Paper

Design Exploration of AES Accelerators on FPGAs and GPUs

Vincenzo Conti[1] and Salvatore Vitabile[2]

[1] *Faculty of Engineering and Architecture, University of Enna Kore, Enna, Italy*
[2] *Department of Biopathology and Medical Biotechnologies, University of Palermo, Palermo, Italy*

Abstract—The embedded systems are increasingly becoming a key technological component of all kinds of complex technical systems and an exhaustive analysis of the state of the art of all current performance with respect to architectures, design methodologies, test and applications could be very interesting. The Advanced Encryption Standard (AES), based on the well-known algorithm Rijndael, is designed to be easily implemented in hardware and software platforms. General purpose computing on graphics processing unit (GPGPU) is an alternative to reconfigurable accelerators based on FPGA devices. This paper presents a direct comparison between FPGA and GPU used as accelerators for the AES cipher. The results achieved on both platforms and their analysis has been compared to several others in order to establish which device is best at playing the role of hardware accelerator by each solution showing interesting considerations in terms of throughput, speedup factor, and resource usage. This analysis suggests that, while hardware design on FPGA remains the natural choice for consumer-product design, GPUs are nowadays the preferable choice for PC based accelerators, especially when the processing routines are highly parallelizable.

Keywords—AES, accelerators, FPGA prototyping, GPGPU, OpenCL.

1. Introduction

In the last decade the complexity of the architecture of graphical processing units has grown exponentially, pushing them outside the world of the dedicated processors to embrace the general-purpose applications field.

Recently Graphic Processing Unit (GPU) manufacturers have focused their attention not only on typical graphical processing tasks, equipping their products with characteristics explicitly aimed to the general purpose computing (IEEE-754 compliance floating point units is just an example). Nevertheless, the massive parallel design, which is a key feature for a GPU architecture, is an attractive property in many number crunching applications.

With the introduction of the Nvidia Fermi architecture [1] the interest in GPGPU has grown, because of its ambitious goal: for the first time, a GPU architecture was expressly designed to allow general-purpose computations. Even before the Fermi architecture, with the introduction of technologies such as CUDA, Stream and OpenCL the word GPGPU has assumed a new meaning. Before these frameworks, the only way to access the GPU processing power for general computing was to use shaders, by resorting to a cumbersome process in which data to process was encoded in textures pixels with many piratical limitations. However, many of this proof of concept showed the true potential of GPU devices. Subsequently GPU devices were used as accelerators for many scientific applications, ranging from image processing to Basic Linear Algebra Subprograms (BLAS), with successful results.

At the same time, FPGA devices have been traditionally used for various and different purposes, thanks to the very high degree of customization available to the designer. With more details this technology has been used to implement video processing [2] and [3], biometric recognition systems [4] and [5], mathematical and/or biological coprocessors [6] and [7], security access management [8], [9] and [10], and so on.

The difference in terms of overall costs, development time and background knowledge required to target both platforms justifies the interest by the scientific community in a full comparison. To make this comparison effective, an algorithm that can be easily implemented in both hardware and software platforms is needed. Rijndael algorithm is a good candidate for this purpose as it was designed keeping an eye on both platforms.

In this paper, two implementations of the AES encryption cipher in counter (CTR) mode are presented: a novel FPGA design for the Celoxica RC1000 board, developed with Agility's Handel-C compiler, and parallel OpenCL software which runs on GPU. GPGPU is an alternative to reconfigurable accelerators based on FPGA devices. The FPGA implementation consists of four AES cores, each of which performs a single AES encryption in 0.48 μs with 70 MHz clock, delivering a throughput of about 1036 Mb/s. The OpenCL software is a simple port of an ANSI C implementation of the Rijndael algorithm. The two solutions exhibit good performance compared to a general-purpose CPU implementation, thus are both suitable to be used as accelerators. In addition, the architectural constraints, power consumption, speedup factors, overall costs of the two projects and their analysis has been compared to several others in order to establish which device is best at playing the role of hardware accelerator by each solution showing interesting considerations in terms of throughput, speedup factor, and resource usage. This analysis suggests that, while hardware design on FPGA remains the natural choice for consumer-product design, GPUs are nowadays the preferable choice for PC based accelera-

tors, especially when the processing routines are highly parallelizable.

The paper is structured as follows. Section 2 presents a review of other works available in literature in which the two technologies are compared. In Section 3 the Rijndael algorithm is briefly described, together with the implemented CTR mode of operation. Sections 4 and 5 illustrate the FPGA and OpenCL proposed implementations respectively. Section 6 describes the testing environment while in Section 7 the results are extensively analyzed and commented. Section 8 presents an overview of similar works with a comment on the performance achieved by the proposed solutions. Finally, Section 9 contains the conclusions of this work.

2. Related Works

There is a variety of publications in literature that compare FPGA and GPGPU implementations and the results may vary depending on the platforms used.

In 2005 Cope *et al*. [11] pointed out the limitations of GPU based solutions compared to FPGA devices due to the low memory bandwidth. In the same year, Mali *et al*. [12] showed an implementation of AES on FPGA, using the same platform used for this paper (the Celoxica RC1000 board). The proposed solution is c.a. 5.7 times faster running at a lower clock frequency.

Lately, in 2007 another interesting comparison was made by Baker *et al*. [13]. In their work, they implemented a matched filter on both FPGA and GPU devices, obtaining similar throughput. Moreover, when comparing throughput against costs, they show how GPU solutions are the cheapest.

Costs involved in targeting FPGAs and GPUs have been analyzed by Shuai Che *et al*. in [14] comparing the two solutions in three different tasks: Gaussian elimination, Needleman-Wunsch and DES.

However, the answer to the question "Have GPUs made FPGAs redundant as accelerator devices?" is still open. Contrasting results were shown depending on many factors, including the algorithm implemented, the targeted devices and the programming frameworks.

For example, in [15] the performance of common image processing algorithm implemented in FPGA and GPU are compared. The FPGA implementation outperforms the GPU, especially in those algorithms were a careful memory access policy is necessary to synchronize the GPU threads.

Different results are shown in [16] where an implementation of common SPICE routines is presented giving similar results in both hardware and software approaches. Even if FPGA can outperform small factor devices, when compared to most powerful GPU they suffer for the limited resources on board and the poor scalability.

Depending on the application, the results may be even more different. In [17] sparse matrix vector multiplication implemented on GPU outperforms the FPGA counterpart, although the authors point out that their FPGA solution is highly penalized by a very poor memory bandwidth.

Finally, in [18] a SEAL encryption implementation is presented in both FPGA and GPU. Both platforms achieve the same overall performance. In this paper, the implementation of an encryption algorithm is also discussed, but it is worth to note that AES is slightly more complex than SEAL and so it better exploit the differences between the two processing platforms.

3. The AES Standard

AES is the standard currently recommended by NIST for symmetric block cipher encryption. The actual standard publication [19], issued in November 2001, includes a detailed description of the Rijndael algorithm, which was chosen among others like MARS, RC6, Serpent and Twfish, because of its high degree of cryptographic security and its simplicity. The Rijndael selection process was carried through openly and with the full support of the scientific community. This has gained to AES the interest of many operators in the cryptographic security field and made the transition to the new standard very quick.

3.1. The Rijndael Algorithm

Rijndael is a symmetric block cipher algorithm, which runs a certain number of rounds on every input block. In Fig. 1 the algorithm structure is shown. Its design, which is totally different with respect to the previous standard DES, is very far from the traditional Feistel cipher structure. The Rijndael cipher applies Galois's Finite Field arithmetic to match the confusion and diffusion requirements and it is composed of two distinct procedures for encryption and decryption. The input blocks size is of 128 bits while the key can be 128, 192 or 256 bit wide, depending on the security degree required. The key size is also related to the number or rounds of the encryption/decryption procedures as shown in Table 1.

Table 1
Rijndael key size and rounds number

Key size [bit]	Rounds number
128	10
192	12
256	14

3.2. The AES Round Structure

Rijndael iterates the same sequence of operators, named *round*, on every input block. The plaintext is split in chunks of 16 bytes and each of these is treated as a 4×4 matrix called the *state* vector.

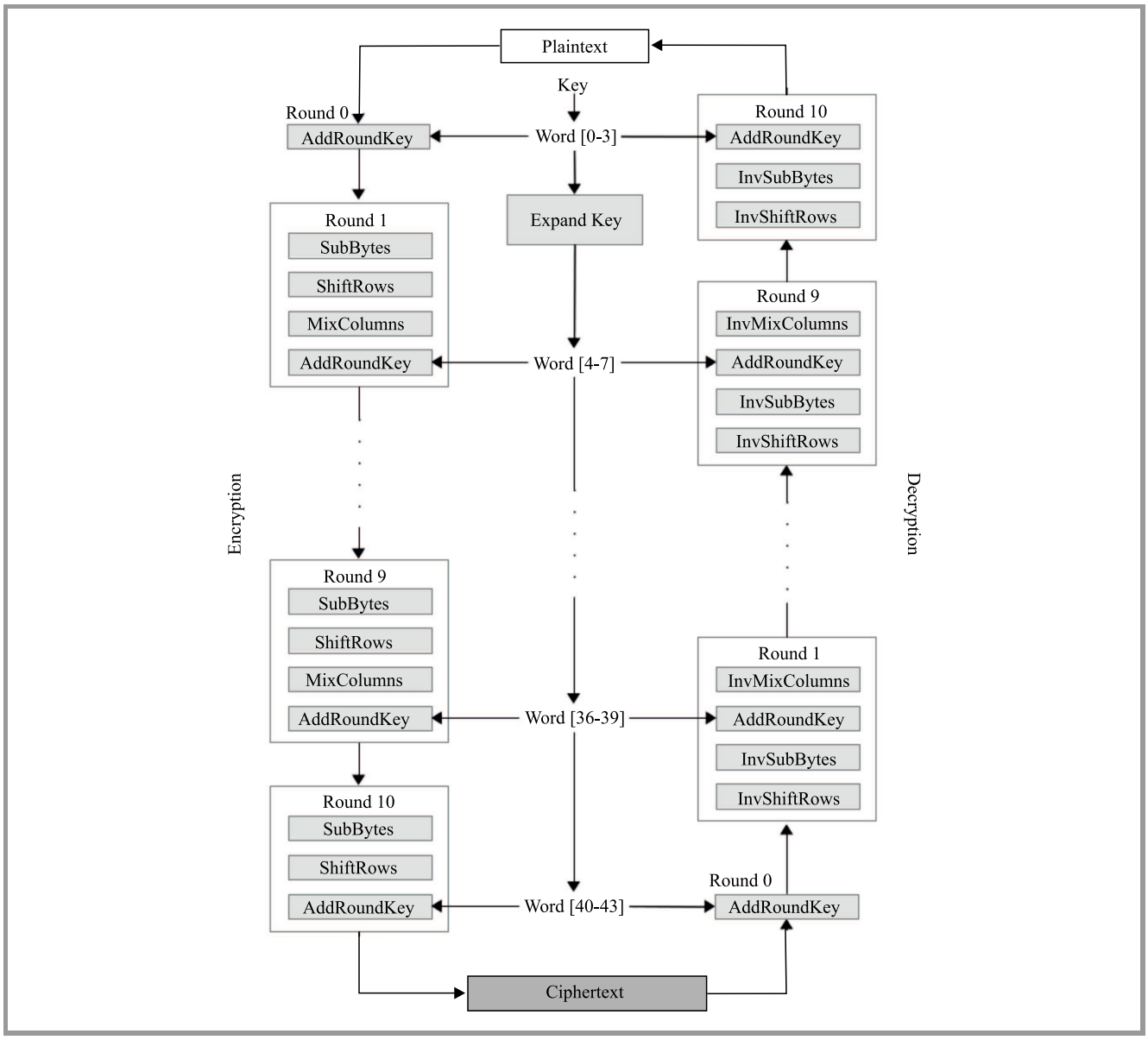

Fig. 1. The Rijndael algorithm.

The four operators *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* are used in every round but the first and the last, which are defined differently.

The SubBytes function uses a substitution box, named *Sbox*, to map every byte in the state vector on a proper 8 bit value. The mapping output is obtained with the following affine transformation applied to the multiplicative inverse $x_7x_6\ldots x_0$ in the $GF(2^8)$ of the input byte:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

The 0x00 value, whose multiplicative inverse is not defined in $GF(2^8)$, is simply mapped to the 0x63 byte. The Sbox is usually stored in memory and accessed like a look-up table to speed up the substitution function.

The ShiftRows function consists of a circular left shift of 1, 2 and 3 positions for the rows 2, 3 and 4 respectively of the state vector. The first row remains unchanged.

The MixColumn function consists of a linear transformation which is applied to the elements of each column:

$$\begin{bmatrix} \hat{s}_{c0} \\ \hat{s}_{c1} \\ \hat{s}_{c2} \\ \hat{s}_{c3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{c0} \\ s_{c1} \\ s_{c2} \\ s_{c3} \end{bmatrix}.$$

The c subscript is the column index. The multiplication and the add operators used in the matrix product are those defined in $GF(2^8)$.

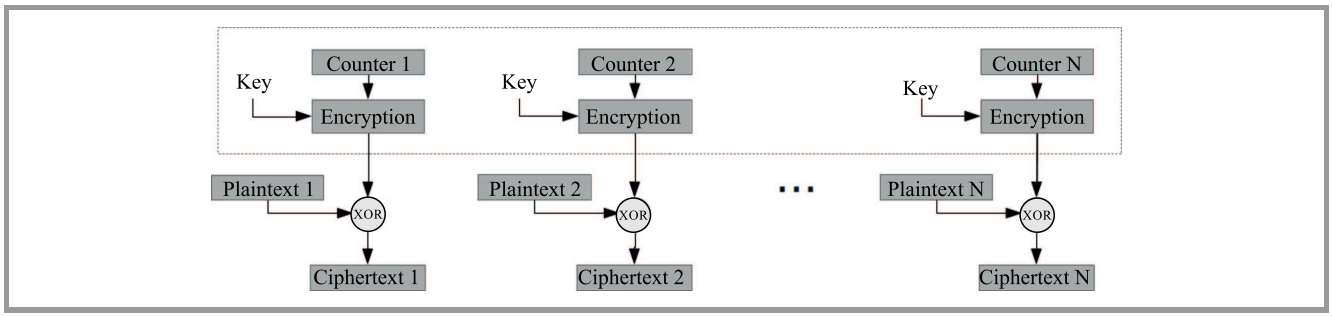

Fig. 2. CTR mode encryption.

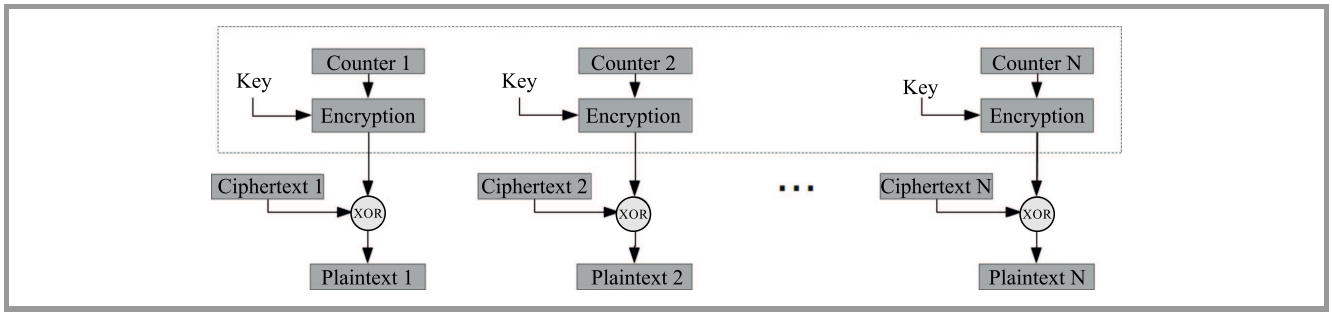

Fig. 3. CTR mode decryption.

The AddRoundKey function is the only operator, which involves the secret key. A distinct 128 bit subkey for each round is extracted from the key and is XOR-ed with the state vector. The key scheduling procedure is also described in [19].

3.2.1. Counter Mode

The design presented in this work uses the counter (CTR) mode of operation [20] because it allows the parallel execution of the cipher on each block while ensuring a strong degree of resistance to cryptanalysis.
Another interesting feature of the CTR mode consists in the use of the same encryption procedure for both encryption and decryption. This comes very useful for the AES cipher, which would normally require two distinct implementation for the encryption/decryption routines.
Looking at the Fig. 2 it is easy to note that the data being codified by the cipher is a special value, named *counter*, which is XOR-ed with each block, and is different for every block (e.g. incremented by 1 for each block encryption).
The same operation has to be performed in decryption: the reversibility of the cipher actually resides on the use of the XOR operator (see Fig. 3).
The seed value for the counter can also be kept secret to increase the overall degree of security of the AES cipher with respect to brute-force attack.

4. AES Processor Design

The proposed design is an implementation of the 8 bit oriented version of AES. Each round operation takes a single clock cycle, except the SubBytes and ShiftRows operation that were mixed together. Some of the suggestions shown in [21] where used to reduce area occupation and maximum delay path without compromising the throughput. This lead to a total of 33 clock cycles required to perform a single AES encryption. A summary of the characteristics of this design is shown in Table 2, while the overall architecture is shown in Fig. 4.

Table 2
Proposed AES processor summary

Core operating frequency	70 MHz
Memory operating frequency	33 MHz
Average throughput	1036 Mb/s
Occupation	18048 slices
Maximum delay path	13.92 ns

In next subsection a detailed description of the proposed architecture is discussed. Parallel and pipelined processing has been used to achieve high throughput performance.

4.1. Overall Architecture

A first level of parallelization is easily achieved by instantiating multiple AES cores on the chip. The memory interface of the Celoxica RC1000 board allows parallel access of the 4 memory banks. So in the proposed design four independent AES blocks are capable of running with full parallelism, achieving an overall performance of 4 times the single AES core, scoring a little more than 1 Gb/s.

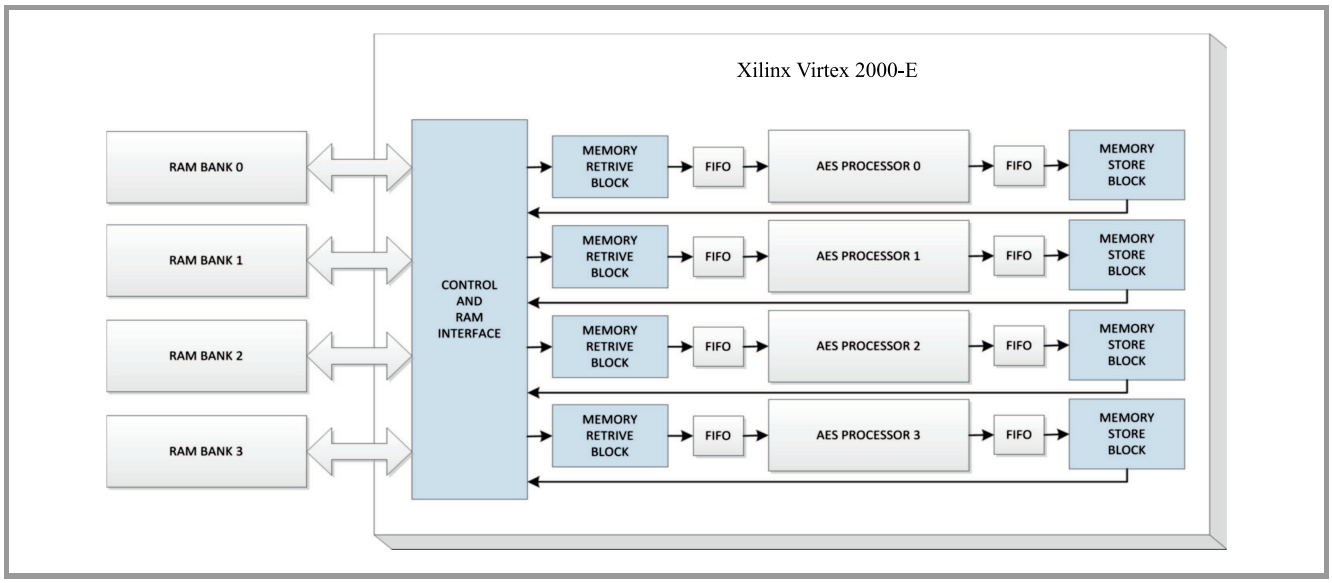

Fig. 4. Overall architecture implementing AES processor using Xilinx Virtex 2000-E FPGA.

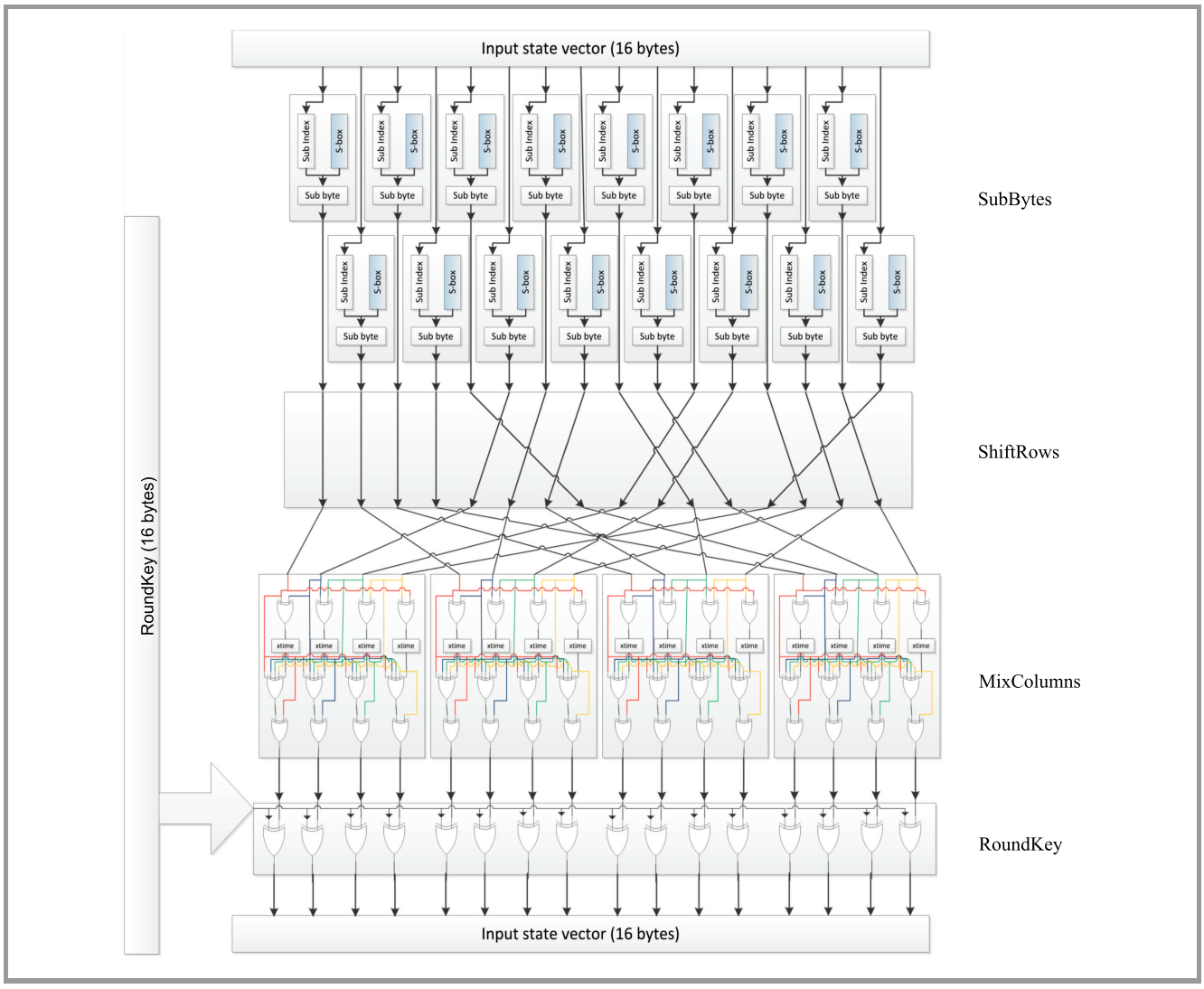

Fig. 5. AES round architecture.

4.2. AES Core Architecture

One AES core contains the circuitry required to perform AES 128 bit encryption. Table 3 shows the performance of the proposed AES core. Figure 5 shows the architecture of a single round circuit. The full round operation takes 3 clock cycles. To allow full parallelism to the SubBytes operation, 16 S-boxes have been instantiated in the ROM memory. Allocating registers array in Handel-C is very resource consuming compared to the usage of ROM bits, but obviously, the same ROM cannot be accessed simultaneously by multiple circuits. This led to the choice of allocating multiple S-boxes. Even though this choice sacrifices more area, the high advantage in the overall performance is a good compromise. Each AES core is implemented in 3360 slices (c.a. 17.5% of the total available on chip).

Table 3
Proposed AES block summary

Total latency	0.48 μs
Operating frequency	70 MHz
Average throughput	259 Mb/s
Occupation	3360 slices
Maximum delay path	13.92 ns

4.3. Pipelined Design

Unfortunately, the Celoxica RC1000 has very high latency memory, which cannot be accessed at frequencies higher than circa 33 MHz [22], [23]. The proposed AES circuit has a maximum delay path of 13.92 ns, so it can theoretically reach up to 71 MHz. To reduce the penalization introduced by the very poor memory interface, a double domain clock design was used. One domain clock, running at 33 MHz, contains the circuitry for data fetching and write back, while the other, running at 70 MHz, contains the 4 AES cores. The communication between the two clock domains is ensured by eight 128 bit channels, each of which equipped with a FIFO queue. The data-fetching block and the write back block are running in a parallel fashion. The synchronization between these two blocks is guaranteed by 4 semaphores. With this solution, memory fetches can happen while encrypting previously fetched blocks, increasing the overall performance (see Fig. 6). The total time required to encrypt 8 MB is nearly the same required to simply access the data to the on board RAM.

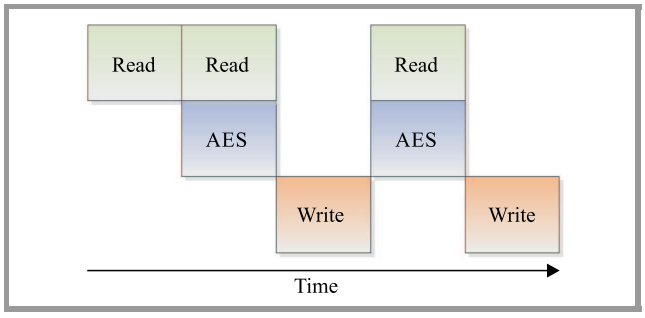

Fig. 6. Pipelined execution.

5. OpenCL Implementation

The GPU version has been implemented using OpenCL rather than similar but proprietary technologies for its portability. The results obtained by running the same implementation on different platforms (the Nvidia GT520 and GT555M and the Intel Core i7 processor) are reported in the following subsections.

5.1. The Threading Model

AES in CTR mode is perfectly suitable for parallel applications. As previously discussed in Subsection 3.2.1, in CTR mode every block encryption is independent, and thus there is no need to implement ad-hoc thread synchronization polices. Therefore, the adopted threading model can be simply summarized as follows:

- a grid is defined with only one work-group, and a single kernel running the Rijndael algorithm;
- in the work-group, each 128 bit data block is mapped into a single work-item. Thus, the number of work-items will be equal to the number of 128 bit data block in our stream;
- parallel execution of the work-items. The counter to use in CTR mode is calculated from the thread ID, as the threads are mapped 1:1 to the data blocks.

5.2. Targeting the GPU on a Consumer Grade PC

Special care need to be taken when working with consumer grade computers, as most probably the GPU used as accelerator will be the only one available to the system, and so it will be shared by several concurrent tasks, such as: desktop environment running in background updating the screen content, any application using 3D capabilities, accelerated video playback, etc. Therefore, it is important to understand that a single OpenCL program cannot lock the GPU for an undefined time. On some platforms, this may be a strict requirement. In the Microsoft Windows environment, for example, the video driver is automatically reset if the GPU doesn't respond to the OS commands within a predefined timeout (usually just a couple of seconds). A bad designed OpenCL program could never terminate correctly. The solution used in this work is simple but effective: the input data is divided into chunks that the GPU can process without hogging the system. The size of the chunk is a critical point of choice: a small size will cause an underutilization of the GPU processing power, while a large size can cause system hogging. In conducted experiments, the input chunk was set to 8 MB, as this size showed the best compromise between the utilization of the device and the

overall performance. Moreover, 8 MB is exactly the same size used for the FPGA implementation, as it is the total amount of on board RAM memory. Using the same size increase the accuracy of our measurements as the overhead introduced to divide the data in multiple chunks is the same regardless of the processing platform being tested.

5.3. Practical Aspects

Another great advantage offered by OpenCL is the compatibility of the C99 specification [24]. With a few adjustments, our C code developed for the CPU was ported successfully to the GPU. In particular, only some decoration was added to the function prototypes to correctly address the various memory spaces available in OpenCL. A sensible increase in performance over the standard CPU implementation required less than a person-day work. OpenCL programs are compiled on the fly at run time, so the compatibility with different platforms is guaranteed by the underlying software layer. This may contrast with the possibility to optimize the code for a particular device or architecture. In this case, multiple versions of the same OpenCL software can be developed and then selected at run time depending on the running platform. As an example, consider how an OpenCL program accesses the global memory. Since the memory hierarchy may vary from architecture to architecture, different ways of implementing global memory access were examined. In particular, to ensure the maximum performance the data alignment of the write back operation matched the alignment of the running device.

6. Testing Environment

Each implementation was initially tested using the AES standard test vectors recommended in [19]. A software library named *FastAESlib* was then developed to create a common interface for accessing each processing platform (FPGAs, GPUs and CPUs) addressed in this work. It can perform several tasks, as summarized below:

- enumerate at run-time the processing platforms available in the system (FPGAs, GPUs and CPUs),
- offload the processing task to any of the available processing platforms,
- setup platform specific parameters (e.g. the working frequency of the FPGA),
- report the progress of the current task,
- measure the overall execution time (using the OS high resolution timers),
- measure the processing execution time reported from the devices (on board timers for FPGAs and OpenCL event timers for GPUs).

This library was then used to develop three software applications designed to test the various platforms on different scenarios. The first one is an image encipher/decipher, which processes uncompressed image data. Figures 7 and 8 show the user interface of this application. The user can set all the processing parameters exposed by the FastAESlib library and obtain on screen the performance counters measurements (both execution time and throughput). As shown in Fig. 8, after the encryption phase the image data is completely scrambled, without exposing neither the chromatic information nor the original image structure. This visually proves how powerful is the CTR mode compared to other standard modes of operation. As a proof of concept, another software application named *FileCrypter* was developed to test implementations with large files. This application can encipher/decipher a file with a single password.

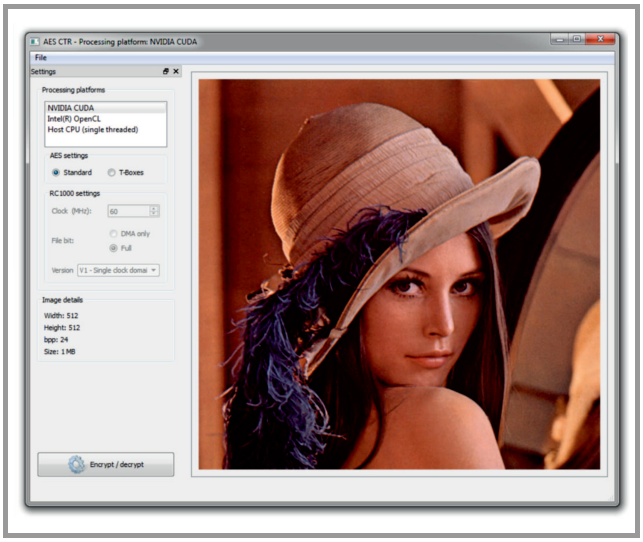

Fig. 7. Screenshot of the software ImageCrypt. (See color pictures online at www.nit.eu/publications/ journal-jtit)

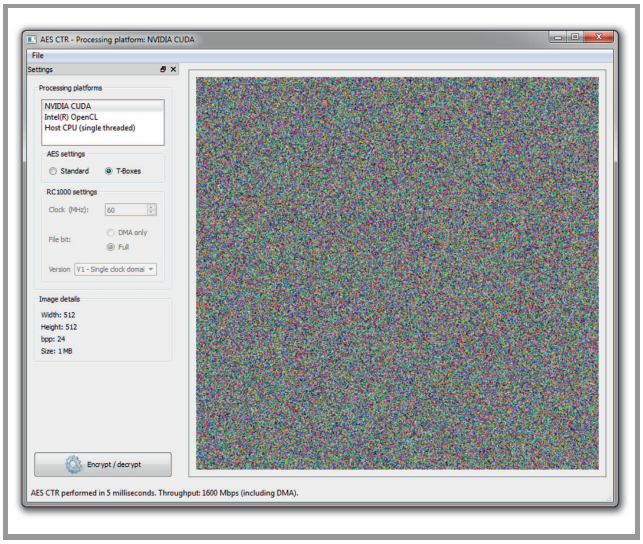

Fig. 8. Screenshot of the software ImageCrypt after encryption.

Lastly, a scripted application was developed to benchmark the various implementations discussed in this work.

This software utility performs AES encryption/decryption a specified number of times (in our tests 20 times) and calculates the average execution time and throughput. Moreover, when using the FPGA based processing platform, it can repeat the testing sequence at different clock frequencies, verifying the correctness of the result at each iteration. The results obtained using this tool is discussed in Section 6.

7. Experimental Results

The presented implementations show interesting results compared against a standard CPU. Table 4 shows the overall performance of the target systems including memory transfers time. When considering only the data rate, the fastest solution appears to be the OpenCL based implementation. However, it is important to consider the differences in the following three areas:

- the memory bandwidth can have a significant impact on the overall performance,
- the throughput should be normalized considering the different working frequencies,
- the various devices have a very different power consumption levels.

Table 4
Overall performance of the target platforms

Platform	Clock [MHz]	Rate [Mb/s]	Rate/clock ratio
FPGA	70	198	2.828
Nvidia GT 520	1620	520	0.321
Nvidia GT 555M	1180	1280	1.084
Intel Pentium 4	2000	42	0.0210
Intel Core i7	2500	81	0.0324

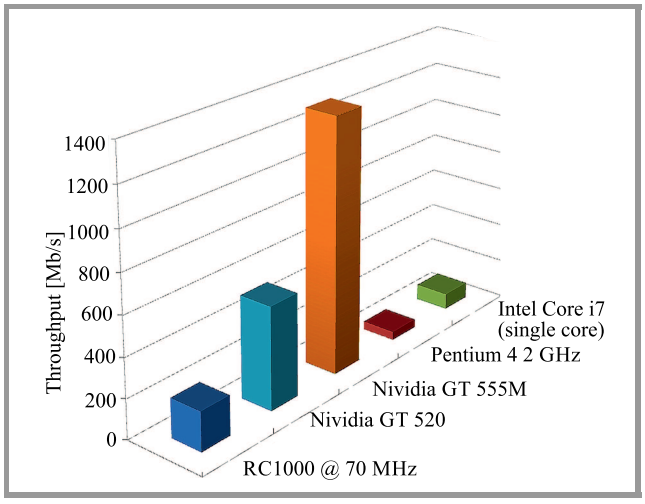

Fig. 9. Overall performance of the target platforms.

In Table 4 and Fig. 9 the overall performance measurements, but normalized with respect to the clock frequency, are shown. It is clear that the FPGA based solution can achieve better performance at lower clock rates, but it is worth to note that the GPU based solution exhibit a similar throughput/clock ratio, while the values reached by general purpose CPUs are two orders of magnitude lower. Interesting results are obtained when filtering out the time consumed by memory transfers (from the central memory to the on board memory). Table 5 and Fig. 10 show the processing throughput. This shows how the memory latency negatively affects the throughput of the RC1000 board, while the GPU based solutions are only lightly affected by the DMA operation. This is a logical consequence of the different technologies used by the two devices. Table 6 highlights the main differences.

Table 5
Performance of the target platforms without DMA time

Platform	Clock [MHz]	Rate [Mb/s]	Rate/clock ratio
FPGA	70	1036	14.8
Nvidia GT 520	1620	548	0.338
Nvidia GT 555M	1180	1440	1.22

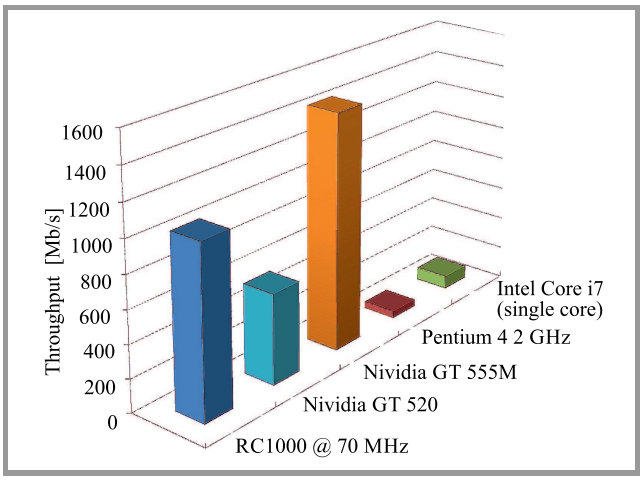

Fig. 10. Overall performance of the target platforms without DMA time.

Table 6
RAM memory comparison

Property	FPGA	GT 520	GT 555
Latency [ns]	25	10	10
Bus width [bit]	32	64	192
Clock [MHz]	33	900	900
Technology	SRAM	DDR3	SDDR3

The normalized throughput/clock ratio without DMA time shows how powerful the FPGA implementation is (see Table 5). When artificially scaling the three platforms'

clocks to the same frequency, the FPGA is 12 times faster of the fastest GPU based solution. Other interesting considerations can be made about the power consumption. While FPGAs are low power devices, GPUs are generally power-demanding processors. However, compared to a general purpose CPU, both the FPGA and the GPU platforms are the most energy efficient. An important aspect worth to note is the development cost. Regardless of the hardware cost, where a substantial difference exists between FPGAs and GPUs, another major disequilibrium can be found in the Time To Market (TTM) parameter. Even if TTM is low for FPGAs, designing hardware is generally a more time consuming task when compared to software development. Lastly, another key advantage of GPGPU technologies is the portability of the code. The same code can be executed on different OpenCL compliant devices without adjustments exploiting their potentials. FPGA designs need careful handling when ported from one device to another, making the porting operation hard and the previously developed code less reusable.

8. Discussion and Comparison

This section is devoted to the analysis of several other AES implementations on both GPU and FPGA devices. The direct experience of the implementation described in the previous sections is the starting point of our analysis, but first comes a little digression on the parameters that will be considered as terms of comparison. A comparison based on throughput vs. clock rate would give no useful results when comparing such different architectures. A targeted approach is needed to analyze each one's peculiarities before a direct comparison can be evaluated. FPGAs throughput will be analyzed against resources usage while GPUs' total number of stream processors will be considered as the main trade-off factor. When comparing the performance of such different devices it is important to investigate the different approaches available to the designers. For instance,

Table 7
Comparison of discussed FPGA implementations

Paper (characteristic)	Slices	Clock [MHz]	Throughput [Gb/s]
Rodriguez *et al.* [26] (pipelined)	5677	34.2	4.21
Mali *et al.* [12]	–	74	0.18
Kotturi *et al.* [27] (pipelined)	5408	232.6	29.77
Sivakumar *et al.* [28] (AES-CTR)	6766 CLB	194	2.257
Singh e*t al.* [29] (pipelined)	6352	347.6	44.2
Hoang *et al.* [25]	895	–	1.03
The proposed system (AES-CTR)	3360	70	0.25

Table 8
Comparison of discussed GPU implementations

Paper (characteristic)	GPU	Clock [MHz]	Throughput [Gb/s]
Manavski *et al.* [30] (CUDA)	128	575	8.2
Wang *et al.* [31] (OpenCL)	240	1476	1.05
Wang *et al.* [31] (CUDA)	240	1476	1.2
Keisuke *et al.* [32] (CUDA)	240	1476	32.5
The proposed system (OpenCL)	144	1180	1.25

AES can be implemented with or without look-up tables (T-boxes). Moreover when targeting hardware, pipelining is a natural choice against task parallelism, which is the foundation of the GPGPU computing model. In what follows, different FPGAs designs for AES are analyzed first. Next, parallel implementation of AES on GPU is examined. Table 7 shows a summary of the results of several AES implementations on FPGAs. In [12], Mali *et al.* presented a AES processor design in Handel-C on the same FPGA device used in this paper. The clock rate is slightly different, but the maximum throughput achieved from the solution proposed in this paper is higher. This may be due to the Handel-C compiler, which is very sensitive to the instruction order, and the control flow structures used. As another example of the impact of the Handel-C designing process on the result, consider that the AES processor design proposed in this paper requires 48 clock cycles to complete one 128-bit block encryption. Hoang *et al.* [25] proposed a VHDL design that completes 128-bit block encryption in 13 clock cycles requiring a lower number of slices, and therefore can potentially achieve higher throughput at the same clock speed. As previously mentioned, another important point is the processor design. The highest throughputs reported in Table 7 are relative to fully pipelined implementation of AES ([26], [27] and [29]). In this case, it is interesting to notice that, while the slices usage is slightly varying, the throughput/clock ratio is almost the same for each of these implementations. This observation leads to the conclusion that the performance of an optimal AES processor design for FPGA scales almost linearly with the clock rate given a fixed slices usage. Table 8 shows the results achieved by several AES parallel implementations running on GPU. The OpenCL implementation proposed in this paper was made out of an ANSI C implementation of the AES encryption routine. Therefore, no particular code optimization technique was adopted. Several test runs on the same GPU device showed heavy performance variations with different number of executing threads. In general, particular care must be taken in order to achieve optimal performance on GPU. As an example,

[31] and [32] report dramatically different throughputs on the same GPU, but is worth to mention that Wang *et al.* are relative to the XTS mode of operation of AES, which implies that some additional operations are executed within the Rijndael encryption procedure. Another critical parameter is, of course, algorithm design. Manavski [30] achieves a significantly higher throughput on a graphics processor equipped with less cores and a lower clock speed that the present work. As a side note, when directly compared, the solutions based on CUDA achieve a slightly better throughput than OpenCL. From this analysis results that GPUs and FPGAs achievements are comparable in terms of throughput. However, several differences are noticeable in the way these results are achieved on both devices. One of the factors that make the difference in achieving high throughput for FPGAs is the presence of high-bandwidth I/O capabilities, since clock speed is relatively small compared to ordinary graphics processor units. On the other hand, host-device I/O bandwidth is usually a limiting constraint for performance achievement on GPU, but this is usually compensated by the possibility of limiting data transfer for devices equipped with extended on board memory and by the high clock speed at the expense of increased power consumption levels. In the context of hardware accelerator design, where both FPGAs and GPUs are currently widely used, I/O capabilities are maybe the best point to evaluate the choice of one over the other achievement when the main concern is high performance.

9. Conclusion

This paper presents a direct comparison between FPGA and GPU used as accelerators for the AES cipher. The analysis of the results achieved on both has been compared to several others in order to establish which device is best at playing the role of hardware accelerator. In addition, the possibility of making a direct comparison between such different architectures have been investigated. This analysis suggests that, while hardware design on FPGA remains the natural choice for consumer-product design, GPUs are nowadays the preferable choice for PC based accelerators, especially when the processing routines are highly parallelizable. In fact, FPGA devices are still capable of delivering very high performance at low power consumption, but the possibility of programming GPUs with procedural paradigms, using the OpenCL or CUDA technologies, helped in making GPGPU an alternative to the use of FPGAs in the context of high performance computing, compensating for high power consuming levels.

References

[1] Nvidia "Fermi Compute Architecture Whitepaper", 2010.

[2] D. Lu, Q. Chen, K. Weng, and A. Zhang, "High speed video signal acquisition and processing system based on FPGA technique", in *Proc. Int. IEEE Conf. on Neural Netw. & Sig. Process.*, Nanjing, China, 2003, pp. 1233–1236.

[3] K. F. K. Wong, V. Yap, and T. P. Chiong, "Hardware accelerator implementation on FPGA for video processing", in *Proc. IEEE Conf. on Open Syst. ICOS 2013*, Kuching, Malaysia, 2013, pp. 47–51 (doi: 101109/ICOS.2013.6735046).

[4] V. Conti, C. Militello, F. Sorbello, and S. Vitabile, "Biometric sensors rapid prototyping on FPGA", *The Knowl. Engin. Rev.*, vol. 30, no. 2, pp. 201–219, 2015.

[5] C. Militello, V. Conti, S. Vitabile, and F. Sorbello, "An embedded iris recognizer for portable and mobile devices", *Comp. Syst. Science and Engin.*, vol. 25, no. 2, pp. 119–131, 2010.

[6] N. Neveset *et al.*, "BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment", in *Proc. Int. IEEE Conf. on Appli.-Spec. Syst., Architec. & Process. ASAP 2013*, Ashburn, VA, USA, 2013, pp. 241–244.

[7] E. Fusella, A. Cilardo, and A. Mazzeo, "Scheduling-aware interconnect synthesis for FPGA-based Multi-Processor Systems-on-Chip", in *Proc. 25th Int. IEEE Conf. on Field Programmable Logic & Appl. FPL 2015*, London, UK, 2015, pp. 1–2.

[8] S. Vitabile, V. Conti, C. Militello, and F. Sorbello, "An extended jade-s based framework for developing secure multi-agent systems", *Comp. Stand. and Interfaces*, vol. 31, no. 5, pp. 913–930, 2009.

[9] Y. Wang and Y. Ha, "FPGA based Rekeying for cryptographic key management in Storage Area Network", in *Proc. 23rd Int. IEEE Conf. on Field Programmable Logic and Appl. FPL 2013*, Porto, Portugal, 2013, pp. 1–6.

[10] C. Militello, V. Conti, S. Vitabile, and F. Sorbello, "Embedded access points for trusted data and resources access in HPC systems", *The J. of Supercomput.*, vol. 55, no. 1, pp. 4–27, 2011.

[11] B. Cope, P. Cheung, W. Luk, and S. Witt, "Have GPUs made FPGAs redundant in the field of video processing?", in *Proc. Int. IEEE Conf. on Field-Programmable Technol. FPT 2005*, Singapore, 2005, pp. 111–118.

[12] A. B. M. Mali and F. Novak, "Hardware implementation of AES algorithm", *J. of Elec. Engin.*, vol. 56, no. 9–10, pp. 265–269, 2005.

[13] Z. Baker, M. Gokhale, and J. Tripp, "Matched filter computation on FPGA, cell and GPU", in *Proc. 15th IEEE Symp. on Field-Programmable Custom Comput. Machin. FCCM 2007*, Napa, CA, USA, 2007, pp. 207–218.

[14] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs", in *Proc. Symp. on Appl. Specific Process. SASP 2008*, Anaheim, CA, USA, 2008, pp. 101–107.

[15] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing", in *Proc. 19th Int. IEEE Conf. on Field Programmable Logic and Appl. FPL 2009*, Prague, Czech Republic, 2009, pp. 126–131.

[16] N. Kapre and A. DeHon, "Performance comparison of single-precision spice model-evaluation on FPGA, GPU, cell, and multi-core processors", in *Proc. 19th Int. IEEE Conf. on Field Programmable Logic and Appl. FPL 2009*, Prague, Czech Republic, 2009, pp. 65–72.

[17] Y. Zhang, Y. Shalabi, R. Jain, K. Nagar, and J. Bakos, "FPGA vs. GPU for sparse matrix vector multiply", *Proc. Int. IEEE Conf. on Field-Programmable Technol. FPT 2009*, Sydney, Australia, 2009, pp. 255–262.

[18] K. Theoharoulis, C. Antoniadis, N. Bellas, and C. Antonopoulos, "Implementation and performance analysis of seal encryption on FPGA, GPU and multi-core processors", in *Proc. 19th Ann. Int. IEEE Symp. on Field-Programmable Custom Comput. Machines FCCM 2011*, Salt Lake City, UT, USA, 2011, pp. 65–68.

[19] "Federal Information Processing Standards Publication 197", Tech. rep., National Institute of Standards and Technology, 2001 [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[20] M. Dworkin, "Recommendation for Block Cipher Modes of Operation", Tech. rep., National Institute of Standards and Technology, 2001 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[21] Celoxica – Mentor Graphics, Handel-C advanced optimization [Online]. Available: https://www.mentor.com/

[22] Celoxica – Mentor Graphics, RC1000 Hardware Reference Manual [Online]. Available: https://www.mentor.com/

[23] Cypress. CY7C1049 512K x 8 Static RAM datasheet [Online]. Available: http://www.cypress.com/file/42811/download

[24] "The OpenCL Specification v 1.2", Tech. rep., Khronos OpenCL Working Group, 2011 [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[25] T. Hoang and V. L. Nguyen, "An efficient FPGA implementation of the advanced encryption standard algorithm", in *Proc. Int. IEEE Conf. on Comput. & Commun. Technol., Res., Innov., and Vision for the Future RIVF 2012*, Ho Chi Minh, Vietnam, 2012, pp. 1–4.

[26] N. S. E. Rodriguez-Henriquez and A. Diaz-Pkrez, "4.2 Gbit/s single-chip FPGA implementation of AES algorithm", *Electron. Lett.*, vol. 39, no. 15, pp. 1115–1116, 2003.

[27] D. Kotturi, S.-M. Yoo, and J. Blizzard, "AES crypto chip utilizing high-speed parallel pipelined architecture", in *Proc. Int. IEEE Symp. on Circ. and Syst. ISCAS 2005*, Kobe, Japan, 2005, vol. 5, pp. 4653–4656.

[28] C. Sivakumar and A. Velmurugan, "High speed VLSI design CCMP AES cipher for WLAN (IEEE 802.11i)", *Int. Conf. on Sig. Process., Commun.*, Chennai, India, 2007, pp. 398–403.

[29] R. M. Gurmail Singh, "FPGA based high speed and area efficient AES encryption for data security", *Int. J. of Res. and Innov. in Comp. Engin.*, vol. 1, no. 2, pp. 53–56, 2011.

[30] S. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography", in *Proc. International IEEE Conference on Signal Processing and Communications ICSPC 2007*, Dubai, United Arab Emirates, 2007, pp. 65–68.

[31] X. Wang, X. Li, M. Zou, and J. Zhou, "AES finalists implementation for GPU and multi-core CPU based on OpenCL", in *Proc. Int. IEEE Conf. on Anti-Counterfeiting, Secur. & Identif. ASID 2011*, Xiamen, China, 2011, pp. 38–42.

[32] T. K. Keisuke Iwai, N. Nishikawa, "Acceleration of AES encryption on CUDA GPU", *Int. J. of Netw. & Comput.*, vol. 2, no. 1, pp. 131–145, 2011.

Vincenzo Conti is an Assistant Professor with the Faculty of Engineering and Architecture at the Kore University of Enna, Italy. He received his Laurea cum Laude degree and his Ph.D. in Computer Science from the University of Palermo in 2000 and 2005, respectively. His research interests include biometric recognition systems, programmable architectures, bio-inspired processing system, and user ownership in multi-agent systems. He has joined the Editorial Board of the International Journal on Security and Communication Networks (Hindawi), of the International Journal of Biosensors and Bioelectronics (MedCrave) and of the Complex and Intensive Systems Journal (Springer), has chaired and participated at several national and international conferences, and he has co-authored over 60 scientific publications, journals and conferences. Moreover, he has participated to several research projects funded by industries and research institutes in his research areas. Currently, he collaborates with the Department of Industrial and Digital Innovation (DIID) and the Department of Biopathology and Medical Biotechnologies (DBMED) of the University of Palermo, and with Italian National Council of Researches (CNR – Cefalù, Palermo).

E-mail: vincenzo.conti@unikore.it
Faculty of Engineering and Architecture
University of Enna KORE
Via delle Olimpiadi - Cittadella Universitaria
94100 Enna, Italy

Salvatore Vitabile received the Laurea degree in Electronic Engineering and the Ph.D. degree in Computer Science from the University of Palermo in 1994 and 1999, respectively. He is currently an Associate Professor with the Department of Biopathology and Medical Biotechnologies, University of Palermo, Italy. In 2007, he was a visiting professor in the Department of Radiology, Ohio State University, Columbus, USA. Dr. Vitabile is the Editor-in-Chief of the International Journal of Adaptive and Innovative Systems (Inderscience Publishers). He has also joined the Editorial Board of the Journal of Ambient Intelligence and Humanized Computing (Springer), the International Journal of Information Technology, Communications and Convergence (Inderscience Publishers), and the International Journal of Space-Based and Situated Computing (Inderscience Publishers). He has co-authored more than 150 scientific papers in referred journals and conferences. He has chaired, organized, and served as member of the organizing committee of several international conferences and workshops. Dr. Vitabile is currently member of the Board of Directors of SIREN (Italian Society of Neural Networks). His research interests include specialized architecture design and prototyping, biometric authentication systems, driver assistance systems, and medical data processing and analysis.

E-mail: salvatore.vitabile@unipa.it
Department of Biopathology and Medical Biotechnologies
University of Palermo
Via del Vespro 129
90127 Palermo, Italy