25] or Apache Spark [26], were created to cope with the computational challenges that these new scenarios create. There are machine learning libraries [10,18,19,23] that have been created to work with these platforms on Big Data problems but there are still challenges that have not been solved, like imbalanced classification [9].

In traditional knowledge discovery, it is not unusual to find situations where the number of instances of each class of a problem is significantly different, this problem is usually known as imbalanced classification [13,15,20] and poses challenges to traditional learning algorithms. Considering a binary problem with a majority class and a minority class it is likely that a learning algorithm ignores the later and still achieves a high accuracy. There are three main ways of dealing with these situations [16]:

- **Algorithmic modification** Modifying learning algorithms in order to tackle the problem by design.
- **Cost-sensitive learning** Introducing costs for misclassification of the minority class at data or algorithmic level.
- **Data sampling** Preprocessing the data in order to reduce the breach between the number of instances of each class.

The algorithms that where used to sample the data on traditional scenarios could be used for Big Data, since the ideas behind them are not related to the number of instances of the problem, but traditional implementations cannot handle the large amounts of data required. These algorithms can be redesigned to use MapReduce platforms but these platforms require large clusters [8] that are usually expensive and shared among different users. Usually several configurations are tried in order to obtain the best performance [24].

An interesting alternative would be if the data sampling could be performed separated from the learning process and in commodity hardware using the MapReduce platforms only to get real classification results.

Graphics Processing Units (GPU) are available in almost every medium or high end commodity computer. These devices were created to compute 3D related computations in an efficient way but their parallel architecture makes them interesting for many other applications as varied as fingerprint identification [12], molecular simluation [22] or data mining [21]. Libraries like NVIDIA CUDA [1], have made these devices easier to use for general purpose applications, presenting the GPU device as a parallel co-processor. Modern GPU devices should be powerful enough to perform the computations required for data sampling algorithms in a reasonable time.

In this paper we explore the use of GPU devices combined with a proper data handling design in order to perform data sampling algorithms based on the well known SMOTE algorithm [7] in a reasonable time and on commodity hardware, called SMOTE-GPU. Our test covers datasets up to 10 millions of instances with imbalanced ratios (the relation between the number of instances of both classes) up to 49 and hardware configurations ranging from a server class GPU device to a medium range laptop.

The rest of the paper is organized as follows: Section 2 shows the characteristics of GPU devices and comments the main sampling solutions to deal with imbalanced data and its suitability to be adapted to commodity hardware; Section 3 describes our design; Section 4 presents the experiments and obtained results; Section 5 shows the conclusions and future work.

2 Background

In this section we describe the main characteristics of GPU devices (Section 2.1) and we discuss the suitability for commodity hardware of different data sampling algorithms (Section 2.2).

2.1 Graphics Processing Units

As commented before, GPU devices were created to offload the computations related to 3D related computations from the CPU device to a specific and efficient device. These devices have a parallel architecture, usually Single Instruction Multiple Data (SIMD), that allows them to perform the same operation on different data at once. This architecture is different to the architecture that we find in CPU devices, making GPU device programming quite different from traditional parallel and distributed programming. NVIDIA CUDA is one widely used library that allows general purpose programming of NVIDIA GPU devices by presenting them as parallel co-processors.

The functions that are run on GPU devices are called kernels. When programming a kernel, the kernel code includes the operations that a single thread is going to perform. When the kernel is called, the code that calls it specifies a set of threads that will run the kernel, called grid. The grid is divided in blocks of threads, each block has a three-dimensional block identifier and each thread within a block has another three-dimensional thread identifier. These identifiers are accessible in the kernel code and are used to access data from each thread and to make threads cooperate in computations. The threads that belong to the same block can cooperate and communicate using a programmable cache that works as shared memory. The size of this cache is limited, if a kernel requires a large amount of shared memory, the number of blocks running at once will be reduced, reducing also the performance obtained.

The grid is distributed on the GPU cores in a transparent way, but there are some lower level characteristics that have to be taken into account in order to obtain good performance. The GPU cores are organized in streaming processors (SMX) that run sets of 32 threads, called warps in a synchronous way. This means that a warp of threads is always running the same instruction and that if there is code divergence, for instance an if clause with different results within the warp, that section of code will be serialized. A block has to be run on the same SMX, this is because the shared memory is implemented at the SMX level. There are also limits to the maximum number of warps and blocks that an SMX can handle at once, if a block has a small number of threads the resources will not be fully used, but a large number of threads can also create this situation since the SMX only can handle complete blocks. In both situations the performance obtained is reduced.

A proper use of the GPU resources is usually a critical factor on the performance because of the way the

GPU optimizes its memory accesses. GPU devices have a large number of registers per SMX, this is because each thread is assigned the registers that is going to need to run the code before hand. Because of this, a GPU device can switch from one warp to another in an extremely fast way since all the required information is always in registers. When the main memory is accessed the GPU device switches to another warp and runs it, this way, the latencies of memory accesses are hidden with useful computation.

Since GPU devices are separated from CPU devices, they have their own memory. This means that the data used in GPU computations needs to be copied from the computer's main memory to the GPU device memory. This computations can be performed asynchronously at the same time other computations are performed on the GPU device. Also, the GPU memory cache system is optimized for coalescent accesses, subsequent threads are expected to access subsequent memory positions. If the memory is not accessed in this way it can also lead to lower performance.

A good GPU-based design has to tackle all these restrictions and dependencies, identify which parts of the algorithm are suitable for the GPU device and try to take advantage of the asynchronous memory transferences in order to obtain the best performance.

2.2 Imbalanced Classification

There are three main ways to deal with imbalanced classification: modifying the algorithm, introducing misclassification cost and data sampling. Data sampling is the only one that can be performed in a separate way from the classification algorithm, since the other two require direct or indirect modifications of the algorithm. When sampling data there are two obvious strategies to solve the imbalanced problem, under-sampling the majority class or oversampling the minority class. However, the memory requirements of each type of algorithm are quite different making only one type suitable to be run on commodity hardware.

Under-sampling methods, such as random under-sampling or instance selection methods, balance the number of instances of the classes by reducing the number of instances of the majority class. The instances selected can be chosen in a random way or using some type of expert knowledge over the majority class. This requires to manipulate most of the data in order to perform the selection. In a Big Data scenario, the resources required to handle the majority class are virtually the same required to handle the whole problem, for this reason this type of approach is not advised if we want to perform this data transformation on commodity hardware.

Oversampling methods, on the other hand, balance the number of instances of the classes by increasing the number of instances of the minority class. The new instances can be obtained by duplicating existing instances or using some type of interpolation between existing instances or copying existing instances. Since these algorithms only need to handle the information referred to the minority class it is likely that this can be performed on a single computer. The Synthetic Minority Oversampling Technique (SMOTE) [7] and the Random Over-Sampling technique (ROS) are some of the most common techniques of this type. The first one is an interpolation technique while the second one is based on duplicating instances.

The Random Over-Sampling technique creates a new instance by selecting one real instance randomly and duplicating it. This procedure is repeated until the number of instances of both classes has been balanced or a user specified parameter value is reached.

The SMOTE technique is based on the idea of neighborhood of the k-nearest neighbor(kNN) rule. When used in classification, the kNN rule sets the class of an instance as the majority class of the k closest instances of the training set. SMOTE considers that a instance can be interpolated between an instance and one of it neighbors within the class. The algorithm computes the neighborhood of each instance of the minority class, chooses one of its neighbors and randomly interpolates a new instance using the values of each attribute as limits. When the number of artificial instances is larger than the number of real instances present in the dataset, the algorithm ensures that every real instance is used to create an artificial one, at least, as many times as the number of artificial instances is larger than the number of real instances.

The main computational bottleneck of the SMOTE algorithm is the neighborhood computation. The distance between each pair of instances needs to be measured and all those distances needs to be compared in order to find the neighborhood. GPU devices have proven to be efficient in the computation of the kNN rule, reducing the time required for its computation in a significant way. The next Section shows how we can run these algorithms on commodity hardware combining a proper data handling design and a GPU device.

3 Design for preprocessing on commodity hardware: SMOTE-GPU

In this section we present how the memory requirements for SMOTE can be adjusted to fit in commodity hard-

ware and how the computations can be performed on GPU devices.

3.1 Memory requirements

The usual applications that perform over-sampling algorithms work in a pretty naive way, loading the complete dataset in memory, separating it in different classes, computing all the new instances and writing the whole result on the hard drive. This approach works perfectly for traditional problems but working on Big Data scenarios it is unlikely that the whole dataset fits in device memory.

As commented in Section 2.2, over-sampling methods only require to work with the data of the minority class. Our approach only keeps in memory that data. When reading the dataset, each instance is written to disk straight away after it is read, then, depending on its class, it is only kept in memory, if the class corresponds with the minority class.

In the same way, it is not necessary to create all the artificial instances before writing them to disk. A single extra instance can be reused to store the results of the interpolation if the results are stored on the hard drive just after that.

This way, the memory requirements for the application, in terms of dataset storage, are reduced to the size of the instances of the minority class plus one extra instance. An over-sampling method that does not perform much computation with the data, like ROS, can work with only these changes, but other methods, like SMOTE, would still require too much time to perform their computations.

3.2 SMOTE-GPU design

Applying the previously mentioned memory scheme, the SMOTE algorithm needs to compute the neighborhood of each instance to interpolate. As commented in Section 2.2, there are several GPU implementations for the kNN classification problem that can be adapted to this situation.

The computation of the kNN rule on GPU devices is split into two kernels, one kernel that builds a distance matrix between test and training sets and another kernel that searches for the k minimum distances obtained. Most GPU-based designs struggle when the distance matrix does not fit on GPU device memory. In [11], the different versions that can handle large amounts of data are studied and compared. The proposed GPU-SME-kNN obtains the best performance among them, being able to compute the kNN rule in datasets of more than 4.5 millions of instances, so it can be considered a good candidate for the SMOTE method.

GPU-SME-kNN [11] computes the distance matrix using a block based scheme. The size of each block can be defined by user-set parameters, so it can be adapted to a large variety of GPU devices. Each block of the matrix is computed in a kernel call, each row corresponds to a thread block but, the number of threads per block is fixed to a value d, smaller than the length of the matrix block. Each thread computes several distances in a coalescent way.

The selection of the neighborhood is computed sequentially after each block of the distance matrix is computed. For the first matrix block of a strip the neighborhood is computed, when the second block of the strip is ready the new neighborhood is computed combining the previous one and the matrix block, and the process continues until the las block of the matrix strip has been computed.

GPU-SME-kNN [1] uses a quicksort [14] based selection. This type of selection allows to discard all the elements of a block that do not improve the previous neighborhood in lineal time, using as pivot the furthest neighbor of the previous block.

Another particularity of GPU-SME-kNN is that it uses a separate kernel to compute the square root operation required for the distance computation. The neighborhood comparison can be performed obtaining the same results without performing that operation that it is only applied to the finally selected neighbors in order to obtain the real distance results.

Finally, one of the key aspects of GPU-SME-kNN is the use of asynchronous memory transfers. The data required for the distance computation corresponds with the instances attributes values, this data is not used during the selection process so the data required for the computation of the next matrix block can be loaded while the selection is computed. In the same way, the final neighborhood can be copied to CPU main memory while the computations of the next matrix block is performed. This way all the memory transfers between CPU and GPU devices are overlapped with computation, except the initial transfer for the first matrix block and the transfer of the last neighborhood.

The last step of the SMOTE technique is the interpolation of the instances. This part is performed on CPU for two reasons, the first one is to minimize the memory requirements since each instance needs to be write as soon as is computed; the second one is because it would require to store large portions of the dataset, if not all of it, on device memory. Furthermore, the data is not accessed on a coalescent way which would lead to

[1] `http://sci2s.ugr.es/GPU-SME-kNN`

Table 1 Datasets information

Dataset	Majority	Minority	Att.	IR
ECBDL14-05mill-90	470400	9600	90	49
ECBDL14-10mill-90	9408000	192000	90	49
SUSY_IR_16	2169740	135610	18	16
SUSY_IR_4	2169740	542435	18	4
SUSY_IR_8	2169740	271219	18	8
HIGGS_IR_16	4663300	291457	28	16
HIGGS_IR_4	4663300	1165825	28	4
HIGGS_IR_8	4663300	582913	28	8
HEPMASS_IR_16	4200100	262507	28	16
HEPMASS_IR_4	4200100	1050029	28	4
HEPMASS_IR_8	4200100	525013	28	8

bad performance, if this step were computed on GPU devices.

SMOTE-GPU can be adapted to a broad range of GPU devices and can work with large datasets with the only prerequisite that the fit on CPU main memory. The memory scheme proposed in Section 3.1 produces a scenario, where all the data required for the over-sampling process is stored in main memory, that combined with this GPU-based kNN computation makes possible the use of the SMOTE technique over large datasets in a single machine.

4 Experimental study

Different experiments have been carried out to check the results obtained by our design for SMOTE-GPU. The section is organized as follows: the experiments are described in Section 4.1; the different hardware configurations are detailed in Section 4.2; the obtained results are shown and discussed in Section 4.3.

4.1 Experimental framework

Two different types of experiments have been performed. The first type focusses on the time needed to apply the sampling technique to different datasets, while the second types studies the classification results obtained after applying the sampling method.

Four different datasets have been used: ECBDL14, HEPMASS, Higss and Susy. The first one from the ECBDL 14 competition [2], after a feature selection process to reduce it to 90 features [24], and the other three from the UCI repository [4,5]. The UCI datasets have been modified to create datasets with different imbalanced ratio. Table 1 shows the number of instances of each class, the number of attributes of the problem and the imbalance ratio for each dataset used. These datasets have been split following a 5-fold cross validation scheme so every result shown in this paper corresponds to the average of the results obtained on each fold.

SMOTE and ROS have been used as over-sampling method, with two different configurations. The first configuration balances the number of instances of the minority class while the second one introduces 50% of overhead for the minority class. The value of k for the SMOTE algorithm is set to 5, the rest of the parameters for the kNN algorithm have been the same specified in [11].

We also wanted to compare the time performance with the ones obtained by software available that performs data preprocessing. We tried to run the SMOTE implementation available on Keel [3] but we were not able to obtain results for a single experiment on these datasets after more than 8 hours of runtime on a server node.

To check the accuracy of the over-sampled datasets the decision tree from MLlib has been used. Since the datasets are large, the depth of the trees has been set to the maximum value that MLlib allows for this algorithm, which is 30. The area under the ROC curve (AUC) [6] has been used as measure to show the quality of the results since the classification accuracy of the algorithm is not useful when the data is imbalanced.

4.2 Hardware configurations

Different hardware configuration have been use for the sampling methods in order to prove their suitability on hardware of different characteristics. The first configuration corresponds to a cluster node that uses an NVIDIA Tesla k20 GPU with 5 GB of memory and 2496 CUDA cores, an Intel Xeon E5-2630 processor at 2.30 GHz and 128 GB of main memory. The second configuration is a desktop computer that uses an NVIDIA GeForce GTX 680 with 2GB of memory and 1532 CUDA cores, an Intel core i7 3820 processor at 3.60 GHz and 24 GB of main memory. The last configuration correspond with a laptop computer that uses an NVIDIA GeForce GTX 740m with 384 CUDA cores, an Intel Core i5 3337U processor at 1.8 GHz and 8 GB of main memory.

The accuracy MLlib experiments have been performed on a small spark cluster with four worker nodes, each worker has 8 threads and 28 GB of main memory.

Table 2 Time results, in seconds, for the SMOTE algorithm

	Server	Desktop	Laptop
ECBDL14-05mill-90	99.64	73.19	110.55
ECBDL14-10mill-90	2066.48	1553.82	2546.65
SUSY_IR_16	164.22	128.91	233.99
SUSY_IR_4	308.49	323.20	1149.29
SUSY_IR_8	196.51	169.55	420.34
HIGGS_IR_16	584.09	445.31	930.70
HIGGS_IR_4	1387.20	1508.78	6701.97
HIGGS_IR_8	730.63	661.74	2093.40
HEPMASS_IR_16	504.58	391.31	799.39
HEPMASS_IR_4	1174.35	1238.15	5477.82
HEPMASS_IR_8	637.18	553.78	1742.41

Table 3 Time results, in seconds, for the ROS algorithm

	Server	Desktop	Laptop
ECBDL14-05mill-90	55.90	40.25	58.43
ECBDL14-10mill-90	1101.16	777.76	1197.57
SUSY_IR_16	90.71	63.21	94.88
SUSY_IR_4	107.06	78.95	112.88
SUSY_IR_8	97.60	68.47	102.82
HIGGS_IR_16	281.42	204.17	305.73
HIGGS_IR_4	340.56	245.06	362.14
HIGGS_IR_8	301.10	215.81	324.14
HEPMASS_IR_16	254.42	180.47	269.97
HEPMASS_IR_4	299.76	214.69	317.97
HEPMASS_IR_8	268.01	190.02	283.71

Table 4 Time results, in seconds, for the SMOTE algorithm with extra 50% of minority class

	Server	Desktop	Laptop
ECBDL14-05mill-90	122.35	91.58	134.15
ECBDL14-10mill-90	2535.94	1858.10	3087.82
SUSY_IR_16	196.21	148.13	267.96
SUSY_IR_4	333.55	332.90	1183.28
SUSY_IR_8	226.92	185.30	447.18
HIGGS_IR_16	648.01	512.93	1034.11
HIGGS_IR_4	1440.68	1549.95	6782.25
HIGGS_IR_8	803.61	710.07	2186.71
HEPMASS_IR_16	582.31	448.76	910.48
HEPMASS_IR_4	1233.46	1292.08	5561.03
HEPMASS_IR_8	717.06	612.99	1840.47

4.3 Analysis of the results

Tables 2 to 5 show the average time results, in seconds, obtained on each cluster for each algorithm.

The time to perform the complete process is included in these results, considering also the time spend in reading the dataset and writing the results on the hard disk. As expected, the largest time is obtained by the laptop computer. However, even for more time demanding experiment, it is shorter than two hours and it is only around four times slower than the fastest time obtained for the same experiment.

Table 5 Time results, in seconds, for the ROS algorithm with extra 50% of minority class

	Server	Desktop	Laptop
ECBDL14-05mill-90	56.34	39.30	59.28
ECBDL14-10mill-90	1105.64	787.02	1181.07
SUSY_IR_16	90.67	64.75	97.82
SUSY_IR_4	106.89	76.13	112.46
SUSY_IR_8	96.73	69.72	101.96
HIGGS_IR_16	286.13	208.64	300.95
HIGGS_IR_4	332.43	241.33	356.74
HIGGS_IR_8	300.47	221.16	322.44
HEPMASS_IR_16	262.08	180.08	270.22
HEPMASS_IR_4	303.07	212.66	317.32
HEPMASS_IR_8	269.94	192.54	285.68

Table 6 Time results, in seconds, for the neighborhood computation in the SMOTE algorithm

	Server	Desktop	Laptop
ECBDL14-05mill-90	0.34	0.33	1.65
ECBDL14-10mill-90	57.75	64.85	390.17
SUSY_IR_16	10.63	13.50	64.81
SUSY_IR_4	149.77	201.35	979.60
SUSY_IR_8	38.81	51.16	247.95
HIGGS_IR_16	56.24	71.20	390.14
HIGGS_IR_4	862.24	1119.96	6149.25
HIGGS_IR_8	218.75	281.32	1543.06
HEPMASS_IR_16	47.39	59.44	327.08
HEPMASS_IR_4	701.65	911.01	4997.47
HEPMASS_IR_8	180.41	230.96	1269.28

The desktop configuration is faster than the server one in many cases. The reason for this is that the interpolation process is performed on CPU device and single thread performance of the CPU device on the desktop is higher than the CPU device on the server, as it was shown in the hardware description. This behavior could be expected for the ROS algorithm but it also happens in SMOTE.

The NVIDIA Tesla k20 GPU from the server delivers a higher performance than the NVIDIA GTX 680 from the desktop but the faster CPU of the desktop compensates this differences in most cases. The number of instances to create has more importance now, comparing the results for the dataset SUSY_IR_4 in Tables 2 and 4, corresponding to both SMOTE configurations. The server is faster that the desktop in the first case but slower in the second. The only difference between both settings is the number of artificial instances created. This means that, if the GPU device is powerful enough, the bottleneck moves from the neighborhood computation to the interpolation and data reading and storage.

Table 6 shows the time required for the kNN algorithm on each dataset and confirms what could be expected considering the different capabilities of the GPU

Table 7 Area under the ROC classifying with decision tree

	Original	SMOTE	ROS	SMOTE +50%	ROS +50%
ECBDL14-05mill-90	0.544881	0.567342	0.545054	**0.573031**	0.547924
ECBDL14-10mill-90	0.563563	0.594264	0.564074	**0.605921**	0.564196
SUSY_IR_16	0.665744	0.717567	0.679827	**0.723325**	0.665962
SUSY_IR_4	0.711592	0.734715	0.712165	**0.735505**	0.71156
SUSY_IR_8	0.691066	0.726985	0.697648	**0.729704**	0.691832
HIGGS_IR_16	0.570898	0.523949	0.597522	**0.619572**	0.571279
HIGGS_IR_4	0.62884	0.52789	0.635897	**0.637591**	0.62936
HIGGS_IR_8	0.598123	0.525692	0.61372	**0.625987**	0.598622
HEPMASS_IR_16	0.769566	0.808801	0.769752	**0.812846**	0.77002
HEPMASS_IR_4	0.808041	0.818099	0.807909	**0.818907**	0.808038
HEPMASS_IR_8	0.79149	0.81361	0.79107	**0.816336**	0.791037

devices used. It can be seen how the server configuration is faster than any other in this step. It also shows that the importance of this step, in terms of time, is much higher than in the other configurations, reaching up to 90% of the total time in some cases. However, the fact that a medium range three-years-old laptop can handle these datasets in less than two hours shows how powerful this design can be.

Considering that the time required to read and store the data has now a significant importance on the global performance of the algorithm it is important to state that in all the experiments the data was stored in a traditional magnetic hard drive. It is likely that using a Solid State Disk these times would be reduced, especially for the server and desktop configurations.

Table 7 shows the values for the Area under the ROC curve obtained using the MLlib decision tree with the original dataset and the sampled datasets.

These results show how the oversampling methods outperform the results obtained by the original dataset. SMOTE with an extra 50% of the minority class achieves the best results in all datasets. We can also observe how the results of the first configuration of SMOTE are better than ROS and than the original problem for all datasets except for HIGGS. The ROS algorithm seems to be leading to over-fitting when an extra 50% of the minority class is created, in that case, only four experiments improve the results obtained by the first configuration of ROS.

5 Conclusions

In this work, we have shown that it is possible to perform data oversampling for BigData datasets on commodity hardware by combining an efficient data handling scheme and the computational capacities of GPU devices. Different settings for the methods have been tested on different datasets up to 10 millions of instances and imbalanced ratios up to 51.

The Area under the ROC curve results show that the use of oversampling methods improves the detection of the minority class in Big Data datasets. We have also shown how our design can successfully work on a wide range of devices, including a laptop, while requiring reasonable times, around 25 minutes on high end devices and less than two hours on the laptop, for the most time-demanding experiment.

Acknowledgements This work was supported by the Spanish National Research Projects TIN2013-47210-P, TIN2014-57251-P and TIN2016-81113-R and by the Andalusian Regional Government Excellence Research Project P12-TIC-2958.

P.D. Gutiérrez holds an FPI scholarship from the Spanish Ministry of Economy and Competitiveness (BES-2012-060450).

References

1. CUDA. http://www.nvidia.com/object/cuda_home_new.html. [Online; accessed March 2017]
2. ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition (2014). URL http://cruncher.ncl.ac.uk/bdcomp/
3. Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V., Fernández, J., Herrera, F.: KEEL: A software tool to assess evolutionary algorithms for data mining problems. Soft Computing **13**(3), 307–318 (2009)
4. Bache, K., Lichman, M.: UCI machine learning repository (2013). URL http://archive.ics.uci.edu/ml
5. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature communications **5** (2014)
6. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recogn. **30**(7), 1145–1159 (1997). DOI 10.1016/S0031-3203(96)00142-2
7. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. **16**(1), 321–357 (2002)
8. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)

9. Fernández, A., del Río, S., Chawla, N.V., Herrera, F.: An insight into imbalanced big data classification: outcomes and challenges. Complex & Intelligent Systems pp. 1–16 (2017)
10. Foundation, A.S.: Apache Mahout (2017). URL `http://mahout.apache.org/`. [Online; accessed March 2017]
11. Gutiérrez, P.D., Lastra, M., Bacardit, J., Benítez, J.M., Herrera, F.: GPU-SME-kNN: Scalable and memory efficient kNN and lazy learning using GPUs. Information Sciences **373**, 165–182 (2016)
12. Gutiérrez, P.D., Lastra, M., Herrera, F., Benitez, J.M.: A high performance fingerprint matching system for large databases based on GPU. IEEE Transactions on Information Forensics and Security **9**(1), 62–71 (2014)
13. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering **21**(9), 1263–1284 (2009). DOI 10.1109/TKDE.2008.239
14. Hoare, C.A.R.: Algorithm 64: Quicksort. Commun. ACM **4**(7), 321 (1961). DOI 10.1145/366622.366644
15. Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence **5**(4), 221–232 (2016)
16. López, V., Fernández, A., García, S., Palade, V., Herrera, F.: An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Information Sciences **250**, 113 – 141 (2013). DOI http://dx.doi.org/10.1016/j.ins.2013.07.007
17. Madden, S.: From databases to big data. IEEE Internet Computing **16**(3), 4–6 (2012). DOI 10.1109/MIC.2012.50
18. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: Mllib: Machine learning in apache spark. Journal of Machine Learning Research **17**(34), 1–7 (2016)
19. Owen, S., Owen, S.: Mahout in action (2012)
20. Prati, R.C., Batista, G.E.A.P.A., Silva, D.F.: Class imbalance revisited: a new experimental setup to assess the performance of treatment methods. Knowledge and Information Systems **45**(1), 247–270 (2015)
21. Rajaraman, A., Ullman, J.: Mining of Massive Datasets. Cambridge University Press (2011)
22. Salomon-Ferrer, R., Gtz, A., Poole, D., Le Grand, S., Walker, R.: Routine microsecond molecular dynamics simulations with amber on gpus. 2. explicit solvent particle mesh ewald. Journal of Chemical Theory and Computation **9**(9), 3878–3888 (2013). DOI 10.1021/ct400314y
23. Spark, A.: Machine Learning Library (MLlib) for Spark (2017). URL `http://spark.apache.org/docs/latest/mllib-guide.html`. [Online; accessed March 2017]
24. Triguero, I., del Río, S., López, V., Bacardit, J., Benítez, J.M., Herrera, F.: Rosefw-rf: The winner algorithm for the ecbdl14 big data competition: An extremely imbalanced big data bioinformatics problem. Knowledge-Based Systems **87**, 69 – 79 (2015)
25. White, T.: Hadoop: The Definitive Guide. 4th edn. O'Reilly Media, Inc. (2015)
26. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 1–14. USENIX Association (2012)
27. Zikopoulos, P.C., Eaton, C., deRoos, D., Deutsch, T., Lapis, G.: Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. 1st edn. McGraw-Hill (2011)

Bibliography

[AFSG⁺09] Alcalá-Fdez J., Sánchez L., García S., del Jesus M., Ventura S., Garrell J., Otero J., Romero C., Bacardit J., Rivas V., Fernández J., and Herrera F. (2009) KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing* 13(3): 307–318.

[AMMR95] Anand R., Mehrotra K., Mohan C. K., and Ranka S. (Jan 1995) Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks* 6(1): 117–124.

[ARBM12] Arefin A. S., Riveros C., Berretta R., and Moscato P. (08 2012) GPU-FS-kNN: A Software Tool for Fast and Scalable kNN Computation Using GPUs. *PLoS ONE* 7(8): e44000.

[Bid85] Bidloo G. (1685) *Anatomia Humani Corporis*. Tot Amsterdam.

[BPM04] Batista G. E. A. P. A., Prati R. C., and Monard M. C. (Junio 2004) A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.* 6(1): 20–29.

[CBHK02] Chawla N. V., Bowyer K. W., Hall L. O., and Kegelmeyer W. P. (Junio 2002) Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.* 16(1): 321–357.

[CFM10] Cappelli R., Ferrara M., and Maltoni D. (2010) Minutia cylinder-code: A new representation and matching technique for fingerprint recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 32(12): 2128–2141.

[CH67] Cover T. M. and Hart P. E. (1967) Nearest Neighbor Pattern Classification. *IEEE Trans. Inf. Theory* 13(1): 21–27.

[CMM02] Cappelli R., Maio D., and Maltoni D. (2002) Synthetic fingerprint-database generation. In *Proc. 16th Int. Conf. Pattern Recognit.*, volumen 3, pp. 744–747.

[CUD] NVIDA CUDA.
http://www.nvidia.com/object/cuda_home_new.html. [Online; accessed March 2017].

[DG08] Dean J. and Ghemawat S. (jan 2008) {MapReduce}: simplified data processing on large clusters. *Commun. ACM* 51(1): 107–113.

[DHS12] Duda R. O., Hart P. E., and Stork D. G. (2012) *Pattern classification.* John Wiley & Sons.

[Dom99] Domingos P. (1999) Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pp. 155–164. ACM, New York, NY, USA.

[FdRL+14] Fernández A., del Río S., López V., Bawakid A., del Jesus M. J., Benitez J. M., and Herrera F. (2014) Big Data with Cloud Computing: an insight on the computing environment, {MapReduce}, and programming frameworks. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 4(5): 380–409.

[FEV+09] Friedrichs M., Eastman P., Vaidyanathan V., Houston M., Legrand S., Beberg A., Ensign D., Bruns C., and Pande V. (2009) Accelerating molecular dynamic simulation on graphics processing units. *Journal of Computational Chemistry* .

[FHJ51] Fix E. and Hodges Jr J. L. (1951) Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, DTIC Document.

[GCH08] García S., Cano J. R., and Herrera F. (2008) A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition* 41(8): 2693 – 2709.

[GDNB10] Garcia V., Debreuve E., Nielsen F., and Barlaud M. (2010) K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *Proceedings - International Conference on Image Processing, ICIP*, pp. 3757–3760.

[GDP+15a] Galar M., Derrac J., Peralta D., Triguero I., Paternain D., Lopez-Molina C., García S., Benitez J. M., Pagola M., Barrenechea E., Bustince H., and Herrera F. (2015) A survey of fingerprint classification Part II: Experimental analysis and ensemble proposal. *Knowledge-Based Syst.* 81: 98–116.

[GDP+15b] Galar M., Derrac J., Peralta D., Triguero I., Paternain D., Lopez-Molina C., García S., Benitez J. M., Pagola M., Barrenechea E., Bustince H., and Herrera F. (2015) A survey of fingerprint classification Part I: Taxonomies on feature extraction methods and learning models. *Knowledge-Based Syst.* 81: 76–97.

[GFB+11] Galar M., Fernández A., Barrenechea E., Bustince H., and Herrera F. (2011) An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition* 44(8): 1761 – 1776.

[Gre84] Grew N. (1684) The description and use of the pores in the skin of the hands and feet. *Philos. Trans. R. Soc. London* 14: 566–567.

[GW07] Gao Q.-B. and Wang Z.-Z. (2007) Center-based nearest neighbor classifier. *Pattern Recognition* 40(1): 346–349.

[Hen00] Henry E. (1900) *Classification and Uses of Finger Prints.* George Routledge and Sons, Broadway, Ludgate Hill, United Kingdom.

[HG09] He H. and Garcia E. A. (Sept 2009) Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering* 21(9): 1263–1284.

[iaf] Integrated Automated Fingerprint Identification System.

[JC08] Jiang R. M. and Crookes D. (2008) Fpga-based minutia matching for biometric fingerprint image database retrieval. *Journal of Real-Time Image Processing* 3(3): 177–182.

[JFR07] Jain A. K., Flynn P., and Ross A. A. (2007) *Handbook of biometrics.* Springer.

[JHB97] Jain A. K., Hong L., and Bolle R. (1997) On-line fingerprint verification. *IEEE Trans. Pattern Anal. Mach. Intell.* 19(4): 302–314.

[JHPB97] Jain A. K., Hong L., Pankanti S., and Bolle R. (1997) An identity-authentication system using fingerprints. *Proc. IEEE* 85(9): 1365–1388.

[JPC99] Jain A. K., Prabhakar S., and Chen S. (1999) Combining multiple matchers for a high security fingerprint verification system. *Pattern Recognit. Lett.* 20: 1371–1379.

[JY00] Jiang X. and Yau W. Y. (2000) Fingerprint minutiae matching based on the local and global structures. In *Proc. 15th Int. Conf. Pattern Recognit.*, volumen 2, pp. 1038–1041. IEEE.

[KDD14] Komarov I., Dashti A., and D'Souza R. M. (05 2014) Fast k-NNG Construction with GPU-Based Quick Multi-Select. *PLoS ONE* 9(5): e92409.

[KPD90] Knerr S., Personnaz L., and Dreyfus G. (1990) *Single-layer learning revisited: a stepwise procedure for building and training a neural network*, pp. 41–50. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kra16] Krawczyk B. (2016) Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence* 5(4): 221–232.

[KSH+12] Krizhevsky A., Sulskever I., Hinton G. E. G. E., Sutskever I., and Hinton G. E. G. E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* pp. 1097–1105.

[LBH15] LeCun Y., Bengio Y., and Hinton G. (2015) Deep learning. *Nature* 521(7553): 436–444.

[LFG+13] López V., Fernández A., García S., Palade V., and Herrera F. (2013) An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences* 250: 113 – 141.

[LGH12] Luengo J., García S., and Herrera F. (2012) On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and Information Systems* 32(1): 77–108.

[LMUn+09] Lastra M., Mantas J. M., Ureña C., Castro M. J., and García-Rodríguez J. A. (Noviembre 2009) Simulation of shallow-water systems using graphics processing units. *Math. Comput. Simul.* 80(3): 598–618.

[Mad12] Madden S. (May 2012) From databases to big data. *IEEE Internet Computing* 16(3): 4–6.

[MFV02] Mollineda R. A., Ferri F. J., and Vidal E. (Oct 2002) A merge-based condensing strategy for multiple prototype classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 32(5): 662–668.

[MMJP09] Maltoni D., Maio D., Jain A. K., and Prabhakar S. (2009) *Handbook of fingerprint recognition.* Springer, New York.

[NM06] Nanni L. and Maio D. (2006) Combination of different fingerprint systems: A case study FVC2004. *Sens. Rev.* 26(1): 51–57.

[NSB03] Nock R., Sebban M., and Bernard D. (2003) A simple locally adaptive nearest neighbor rule with application to pollution forecasting. *International Journal of Pattern Recognition and Artificial Intelligence* 17(8): 1369–1382.

[PBS15] Prati R. C., Batista G. E. A. P. A., and Silva D. F. (2015) Class imbalance revisited: a new experimental setup to assess the performance of treatment methods. *Knowledge and Information Systems* 45(1): 247–270.

[PGT+15] Peralta D., Galar M., Triguero I., Paternain D., García S., Barrenechea E., Benitez J. M., Bustince H., and Herrera F. (2015) A survey on fingerprint minutiae-based local matching for verification and identification: Taxonomy and experimental evaluation. *Inf. Sci.* 315: 67–87.

[PS85] Preparata F. P. and Shamos M. I. (1985) *Computational Geometry: An Introduction*. Springer.

[PTG+16] Peralta D., Triguero I., García S., Herrera F., and Benitez J. M. (2016) DPD-DFF: A Dual Phase Distributed Scheme with Double Fingerprint Fusion for Fast and Accurate Identification in Large Databases. *Inf. Fusion* 32: 40–51.

[PTSR+14] Peralta D., Triguero I., Sanchez-Reillo R., Herrera F., and Benitez J. M. (feb 2014) Fast Fingerprint Identification for Large Databases. *Pattern Recognit.* 47(2): 588–602.

[RNJ06] Ross A. A., Nandakumar K., and Jain A. K. (2006) *Handbook of multibiometrics*, volumen 6. Springer.

[STDV07] Schatz M., Trapnell C., Delcher A., and Varshney A. (2007) High-throughput sequence alignment using Graphics Processing Units. *BMC Bioinformatics* .

[TdRL+15] Triguero I., del Río S., López V., Bacardit J., Benítez J. M., and Herrera F. (2015) Rosefw-rf: The winner algorithm for the ecbdl'14 big data competition: An extremely imbalanced big data bioinformatics problem. *Knowledge-Based Systems* 87: 69 – 79.

[uid] Unique Authentication Authority of India.

[Wat93] Watson C. I. (1993) NIST Special Database 14. Technical report, NIST.

[Whi15] White T. (2015) *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition.

[WK10] Wu X. and Kumar V. (2010) *The top ten algorithms in data mining*. CRC Press.

[WNC07] Wang J., Neskovic P., and Cooper L. (2007) Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters* 28(2): 207–213.

[WW92] Watson C. I. and Wilson C. L. (1992) NIST Special Database 4. Technical report, NIST.

[ZCD+12] Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M. J., Shenker S., and Stoica I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 1–14. USENIX Association.

[ZE01] Zadrozny B. and Elkan C. (2001) Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 204–213. ACM, New York, NY, USA.

[ZEd+11] Zikopoulos P. C., Eaton C., deRoos D., Deutsch T., and Lapis G. (2011) *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill, 1st edition.

[ZLA03] Zadrozny B., Langford J., and Abe N. (Nov 2003) Cost-sensitive learning by cost-proportionate example weighting. In *Third IEEE International Conference on Data Mining*, pp. 435–442.