Vectorized AES core for high-throughput secure environments

Miquel Pericàs[1,2], Ricardo Chaves[4], Georgi N. Gaydadjiev[3], Stamatis Vassiliadis[3], and Mateo Valero[1,2]

[1] Computer Sciences, Barcelona Supercomputing Center
[2] Computer Architecture Department, Technical University of Catalonia
Jordi Girona, 1-3, Mòdul D6 Campus Nord, 08034 Barcelona, Spain
Phone: +34 934 017 001, Fax: +34 934 017 055
mpericas@ac.upc.edu,mateo@ac.upc.edu
[3] Computer Engineering, Technical University of Delft
Mekelweg 4, 2628 CD Delft, The Netherlands
Phone: +31 15 2786196, Fax: +31 15 2784898
g.n.gaydadjiev@tudelft.nl,s.vassiliadis@tudeflt.nl
[4] Instituto Superior Tecnico, INESC-ID
ricardo.chaves@inesc-id.pt

Abstract. Parallelism has long been used to increase the throughput of applications that process independent data. It has been used in a broad range of levels, from functional units to large parallel clusters. With the advent of multicore technology designers and programmers are increasingly forced to think in parallel. In this paper we present the evaluation of an encryption core capable of handling multiple data streams. The design is oriented towards future scenarios for internet, where throughput capacity requirements together with privacy and integrity will be critical for both personal and corporate users. To power such scenarios we present a technique that increases the efficiency of memory bandwidth utilization of cryptographic cores. We propose to feed cryptographic engines with multiple streams to better exploit the available bandwidth. Several specific cases in which such a cryptographic engine can be successfully implemented are described. We also show how multiple interfaces – such as vector or hardware scheduling – can be used to control such engines. To validate our claims, we have developed an AES core capable of encrypting two streams in parallel using either ECB or CBC modes. Our AES core implementation consumes trivial amount of resources when a Virtex-II Pro FPGA device is targeted.

Keywords: *Parallel and Distributed Computing, Encryption*

1 Introduction

Advances in semiconductor technology have enabled industry to manufacture cores with hundreds of millions of transistors. Industry is exploiting this feature to implement chip-level parallelism in the form of multi-core on chip architectures. While 2-8 multicore chips are now common in the market it is expected

that this trend will continue with even larger amounts of cores. Programmers and designers will find themselves forced into thinking concurrently in order to efficiently exploit such platforms.

Parallelism is, of course, not a new concept and has been implemented extensively in the past. Since the earliest machines this technique has been used to improve throughput. Parallelism can be found on all levels, from the smallest circuits to parallel clusters. From the programmer point of view, there are several ways in which to express parallel programs. One class are concurrent programming models. They map directly onto multicore architectures, but have the disadvantage that they leave the parallelization to the programmer, a task which has been shown to be often quite complex. A different way to exploit parallelism is by using SIMD programming techniques. Vector processors, for example, operate on entire vectors instead of scalar types. This programming model is effective and simple, as it retains the sequential property of single-threaded programs. However, it requires data parallelism with strict organizations in memory.

In this paper we investigate how parallelization can be used to achieve high data transfer performance in future high-throughput networks. Personal users and companies are placing growing demands of security on devices they use for their daily work. Four requirements are in demand: privacy, authentication, replay protection and message integrity. For this reason, implementations of Virtual Private Networks (VPN) rely more and more on technologies such as IPsec to secure the communication links.

A technology such as IPsec can work in two modes. In *transport mode*, the endpoint computers perform the security processing. In *tunnel mode*, packet traffic is secured by a single node for the entire computer network. In case of large networks, high performance encryption devices are required. Such is also the case with *mobile* VPNs. In a mobile VPN a device such as a handheld can have secure access to a corporate LAN to securely perform such tasks as reading email or using remote terminal sessions. It is expected that this type of networks will grow very fast in popularity in the near future.

One of the most important encryption algorithms supported by many different protocols is the Advanced Encryption Standard (AES). This encryption algorithm encrypts/decrypts blocks of 128 bits of data in 10, 12 or 14 serial stages, using 128, 192 or 256 bit-keys, respectively. To simplify our study, but without loss of generality, we will be focusing only on AES using a key size of 128 bits. In this variant the algorithm performs 10 stages to encrypt/decrypt one data block.

There are several ways in which a stream of data can be encrypted. These are referred to as *Block Cypher* modes of operation. Most of these modes require an *Initialization Vector*, which is a fixed block of data used to trigger the encryption mode. The simplest mode is the *Electronic CodeBook* mode (ECB). In this mode the data stream is partitioned into blocks of equal length and all the resulting blocks are encrypted independently. The obvious benefit of this scheme is high parallelism. All blocks that make up the stream can be encrypted simultaneously. The disadvantage of this scheme are known security concerns. More precisely,

ECB does not provide good confidentiality as it does not hide data patterns well. To come up with a more robust solution several modes have been introduced. The most common of these is the *Cypher Block Chaining* mode (CBC). In this mode, when a block is going to be encrypted, it has first to be exclusively `OR`'ed with the encryption resulting from the previous block. The first block itself is `XOR`'ed with the Initialization Vector. One drawback of this scheme is the dependency between data blocks that the cypher mode introduces. This results in a reduced efficiency concerning the available bandwidth. In this mode, the AES encryption engine can only output a block of data every 10 stages. This means that only 10% of the output capacity is used. Note, however, that the interconnect itself is independent from the engine capacity and may limit the throughput.

In cases where the available network bandwidth is larger than the single-stream CBC output, we may want to search for ways to exploit the additional bandwidth. In the domain of VPN tunnels, where a gateway is in charge of encrypting large quantities of data, we can profit from the fact that multiple (independent) channels are simultaneously active to improve the throughput of the encryption.

In this paper we propose to design AES cores capable of encrypting multiple streams at once. Using multiple streams enables parallelism and allows to better exploit the available network bandwidth. This is analogous to using vector processors to better exploit memory hierarchy in supercomputers. Further, we propose to use these cores to provide high performance file transfer between computers in the case where a large file or multiple files are being transferred. In this scenario, a user using a `scp` protocol to transfer the files, would experience a large speed-up using our proposal together with a small modification of the scp application. Finally, we also perform a pencil and paper evaluation of how the proposed core can be fitted into current system architectures.

This paper makes the following contributions:

- We observe that encrypting several streams in parallel is a way to accelerate the otherwise sequential CBC encryption.
- We implement a cryptographical unit capable of encrypting two streams in parallel using the AES encryption algorithm.
- We analyze several applications of this scheme. In particular, we discuss how this scheme can accelerate VPN networks and secure transfers of large files.
- We study two important issues relating to the implementation of the multiple-stream encryption scheme: the programming model and the system architecture.

This paper is organized as follows. Section 2 presents an overview of related work. The design of the multiple-stream encryption unit is presented in section 3 while section 4 evaluates it. Section 5 analyzes the system architecture. Section 6 discusses several issues related to this design: sections 6.1 and 6.2 analyze possible applications of this work and section 6.3 analyzes the programming model. Finally, section 7 concludes this discussion.

2 Related Work

Vectorization has long been an important technique to increase performance. Vector processors handle complete vectors instead of registers as the basic type. Because no dependencies need to be tracked among the elements of a vector and because the memory system can be optimized to efficiently cater the large data amounts to the system, very high performance pipelines can be built. Vector implementations have been exploited mostly by numerical codes and scientific computing. These codes often feature large parallel loops that are well suited to be implemented on a vector processor.

However, attempts to build parallel implementations of cryptographic engines have not been very successful in the past, particularly those attempting to exploit algorithm-level parallelism. This can be explained intuitively. In order to provide a strong and hard-to-break encryption, algorithms rely on operations manipulating the whole data set and imposing tight dependencies among all data. Parallel execution would be possible in a higher level by encrypting multiple blocks in parallel, but this is in general precluded by the usage of block cypher modes such as CBC. Therefore, the few successful attempts to have parallel hardware accelerate an encryption procedure have relied on exploiting parallelism within the individual operations of the algorithm.

There are some examples of this kind of optimization in the literature. For example, Page et al. [1] used the SSE2 extensions of the Pentium4 are used to accelerate long precision modular multiplication. Similarly, Crandall et al. [2] used the AltiVec extension to implement long precision multiplications for the RSA algorithm. These two approaches are targeted at accelerating the 1024-bit multiplications frequently appearing in cryptographic algorithms. Another use of AltiVec is the approach introduced by Bhaskar et al. to accelerate Galois Field arithmetics [3]. This has been used to accelerate the AES algorithm, achieving an encryption rate of one block every 162 cycles. While this is impressive, it is far from what can be obtained with a hardware implementation like the one discussed in this paper. One final attempt at vectorization is the one proposed by Dixon et al., where a parallel approach is used to factorize large integers for Elliptic Curve Cryptography [4].

In this paper, instead of optimizing the basic operations, we propose a vector implementation for AES that exploits parallelism at the data level by processing multiple streams concurrently. The proposal is based on the MOLEN polymorphic architecture and programming paradigm [5, 6] proposed by Vassiliadis et al. as a way to expose hardware resources to software system designers and allow them to modify and extend the processor functionality at will. The outcome of this paper is a cryptographic engine that exploits multiple streams using the vector engine paradigm. The core of the cryptographic unit is based on work by Chaves et al. within the context of the MOLEN polymorphic processor [7].

3 Multiple-Stream AES Core

To validate our assumptions we implemented an AES core capable of processing two streams concurrently. The AES-MultipleStream core (AES-MS) was implemented using the MOLEN prototype framework [8, 5] and as such considers a 64-bit wide IO bus running at 100MHz. Although the width of the IO BUS has been set at 64 bits, this is not a constraint and can be adjusted to accommodate more or less data streams. The global design of the AES core with two streams can be seen in Figure 1. It consists of two independent AES cores controlled by a Control Unit. The unit activates the AES cores when needed and manages the multiplexors that control bus access.

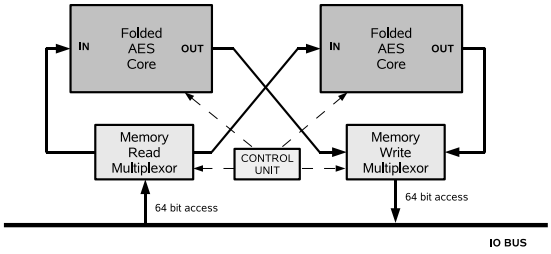


Fig. 1. Architecture of AES core handling two streams

Each core implements an independent AES folded structure [7]. On the 64-bit/100MHz bus, a single AES core takes two cycles to read 128 bits of input data and two more cycles to output 128 bits of encrypted data. The processing amounts to 10 cycles. Thus, once the core is running it moves 256 bits every 10 cycles.

Given that in 4 out of the 10 computational AES cycles the IO Bus is used to read or write data blocks, the multiple stream version has been implemented using two streams. This results in a bus occupation of 8 out of 10 cycles (80%). No more streams can be added without changing the AES pipeline depth. The folded AES cores themselves have no information on the number of active streams; this information is handled by a small external control unit that drives the AES cores and activates the necessary multiplexors to access the external memory system.

Assuming that the AES core and the IO bus run at the same frequency, it is possible to accommodate a higher or lower number of streams depending on the IO bus width. If the bus is 128 bits wide, a 128-bit data packet can be read in a single cycle and written in another one. Given that an encryption takes 10 cycles, this would allow to encrypt up to 5 streams in parallel. Generically, with the bus and engine running at the same speed, the number of streams that can

Table 1. 2-stream vs single-stream AES performance comparison

Architecture	AES – 1 stream [7]	AES – 2 streams
Cipher	Enc./Dec.	Enc./Dec.
Device	XC2VP30	XC2VP30
Number of Slices	1083	2162
Number of BRAM	12	24
Operating Frequency	100 MHz	100 MHz
Latency (cycles)	10	10
Throughput (Mbps)	1280	2560
Throughput/Slice (Mbps/s)	1.1819	1.1841

be accommodated as a function of the bus width is expressed by:

$$MaxNStreams = \lfloor 5 \cdot \frac{BusWidth}{128bits} \rfloor. \quad (1)$$

4 Performance and Results

The complete design of the two-stream AES unit was implemented in VHDL targeting the Virtex-II Pro xc2vp30-7fg676 device. Synthesis and Place & Route were both performed using Xilinx ISE 8.1. The AES core for single stream [7] spans 12 BlockRAMs and 1083 logic slices. The two-streams AES core spans 24 BlockRAMs and 2162 logic slices. The two-streams implementation puts two single-stream AES cores side-by-side and adds multiplexors that arbitrate the memory access. Some logic is shared and in the end the number of logic slices approximately doubles. Place & Route results show that the design can run at 100MHz which is the target frequency of the current MOLEN prototype. The two-streams AES-MS core consumes 17% of available BlockRAMs, 15% of all logic slices, and 38% of external IOBs while reaching a throughput of 2.56 Gbps. Table 1 shows a summary of the results and compares them against [7]. Note that the numbers provided for the original implementation of the AES core differ from those provided in [7]. This is due to some different parameters that have been used in the synthesis environment.

It should be noted that the initialization of the AES core, which includes the transmission of the key, the transmission of the initialization vector and the processor↔co-processor communication overhead, has a cost on performance. If only one data block is ciphered, the cost of initializing the AES core is an order of magnitude higher than the cost of processing the data itself. When the input data is sufficiently large, the initialization cost becomes negligible. The ciphering throughput varies from 60 Mbps for a single data block packet (128 bits) to 1.28 Gbps for a 16 kbyte packet.

5 Analysis of System Architecture

We will now present an evaluation of different system environments in which the AES core may be implemented together with performance estimations. The following cases of IO communication have been considered: the current MOLEN prototype, HyperTransport eXpansion (HTX), PCI-X, and PCI-Express (PCIe).

As mentioned earlier, the AES core has been developed and tested within the MOLEN environment. In this platform, the two-stream AES core runs at 100MHz and can encrypt and decrypt at a rate of 2.56Gbps. Considering input and output this amounts to a total traffic of 5.12Gbps, which corresponds to 80% of the total memory bandwidth in this scenario. In the following study we will assume an AES-MS core running at 100MHz, even though the busses themselves are operated at different frequencies. We assume some sort of hardware performs the interfacing without loss of capacity.

Recently, a protocol that has emerged with good support for reconfigurable devices as coprocessors is the *point-to-point* HyperTransport protocol [9, 10]. HyperTransport defines an extension protocol for coprocessors called the HyperTransport eXpansion (HTX). In the current incarnation, this standard defines a protocol that is 16 bits wide and runs at 800MHz. The bandwidth provided by a single link in single-data rate (SDR) is thus 12.8Gbps. Using two links at double-data rate (DDR) yields the maximum aggregate bandwidth of 51.8 Gbps. The single link SDR bandwidth is exactly twice that which is available in the current MOLEN prototype. Without changing the frequency of the AES core (100MHz) one could double the amount of streams (4 streams, 10.24Gbps). The remaining 2.56 Gbps are exactly the bandwidth required for one additional stream so it is possible to add a 5th stream and thus run a 5-stream AES-MS core attached to a HTX interface. Using the two HTX links with DDR would enable to accommodate up to 20 streams. Note that in this analysis the AES core and the bus operate at different frequencies. Thus we must calculate the number of streams based on available bandwidth rather than using the formula presented in section 3.

PCI-X [11] is a popular multidrop bus interconnect standard. PCI-X 1.0 features a maximum bandwidth of 8.48 Gbps at speed grade PCI-X 133, which would allow up to three streams using the AES-MS engine. A newer revision of this standard, called PCI-X 2.0, has a maximum speed grade of PCI-X 533 resulting in a bandwidth of 34.4 Gbps. This can accommodate up to 13 streams in parallel.

PCI Express (PCIe) [12] is yet another bus designed to subsitute the ancient PCI bus. Like HTX, it is a point-to-point bus, but designed to manage a wider range of devices. As a downside, it operates with slightly larger latencies. At 64 Gbps capacity (using 16 links) PCIe 1.0 would allow to interleave up to 25 streams. PCIe 2.0 runs twice as fast and would be able to accommodate up to 50 streams at maximum throughput.

All these numbers may seem quite high. However, if the network capacity is not as large, the AES-MS output capacity will be underutilized. In addition, as already pointed out at the end of section 4, if keys are not static and the amount

Table 2. Maximum Number of Streams using 100MHz AES-MS cores

Interconnect Type	Max Bandwidth	Max Number of Streams
MOLEN Prototype	6.4 Gbps	2
HTX @ 1 Link (SDR)	12.8 Gbps	5
HTX @ 2 Links (DDR)	51.2 Gbps	20
PCI-X 133 (v1.0)	8.48 Gbps	3
PCI-X 533 (v2.0)	34.4 Gbps	13
PCIe 1.0	64 Gbps	25
PCIe 2.0	128 Gbps	50

of data is not sufficiently large, throughputs of Gbit/s cannot be reached as the encryption processes will be limited by the initialization phase. The previous results are summarized in Table 2. Note that in this table, *Max Bandwidth* refers to the maximum bandwidth of the interconnect, not the maximum bandwidth of the multiple stream encryption unit. Although we have not mentioned access latencies for these technologies, we assume that in stationary mode the effects of these latencies are negligible.

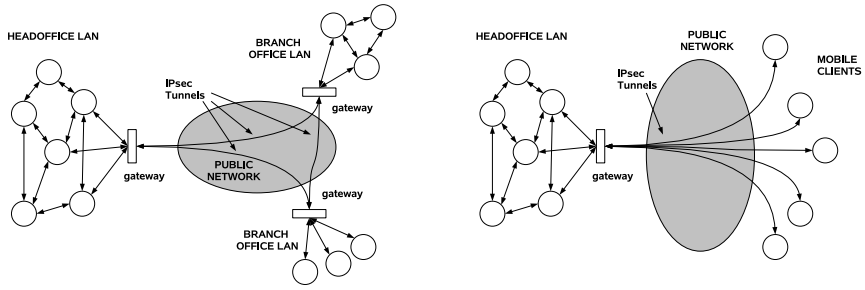
6 Discussion

In this section we discuss various issues related to AES-MS. So far we have implemented a core capable of exploiting multiple streams. We will now present some scenarios that can profit from the implementation and a programming model to exploit the multiple-streams feature.

6.1 Virtual Private Networks

Figure 2 (a) shows the typical architecture for a virtual private network (VPN) using unreliable connections, e.g. Internet. Such an architecture is used to securely connect multiple networks. Locally, the networks can be considered secure since the infrastructure belongs to the companies/institutions. However, on the public infrastructure no such assumptions can be made. Privacy and authentication support are required. To this end, encrypted tunnels are established. The tunnels are authenticated when a session is established. Once established, the session is kept mostly unmodified and the same keys are used to encrypt all packets.

Figure 2 (b) shows the same scenario in the case of a mobile VPN. In a mobile VPN the connection is not *network-to-network* but *client-to-network*. Every client needs to have security software installed (e.g. its own IPsec stack). The corporate side, however, looks fairly similar to the static VPN case. From the point of view of the gateway, a mobile VPN will generate many more tunnels, each of which moving a smaller quantity of data. Also, in a mobile VPN there is much higher connect/disconnect activity.



(a) (b)

Fig. 2. Sample Architecture for a static VPN (a) and for a mobile VPN (b). In both cases the gateways may need to support high encryption throughput.

In both cases the VPN gateways may require enormous encryption throughput. Using AES-MS on the gateways may enable these requirements. Implementation details may vary a little for both cases of VPN. In *static* VPNs the keys are mostly static. This means that a single trusted key could be used to encrypt multiple communications inside a single *gateway→gateway* channel. From the point of view of a multiple-stream encryption core this has the benefit that the key need not be replicated. But this would imply that more ports are needed into the key register. Adding simple circuitry, it is possible to read the register only once and route the key segments to the corresponding encryption engine without increasing the number of ports. For an architecture in which the encryptions proceed synchronously this technique is trivial; however, in our case, multiple encryptions are performed in parallel but in different iterations. A different strategy is needed in order to maintain the *read only once* property. There are two ways to proceed. One option is to store each segment of the key in a different register. Every engine reads the corresponding key segment from the corresponding register every cycle. Alternatively, we can reduce the number of reads by having the segments read once and then routed to the engine through an appropriate number of latches.

These techniques are easily implemented in ASIC technology; however, when using FPGAs there are additional constraints. For our AES-MS implementation on the Virtex2P this optimization was not readily available due to fact that the base implementation [7] is already optimized to store the full key register in a single BlockRAM using both available ports. However, it may be possible to implement this technique using multiple BlockRAMs of 128 bits. This would then allow to store the complete key schedule and to access the portions independently. However, this particular implementation also consumes many more BlockRAMs, a feature which is undesirable.

In the case of a mobile VPN the technique of sharing the key register is unlikely to result in any benefits. In this environment every client is associated to

a different tunnel and each tunnel has its own key, so the gateway cannot share them. Nevertheless, making use of multiple streams is still effective as the aggregate bandwidth of all streams may be very large and serializing the encryption of the packets could otherwise result in network communication degradation.

6.2 Secure File Transfers

When citing VPN we commented that having multiple streams is a key condition to enable our vectorized AES implementation. We now present a particular but still common scenario in which having an AES-MS core can greatly benefit the user.

Transferring files among computers is one of the most common tasks happening on the internet. In general, bulk file transfers can take a long time as they may consist of very large files such as backups, media files, software distributions, etc., being sent to some remote computer. To avoid serialization in this scenario we propose to implement a specialized transfer protocol that opens several tunnels and encrypts multiple parts simultaneously. A single file is subdivided into chunks and sent as multiple files through different channels. This could be done, for example, on modified versions of the `scp` or `sftp` protocols. Figure 3 shows how this would look in the case of a parallel transfer of a single file subdivided into chunks. AES-MS can then be used to accelerate the whole operation.

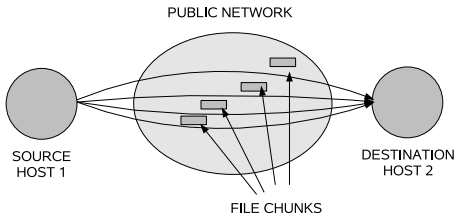


Fig. 3. Parallel file transfer using multiple channels by subdivision of a file into multiple chunks

6.3 Vector Programming Models

So far we have mentioned the application of our technique to gateways in VPN environments but have not commented about the architecture of the gateway itself. There are various levels in which the multiple-stream technique can be implemented. In a pure network device implementation it could be a hardware-only implementation. In this case the gateway just requires a peripheral board

with the encryption engine, but this comes at the cost of versatility. The gateway can also be implemented in a higher level using a special programming interface to the device.

Vector architectures provide a special ISA interface in which vector registers can be manipulated as regular registers. The addition of vector loads and stores allows the memory controller to efficiently schedule memory access instructions and better exploit memory bandwidth. Our multiple-stream interface follows an equivalent goal. From the programmers point of view, the AES-MS engine may be programmed as a vector device. Sending multiple unrelated files through input/output channels in a single system call is known as *Vector I/O* or *scatter/gather*. In Figure 4 we show how multiple streams could be encrypted using scatter/gather. The key element of *scatter/gather* is a data structure that holds a vector of data buffers and a corresponding vector with the sizes of each data buffer. The system then reads this data structure and schedules the I/O accesses to the different data buffers in order to maximize system performance. In the example, an AES-MS core with two streams is about to process two input data buffers: inA and inB. The system reads the two buffers and interleaves them in blocks of 128 bits. Once this interleaving is done the joint stream can be fed to the AES-MS core for processing. After encryption, the procedure is inverted to store the encrypted data in the corresponding output buffers.

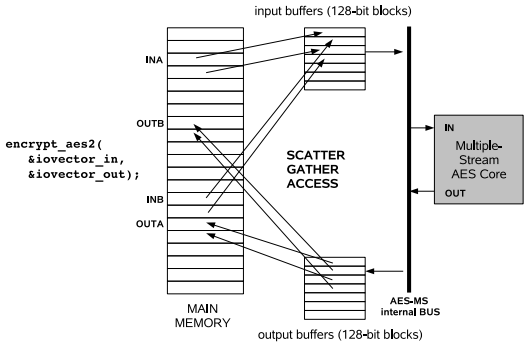


Fig. 4. A simplified Scatter-Gather Interface to Multiple-Stream AES

7 Conclusions

In this paper we have described an AES unit capable of processing multiple data streams. Like Vector Engines, our AES unit uses vectors of data to efficiently exploit the external IO bandwidth. The proposed technique can be used to improve throughput in important scenarios such as Virtual Private Networks

(VPN) or secure file transfer where large quantities of data are being transferred. We have presented characteristics of the design and proposed a possible programming interface together with possible system architectures for using the core as a coprocessor. The use of the Molen paradigm and the systems reconfigurability, allows to extrapolate these results to other encryption cores. Also, the flexible and modular structure of the used multi-stream AES core allows for an easy integration of additional processing streams, if a higher bandwidth IO bus is used. The bandwidth is being limited by the IO bus and its conservative frequency value, used in the implemented prototype.

Acknowledgments

This work was supported by the HiPEAC European Network of Excellence under contract IST-004408, by the Ministerio de Educación y Ciencia of Spain under contract TIN–2004–07739–C02–01 and by the Portuguese FCT–Fundação para a Ciência e Tecnologia.

References

1. Page, D., Smart, N.P.: Parallel cryptography arithmetic using a redundant montgomery representation. IEEE Trans. Comput. **53** (2004) 1474–1482
2. Crandall, R., Klivington, J.: Vector implementation of multiprecision arithmetic. Technical report, Apple Computer Inc. (1999)
3. Bhaskar, R., Dubey, P.K., Kumar, V., Rudra, A.: Efficient Glois Field Arithmetic on SIMD Architectures. In: Proc. of the 15th Annual ACM Symp. on Parallel Algorithms and Architectures. (June 2003) 256–257
4. Dixon, B., Lenstra, A.K.: Massively Parallel Elliptic Curve Factoring. In: Proc. of the Workshop on the Theory and Application of of Cryptographic Techniques. (1992) 183–193
5. Vassiliadis, S., Wong, S., Gaydadjiev, G., Bertels, K., Kuzmanov, G., Panainte, E.M.: The MOLEN Polymorphic Processor. IEEE Trans. Comput. **53**(11) (November 2004) 1363–1375
6. Vassiliadis, S., Gaydadjiev, G.N., Bertels, K., Panainte, E.M.: The Molen Programming Paradigm. In: Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation. (July 2003) 1–10
7. Chaves, R., Kuzmanov, G., Vassiliadis, S., Sousa, L.: Reconfigurable Memory Based AES Co-Processor. In: Proc. of the 13th Reconfigurable Architectures Workshop (IPDPS). (January 2006)
8. Kuzmanov, G., Gaydadjiev, G.N., Vassiliadis, S.: The MOLEN Processor Prototype. In: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004). (April 2004) 296–299
9. : HTX Electrical and Operational Profile. Technical report, The HyperTransport Consortium
10. : HyperTransport HTX: Extending Hypertransport Interconnect Leadership. Presentation, The HyperTransport Consortium (2007)
11. : PCI-X 2.0 Overview. Presentation, PCI SIG
12. : PCI-SIG - PCI Express Base 2.0 Specification. Technical report, PCI SIG