Hardware-Software Codesign of a Vector Co-processor for Public Key Cryptography

Jacques Jean-Alain Fournier, Simon Moore

Hardware-Software Codesign of a Vector Co-processor for Public Key Cryptography

Jacques J.A. Fournier

University of Cambridge, Computer Laboratory, UK

`Jacques.Fournier@cl.cam.ac.uk`

Gemplus S.A, La Ciotat, France

`Jacques.Fournier@gemplus.com`

Simon Moore

University of Cambridge, Computer Laboratory, UK

`Simon.Moore@cl.cam.ac.uk`

Abstract

Until now, most cryptography implementations on parallel architectures have focused on adapting the software to SIMD architectures initially meant for media applications. In this paper, we review some of the most significant contributions in this area. We then propose a vector architecture to efficiently implement long precision modular multiplications. Having such a data level parallel hardware provides a circuit whose decode and schedule units are at least of the same complexity as those of a scalar processor. The excess transistors are mainly found in the data path. Moreover, the vector approach gives a very modular architecture where resources can be easily redefined. We built a functional simulator onto which we performed a quantitative analysis to study how the resizing of those resources affects the performance of the modular multiplication operation. Hence we not only propose a vector architecture for our Public Key cryptographic operations but also show how we can analyze the impact of design choices on performance. The proposed architecture is also flexible in the sense that the software running on it would offer room for the implementation of counter-measures against side-channel or fault attacks.

1 Introduction

The use of sophisticated cryptography has been widely deployed in our everyday life. In 'conventional' computers (i.e. in the non-embedded world) crypto-oriented hardware is a rare commodity unlike media applications for which dedicated parallel architectures have been developed (e.g., the MMX architecture [21] for the Pentium family or the AltiVec co-processor for the PowerPC). A rare counter-example to this is the Sparc processor [9] where special instructions have been deployed.

On the other hand, for the embedded world (particularly for smart-cards), given the constraints of speed, power, size and security, special cryptographic accelerators have been deployed. Most of those Public Key (PK) crypto-accelerators propose very elaborate arithmetic processors that work on long precision numbers of fixed lengths, resulting in complicated, bulky and inflexible architectures. Others have been trying to have a more general approach by enhancing the instruction set of general purpose scalar processors [13]. However, none of those approaches have embraced a hardware-software co-design approach for data level parallel techniques to enhance cryptographic computations.

We begin this paper by performing an extensive study about how cryptography has been implemented on SIMD (Single Instruction Multiple Data) architectures. We then give a rapid description of the vector architecture presented in [12]. We show how a design-to-cost approach can be adopted by doing a quantitative analysis on the functional simulation of a modular multiplication operation. We finally summarize our results and compare our work to previous contributions in the field of cryptography.

2 Parallel implementations of cryptography

In the 'conventional' or non-embedded computing world, most of the research has concentrated around parallelizing the cryptographic operations in order to take advantage of the SIMD architecture originally developed for media applications:

- In [20], the authors implement a long precision modular multiplication on a Pentium4 using the SSE2 (*Streaming SIMD Extensions 2*) instructions. The authors execute four exponentiations in parallel, each exponentiation being implemented using a Redundant Representation of Montgomery's multiplication. The authors report that a 1024-bit modular multiplication takes $60\mu s$, which roughly corresponds to 120000 clock cycles for a 2GHz Pentium4 processor.

- Crandall and Klivington illustrate in [7] how the Velocity Engine of the PowerPC can be used to implement long precision multiplications for RSA. Based on the figure given in the paper, we can infer that a 1024-bit multiplication takes about 3600 clock cycles with their approach. However, no figures were reported for a full modular multiplication.

- The AltiVec [8] extension to the PowerPC was originally developed to target media applications. This vector extension is made of 32 128-bit vector registers. AltiVec also offers some superscalar capabilities since instructions belonging to different 'classes' can be executed in parallel. Galois Field arithmetics has been implemented on the AltiVec in [2]. In the latter paper, the authors show how the Rijndael algorithm [19] can be made to execute in 162 clock cycles on the AltiVec or, even better, in only 100 clock cycles if a bit-sliced approach is used.

For embedded applications, studies around the use of SIMD architectures for cryptography are even more scarce:

- In the embedded world, Data Parallel architectures are mostly deployed in DSPs (*Digital Signal Processors*) for signal processing. In [15], the authors present how modular multiplication based on Montgomery's method [18] can be implemented on a TMS320C6201 [26]. With their approach, a 1024-bit RSA verification (with $e = 2^{16} + 1$) takes 1.2ms. If we agree that we need approximately 17 modular multiplications for this, then, with a processor clock at 200MHz, we can infer that the one 1024-bit modular multiplication takes about 14000 clock cycles. Applying the same reasoning to other data given in the paper, we find out that one 2048-bit modular multiplication takes 53000 clock cycles on this architecture.

- In the fascinating world of smart-cards, the only examples where parallel approaches have been reported are in [16, 10]. In both papers, the authors focus on fast elliptic curve multiplications. In [16], the authors show how, with a projective coordinates representation [3], calculations can be parallelized on the Crypto2000. On the other hand Fisher *et al* [10] focus more on elliptic curve implementations resistant to side channel attacks.

3 Our vector approach

In all the work reported in Section 2, we have seen how long precision modular multiplications were adapted to SIMD architectures. However, none of those approaches actually studies how a data parallel hardware could be tailored for cryptographic operations. In the example given in [16, 10], even if there are two units working in parallel, the data itself is not decomposed, nor is it very clear whether the underlying architecture is flexible.

The vector architecture illustrated in this paper is designed to offer a scalable, power efficient, high performance and software-flexible architecture for the implementation of long precision modular multiplications.

3.1 Choice of a data parallel approach

We are looking at high performance parallel architectures for cryptography. To reduce circuit complexity, we avoid multiple instruction schemes and instruction parallel ones. In the latter architectures, the instruction decoders and issuers consume a lot of power as shown in [11] where for a superscalar microprocessor, one quarter of the total power is consumed by the instruction issue and queue logic while another quarter is taken by the instruction reorder buffers.

A *Data Level Parallel* approach is a better fit for cryptographic applications because the data in such applications can be decomposed into a vector of shorter data onto which operations can be applied in parallel, the instruction decoding is simpler (as illustrated in [17]) and in terms of security, working on data in parallel is expected to reduce the relative contribution of each data piece in the side channel leakages [4, 5].

3.2 The proposed vector architecture

The theory of vector processing and its application to micro-processors is given in Appendix A of [14]. Vector Processor techniques have been widely used from supercomputers like the Cray machine [23] to Digital Signal Processing applications like in Intel's MMX or in embedded media architectures like VIRAM [17], but never for cryptography. In our architecture we use a scalar MIPS to provide good scalar performance. To keep instruction decode simple, we delegate both vector and scalar instruction fetch and decode to the MIPS core. To suit the MIPS 'load-store' architecture and to avoid complex memory accesses, we chose a *Register-to-Register* vector architecture. With this

approach we reduce memory-register transfers, which are also the privileged attack paths for side channel analysis.

Details about the vector architecture implemented for the analysis done in this paper are given in [12]. In order to understand the analysis performed in Section 4, we need to highlight some of the vector processor's architectural details.

The architecture of the vector register file is illustrated in Figure 1. Six architectural parameters influence the structure of our vector register file:

- m: The size of each element of the vector registers ($m = 32$).

- q: The number of vector registers.

- p: The number of elements, called *depth*, in each vector register.

- r: The number of lanes which correspond to the number of Vector Processing Units (VPUs). This notion is borrowed from [1]. Ideally we would have $r = p$, allowing us to work on all p elements in parallel. However, in some cases, for size and power constraints we won't have p VPUs. We leave r as a parameter to allow us to analyse the best performance to size trade-off.
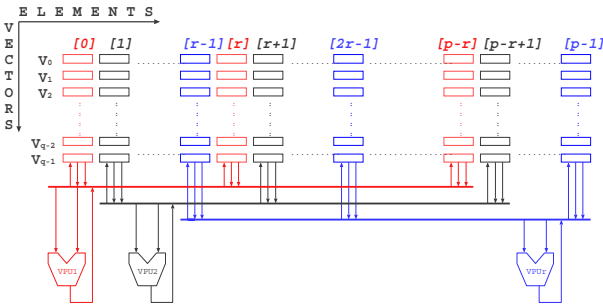

Figure 1. Distribution of the Vector Register File across Vector Processing Units (VPUs)

A vector instruction is meant to replace a "software loop" where the data being operated on are independent from each other and where the calculation of each iteration of the loop is independent from the calculation of the 'adjacent' iterations. By looking at some of the instructions in Appendix A[1], we can see that operations like VADDU do not obey this rule. For such instructions, we take advantage of the fact that the calculation on each element of the vector is only 'partially' independent from that of its neighbors. We hence define the GIVI (Genuinely Independent Vector

---

[1]Appendix A only describes the vector instructions needed for the modular multiplication.

Instruction), PIVI (Partially Independent Vector Instruction) and the MAVI (Memory Accessing Vector Instruction) as described in [12]. For example, for PIVI instructions like VADDU, each VPU has a 32-bit Carry Select Adder (CSA) such that each execution of the instruction is decomposed into two pipelined stages as illustrated in Figure 2:

- a first stage (the **EXM** stage) where the CSA performs two addition operations in parallel: one for an incoming 0 carry and one for an incoming 1 carry.

- a second stage (the **EXC** stage) where the correct result is chosen as a function of the correct incoming carry.
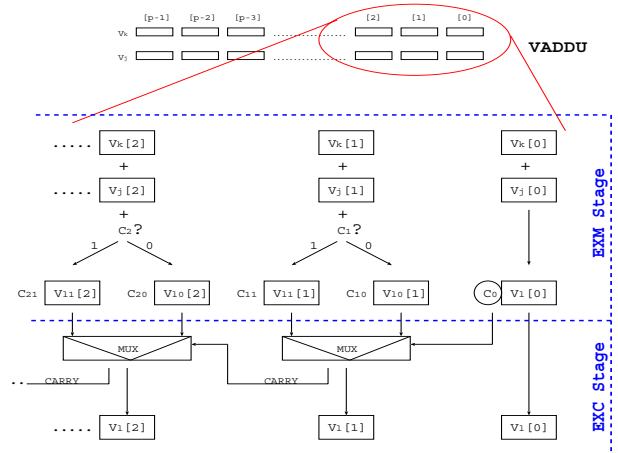

Figure 2. Execution of VADDU instruction

For our analysis, a functional model of the vector architecture was built using the ArchC tool [24]. An architectural instruction simulator was built based on a language description implementing the target architecture and a simulator generator built out of SystemC. The simulator generates a series of basic statistics among which the number of "cycle-counts" giving our "instruction cycles". The vector code was compiled using modified GCC tools.

3.3 Public key cryptography on the vector architecture

In [12], we have been looking at vector implementations of AES and modular multiplications in fields of characteristic 2 for Elliptic Curves Cryptography (ECC). In this section we concentrate on the modular multiplication for RSA [22] because the latter is used for our quantitative analysis in Section 4.

Modular Multiplication is implemented based on Montogomery's method [18]. Efficient implementations of the

3

latter algorithm are given in [6, 25]. We first implemented the CIOS (*Coarsely Integrated Operand Scanning*) method as described in [6]. To calculate $R = A \times B \bmod N$ we use Montgomery's method which yields $R' = A \times B \times r^{-1} \bmod N$ where N is long precision modulus of length l bits and r can be chosen such that $r = 2^l$. Suppose that each l-bit data Y can be decomposed into a linear combination of 32-bit integers denoted by Y_i such that

$$Y = Y_{M-1}.2^{32(M-1)} + \ldots + Y_1.2^{32} + Y_0 \qquad (1)$$

with $M = \lceil \frac{l}{32} \rceil$. We then have the algorithm in Figure 3 for the RSA's modular multiplication where J_0 is precalculated as the multiplicative inverse of N_0 modulo 2^{32}.

Input : A, B, N, M **and** J_0
Output : $R' = A.B.2^{-32M} \bmod N$

1. $R' \leftarrow 0$
2. **for** $j = 0$ **to** $M - 1$ **do**
3. $\quad R' \leftarrow R' + A_j \cdot B$
4. $\quad J \leftarrow R'_0 \cdot J_0 \bmod 2^{32}$
5. $\quad R' \leftarrow R' + J \cdot N$
6. $\quad R' \leftarrow R'/2^{32}$
7. **endfor**
8. **return** R'

Figure 3. Modular multiplication for RSA on a 32-bit machine

The algorithm is implemented in assembly language using the vector instructions given in Appendix A. On the functional simulator, the code takes 4095 instruction cycles. We also investigated the FIOS (*Finely Integrated Operand Scanning*) approach [6]. This improvement reduces the instruction cycle count to 3296. This 19.5% gain in performance is achieved at the expense of one additional vector register. These results seem to be in contradiction to those shown in [6] where the CIOS method outperforms the FIOS one by around 7.6%. This is because in the latter paper, even if there are less loops in the FIOS method, the potential gain is counterbalanced by the higher number of memory reads and writes. In our vector architecture, this increase in memory accesses has less impact because of the architecture of our vector register file.

4 Quantitative study of the modular multiplication

In this section, we show how a quantitative analysis is carried on a functional simulator of our vector architecture.

We look at the performance of the modular multiplication between two long precision numbers for RSA. We change the depth p of each vector register and the number of lanes r to see how performance is affected by those design parameters. Future cycle-accurate models (in synthesizable Verilog) will incorporate other parameters such as gate count and power estimations. The measurements are done for some characteristic values of data length l, register depth p and number of lanes r.

4.1 Varying data size

We first varied the size of the data used. We performed measurements for data sizes of 512, 1024, 2048 and 4096 bits. In theory, based on the algorithm in Figure 3, every time we double the size of the key, we expect the number of clock cycles to be multiplied by 4 (or even more according to [6]).
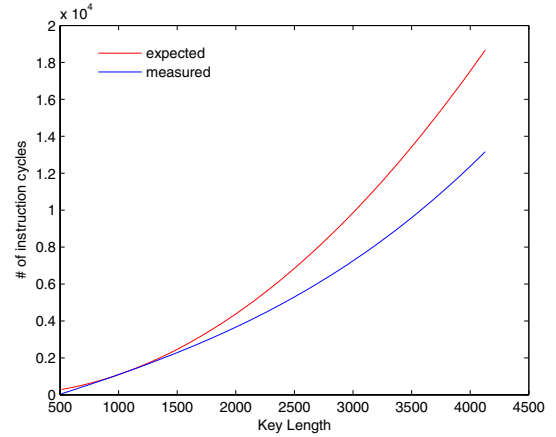

Figure 4. Number of cycles *versus* data size (Vector Depth= 32)

In Figure 4, the "upper" curve represents what is expected and the "lower" one represents what is actually measured. We see that the performance penalty decreases as the size of the data increases. Note that this observation is in line that made in [7] where the authors show that the timing on the vector processor is "*more linear with size*" than on scalar architectures.

4.2 Changing depth of vector registers

We repeated the above experiment but this time for different values of the depth p.

4.2.1 The vectorization effect on varying data sizes

From the graphs in Figures 5 to 8 we see that as p decreases, the measured variation in performance gets closer and closer to the expected theoretical behaviour. When $p = 1$, the practical results match the theoretical ones. The case of $p = 1$ is the limit where our vector architecture becomes a scalar one. This supports the fact that with the vector architecture, we can work on larger data sizes with a performance penalty which is less than the expected one. This can be viewed as a relative gain in efficiency. This behavior could be due to the register-to-register vector architecture where large data words are loaded at once, thus reducing the accesses to external memory.
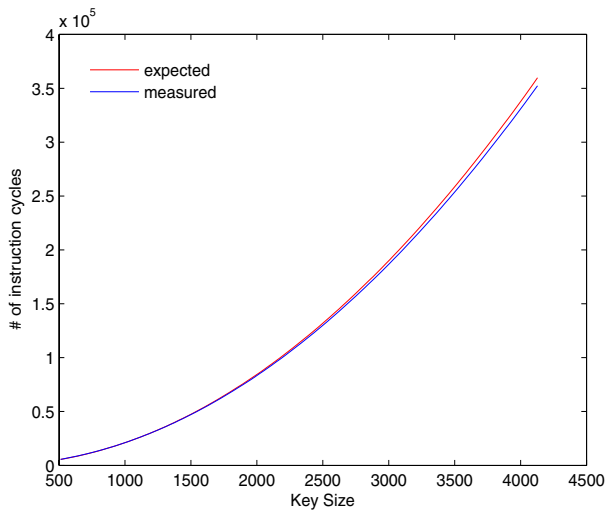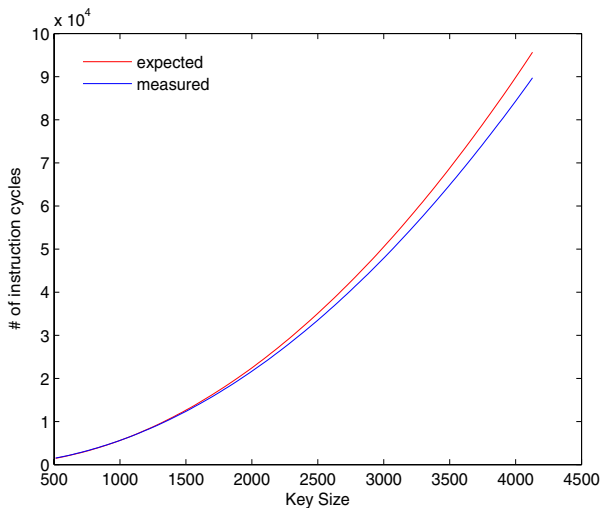

Figure 5. Number of cycles *v/s* key size: $p = 1$
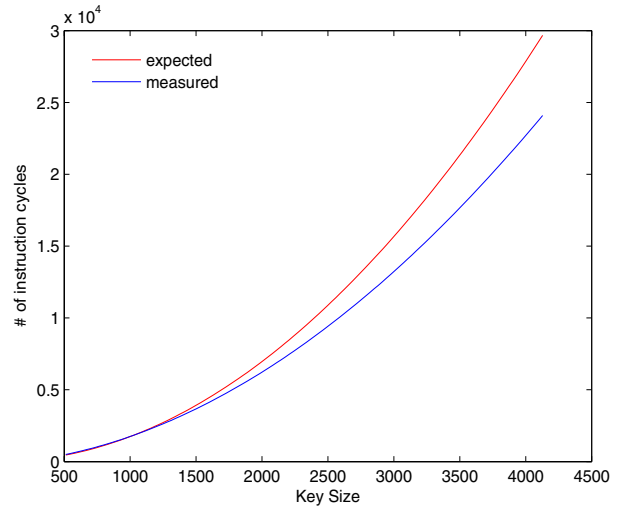

Figure 6. Number of cycles *v/s* key size: $p = 4$

Figure 7. Number of cycles *v/s* key size: $p = 16$
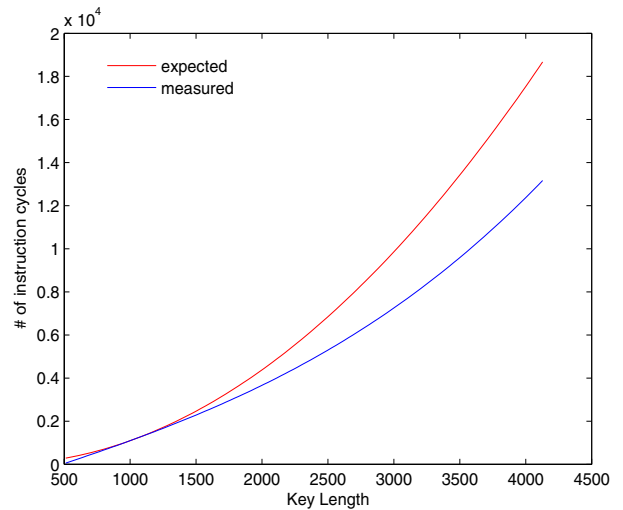

Figure 8. Number of cycles *v/s* key size: $p = 32$

4.2.2 Profiling the rate of performance change with increasing depth

The other interesting aspect of the collected figures is to see how, for a given data size, the performance of the modular multiplication routine is affected by changing p. Figure 9 shows such profiles for two key lengths (we use a semi-logarithmic scale). From those figures, we can infer that the number of cycles decreases exponentially with the depth p. We also found that beyond what seems to be the critical value $p = 16$, the number of cycles decreases asymptotically. Typically, for a hardware designer, this would mean that beyond 16, increasing p will result in negligible performance gain.
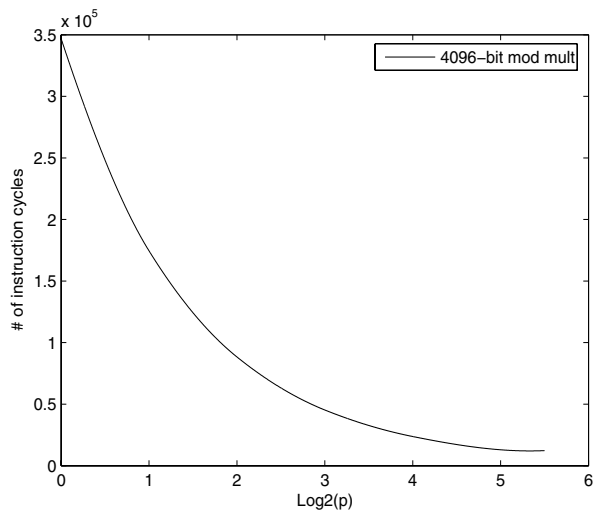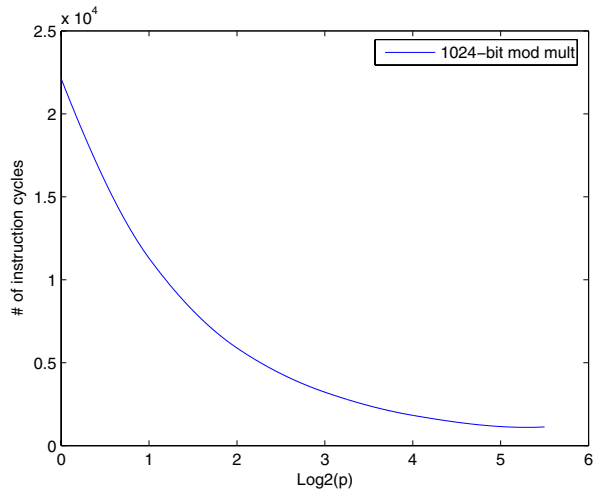
Figure 10. Number of cycles *v/s* **number of lanes for different vector depths**

4.4   Summary of results

The quantitative analysis made on the RSA's modular multiplication allows us to draw the following conclusions:

1. As data size gets bigger, the rate of increase in instruction count gets smaller than the theoretical values. Actually the rate of increase of instruction cycles decreases as the data sizes get bigger.

2. The above difference between the theoretical and experimental behaviors gets more important as p increases.

3. For a given length of data, increasing p decreases the number of instruction cycles logarithmically. From $p = 1$ to $p = 16$ the rate of loss in performance decreases more or less linearly but beyond $p = 16$ there seems to be a more important loss.

4. Increasing the number of lanes decreases the number of instruction cycles logarithmically. For a given p, the more lanes we have, the smaller is the relative gain in performance.

In terms of performance, a fair comparison (for a given data size, say 1024 bits) between our work and the previous publications enumerated in Section 2 wouldn't be justified. All of those hardware architectures could be considered to be multiple instructions issue architectures while ours is a single instruction issue one. Moreover, our architecture is designed to be scalable and favor design-to-cost approaches where one design can be resized to suit performance, size

Figure 9. Number of cycles *versus* **Vector Depth (Log2 scale)**

4.3   Varying the number of lanes

We also looked at the effect of varying the number of vector processor lanes. The curves in Figure 10 show how the number of cycles for a 1024-bit modular multiplication varies when increasing the number of lanes. For each value of p, we increase r from 1 to p, doubling the value of r every time. We then 'interpolated' in between the measured points to obtain the 'trend'. We see that as r gets larger, increasing r provides a gain in performance which tends to decrease: the reduced gain in performance is more or less linear. In [17], the authors already observed that for media applications, the *"efficiency"* of the vector architecture decreases as we increase the number of lanes.
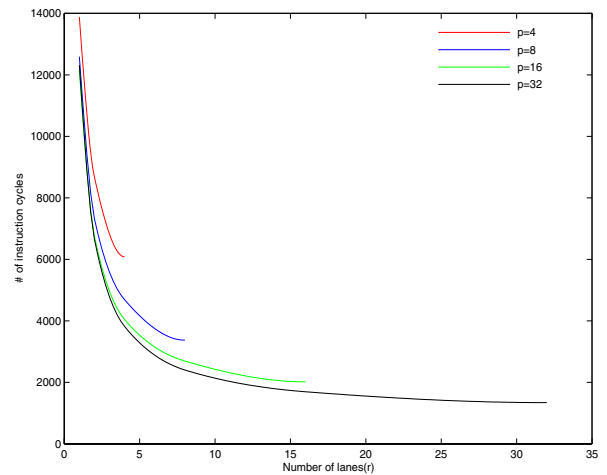
6

and power consumption constraints. Even so, if we wanted to compare figures, we would note that our implementation for 1024-bit data ideally takes 3296 cycles, that on the Power PC's Velocity Engine [7] takes 3600 cycles and the one on a DSP [15] takes 14000 cycles. These figures indicate that a relatively simple to implement vector processor like ours is able to achieve good performance without the complexities of multiple instruction issue.

5 Conclusions and future work

In this paper, we showed that high performance can be achieved through parallel computation of cryptography on a vector architecture. It is well known that instruction level parallelism is very expensive in terms of hardware and in particular very complex in terms of instruction decoding and scheduling. On the other hand, taking a vector approach is a relatively cheap way of achieving high performance parallelism as most of the logic goes into the data path and not in the control path. The latter point also renders our vector architecture very modular. We showed how this modularity can be exploited to provide a range of speed and area choices for RSA cryptography.

Our vector approach provides a new vision of parallel executions of cryptography compared to previous work, specially for embedded applications where designers are permanently seeking for performance, size and power trade-offs. Our modular design approach gives us a quantitative tool to build a cryptographic PK coprocessor based on such a *design-to-cost* philosophy. There are two other parameters missing: gate count and power consumption. We are currently constructing a synthesisable Verilog RTL model which will allow us to obtain cycle accurate performance figures, gate count and power consumption estimations.

References

[1] K. Asanovic̀. *Vector Microprocessors*. PhD thesis, University of California, Berkeley, Spring 1998.

[2] R. Bhaskar, P. K. Dubey, V. Kumar, and A. Rudra. Efficient Galois Field Arithmetic on SIMD Architectures. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 256–257, New York, NY, USA, 2003. ACM Press.

[3] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*, volume 265 of *Lecture Note Series*. London Mathematical Society.

[4] E. Brier, C. Clavier, and F. Olivier. Optimal Statistical Power Analysis. *Cryptology ePrint Archive,* http://eprint.iacr.org/, Report 2003(152), 2003.

[5] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.

[6] Çetin Koç, T. Acar, and B. S. Kaliski. Analysing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[7] R. Crandall and J. Klivington. Vector implementation of multiprecision arithmetic. Technical report, Apple Computers, Inc., October 1999.

[8] K. Diefendorff, P. Dubey, R. Hochsprung, and H. Scales. AltiVec Extension to PowerPC Accelerates Media Processing. *IEEE Micro*, 0272(1732):85–95, March-April 2000.

[9] H. Eberle, S. Shantz, V. Gupta, N. Gura, L. Rarick, and L. Sparcklen. Accelerating NEXT-Generation Public-Key Cryptosystems on General-Purpose CPUs. *IEEE Micro*, 0272-1732(05):52–59, March-April 2005.

[10] W. Fischer, C. Giraud, E. W. Knudsen, and J.-P. Seifert. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks. Cryptology ePrint Archive, Report 2002/007, 2002. http://eprint.iacr.org/.

[11] D. Folegnani and A. González. Energy Effective Issue Logic. *Proceedings of 28th Annual International Symposium on Computer Architecture 2001 (ISCA'2001)*, pages 230–239, June-July 2001.

[12] J. J. Fournier and S. Moore. A Vector approach to Cryptography Implementation. In R. Safavi-Naini and M. Yung, editors, *Proceedings of 1st International Conference on Digital Rights Management - Technologies, Issues, Challenges and Systems (DRMTICS'2005)*, volume LNCS, pages 277–297. Springer-Verlag Berlin Heidelberg 2006, 2006.

[13] J. Groszschaedl and G. Kamendje. Instruction Set Extension for Fast Elliptic Curve Cryptography over Binary Finite Fields GF(2^m). *Proceedings of IEEE International Conference on Application Specific Systems Architectures and Processors (ASAP'2003)*, pages 455–468, June 2003.

[14] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3 edition, 2003.

[15] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara. Fast Implementation of Public-Key Cryptography ona DSP TMS320C6201. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 61–72, London, UK, 1999. Springer-Verlag.

[16] T. Izu and T. Takagi. Fast Elliptic Curve Multiplications with SIMD Operations. *Fourth International Conference on Information and Communications Security (ICICS'02)*, LNCS(2513):217–230, 2002.

[17] C. E. Kozyrakis and D. A. Patterson. Scalable Vector Processors for Embedded Systems. *IEEE Micro*, 23(6):36–45, Nov/Dec 2003.

[18] P. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[19] NIST. Specification for the Advanced Encryption Standard. Technical Report 197, Federal Information Processing Standards, November 26 2001.

[20] D. Page and N. P. Smart. Parallel Cryptographic Arithmetic using a Redundant Montgomery Representation. *IEEE Traansaction on Computers*, 53(11):1474–1482, November 2004.

[21] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4):42–50, August 1996.

[22] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[23] R. M. Russell. The CRAY-1 Computer System. *Communications of the ACM*, 21(1):63–72, January 1978.

[24] T. A. Team. The Archc Architecture Description Language - Reference Manual. Technical report.

[25] A. F. Tenca and Çetin K. Koç. A Scalable Architecture for Montgomery Multiplication. *Proceedings of CHES'99*, LNCS(1717):94–108, 1999.

[26] Texas-Instruments. TMS320C6201 Fixed Point Digital Signal Processors, January 97, Revised March 2004.

A Vector Instruction Set

The VeMICry is composed of q vector registers each of p words of 32 bits . The p–bits wide Vector Condition Register (VCR) is used for conditional vector instructions. The Carry Register (CAR) stores the 'carry words'. V_i designates the ith vector register, R_j is the jth scalar register and n is a 16-bit immediate value.

VADDU V_l, V_j, V_k
Does the unsigned addition between the ith elements of V_j and V_k, writing the result as the ith element of V_l. The carry is propagated and added to the $i + 1^{st}$ element of V_l. The carry from the addition of the corresponding p^{th} words is added to CAR.

VLOAD V_l, R_i, n
Loads in V_l the n consecutive 32-bit words from memory starting from address stored in R_i in steps of 1.

VEXTRACT R_i, V_j, n
Copies the value of the $V_j[n-1]$ into R_i. If $n = 0$, CAR is written to R_i.

VSAMULT V_l, V_j, R_k
Vector-Scalar-Arithmetic-Multiplication: multiplies R_k by $V_j[p]||\ldots||V_j[1]||V_j[0]$ with carry propagation, writing the result into V_l. The most significant carry bits are written to CAR.

VSMOVE V_l, R_k, n
Copies the value in register R_k to the first n words of V_l. If n is *zero*, then R_k is copied to every word of V_l.

VSTORE V_l, R_k, n
Stores the first n consecutive 32-bit words from register V_l to memory starting from address stored in R_k in steps of 1.

VSPMULT V_l, V_j, R_k
Vector-Scalar-Polynomial-Multiplication: does the polynomial multiplication of R_k by $V_j[p]||V_j[p-1]||\ldots||V_j[0]$ and writes the result to V_l. The resulting $p + 1^{st}$ word is written to CAR.

VXOR V_l, V_j, V_k
XORs corresponding words between V_j and V_k and stores the result in V_l

VWSHL V_l, V_j, n
Vector-Word-Shift-Left shifts the contents of vector V_j by n positions to the left inserting *zeros* to the right. The resulting vector is written to V_l and the outgoing word to CAR

VWSHR V_l, V_j, n
Vector-Word-Shift-Right shifts the contents of vector V_j by n word position to the right inserting the data stored in CAR to the left. The resulting vector is written to V_l.

MTVCR R_j
Writes to VCR the value contained in the scalar register R_j.

MFVCR R_j
Copies the value contained in VCR to the scalar register R_j.