Workload Characteristics for Iris Matching Algorithm: A Case Study

Jed Kao-Tung Chang Fang Hua Gildo Torres Chen Liu Stephanie Schuckers
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, NY 13699 USA
{jchang, huaf, torresg, cliu, sschucke}@clarkson.edu

***Abstract*—With the rapidly expanded biometric data collected by various sectors of government and industry for identification and verification purposes, how to manage and process such Big Data draws great concern. Even though modern processors are equipped with more cores and memory capacity, it still requires careful design in order to utilize the hardware resource effectively and the power consumption efficiently. This research addresses this issue by investigating the workload characteristics of biometric application. Taking Daugman's iris matching algorithm, which has been proven to be the most reliable iris matching method, as a case study, we conduct performance profiling and binary instrumentation on the benchmark to capture its execution behavior. The results show that data loading and memory access incurs great performance overhead and motivates us to move the biometrics computation to high-performance architecture.**

Keywords-Biometrics; Iris Matching; Workload Profiling; Binary Instrumentation

I. INTRODUCTION

The noticeable increase in the utilization of biometrics has dramatically expanded the amount of biometric data collected and examined everyday in health care, employment, law enforcement and security systems. There can be up to peta-bytes of biometric data from hundreds of millions of identities that need to be accessed in real-time. For example, the Department of Homeland Security (DHS) IDENT Database holds 110 million identities and enrolls or verifies over 125,000 individuals per day in 2010 [1]. India has their UID program with over 100 million people to date and expected to cover more than 1 billion individuals [23]. One major challenge that has been reported by U.S. Government in advancing biometrics is the operational capabilities of managing and analyzing large-scale biometric information in an effective and efficient manner [22]. In 2012, Sussman *et al.* suggested the foundation for the next-generation biometric systems being "expandable, scalable and flexible to accommodate new technologies and biometric standards, as well as interoperable with existing systems" [1].

Therefore, it is of great interest and importance to develop more efficient and effective large-scale biometric identity management systems. In related work, Booz Allen Hamilton Inc. (a Strategy and Technology Consulting Firm) is leveraging the big-data biometrics to the cloud [2][3]. They made an assessment to predict the growth rate of large scale database in FBI and measured the effectiveness of modern biometric matching algorithms. Then they would create a test database, run on the cloud computing platform and document the gains from the solution. NEC just announced a cloud computing service for facial recognition in the Japanese market [4]. The facial recognition service allows the merchant to estimate the age and gender of the client, along with their shopping favorites. This can help the merchant adjust the marketing strategy. AFIS and Biometrics Consulting Inc. suggested a solution for large-scale cloud-based biometric identification systems using clusters [5]. They borrowed techniques from famous search engine companies such as Google and Amazon. Data and calculations can be broken up into small datasets and tasks and distributed through the networks. In addition, they designed a data replication file system to improve the system reliability. The databases are designed to be flat and minimal indexed. Wang *et al*. investigated a reliable scheme for fingerprint authentication in enhanced cloud security [6]. The scheme is based on a secret-splitting concept in which the password-based login process and fingerprint matching process are executed on the cloud computing environment. It is robust against attacks such as man-in-the-middle, dictionary, and replay attack. Vallabhu *et al*. provided a conceptual framework for the Biometrics on Cloud [7]. In this framework, a large database stores users' fingerprint and voice data, which are fused together and encrypted. As a user tries to log in, the biometric authentication service performs the verification. Once authenticated, the user will be directed to the cloud computing service. Chang *et al.* [8] introduced a fast access control applied to Hadoop cloud computing for fingerprint identification and facial recognition. This system accomplished rapid face and fingerprint identification in 2.2 seconds when conducting cross-examination, accordingly verified the effectiveness in access control for cloud computing.

This paper provides an analysis of execution behavior of biometrics, such as the program trace, memory access ratio, *etc.*, to facilitate the thorough understanding of the workload characteristics of such application. In addition, the future architectural solution should consider emerging multi-core processors, cluster computing and high-performance computing architecture based on big data processing techniques. While operational systems have clearly demonstrated the capacity for large data processing, few related approaches have been proposed. Study of the tradeoffs in system design can be useful for future developers of large scale systems, particularly, when considering emerging technologies.

We will perform the workload characterization on a selected biometric application as a case study. The benchmark of choice is the iris matching algorithm, one stage of Daugman's iris recognition algorithm [16]. This algorithm compares two samples represented by two 2D bit matrices by performing Hamming Distance calculation. The result in the form of matching score shows the similarity of the compared samples. We study the behavior of the iris matching algorithm via performance profiling and binary instrumentation. The purpose of performance profiling is to identify the part of the program that takes the most of the total execution time, which forms the performance bottleneck. The identified functions and/or operations, which we call "hotspot functions" or "hot-blocks", are suitable for hardware acceleration. Binary instrumentation observes the program at the assembly instruction level. Instrumentation code injected during the program execution reveals the major instructions employed by the program, the memory access overhead in terms of the ratio of instructions with memory read or write, and execution behavior of certain instruction pattern. We study the benchmark using one customized instruction pattern called "Load-Store Block". The knowledge on the instruction mix and memory-access ratio of the iris matching algorithm can help us for future instruction set extension design.

The rest of the paper is organized as follows: in Section II we will describe the related work; Section III introduces the iris matching algorithm and experimental methodology; the results and discussion are presented in Section IV; finally, in Section V conclusions and future work are described.

II. Related Work

Iris recognition normally requires real-time accessing and processing, which is very computation-intensive. Previous researches focused on accelerating from both the software and hardware sides. If purely improved the algorithm from a software point of view and running on a general-purpose processor, however, the efficiency would be less than implementing on dedicated hardware with strong computability and large degree of hardware resources. Two representative types of hardware for accelerating iris recognition are digital signal processor (DSP) and field-programmable gate array (FPGA).

DSP is used by Miyazawa *et al.* [9-11] for iris recognition hardware acceleration. They proposed 2D Discrete Fourier Transforms (DFT) technology to improve image alignment and matching score calculation as part of the iris recognition algorithm. The improved algorithm was then implemented into DSP. This implementation has the advantage of low error rate, real-time execution, and low hardware cost.

FPGA prototyping has also been exploited for hardware acceleration of iris recognition algorithm. For example, in order to accelerate the iris matching algorithm, the large datasets, which is the iris template database, can be stored in Read-Only Memory (ROM), while the live templates, which are the incoming samples, can be stored in Random-Access Memory (RAM). The algorithm itself can be implemented using hardware description language (HDL). Based on this concept, more advanced optimization techniques on FPGA are suggested.

For example, Rakvic *et al.* [12] parallelized the code for the iris segmentation, template creation, and template matching stages and implemented them on FPGA. Their implementation achieved speedups of 9.6X, 324X, and 19X, separately, compared with the state-of-art CPU with moderate hardware cost. Mohd-Yasin *et al.* [21] employed neural network concepts into FPGA prototyping in order to enhance the iris recognition accuracy and efficiency. The back propagation and training process enhanced the accuracy rate in recognizing iris samples.

Different from the above works, which accelerate certain parts considered to be performance bottleneck, Raida *et al.* [13] performed dynamic-based workload profiling for the iris recognition application prior to hardware implementation. The profiling identified the computation-expensive parts as Gabor filter, Gaussian masque, Canny, and Hough transform. They implement these parts into hardware using three approaches. The C2H approach is to implement the Hough transform hardware by C2H compiler [14] and add it to the system. The single-accelerator approach is to implement the morphologies operations into hardware IP and add it to the system using one accelerator counterpart, while the double-accelerator approach is to implement the gauss filter plus the morphologies operations into hardware IPs and add them to the system. Finally, they compared the power consumption, hardware cost, and performance of the three approaches. C2H method can save most design time and power consumption, while the double-accelerator approach has the highest performance.

Different from the work of Raida *et al.* [13], we conduct dynamic profiling on the final stage of the iris recognition, the iris matching, and demonstrate that the iris matching spent significant amount of time on loading the samples into memory and hamming distance calculation. Besides the performance profiling, we also conduct the binary instrumentation to observe the execution behavior at the instruction-level granularity. By monitoring the instruction mix and instruction with memory read/write, we find that the data-transfer-type instruction and memory access downgrade the performance. Based on the heavy memory access overhead, we propose the instruction pattern called Load-Store Block (LSB) and perform the instrumentation to see how frequently this instruction pattern occurs in the iris matching algorithm. By implementing LSBs into hardware in the form of memory module using processor-in-memory (PIM) architecture, the memory overhead can be significantly reduced.

III. Proposed Scheme

The iris recognition method we employed in this study was originally proposed by Daugman [16]. It mainly consists of four stages: segmentation, normalization, feature encoding, and matching. In this work we only focus on the matching stage for the investigation and performance enhancement.

A. Iris Matching Algorithm

The iris matching algorithm compares two samples based on the matrix Hamming Distance calculation. The calculated matching score is between 0 and 1, where lower score indicates that the two samples are more similar.

Each iris sample is represented with two 2-dimensional matrices of bits. The size of the matrix is 20 rows by 480 columns. The first matrix is the template representing the iris pattern. The second matrix is the mask matrix representing aspects that should not be considered in the matching, such as eyelid, eyelash, and noise (e.g., spectacular reflections) which may mask parts of the iris. The template matrix is computed by the following steps. First, iris region is segmented from a near-IR image of an eye and transformed from a circular image into a rectangular image. Next, Gabor filtering is used to extract the most discriminating information from iris pattern as a matrix.

Algorithm 1 Iris_Matching Algorithm

```
template1, mask1, template2, mask2;
{Input 2-D Matrix with ROW rows and COL columns}
Hamming_Distance;                {Return value}
template1s, mask1s, temp, mask, result;   {Intermediate 2D Matrix}
hd=0, hd_min=1;                  {Intermediate variable}

for shift=-8 to 8 do
    template1s ← Rotate_Column(template1, shift);
    mask1s ← Rotate_Column(mask1, shift);
    temp←Matrix_XOR(template1s, template2);
          mask←Matrix_OR(mask1s, mask2);
    result←Matrix_AND(temp,mask);

    hd ← (number of 1s in Matrix result) / (ROWxCOL –number
                                            of 1s in Matrix temp)
    if hd < hd_min then
        hd_min ← hd
    end if
 end for
 Hamming_Distance ← hd_min;
 return Hamming_Distance;
```

Algorithm 2 Rotate_Column

```
{Two Input Parameters: Matrix MAT and variable shift}
{Return Matrix MAT}
if shift ≤ 0 then
   rotate left MAT for 2 × abs(shift) columns;
else
   rotate right MAT for 2 × abs(shift) columns;
end if
return MAT;
```

Algorithm 1 shows Daugman's iris matching algorithm. Algorithm 1 calls Rotate Column function which is described in Algorithm 2. Matrix_XOR, Matrix_OR, and Matrix_AND are all functions performing bit-wise logical operations of the two matrices. The main body of the algorithm is a 17-round for-loop structure with the index *shift* from -8 to 8. Suppose the matrices of the first sample are named as *template1* and *mask1*, and the matrices of the second sample are named as *template2* and *mask2,* respectively. In each round, first, *template1* and *mask1* are left-rotated if shift is less than 0 or right-rotated if shift is larger than 0 for 2 × *abs*(*shift*) columns to form two intermediate matrices, named *template1s* and *mask1s*. Next, we XOR *template1s* with *template2* into *temp* and OR *mask1s* with *mask2* into *mask*. Finally, the matrix *temp* will be ANDed with the matrix *mask* to form the matrix *result*. So the Hamming Distance for this round is calculated by dividing the number of 1s of the matrix *result* by the total number of the 2D matrix entries (which is 20 × 480) minus the number of 1s in matrix *temp*. The minimum-valued Hamming Distance of all 17 iterations is the calculated Hamming Distance, which is returned as the matching score of the two samples, showing their similarity.

B. Proposed Scheme

We study the workload characterization of iris matching based on the hotspot point detection and binary instrumentation. The hotspot point is the part of a benchmark that is identified as the most time-consuming, which might be a function, a basic block, or several lines of code. The binary instrumentation consists of injecting code dynamically to collect statistical data about the program running behavior, such as instruction count and program trace. In this research we will study three workload characteristics of iris matching algorithm: the instruction mix, memory access rate, and the program behavior of an instruction pattern we call Load-Store Block (LSB), which will be explained later. The steps of the performance analysis are shown below:

1. Porting Work: The source code written in Matlab is translated into C. We tested the correctness and validity against the original results obtained with Matlab.
2. Hotspot Detection Work: We employed Intel VTune Amplifier [17] to detect the performance-intensive parts of the iris matching algorithm. VTune Amplifier analyzes the software performance on IA-32 and Intel64-based machines and the performance data is displayed in an interactive view. At the granularity of function level, VTune Amplifier breaks down the total execution time and shows the time spent on each function, so the function consuming much of execution time is identified as the "hotspot function". At a finer granularity level, it can also show the time spent on a single line of the source code inside a function. Thus, we can identify the performance-intensive function (hotspot function) or lines of code (hot-block) in the iris matching algorithm.
3. Binary Instrumentation Work: We use PIN [18] as the binary instrumentation tool. We are interested in observing three aspects of the execution behavior: instruction mix, memory access ratio, and load-store block.
    a. Instruction mix: The x86 Assembly Language Reference Manual [19] classifies IA-32 instruction set into several categories based on their functionalities. Information about the majority of the instruction class can provide us with the basis for the instruction-set extension design.

       The iris matching benchmark is compiled into IA-32 assembly instructions. One hundred and one distinct IA-32 instructions are used. We classify them into seven classes, which are Binary Arithmetic instructions (BA), Bit and Byte instructions (BB), Control Transfer instructions (CT), Data Transfer instructions (DT), Logical instructions (L), Shift and Rotate instructions (SR), and Miscellaneous instructions (M).

       Please note that I/O and Segment Register instructions are not included since they are not used by the benchmark. The Flag Control instructions and String instructions do not appear frequently and have similar

features to the Control Transfer and Data Transfer instruction class, so they are merged into these two classes, respectively. We use PIN to measure the ratio of the time of each instruction class to the total execution time.

b. Memory access ratio: We are interested in the percentage of instructions that are related to memory access since in biometric applications a large amount of data needs to be transmitted on- and off-chip for calculation. And off-chip transmission is the performance bottleneck as the "Memory Wall" problem [15] is becoming more serious. The memory access ratio can demonstrate the memory access overhead.

   Again, we employ PIN to instrument the percentage of instructions with memory read and write. Since IA-32 is of Complex Instruction Set Computer (CISC) Instruction Set Architecture (ISA), some instructions will have both memory read and write operations. Thus, the ratio of memory access is the sum of instructions with only memory read and with only memory write, minus those with both.

c. Load-Store Block: If an application has heavy memory access, this means it must have many load and store instructions. We propose an instruction pattern called Load-Store Block (LSB) [20]. LSB is a pair of load and store instructions pointing to the same effective address with some instructions between the pair. If there are a lot of LSBs in an application, we can perform the hardware acceleration by implementing the LSB into the memory module in the form of a memory-SIMD hybrid architecture. Thus, data in LSB can be accessed and executed in the local module and multiple data can be processed by many modules at the same time. Thus, the performance of the total application can be enhanced considerably.

   We instrument the LSB of iris matching algorithm in two aspects: firstly, the percentage of instructions that falls into LSB; secondly, the size of LSB dominating the execution of the benchmark is large or small.

   The first aspect is obvious: the higher the LSB percentage, the higher probability it can be accelerated. We measure it by the metric, *LSB_Perc*, which is defined as:

$$LSB\_Perc = \sum_{i=0}^{N} \frac{(i+2) \times Occr(i)}{IC} \quad (1)$$

   where i is the size of LSB, *e.g.*, the number of instructions inside Load and Store instruction pair; *Occr(i)* is the number of occurrences of the LSB of size i; and *IC* would be the total instruction count. Note that we use *(i+2)* because we need to include the starting Load and ending Store instructions for the instruction count.

   The second aspect is also important. If most of the LSBs are small, then they can be implemented into a smaller memory module, and more modules can be implemented compared to the LSB with large size. Therefore a higher degree of data parallelism can be achieved. Thus, we use PIN to record the frequency of LSB of different sizes, and then calculate the ratio of various sizes of LSB in iris matching algorithm.

Table I summarizes the experimental environment for the performance profiling.

TABLE I. EXPERIMENTAL ENVIRONMENT

CPU	Intel Xeon CPU E5-2630 2.30 GHz 6-core
I-Cache and D-Cache	192KB each (per core)
L2 Cache	1.5MB
L3 Cache	15MB
Memory Size	16GB
Operating System	Windows 7 64-bit for VTune UBuntu 10.04.4 on Oracle VM VirtualBox
Compiler	For VTune: Visual C++ For PIN: gcc version 4.4.7
Binary Instrumentation Tool	PIN 2.12
Performance Analyzer	VTune AmplifierXE 2013

IV. EXPERIMENT AND DISCUSSION

The data used in this study is generated from Q-FIRE dataset [24] and Q-FIRE II Unconstrained dataset, containing 616 iris images from 40 subjects. The iris matching process is to match one iris sample against the entire dataset, resulting in a total of 189420 comparisons. In this section, we will demonstrate the results of the performance analysis.

The VTune Amplifier performs the function breakdown in order to identify the software performance bottleneck. As we mentioned in previous sections, in our iris matching benchmark each sample needs to be compared against all the other samples in the database so the *main* function will have a two-level nested loop for comparison. Each iteration will call the *ReadFile* function to load two samples (template and mask) for comparison and then call the *GetHammingDistance* function (Algorithm 1) to calculate the matching score. The *GetHammingDistance* calls the *shiftbits* function (Algorithm 2: Rotate_Column).

TABLE II. PERCENTAGE OF EXECUTION TIME OF THE IRIS MATCHING FUNCTIONS

Function Name	Percentage of Total Execution Time
main	0.52%
ReadFile	86.53%
GetHammingDistance	9.65%
shiftbits	3.17%
Others	~0.13%

Table II lists the functions of iris matching algorithm along with their consumed percentage of total execution time. Note that the time recorded for each function does not include the time spent on sub-function calls. For example, the time for function *GetHammingDistance* does not include the time spent on function *shiftbits*. The results show that *ReadFile* function, which loads the samples into memory, accounts for majority of the execution time, meaning that the data loading work from disk is the performance bottleneck of the matching algorithm.

The second and the third heaviest would be the *GetHammingDistance* and the *shiftbits* functions, which perform the matching score calculation.

When we examine the code at the instruction-level granularity, VTune Amplifier shows that the logical OR, XOR, and AND operations between the matrices inside the *GetHammingDistance* and the data movements inside *shiftbits* are the most time-consuming operations.

TABLE III. INSTRUCTION CLASS FREQUENCIES IN THE IRIS MATCHING BENCHMARK

	BA	BB	CT	DT	L	SR	M
Perc	20.86%	5.00%	13.00%	50.67%	1.40%	9.00%	1.00%

For the instruction mix profiling, Table III shows the results. We can see that Data Transfer instruction class accounts for more than half of the execution time at 50.67%. Binary Arithmetic, Control Transfer, and Shift and Rotation classes account for 20.86%, 13.00%, and 9.00% of the total execution time, respectively. This can be explained from the algorithm which contains lots of addition, comparison, and column-shifting operations. If we want to design instruction set extension, we can focus on Data Transfer and Binary Arithmetic classes as the candidate for acceleration.

TABLE IV. THE PERCENTAGE OF MEMORY READ AND WRITE INSTRUCTIONS

	Memory Read	Memory Write	Memory Read / Write	Total Memory Access
Percentage	33.47%	10.02%	3.78%	39.71%

Table IV shows the percentage of the memory read and memory write instructions over the total instruction count. The total memory access rate is 39.71%. This demonstrates that memory access is the major overhead that downgrades the overall performance.

The results of LSB instrumentation on iris matching benchmark is shown in the upper half of Table V. The percentage of LSB instructions (*LSB_Perc*) accounts for 9.4% of total instructions, which is not that significant as we would have expected. LSB(0), LSB(2), LSB(3), and LSB(17) predominate over the LSBs of other sizes. The ratios of LSB(3) and LSB(17) to total LSB instructions are 46.5% and 44.77%, respectively. The occurrences of other LSB sizes are so small that we omit them.

TABLE V. LSB INSTRUCTION INSTRUMENTATION

Benchmark	*LSB_Perc*	LSB(0)	LSB(2)	LSB(3)	LSB(17)
Whole Iris Matching Algorithm	9.40%	1.20%	7.40%	46.50%	44.77%
Hamming Distance	9.19%	<0.01%	<0.01%	<0.01%	>99.99%

Knowing that the data loading causes the performance burden, we also perform the LSB instrumentation on *GetHamming Distance* and *shiftbits* functions to focus on the Hamming Distance calculation. The results are shown in the bottom half of Table V. The *LSB_Perc* is even lower around 9.19%. The LSB of size 17 dominates the LSB instructions.

Overall, the profiling results from VTune Amplifier and PIN instrumentation on instruction mix and memory access ratio shows that data loading from disk to memory and data transfer between on- and off-chip are the critical parts to be accelerated in the iris matching benchmark. The problem will worsen in the cloud computing environment where a larger amount of data would need to be accessed and executed for biometric applications in real-time. In addition, the biometric matching template format varies in complexity. For example, representation of the matrix's entry will be changed from integer binary to decimal point numbers, which requires a much larger storage space and needs higher memory bandwidth to remain performance-efficient.

Since there are always huge amount of samples to be compared for iris matching, we can parallelize and port the iris matching on high-performance many-core architecture. The many-core architecture usually has tens or hundreds of CPU cores in which each core can execute the assigned tasks independently. We can assign one core as the master core in charge of distributing the comparison tasks among all cores and retaining the result, and other cores as computation cores. Each computation core independently finishes the assigned comparisons, gets the lowest matching score along with the sample pairs, and sends it back to the master core. Upon receiving the results from all the computation cores, the master core can determine the most similar sample pairs. Thus, the performance can be enhanced because the comparison work is processed in parallel by the cores.

V. CONCLUSION

Biometrics provides a way for secure verification and identification. Accessing and processing large-scale biometric datasets in real-time while achieving power and energy efficiency, however, can pose a great challenge. This research conducts workload characterization on iris matching algorithm as a case study to identify the computation-intensive parts by means of dynamic-based profiling and binary instrumentation.

The contributions of this paper can be summarized as follows:

1. We profiled the iris matching algorithm and identified that loading the samples from dataset is the performance bottleneck.
2. We analyzed the instruction mix and found that Data Transfer and Binary Arithmetic class account for the majority of the instruction mix. This will provide an aid in designing instruction set extension for hardware acceleration.
3. The memory access ratio demonstrates that memory access accounts for majority of the performance overhead. The results of heavy overhead on Data Transfer Class hence memory access also show the necessity of shifting to high-performance architectures with many-core and large memory bandwidth.
4. We also instrument the Load-Store Block (LSB) instruction pattern behavior on the iris matching algorithm.

The future work includes extending our methodology of performance profiling to other stages of the iris recognition algorithm, as well as other biometric applications, such as fingerprint and facial recognition, which may contain more complicated matching process that worth further investigation. First, we will profile and instrument on other biometric applications to study their instruction mix and memory access behavior. Additionally, we will study and instrument more instruction patterns other than LSB for hardware acceleration. The final goal is to accelerate the biometrics via hardware implementation. Based on the profiling result, we will port the biometrics onto the emerging many-core high-performance machine as the solution architecture and observe the performance enhancement.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grants No. ECCS-1301953, IIP-1068055, and IIP-1332046. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] A. Sussman, "Biometrics and Cloud Computing," presented at the Biometrics Consortium Conference 2012, 19-Sep-2012.

[2] "Leveraging the Cloud for Big Data Biometrics." [Online]. Available: http://www.boozallen.com/insights/ideas/booz-allen-ideas-festival/winning-ideas/ideas-cloud-biometrics.

[3] E. Kohlwey, A. Sussman, J. Trost, and A. Maurer, "Leveraging the Cloud for Big Data Biometrics: Meeting the Performance Requirements of the Next Generation Biometric Systems," in *2011 IEEE World Congress on Services (SERVICES)*, 2011, pp. 597 –601.

[4] "NEC outs cloud computing facial recognition service for merchants," 12-Nov-2012. [Online]. Available: http://www.engadget.com/2012/11/12/nec-outs-880-facial-recognition-system/.

[5] "Biometric Identification on Cloud Computing." [Online]. Available: http://www.afisandbiometrics.com/biometric_identification_on_cloud_computing.

[6] J. Yang and N. Poh, *Recent Application in Biometrics*. InTech, 2011.

[7] H. Vallabhu and R. V. Satyanarayana, "Biometric Authentication as a Service on Cloud: Novel Solution," *Int. J. Soft Comput.*, vol. 2.

[8] B. R. Chang, H.-F. Tsai, C.-M. Chen, and C.-F. Huang, "Access Control of Cloud Computing Using Rapid Face and Fingerprint Identification," in *2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA)*, 2011, pp. 179 –182.

[9] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and A. Katsumata, "An iris recognition system using phase-based image matching," in Image Processing, 2006 IEEE International Conference on, 2006, pp. 325–328.

[10] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and H. Nakajima, "An efficient iris recognition algorithm using phase-based image matching," in Image Processing, 2005. ICIP 2005. IEEE International Conference on, vol. 2, 2005, pp. II–49–52.

[11] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and H. Nakajim, "A phasebased iris recognition algorithm," in Proceedings of the 2006 international conference on Advances in Biometrics, ser. ICB'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 356–365.

[12] R. N. Rakvic, B. J. Ulis, R. P. Broussard, R. W. Ives, and N. Steiner, "Parallelizing iris recognition," Information Forensics and Security, IEEE Transactions on, vol. 4, no. 4, pp. 812–823, 2009.

[13] H. Raida and M. A. YassineAoudni, "Hwn sw implementation of iris recognition algorithm in the fpga," International Journal of Engineering Science, vol. 4, 2012.

[14] Altera, "C2h user guide," http://www.altera.com/.

[15] J.L. Hennessy and D.A. Patterson, "Computer Architecture: a Quantitative Approach," Morgan-Kaufman, San Mateo, CA, 1990.

[16] J. G. Daugman, "High confidence visual recognition of persons by a test of statistical independence," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 15, no. 11, pp. 1148–1161, 1993.

[17] Intel® VTune™ Amplifier XE 2013 http://software.intel.com/en-us/intel-vtune-amplifier-xe

[18] PIN: a dynamic binary instrumentation tool. http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[19] x86 Assembly Language Reference Manual. http://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf

[20] J. Chang, C. Liu, S. Liu, and J-L. Gaudiot, "Workload Characterization of Cryptography Algorithms for Hardware Acceleration," in Proceedings of the Second ACM International Conference on Performance Engineering (ICPE 2011), Karlsruhe, Germany, March 14-16, 2011.

[21] F. Mohd-Yasin, A. Tan, and M. Reaz, "The fpga prototyping of iris recognition for biometric identification employing neural network," in Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on. IEEE, 2004, pp. 458–461.

[22] National Science and Technology Council Subcommittee on Biometrics and Identity Management, "The National Biometrics Chanllenge", Spetember, 2011

[23] https://en.wikipedia.org/wiki/Unique_Identification_Authority_of_India#cite_note-NPRCard-16

[24] P. A. Johnson, P. Lopez-Meyer, N. Sazonova, F. Hua, and S. Schuckers, "Quality in face and iris research ensemble (Q-FIRE)," in 2010 Fourth IEEE International Conference on Biometrics: Theory Applications and Systems (BTAS), 2010, pp. 1–6.