VECTORIZED AES CORE FOR HIGH-THROUGHPUT SECURE ENVIRONMENTS

Miquel Pericàs, Mateo Valero
Computer Sciences
Barcelona Supercomputing Center
{miquel.pericas,mateo.valero}@bsc.es

Ricardo Chaves
Instituto Superior Técnico
IST/INESC-ID
ricardo.chaves@inesc-id.pt

Georgi N. Gaydadjiev, Stamatis Vassiliadis
Computer Engineering Laboratory
Technical University of Delft
{g.n.gaydadjiev,s.vassiliadis}@tudelft.nl

ABSTRACT

Parallelism has long been known to increase the throughput of applications that process independent data. It has been used in a broad range of designs, from functional units to large parallel supercomputers. With the advent of multicore technology designers and programmers are increasingly forced to think in parallel. It is clear that parallelism will be key to the implementation of efficient computing systems in the future. 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. We describe some specific cases in which such a cryptographic engine can be successfully implemented. We also show how multiple interfaces – such as vector or hardware scheduling – can be used to control such engines. To validate our claim, 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 Virtex-II Pro FPGA device is targeted.

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 not a new concept and has been studied 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 the largest parallel supercomputers. From the programmer point of view, there are several ways in which to express parallel programs. One class of models are the 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 in the form of 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 conditions 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 the case with *mobile* Virtual Private Network (VPN) networks. 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 standards is the so-called Advanced Encryption Standard (AES). This encryption algorithm encrypts/decrypts a block of 128-256 bits of data in 10-14 consecutive stages using 128-256 bit keys. To simplify our study, but without loss of generality, we will be focusing only on AES-128 using a block 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 the *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 data parallelism. All blocks that make up the stream can be encrypted simultaneously. A big disadvantage of this scheme are the well 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 other modes have been introduced. The most common of these is probably 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 result of the previous block. The first block itself is XOR'ed with the Initialization Vector. The main drawback of this scheme is that the encryption process is serialized. 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 cycles (assuming that a single stage takes one cycle). This means that only 10% of the output capacity can be used. Note, however, that the network bandwidth itself is independent from the engine capacity and may preclude the use of the full capacity.

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 super-computers. Further, we propose to use these cores to provide high performance file transfer between computers in the case where multiple files are being transferred. In this scenario, a user using the `scp` protocol to transfer several files, will experience a large speed-up using our AES core together with a small modification of the scp application. Finally, we 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 AES.
- 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 previous work performed by our groups and cites the motivation for the present work. The design of the multiple-stream encryption unit is presented in section 3 while section 4 evaluates the design. Section 5 discusses several issues related to this design: sections 5.1 and 5.2 analyze possible applications of this work; section 5.3 analyzes the programming model and section 5.4 analyzes the system architecture. Finally, section 6 concludes this discussion.

2. MOTIVATION AND PREVIOUS WORK

This work builds upon previous work performed by the HPC group of the Computer Architecture Department at UPC, Spain, and the Computer Engineering Laboratory at TU Delft, the Netherlands. Both groups have a long trajectory on vector architecture research in the past.

The work in Barcelona started with the proposal of a vector memory architecture capable of accessing vector elements out–of–order [1]. This technique allows vector elements to be accessed avoiding conflicts in the memory modules. The technique was later extended to multiprocessor systems [2], allowing to reduce the complexity of the network interconnect. The concept has been completed with the mechanism known as *Command Vector Memory* [3]. This mechanism provides a degree of intelligence to the memory system, allowing the memory controller to handle complete vector data sets from a single command. This allows the cost and power of the network to be reduced while increasing overall throughput. All these concepts have been joined in a proposal to accelerate numerical applications [4]. Vector concepts have also been used to design high–speed packet switching networks! [5].

The HPC group in Barcelona has also worked on the proposal and evaluation of advanced vector processors. The following architectures have been proposed: Out–of–order vector architectures [6], Decoupled Vector Architectures [7], Multithreaded Vector Architectures [8] and combinations of them [9]. As an optimization, new cache memory schemes as well as improvements for the register file have been proposed [10]. The Cray SX-1 supercomputer internally uses a decoupled vector architecture design, conceptually similar to [7]. The out–of–order vector architecture proposed in [6] was implemented by Compaq in their Tarantula proposal [11]. Finally, modern architectures such as the Cell processor [12] make use of both vector instructions and multithreading. Alth! ough the implementation varies considerably such a combination had already been studied in [8].

The design philosophy of vector processors is well suited for the acceleration of multimedia applications. Professor Valero's HPC group has proposed some architectures with this goal in mind. The first one was an MMX–like architecture [13]. Afterwards, a multi–threaded architecture was presented for the execution of applications such as MPEG-4 [14]. Moving one step further, this group presented an architecture that included 3-D memory instructions [15]. Also oriented towards multimedia applications, but following a completely different path, the concept of fuzzy processing was introduced [16]. This technique makes use of the natural redundancy of the algorithms to increase performance and reduce power consumption.

Finally, the HPC group in Barcelona has proposed to apply dynamic vectorization techniques, which are then added to superscalar processors to augment the performance of integer applications [17].

In Delft, Cheresiz et al. studied the *complex streamed instruction* (CSI) set [18], where a single instruction can process vector data streams of arbitrary length and stride and which combines complex memory accesses (with implicit prefetching), program control for vector sectioning, and complex computations on multiple data in a single operation. This technique eliminates overhead instructions and allows to improve performance.

Computer Engineering was among the first to recognize the importance of handling both sparse and dense matrix format in an uniform way [19]. Several new formats for sparse matrix representation, dedicated vector functional units and novel multiplication algorithms were proposed [20, 21, 22]. In addition, novel techniques to handle normal and transposed sparse matrix by vector multiplication [23], hierarchial sparse matrix storage formats [24] and a special benchmark suite [25] were also introduced.

The MOLEN polymorphic architecture [26] and programming paradigm [27] were proposed by Computer Engineering in Delft to expose hardware resources to the software system designers and allow them to modify and extend the processor functionality at will. This architecture was successfully applied to problems as high-performance dense matrix multiplication [28] and dedicated high-efficiency memory hierarchies [29, 30, 31].

Both groups combined the above research directions in the context of the HiPEAC network of excellence and performed the work reported in this paper. The outcome is a cryptographic engine that exploits multiple streams using the vector engine paradigm. The core of the cryptographic unit [32] is based on work by Chaves et al. developed in the context of the MOLEN polymorphic processor [33].

3. MULTIPLE-STREAM AES CORE

To validate our assumptions we implemented an AES core capable of processing two streams concurrently. The AES-MultiStream core (AES-MS) was implemented using the MOLEN prototype framework [34] 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.

Each core implements an independent AES folded structure [32]. 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.

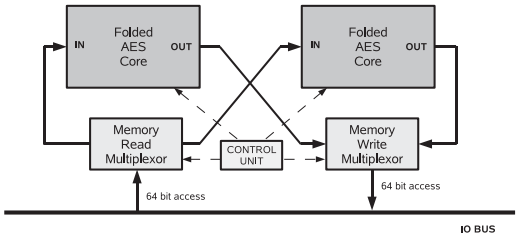


Fig. 1. Architecture of AES core handling two streams

Given that in 4 out of the 10 computational AES cycles the IO Bus is used to read or write the data blocks, 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, the number of streams that can be accommodated, as a function of the bus width is expressed by:

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

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 [32] spans 12 BlockRAMs and 1083 logic slices. The two-streams AES core spans 24 BlockRAMs and 2162 logic slices. The two-streams version 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 [32]. Note that the numbers provided for the original

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

Architecture	AES – 1 stream [32]	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

implementation of the AES core differ from those provided in [32]. This is due to different parameters that have been used

in the synthesis environment. In addition, AES-MS has not yet been optimized for frequency so we report only data for the 100MHz implementation developed on top of the MOLEN prototype.

It should be noticed that the initialization of the AES core, which includes the transmission of the key and the initialization vector, has a cost in the performance. If only one data block is ciphered, the cost of initializing the AES core is higher than the cost of processing the data itself. When the data packet is sufficiently big, 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. This mean that, when only one AES core is used and several streams have to be encrypted, smaller packets are processed by the single-stream AES core, in order to maintain a low processing latency. When the multiple-stream AES core is used, bigger data packet can be created, improving the ciphering throughput.

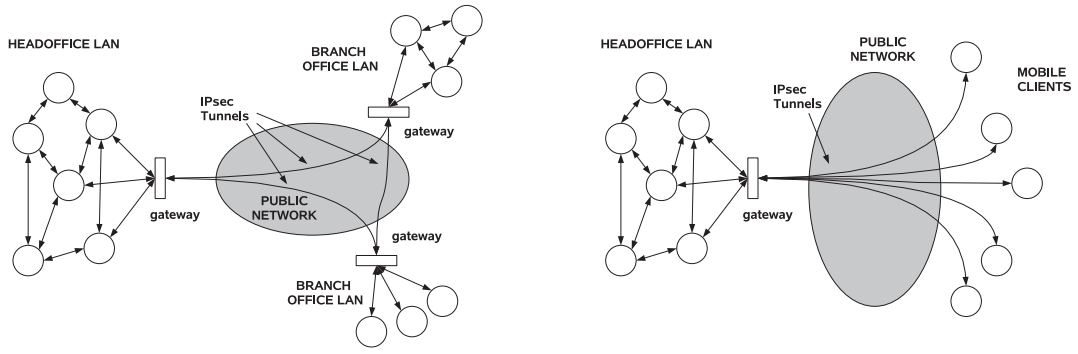
5. DISCUSSION

In this section we discuss various issues related to AES-MS. So far we have implemented a core capable of exploiting multiple streams. We now want to analyze such issues as:

- Who can profit from this implementation? (Sections 5.1 and 5.2)
- How could the new functionality be accessed? (Section 5.3)
- Which interconnect/system architecture should be chosen for a realization of this scheme? (Section 5.4)

5.1. 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 lines belong 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.



(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.

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.

In both cases the VPN gateways may require enormous encryption throughput. Using AES-MS on both sides 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 [32] 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. Consequently, this particular implementation also consumes many more BlockRAMs, a feature which is highly 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.

5.2. 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 between 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 or movies that are being sent to some remote server. To avoid serialization in this scenario we propose to implement a specialized transfer protocol that opens several tunnels and encrypts many files simultaneously. If only one file is being transferred, but the transmission overhead is large, the file can be subdivided in 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 file subdivided into chunks.

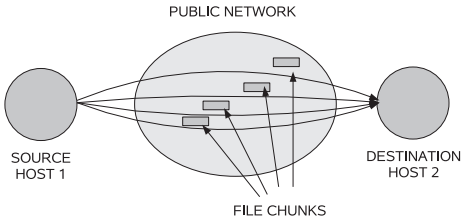


Fig. 3. Parallel file transfer using multiple channels by subdivision of a file into multiple chunks

5.3. 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 an implementation can be almost hardware-only. 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 more versatility in the scheduling of memory access instructions. This allows for a more efficient exploitation of the 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, a 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.

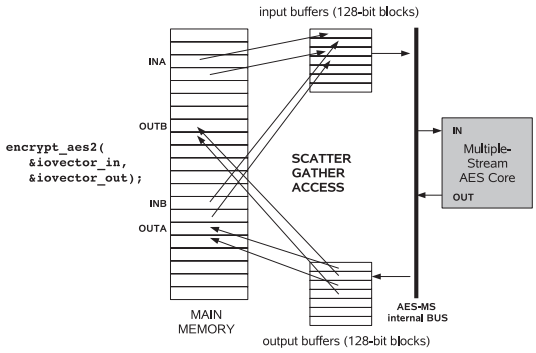


Fig. 4. A simplified Scatter-Gather Interface to Multiple-Stream AES

5.4. System Architecture

Finally, we 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 will be 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 buses 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 [35, 36]. 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 we are assuming that the AES core works at a fixed 100 MHz. Thus we must calculate the number of streams based on available bandwidth rather than using the formula presented in section 3.

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

PCI-X [37] 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) [38] 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 has already been pointed at at the end of section 4, if keys are not static and the amount of data is not sufficiently large, high throughputs cannot be reached as the encryption processes will be limited by the encryption core 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. 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 use. 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 and by the Ministerio de Educación y Ciencia of Spain under contract TIN–2004–07739–C02–01

7. REFERENCES

[1] M. Valero, T. Lang, J. M. Llaberia, M. Peiron, and J. J. Navarro, “Increasing the Number of Strides for Conflict-Free Vector Access,” in *Proc. of the 19th Intl. Symp. on Computer Architecture*, May 1992.

[2] M. Peiron, M. Valero, E. Ayguade, and T. Lang, “Vector Multiprocessors with Arbitrated Memory Access,” in *Proc. of the 22nd Intl. Symp. on Computer Architecture*, June 1995, pp. 243–252.

[3] J. Corbal, R. Espasa, and M. Valero, "Command-Vector Memory Systems," in *Proc. of the 7th Intl. Conf. on Parallel Architectures and Compilation Techniques*, November 1998.

[4] F. Quintana, J. Corbal, R. Espasa, and M. Valero, "Adding a Vector Unit to a Superscalar Processor," in *Proc. of the 13th Intl. Conf. on Supercomputing*, June 1999, pp. 1–10.

[5] J. Garcia, J. Corbal, L. Cerda, and M. Valero, "Design and implementation of high-performance memory systems for future packet buffers," in *Proc. of the 36th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, December 2003, pp. 372–384.

[6] R. Espasa, M. Valero, and J. E. Smith, "Out-of-order vector architectures," in *Proc. of the 30th Annual Intl. Symp. on Microarchitecture*, December 1997, pp. 160–170.

[7] R. Espasa and M. Valero, "Decoupled vector architectures," in *Proc. of the 2nd Intl. Symp. on High Performance Computer Architecture*, February 1996, pp. 281–290.

[8] ——, "Multithreading Vector Architectures," in *Proc. of the 3rd Intl. Symp. on High Performance Computer Architecture*, February 1997, pp. 237–248.

[9] ——, "Exploiting Instruction and Data-Level Parallelism," *IEEE Micro*, vol. 17, no. 5, pp. 20–27, September/October 1997.

[10] ——, "A Victim Cache for Vector Registers," in *Proc. of the 11th Intl. Conf. on Supercomputing*, July 1997.

[11] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec, "Tarantula: a Vector Extension to the Alpha Architecture," in *Proc. of the 29th Annual Intl. Symp. on Computer Architecture*, June 2002, pp. 281–292.

[12] D. Pham., T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 179–196, January 2006.

[13] J. Corbal, M. Valero, and R. Espasa, "Exploiting a new level of DLP in multimedia applications," in *Proc. of the 32nd Annual Intl. Symp. on Microarchitecture*, December 1999, pp. 72–79.

[14] J. Corbal, R. Espasa, and M. Valero, "DLP + TLP Processors for the Next Generation of Media Workloads," in *Proc. of the 7th Intl. Conf. on High-Performance Computer Architectures*, January 2001.

[15] ——, "Three-dimensional memory vectorization for high bandwidth media memory systems," in *Proc. of the 35th Annual Intl. Symp. on Microarchitecture*, December 2002, pp. 149–160.

[16] C. Alvarez, J. Corbal, and M. Valero, "Initial Results on Fuzzy Floating Point Computation for Multimedia Processors," *IEEE TCCA Letters*, January 2002.

[17] A. M. Pajuelo, A. Gonzalez, and M. Valero, "Speculative Dynamic Vectorization," in *Proc. of the 29th Intl. Symp. on Computer Architecture*, May 2002.

[18] D. Cheresiz, B. Juurlink, S. Vassiliadis, and H. A. G. Wijshoff, "The CSI multimedia architecture," *IEEE Trans. VLSI Syst.*, vol. 13, no. 1, pp. 1–13, January 2005.

[19] S. Vassiliadis, S. D. Cotofana, and P. Stathis, "Vector isa extension for sprase matrix multiplication," in *Proceedings of EuroPar'99 Parallel Processing*, September 1999, pp. 708–715.

[20] ——, "Block based compression storage expected performance," in *Proc. 14th Int. Conf. on High Performance Computing Systems and Applications (HPC 2000)*, June 2000, pp. 389–406.

[21] ——, "Bbcs based sparse matrix-vector multiplication: initial evaluation," in *Proc. 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, August 2000, pp. 1–6.

[22] P. Stathis, S. D. Cotofana, and S. Vassiliadis, "Sparse matrix vector multiplication evaluation using the bbcs scheme," in *Proc. of 8th Panhellenic Conference on Informatics*, November 2001, pp. 40–49.

[23] S. D. Cotofana, P. Stathis, and S. Vassiliadis, "Direct and transposed sparse matrix-vector multiplication," in *Proceedings of the 2002 Euromicro conference on Massively-parallel computing systems, MPCS-2002*, April 2002, pp. 1–9.

[24] P. Stathis, S. Vassiliadis, and S. D. Cotofana, "A hierarchical sparse matrix storage format for vector processors," in *Proceedings of IPDPS 2003*, April 2003, p. 61a.

[25] ——, "D-sab: A sparse matrix benchmark suite," in *Proceedings of 7th International Conference on Parallel Computing Technologies (PaCT 2003)*, September 2003, pp. 549–554.

[26] S. Vassiliadis, S. Wong, and S. D. Cotofana, "The molen $\rho\mu$-coded processor," in *in 11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, August 2001, pp. 275–285.

[27] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. M. Panainte, "The molen programming paradigm," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2003, pp. 1–10.

[28] Y. Dou, S. Vassiliadis, G. Kuzmanov, and G. N. Gaydadjiev, "64-bit floating-point fpga matrix multiplication," in *ACM/SIGDA Thirteenth International Symposium on Field Programmable Gate Arrays (FPGA 2005)*, February 2005, pp. 86–95.

[29] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "Visual data rectangular memory," in *Proceedings of the 10th International Euro-Par Conference (Euro-Par 2004)*, September 2004, pp. 760–767.

[30] G. N. Gaydadjiev and S. Vassiliadis, "Flux caches: What are they and are they useful?" in *Proceedings of the 5th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2005)*, July 2005, pp. 93–102.

[31] ——, "Sad prefetching for mpeg4 using flux caches," in *Proceedings of the 6th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2006)*, July 2006, pp. 248–258.

[32] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Souza, "Reconfigurable Memory Based AES Co-Processor," in *Proc. of the 13th Reconfigurable Architectures Workshop (IPDPS)*, January 2006.

[33] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN Polymorphic Processor," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1363–1375, November 2004.

[34] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "The molen processor prototype," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, April 2004, pp. 296–299.

[35] "HTX Electrical and Operational Profile," The HyperTransport Consortium," Online Materials. [Online]. Available: http://www.hypertransport.org/tech/tech_htx_eop.cfm?m=9

[36] "HyperTransport HTX: Extending Hypertransport Interconnect Leadership," The HyperTransport Consortium," Presentation, 2007. [Online]. Available: http://www.hypertransport.org/docs/pres/HTX_&_Torrenza_01-28-07.pdf

[37] "PCI-X 2.0 Overview," PCI SIG," Presentation. [Online]. Available: http://www.pcisig.com/specifications/pcix_20/pci_x_2dot0_presentation.pdf

[38] "PCI-SIG - PCI Express Base 2.0 Specification," PCI SIG," Online Materials. [Online]. Available: http://www.pcisig.com/specifications/pciexpress/base2/