Multi-core Approach towards Efficient Biometric Cryptosystems

Charles McGuffey Chen Liu
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, NY 13699, USA
e-mail: {mcguffcj, cliu}@clarkson.edu

***Abstract*—Protecting data is a critical part of life in the modern world. The science of protecting data, known as cryptography, makes use of secret keys to encrypt data in a format that is not easily decipherable. However, most modern cryptography systems use passwords to perform user authentication. These passwords are a weak link in the security chain, and are a common point of attack on cryptography schemes. One alternative to password usage is biometrics: using a person's physical characteristics to verify whom the person is and unlock the data correspondingly. This study focuses on the Cambridge biometric cryptosystem, a system for performing user authentication based on a user's iris data. We converted our implementation of this algorithm from a single-core system to a system that can run on multiple cores. The experiment takes place on an Intel Single Chip Cloud Computer (SCC), an experimental processor created by Intel Labs. A design pattern has been created for the parallelization of software functions using multiple cores. Using this design pattern, two functions in the system were accelerated. The system generated by this acceleration process produced limited computational speedup, but provided results that can be applied to a variety of hardware acceleration problems.** *(Abstract)*

Keywords- Biometric Cryptosystem; Hadamard Code; Hardware Acceleration; Multi-Core; Iris Recognition; Reed-Solomon Code (key words)

I. INTRODUCTION

In the modern world, data is becoming more prolific, and much of it is digitally accessible [1]. This widespread availability of data leads to sensitive information becoming vulnerable, meaning that it must be protected. One way for this protection to happen is through cryptography, the science of encoding or decoding information. Cryptography works by taking data to be protected, known as a "secret", and performing a transformation on that data to make it appear as though the data is a random sequence of bits or characters. This transformation is known as "encryption". Intended users of the data will be able to make sense of the random sequence, or to "decrypt" the data, through a methodology known only to them. Other parties that do not have access to the decryption instructions will not be able to make sense out of the data. If done correctly, converting the random sequence of bits back to the real meaningful data with no knowledge of the decryption key should be computationally complex, meaning that it cannot be achieved with current computing technology within a reasonable amount of time.

Many current cryptography systems make use of passwords for verifying user authenticity. However, passwords are not ideal in that they are prone to being lost, stolen, or forgotten. To overcome the weakness of traditional password-based design, many research groups have investigated the use of biometrics, which are unique physical or behavioral characteristics of people, to protect cryptographic keys. Several researchers have developed algorithms using a biometric measurement for cryptography systems in a secure and accurate manner. These implementations, called biometric cryptosystems, combine biometrics and cryptography in a manner that allows the matching of biometrics in an encrypted, secure domain [2].

In order to take advantage of biometric cryptosystem design, the algorithm needs to be able to run at a speed that is effective for use in commercial products. For most applications, this means calculations and cryptography must occur in real-time. This study attempts to achieve that goal by performing multi-core acceleration of a biometric cryptosystem algorithm.

Hardware acceleration allows a biometric cryptosystem algorithm to be executed more quickly by providing a hardware platform that can more easily handle the computation. For each function that was accelerated, the single core software code was replaced with software making use of multiple CPU cores, allowing multiple computations to be run simultaneously, which improves the speed of the function relative to running on the single-core processor only. This performance increase often significantly outweighs the extra power consumed by the additional core(s), reducing the total energy consumed by the system for the given task.

The performance improvement presented in this study provides a template for making biometric cryptosystems more viable in an industrial setting, where they can provide secure access to data without passwords. In addition to the practical benefits of cryptography, this study provides insights on the strengths and weaknesses of the hardware acceleration process. Analysis of what target function characteristics lead to the greatest efficiency gain provides a framework for determining the most efficient use of

hardware acceleration. This will also allow more effective usage of hardware acceleration in other applications, including those beyond the scope of biometric cryptosystems.

The rest of the paper is organized as follows: a brief overview of biometric cryptosystems is provided in Section II; Section III discusses the reference biometric cryptosystem design in detail; hardware acceleration methods and benefits are discussed in Section IV; Section V provides details on the experimental setup and procedure; results and associated discussion are provided in Section VI and final conclusions are drawn in Section VII.

II. BIOMETRIC CRYPTOSYSTEMS

Traditional cryptography methods make use of password protection and cryptographic keys to guard valuable information from attackers. The key is used to encrypt or decrypt data as necessary and is usually long and difficult to guess or memorize. This has led to keys being released based on password authentication, which allows users to memorize a relatively short password while providing data encryption that is still strong enough to resist attackers [3]. However, passwords themselves are a relatively vulnerable medium. Passwords are often poorly chosen, allowing them to be compromised through intelligent guessing or brute force attacks [3]. Additionally, passwords are often recorded on unsecure mediums, such as unencrypted files or pieces of paper that are left lying around. If the password is not recorded, it must be memorized, causing the existence of password recovery services, which take additional time and provide additional potential for security flaws. These weaknesses have caused a significant amount of research into alternatives to passwords.

One alternative method to the utilization of passwords is the biometric cryptosystem. A biometric cryptosystem takes data about a particular feature of each user, called a "biometric", and records that data. Biometrics are defined as behavioral or physiological characteristics of a user. Examples of biometrics include fingerprints, iris scans, facial patterns, hand geometry, signatures, and keystrokes. Biometric systems enroll users by storing information about the users' biometrics. If a user attempts to gain access to the system under protection, the user's biometric is checked. If the biometric data matches an entry in the system that has the appropriate privileges, the user is granted access. If the biometric data does not match, or matches an entry without the required privileges, then the user is denied access to the system. This eliminates the need for passwords and the security risks associated with their usage.

There are several issues associated with the use of biometrics prior to their application to cryptography. In order for a biometric to be useful, it must be able to be collected and accurately compared to other data collections. The difficulty of biometric collection is usually due to the requirement of a specific hardware setup, and is therefore irrelevant to our discussion. Accurate comparison, however, is a relevant issue. When biometric collection occurs, there is a high variance in the data collected due to differing positions of the subject of the collection, variance in the machine, noise, and other issues [2]. Additionally, biometric data can change due to human growth, injury, habit changes, and other real-world events [2]. A biometric system must be able to handle these data variances in order to be effective. Complicating matters, the analysis system must also be able to distinguish between the biometric data of different people. Biometric systems must seek to minimize the false acceptance rate (FAR), where two different people are identified as the same person, and the false rejection rate (FRR), where one person is not identified as himself/herself on separate data collections [3]. This dual constraint poses a significant challenge to biometric systems even prior to their application in cryptography.

There are also problems that arise specifically when biometrics are applied to cryptography. The first of these issues is privacy. If biometric templates are stored unencrypted on the system, attackers could potentially steal the biometric data of users of the system [4]. This has privacy implications that cause the system to be unreasonable from both ethical and economic standpoints. In addition to these issues, the scenario of a specific account being compromised must be taken into consideration. It is important that this scenario would not result in any other system that uses the same biometric being compromised. Revocability, or the ability to generate multiple secure identities for a user, is a requirement used to prevent a security breach from permanently compromising a user's access to a system. This prevents the use of the raw biometric data, and instead necessitates using a transformation that is not based exclusively on the biometric data. However, working with biometric templates post-transformation provides significant challenges in coming up with alignment processes necessary to deal with the variances and tolerances that must be accounted for as discussed above [5].

A large amount of research has been performed attempting to solve these problems [5]. Many of them failed to achieve their goal or produced a result under unrealistic assumptions or with limited applications. However, some research results do seem promising. One of these is an algorithm using iris biometrics through an error correction method proposed by Anderson et al. from a research group in Cambridge University [6]. This algorithm is referred to herein as the "Cambridge biometric cryptosystem", which we will discuss in detail next.

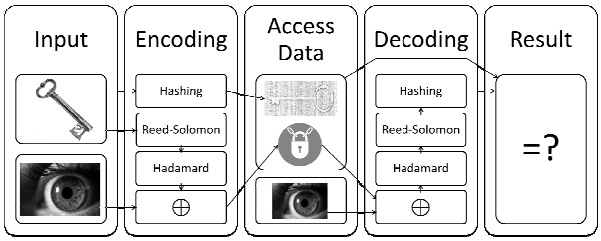

Figure 1 Cambridge Biometric Cryptosystem

III. THE CAMBRIDGE BIOMETRIC CRYPTOSYSTEM

The Cambridge biometric cryptosystem is an algorithm that confirms user authenticity through the use of an iris template. Each user in the system is enrolled by providing a 256-byte iris template and receiving a randomly generated 140-bit key. These two inputs are used to generate two variables that are stored on a physical token that the user receives. The first of the generated variables is a hash of the original 140-bit key. This hash function is a mapping function that is used to obscure the original 140-bit key. The second variable, called a locked template, is the result of performing an exclusive-or (XOR) function between the enrollment template and the result of putting the randomly generated key through Reed-Solomon and Hadamard encoding sequentially [6] [7]. When a user attempts to gain access to the system, they provide an iris sample and their physical token. The locked template is XORed with user's sample template, producing the encoded key with errors introduced by the differences between the enrollment template and the sample template. This result is then put through Hadamard decoding, followed by Reed-Solomon decoding. If the person attempting to access the system is a valid user with the correct token, the result of the decoding will be the original key. If someone is trying to access the system using someone else's token, the result will be different. The key is hashed after calculation using the same hash method used in the enrollment process, and then compared to the hashed result stored on the token. If they are the same, the user is deemed valid and given access. If the results are different, then the user is treated as an imposter and is not given access to the system. A block diagram of this cryptosystem is shown in Fig. 1.

The Cambridge biometric cryptosystem makes use of Reed-Solomon error-correction code to handle burst errors in the iris template. These are large errors that affect many contiguous bits of an iris template. Burst errors are often caused by eyelashes blocking views of the iris or other devices that cause significant error in the picture. Reed-Solomon code handles these errors by dividing the input into several blocks. The blocks are interpreted as coefficients to a polynomial using the domain of a Galois field. Additional parity blocks are added to create a resulting polynomial that is evenly divisible by the defining polynomial of the Galois field. During the decoding process, the division is performed on the input blocks, and the remainder is used to locate blocks that are in error. These error blocks are then entirely recalculated based on the results. This means the number of errors in a particular block does not matter, only the number of blocks that are in error. For the Cambridge biometric cryptosystem, the 140-bit input is divided into 20 blocks of 7 bits. 12 parity blocks are added to create a result of 32 blocks of 7 bits. This allows the system to correct up to 6 block errors for any given encoding and decoding process.

Hadamard coding is used by the Cambridge biometric cryptosystem to handle random errors in the iris template. These are small errors that may be caused by transmission errors, or may be related to minor changes in the image of the iris taken. Hadamard coding is based on the concept of the Hadamard matrix, a matrix containing positive and negative values with a magnitude of one, where the values are arranged in a particular order. The initial Hadamard matrix, of order one, is shown in Equation 1.

$$H_1 = \begin{bmatrix} + & + \\ + & - \end{bmatrix} \tag{1}$$

Subsequent Hadamard matrices are computed as shown in Equation 2.

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \tag{2}$$

A Hadamard code uses a matrix that differs from a Hadamard matrix in two ways. The first is that the additive inverse of the Hadamard matrix is stacked below the original matrix as shown in Equation 3. The second is that all negative ones in the Hadamard matrix are replaced by zeros.

$$H_c = \begin{pmatrix} H \\ -H \end{pmatrix} \tag{3}$$

Hadamard encoding interprets its input as a series of blocks. Each block is treated as a row index. The output result of the block is the entirety of the row corresponding to that index. During the decoding process, each block is compared to every row of the Hadamard matrix, and the row index corresponding to the row with the least bit differences from the input block is the output. This allows the system to return the correct output if the number of bit errors is less than half the minimum distance between rows of the matrix. For the Cambridge biometric cryptosystem, the input consists of 32 7-bit blocks. A block size of seven means that there are 2 ^ 7 = 128 rows in the matrix. Since the matrix used is constructed from two square matrices stacked on top

of each other, this means that each row consists of 64 bits. Thus the resulting output is 32 64-bit blocks. This choice allows each block to contain up to 15-bit errors before becoming a burst error.

The Cambridge biometric cryptosystem appears to successfully fulfill the requirements of a strong and safe biometric cryptosystem. According to tests performed by Anderson et al., the system performed with a 99.5% correct match rate [6]. The algorithm answers security concerns by storing only modified versions of the key and iris template, protecting these from being accessed by intruders. In the event that a user's access data is compromised, a new random key can be generated for that user, resetting the system without loss of functionality. This combination of accuracy and security makes the Cambridge biometric cryptosystem a strong choice for software implementation and subsequent hardware acceleration.

IV. Hardware Acceleration

Since the development of the computer, people have been trying to make computation faster. This was initially achieved through increasing the speed of Central Processing Units (CPUs) that handle a single stream of instructions. But recently, CPUs have started to reach a level where their performance cannot be increased without generating more heat than can be dissipated in a reasonable amount of time and space [8]. This has caused the growth of research into new areas of computational acceleration.

New methods of computation focus on two aspects: running multiple instruction sequences (known as "threads") simultaneously, and decreasing the execution time of certain instructions. These goals are usually accomplished through the use of multiple CPU cores running in parallel, a graphical processing unit (GPU), or custom hardware. Using multiple CPU cores allows the execution of multiple threads simultaneously on one CPU. This makes it effective in scenarios where programs have many sequences of instructions that are not dependent on one another. However, the fact that the hardware in the CPU is not being modified, just made more plentiful, means that instructions are not executed any faster. This means that programs without opportunity for parallel computation do not benefit from a multi-core hardware architecture. Similarly, making use of a GPU allows significantly increased parallelism by performing multiple calculations simultaneously. In addition, GPUs are optimized to perform certain types of calculations commonly used in graphics processing, allowing faster execution time for these instructions. The disadvantage of GPUs is that they are optimized for particular types of calculations, reducing the benefits they provide for computation of a more general type. Alternatively, custom hardware allows particular hardware units to be defined to handle certain computations [9]. This means that systems can be custom designed to optimize for the most common form of computation in the target application. For custom hardware, the amount of parallelism in computation is defined by the amount of hardware space allocated by the designer for additional hardware for each particular functionality. The disadvantage of this method is that custom hardware must be designed specifically for the system it is intended for, as opposed to off-the-shelf CPUs or GPUs. This takes significant amounts of design time, and also requires the use of customizable hardware resources, whether a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD). These hardware devices consist of a variety of hardware resources that can be configured in varying manners to produce the custom hardware.

Hardware acceleration also changes energy efficiency in addition to processing speed. A system using hardware acceleration has more hardware to power than the same system without hardware acceleration, causing an increase in power consumed. However, the energy usage also depends on the time, which typically decreases during the process of hardware acceleration [10]. Depending on the rates of change in time and power consumption, total energy usage per computation can increase, remain the same, or decrease after hardware acceleration occurs.

This project makes use of the Intel Single-chip Cloud Computer (SCC) for the implementation and testing of the biometric cryptosystem. The Single-chip Cloud Computer experimental processor [11] is a 48-core concept vehicle created by Intel Labs as a platform for multi-core software research. It contains 48 Pentium class IA-32 cores on a 2D mesh network. The chip provides hardware that allows communication between the cores, in addition to a shared memory model that cores can use via off-chip DRAM. These communication capabilities in combination with the fact that inter-core communication does not require changing chips makes this platform ideal for testing multi-core acceleration. Multi-core acceleration was chosen due to its increasing availability in the computing environment. Commercially available computers are transitioning towards greater numbers of cores, making this acceleration method something that could be adopted relatively easily. The fact that the SCC contains 48 cores allows the scalability of the algorithms to be tested.

V. Methodology

The process used to generate a multi-core implementation of the Cambridge Biometric Cryptosystem was broken up into several steps. The first step of this process was to port the initial C implementation of the system over to the SCC hardware setup. Once this step had been completed, it was necessary to design a core-scalable method for data sharing and communication between the cores involved in the

process. After the inter-core communication setup had been decided upon, the algorithms for multi-core computation were designed and implemented. The final step in the process was to update the software system so that it could run under a variety of operating voltages and frequencies. This process produced a working multi-core implementation of the Cambridge Biometric Cryptosystem.

The process of performing multi-core acceleration began by adapting the existing software to run on the Intel SCC. This involved adapting the software to make use of the Intel RCCE library, which handles system initialization, hardware interfacing, and inter-core communication [12]. Porting the initial software implementation to this platform involved adjusting the code entry and exit sections to handle the appropriate initialization and conclusion steps. After the completion of this process, the software could be run on the SCC. However, each core would perform the exact same steps, duplicating computations. Adding a check to the program to ensure that only the core designated as the lead core would perform computation fixed this issue and generated a software system compatible with the target hardware platform.

Once a compatible software project was completed, a methodology for data transmission and synchronization needed to be designed. Since the majority of computations for this system were confined to one core, this core would have a significantly different software stack than the cores designated as helper cores. This means that the cores could not simply run the same function with the same arguments and perform the computation appropriately. The problem was solved using a design pattern being referred to as the controller-helper design pattern.

The controller-helper is a software design pattern intended to help solve problems involving multiple cores performing computation on data originating from a single core. Fig. 2 shows a system where the results are returned to the core the data originated from, but the pattern could be extended to return data to a different core. The core dealing with the input and output data is known as the controller. The other cores participating in the computation are referred to as helpers.

The pattern involves three functions, the controller function, the helper function, and the worker function. At the start of the function being accelerated, the controller calls the controller function and the helpers call the helper function. Both of these functions attempt to allocate a shared memory pointer for each input and output argument. This is done using the *RCCE_shmalloc* function, which allocates an area of shared memory and returns each core a pointer to the same memory location [12]. The controller then places the input data in the shared memory, making it accessible to all of the cores. Once each core has gained access to this data, the situation has become similar to a traditional multi-core problem. From this point, both the controller and the helpers call the worker function. The worker function takes pointers to the input arguments to the original function as its input arguments. At this point the multi-core algorithm can divide up jobs as it deems fit without having to worry about securing input data. Each worker function deposits its output results into the shared memory pointer created for the output data. Once a worker function finishes its computations, it returns. After all computations are complete, the controller function reads the results of the computation from the shared memory into local memory. Upon completion of this data transfer, each core uses an *RCCE_shfree* command to free the shared memory [12]. At this stage, the helper functions have completed their work, and the controller function can pass the data to its caller. This design pattern can be generalized to any other multi-core setup by replacing the *RCCE_shmalloc* and *RCCE_shfree* calls with the appropriate calls for allocating and freeing shared memory, respectively. A diagram displaying this design pattern is shown above in Fig. 2.

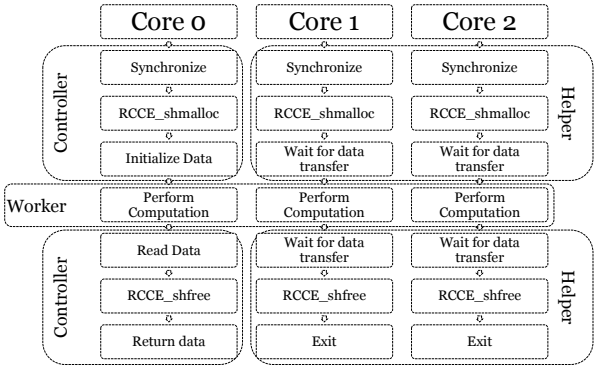

Figure 2 Controller-Helper Design Pattern

Once a memory management model had been created, attention was turned to the individual function to be profiled. The target functions, *mult_polys* and *hadamard_decode*, were chosen based on the results of profiling the original software implementation. These functions took approximately 45 and 20 percent of the total system runtime, respectively.

The *mult_polys* function performed polynomial multiplication across a Galois field. This process is similar to polynomial multiplication, where individual terms are multiplied, and then terms of the same order are gathered and added together. The difference with this function is that the multiplication operation is replaced with multiplication across a Galois field, and like term gathering makes use of the XOR operation rather than addition. This function takes two arrays of 24 integers as input, and returns one 48 integer array as an output. The multi-core algorithm for this function asked each core to solve for a certain set of the output terms. This involved performing Galois multiplication for each pair of inputs that resided in different input arrays with indices that added up to the index of the

output being computed. After each computation, the result was XORed with the total. Thus each core computed some number of terms of the resultant polynomial. This algorithm allowed evenly distributed simultaneous computation without conflicting writes to the shared memory locations. Since input accesses were read-only, they did not cause data dependency. This algorithm was completed and tested to ensure that it performed to specification.

The second function targeted for acceleration was the *hadamard_decode* function. This function takes 32 blocks containing 64 bits each as its input. Each of these blocks is converted to a seven-bit block as described in Section II above. Similar to the process used for the *mult_polys*, this function was accelerated by assigning each core a group of output blocks to decode. The core will compare the input block to each row of the Hadamard matrix, and will put the index number of the row with the least bitwise differences in the output location. This function has also been implemented and tested to ensure its effectiveness.

The software system generated was designed to allow the user to choose whether they wanted the system to use either or both of the accelerators programmed. This was completed by adding additional runtime parameters. These parameters were checked whenever an accelerated function would be called. On one input, the accelerated function would be called, while on another the original function would be called. This configuration permits the user to control how they would like to perform their computation and allows for easier runtime testing.

The final step in the process used to generate a multi-core system that takes advantage of the functionality of the SCC was adding voltage and frequency control. These were also added as additional parameters provided at function runtime. The RCCE library functions were used to change the core operating voltage and frequency to the desired values prior to performing computation and return the system to its original state upon the completion of the program.

VI. Results and Discussion

The methodology used in this project resulted in a software implementation of the Cambridge Biometric Cryptosystem that functions on the Intel SCC. This software system can make use of four possible computational methods and several different voltage and frequency settings.

In order to test the effectiveness of the hardware acceleration, the system was profiled using the default operating characteristics of 1.1 Volts and 533 megahertz. Tests were performed on the system making use of no hardware acceleration, each accelerated function individually, and using both accelerated functions. Each configuration was tested using 1, 2, 4, 8, and 16 cores on the SCC for enrollment, verification, and the two processes sequentially. Due to issues with the hardware, the system was unable to be tested using larger numbers of cores. Each test was performed three times and the average runtime between the tests was used for the calculation.

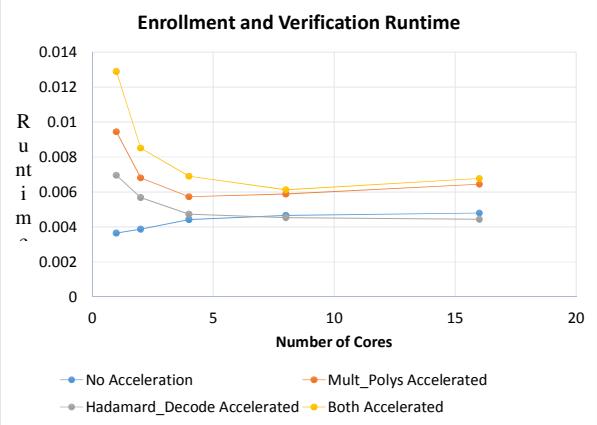

Figure 3 Enrollment and Verification Runtimes at 533 MHz

The results of the testing when running enrollment and verification sequentially are shown in Fig. 3. This graph shows that the best runtime for the system with these operating characteristics occurs when the non-accelerated system is run using one core. It is interesting to note that the runtime of the non-accelerated version increases with the increased number of cores. This is likely due to the synchronization constraints that were placed on the system to handle the multi-core algorithms. Not all of these synchronization points are disabled when using the non-accelerated version of the system. The increased number of cores requires additional coordination messages leading to increased latency for the system.

It is also important to note that the runtime of the accelerated versions of the system generally compare poorly to the non-accelerated versions. In all instances except the *hadamard_decode* using 8 or 16 cores the accelerated function is actually slower than the non-accelerated version using the same number of cores. The accelerated function in these particular situations has a better runtime than the non-accelerated system using the same number of cores but worse than the single core version. This means that this acceleration cycle is having a small positive effect on the runtime, but that the coordination costs overall outweigh this benefit.

The general results discussed above are confirmed by the graphs of enrollment and verification run individually in Fig. 4 and Fig. 5, respectively.

Fig. 4 provides two important takeaways. The first is that the runtime of the *hadamard_decode* accelerated enrollment is roughly the same as the non-accelerated version of

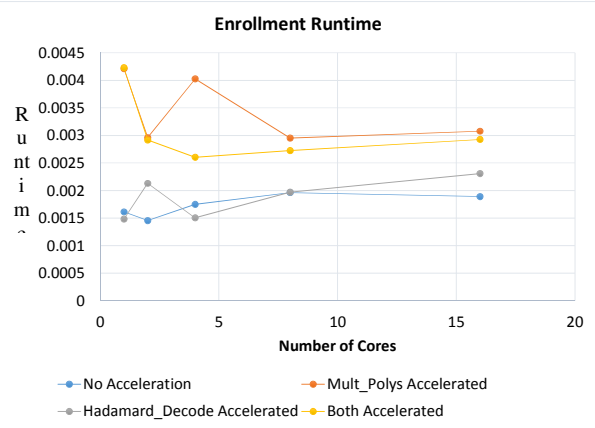

Figure 4 Enrollment Runtimes at 533 MHz

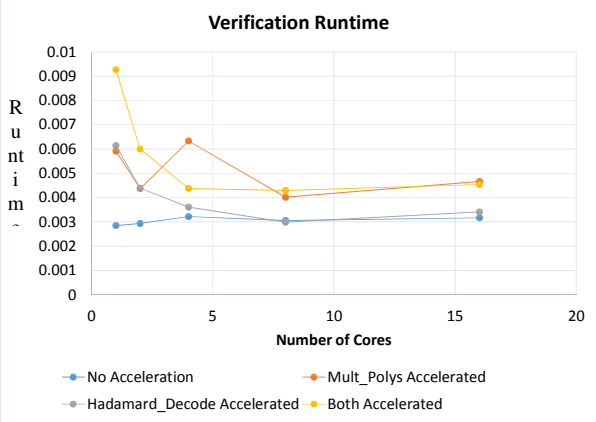

Figure 5 Verification Runtimes at 533 MHz

enrollment. Since the enrollment process does not call the *hadamard_decode* function, this is expected. The second concept to notice from this graph is the noise. There are significant spikes in the runtimes recorded that do not appear to agree with the general trend of the graph. These are likely due to variations in the system at runtime.

Fig. 5 shows the runtimes of the various systems when performing verification. This graph provides evidence that appears to confirm the conclusions drawn previously: the high levels of noise, the ineffectiveness of the *mult_polys* acceleration, and the limited effectiveness of the *hadamard_decode* acceleration.

In addition to performing tests at the default operating characteristics, a small subset of tests were performed at a higher operating frequency of 875 megahertz. The tests performed at this frequency were the same as the tests detailed above, but the algorithm was only run using enrollment and verification run sequentially. The results of these tests are shown in Fig. 6. The hardware acceleration shows more promise in these results. The *mult_polys* function still has higher runtimes than the non-accelerated system. However, the accelerated *hadamard_decode* functionality using 8 or 16 cores has the lowest total runtime, eclipsing the original system run with one core. The runtime drops from approximately 1.93 milliseconds to approximately 1.64 milliseconds, a speedup of about 1.18.

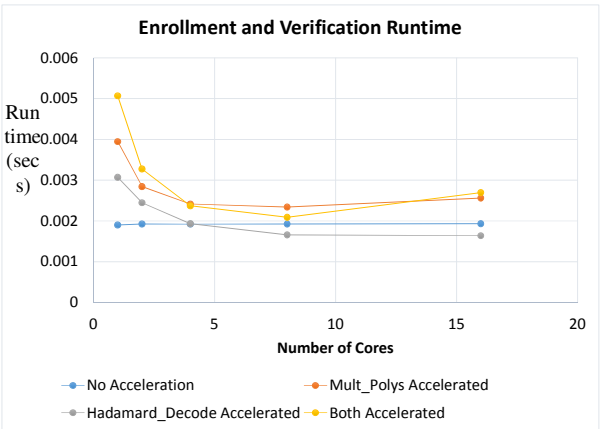

Figure 6 Enrollment and Verification Runtimes at 875 MHz

The results of testing done on this system indicate that the Cambridge biometric cryptosystem is likely not computationally intensive enough to gain significant benefits from multi-core acceleration. However, even on this system, one function was found where acceleration proved effective to a certain degree. Based on the algorithm design, we believe that this performance improvement will scale to 32 cores, at which point there become as many cores as total outputs to be processed.

VII. Conclusion

Improving the efficiency of biometric cryptography allows it to more easily become part of modern life. This is an important step towards creating a world where data can be protected efficiently. Biometric cryptosystems provide a secure method of providing user security without the drawbacks of passwords. The biometric cryptosystem developed by Anderson, Daugman, and Hao at the University of Cambridge provides a strong algorithm for a biometric cryptosystem that overcomes many of the challenges of biometric cryptography while providing security and accuracy.

Combining hardware acceleration with software optimization allows for an increased improvement in computation. Hardware acceleration works well for functions that have long series of computations, especially when many of the calculations are independent of one another. This allows the increased parallelism that leads to better performance improvements. However, there are situations, such as less computationally intensive and less parallel functions, where the communication required to use multiple cores is less than ideal. Knowing how and when to apply this approach allows system designers to generate systems with much better efficiency than a naive approach.

In this work, a multi-core software implementation of the Cambridge biometric cryptosystem has been implemented and tested. The resulting system has varied performance, achieving a speedup of 1.18X using one acceleration method at 875 megahertz. This same acceleration method at 533 MHz, and the other acceleration method result in an increase in the overall runtime of the system. The result is a system that can perform user enrollment and verification in 1.64 milliseconds on an Intel Single-chip Cloud Computer.

In addition to providing a concrete implementation of a multi-core biometric cryptosystem, this project provides a process that can be used as a guideline for future applications of hardware acceleration to biometric cryptosystems. Making use of the controller-helper design pattern generated will allow for the development of systems that take significantly less runtime and possibly less energy per computation, significantly increasing their useful potential. This could also be compared to the results of porting this algorithm onto GPU and FPGA platforms in order to assess the potential of these varying platforms.

ACKNOWLEDGMENT

The authors would like to thank Gildo Torres at Clarkson University. The knowledge, helpfulness, and energy that Mr. Torres displayed in support of this project was beyond the call of duty, and integral to the success of the resulting system. This work is supported by the National Science Foundation under Grant Numbers IIP-1332046, IIP-1068055, ECCS-1301953. 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] Hey, T., Tansley, S., Tolle, K., [The fourth paradigm, data-intensive scientific discovery], Microsoft Research, Redmond, WA.

[2] Jain, A.; Ross, A., Uludag, U., "Biometric template security: challenges and solutions," European Signal Processing Conference, (2005).

[3] Uludag, U.; Pankanti, S.; Prabhakar, S.; Jain, A.K., "Biometric cryptosystems: issues and challenges," Proceedings of the IEEE, vol.92 (6), pp.948-960 (2004).

[4] Ross, A.; Othman, A., "Visual cryptography for biometric privacy," IEEE Transactions on Information Forensics and Security, vol. 6 (1), pp. 70-81 (2011).

[5] Rathgeb, C., Uhl, A., "A survey on biometric cryptosystems and cancelable biometrics," Journal on Information Security, vol. 2011, pp. 3 (2011).

[6] Anderson, R., Daugman, J, Hao, F., "Combining cryptography with biometrics effectively," Technical Report 640, University of Cambridge, 2005.

[7] Clark, C., "Reed-Solomon error correction," British Broadcasting Company Research and Development White Paper, WHP 031, July 2012.

[8] Liu, C., Duarte, R., Granados, O., Tang, J., Liu, S., Andrian, J., "Critical path based hardware acceleration for cryptosystems," International Journal of Advancements in Computing Technology, vol. 4 (1), pp.438-452 (2012).

[9] Lau D.; Blackburn J., Jenkins, C., "Using c-to-hardware acceleration in FPGAs for waveform baseband processing," Software Defined Radio Technical Conf. Product Exposition, (2006).

[10] Liu, C., Granados, O., Duarte, R., Andrian, J., "Energy efficient architecture using hardware acceleration for software defined radio components," Journal of Information Processing Systems, vol. 8 (1), pp. 133-144 (2012).

[11] Mattson, T., van der Wijngaart, R., Riepen, M., Lehnig, T., Brett, P., Haas, W., Kennedy, P., Howard, J., Vangal, S., Borkar, N., Ruhl, G., and Dighe, S., "The 48-core scc processor: the programmer's view," in High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for, Nov 2010, pp. 1–11.

[12] Mattson, T., van der Wijngaart, R., "RCCE: a small library for many-core communication," Intel Labs, Jan 2011.