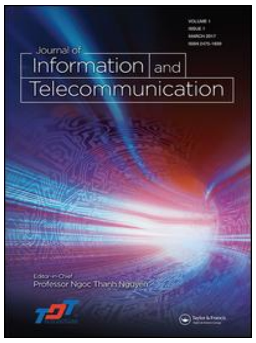

Journal of Information and Telecommunication

ISSN: 2475-1839 (Print) 2475-1847 (Online) Journal homepage: http://www.tandfonline.com/loi/tjit20

Speeding up and enhancing a large-scale fingerprint identification system on GPU

Hong Hai Le, Ngoc Hoa Nguyen & Tri-Thanh Nguyen

JOURNAL OF INFORMATION AND TELECOMMUNICATION, 2017
https://doi.org/10.1080/24751839.2017.1404712

Speeding up and enhancing a large-scale fingerprint identification system on GPU*

Hong Hai Le, Ngoc Hoa Nguyen and Tri-Thanh Nguyen

University of Engineering and Technology (UET), Vietnam National University, Hanoi (VNU), Hanoi, Vietnam

ABSTRACT

Fingerprint identification is one of the most common biometric feature problems which is used in many applications. Although state-of-the-art algorithms are very accurate, the need for fast processing a big database of millions of fingerprints is highly demanding. In this paper, we propose to adapt the fingerprint matching algorithm based on Minutia Cylinder-Code on Graphics Processing Units to speed up the matching. Another contribution of this paper is to add a consolidation stage after matching to enhance the precision. The experimental results on a GTX-680 and K40 tesla devices with standard data-sets prove that the proposed algorithm can be comparable with the state-of-the-art approach, and is suitable for a real-time identification application.

ARTICLE HISTORY
Received 5 May 2017
Accepted 10 November 2017

KEYWORDS
Fingerprint identification; matching; minutiae; Minutia Cylinder-Code; GPU; parallel processing

1. Introduction

Fingerprint matching is the task of estimating the degree of similarity between two given fingerprint images (hereafter called fingerprint, for short). Current solutions to this problem can be classified into three types: correlation-based, minutiae-based, and ridge feature-based matching, of which the minutiae-based matching is the most popular approach thanks to its accuracy as stated in Fingerprint Verification Competitions (FVC) (Cappelli, Maio, Maltoni, Wayman, & Jain, 2006). A minutia is the point where a ridge continuity breaks, and it is typically represented as a triplet (x, y, θ), where x and y are the coordinates of the point and θ is the ridge direction at that point. The similarity of the two given fingerprints with minutiae-based matching approach is proportional to the number of matched minutiae pairs in them. Figure 1 illustrates the matched minutiae of two fingerprints.

The state-of-the-art matching algorithm Minutia Cylinder-Code (MCC) (Cappelli, Ferrara, & Maltoni, 2010) has the best accuracy; however, its matching speed is not good enough to apply in a real application which has a large database of millions of fingerprints, e.g. the database of civil identification systems (Unique Identification Authority of India, 2012).

CONTACT Tri-Thanh Nguyen ntthanh@vnu.edu.vn University of Engineering and Technology (UET), Vietnam National University, Hanoi (VNU), 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

*This paper was improved from the paper 'Exploiting GPU for Large Scale Fingerprint Identification' in the proceedings of ACIIDS 2016, and 'A complete fingerprint matching algorithm on GPU for a large scale identification system' in the proceedings of Information Science and Applications (ICISA) 2016.

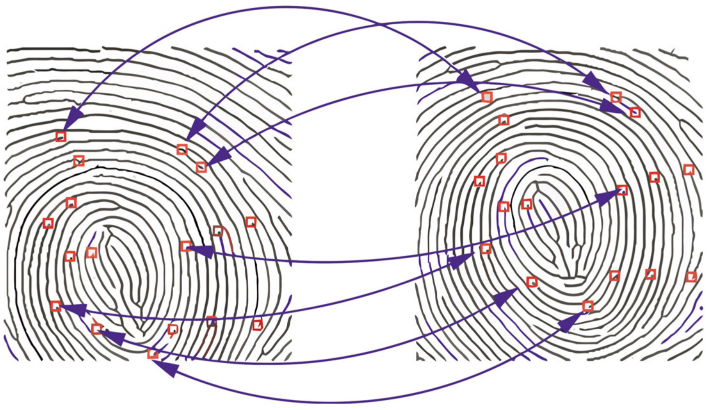

Figure 1. The matched minutia pairs between two fingerprints.

There are two popular methods to increase the speed of fingerprint identification:

- Reducing the number of fingerprints for comparisons by filtering, i.e. fingerprint classification (Cappelli & Maio, 2004; Hong, Min, Cho, & Cho, 2008), pre-filtering using indexing techniques (Cappelli, Ferrara, and Maltoni (2011; Bhanu & Tan, 2001). A fingerprint is only compared again the fingerprints of the same class.
- Parallelizing the matching algorithm (Cappelli, Ferrara, & Maltoni, 2015; Gutierrez, Lastra, Herrera, & Benitez, 2014; Peralta, Triguero, Sanchez-Reillo, Herrera, & Benitez, 2014).

Graphic cards with *general purpose graphics processing units* (GPGPUs, hereafter called GPU for short) are a new parallel architecture which have been proven to be very useful for accelerating the processing speed of computationally intensive algorithms. These devices include thousands of processing units; thus, they provide massive parallel calculations and have been applied successfully in many fields such as artificial intelligence (Krizhevsky, Sutskever, & Hinton, 2012; Zhang, Yi, Wei, & Zhuang, 2014), simulation (Friedrichs et al., 2009), bioinformatics (Schatz, Trapnell, Delcher, & Varshney, 2007), as well as fingerprint matching (Cappelli et al., 2015; Gutierrez et al., 2014). In this paper, we propose a novel approach to adapt/parallelize the MCC algorithm to GPU devices.

In addition, two minutia pairs between the two given fingerprints can be locally matched; however, it is not guaranteed to be globally matched. It is incorrect to use the minutiae, which are only locally matched, to estimate the similarity degree of the two given fingerprints. Another contribution of this paper is to parallelize the consolidation step to the MCC algorithm to ensure that the minutiae are globally matched before estimating the global similarity. The experiments on the standard fingerprint database indicate that our proposal helps to decrease the error rate.

The contribution of this paper includes:

- Propose a novel parallelization method of MCC algorithm (cf. Section 4.1)
- Parallelize the consolidation stage to enhance the precision (cf. Section 4.2)

This paper was improved from Le, Nguyen, and Nguyen (2016a, 2016b), in which the parallelization strategies are clearly discussed in detail; further experiments with a new GPU device were carried out to confirm the correctness of the proposed solution.

The rest of the paper is organized as follows. Section 2 reviews the MCC algorithm. Section 3 briefly describes the GPU architecture and programming model. Section 4 describes the state-of-the-art approach. Section 5 describes our adapted MCC algorithm on GPU devices. Section 6 analyses the experimental results on the open databases. The final section gives the conclusions and future directions.

2. Introduction to MCC-based matching algorithm

2.1. Fingerprint identification problem

From an existing database of N fingerprints (maybe accompanied with personal information having fingerprints in the database), given a new (unknown) fingerprint, the task of fingerprint identification is to find out the matched ones in the database, i.e. find the *match score* or *similarity* between the new fingerprint with everyone in the database. Two fingerprints are deemed to be matched if their *match score* (or *similarity*) is greater or equal a predefined threshold.

2.2. Minutia Cylinder-Code

Given a minutia m represented as $\{x_m, y_m, \theta_m\}$, its local structure in MCC is defined as a cylinder having the centre at the coordinate $\{x_m, y_m\}$, the radius R, and the height 2π. Each cylinder is divided into $N_s \times N_s \times N_d$ cells as shown in

Figure 2, where N_s is the resolution of the discretized 2D space around minutia m ($N_s \times N_s$) and N_d is the height of the cylinder (i.e. 2π). From this cylinder, it is possible to find its neighbouring minutiae of which the information used to form the characteristics of this minutia is the position (Chikkerur, Cartwright, & Govindaraju, 2006; Medina-Pérez, García-Borroto, Gutierrez-Rodriguez, & Altamirano-Robles, 2011), ridge (Feng, Ouyang, & Cai, 2006; Wang, Li, & Niu, 2007), orientation (Qi, Yang, & Wang, 2005; Tico & Kuosmanen, 2003), or a combination of these (Feng, 2008).

The local structures are invariant to the fingerprint rotation and translation. Therefore, it allows to match two fingerprints with different rotation and translation. The local structure matching allows to quickly find pairs of minutiae that can be matched locally and can be the candidate for aligning between the two fingerprints.

The contribution of each minutia m_t to a cell (of the cylinder corresponding to a given minutia m_i) depends both on spatial information (how much m_t is close to the centre of the cell) and directional information (how much the directional difference between m_t and m_i, is similar to the directional difference associated to the section where the cell lies). In other words, the value of a cell represents the likelihood of finding minutiae that are close to the cell and have directional difference with respect to m_i.

With a negligible loss of accuracy, the cylinder c_i of the minutia m_i can be simply treated as a single feature vector, and each element of the feature vector can be represented as one bit (Cappelli et al., 2010).

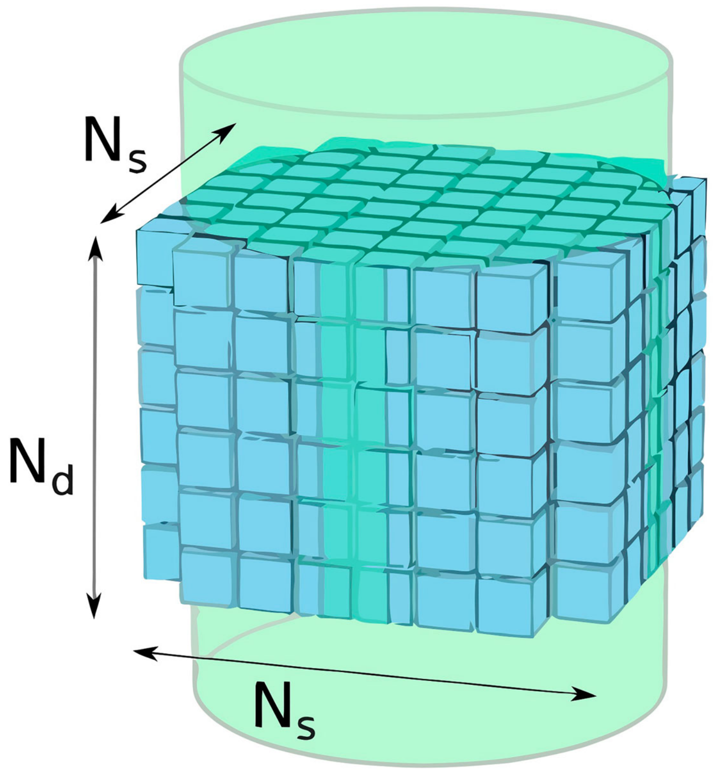

Figure 2. The local structure of a minutia represented in MCC (Gutierrez et al., 2014).

In order to compare two given fingerprints, most minutiae-based matching algorithms consist of two steps: the first step performs a local structure matching followed by a global step, i.e. calculate the aggregated similarity based on the locally matched minutiae. Recently, an MCC-based matching algorithm (Cappelli et al., 2010) shows a good performance in terms of both accuracy and speed.

2.3. Similarity score calculation

Let v_i and v_j be the (cylinder) bit vectors of minutiae m_i and m_j, correspondingly, a simple but effective similarity *sim*(.) between the two minutiae is defined as (Cappelli et al., 2010):

$$sim(m_i, m_j) = \begin{cases} 1 - \dfrac{||v_i \oplus v_j||}{||v_i|| + ||v_j||} \text{if } d(m_i, m_j) \leq t_\theta, \\ 0 \qquad \text{otherwise,} \end{cases} \tag{1}$$

where

- $\oplus$ represents the bitwise XOR operator,
- $||.||$ represents the Euclidean norm,
- $d(m_i, m_j)$ is the angle difference (i.e. θ_i, θ_j) of the two minutiae m_i and m_j. The formula is $d(m_i, m_j) = d_\varphi(\theta_i, \theta_j) = \min(|\theta_i - \theta_j|, 360 - |\theta_i - \theta_j|)$.
- t_θ is the acceptable threshold.

Given two fingerprints T_1 and T_Q, a local matching is performed on every pair of cylinders, and the results are stored in a matrix. In order to aggregate the similarity score s of the two fingerprints, there are two strategies:

- Local Similarity Sort (LSS) technique sorts the scores of the matrix and computes the average of the top n_P highest scores.
- A more accurate but less efficient similarity measure is the Local Similarity Sort with Distortion-Tolerant Relaxation (LSS-DTR). LSS-DTR adds a consolidation stage to LSS in order to obtain a score that modifies the local similarities to hold at the global level. Figure 3 shows the steps to calculate similarity score s using LSS, where s is the average of the top n_P values chosen from the similarity matrix.

2.4. MCC-based matching algorithm

From a database T of N fingerprints in the format of minutia templates, i.e. $T = \{T_1, \ldots, T_N\}$, given an unknown query template T_q, the first step (local matching) MCC matching algorithm is to calculate similarity matrix between each minutia pairs of T_i and T_q. In the second step, i.e. global score estimation, the similarity score S_i is calculated from the average of top n_P values of the similarity matrices (LSS approach). In step 3, $S = \{S_1, S_2, .., S_N\}$ is used to determine which templates in the database are the matches of the query. Figure 4 demonstrates the calculating steps of the MCC algorithm. The

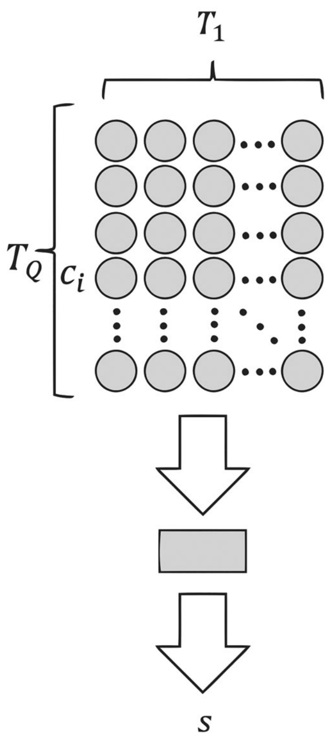

Figure 3. Similarity score computing process using LSS (Cappelli et al., 2015).

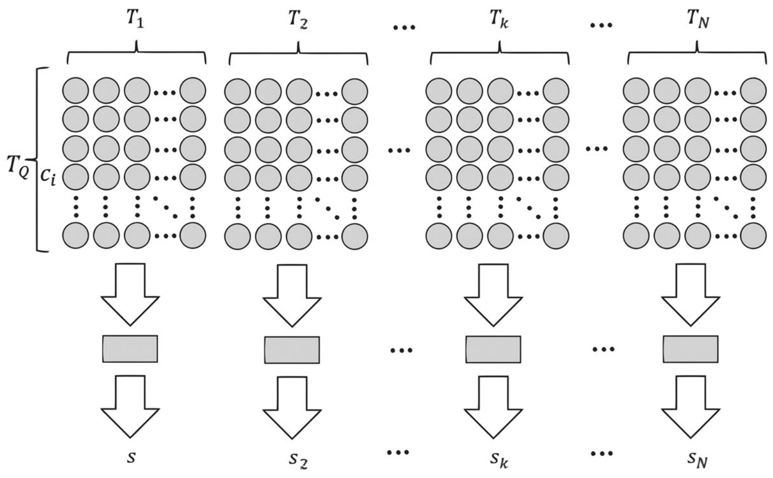

Figure 4. Fingerprint identification processing steps based on MCC (Cappelli et al., 2015).

most time-consuming step is Step 1, i.e. estimating the similarity scores; thus, this is where parallelization is applied. The pseudo-code of the algorithm is described in Algorithm 1, where $S[]$ is the matched score array and $maxValue[]$ is the maximum matched score of each minutia in T_q with other minutiae in the other fingerprint T_i. Since only top n_P values of the similarity matrix are used to calculate the global score, the array $maxValue[]$ is used instead of a matrix.

ALGORITHM 1 The sequential MCC-based fingerprint matching algorithm with CPU.

Sequential fingerprint identification on CPU

Input: The template set T
The query template T_q
Output: The possibly matched templates from T

```
1.   Let $S[]$ be an array of size $N$; //Similarity score set
2.   Load the template set $T = \{T_1, \ldots, T_N\}$ into the main memory;//Done only once
3.   Let $T_q$ be the query template;
4.   For $i = 1$ to $N$//$N$ is the number of templates in the database
     //Step 1: local minutia matching
5.     Let $maxValue[]$ be an array of size $T_q$.length;//The similarity of matched pairs
6.     For $j = 1$ to $T_i$.length // $T_i$.length is the number of minutiae of $T_i$
7.       Let $m[j]$ be the $j^{th}$ minutia of the query fingerprint $T_i$;
8.        $maxValue[j] = 0$;
9.       For $k = 1$ to $T_q$.length // $T_q$.length is the number of minutiae of $T_q$
10.        Let $m[k]$ be the $k^{th}$ minutia of the template $T_q$;
11.        If $d(m[j], m[k]) \leq t_\theta$
12.            $tempScore = sim(m[j], m[k])$;
13.           $maxValue[j] = \max(maxValue[j], tempScore)$;
14.        End If
15.      End For
16.    End For
     //Step 2: Similarity estimation
17.    $S[i] = \sum_{t=1}^{T_q.length} (maxValue[t])/T_q.length$
18.  End For
     //Step 3: Find and return the match from the score set S[]
```

Note that, the matched score of two fingerprints is calculated based on the array *maxValue*[], i.e. local matching scores. Step 3 is not mentioned in detail, since it is not the point for parallelization.

3. The parallelism of GPU

NVIDIA introduces the *compute unified device architecture* (CUDA), one of the two common frameworks for developing applications on their GPU devices (the other framework is OpenCL[1]). With CUDA, applications written in C/C++ running on the CPU (of the hosted machine) can call to execute parallel codes called *kernels* (written in C) on NVIDIA GPUs. The physical architecture of CUDA-enabled GPUs consists of a set of *Streaming Multiprocessors* (SM), each containing 32 cores following *single instruction, multiple data* (SIMD) scheme. The architecture of NVIDIA GPU device is depicted in Figure 5.

Threads are the instances of a kernel (i.e. run the same code), and each processes a different dataset (i.e. SIMD). Threads are grouped into blocks. All threads of the same

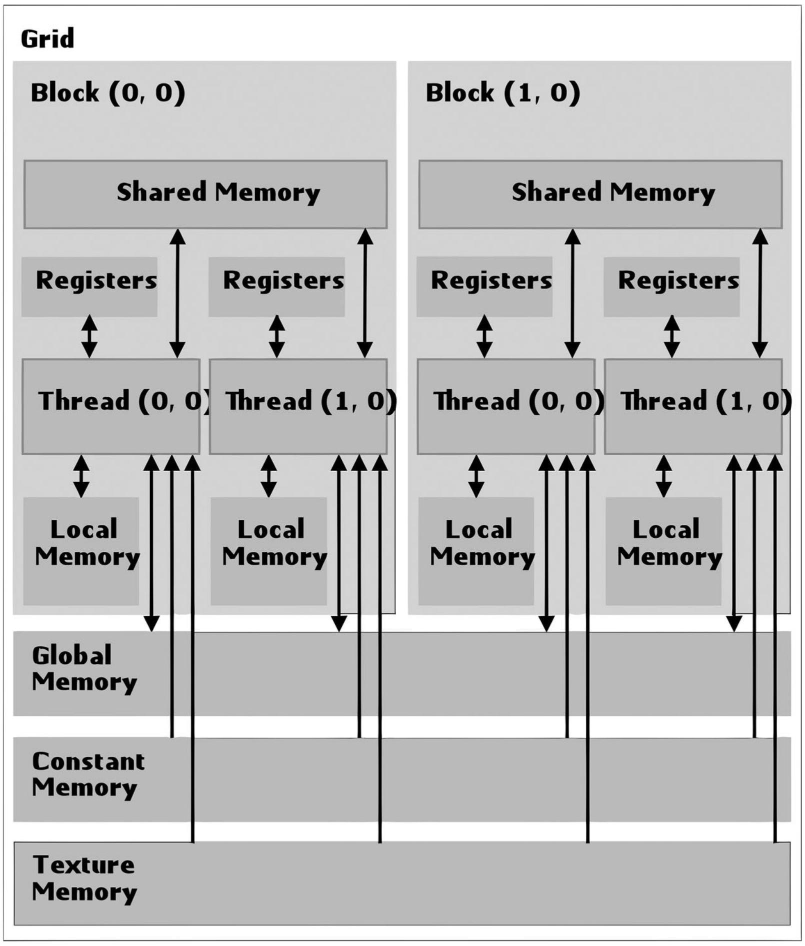

Figure 5. NVIDIA GPU architecture (Luebke et al., 2006).

block are executed on the same SM and share the limited memory resources of that multiprocessor. The maximum number of threads in a block cannot be too large (1024 for the GPU device used in this work). However, a kernel can be executed by multiple, equally sized blocks, forming a grid: the total number of threads is then equal to the number of blocks times the number of threads per block. Each SM schedules and executes threads in groups of maximum 32 parallel threads (i.e. the maximum number of cores in an SM) called *warps*. A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp synchronize their execution path. If threads of the same warp take different paths (due to flow control instructions), they *have to wait for each other*. It is important to make GPU threads extremely lightweight.

CUDA threads have access to various memory types (Figure 4): each thread has its registers, which are the fastest memory, and its private local memory (which is slower); each block has a small shared memory, accessible to all threads of the block and with the same lifetime of the block; all threads have access to the global memory: the largest memory and slowest memory type, which is accessible by all threads of all blocks, so it is used for communication between different blocks and with the host (the program running on CPU). When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions. Therefore, a very important optimization in CUDA is ensuring that global memory accesses are as much coalesced as possible.

A CUDA program includes two parts: one is the kernel running in parallel on the GPU device following SIMD paradigm; the other running on CPU acts as a coordinator, i.e. prepares the data, sends the data to the GPU device, calls the kernels, and retrieves the results from the GPU.

4. Cappelli's parallelizing approach

The pseudo-code of Cappelli's parallelizing approach is listed in Algorithm 2, which includes three GPU kernels:

- The first kernel ComputeLMC is responsible for building the lookup table for identifying whether two minutiae m_i, m_j (in two fingerprints) have the angle difference less than a threshold, i.e. $d(m_i, m_j) \leq t_\theta$ (cf. 1). This strategy helps to avoid if branch in the SIMD model, thus improving the performance.
- The second kernel Step-1 computes the similarity of minutiae pairs, i.e. a minutia of the query T_q and another one of a T_i. To follow the SIMD model, each thread in a block is responsible for calculating the similarity between a minutia of a T_i with all minutiae of T_q. There are DB_1 threads in a block and $\left\lceil \frac{NC}{DB_1} \right\rceil$ blocks in this kernel. In the experiments, DB_1 is set to 192.
- The third kernel Step-2 computes the similarity between T_q and all T_i in T. Each thread calculates the similarity of (T_q, T_i) pair. To make all the threads have an equal task, the author considered only w top (highest) scores of matched cylinders of (T_q, T_i). The

bucket $B \in \mathbb{N}^{N \times w}$ is used to store these highest scores. There are DB_2 threads in a block and $\left\lceil \frac{N}{DB_2} \right\rceil$ blocks in this kernel. In the experiments, DB_2 is set to 64.

ALGORITHM 2 Cappelli's parallel algorithm on GPU

```
Parallel fingerprint identification on GPU
1.  Load the template set $T = \{T_1, \ldots, T_N\}$ into GPU;//Done only once
2.  Reset bucket matrix $B \in \mathbb{N}^{N \times w}$; //Set matrix cell's value to 0
3.  Load the query template $T_q$;
4.  Initialize $S$ in GPU
5.  Let $NC = \sum_i Cylinder(T_1)$ ;
6.  Call ComputeLMC kernel ($z$ threads per block, 1 block);
7.  Call Step-1 kernel ($DB_1$ threads per block, $\left\lceil \frac{NC}{DB_1} \right\rceil$ block);
8.  Call Step-2 kernel ($DB_2$ threads per block, $\left\lceil \frac{N}{DB_2} \right\rceil$ block);
9.  Copy $S$ from GPU memory to RAM for finding the possible matches
```

5. The proposed approach

5.1. Adapting MCC matching algorithm to GPU

5.1.1. Strategy 1: maximize the active threads

The primary goal in writing a kernel to run on a GPU is to maximize the active 32 threads in a *warp*. Since the number of minutiae in a fingerprint is variable, to avoid divergence between threads in the warps, several approaches suggest to mainly divide the MCC matching algorithm into two separate kernel GPU calls (Cappelli et al., 2015; Gutierrez et al., 2014). The first kernel GPU call is to calculate all similarity matrices. The second call is to calculate the global scores from similarity matrices. When dividing the algorithm into separating calls, it is necessary to transfer data between kernel calls. Meanwhile, some advantages of the GPU architecture like shared memory are not utilized. Cappelli et al. (2015) use a very careful design to adapt the algorithm and an ad hoc technique to translate the similarity matrix to have a fixed size.

From our investigation on the FVC 2002 fingerprint databases (Cappelli et al., 2006), the average number of minutiae of each fingerprint is 30, and the average number of matches for a genuine matching is only 6. Thus, the first optimization strategy, *we propose to use 32 minutiae* for each fingerprint which is enough for the matching process in order to fit well on a GPU block. However, we carefully select 32 minutiae (from fingerprints having more than 32 minutiae) as follows:

- All existing minutiae are used to generate cylinders, so the MCC bit vectors of cylinders are not affected in comparison with the case of using all minutiae.
- From the fact that, the more bit 1 the vector (of a minutia) has, the more neighbouring minutiae it has. Minutiae having the cylinder with a small number of 1 tend to be the outlines of a fingerprint. We sort the cylinder vectors in the decreasing order of number of bit 1 (in the vector), and then select the top 32 minutiae.

Since some fingerprints have more than 32 minutiae, we have to accept an increase in error rate. However, this is acceptable as detailed in the next section.

In our approach, all the similarity matrices in Figure 5 have the same size 32×32. We set the number of threads per block to be 32, i.e. equal the maximum number of threads per warp (or per SM). Then we use one GPU block to calculate the similarity (i.e. using LSS) between a template T_k and the query T_q. This is the key difference between our approach and Cappelli's. In Step 1, each thread of the block is responsible for calculating a column in the similarity matrix and finding the maximum value in that column. In Step 2, the first thread of the block calculates the global similarity from the average of maximum values found from 32 threads of the block.

5.1.2. Strategy 2: exploit the shared memory

Since the shared memory space is faster than global memory, we exploit it to store the results in each block. Table 1 lists the data structures and variables along with its memory type.

5.1.3. Strategy 3: pre-compute costly functions

Since a minutia is represented by a triple $(x, \ y, \ \theta)$, where $(x, \ y)$ is its coordinate in integers and the ridge θ is represented in degree, it is possible to pre-compute cosine and sine functions with the range of $0 \leq \theta \leq 360$. Though these functions accept only radian values, a simple mapping from degree to radian is used before calling the functions. Square root function is needed in some calculations (cf. Section 4.2) and we found out the range of the input (i.e. from 0 to 1281); thus, it is possible to calculate the square root of these values.

Consequently, time-consuming functions (e.g. manipulate real numbers) like square root, cosine, and sine can be pre-computed and stored in an array. Later, we just lookup the result instead of calling the function. These details are not mentioned clearly in Algorithm 3 for simplicity.

5.1.4. Strategy 4: void branching instructions

The branching statement like *if* in the kernel will make the threads to take different branches; thus, the SIMD model does not work well. The statement 'If $d(m_j, m_k) \leq t_\theta \ldots$' in line 11 in Algorithm 1 will be removed as follows: we pre-compute $d(m_j, m_k)$ and store the results in a list of arrays, i.e. for each θ (in degree instead of radian) $(0 \leq \theta \leq 360)$, find all the minutiae m_k of the query template T_q satisfying $d(m_j, m_k)$. Thus, given a new minutia m_j with the angle θ, the list of minutiae of T_q satisfying the constraint is readily available. Lines 10–13 in Algorithm 2 demonstrate this method.

Table 1. Data structures.

Name	Type	Note
T	Array[N]	The template set $T = \{T_1, T_2, .., T_N\}$ (global memory)
b_{idx}	Int	Block index, each block process a template $T[b_{idx}]$
S	Array[N]	Store the matching score of the templates (global memory)
t_{idx}	Int	The thread index within each block
maxMatching	Array[32]	Store the maximum number of globally matched minutia pairs between two templates (shared memory)
maxValue	Array[32]	Store the maximum matched scores for each minutia (shared memory)
maxIndex	Array[32]	Store the index of the matched minutiae (shared memory)
gridDim.x	Int	The system's variable that stores the grid size

5.2. Enhancing the precision with consolidation stage

The simplest consolidation approach uses the local matched minutiae pair having the maximum similarity value in order to align the fingerprint T and T_q for the consolidation stage.

Let (m_{i1}, m_{i2}) and (m'_{j1}, m'_{j2}) be two minutia pairs to be verified. Let $[\Delta x, \Delta y]$ be the translation between two minutiae m'_{j1} and m_{i1}. Let θ be the angle to rotate m'_{j1} to m_{i1}. Let m''_{j2} be the mapped point from m'_{j2} using translation $[\Delta x, \Delta y]$, and rotation θ. Let $(x''_{j2}, y''_{j2}, \theta''_{j2})$ and $(x_{i2}, y_{i2}, \theta_{i2})$ be the triplets of m''_{j2} and m_{i2}. Then x''_{j2} and y''_{j2} are calculated as:

$$\begin{bmatrix} x''_{j2} \\ y''_{j2} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x'_{j2} \\ y'_{j2} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}.$$

The two minutia pairs are deemed to be globally matched if they satisfy following constraints:

- The spatial distance between the two minutiae does not exceed a threshold t_s, i.e. $d(m''_{j2}, m_{i2}) = \sqrt{(x''_{j2} - x_{i2})^2 + (y''_{j2} - y_{i2})^2} \le t_s$, where $d(m''_{j2}, m_{i2})$ is the spatial distance between m''_{j2} and m_{i2}.
- The directional difference between the two minutiae does not exceed a threshold t_θ, i.e. $\theta(m''_{j2}, m_{i2}) = \min(|(\theta''_{j2} + \theta) - \theta_{i2}|, \ 360 - |(\theta''_{j2} + \theta) - \theta_{i2}| \le t_\theta)$.

Figure 6 demonstrates the step of translation and rotation to compare the two minutia pairs. The two parameters t_s and t_θ represent the tolerance window, and their value can be determined by experiments. For example, in TK algorithm (Tico & Kuosmanen, 2003), the distance threshold $t_s = 12$ and the angle threshold $t_\theta = \pi/6$ brought in a good performance.

The transformation on the minutiae pair having maximum similarity value may not be the best transformation at the global level. Several authors have adopted multiple candidate transformations for the alignments. Finally, the transformation that maximizes the number of consolidation stage minutiae pairs will be chosen. For instance, Medina-Pérez et al. (2011) reduced the number of local matching minutiae pairs by for each minutia p and minutia q, selecting only minutia that maximizes their similarity values, then perform the transformation for each minutiae pair in the reduced set. Feng (2008) sorted minutiae pairs by descending similarity values and chose top n minutia pairs for

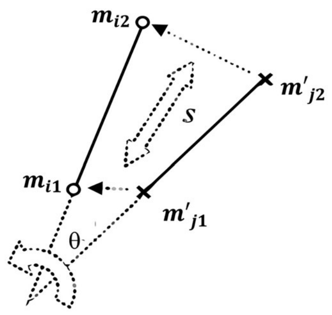

Figure 6. Global checking of the match between to minutia pairs (Maltoni, Maio, Jain, & Prabhakar, 2009).

the transformation. Normally, these approaches allow to get better accuracy than that of the single transformation approach. In our approach, we compare all local matched pairs of minutiae between two fingerprints to calculate the global similarity.

5.3. The proposed algorithm

The data structures and variables used in our algorithm are listed in Table 1.

The pseudo-code of our algorithm is described in Algorithm 2.

ALGORITHM 3 Our proposed parallelized algorithm

Parallel fingerprint identification on GPU

```
     /***CPU code***/
1.   Initialize $S[]$ be an array of size $N$ on GPU; //Similarity score set
2.   Load the template set $T = \{T_1, \ldots, T_N\}$ into GPU; //Do once only
3.   Load $T_q$ into GPU; //Do for every query
4.   Enumerate the minutiae of each $T_i \in T$ satisfied $d(m_q, m_k) \leq t_\theta$, where $m_q$ and $m_k$ are minutiae of $T_q$ and $T_i$,
       correspondingly.
     /***Kernel code (running on GPU), call for each query $T_q$***/
     **//Step 1: local minutia matching**
5.   While $b_{idx} < N$
6.     //Each thread $t_{idx}$ processes a minutia of $T[b_{idx}]$, thus, no for loop is needed
7.       Let $m[t_{idx}]$ be the $t_{idx}^{th}$ minutia of the query fingerprint $T[b_{idx}]$;
8.       For $k = 1$ to number of minutiae of $T_q$ satisfied $d(m_{t_{idx}}, m_k) \leq t_\theta$
9.           $tempScore = sim(m_{t_{idx}}, m_k)$;
10.          $maxValue[t_{idx}] = \max(maxValue[t_{idx}], tempScore)$ ;
11.          If $tempScore == maxValue[t_{idx}]$
12.            $maxIndex[t_{idx}] = k$;
13.          End if
14.      End For
15.    __syncthreads();
     **//Step 2.1: Consolidation stage**
16.    $maxMatching[t_{idx}] = 0$;
17.    For $i = 1$ to 32
18.      If $i <> t_{idx}$ and $((m[t_{idx}], m[i])$ pair of $T[b_{idx}]$ matches $(m[maxIndex[t_{idx}]], m[maxIndex[i]])$ pair of $T_q)$ //c.f. Section
       4.2
19.        $maxMatching[t_{idx}]$++;
20.      End if
21.    End for
22.    __syncthreads();
     **//Step 2.2: Similarity estimation**
23.  atomicMax($maxMatches$, $maxMatching[t_{idx}]$);
24.  If $(t_{idx} == 0)$ //This is the first thread in a block, calculate the global score
25.      $S[b_{idx}] = maxMatches/32$
26.  End if
27.  $b_{idx}$+= $gridDim.x$; //Jump to next chunk of template set $T$
28.  End while
     /***CPU code***/
29.  Copy $S$ from GPU to the main memory;
30.  **//Step 3: Find the matches from the score set $S[]$**
```

The __syncthreads() function in lines 15 and 23 are the barriers for all the threads of the block, i.e. all the threads in the block must finish running the code above the call to __syncthreads(). This function ensures the results of all the threads of the block are available for aggregation.

The function atomicMax($maxMatches$, $maxMatching[t_{idx}]$) performs $maxMatches = \max(maxMatches, maxMatching[t_{idx}])$ while ensuring the critical section due to the race condition caused by multiple concurrent threads within the same block. Note that the matched score of the two fingerprints is calculated based on the variable *maxMatches*, i.e. the consolidation results. Thus, it is different from that in Algorithm 1.

5.4. The complexity of the proposed algorithm

From the above text, some discussions about our algorithm are:

- No modification to the original MCC matching algorithm is added, thus the complexity of our algorithm is intact.
- The reduction the number of minutiae of each fingerprint to 32 to fit the GPU architecture reduces the search space, thus the matching will be quicker for fingerprints having more than 32 minutiae in comparison with the original algorithm.
- The main reason to speed up the matching is the parallelization method as mentioned in Sections 4.1–4.3.

6. Experimental results

In this paper, the problem of processing fingerprints with different size is out of scope. In order to evaluate our proposed approaches, we used the FVC 2002 fingerprint database, which contains equal-sized fingerprints, for the experiments. The tool by Medina-Pérez, Loyola-González, Gutierrez-Rodríguez, García-Borroto, and Altamirano-Robles (2014) was used for minutiae extraction and MCC cylinder creation process. Some parameter values in Section 4.2 are the same as those of Tico and Kuosmanen (2003), i.e. $t_s = 12$ and the angle threshold $t_\theta = \pi/6$. The generated MCC templates of fingerprints were stored on the disk for the experiments. The experiments were carried out on CentOS 7.1 operating system with CUDA 7.5. The configuration of GPU devices is listed in Table 2, where Testla C2075 is the device used in Cappelli et al. (2015).

For evaluating the accuracy of the proposed algorithm, the result of the proposed algorithm is compared with the result of the MCC baseline algorithm in which all minutiae are used for the matching process. Table 3 shows the error rate of different algorithms, where WCS and WOCS stand for 'with consolidation stage', 'without consolidation stage', correspondingly. Without the consolidation stage, the action of selecting 32

Table 2. NVIDIA CPU configuration.

Feature	GTX 680	Tesla K40	Tesla C2075
CUDA cores	1536	2888	448
Base clock (MHz)	1006	745	1150
Standard memory	2GB	12GB	6GB

Table 3. Experimental results on DB1 of FVC 2002 (the smaller error rate, the better performance).

Algorithm	EER	FMR 100	FMR 1000	FMR Zero
MCC baseline	1.64%	2.10%	3.89%	4.85%
Our algorithm WOCS	1.76%	2.29%	4.32%	5.46%
Our algorithm WCS	**1.34%**	**1.68%**	**3.23%**	**4.03%**

Table 4. Average results of the 10 queries without consolidation stage.

Template set size	GTX 680 WOCS (KMPS)	GTX 680 WCS (KMPS)	Tesla K40 WOCS (KMPS)	Tesla K40 WCS (KMPS)
10,000	7142	1960	8575	2849
50,000	8196	2008	9260	3048
100,000	8403	2020	9615	3076
150,000	8474	2024	9740	3086
200,000	8510	2040	9803	3091

minutiae for matching increases the *equal error rate* (EER), otherwise it decreases the EER to be lower than the original algorithm.

For evaluating the speed of the proposed algorithm, we carried out all the experiments on an NVIDIA GeForce GTX 680 with 1536 CUDA cores, Kepler Architecture, and 2GB of memory. The FVC 2002 database was scaled to different sizes (ranging from 10,000 to 200,000) to study how the GPU-based algorithm scaled with the database size. Ten fingerprints were randomly selected to be identified. Table 4 shows the results of experiments with different database sizes, where KMPS stands for 'kilo (thousand) match per second'.

At larger template set sizes, the throughput of the proposed algorithm is stable at 8.5 and 9.8 million matches per second on GTX 680 and Tesla, no scalability issues were found providing that the fingerprint templates can be stored on the GPU's memory. Our results are higher than those reported in Gutierrez et al. (2014), which gains 55.7 KMPS on the same GeForce GTX 680 device. In addition, these results are comparable to state-of-the-art result (Cappelli et al., 2015), which gains 9 million matches per second on Tesla C2075 GPU.

7. Conclusions and future work

This paper proposed a novel method to adapt MCC-based fingerprint matching to GPU. After using all minutiae for calculating cylinders, top 32 minutiae having the biggest number of neighbours are selected (to fit a GPU block) for matching. A consolidation stage step was added to the algorithm to enhance the accuracy. The speed of the proposed algorithm is comparable with the results of the state-of-the-art published algorithm. The proposed approach can be easily scaled-up to be able running in a real-time system. Thus, it is possible to implement a large-scale fingerprint identification system on inexpensive hardware.

One possible direction in the future is to apply a mixed parallel model with multiple machines with a *message passing interface* model, each of which has multiple GPU cards to process large fingerprint data-sets for real situations.

Note

1. See https://developer.nvidia.com/opencl

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Hong-Hai Le received a Bachelor degree in Information Technology at Vietnam National University, Hanoi (VNU) in 2003; Master of Science degree at the Asian Institute of Technology (AIT), Thailand in

2006. Since 2007, he has worked as a lecturer for Faculty of Information Technology, VNU - University of Engineering and Technology. He is currently a PhD student in Information Technology.

Ngoc-Hoá Nguyen received the degree of Engineer in Information Technology at Hanoi University of Science and Technology in 1999; Master of Computer Science degree at Informatic Francophone Institute (IFI) in 2001; and PhD degree (in Informatics) at Joseph Fourier University (France) in 2005. Since November 2005, he has worked as a lecturer at the Faculty of Information Technology, VNU University of Engineering and Technology, Vietnam. He was granted the Associate Professor title in Information Technology in 2015.

Tri-Thanh Nguyen received a Bachelor degree in Information Technology at Vietnam National University, Hanoi (VNU) in 1999; Master of Science degree at the Asian Institute of Technology (AIT), Thailand in 2002; and PhD degree (in Information Science) at Japan Advanced Institute of Science and Technology (JAIST). Since February 2003, he has worked as a lecturer at the Faculty of Information Technology, College of Technology (later changed to VNU - University of Engineering and Technology). He was granted the Associate Professor title in Information Technology in January 2015.

References

Bhanu, B., & Tan, X. (2001). A triplet based approach for indexing of fingerprint database for identification. *Proc. Int. Conf. on Audio- and Video-Based Biometric Person Authentication (3rd)*, pp. 205–210.

Cappelli, R., Ferrara, M., & Maltoni, D. (2010). Minutia cylinder-code: A new representation and matching technique for fingerprint recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*, 2128–2141.

Cappelli, R., Ferrara, M., & Maltoni, D. (2011). Fingerprint indexing based on minutia cylinder-code. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(5), 1051–1057.

Cappelli, R., Ferrara, M., & Maltoni, D. (2015). Large-scale fingerprint identification on GPU. *Information Sciences*, *306*, 1–20.

Cappelli, R., & Maio, D. (2004). State-of-the-art in fingerprint classification. In N. Ratha & R. Bolle (Eds.), *Automatic fingerprint recognition systems* (pp. 183–205). New York: Springer.

Cappelli, R., Maio, D., Maltoni, D., Wayman, J. L., & Jain, A. K. (2006). Performance evaluation of fingerprint verification systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*, 3–18.

Chikkerur, S., Cartwright, A. N., & Govindaraju, V. (2006). *K-plet and cbfs: A graph based fingerprint representation and matching algorithm*. International Conference on Biometrics, pp. 309–315.

Feng, J. (2008). Combining minutiae descriptors for fingerprint matching. *Pattern Recognition*, *41*, 342–352.

Feng, J., Ouyang, Z., & Cai, A. (2006). Fingerprint matching using ridges. *Pattern Recognition*, *39*, 2131–2140.

Friedrichs, M. S., Eastman, P., Vaidyanathan, V., Houston, M., Legrand, S., Beberg, A. L., … Pande, V. S. (2009). Accelerating molecular dynamic simulation on graphics processing units. *Journal of Computational Chemistry*. *30*(6), 864–872.

Gutierrez, P. D., Lastra, M., Herrera, F., & Benitez, J. M. (2014). A high performance fingerprint matching system for large databases based on GPU. *IEEE Transactions on Information Forensics and Security*, *9*(1), 62–71.

Hong, J. H., Min, J. K., Cho, U. K., & Cho, S. B. (2008). Fingerprint classification using one-vs-all support vector machines dynamically ordered with naı¨ve Bayes classifiers. *Pattern Recognition*, *41*(2), 662–671.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Image net classification with deep convolutional neural networks. NIPS 2012, pp. 1106–1114.

Le, H. H., Nguyen, N. H., & Nguyen, T. T. (2016a). *A complete fingerprint matching algorithm on GPU for a large scale identification system*. The 7th International Conference on Information Science and Applications, LNEE, Springer, pp. 679–688.

Le, H. H., Nguyen, N. H., & Nguyen, T. T. (2016b). *Exploiting GPU for large scale fingerprint identification*. The 8th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2016), LNCS, Springer, pp. 688–697.

Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., … Buck, I. (2006). GPGPU: General-purpose computation on graphics hardware. *SC '06 Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL.

Maltoni, D., Maio, D., Jain, A. K., & Prabhakar, S. (2009). *Handbook of fingerprint recognition*. Springer.

Medina-Pérez, M. A., García-Borroto, M., Gutierrez-Rodriguez, A. E., & Altamirano-Robles, L. (2011). *Robust fingerprint verification using m-triplets*. International Conference on Hand-Based Biometrics (ICHB 2011), Hong Kong, pp. 1–5.

Medina-Pérez, M. A., Loyola-González, O., Gutierrez-Rodríguez, A. E., García-Borroto, M., & Altamirano-Robles, L. (2014). Introducing an experimental framework in C # for fingerprint recognition. *Lecture Notes in Computer Science*, *8495*. doi:10.1007/978-3-319-07491-7_14

Peralta, D., Triguero, I., Sanchez-Reillo, R., Herrera, F., & Benitez, J. M. (2014). Fast fingerprint identification for large databases. *Pattern Recognition*, *47*(2), 588–602.

Qi, J., Yang, S., & Wang, Y. (2005). Fingerprint matching combining the global orientation field with minutia. *Pattern Recognition Letters*, *26*, 2424–2430.

Schatz, M. C., Trapnell, C., Delcher, A., & Varshney, A. (2007). High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, *8*, 474. doi:10.1186/1471-2105-8-474

Tico, M., & Kuosmanen, P. (2003). Fingerprint matching using an orientation-based minutia descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*, 1009–1014.

Unique Identification Authority of India, Role of Biometric Technology in Aadhaar Enrollment. (2012).

Wang, X., Li, J., & Niu, Y. (2007). Fingerprint matching using orientation codes and polylines. *Pattern Recognition*, *40*, 3164–3177.

Zhang, Y., Yi, D., Wei, B., & Zhuang, Y. (2014). A GPU-accelerated non-negative sparse latent semantic analysis algorithm for social tagging data. *Information Sciences*, *281*, 687–702.