SOLUTIONS FOR SPEEDING-UP ON-LINE DYNAMIC SIGNATURE AUTHENTICATION

Valentin Andrei, Sorin Mircea Rusu, Ştefan Diaconescu
Research & Development Department, SOFTWIN S.R.L., Sos. Pipera-Tunari, Bucharest, Romania
vandrei@softwin.ro, srusu@softwin.ro, sdiaconescu@softwin.ro

Keywords: on-line authentication, biometrics, dynamic signature, FPGA processing, GPU computing, Levenshtein algorithm, distributed computing

Abstract: The article presents a study and its experimental results, over methods of speeding-up authentication of the dynamic handwritten signature, in an on-line authentication system. We describe 3 solutions, which use parallel computing by choosing a 16 processor server, a FPGA development board and a graphics card, designed with nVidia CUDA technology. For each solution, we detail how can we integrate it into an authentication provider system, and we specify it-s advantages and disadvantages.

1 INTRODUCTION

The evolution of biometry in the last few years was predictable due to the increasing amount of sensible data, like bank accounts information or new technologies documentation, which needed to be stored into databases, safe from any attack attempt. The Internet held more and more valuable information so the need for high security kept increasing.

Biometry appeared relatively recently and used unique characteristics of a person in order to secure and authenticate his actions. Some of these characteristics are: the iris, the fingerprint, the signature, the voice, the face anatomy, etc. One of the most non-intrusive methods of biometric authentication is based on using the dynamic handwritten signature. The term does not define only the image drawn on a piece of paper, but it refers mainly to the movement of the owner-s hand. The image can be copied but the movement of the hand is almost impossible to be reproduced. This characteristic belongs to behavioural biometrics because a person changes his way of signing over the years. Due to this particularity, the offered security level is very high.

In the last 2 years, we have focused our research towards using dynamic signature in the purpose of on-line authentication (Marcu, 2009). We have built a web-service, capable of securing internet applications, which need authentication. It can be used in every web-application that provides authentication services, as an extra-security layer. The problem that appears is providing a short response time when the server is being overloaded with numerous requests. In this article we will describe 3 solutions that can help us solve the problem and to offer a short authentication time for a reasonable number of clients, accessing the service simultaneously.

2 AUTHENTICATION

In order to verify if a signature is genuine, some processing needs to be done. First of all the user needs to input 5 signatures, that will be considered specimens and every new signature will be compared to them.

An electronic device will be used to capture the signature. This electronic device is an intelligent pen, that contains 2 MEMS accelerometers and an optical navigation system (ONS) having the ability to extract the user-s hand movement and to transmit it via USB port. The pen transmits at 1000 Hz sampling rate, the hand acceleration values on 2 axes and the data extracted by the ONS. Inside the PC, the raw signals can be stored into CSV files for analysis.

These files contain a fair amount of redundant data therefore, in order to make the process more

efficient we need to apply a compression method. For this reason we have developed a set of signature recognition algorithms (SRA), which build a set of invariants, from the raw signals. The following picture represents the signature processing. The electronic pen captures the hand movement, transmitting the raw signals through the driver, to the SRA block.

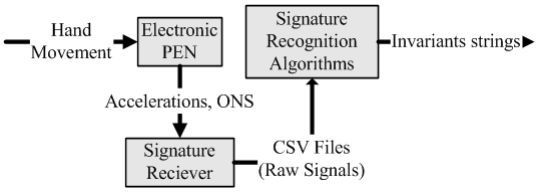

Figure 1: Signature processing steps.

In order to say if a signature is authentic, we must compute a distance between the invariants extracted from the given signature and the ones extracted from the user-s specimens. The distance is computed by using the Levenshtein algorithm.

After that, the resulted distance, is passed to a classifier that will give the final answer over the originality of the signature (Marcu, 2009). This classifier can be a threshold based decision system, a neural network, etc. The next picture shows the dynamic signature authentication process.

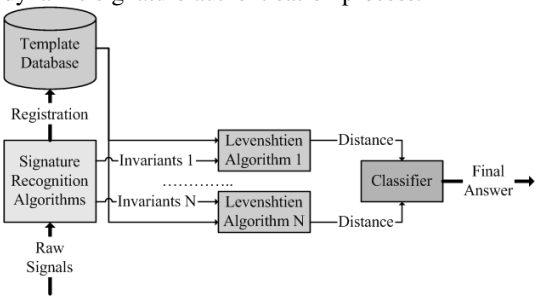

Figure 2: Signature authentication process.

As we mentioned before, we have built a web-service, capable of providing biometric authentication based on the dynamic handwritten signature. The acquired signature is being sent to the web-service, where comparison with the stored specimens is made. The service just sends the response, telling if the given signature is original or false. If the signature is genuine, the user receives access to his account. Given this context, a proportion of the authentication time belongs to the transfer operation, between the client and the web-service. However we are interested in accelerating the most time consuming operation, of the whole system.

The following diagram shows the main operations being made, in order to offer the client access to his account data based on his dynamic signature.

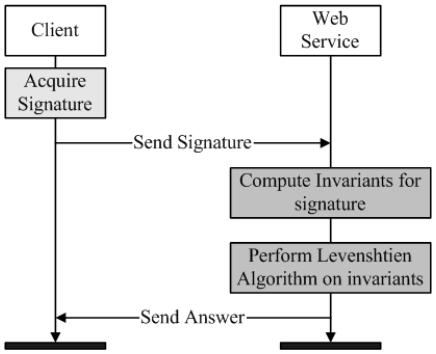

Figure 3: On-line signature authentication operations.

We have measured the time consumed by each operation, in order to find the component whose function needs to be optimized. The following table shows percentages of the authentication time, consumed by every system block.

Table 1: Time consumed by each operation of the authentication process.

Operation	Time (% of total)
Signature acquisition	< 1%
Transfer time	< 2%
Invariants computation	< 5%
Distances computation	> 92%

As we can see from the previous table, distance computation's time has the greatest proportion. Therefore we need to find a solution in order to compute the distances as fast as we can, to provide the client a reasonable authentication time.

The Levenshtein algorithm is based on the following formula:

$$D[i, j] = \text{Minimum}(D[i-1, j] + \text{Deletion Cost}, D[i, j+1] + \text{Insertion Cost}, D[i-1, j-1] + \text{Substitution Cost}) \quad (1)$$

It involves the usage of a matrix of N x M size where N and M are the lengths of the strings being compared. The distance between the 2 strings is at D[N, M] cell inside the matrix.

The parallelizing possibilities of a single algorithm instance are poor. However parallelizing is possible by using systolic arrays (Hoang, 1993) but at increased difficulty cost. If the length of the strings is large, then the parallelization of the algorithm is worth being implemented. If it-s small,

having more distances computed in parallel by several processing units, has more advantages.

We have chosen to distribute distance computing tasks into several processing units. We have studied the possibility of using the following:

- 16 processor powerful server;
- FPGA development board;
- Video card using nVidia CUDA technology.

3 ACCELERATING

As said before, we need to start several instances of the Levenshtein algorithm in multiple processing units. We focused our work just to study how we can speed-up the distance computing block, so this is the reason we have built a block that generates strings of symbols, to be passed as input data to the processing units. We have measured the time it took for the whole block of data to be processed and we compared the 3 solutions. The following figure gives an overview upon the evaluating procedure.

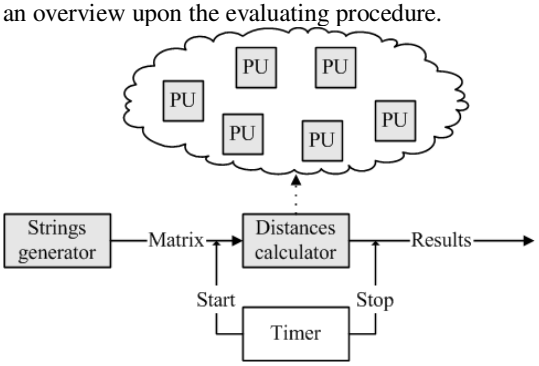

Figure 4: Time evaluation procedure.

The strings generator builds a queue of string pairs. Each pair will serve as input for an instance of Levenshtein algorithm. This queue will be a symbol matrix of (2 x N) x M cells where N represents the number of distances that need to be computed, and M represents the length of the strings being compared. In the real system we provide a mechanism of queuing for the authentication requests. The request queue is very similar to the matrix generated by the strings generator so the estimations we will make based on the results of the 3 solutions give, will be close to truth.

3.1 Speeding-up the authentication process using a 16 processor server

The goal is to use the server hardware capabilities at full power, in order to achieve the best time. We have done this by starting a number of threads in our main program and by assigning them a high priority among the tasks of the operating system.

First of all we needed to discover the optimum thread number in order to minimize the overhead. For that we have computed 2048 distances between strings of 750 symbols on 32 bits each. We have distributed these tasks to a variable number of threads and measured the computing time. If the number of threads is too small, then the computing power of the server is not used at maximum, which will result in a time increase. If the number of threads is too high, then the generated overhead will also result in a time increase. The average computing time for a variable number of threads is centralized in the following table.

Table 2: Choosing the optimum thread number.

Number of threads	Duration (milliseconds)
8	2281
14	1328
16	1412
32	1515

This behaviour was predictable because the server has 16 processors and the optimum thread number should be near 16. The resulted number of threads that would give best results is 14. In this case 14 of the system-s processors will be used at full power and the rest of 2 processors will be used for the vital tasks of the operating system.

In order to obtain a short response time, the processor and memory frequency, should be as high as possible. If the memory frequency is too low, then the reading and writing from and into it, will require a high amount of time, seriously slowing down the process. However, caching mechanism should be also available on each processor but all the generated data can't fit into the cache memory so that is why the RAM frequency is an important element. The on-line authentication system, using the first solution, is drawn in the following image:

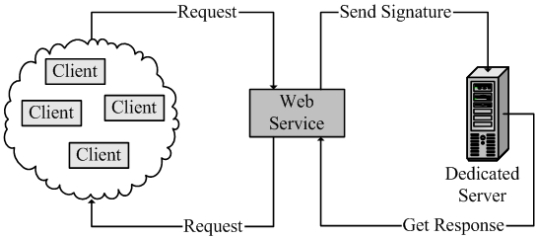

Figure 5: Authentication system using a 16 processor server.

The system will host a web-service that receives all the client-s requests, and all the distance computations are done inside the server. The web-service can be hosted on a different station. The disadvantage of such a system is that a powerful server is very expensive and it-s purpose is for general usage, rather than to be used exclusively for authentication purposes.

Therefore we need a cheaper solution than this one, and designed specifically to accelerate the authentication process.

3.2 Accelerating the authentication process using a FPGA development board

FPGA development boards have became more and more used lately, especially where computation power is needed, in systems with an increased computational complexity. Devices composed of several FPGA boards are used for example in biotechnology, performing sequence alignments (Hoang, 1993), proteins matching, docking, networking devices (Mohd, 2008) etc.

The FPGA acronynm, stands for Field Programmable Gate Array, which is a chip, which contains a matrix of elementary electronic circuits, which can be combined to build a complex function. A digital circuit systems designer can implement inside a FPGA board, a processing unit that can perform one specific algorithm. He designs the system's blocks and describes them in a hardware description language. After that a synthesizer like XilinX for example, is used to implement the new system, inside the FPGA chip.

One key element for a system working on a FPGA is the achieved frequency. If it-s high, then the system will be efficient. To increase the frequency, the delay from a combinational circuit-s input to it-s output should be as small as possible. A combinational circuit is one that does not work by using a clock input and it performs it-s function asynchronously.

One feature the FPGA board should have is the presence of block-RAM or if possible external RAM blocks. The block-RAM are the RAM cells located inside the FPGA chip, and it-s access frequency is the same with the system-s working frequency. To use external memory chips, frequency adapters are needed and not always these blocks can be used at full speed.

The number of elementary circuits inside the FPGA is also crucial. If the chip has a large number of gates, which we can implement several processing units inside a single chip, that perform in parallel. The following picture, describes the general arhitecture of a system composed of several Levenshtein processing units, implemented on a FPGA chip.

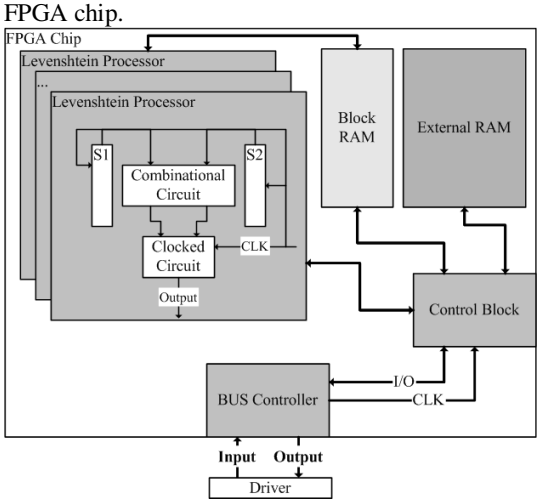

Figure 6: General architecture of a system performing multiple Levenshtein comparisons on a FPGA board.

The system can compute multiple Levenshtein distances in parallel, on a single chip. If we need more speed, we can build a custom hardware device composed of several FPGA boards that work independently, controlled by a processor. However this solution raises high implementing problems so it-s preferable to use just FPGA-s connected to a single computer that will distribute processing tasks to them.

The following table presents the time needed for a number of comparisons, depending on the system-s frequency. We assumed that our FPGA board will host 10 processing units. We calculated the time needed to compute 2050 distances between strings of 750 symbols, 32 bits each. We intend to make a comparison between the 3 proposed solutions so the number of comparisons should be the same for all systems. Of course we can approximate 2050 with 2048.

Table 3: Computing duration using a FPGA board.

System-s Frequency	Used BUS	Duration (ms)
50 MHz	USB 2.0	22700
100 MHz	USB 2.0	11448
150 MHz	USB 2.0	7700
200 MHz	USB 2.0	5800
300 MHz	USB 2.0	4500
300 MHz	PCI Express	4200

A frequency of 200 MHz is achievable even by using a low cost FPGA, such as a Spartan 3. More performant boards such as Virtex 5 can achieve even a higher frequency. However to reach this goal, many optimizations need to be done, like adding pipeline stages in combinational circuits that reduce frequency.

We can also see that the used BUS does not influence considerably the response time. An on-line authentication system using FPGA boards to perform the distances, is described in the following figure. Besides the Levenshtein processors, a BUS controller must be implemented inside the FPGA, in order to synchronize the system with the computer's BUS and to realize the data transfer.

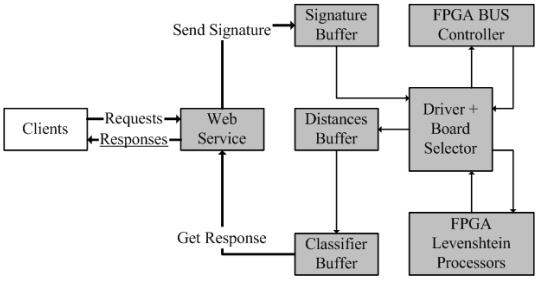

Figure 7: Authentication system using multiple FPGA boards.

3.3 Accelerating the authentication process using CUDA enabled video cards

A solution proposed in the last few years for high computing processes is using graphics cards. nVidia producer launched the CUDA architecture and also a software development kit, which allows programmers to use the graphic card-s capabilities at full power (nVidia 2008). For example, a developer familiarized with the C language, can easily write C code for CUDA, respecting a number of conventions and that code will be computed by the graphics card. CUDA is very suitable for algorithms that can be parallelized, matrix multiplications, etc. In the proposed solution we will use the graphics card by launching multiple Levenshtein algorithms instances that compute in parallel.

The CUDA architecture is described in the following image (nVidia 2008). Each graphics card chip is composed of a number of multiprocessors each containing a number of 8 streaming processors. Fast shared on-chip memory is available to use and also the board offers a high amount of RAM connected externally, called device memory.

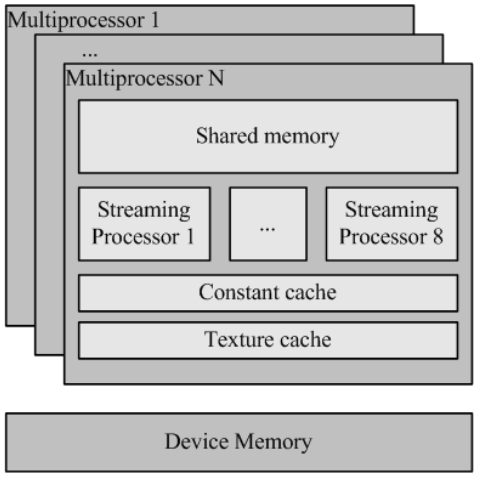

Figure 8: nVidia CUDA architecture.

The device memory is the slowest RAM available. The streaming processors can read and write to it, but at great time cost. The shared memory is the best solution to use when time is critical. Each streaming processor has a number of fast registers that can also be used for optimisations.

We used the same evaluating method as for the dedicated server solution, by generating a matrix of input strings. We have copied the matrix into the device memory and then launched threads on each streaming processor of each multiprocessor. The operations needed to perform the comparisons are mentioned below:

- Start Timer
- Copy strings matrix into device memory
- Fill the shared memory of each processor
- Launch Levenshtein algorithm on several threads
- Copy distances into device memory
- Copy distances from the device memory into the system's RAM
- Stop Timer

The results we have obtained by using this solution are presented in the following table. We have used 3 CUDA enabled graphics cards of different computing capabilities. We initiated the comparison of 2048 pairs of strings, 750 symbols of 32 bits each. This amount of data is generated by a number of around 40 clients, accessing the system simultaneously. We can see that the last client will receive a response to his request in less than 2 seconds. If the number of available boards increases, the incoming data will be processed proportionally faster.

Table 4: Computing duration using nVidia CUDA enabled video cards.

Video Card	Capabilities	Duration (ms)
nVidia Quadro NV 135 M	2 Multiprocessors	23625
nVidia GeForce 9500 GT	4 Multiprocessors	7266
nVidia GeForce GTX275	30 Multiprocessors	1390

An on-line authentication system, using several graphics cards to process distances between invariants strings, will have the architecture presented below.

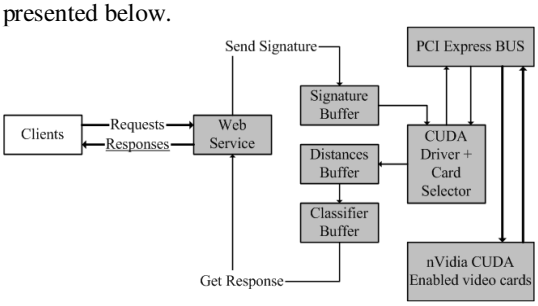

Figure 9: Authentication system using multiple nVidia CUDA enabled video cards.

The communication through the PCI Express BUS is transparent to the developer because it is realised by the provided CUDA driver (nVidia 2008). The implementing problems are similar to ones that appear when implementing the server solution. The main advantage of this solution is that the video cards are relatively cheap and their applicability area is rather large. The system is also scalable, because adding extra graphic cards is relatively easy, without major code modifications.

4 CONCLUSIONS

We have presented 3 solutions of accelerating on-line authentication, by using dynamic handwritten signature. We have presented the signature processing which is made, and we have shown 3 methods of speeding-up the most computational blocks. The following table synthesizes the results we have obtained.

We have considered a system using a 16 processor powerful server, one using 3 USB 2.0 FPGA boards with 10 Levenshtein processors each, working at 300 MHz and also a system using a single nVidia GeForce GTX275 video card.

Table 5: Comparison between the proposed acceleration solutions.

Solution	Price	Duration (ms)
16 Processor Server	~8000 USD	1328
3 FPGA Boards on a PC	~2000 USD	1400
1 nVidia GeForce GTX275 video card	~300 USD	1390

As we can see from the table above, the solution that should be used is obvious. Motherboards built using nVidia SLI technology allow up to 3 video cards on one single system so the speed achieved can be highly improved with minimal costs.

Given this context, an on-line system of authenticating users by their dynamic signature, can respond to a number of around 100 requests per second, when using 3 video cards, which makes it a high security feature needed to be considered. Using the proposed architectures, the system is very scalable and if the number of requests increases, more computing power can be added at a small price.

REFERENCES

Marcu, E., 2009. Method of combining the degrees of similarity in handwritten signature authentication, using neural networks. *In AI-2009, The Twenty-ninth SGAI International Conference Cambridge, UK.* Springer

Marcu, E., 2009. Self-built grid. *In IDC'2009, 3rd International Symposium on Intelligent Distributed Computing*. Springer

Hoang, D. T., Lopresti, D., 1993. FPGA Implementation of Systolic Sequence Alignment. *In International Workshop on Field Programmable Logic and Applications*. Springer Berlin

Mohd, E. T., Mohd, Y. I. I., Tee., H. H., Madhihah, S., 2008. Hardware based SPAM/UCE Filter Design with Levenshtein Distance Algorithm: A Framework. *In Proceedings of Internet Convergence Conference, 11-13 March 2008, Kuala Lumpur*. Non-Scopus Cited Publication.

Manavski, A. S., Valle. G., 2008. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *In BMC Bioinformatics 2008*. BMC Bioinformatics.

nVidia CUDA Zone Examples of GPU Processing, *http://www.nvidia.co.uk/object/cuda_home_uk.html*

nVidia CUDA Programming Guide, 2008. *http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf*