

METHODS FOR ACCURACY-PRESERVING ACCELERATION OF LARGE-SCALE COMPARISONS IN CPU-BASED IRIS RECOGNITION SYSTEMS

C. Rathgeb¹ • N. Buchmann¹ • H. Hofbauer² • H. Baier¹ • A. Uhl² • C. Busch¹

¹da/sec – Biometrics and Internet Security Research Group, Hochschule Darmstadt, Germany

{christian.rathgeb, nicolas.buchmann}@h-da.de

²Multimedia Signal Processing and Security Lab, University of Salzburg, Austria

{hhofbaue, uhl}@cosy.sbg.ac.at

Abstract

To confirm an individual's identity accurately and reliably iris recognition systems analyse the texture that is visible in the iris of the eye. The rich random pattern of the iris constitutes a powerful biometric characteristic suitable for biometric identification in large-scale deployments. Identification attempts or deduplication checks require an exhaustive one-to-many comparison. Hence, for large-scale biometric databases with millions of enrollees the time required for a biometric identification is expected to significantly increase.

In this work we analyse techniques to accelerate Hamming distance-based comparisons of binary biometric reference data, i.e. iris-codes, in large-scale iris recognition systems, which preserve the biometric performance. Focus is put on software-based optimizations, an efficient two-step iris-code alignment process referred to as *TripleA*, and a combination thereof. Benchmarking the throughput and identifying potential bottlenecks of a portable commodity hardware-based iris recognition system, is of particular interest. Based on conducted experiments we point out practical boundaries of large-scale comparisons in CPU-based iris recognition systems, bridging the gap between the fields of iris recognition and software design.

Contents

1	Introduction	2
1.1	Contribution of Work	2
1.2	Organisation of Article	2
2	Related Work	2
3	Iris Recognition System	3
3.1	Preprocessing and Feature Extraction	3
3.2	Iris-Code Comparison	4
4	Software-based Optimizations	4
4.1	Look-up Tables, Intrinsic and Loop-Unrolling	4
4.2	Multithreading and Statistical Micro-Ops Optimisation	4
5	Accelerated Accuracy-preserving Alignment	5
5.1	Iris-Code Analysis	5
5.2	TripleA	5
6	Experiments	6
6.1	Experimental Setup and Methodology	6
6.2	Software-based Optimizations	7
6.3	Accelerated Accuracy-preserving Alignment	9
6.4	Simulation of Large Scale Identification	9
7	Conclusions	10

1 Introduction

The rich random structure of the iris, and hence its resistance to false matches, constitutes one of the most powerful biometric characteristics [1]. Following Daugman’s approach [1], which represents the core of most public operational deployments, four processing components form an iris recognition system: (1) *acquisition*, where most current deployments require subjects to fully cooperate with the system in order to capture images of sufficient quality; (2) *pre-processing*, which includes the detection of the pupil and the outer iris boundary. Subsequently, the iris (approximated in the form of a ring) is normalized to a rectangular texture. To complete the preprocessing, parts of the iris texture which are occluded by eye-lids, eye-lashes or reflections are detected and stored in an according noise-mask; (3) *feature extraction*, in which an iris-code is generated by convolving local regions of the pre-processed iris texture with filters and encoding responses into bits. This binary data representation enables compact storage and rapid (4) *comparison*, which is based on the estimation of Hamming distance (*HD*) scores between pairs of iris-codes and corresponding masks. In the comparison stage circular bit shifts are applied to iris-codes and *HD* scores are estimated at K different shifting positions, i.e. relative tilt angles. The minimal obtained *HD*, which corresponds to an optimal alignment, represents the final score. It is important to note, that the number of shifting positions employed to determine an appropriate alignment between pairs of iris-codes may vary depending on the application scenario. Some public deployments of iris recognition go as far as $K = 21$ shifting positions when handheld cameras are used for which it is more difficult to ensure an upright capture orientation [2]. Hence, score distributions are skewed towards lower *HD* scores, which (for a given threshold) increases the probability of a false match by the factor K [2].

Nowadays iris recognition technologies are already deployed in numerous nation-wide projects. Simplicity in design and development as well as the usage of commodity hardware are driving factors behind the deployment of large-scale biometric systems, e.g. the Indian Aadhaar project [3] in which thousands of CPU cores are processing millions of transactions on a daily basis. In such systems identification attempts or de-duplication checks might represent a bottleneck, since these require an exhaustive $1 : N$ comparison where N represents the number of subjects registered with the system. In particular, comparison time represents a crucial factor, which dominates the overall computational workload in any large-scale biometric identification system, especially if large values of K are unavoidable.

1.1 Contribution of Work

In this work focus is put on an iris recognition system, which performs a CPU-based exhaustive search for each

authentication attempt. The presented study represents a more common scenario, in contrast to proposed studies, which analyse hardware-specific acceleration of iris recognition systems. Our analyses include a comparative study of the most efficient ways to count disagreeing bits between iris-codes. The potential of manual loop-unrolling as well as different extensions to the x86 instruction set architecture for microprocessors are analysed. In addition, multi-threading techniques and statistical optimization of micro-operations are considered. Furthermore, we estimate the inter-relation between throughput and rotation compensation provided by an iris recognition system. In order to further accelerate a single pair-wise comparison of iris-codes, we build upon the work of [4], where we proposed a novel technique for comparing pairs of iris-codes, which we refer to as Accelerated Accuracy-preserving Alignment – *TripleA*. This method focuses on the alignment process, in which an adjustable two-step search-procedure is employed in order to efficiently determine alignments between iris-codes. Within this procedure only a fraction of K shifting positions has to be considered during a single pair-wise comparison, while covering the same range of possible tilt angles. In this work, we enhance the *TripleA* scheme by applying it to an optimized CPU-based iris recognition scheme. We show that, the *TripleA* method can be seamlessly integrated, such that the resulting system takes full advantage of *TripleA* on top of software-based optimisations. In summary, this work provides a detailed guidance of how to substantially accelerate large-scale iris biometric systems on commodity hardware in an accuracy-preserving manner, by combining software-based optimizations with a technique for efficient iris-code alignment. Moreover, summarized key observations might as well provide explanations for anomalies reported in existing studies.

1.2 Organisation of Article

This article is organized as follows: related works are discussed in Sect. 2. In Sect. 3 the employed iris recognition system is summarized. A detailed analysis of software-based acceleration techniques is given in Sect. 4 and the *TripleA* method is described in Sect. 5. Experimental results are presented in Sect. 6. Finally, conclusions are drawn in Sect. 7.

2 Related Work

To circumvent the bottleneck of an exhaustive $1 : N$ comparison, different concepts have been proposed in order to reduce the workload in an iris biometric (identification) system. We might differentiate between four key concepts: (1) coarse classification or “binning”, (2) a serial combination of a computationally efficient and a conventional system, (3) indexing schemes, and (4) hardware-based acceleration.

By binning an iris biometric database into several classes, the workload can be divided by the number of classes, given that irises of registered subjects are equally distributed among them. Natural features to be utilized include eye position (left or right) [5] or eye colour [6, 7]. Recent advances in the field of soft biometrics suggest further possible classification based on gender [8], age groups [9], or ethnicity [10, 11] (for further details on soft biometrics the reader is referred to [12]). Instead of creating tangible, human-understandable classes, it is also possible to rely on distinct iris texture features [13–15]. Binning is equivalent to the combination of biometric systems. Hence, classification errors might significantly increase the false non-match rate (FNMR) of the overall system. Moreover, the potential benefit of binning is limited by the number of bins which determines the factor by which the database size can be reduced.

Within serial combinations computationally efficient biometric systems are used to extract a short-list, i.e. small fraction, of most likely candidates. This procedure might be referred to as pre-screening. While generic iris recognition systems already provide a rapid comparison, more efficient biometric comparators can be obtained by employing compressed versions of original iris-codes during pre-screening [16, 17]. Further, a rotation-invariant iris recognition scheme can be applied in the pre-screening step [18]. Similar to binning approaches, a serial combination of a computationally efficient and an accurate (but more complex) scheme might increase the FNMR of the overall system. However, a serial combination enables a more accurate operation of the resulting trade-off between computational effort and accuracy by choosing an adequate size for the short-list.

Indexing schemes aim at constructing hierarchical search structures for iris biometric data, which tolerate a certain amount of biometric variance. Such schemes substantially reduce the overall workload of a biometric identification, e.g. $\log N$ in case of a binary search tree. Such search structures might be designed for iris-codes [19, 20] as well as iris images [21–23]. While the majority of works report hit/ penetration rates on distinct datasets, required computational efforts are frequently omitted. The application of complex search structures on rather small datasets may as well cloud the picture about actual gains in terms of speed and leaves the scalability of some approaches questionable.

Adapting comparison procedures to adequate hardware, e.g. multiple cores within a CPU, allows for parallelization [24]. By simultaneously executing a number of threads the workload can be significantly reduced since a $1 : N$ comparison can be performed in parallel on various subsets of equal size. Also the estimation of HD scores at various shifting positions during alignment can be parallelized. Moreover, iris-code comparisons can be efficiently performed on the GPU using GPGPU or CUDA [25], FPGA [24, 26], or other specialized hardware like CELL processors [27].

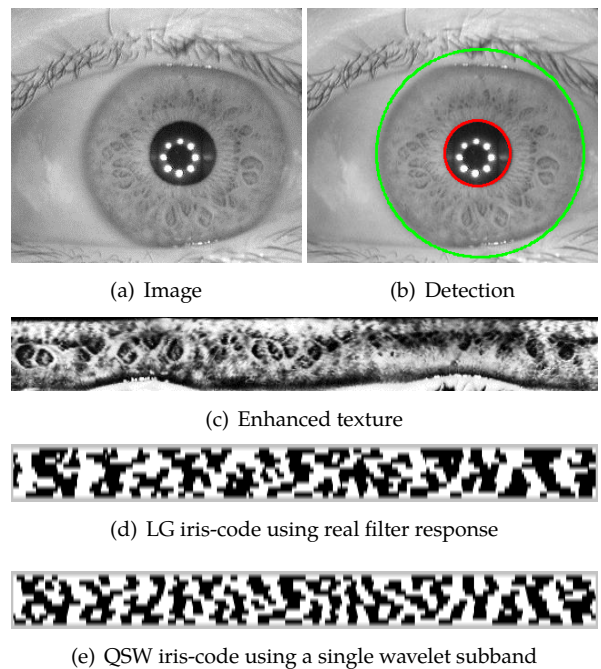


Figure 1: Common iris biometric processing chain for image S1008L02 of the CASIAv4-Interval iris database.

Apart from hardware-based acceleration, most of presented schemes either fail to provide a significant acceleration or they suffer from a significant decrease in recognition accuracy. Hence, existing approaches often obtain a trade-off between biometric performance (recognition accuracy) and speed-up, compared to a traditional iris recognition system. In practice most concepts do not allow for a seamless integration into a conventional identification system. The majority of hardware-specific acceleration techniques of iris recognition systems is custom-built, which makes it difficult to derive generally applicable methodologies or concepts. Moreover, anomalies in runtime tests are frequently left uncommented.

3 Iris Recognition System

The following subsections summarize the key components of the employed iris recognition systems.

3.1 Preprocessing and Feature Extraction

In the employed iris recognition system, which builds upon common processing components, the iris of a given sample image is detected and transformed to a rectangular texture of 512×64 pixels applying a contrast-adjusted Hough transform. The enhanced texture is obtained by applying contrast limited adaptive histogram equalization (CLAHE). In the feature extraction stage the enhanced texture is divided into stripes resulting in 10 one-dimensional signals, each one averaged from the pixels of 5 adjacent rows (the upper 512×50 rows are analysed). The first feature extraction method follows the Daugman-like 1D-LogGabor fea-

ture extraction algorithm of Masek [28] (LG) and the second follows the algorithm proposed by Ma *et al.* [29] (QSW) based on a quadratic spline wavelet transform. Both feature extraction techniques generate an iris-code IC , which consists of $B=512 \times 10=5,120$ bits. Fig. 1 illustrates the described processing chain for a sample iris image. Custom implementations of employed segmentation and feature extractors are freely available in the University of Salzburg Iris Toolkit (USIT) [30]. For further details on the employed feature extraction algorithms the reader is referred to [31]. Note that a compression of iris-codes, e.g. to 2,048 bits as suggested in [1], might cause a decrease in biometric performance [16], especially in challenging unconstrained scenarios.

3.2 Iris-Code Comparison

In the comparison stage circular bit shifts are applied to iris-codes and HD scores are estimated at K different shifting positions, i.e. relative tilt angles. In the used scheme a 1-bit shift equals 0.7° of rotation. Let $f(IC, i)$ denote an iris-code shifted by i bits. Assuming that blocks of L bits are processed at a time, the final comparison score between a query and a reference iris-code, IC_Q and IC_R , and their corresponding noise masks, M_Q and M_R , is estimated as,

$$\min_{i \in K} \frac{\sum_{j=1}^{B/L} \|(IC_{Q_j} \oplus f(IC_R, i)_j) \cap M_{Q_j} \cap f(M_R, i)_j\|}{\sum_{j=1}^{B/L} \|M_{Q_j} \cap f(M_R, i)_j\|}. \quad (1)$$

Since iris-codes can be shifted prior to comparison and only a single division is required, the workload for calculating scores between iris-codes is dominated by the following three (per-block) processing steps:

1. *XOR*: the exclusive or (\oplus) detects disagreeing bits between two L -bit blocks, resulting in bit block of same size where 1s indicate differing bits.
2. *POPCNT*: the population count ($\|\cdot\|$), or Hamming weight, counts the number of 1s in the vector extracted in the first step, i.e. the amount of detected differences.
3. *ADD*: the amount of disagreeing bits is added up (\sum) for all L -bit blocks.

Of these processing steps, *POPCNT* represents the most complex one and most of presented software-based optimisations will focus on speeding up its calculation (see Sect. 4). Nevertheless, the other two steps are also analysed where appropriate.

4 Software-based Optimizations

From a practical point of view, we identified seven settings as most relevant, *S-1* to *S-7*, which are described in the following subsections.

4.1 Look-up Tables, Intrinsics and Loop-Unrolling

Look-up table (S-1): the population count of $L = 8$ bit blocks is stored in a pre-computed look-up table. An 8-bit look-up table has a small memory footprint (256 byte) and is universally applicable in contrast to a register-sized look-up table, e.g. 64-bit (~ 16.7 million terabyte), which is far too big even for common memory sizes in the foreseeable future. For the *XOR* and *ADD* step common arithmetics are used.

Hardware POPCNT (S-2): intrinsics are used to calculate the population count with the SSE4 *POPCNT* CPU instruction. Experiments are performed in 32-bit and 64-bit operation mode.

Assembler POPCNT (S-3): instead of high level intrinsics the *POPCNT* command is directly invoked via inline assembler code in a C++ function.

Manual loop-unrolling (S-4): even though loop-unrolling is activated for the compiler, this experiment measures the impact on the overall duration regarding the (manually adjusted) number of bit blocks processed per loop iteration.

SSE2 and AVX (S-5): we also consider calculating *XOR* for 128-bit blocks with the Streaming SIMD Extensions 2 (SSE2) instruction *PXOR*, the Advanced Vector Extensions (AVX) 256-bit equivalents *VXORPD*, the AVX2 256-bit version *VPXORPD* and measure the impact of addition trees using the AVX2 8-bit and 16-bit vectoring commands *VPADDB* and *VPADDW*. The latter operations can add 32 8-bit packed integers and 16 16-bit packed integers with one operation, respectively.

4.2 Multithreading and Statistical Micro-Ops Optimisation

Multithreading (S-6): iris-code comparisons are split upon multiple threads. Like in the previous settings, *S-2* to *S-5*, *POPCNT* and *ADD* operations are performed alternately (*PAPA*). First a given query iris-code is compared to all pre-shifted versions of stored reference iris-codes. Hence, no shifting operations have to be performed at the time of comparison, while storage requirement, which is usually not a crucial factor, increases. In an alternative implementation the *query* iris-code is shifted prior to comparison against all stored non-shifted *reference* iris-codes. Both settings, which are referred to as *PAPA_R* and *PAPA_Q*, describe the same transposed algorithm and result in the same amount of bit comparisons.

Statistical micro-ops optimisation (S-7): static data dependency, latency and throughput analysis are utilized to minimise latencies of micro-operations. The resulting strategies, which are referred to as *PPAA_R* and *PPAA_Q*, perform all *POPCNT* operations first and add up all intermediate results afterwards.

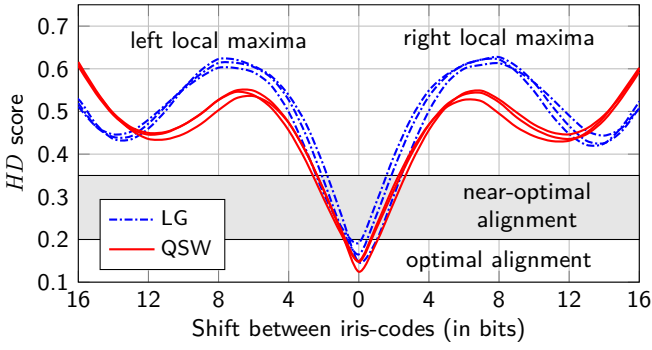


Figure 2: Sample HD-scores obtained from three genuine pairs of iris-codes at various shifting positions.

5 Accelerated Accuracy-preserving Alignment

The following subsections present an analysis of HD scores estimated from genuine iris-code comparisons across various shifting positions, which motivates the adjustable two-step search-procedure, referred to as *TripleA* [4].

5.1 Iris-Code Analysis

For both feature extractors Fig. 2 shows the HD scores across different shifting positions for three genuine comparisons of iris-codes. It can be seen that, for each feature extraction algorithm the HD scores of the three genuine comparisons seem almost identical. Within a certain range HD scores constantly decrease towards the minimum (best) score. This range is enclosed by local maxima resulting in HD scores significantly beyond 0.5. For the sample HD scores in Fig. 2 these local maxima can be detected at shifting positions of ± 8 bits for LG and ± 6 bits for QSW. A detailed analysis of this phenomenon is provided in [4].

Intuitively, the distance between the shifting position resulting in a minimum HD score and those of surrounding local HD score maxima might be approximated by the average length of 1-bit and 0-bit sequences μ , as $\pm\mu$ bit shifts are expected to cause the most drastic misalignment. The sequence of HD scores between genuine iris-codes across various shifting positions might be interpreted as an oscillation which decreases its amplitude with the distance to the minimum score. For such a signal it can be empirically verified that distances between consecutive vertices are virtually the same for a constant value of μ even in case of large standard deviations.

5.2 TripleA

The *TripleA* approach comprises the following two key steps: (1) estimation of near-optimal alignment and (2) estimation of subset-minimum. An example of the approach is illustrated in Fig. 3.

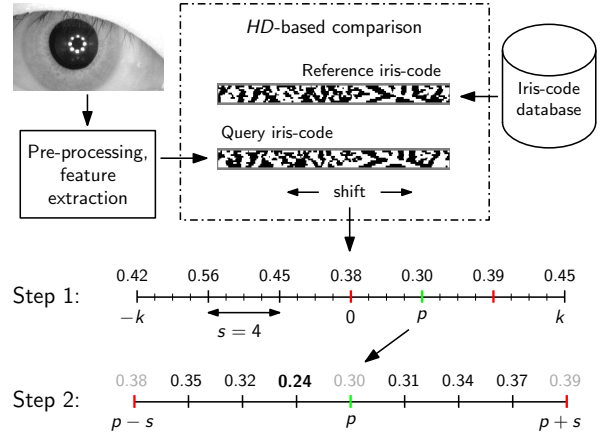


Figure 3: Example of the *TripleA* procedure: In the first step comparisons between a query and reference iris-code are performed at $2 \lfloor k/s \rfloor + 1 = 7$ positions according to the reference's step size $s = 4$. After detecting the near-optimal shifting position $p = 4$, the final score (marked bold) is detected in the interval $[p - s + 1 = 1; p + s - 1 = 7]$ at a shifting position of 3. HD scores are estimated at a total number of 13 shifting positions compared to $K = 25$ in a linear search.

In the first step the range of $K = 2k + 1$ shifting positions $[-k; k]$ is divided into $2 \lfloor k/s \rfloor$ intervals, where s denotes the employed *step-size*. Then HD scores are estimated at interval boundaries, i.e. for a subset of $2 \lfloor k/s \rfloor + 1$ shifting positions. In other words, the sequence of scores, interpreted as signal, is sampled every s bits. For a genuine comparison a sampling with at most the average length of 1-bit and 0-bit sequences, $s < \mu$, is expected to detect a minimum score which represents a near-optimal alignment. We consider an alignment as near-optimal if the corresponding shifting position is close enough to the optimal alignment revealing a HD score, which is significantly smaller compared to remaining sampling positions. For the sample comparisons of Fig. 2 near-optimal alignments would be found in the range of approximately ± 2 bit shifts.

After detecting a near-optimal alignment at shifting position p the interval $[p - s + 1; p + s - 1]$ is considered for the second step. Note that the scores for positions $p \pm s$ have already been estimated in the first step. Based on a linear search the second step detects a minimum HD score for a subset of $2(s - 1)$ shifting positions. That is, the number of shifting positions to be considered is reduced to $C = 2 \lfloor k/s \rfloor + 1 + 2(s - 1)$. To further accelerate the *TripleA* alignment procedure it is suggested to process only half of the subset detected in the first step during the second step. This bisected interval is defined by p and minimum of surrounding HD scores at $p \pm s$. Hence, the number of shifting positions is further reduced to $C = 2 \lfloor k/s \rfloor + s$. In the example of Fig. 3 the interval $[p - s + 1, p - 1]$ would be chosen for the linear search of the second step, since the HD score at shifting position $p - s$ is smaller than that at $p + s$. This derivation is referred to as *TripleA-Single-Sided*. In Fig. 4 the number of

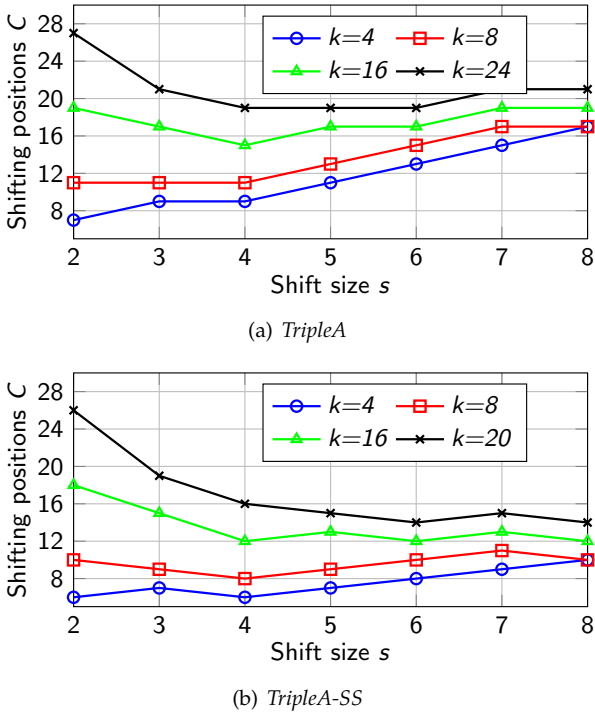


Figure 4: Number of shifting positions to be considered C using *TripleA* and *TripleA-SS* for different values of k and s .

shifting positions C is plotted for different values of k and s . To obtain a maximum speed-up C has to be minimized, such that $s = \sqrt{2k}/\sqrt{2}$ and $s = \sqrt{2k}$ represent the theoretical optimal step-size in terms of speed-up for *TripleA* and *TripleA-SS*, respectively.

In [4] we showed that, μ can be dynamically estimated from a single reference iris-code during enrolment, however, this dynamic estimation was not found to yield any significant gains in terms of performance are obtained. Hence, we restrict to applying static values of s for each comparison performed by the system. In this case μ can be averaged from a training set of extracted iris-codes.

6 Experiments

The following subsections describe the experimental setup and summarize results obtained by the presented approaches.

6.1 Experimental Setup and Methodology

Experimental evaluations are carried out on the CASIAv4-Interval iris database [32]. The database consists of $N=2,639$ good-quality 320×280 pixel NIR iris images of 249 subjects. We consider two types of experiments, where in both experiments an iris-code is compared against K shifted versions of another one:

Experiment 1 (E-1): the maximum number of $N(N-1)/2 = 3,480,841$ iris-code cross-comparisons is performed. Based on obtained scores we identify an adequate trade-off between biometric performance and provided rotation compensation. Subsequently, diverse settings with the aim

of accelerating these iris-code cross-comparisons are compared and the best setting is identified. For time measurements we execute a total number of 40 iterations and the obtained median time elapsed is reported. The considered number of iterations minimizes the influence of outliers with respect to time measurements, which assures significance of relative improvements or degradations in comparison speed. This experiment might reflect a de-duplication check on an iris-code database with N registered subjects.

Experiment 2 (E-2): the dataset is partitioned into a reference set of 2,500 iris-codes and a query set of 139 iris-codes. To simulate identification attempts on a large-scale database the reference set is extended to a large-scale dataset by replicating the subset 20,000 times, resulting in a set of $N=2,500 \times 20,000=50,000,000$ iris-codes. Note that the obtained set is used for runtime experiments only. For the best setting of *E-1*, in terms of throughput, all 139 identification attempts (1:N) are performed and the obtained median time elapsed is reported for various degrees of rotation compensation. Subsequently, the *TripleA* method is applied with different parameter configurations on top of the best setting of *E-1* in order to obtain further speed-ups.

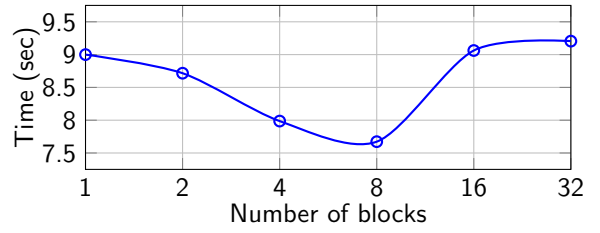
The main difference between these experiments is that, while in *E-1*, the de-duplication experiment, a total number of N query iris-codes are successively compared against the database, in *E-2*, the identification experiment, a single query iris-code is compared against a huge database.

Biometric performance is estimated in terms of FNMR at a target false match rate (FMR) and equal error rate (EER) obtained from *E-1*. The test system for measuring the duration of *E-1* and *E-2* with different settings uses an x86_64 Linux operating system with kernel version 4.4 and GCC 5.3.0 as C++ compiler. While other CPU-types, e.g. ARM-based, have been analysed with respect to the required operations [33], focusing on large-scale biometric systems x86_64 hardware is considered as most relevant. The utilised CPU is an Intel Core i7-6700 with sufficient DDR4-SDRAM 2133.

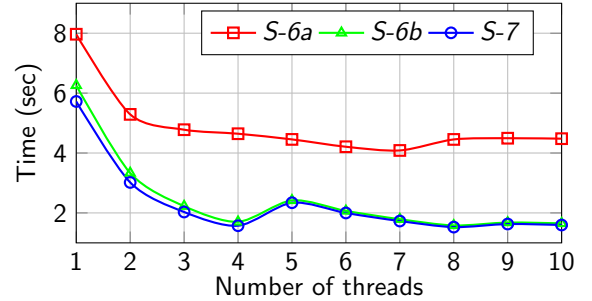
In order to identify an appropriate degree of rotation compensation in *E-1*, we first calculate EERs and FNMRs at a FMR of 0.01%, denoted as $\text{FNMR}_{0.01}$, considering $\pm k$ shifting positions during alignment. The progress in terms of EER and $\text{FNMR}_{0.01}$ with respect to rotation compensation is shown in Table 1. As can be seen, the majority of misalignments is compensated by ± 8 bit shifts ($\sim 6^\circ$) while biometric performance converges at approximately ± 16 bit shifts ($\sim 11^\circ$). Focusing on recognition accuracy versus required bit-shifting we choose $k = \pm 16$, resulting in $2k + 1 = 33$ shifting positions, is considered as reasonable trade-off for the used iris recognition systems resulting in an EER of 0.80% and a $\text{FNMR}_{0.01}$ of 1.75% for LG and an EER of 0.74% and a $\text{FNMR}_{0.01}$ of 1.06% for QSW.

Rot. comp. $\pm k$ bits	LG		QSW	
	EER	FNMR _{0.01}	EER	FNMR _{0.01}
0	6.81	11.98	16.14	20.35
1	5.65	10.73	11.51	15.12
2	5.01	10.18	8.22	11.33
4	2.78	9.89	3.03	6.24
8	1.04	2.26	0.94	1.46
12	1.01	2.23	0.79	1.28
16	0.80	1.75	0.74	1.06
20	0.80	1.75	0.73	1.05
24	0.79	1.71	0.70	1.01

Table 1: Progression of EERs and FNMR_{0.01}s in relation to rotation compensation (the selected setting for the used iris recognition system is marked bold).



(a) S-4



(b) S-6 and S-7

Figure 5: Time measurements (in seconds) obtained for (a) setting S-4 and (b) settings S-6 and S-7 in experiment E-1 performing all 3,480,841 iris-code cross-comparisons at 33 shifting positions.

Setting			Setting			
ID	Description	Time	ID	Description	Time	
S-1	8-bit Look-up table	157.95	S-6a	3 Threads PAPA _R	4.78	
S-2	POPCNT 32-bit	14.98		4 Threads PAPA _R	4.62	
	POPCNT 64-bit	9.16		5 Threads PAPA _R	4.46	
S-3	POPCNT ASM	8.16		6 Threads PAPA _R	4.21	
S-4	2 Blocks	8.71		7 Threads PAPA _R	4.09	
	4 Blocks	7.98		8 Threads PAPA _R	4.45	
	8 Blocks	7.65		9 Threads PAPA _R	4.50	
	10 Blocks	9.54		10 Threads PAPA _R	4.47	
S-5a	16 Blocks	9.05		S-6b	1 Thread PAPA _Q	6.24
	32 Blocks	9.21			2 Threads PAPA _Q	3.33
	2 Blocks SSE2	12.08	3 Threads PAPA _Q		2.21	
	4 Blocks SSE2	12.43	4 Threads PAPA _Q		1.70	
	8 Blocks SSE2	10.26	5 Threads PAPA _Q		2.42	
S-5b	16 Blocks SSE2	9.37	6 Threads PAPA _Q		2.06	
	32 Blocks SSE2	8.33	7 Threads PAPA _Q		1.79	
	4 Blocks AVX	10.76	8 Threads PAPA _Q		1.58	
	8 Blocks AVX	12.34	9 Threads PAPA _Q		1.68	
	16 Blocks AVX	8.01	10 Threads PAPA _Q		1.65	
S-5c	32 Blocks AVX	8.05	S-7	1 Thread PPAA _Q	5.73	
	4 Blocks AVX2	11.90		2 Threads PPAA _Q	3.02	
	8 Blocks AVX2	12.32		3 Threads PPAA _Q	2.03	
	16 Blocks AVX2	8.00		4 Threads PPAA _Q	1.57	
S-5d	32 Blocks AVX2	8.06		5 Threads PPAA _Q	2.33	
	AVX2 8-bit ADD	9.63		6 Threads PPAA _Q	1.99	
S-5e	AVX2 16-bit ADD	9.32		7 Threads PPAA _Q	1.72	
	SSSE3	20.66		8 Threads PPAA _Q	1.54	
S-6a	1 Thread PAPA _R	7.96		9 Threads PPAA _Q	1.63	
	2 Threads PAPA _R	5.31		10 Threads PPAA _Q	1.58	

Table 2: Overview of time measurements (in seconds) obtained for different settings in experiment E-1 performing all 3,480,841 iris-code cross-comparisons at 33 shifting positions.

6.2 Software-based Optimizations

Table 2 summarizes time measurements for all settings in experiment E-1. Since time measurements might highly depend on hardware components of a system, emphasis should be placed on relative improvements obtained by according optimization techniques. Optimal parameters of each setting are preserved in subsequent settings where appropriate.

With over two minutes runtime the 8-bit look-up table of S-1 turns out to be by far the slowest implementation. Nevertheless it represents a baseline for a hardware independent implementation.

Without any optimisation the 32-bit population count implementation in S-2, using intrinsics to invoke the SSE4 POPCNT instruction provides a tenfold speed-up compared to S-1. The 64-bit version can double the data processing per instruction and is therefore even faster. It is not twice as fast as the 32-bit implementation due to overhead of the bigger 64-bit address handling for data access and pointer dereferencing. Based on this observation subsequent settings process blocks of $L = 64$ bits.

The inline assembler of S-3 also provides a clear speed-up over high level POPCNT intrinsic calls used in S-2.

Focusing on S-4, Fig. 5(a) shows that the preferred number of L -bit blocks processed per loop iteration is 8. We identify two reasons to justify this behaviour: on the one hand 8 64-bit blocks fit very well in the general purpose registers of the x86_64 processor and no memory access is needed for the XOR, POPCNT, ADD operation, see Fig. 6(b) lines 20-36; on the other hand 8×64 bit are exactly 64 byte which is the

same size as one CPU cache line. Since a cache line copied from memory is exactly 64 byte it is preferable to process the complete cache line resulting in a favourable cache hit/miss ratio. We therefore recommend the processing of data in 64 byte blocks and storing it as a continuous array for an optimal exploitation of the CPU caches. Hence, in settings *S-6* and *S-7* a total number of 8 64-bit blocks are processed per loop iteration.

Settings *S-5a*, *S-5b* and *S-5c* make use of SSE2, AVX and AVX2 instructions to process bigger data chunks with the XOR operation. However, no significant speed-up over the common x86 64-bit XOR instruction is obtained. The reason for this is very straightforward, since SSE works on specific registers, the so called 128-bit XMM registers and AVX on the 256-bit YMM registers. Data has to be loaded to and retrieved from these registers before it can be used with SSE/AVX instructions. In contrast, the SSE4 *POPCNT* command operates on 64-bit general purpose registers of a CPU. Therefore, a transfer between these registers is necessary where the overhead for these transfers is higher than a straightforward processing by the common XOR command which operates on the same registers as the *POPCNT* instruction. SSE and AVX are optimised for algorithms which do a lot of operations on a comparably low amount of data. Calculating a great amount of iris-code comparisons, which requires only very few operations on extreme amounts of data, is no such problem. Settings *S-5d* and *S-5e*, which implement the AVX2 vector addition, are slower for the same reasons. Note that the SSSE3 implementation tested in *S-5e* is considered the fastest *POPCNT* implementation by experts in the field [34]. In contrast, we observe that the hardware *POPCNT* instruction used in *S-2* to *S-4*, is clearly superior to the SSSE3 implementation. Still, for older CPUs where no *POPCNT* instruction is available, this could still be of interest since it is faster than an 8-bit look-up table.

The common idea to compare a freshly extracted query iris-code to a large pre-shifted database of reference iris-codes is represented in *S-6a*. As shown in Fig. 5(b) for 1 to 3 threads this setting behaves as expected, but starting from 4 threads the runtime stagnates at roughly 4 seconds, i.e. dividing the workload in more threads provides no further speed-up. As one iris-code consists of 512×20 bits (1280 byte), we have 3,480,841 comparisons and for each comparison a new iris-code has to be loaded from memory, resulting in roughly 137 GB of data transferred from memory to the CPU. Our experiment computer uses DDR4-2133 RAM with a speed of 17.0 GB/s per channel according to specification [35]. We are using a common dual channel setup and, hence have a maximum RAM bandwidth of 34 GB/s. Hence, transferring 137 GB from memory to CPU takes at least 4 seconds. In this setup the execution speed of the implemented algorithm is interfered by the relatively slow RAM to CPU interface. The RAM as bottleneck is a common problem for highly multithreaded tasks performing a few

1 buf[0]=ic[x].dat[k][i]^ic[y].dat[i];	1 prefetch0 ptr [r13+r11*1]
2 buf[1]=ic[x].dat[k][i+1]^ic[y].dat[i+1];	2 prefetch0 ptr [r13]
3 buf[2]=ic[x].dat[k][i+2]^ic[y].dat[i+2];	3 xor edi, edi ; XOR
4 buf[3]=ic[x].dat[k][i+3]^ic[y].dat[i+3];	4 mov rax, qword ptr [r12]
5 buf[4]=ic[x].dat[k][i+4]^ic[y].dat[i+4];	5 mov rdx, qword ptr [r12+0x8]
6 buf[5]=ic[x].dat[k][i+5]^ic[y].dat[i+5];	6 xor rax, qword ptr [r13-0x138]
7 buf[6]=ic[x].dat[k][i+6]^ic[y].dat[i+6];	7 xor rdx, qword ptr [r13-0x130]
8 buf[7]=ic[x].dat[k][i+7]^ic[y].dat[i+7];	8 mov rcx, qword ptr [r12+0x10]
9	9 mov r8, qword ptr [r12+0x18]
10 asm(".intel_syntax noprefix\n");	10 xor rcx, qword ptr [r13-0x128]
11	11 xor r8, qword ptr [r13-0x120]
12 __asm__(12 mov r9, qword ptr [r12+0x20]
13 "popcnt %1, %1 \n\t"	13 mov r10, qword ptr [r12+0x28]
14 "popcnt %2, %2 \n\t"	14 xor r9, qword ptr [r13-0x118]
15 "popcnt %3, %3 \n\t"	15 xor r10, qword ptr [r13-0x110]
16 "popcnt %4, %4 \n\t"	16 mov rbx, qword ptr [r12+0x18]
17 "popcnt %5, %5 \n\t"	17 mov rsi, qword ptr [r12+0x38]
18 "popcnt %6, %6 \n\t"	18 xor rbx, qword ptr [r13-0x108]
19 "popcnt %7, %7 \n\t"	19 xor rsi, qword ptr [r13-0x100]
20 "popcnt %8, %8 \n\t"	20 popcnt rax, rax ; POPCNT
21	21 popcnt rdx, rdx
22 "add %0, %1 \n\t"	22 popcnt rcx, rcx
23 "add %0, %2 \n\t"	23 popcnt r8, r8
24 "add %0, %3 \n\t"	24 popcnt r9, r9
25 "add %0, %4 \n\t"	25 popcnt r10, r10
26 "add %0, %5 \n\t"	26 popcnt rbx, rbx
27 "add %0, %6 \n\t"	27 popcnt rsi, rsi
28 "add %0, %7 \n\t"	28 add rdi, rax ; ADD
29 "add %0, %8 \n\t"	29 add rdi, rdx
30	30 add rdi, rcx
31 : "+r" (dist)	31 add rdi, r8
32 : "r" (buf[0]), "r" (buf[1]),	32 add rdi, r9
33 : "r" (buf[2]), "r" (buf[3]),	33 add rdi, r10
34 : "r" (buf[4]), "r" (buf[5]),	34 add rdi, rbx
35 : "r" (buf[6]), "r" (buf[7])	35 add rdi, rsi
36);	36 mov rcx, rdi ; final result

(a) C++ / Inline ASM

(b) ASM

Figure 6: Comparison between (a) C++ code using Inline Assembler and (b) corresponding Assembler code for setting *S-7*.

operations on a big amount of data [36]. The bottleneck gets enhanced by the fact that this biometric scenario floods the CPU caches with all new data and is practically not using them at all, resulting in a very poor cache hit/miss ratio. In *S-6b* *K* shifted versions of the given query iris-code are computed and compared to *N* non-shifted reference iris-codes of the database. From a computational perspective, this setup seems less intuitive because the shifted versions have to be computed before the actual comparison can start, but the *K* iris-codes can stay in the CPU caches across all comparisons and only one 1,280 byte block has to be loaded for each comparison, resulting in much less actual memory access since the CPU caches have a high hit count for the shifted iris-codes [37]. Therefore, *S-6b* scales much better with multiple threads as highlighted in Fig. 5(b). Hence, the subsequent setting will be based on this strategy. Moreover, in *S-6b* we observe the effect that 5 threads are actually slower than 4 threads. The used Intel Core i7-6700 processor has 4 physical cores of which each can process 2 threads at once due to hyper threading [38]. As depicted in Fig. 7, in case 5 threads are used 2 threads have to share the L1 and L2 cache on one core. Therefore, the iris-code prefetching, see Fig. 6(b) lines 1-2, is not as effective as if one thread uses the complete cache. This effect occurs since both threads are working on completely independent parts of the iris-code database. Due to this aspect 8 threads are only negligibly faster than 4 threads.

Setting *S-7* implements the results obtained by the Intel Architecture Code Analyzer [39] which suggests the *PPAA* strategy instead of the *PAPA* strategy of previous settings

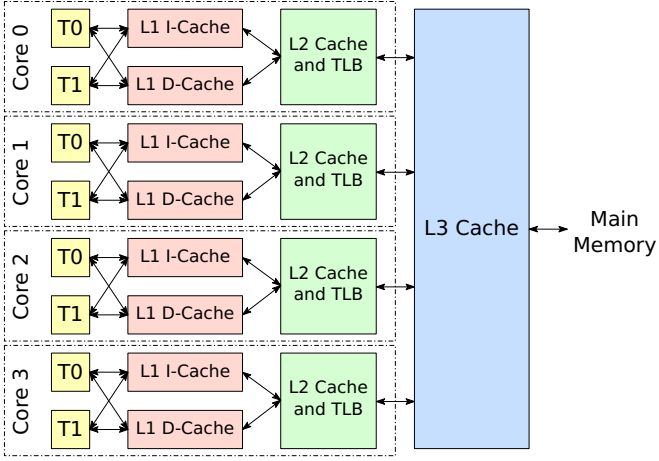


Figure 7: Cache hierarchy of the Intel Core i7-6700 CPU in the employed test system [37].

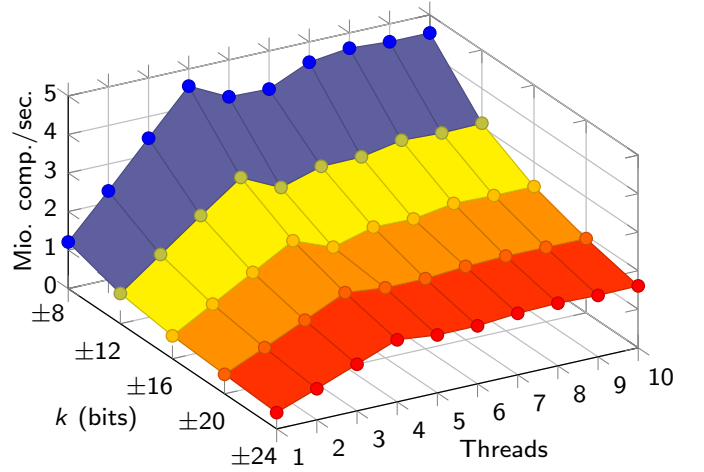


Figure 8: Throughput (in millions of iris-code comparisons per second) in *E-2* in relation to shift size and number of threads.

Step size	<i>TripleA</i>				<i>TripleA-SS</i>			
	LG		QSW		LG		QSW	
	EER	FNMR _{0.01}	EER	FNMR _{0.01}	EER	FNMR _{0.01}	EER	FNMR _{0.01}
2	0.80	1.75	0.74	1.05	0.80	1.75	0.74	1.05
3	0.79	1.75	0.74	1.06	0.80	1.75	0.74	1.06
4	0.80	1.78	0.77	1.14	0.80	1.78	0.77	1.14
5	0.78	1.76	0.77	1.10	0.81	1.78	0.77	1.13
6	0.88	2.70	0.80	1.31	1.09	4.08	0.91	1.70
7	0.82	1.98	1.58	2.07	0.92	2.80	3.91	7.79
8	0.80	1.75	0.89	1.29	0.82	1.84	1.73	5.43

Table 3: EERs and FNMR_{0.01} for different settings of *TripleA* for the LG and QSW feature extraction (LG baseline: EER=0.80%, FNMR_{0.01}=1.75%; QSW baseline: EER=0.74%, FNMR_{0.01}=1.06%).

(see Sect. 4), as shown in Fig. 6. As can be seen in Table 2 and Fig. 5(b), this results in minor speed-up which would be more significant for larger databases. That is, optimising the order of the instruction sequence for the used microarchitecture by static code analyser can still improve the overall performance even in case modern CPUs support out of order execution, which should (in theory) do this automatically.

The presented results are obtained using a Linux operating system. It is important to note that identical performance rates are achieved on other types of operating systems (OSs), since basic memory operations, in particular cache management, is independent of the used OS.

6.3 Accelerated Accuracy-preserving Alignment

For different configurations of *TripleA* using static step-sizes, Table 3 summarizes obtained EERs and FNMR_{0.01}s. For the general approach it can be observed that biometric performance is maintained across most step-size settings. For both feature extractors the *TripleA-SS* approach causes no

Method	Number of threads									
	1	2	3	4	5	6	7	8	9	10
Baseline	84.10	42.24	28.21	21.17	25.52	22.24	22.49	20.89	21.13	21.13
<i>TripleA</i>	34.36	18.04	12.15	9.26	10.06	9.64	9.64	9.03	9.26	9.26
<i>TripleA-SS</i>	30.65	16.40	11.03	8.37	9.65	8.82	8.69	8.26	8.37	8.32

Table 4: Overview of time measurements (in seconds) for different settings in experiments *E-2* performing an identification with $N = 50,000,000$ at 33 shifting positions using $s = 4$.

drastic decrease in accuracy while providing further speed-up as will be shown in the following subsection.

6.4 Simulation of Large Scale Identification

For *E-2* a large scale identification scenario, the best setting PPAA_Q resulting from *E-1* is selected as baseline. Fig. 8 presents the absolute number of iris-code comparisons per second. Again, emphasis should be placed on relative difference in throughput rates of different configurations. Due to the efficient CPU caches the comparisons per second depend on how well the shifted iris-codes fit into the caches and the break even point from 4 to 5 threads, can similarly be observed as in the 1 : N identification scenario, due to 2 threads sharing one cache. Therefore, having 8 threads reveals no significant speed-up over 4 threads. Both setups roughly compare 4.6 million iris-codes per second using ± 8 bit shifts (≈ 80 million comparisons per second without shifting).

Based on the findings depicted in Table 1 and Table 3 further scenarios in *E-2* utilizing *TripleA* and *TripleA-SS* are performed with the parameters $k = 16$, $s = 4$ as step-size and PPAA_Q as core HD score comparator. These experiment results are summarized in Table 4 and depicted in Fig. 9.

From a theoretical standpoint the expected speed-up can be approximated by comparing the number of shifted

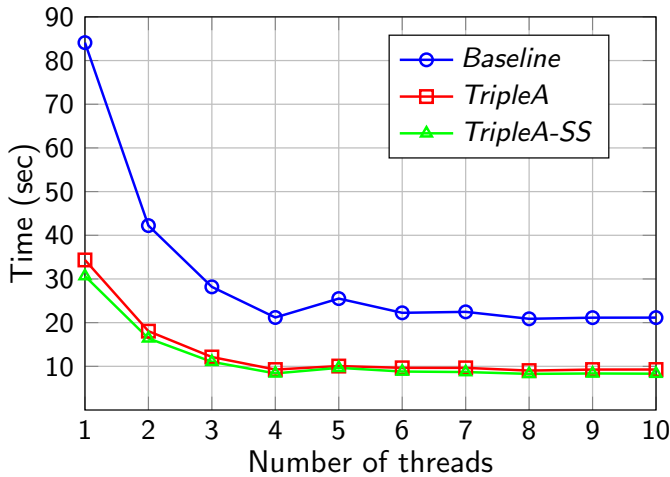


Figure 9: Illustration of time measurements (in seconds) for different settings in experiments *E-2* performing an identification with $N = 50,000,000$ at 33 shifting positions using $s = 4$.

iris-code comparisons to the baseline algorithm $PPAA_Q$. The baseline algorithm has to process all K shifting positions, resulting in 33 comparisons. *TripleA* with the selected parameters does 9 comparisons in Step 1 and in general 6 more in Step 2. In the special case of Step 1 yielding $-k$ or k as result only 3 comparisons are performed in Step 2. This is considered negligible for an approximation and *TripleA* is considered performing 15 comparisons per iris-code. The special case of *TripleA* is the regular case of *TripleA-SS* since only a single side is considered during Step 2. Therefore, the baseline does 33 comparisons, *TripleA* 15 comparisons and *TripleA-SS* 12 comparisons, which results in an approximation of *TripleA* taking 45% and *TripleA-SS* only 36% of the time compared to the baseline. These theoretical considerations match the observed results in Table 4 taking measuring tolerance into account. It means in effect *TripleA* and *TripleA-SS* scale linearly to the number of comparisons relative to the baseline algorithm $PPAA_Q$ and all further k and s combinations can be effectively approximated using the results from *E-2*. Fig. 9 further depicts that *TripleA* and *TripleA-SS* yield no further anomalies that were not present in the $PPAA_Q$ baseline algorithm.

7 Conclusions

In this work we analysed commodity hardware-based iris recognition systems, which perform a CPU-based exhaustive comparison on a large-scale database. We showed that utilising the *POPCNT* hardware instruction can significantly speed up biometric comparisons based on the Hamming distance. We identified that taking the CPU caches into consideration during the algorithm design is the most efficient way to circumvent potential RAM bottlenecks. Especially when making use of multithreading ignoring these caches will lead to bottlenecks and even

make the actual comparison algorithm secondary since the greatest share of time is claimed by the RAM to CPU data transfer and not the actual execution of the algorithm. This observation also impacts the reflection of iris-code comparisons based on GPGPU/CUDA since their speed-up is not only explained due to the high number of cores (hardware shaders), but also the higher memory bandwidth of Video RAM (GDDR) compared to common RAM (DDR). Therefore, GPGPU/CUDA implementations have to deal to a lesser extend with memory bottlenecks. Awareness of cache line sizes on the target system can also greatly improve the data throughput since it maximises cache hits, particular in hotspot loops. Taking into account the aforementioned issues, it is shown that, an optimized conventional CPU-based iris-biometric comparator can achieve a hundredfold speed-up compared to a naïve baseline comparator. As our $1 : N$ results with different shifts sizes show, the number of comparisons alone is no sufficient statement, since the fitting of all shifted iris-code versions into the CPU cache is a high performance factor, independent of the actual algorithm or achieved comparisons per second. Further, our results show that by combining the *TripleA* algorithm with a fast multithreaded *POPCNT* implementation response times of large scale biometric systems can be further decreased, achieving a more than two-hundredfold overall speed-up. Finally, it is important to point out that these findings may also be exploited in other software-based acceleration techniques, e.g. [17].

Acknowledgements

This work was partially supported by the German Federal Ministry of Education and Research (BMBF) as well as by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within the Center for Research in Security and Privacy (CRISP) and the Austrian Science Fund (FWF) project no. P26630.

References

- [1] J. Daugman, "How iris recognition works," *Trans. on Circuits and Systems for Video Technology*, vol. 14, no. 1, 2004. doi: [10.1109/ICIP.2002.1037952](https://doi.org/10.1109/ICIP.2002.1037952) (cit. on pp. 2, 4).
- [2] —, "Information theory and the iriscodes," *Trans. on Information Forensics and Security*, vol. 11, no. 2, 2016 (cit. on p. 2).
- [3] Unique Identification Authority of India, *Aadhaar*: [Http://uidai.gov.in/](http://uidai.gov.in/), retrieved July, 2017. [Online]. Available: <http://uidai.gov.in/> (cit. on p. 2).
- [4] C. Rathgeb, H. Hofbauer, A. Uhl, and C. Busch, "Triplea: accelerated accuracy-preserving alignment for iris-codes," in *Proc. of 9th IAPR/IEEE International Conference on Biometrics (ICB'16)*, 2016, pp. 1–8 (cit. on pp. 2, 5, 6).
- [5] T. Tan, X. Zhang, Z. Sun, and H. Zhang, "Noisy iris image matching by using multiple cues," *Pattern Recognition Letters*, vol. 33, no. 8, pp. 970–977, 2012 (cit. on p. 3).

- [6] N. B. Puhan and N. Sudha, "Coarse indexing of iris database based on iris colour," *Int'l J. on Biometrics*, vol. 3, no. 4, 2011 (cit. on p. 3).
- [7] J. Fu, H. J. Caulfield, S.-M. Yooc, and V. Atluri, "Use of artificial color filtering to improve iris recognition and searching," *Pattern Recognition Letters*, vol. 26, no. 14, 2005 (cit. on p. 3).
- [8] J. E. Tapia and C. A. Perez, "Gender classification based on fusion of different spatial scale features selected by mutual information from histogram of lbp, intensity, and shape," *IEEE Trans. on Information Forensics and Security*, vol. 8, no. 3, pp. 488–499, 2013 (cit. on p. 3).
- [9] M. Erbilek, M. Fairhurst, and M. D. Costa-Abreu, "Improved age prediction from biometric data using multimodal configurations," in *Proc. of 13th Int'l Conf. on Biometrics Special Interest Group (BIOSIG)*, 2014, pp. 1–7 (cit. on p. 3).
- [10] X. Qiu, Z. Sun, and T. Tan, "Global texture analysis of iris images for ethnic classification," in *Proc. 1st Int'l Conf. on Biometrics (ICB)*, 2005, pp. 411–418 (cit. on p. 3).
- [11] H. Zhang, Z. Sun, T. Tan, and J. Wang, "Iris image classification based on hierarchical visual codebook," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, 2014 (cit. on p. 3).
- [12] A. Dantcheva, P. Elia, and A. Ross, "What else does your biometric data reveal? a survey on soft biometrics," *Trans. on Information Forensics and Security*, vol. 11, no. 3, 2016 (cit. on p. 3).
- [13] L. Yu, D. Zhang, K. Wang, and W. Yang, "Coarse iris classification using box-counting to estimate fractal dimensions," *Pattern Recognition*, vol. 38, pp. 1791–1798, 2005 (cit. on p. 3).
- [14] A. Ross and M. S. Sunder, "Block based texture analysis for iris classification and matching," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops (CVPRW)*, 2010, pp. 30–37 (cit. on p. 3).
- [15] P. R. Nalla and K. M. Chalavadi, "Iris classification based on sparse representations using on-line dictionary learning for large-scale de-duplication applications," *SpringerPlus*, vol. 4, pp. 1–10, 2015 (cit. on p. 3).
- [16] J. Gentile, N. Ratha, and J. Connell, "Slic: Short-length iris codes," in *Proc. of 3rd Int'l Conf. on Biometrics: Theory, Applications, and Systems (BTAS)*, 2009 (cit. on pp. 3, 4).
- [17] J. Gentile, N. Ratha, and J. Connell, "An efficient, two-stage iris recognition system," in *Proc. 3rd Int'l Conf. on Biometrics: Theory, Applications, and Systems (BTAS)*, 2009 (cit. on pp. 3, 10).
- [18] M. Konrad, H. Stögner, A. Uhl, and P. Wild, "Computationally efficient serial combination of rotation-invariant and rotation compensating iris recognition algorithms," in *Proc. Int'l Conf. on Computer Vision Theory and Applications (VISIGRAPP)*, 2010, ISBN: 978-989-674-028-3. DOI: [10.5220/0002821100850090](https://doi.org/10.5220/0002821100850090) (cit. on p. 3).
- [19] F. Hao, J. Daugman, and P. Zielinski, "A fast search algorithm for a large fuzzy database," *Trans. on Information Forensics and Security*, vol. 3, no. 2, 2008 (cit. on p. 3).
- [20] C. Rathgeb, F. Breitingner, H. Baier, and C. Busch, "Towards bloom filter-based indexing of iris biometric data," in *Proc. 8th Int'l Conf. on Biometrics (ICB)*, 2015 (cit. on p. 3).
- [21] R. Mukherjee and A. Ross, "Indexing iris images.," in *Proc. Int'l Conf. on Pattern Recognition (ICPR)*, 2008 (cit. on p. 3).
- [22] R. Gadde, D. Adjero, and A. Ross, "Indexing iris images using the burrows-wheeler transform," in *Proc. Int'l Workshop on Information Forensics and Security (WIFS)*, 2010 (cit. on p. 3).
- [23] H. Proença, "Iris biometrics: Indexing and retrieving heavily degraded data," *Trans. on Information Forensics and Security*, vol. 8, no. 12, 2013 (cit. on p. 3).
- [24] R. Rakvic, B. Ullis, R. Broussard, R. Ives, and N. Steiner, "Parallelizing iris recognition," *Trans. on Information Forensics and Security*, vol. 4, no. 4, 2009 (cit. on p. 3).
- [25] N. Vandal and M. Savvides, "CUDA accelerated iris template matching on graphics processing units (GPUs)," in *Proc 4th Int'l Conf. on Biometrics: Theory Applications and Systems (BTAS)*, 2010 (cit. on p. 3).
- [26] M. López, J. Daugman, and E. Cantó, "Hardware-software co-design of an iris recognition algorithm," *IET Information Security*, vol. 5, no. 1, pp. 60–68, 2011 (cit. on p. 3).
- [27] R. N. Rakvic, H. Ngo, R. P. Broussard, and R. W. Ives, "Comparing an FPGA to a cell for an image processing application," *EURASIP J. Adv. Signal Process*, vol. 2010, 2010. DOI: [10.1155/2010/764838](https://doi.org/10.1155/2010/764838) (cit. on p. 3).
- [28] L. Masek, "Recognition of human iris patterns for biometric identification," Master's thesis, Univ. of Western Australia, 2003 (cit. on p. 4).
- [29] L. Ma, T. Tan, Y. Wang, and D. Zhang, "Efficient iris recognition by characterizing key local variations," *Trans. on Image Processing*, vol. 13, no. 6, 2004. DOI: [10.1109/TIP.2004.827237](https://doi.org/10.1109/TIP.2004.827237) (cit. on p. 4).
- [30] *USIT - University of Salzburg iris toolkit*, <http://www.wavelab.at/sources/Rathgeb16a>, Version 2.0.x (cit. on p. 4).
- [31] C. Rathgeb, A. Uhl, and P. Wild, *Iris Recognition: From Segmentation to Template Security*, ser. Advances in Information Security. Springer Verlag, 2013, vol. 59 (cit. on p. 4).
- [32] Chinese Academy of Sciences' Institute of Automation, *Casia iris image database v4.0 — interval*, <http://biometrics.idealtest.org>, 2012 (cit. on p. 6).
- [33] V. Sklyarov, I. Skliarova, and J. Silva, "On-chip reconfigurable hardware accelerators for popcount computations," *International Journal of Reconfigurable Computing*, vol. 2016, p. 11, 2016 (cit. on p. 6).
- [34] W. Mula, *Ssse3: fast popcount*, <http://wm.ite.pl/articles/sse-popcount.html>, accessed March 2016, 2008 (cit. on p. 8).
- [35] *JC-42.3, Ddr4 sdram standard - jesd79-4a*, 2013. [Online]. Available: <https://www.jedec.org/standards-documents/docs/jesd79-4a> (cit. on p. 8).
- [36] I. S. Haque, V. S. Pande, and W. P. Walters, "Anatomy of high-performance 2d similarity calculations," *J. of Chemical Information and Modeling*, vol. 51, no. 9, 2011 (cit. on p. 8).
- [37] Intel Corporation, *Intel 64 and ia-32 architectures optimization reference manual*, 2016 (cit. on pp. 8, 9).
- [38] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-threading technology architecture and microarchitecture," *Intel Technology J.*, no. Q1, 2002 (cit. on p. 8).
- [39] I. Hirsh and Intel, *Intel architecture code analyzer*, <https://software.intel.com/en-us/articles/intel-architecture-code-analyzer>, 2012 (cit. on p. 8).