Information Sciences 306 (2015) 1–20

Contents lists available at ScienceDirect

Information Sciences

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

Large-scale fingerprint identification on GPU

Raffaele Cappelli *, Matteo Ferrara, Davide Maltoni

Department of Computer Science and Engineering of the University of Bologna, 47521 Cesena, FC, Italy

ARTICLE INFO

Article history:
Received 30 June 2014
Received in revised form 15 January 2015
Accepted 8 February 2015
Available online 12 February 2015

Keywords:
Fingerprint identification
Minutiae
GPU
CUDA
SIMD
MCC

ABSTRACT

This paper proposes a new parallel algorithm to speed up fingerprint identification using GPUs. A careful design of the algorithm and data structures, guided by well-defined optimization goals, yields a speed-up of 1946× over a baseline sequential CPU implementation and of 207× over a CPU implementation optimized with SIMD instructions. The proposed algorithm enables a medium-scale AFIS (Automated Fingerprint Identification System) to run on a simple PC with four Tesla C2075 GPUs. On a benchmark with 250 000 fingerprints and 100 000 queries, the proposed system yields state-of-the-art biometric accuracy with a throughput of more than 35 million fingerprint matches per second. The proposed approach can be easily scaled-up, thus making possible the implementation of a large-scale AFIS (i.e., with a database of hundred million fingerprints) on inexpensive hardware.

1. Introduction

Biometric systems are widely used both in forensic and civil applications to automatically recognize the identity of a person on the basis of physiological or behavioral characteristics [24]. Examples of biometric applications span from physical access control to forensic identification, from border crossing to voters authentication. Although many biometric traits (e.g., fingerprints, face, iris, voice, etc.) have been thoroughly studied, fingerprints, due to their peculiarities (i.e., individuality, persistence, cost and maturity of the products), remains the most used one [37].

1.1. Fingerprint matching and identification

A *fingerprint* is the representation of the epidermis of a finger: it consists of a pattern of interleaved ridges and valleys (Fig. 1a). Discontinuities in the ridges (e.g., terminations or bifurcations) are called *minutiae* (Fig. 1a): nowadays, most state-of-the-art fingerprint recognition algorithms are based on minutiae matching [11]. Each minutia can be described by some attributes, including its location in the fingerprint, its direction, its type (termination or bifurcation), and a value representing the quality of the fingerprint pattern in its neighborhood. Minutia-based matching algorithms usually consider each minutia as a triplet $m = \{x, y, \theta\}$ encoding the spatial coordinates and the ridge angle. The set of minutiae attributes extracted from a fingerprint is called *template*; fingerprint matching consists in comparing two templates to determine whether the two sets of minutiae come from the same finger. Two minutiae are considered matching if the spatial distance between them is smaller than a given spatial threshold and their directional difference is smaller than a given angular threshold; such thresholds are necessary to compensate for the unavoidable errors made by minutiae extraction algorithms and to account for small elastic distortions due to skin elasticity. In practice, minutiae matching is an "extended" point

* Corresponding author.
E-mail addresses: raffaele.cappelli@unibo.it (R. Cappelli), matteo.ferrara@unibo.it (M. Ferrara), davide.maltoni@unibo.it (D. Maltoni).

http://dx.doi.org/10.1016/j.ins.2015.02.016
0020-0255/

pattern matching problem. Unfortunately, neither the alignment parameters nor the point-correspondence function are known a priori, hence solving the matching problem is hard. A brute force approach, that is evaluating all the possible solutions, is exponential in the number of minutiae, therefore suboptimal heuristics are typically used in real applications. The first automatic algorithms, developed in the early 50s, were inspired by the manual techniques of forensic experts and were aimed at determining the global (rigid) alignment leading to an optimal spatial (and directional) minutiae pairing: Hough transform was a common solution [47]. In the last decades, with the progress in the field of computational intelligence, many other techniques were proposed to improve fingerprint matching [25], including machine-learning methods [33], evolutionary algorithms [49,50], and fuzzy similarity measures [16]. More recently, researchers have focused on local matching algorithms, based on fixed-length features that characterize the neighborhood of each minutia [4,10,53,40]; these features are usually represented as bit-vectors, allowing efficient similarity measures to be implemented on many hardware architectures and simplifying the design of template protection methods [53,19].

Although state-of-the-art approaches are nowadays very accurate and able to tolerate common perturbations (translation, rotation, deformation, missing or spurious minutiae, etc.) [37], the computational-demanding nature of minutiae-based matching still makes the development of large-scale fingerprint identification systems challenging in terms of efficiency. In fact, the time required to search a query fingerprint on a database grows with the size of the database itself: when the database contains millions of fingerprints, very expensive hardware platforms are required to operate at high throughput. This is the case of the AFIS owned by police agencies such as FBI, or the huge civil identification systems being deployed in emerging countries (e.g., UIDAI project [51]).

There are basically two possibilities to increase the speed of fingerprint identification:

- reducing the total number of fingerprint comparisons (through fingerprint classification [13,14], pre-filtering or multi-stage matching [5,6,9]);
- reducing the processing time (e.g., by designing matching algorithms with low computational complexity [37], or using parallel architectures [47,26,30,22,2,46]).

1.2. General-purpose computing on Graphics Processing Units

A *Graphic Processing Unit* (GPU) is a highly-parallel processor for computer graphics. In response to commercial demand for real-time graphics rendering, GPUs have evolved into many-core processors designed to perform data-parallel computation. The main difference between GPUs and Central Processing Units (CPUs) is that GPUs have proportionally more transistors devoted to arithmetic logic units and less to caches and flow control in comparison to CPUs; GPUs also have higher memory bandwidth compared to CPUs. Recently, the use of GPUs for general-purpose parallel computing is increasingly attracting researchers' interests: General-Purpose computing on Graphics Processing Units (GPGPU) [32] is emerging as a compelling way to deal with computationally demanding tasks, where a large amount of data needs to be processed and/or a large number of operations has to be carried out (e.g., [20,39,54]). The parallel processing capability of the GPU allows complex computing tasks to be divided into thousands of smaller tasks that can run concurrently. A typical hardware configuration for GPGPU consists of a CPU (called *host*) connected to one or more GPUs (called *devices*). Given a computation X to be done with GPGPU, X is split into several parts, some of which can be executed in parallel. In order to take effective advantage of the GPU, it is necessary to analyze which parts of X can be executed in parallel on the many processing units of the GPU and write a CPU program (called *host program*) that sends input data and GPU instructions (called *kernel programs*) to the GPU. The GPU executes the given computation in parallel and returns the result to the CPU. To this purpose, specific tools are needed to schedule execution of kernels and communicate with the GPU.

1.3. Contribution of this work

The recent advances in GPU hardware (with a notable growth in computational power and memory capacity) and their successful adoption in many different applications, suggest that GPUs may drastically improve the efficiency of fingerprint identification; this is particularly true for modern local minutiae-matching algorithms, which are well suited for parallel implementations. However, we quickly realized that a simple porting of existing fingerprint recognition algorithms to GPU hardware does not offer relevant advantages. This is also confirmed by the first studies published on this topic by other researchers [22,2] that just reported minor speed improvements. In fact, the design of an effective GPU fingerprint identification approach requires to address the following issues: (i) limiting data transfer; (ii) using compact data representation (possibly bit-based), (iii) optimizing memory allocation and access; (iv) defining a computation flow that fully exploits the hardware capabilities.

This paper introduces a new parallel algorithm specifically-designed for fingerprint identification on GPUs. The proposed algorithm is based on Minutia Cylinder-Code (MCC) [10,12], recently introduced as a convenient way to represent fingerprint minutiae. The neighborhood of each minutia is encoded into a fixed-length local structure (called *cylinder*), which can be easily compared to the cylinders obtained from other minutiae neighborhoods. To perform a fingerprint identification, the cylinders of the query fingerprint have to be compared against the cylinders of all database fingerprints. The solution proposed in this paper, thanks to a careful design of the algorithm, ad-hoc data structures and look-up tables, special sorting methods, and many other optimizations, achieves remarkable results. Systematic experiments show that the proposed