On Fingerprint Processing Involving Modern Massively Parallel Architectures

Stanislav Sartasov, SPSU, St. Petersburg, Russia.

# Introduction

Fingerprint recognition is still the most widely accepted mean of person identification and authentication. Two recent trends can be observed in automated fingerprint identification systems (AFISs): a shift towards mobile security systems and a querying of a very large fingerprint databases (of order 106-107). Consequently two separate challenges arose in the field of fingerprinting: how to robustly identify a person with limited computational resources and how to speed up the processing of a very large database? The scalability problem of the biometric database is a long known issue. With the number of records increasing so does the response time of the biometric system, and for a subset of AFISs, the real-time AFISs, time is one of the quality characteristics.

In a generic AFIS there are following processing steps:

1. Image acquisition on a scanner
2. Image transmission from a scanner to the processing unit
3. Image enhancement (IE), can be omitted
4. Feature extraction and template building
5. Matching the template against database or part of it
6. Decision making whether to grant or deny access given the matching results

In order to improve the performance of the system in general one should improve the performance of every step, which is usually obtained through the introduction of parallelism. A note should be taken that steps 1 and 2 are usually hard to speed up due to hardware implementation of this process, yet their addition to a total processing time tends to be relatively small. As most systems implement step 6 in a form of a rule-based module with a single rule, the parallel implementation in this case might also be redundant.

With the hyperthreading and multi-core processors becoming available in the mid-2000s the simple way to improve the system performance was implemented in a commercially available systems. The template database is chunked equally by the number of physical or logical cores and each chunk is processed independently, not unlike map-reduce approach. For a large-scale applications computing clusters were designed implementing the same approach. With the development of virtualization technologies and private clouds management of such clusters is simplified, however buying a new device and operational costs (e.g. electricity) are high.

General Purpose Graphical Processing Unit (GPGPU) computations is now a proven way to enhance a system performance in HPC. Among the many technologies available to the programmer (e.g. OpenCL, DirectCompute) NVIDIA CUDA technology is the most advanced and readily available.

In this paper we demonstrate our results of implementing various steps of AFIS processing using NVIDIA CUDA technology. The paper is organized as follows. In section 2 an overview of parallel fingerprinting algorithms research to date is given. A small overview of CUDA technology is provided in section 3. The discussion of various aspects in parallel fingerprint processing is given in sections 4 and 5 and a conclusion is made in section 6.

# Related works

Prabhakar and Kameswara in [1] propose a scheme for a coarse minutia-based matcher for filtering out stored fingerprint templates that are unlikely to match the presented one. The proposed scheme is a de-facto continuous classifier [LINK TO ALESSANDRA LUMINIA] with a percentage of matched minutiae used to form a point in a 1D space. Each feature of a query minutiae vector is loaded to a separate processor elements (PE). Every feature of a stored minutia vector is matched independently with every query feature. If at least one of the PE produces a positive result, total matched count is increased. However the algorithm is made in assumption that two fingerprints are already aligned in orientation and the core point is detected in both of them. It should be especially noted that authors explicitly stated the usefulness of a SIMD processor for implementing the algorithm.

A similar approach is proposed by Ratha, Jain and Rover in [2] and [3] for FPGA which are well known for their parallel capabilities. Additional steps are done to prevent the same feature from being matched more than once. Matching speed of 2.6\*10^5 matches per second is reported. Unfortunately, this algorithm also requires two templates to be aligned in orientation and by their core position. SIMD processor efficiency for the task is once again noted. Authors also make a statement about two levels of parallelism in matching algorithm: macro-level for inter-template matching and micro-level for intra-template matching.

Gowrishankar in [4] describes a scheme of a feature extraction algorithm for a binary skeletonized image based on the neighbor colored pixels count and proposes the usage of massively parallel architecture of a computing device due to the information being extracted from a small neighborhood around each pixel.

Bai et al. in [5] investigate the capabilities of a NVIDIA graphic boards supporting CUDA for the parallel implementation of a well-known FingerCode algorithm [LINK TO JAIN]. It should be noted that authors use a slightly modified version of a FingerCode due to technological limitations of a chosen matching scheme, but report a performance of 2.03 million of matches per second. Our previous studies also indicate a great potential in FingerCode’s parallel implementation, especially after spatial characteristics optimization (band number, band width, innermost radius, fnumber of filters, number of sectors).

Cappelli, Ferrara and Maltoni report of a new local minutiae template based on a metric calculated for a minutiae around a sample one in a 3D lattice points inside a cylinder structure and its matching technique in [6] with a consequent consolidation step. The optimized version of such a template can be reduced to be a bit vector which can be efficiently matched with each other. Different approaches for consolidation step are then proposed. In a later press-release [LINK TO BIOLAB] authors state the increase of performance from 2.5 million mathes per second on a CPU to 4 million on a GPU.

# NVIDIA CUDA overview

The NVIDIA CUDA technology is used in a GeForce family of graphic boards [LINK]. Its programming model is shown on fig.2.



Fig.2. CUDA programming model

According to this model all computations are divided to CPU-based and GPU-based with CPU aliased as host and GPU as devised. During computation the host calls the device sending a program kernel – a compiled GPU shader. Prior to a kernel execution a device threading configuration occurs. It is assumed a large number of threads will be created during kernel execution. Those threads are united as blocks. All threads of a single block are executed on a single device multiprocessor, and the threads in the block are physically united in the warps where instructions are executed simultaneously. The set of blocks makes a grid. Currently the number of threads in a warp is 32, the dimensionality of block and grid is not bigger than 3 with a limitation of a 1024 threads maximum per a single block. Up to 16 blocks with a total number of threads not bigger than 2048 can be executed simultaneously on a single multiprocessor.

The CUDA technology allows the algorithms development for a powerful streaming processor with a possibility of a simultaneous computations in a several thousand threads at once. Of course there are some notable limitations:

1. The block size should be chosen with a register memory consumption per thread in mind as its amount is limited in each multiprocessor.
2. Global memory I/O are recommended to be optimized in such a way that, simply put, threads with a neighboring indexes are accessing neighboring 4-bytes words. This will allow to access the data in batches instead of single word access for each thread.
3. It’s forbidden to use recursive functions.
4. Branching to different paths is executed slower than branching to a single path by all threads.

Thus implementing some algorithms in GPGPU might be difficult.

# Image enhancement and minutiae extraction

Image enhancement is a crucial step for real-time AFIS, as the impression quality is generally lower during system usage than during registration. Different methods were proposed to improve the quality of the acquired fingerprint, i.e [LINKS]. It was shown that contextual enhancement is better than context-less enhancement, and Fronthaler, Kollreider and Bigun in [BIGUN LOCAL FEATURES] demonstrate the superiority of the pixel-wise enhancement versus block enhancement, e.g. it doesn’t produce artifacts at blocks’ borders.

The enhancement method involving Laplacian-like pyramid image decomposition and Linear Symmetry (LS) estimation proposed by Fronthaler, Kollreider and Bigun in [BIGUN LOCAL FEATURES] is also considered to be efficient if implemented on massively parallel architectures for the following reasons:

* It may be viewed as a sequence of transformations where each transformation uses only the previous results to obtain the next one – every transformation is non-iterative.
* Every transformation operates on a 2D matrices in such a way that each element’s computation is independent from the computation of other elements other than by the source data, which is immutable.
* Furthermore, most of the transformations include 2D convolutions, either ordinary or complex, that can be efficiently parallelized [something for convolutions], and most kernels are gaussians, so their separability property may as well be used.

In our studies we’ve implemented convolutions in a straightforward way without using separability due to the simplicity of the code, relatively small size of the kernels and the memory access pattern while a parallel processing was done on an element level. Thus our timings can further be improved. The size of Gaussian kernels is determined using a “3 sigma” rule by a formula. It could also be observed that amount of operations per pixel is constant, so the processing time is a linear function from the amount of pixels in the image or, in other words, image area.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| σ1 | 0.6 | f0 | 1.7 | f2,f3 | 1.3 | τ2 | 0.3 | σdirections | 2.0 |
| σ2 | 3.2 | f1 | 1.21 | τ1 | 0.1 | Annulus radii | Inner 4, outer 6 | Number of directions | 20 |

|  |  |  |
| --- | --- | --- |
| Database (resolution) | NVIDIA GeForce 560, ms | NVIDIA GeForce 570 |
| FVC2000 DB2 (256 x 364) | 20 | 15 |
| FVC2004 DB1 (640 x 480) | 60 | 43 |

Minutiae extraction, also described in [BIGUN LOCAL FEATURES] is performed afterwards. LS and Parabolic Symmetry (PS) are estimated in order to produce the final measure PSi(x,y) = PS(x,y)\*(1-|LS(x,y)|). The area of each local maximum (Xm, Ym) is checked for a consistent high LS, and the list of maximums is then sorted by a descending |PS(Xm, Ym)| as a certainty measure resulting in a minutiae list. For the reasons described below we’re storing only top 32 minutiae as a template. Convolutions and local maximum estimations are implemented in parallel while the minutiae list generation and sorting is done sequentially. Generally, those operations are lightweight compared to LS and PS estimation. Table X contains the parameters of the process being used. Those parameters should be chosen carefully in order to minimize the amount of spurious minutiae. However we’ve observed a consistency in their detection among different impressions of the same finger (fig. X3). The timings are shown in Table X3.

|  |  |  |
| --- | --- | --- |
| Database (resolution) | NVIDIA GeForce 560, ms | NVIDIA GeForce 570 |
| FVC2000 DB2 (256 x 364) | 20 | 20 |
| FVC2004 DB1 (640 x 480) | 60 | 43 |

# Minutia-based fingerprint matching

Minutiae-based fingerprint matching is a cornerstone of a modern AFIS. As noted in [2], database matching can be done in parallel on 2 levels: micro-level where different minutiae are matched in parallel with a given one and macro-level where different fingerprints are matched in parallel. However we need to point out that in [1], [2] and [3] are operating on an already aligned set of fingerprints, and that may not be the case for a real-time AFIS. Fingerprint alignment is a difficult task that can be formulated in a following way: given two fingerprint templates t1 and t2 what are the best parameters for translation and rotation to maximize the matching score? A detailed survey of the developed methods can be found in [Handbook].

In our approach we’re assigning one CUDA block to each of the database fingerprints so it scales naturally on different and more advanced graphic boards. Each block consists of 32x32 threads, and currently it is the technological maximum for CUDA. Thus we’re limiting the template size to 32 minutiae. Both query template Q {qi} and database template D {di} are loaded to the cache memory of a multiprocessor. After that for every minutia qn and dk in these templates a template copy (Qn {qni} and Dk {dki} accordingly) is made and translation is done to make the minutia qnn and dkk coordinates equal to (0;0), so indices n and k can be used to indicate translation parameters. The thread Tij then uses a Qi and Dj templates to calculate the possible rotational alignment.

First, we’re assuming that fingerprints are already coarsely rotationally aligned by a scanner, that is, it’s acquisition window or similar sensor is designed in a way that makes it difficult or inconvenient to acquire a rotated fingerprint, which is usually the case for modern scanners. For every minutiae pair in (qim∈Qi\qii; dil∈ Dj\djj) the difference between Euclidean distances and is computed. After that the slope angles of and and their difference Thetaimjn are calculated. If distance difference and angle difference are below certain thresholds, (I, j, Thetaimjn) is considered to be a plausible alignment parameters and is stored in a cache memory.

The second step is matching two fingerprints given a set of plausible alignment parameters. The set is distributed equally among 1024 working threads. Matching procedure is similar to the one described by Wegstein in [], but it uses Euclidean distance differences instead of coordinate differences. Due to the constant amount of the minutiae in the templates the thread output is a number of the matched minutiae. Its maximum is searched among all thread outputs (and consequently among all the alignment parameters) and is considered to be the final matching score.

FVC 2000 DB2 was used in conjunction with image enhancement and minutiae extraction technique discussed earlier to obtain the quality results for a proposed approach for 3960 genuine and 386760 impostor matches. Its quality characteristics are displayed in Fig. . We report an average time of 0.41 ms for a single fingerprint match.

# Discussion and conclusion

Our results demonstrate that filter-based and. Despite our focus on a direct grayscale minutiae extraction, GPGPU usage is also a perspective approach for a skeleton-based techniques, as there are GPU-accelerated ways for producing image skeletons [Вот эти ребята], and minutia extraction from them (e.g. [Говришанкар?]) is highly parallelizable.

As for the minutiae matching, the matching speed is much lower than one could expect from a massively parallel architecture implementation. First of all, there’s definitely a place for a further optimizations in the implemented algorithm. Secondly, the alignment step of our approach is a “brute force” search which can produce several thousand of possible alignments. Despite the fact this algorithm and matching procedure are highly parallelizable by data, internally they’re implemented in a sequential manner due to the lack of a single block resources. Thus with a finer alignment algorithm resulting in a lesser amount of possible alignment parameters a noticeable speedup is expected. It should also be noted that with a pre-alignment step the implementation of a micro-level parallelism for minutiae matching on CUDA is a straightforward decision.

Another concern is the recognition quality the amount of minutiae being used. 32 minutiae is, on average, only 50 to 75% of its total amount in a given fingerprint. Yet from the ROC one is able to devise a FAR=0.001% at a price of FRR=25%. While some fingerprints in a database contain unrecoverable regions, we consider the spurious minutiae to be the core reason of the high FRR, as any amount of false minutiae in a limited size template worsens the recognition quality. It is another field for improvement.

The code for the algorithms discussed is available at <http://code.google.com/p/cuda-fingerprinting/> and requires CUDA 2.0 compatible graphic board to run.

Literature

1. A parallel algorithm for fingerprint matching, RVSN Prabhakar, Kameswara Rao
2. Ratha, Jain, Rover. An FPGA-based Point Pattern Matching Processor with Application to Fingerprint Matching
3. Ratha, Jain, Rover. Fingerprint Matching on Splash 2
4. Gowrishankar, Fingerprint Identification on a Massively Parallel Architecture
5. Bai et al. GPU-accelerated fingerprint matching
6. Cappelli, Ferrara, Maltoni, Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition