Programming Highly Parallel Reconfigurable Architectures for Symmetric and Asymmetric Cryptographic Applications

Giovanni Agosta, Luca Breveglieri, Gerardo Pelosi, Martino Sykora
Politecnico di Milano
Dipartimento di Elettronica e Informazione (DEI)
Via Ponzio, 34/5, 20133 Milano, Italy
Email: {agosta,brevegli,pelosi,sykora}@elet.polimi.it

Abstract— **Tiled architectures are emerging as an architectural platform that allows high levels of instruction level parallelism. Traditional compiler parallelization techniques are usually employed to generate programs for these architectures. However, for specific application domains, the compiler is not able to effectively exploit the domain knowledge. In this paper, we propose a new programming model that, by means of the definition of software function units, allows domain-specific features to be explicitly modeled, achieving good performances while reducing development times with respect to low-level programming. Identity-based cryptographic algorithms are known to be computationally intensive and difficult to parallelize automatically. Recent advances have led to the adoption of embedded cryptographic coprocessors to speed up both traditional and identity-based public key algorithms. We show the effectiveness of the proposed programming model by applying it to the case of computationally intensive cryptographic algorithms in both identity-based and traditional algorithms. Custom-designed coprocessors have high development costs and times with respect to general purpose or DSP coprocessors. Therefore, the proposed methodology can be effectively employed to reduce time to market while preserving performances. It also represents a starting point for the definition of cryptography-oriented programming languages. We prove that tiled architecture well compare w.r.t. competitors implementations such as StrongARM and FPGAs.**

Keywords: identity-based cryptography, tiled architectures, parallel programming model, reconfigurable architectures, multiobjective exploration.

I. Introduction

Since traditional microprocessors are becoming increasingly complex, leading to high design and manufacturing costs, new trends in architectures are moving towards partitioned register file architectures, such as *tiled architectures*, which allow high levels of instruction level parallelism combined with good scaling properties. These architectures are currently considered for both general purpose and DSP applications.

The public key cryptographic algorithms are computationally intensive, so that the current research trend is oriented towards the adoption of application specific coprocessors, often based on reconfigurable hardware, to reduce time to market.

In several cases, multimedia data may be encrypted for content distribution. To this end, the most commonly applied solution, for efficiency reasons, is to combine a symmetric encryption schema to enforce the desired authentication requirements.

DSP-oriented tiled architectures could be used to obtain further improvements in time to market, cost and performance, provided that the parallel pipelines can be exploited intensively to limit the hardware area. To this end, new programming models are required, because standard compiler techniques are not able to extract parallelism from these algorithms at both task and instruction level.

In this paper, we propose a new programming model that, by means of the definition of software function units, allows domain-specific features to be explicitly modeled, achieving good performances while reducing development time with respect to low-level programming. We show the effectiveness of the proposed programming model by applying it to the case of a computationally intensive cryptographic primitive such as pairings, which is heavily based on the efficient computation of the underlying multi-precision arithmetic as in most public key algorithms. Asymmetric cryptographic systems are especially interesting in embedded systems when the application requires fingerprinting capabilities, either to guarantee that the originator of the data is known, or to guarantee that, if the receiver redistributes the data, these can be traced back to them.

The basic idea is that of including within the transmitted data a watermark, i.e. a modification of the original data (a *watermark* or fingerprint) that is small enough to make the data still useful for their original purpose (in media, this generally means that the pictures, movies and sounds should preserve their quality from the point of view of the user), and allows the identification of either the sender, the receiver, or both. Moreover, properties such as the impossibility to alter the fingerprint without making

This paper is based on "Programming Highly Parallel Reconfigurable Architectures for Public-key Cryptographic Applications," appeared in the Proceedings of the 4th International Conference on Information Technology (ITNG 2007), Las Vegas, USA, April 2007. © 2007 IEEE.

the data useless may be required, depending on the actual application scenario.

Two important applicative scenarios are readily available. The first concerns a typical e-commerce scheme where a merchant and a buyer exchange data. The buyer receives some form of media, possibly on a handheld device, a set-top box or other embedded device, that he should not distributes to third parties. This can be ensured by adding a fingerprint based on the buyer's secret key, which is therefore verifiable using their public key by the merchant and by authorities. In this case, the fingerprint should not be removable without making the data useless, not even by comparing different fingerprints. Additional problems include the anonymity of the buyer – i.e., the merchant should not be able to trace the buyer unless the redistribute the data. Several asymmetric fingerprinting schemes have been developed for this purpose [1]–[3].

A second scenario concerns the case where attribution of a video, audio or picture is needed for legal purpose – such forms of digital evidence need authentication to be used in court, and since most digital data can be easily tampered, such authentication can be provided by fingerprinting techniques. Asymmetric fingerprinting in this case can provide means to attribute data to a specific recording device (assuming of course one can prove that the device itself has not been tampered with). In this scenario, the secret key is stored in the tamper-proof recording device, which performs the encoding process. This way, every datum that is transmitted from the device can be traced back to it.

The rest of this paper is organized as follows. Section II gives the essential mathematical background on cryptographic parings. Section III introduces tiled architectures and their interconnection structure. Section IV outlines the proposed programming model, as well as two case studies showing the implementation of software function units in identity-based and symmetric cryptography. Section VI provides an experimental evaluation of the proposed programming model. Finally, Section VII draws the conclusions and suggests future research directions.

II. Cryptographic Pairing background

A cryptographic pairing is a bilinear map between two groups $\mathbf{G}_1$, $\mathbf{G}_2$ where the discrete logarithm problem is hard.

$$\mathfrak{t}<,>: \mathbf{G}_1 \times \mathbf{G}_1 \rightarrow \mathbf{G}_2$$

Let $P, Q, R \in \mathbf{G}_1$ then

$$\mathfrak{t}(P+R,Q) = \mathfrak{t}(P,Q)\mathfrak{t}(R,Q)$$
$$\mathfrak{t}(P,Q+R) = \mathfrak{t}(P,Q)\mathfrak{t}(P,R)$$

During the last few years, pairings have been successfully employed in order to work out several open problems in cryptography such as, one-round three-way key exchange [4], identity-based encryption [5], and short digital signatures [6]. For further deepenings on the protocols that make use of pairing primitives we send back to [7], [8] and their references. The Weil and Tate pairings on elliptic curves over finite fields represent the mathematical basics to construct identity-based cryptographic primitives. These pairings are bilinear maps from an elliptic curve group $E(\mathbb{F}_q)$ to the multiplicative group of some extension field $\mathbb{F}_{q^k}$. The parameter k is called the embedding degree of the elliptic curve [9], [10].

The pairing is considered to be secure if the discrete logarithms in the groups $E(\mathbb{F}_q)$ and $E(\mathbb{F}_{q^k})$ are both computationally infeasible. For optimal performance, the parameters q and k should be chosen so that the two discrete logarithm problems are of approximately equal difficulty when using the best known algorithms, with the order of $\#E(\mathbb{F}_q)$ having a large prime factor r. The best attack known on the elliptic curve discrete logarithm problem is the *parallel collision search* that improves on the Pollard's ρ-algorithm [11]. A pairing is considered as secure as 1024-RSA, when $r \sim 2^{160}$, k ranges from 2 to 10, depending on the application, and $p^k \geq 2^{1024}$. In the wake of recent works [12]–[14] on pairings over general curves over pairing friendly fields of large prime characteristic, the proposed programming model will be aimed to the implementation of the Tate pairing primitive in characteristic p with $k = 2$ and $p \sim 2^{512}$. The current algorithm to compute the pairing is a careful refinement of the well known BKLS/GHS algorithms as described in [9], [10], [15].

The cryptographic usage of the Tate pairing involves the application of Miller's Algorithm [16] followed by a final exponentiation. The point P is chosen as an element of $E(\mathbb{F}_p)$ with order r. The point Q is chosen as an element of $E(\mathbb{F}_{p^k})$ which is mapped from the twisted curve. Miller's algorithm uses the double and add schema for elliptic curve point multiplication rP, with some more operations to evaluate intermediate values of the pairing that are multiplicatively accumulated to compute the output of the algorithm [17]. Miller's algorithm performs $\lceil \log_2 r \rceil - 1$ iterations executing almost always the block of operations corresponding to a *point doubling*. Indeed, if a low hamming weight r is used then only few *point additions* will be required (e.g. 1-10). The core idea behind this work is to investigate ways to combine instruction-level parallelism that can be found in the implementation of multiprecision arithmetic operations with task-level parallelism among the finite field operations involved in the computation of pairings.

III. Tiled Architectures

Recent trends in microprocessor design are moving towards partitioning processor resources such as register files, cache banks and pipelines. In *Very Long Instruction Word* (VLIW) architectures, a single program counter controls several pipelines that access the same register file. However, this structure does not scale well, since large register files are impractical. *Tiled architectures*, such as Raw [18], Wavescalar [19] and TRIPS [20], represent an evolution of VLIWs, partitioning the register file so that each pipeline or cluster of pipelines (called a tile, a *computational node* or simply a node) can access a private register bank. While this allows smooth scaling, it poses communication problems, as data need be moved among the different pipelines, moreover, since the register file is partitioned, communication must take place on an interconnect network, called a *scalar operand network* [21]. These issues must be dealt with by the compiler, which is in charge of scheduling instructions not only in time, but also in space – that is across different nodes. Explicit communication instructions must be issued to synchronize the register file partitions.

In this work, we focus on DSP-oriented tiled coprocessors with a single control flow, since they are the direct competitor of the FPGA and ASIC solutions for public key cryptographic algorithms. More complex nodes, such as those of Raw (a MIPS pipeline with private data and instruction caches) would be orders of magnitude larger and more costly than the industry standard solutions.

A tiled architecture is an array of nodes, where each node is a computing element accessing its own register file and exposing a set of private function units. When all the nodes have the same type of function units, the architecture is homogeneous, and heterogeneous otherwise. The migration of the operands among clusters is demanded to a word-level communication network and is controlled by special instructions – like `snd` or `rcv` – executed by the nodes themselves, or by dedicated hardware. This kind of architecture belongs to the family of Scalar Operand Networks (SON), and can be characterized by the AsTrO taxonomy [21], which specifies whether the assignment of the instructions, the transport of the operands and the ordering of the instructions are statically or dynamically performed.

DSPFabric [22], by STmicroelectronics, is a tiled architecture specifically designed for modulo scheduling computationally intensive loops of multimedia applications. With respect to the AsTrO taxonomy, it is a Static-Static-Static SONs, which means that the assignment of the instructions, the displacement of the copies and the scheduling passes are compiler tasks. Moreover, DSPFabric is characterized by coarse-grained reconfigurable data-paths. The compiler must select a subset of feasible node connections for data flowing, and emits at compile time the reconfiguration instructions that activate the selected wires. These reconfiguration instructions change at runtime the network topology, tailoring it to the specific code.

The reconfiguration space – the space of feasible topologies – is tailored by the constraints given by the availability of I/O ports with respect to the total number of connecting wires. In the DSPFabric organization, each node can be potentially connected to all the others, exploiting a hierarchical interconnection schema, based on different levels of MUXes. Effective limitations are given by the MUXes capacity. We describe in the following the DSPFabric architecture, focusing the attention on the structure of the interconnections.

A. DSPFabric Architecture

Figure 1 gives an overall picture of a 64 nodes DSPFabric coprocessor. At level 0 it can be seen as an array of four 16-issue processors (clusters), communicating through a collection of multiplexers, which implements a multi input/output switch. Each cluster has N input wires and N output wires, where the output wires are possibly connected to all the others. On the contrary, the input wires can be connected to only one source. Figure 2 shows a feasible data path at level 0, assuming N equal to 4. At level 1, the spatial structure replicates

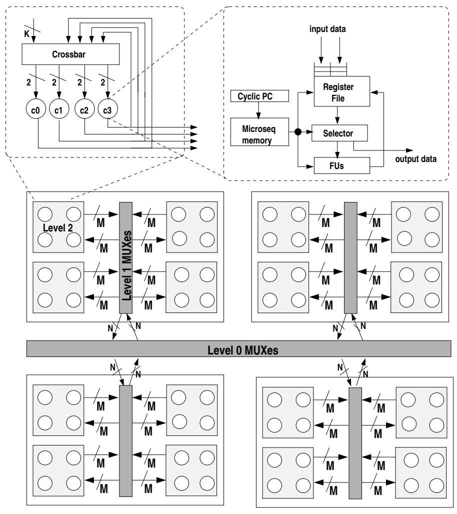


Figure 1. A DSPFabric architecture with 64 nodes.

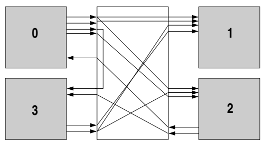


Figure 2. A feasible interconnection among clusters sets. Assuming the output and input capacities of 4 wires, cluster set 0 and 1 have saturated their available output and input wires, respectively.

itself inside each cluster, again with an array of 4-issue processing elements, connected together by multiplexers with capacity M. The last level is composed by the *computation nodes* connected through a reconfigurable crossbar, which takes as input the internal connections and K of the wires outgoing from level 1. Each computation node has two ingoing wires and one outgoing wire.

The computation nodes are single issue pipelined processors, accessing their own register file and functional units. Since DSPFabric has been specifically designed as a loop accelerator coprocessor for multimedia applications, each node is equipped with hardware features for better executing modulo scheduled code [23]; e.g., the node contains support for instruction predication and rotating registers. Precisely, the application is scheduled using the Kernel Only Modulo Scheduling [23] technique, which fully predicates loop prologue and epilogue. Thus, branches are not allowed and the execution is controlled by a cyclic program counter.

The copies between different register files are controlled by the `receive` primitive executed by the destination node. Two regions of the register file are organized as input buffers, which push on top the incoming values, but can be read randomly by the receiver.

The coupling with the main memory subsystem is demanded to a programmable DMA. Each node can generate an address request, which is directly sent to DMA without consuming inter-cluster communication patterns. Only a limited number of requests can be served at the same time, i.e. 8 requests, thus the compiler must ensure that the amount of simultaneous requests does not exceed that limit. Since the memory requests do not have unary latency, the DMA engine provides input and output FIFOs – of depth equal to the serving time – for handling high memory pressure. When a value is ready it is directly loaded into the requesting cluster register file.

Since DSPFabric – and, more generally, all the coprocessors designed for multimedia embedded applications –is specifically designed for performing media streaming, and the input/output streams are characterized by a highly regular structure and largerly independent data, the DMA programmable interface allows to perform efficient data buffering and to mask the memory latencies.

IV. Programming Model & Compiler Techniques

In this Section, we discuss the limitations of the compiler techniques for scheduling the target algorithms on tiled architectures, and propose a new programming model to deal with these issues. We apply the proposed programming model to the case of the Tate pairing computation.

TABLE II.
IMPLEMENTATION OF THE 32×32 MULTIPLIER AS A SW FUNCTION UNIT USING 16 BIT MULTIPLIERS PROVIDED BY THE TARGET ISA.

and a0,X0,0x0000FFFF	and b0,Y0,0x0000FFFF
shftr a1,X0,16.rcv b0	shftr b1,Y0,16. rcv a0
mul c00,a0,b0	mul c01,a0,b1. rcv a1
mul c10,a1,b0	mul c11,a1,b1
and x,c10,0xFFFF0000	shftl w,c01,16
shftl y,c10,16. rcv w	and z,c01,0xFFFF0000.rcv x
add ml0,c00,y	add mh0, c11, z
cmpgt r0,y,ml0	add mh0, mh0, x
add ml0,ml0,w	
cmpgt r00,y,ml0. rcv mh0	
add r0,r0,r00	
add mh0, mh0, r0	

A. *Compiler Techniques for Tiled Architectures*

Tiled architectures are specifically designed for the execution of computationally intensive kernels of multimedia applications, e.g. Inverse Cosine Discrete Transform, interpolation and de-blocking filters. A typical scenario is to employ such machines as innermost loop accelerators – implemented as coprocessors and coupled with the central processing system. These loops are characterized by largely independent operations and low memory aliasing, exposing a high degree of potential Instruction Level Parallelism (ILP). Moreover these kernels are usually quite small – in the range from 100 to 1000 instructions in the loop body.

Since these architectures are conceived for loop acceleration, they typically provide hardware features to enhance Modulo Scheduling [23] compiler technique, like support for predicated execution and rotating registers [24].

The compiler front-end recognizes in-code pragmed loops as suitable to be mapped onto the coprocessor, then translates them into an intermediate representation. At the intermediate level a loop is described by its Data Dependency Graph (DDG), where each node represents a native (assembly) instruction and each edge is a data dependence between two adjacent instructions.

The compiler back end assigns the node of the DDG to the clusters and schedules them, compatibly with the data, resources and communication constraints. The challenge is to extract the maximum degree of parallelism and, at the same time, to limit the penalties due to explicit inter-cluster operand copies. Different approaches have been proposed for performing cluster assignment and scheduling, considering both 2-phases and unified techniques [25]–[29].

The compiler is typically driven by in-code pragmas, which select the loops to map onto the multi-clustered coprocessor. As intermediate representation the loop is described by its Data Dependency Graph (DDG), where each node represents a native instruction and each edge introduces a data dependence between instructions.

The behavior of the compiler back end is to assign the instructions to the clusters and to schedule them, compatibly with the communication net topology, the data dependencies and the resource constraints. The compiler tries to extract the maximum degree of parallelism and, at the same time, to limit the penalties due to explicit inter-cluster operand copies. Different approaches have been proposed for performing cluster assignment and scheduling, considering both 2-phases and unified techniques [25]–[29].

Since these architectures are conceived for loop acceleration, they typically provide hardware features to enhance Modulo Scheduling [23] compiler technique, like support for predicated execution and rotating registers [24].

Trying to map and schedule a complex cryptographic algorithm, i.e. Tate pairing, following this programming model – thought for different scenarios – arises several constraints and rapidly leads to low quality or indeed not schedulable code.

When observed at a high level, the Tate pairing algorithm is a single loop that presents high parallelism at the level of operations between very long integers, e.g. multiplications of 32 words-length operands. These operations, if written in a high level source code as `C`, will appear as cyclic algorithms, exposing an internal degree of parallelism. The only way to exploit the high level parallelism is to completely unroll all the internal loops

TABLE I.
IMPLEMENTATION OF THE MODULAR ADDER AS A SOFTWARE FUNCTION UNIT.

add s1,a1,b1	add s11,a1,b1				
add s1,s1,1	cmpgt c21,b1,s11	add s2,a2,b2	add s21,a2,b2		
cmpgt c2,a1,s1.rcv c1.rcv c21		add s2,s2,1	cmpgt c31,b2,s21		
slct c2,c1,c2,c21		cmp c3,a2,s2.rcv c31		add s3,a3,b3	add s31,a3,b3
		rcv c2		add s3,s3,1	cmpgt c41,b3,s31
add s1,c1,m1	add s11,c1,m1	slct c3,c2,c3,c31		cmp c4,a3,s3.rcv c41	
add s1,s1,1	cmpgt r21,m1,s11	add s2,c2,m2	add s21,c2,m2	rcv c3	
cmpgt r2,c1,s1.rcv r1.rcv r21		add s2,s2,1	cmpgt r31,m2,s21	slct c4,c3,c4,c41	
slct r2,r1,r2,r21		cmp r3,c2,s2.rcv r31		add s3,c3,m3	add s31,c3,m3
		rcv r2		add s3,s3,1	cmpgt r41,m3,s31
		slct r3,r2,r3,r31		cmp r4,c3,s3.rcv r41	
				rcv r3	
				slct r4,r3,r4,r41	

into the outer-most one, and then try to modulo schedule the whole resulting loop.

This approach is computationally hard, since the scheduling problem is NP-complete and the size of the input data in this case (the nodes of the DDG) grows quickly – more than 300000 nodes for a 512-bit Tate pairing implementation.

Therefore we propose a novel compilation approach, which allows to exploit the available parallelism, decoupling the problem into two phases. The former determines the function units needed to support the high level parallelism, the latter programs each function unit scheduling the code at fine-grain of parallelism.

B. Proposed Programming Model

Cryptographic algorithms that use multi-precision integer arithmetic are representative of a class of applications that present peculiar properties in terms of available parallelism and program structure. Specifically, computationally intensive public key cryptographic algorithms such as the Tate pairing implementation in [30]–[32] can be parallelized at *task level* (TLP), as proven by a wide range of literature on the design of hardware implementations that typically use replicated modular arithmetic circuits to exploit this type of parallelism [33], [34]. The design of the individual modular arithmetic circuits highlights the availability of a significant amount of *instruction-level parallelism* (ILP): the parallel operations in hardware can be transposed to parallel instructions in the software implementation. On the other hand, *loop-level parallelism* (LLP), that is the opportunity to perform different iterations of the same cycle on different computational elements, is less easily found in this type of application, due to the need to propagate loop carried data dependencies (such as the carry propagation for the integer or *mod p* arithmetic) across the iterations of a given loop. Since LLP is the type of parallelism most easily exploited by compilers, while TLP is especially difficult to extract by means of a compiler, these algorithms prove to be particularly difficult to parallelize automatically.

To tackle this issue, our method highlights TLP and ILP in the target algorithms, by mirroring typical hardware design concepts, such as specialized arithmetic hardware. Specifically, in the proposed model, the target algorithm is written using a library of software components that perform the same operations as specialized hardware function units for multi-precision integer arithmetic. The code *software function units* are optimized for the target architecture, customizing the connections between tiles of the architecture to fit their data propagation schemata. Since carry propagation flows one-way from the least significant word to the most significant one, it fits well for a very regular structure that can be easily mapped to the configurable connections between computational nodes, as each node need synchronize itself only with its neighbours.

Each software function units is, on a given target architecture, characterized by two parameters: the schedule length and the resource usage, in terms of number of computational nodes. This characterization mirrors closely the area and latency parameters of an hardware function unit. Therefore, a top-down approach can be used, applying well-known methodologies for the design of the controller datapath. In this way, the high-level representation of the algorithm is mapped to the software function units by means of a list-based scheduling algorithm [35].

V. CASE STUDIES

In this Section, we describe two case studies run using the programming model described in the previous Section. The first case study deals with the design of a low-level multi-precision arithmetic library in terms of software function units and its application in the implementation of a Tate pairing algorithm. The second case study presents a more complete design, using the IDEA cryptosystem as the target algorithm.

A. Modular Arithmetics

The goal of this Section is to describe the design of a basic multi-precision arithmetic library. The Montgomery multiplier is the main element of any such library. To this end, we need to first develop basic function units such as the modular adder and the word-by-vector multiplication, with the aim of composing them to implement the main loop of the Montgomery multiplier as described in Algorithm V.1.

Table I shows the basic schema for a modular adder. Each column of the table represents the schedule of a single computational node. For each word of the multi-precision operands to add, a pair of nodes is used to speculatively execute both the case with and without carry. The table considers the case of only three word multi-precision operands, but the extension to larger sizes is straightforward. Figure 3 shows how the adder can be

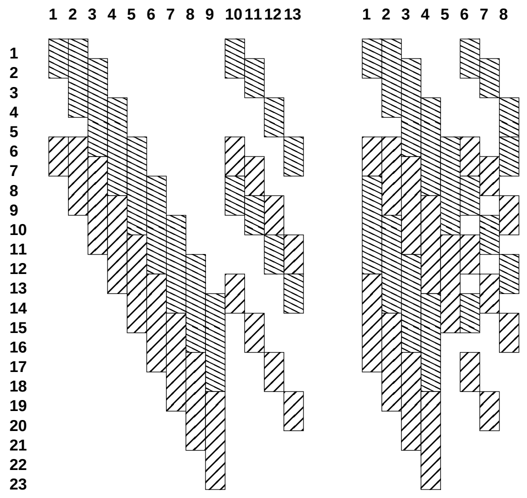


Figure 3. Time/space scheduling of a 128-bit modular addition: the dark shaded areas represent the non-modular adder (as described in Table I), while the light shaded areas implement the modulo operation.

further optimized to reduce resource usage: the modular adder unit is shown on the left, while on the right the pipelined operations have been compacted onto 8 computational nodes only, without penalty for the performance. This kind of optimization, while demonstrated only for a 128-bit modular adder, seamlessly scales to larger input sizes, requiring only 8 nodes and $2n + 6$ clock cycles, where n is the number of words of the input.

Table II provides an implementation of the basic 32 bit multiplier unit using 16 bit multipliers provided in the target architecture. The 32-bit multiplier, not available in the original architecture, is obtained as a software function unit using the native 16 bit multipliers. The word-by-vector multiplication is obtained by juxtaposing 32 bit multipliers, followed by a multi-precision (non modular) adder unit that handles carries. This method of obtaining larger units by composing smaller ones is fully developed in the generation of hierarchic software function units: an adder and the word-by-vector multiplier are used to design the Montgomery multiplier.

The Montgomery multiplier is based on the core loop shown in Algorithm V.1, where A and B are the input operands, while N is the modulus, w is the size of the word, $b = 2^w$ and N'_0 is the least significant word of the modular inverse of N, modulo the Montgomery radix. In this implementation, the number of iterations performed is $n + 2$ to bound the result in the range $[n, 2n]$ for multiplicands up to $2n$. This is achieved by eliminating the final subtraction in the original Montgomery algorithm and, as a consequence, after the inputs are converted in the Montgomery domain, the operations of the high-level algorithm are all performed therein. Note that the composition of the larger function unit takes into account the shape of the scheduled code of the component

Algorithm V.1: Montgomery multiplier core loop.

```
Input: $x, y \in [0, 2N), NN' - RR' = 1, n = \lg_2\lceil N\rceil, b = 2^w, R = b^n, N' \equiv N^{-1} \mod R = (N'_{n-1}, N'_{n-2}, \dots, N'_0)_b, R' \equiv R^{-1} \mod N$
Output: $xyR^{-1} \mod N \in [0, 2N)$
1 $x \leftarrow 0$
2 for $i \leftarrow 0$ to $n + 1$ do
3     $t \leftarrow N'_0 x_0 \, mod \, b$
4     if $t \neq 0$ then
5         $x \leftarrow x + tN$
6     $x \leftarrow x >> w + A_i B$
7 return $x$
```

units: by compacting the pipelined computations, it is possible to achieve a performance gain that would not be possible were the components implemented as functions. C functions either require call mechanisms that enforce a barrier synchronization between the two computation steps, or inline mechanisms that would lead back to the explosion in the node number of the dataflow graph.

B. High-level Scheduling

Given the software function units described in Section V-A, in order to implement a public key cryptographic primitive, we need to encode it in terms of the software function units. Then, we can explore the possible high-level schedules by means of automatic scheduling tools, such as those presented in [35].

For the Tate pairing algorithm in characteristic p, Figure 4 shows the dataflow graph of the doubling step of the core loop body. The nodes are arranged so that high-level parallelism is emphasized, following an ASAP scheduling policy without resource constraints, thereby showing the maximum available parallelism at any given time. The figure highlights the presence of a significant amount of parallelism, making the exploration of performance vs. area tradeoffs worth being conducted. The typical structure of the Miller's algorithm, upon which the implemented Tate pairing algorithm [17] is based, includes a conditional branch that is only taken when the scan of the binary expansion of the scalar r (see Section I) meets a 1. The implementations ensure that the Hamming weight of r is minimal – in the range of 1 to 10. Since this operation is rarely executed (less than 1% of the iterations), it is handled in a tiled architecture such as DSPfabric by the intervention of the controller processor, which causes the coprocessor control to flow from the main iteration loop to a secondary code that is optimized for the branch execution. The alternative of predicating the branch code is feasible, but the size of the secondary code and the fact that the primary path is much faster (it does not have any instruction to execute) would cause the predicated code to negatively affect the performance.

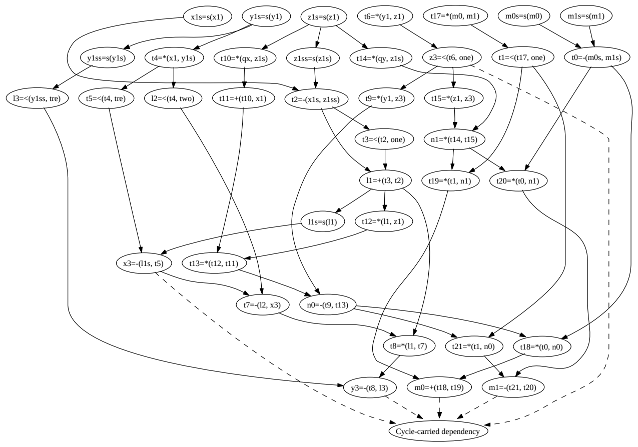


Figure 4. High-level scheduling of the dataflow graph for the doubling step of the Tate pairing algorithm [17]. $<$ is the modular left shift operator, $+/-$ are the modular adder/subtracter, and $*$ is the Montgomery multiplier

C. IDEA

The International Data Encryption Algorithm (IDEA) is a symmetric key block cipher developed by Lai and Massey and published in 1990 [36]. At that time it was suggested as a candidate to replace DES, however its widest adoption has been in PGP which has insured widespread use of the algorithm. Commercial applications show enforcements of the encryption algorithms for IPVT stream decoding.

IDEA uses a 128–bit key to encrypt data blocks of 64 bits by means of an iterative process made of eight rounds followed by a half-round that provides a 64–bit encrypted output. IDEA makes use of three 16–bit operations to implement strong cryptographic confusion properties: 16–bit XOR, 16-bit addition (modulo 2^{16}), and 16-bit multiplication (modulo $2^{16}+1$, a Fermat prime). IDEA round keys are generated using a non-standard rotate-left of 25 bits on the provided 128–bit key. The first eight keys are provided by the input key, and each additional set of eight keys is generated by performing a circular left shift of 25 bits of the previous eight key set. As each round requires only six keys, the subkey bits used differ within each round, providing an effective mechanism for bit variance within the keys used.

IDEA can therefore be decomposed in a small set of primitives can be: modular addition, modular multiplication, and bitwise exclusive or. The most complex primitive, in a software implementation, is the modular multiplication. We present here the implementation of this primitive as a function unit. The remaining primitives and the composition of the complete system are straightforward.

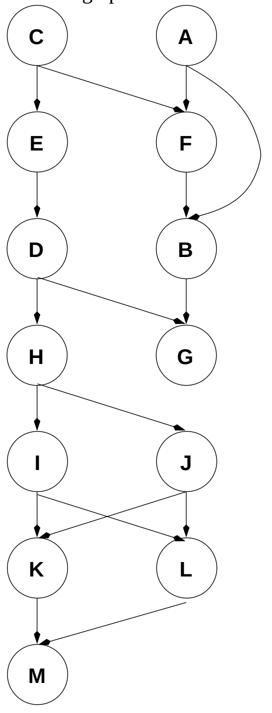
Figure 5 shows a simplified version of the DSPfabric implementation of the modular multiplication used in IDEA, not scheduled. Rx and Ry are the registers that hold the values of the data block and key (actually, several registers are needed when the code is scheduled, to hold copies that are to be moved to different clusters). Registers $R1$, $R2$, $R3$ and $R6$ hold the constants (pre-loaded before starting the coprocessor), while the remaining registers hold short lived temporaries. DSPfabric uses a set of predicated registers to allow efficient conversion of conditionals to predicated executions. Instructions are only performed if all input predicates are true. The *setv* and *setnv* instructions set (to true or false) the predicate of the destination register if the source register is (or is not) zero. An alternate implementation, using *slct* instructions as in the Tate Pairing case study is also possible.

Figure 6 shows the copies that must be performed between the instructions of the modular multiplication algorithm. It can be seen that each instruction can be implemented in a different cluster, having at most two incoming and two outcoming connections. This allows a full pipelining of the algorithm, giving a throughput of one block per cycle.

Figure 5. Modular multiplication for the IDEA cryptosystem

```
A: setv  R1 if Ry
B: setnv Ry if not Ry
C: add   Rx Rx R2
D: setv  R6 if Rx
E: setnv Rx if not Rx
F: sub   Rx Rx R1
G: sub   Rx Ry R6
H: mul   Rx Rx Ry
I: and   Rx Rx R2
J: sub   R7 Rx R3
K: cmplt R5 Rx R7
L: sub   Rx Rx R7
M: add   Rx Rx R5
```

Figure 6. Data transfer graph for the modular multiplication



VI. EXPERIMENTAL RESULTS

In this Section we provide experimental evidence to support the effectiveness of the proposed approach. First, we gauge the complexity of the software function units in terms of both area (that is, number of CPUs) and latency. Table III summarizes the computational complexity for the simpler units, while Tables IV and V show the complexity of two different implementations of the Montgomery multiplier. Analytically, these figures can be derived from Equations (1) and (2), where Equation (1) represents the basic version of the Montgomery multiplier, while Equation (2) refers to the area-optimized version of the same unit, which splits the A_iB word-by-vector multiplication in Algorithm V.1 to execute it in parallel with tN and the subsequent addition, to reduce the number of used processors.

$$T = (n+2)\left(\frac{52n}{cpu} + (2n+1) + 5\right) \qquad (1)$$
$$8 \leq cpu \leq 4n$$

$$T = (n+2)\left(\frac{26n}{cpu} + max\left\{\frac{26n}{cpu}, (2n+1)\right\} + 5\right) \qquad (2)$$
$$16 \leq cpu \leq 2n$$

In these equations, n is the number of input words, while cpu is the total number of nodes in the tiled architecture.

In order to evaluate the effectiveness of the high-level scheduling, we perform a multiobjective exploration of the design space defined by the architectural parameters, that is the number of Montgomery multipliers, modular adders and shifters available in the system, as well as the implementation of the employed Montgomery multipliers, as described in Tables IV and V.

TABLE III.
COMPLEXITY OF THE SOFTWARE IMPLEMENTATIONS OF FINITE FIELD OPERATIONS IN TERMS OF NUMBER OF INPUT WORDS $n = \lceil \log_2 m \rceil / w$.

Finite Field Operations	Clock Cycles	# of CPUs
$x \pm y \; mod \; m$	$2n + 6$	8
$x \cdot y \; mod \; m$	$(n+1)(2n+19)$	$2n$
$x << z \; mod \; m$	$2n + 2$	8

TABLE IV.
EXECUTION TIME AND TIME/AREA PRODUCT FOR THE SOFTWARE IMPLEMENTATION OF THE MONTGOMERY MULTIPLIER AS A FUNCTION OF INPUT WORDS AND NUMBER OF EMPLOYED CPUS.

Input size n	Number of CPUs	Time [clk]	Time $\times$ Area [clk$\times$#CPU]
4	8	200	1600
4	16	135	2160
6	8	399	3192
6	16	259	4144
8	8	666	5328
8	16	432	6912
8	32	315	10080
16	8	2414	19312
16	16	1530	24480
16	32	1088	34816
16	64	867	55488

Figure 7 sketches the Pareto frontier for the multiobjective exploration problem of finding the best configurations in terms of both area and latency. The notion of Pareto optimality states that a solution is optimal if it is impossible to find a solution which improves on one or more of the objectives without worsening any other of them. If one solution is better in one objective than another solution and not worse in any other objectives, the latter is dominated by the former, which is always preferred. This set of solutions is called the Pareto frontier and is guaranteed to contain all optimal solutions, whatever way the individual objectives are weighted relative to each other. To put it in other words: the Pareto frontier exactly captures the available trade-offs between the different objectives. The Pareto frontier shown in Figure 7 gives a set of possible solutions. Then, time or area constraints should be applied to select the best solution. If no constraint is specified, then it is possible to observe that the optimum point using a time $\times$ area figure of merit is the architecture with 4 Montgomery multipliers, each implemented on 16 nodes, plus one shifter and one adder, which needs just over 1 million cycles to perform the entire pairing primitive.

TABLE V.
TIMINGS OF THE SW IMPLEMENTATION OF THE MONTGOMERY MULTIPLIER USING HIGH LEVEL PARALLELIZATION, AS A FUNCTION OF INPUT WORDS AND NUMBER OF EMPLOYED CPUS.

Input size n	Number of CPUs	Time [clk]	Time $\times$ Area [clk$\times$#CPU]
8	16	315	5040
16	16	1088	17408
16	32	867	27744

However, if the goal is to optimize time, then, by employing large hardware resources, it is possible to cut down the execution times by 30%. On the other hand, if a compact device (e.g., 48 CPUs) is required, there is slowdown by a factor of 2 with respect to the time $\times$ area optimum.

The exploration also allows to better evaluate the implementations of the individual units. In our case, it shows that the 16 CPUs Montgomery multiplier implementation is superior to the equivalent implementations on 32 or 8 CPUs. Comparing our approach with FPGA competitors is difficult, since related works [33], [37] are based on

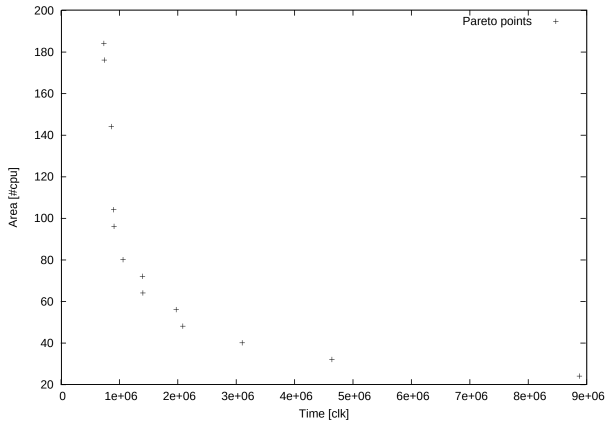


Figure 7. Pareto frontier for the area/latency tradeoff as a multiobjective goal function.

different arithmetic, while the current trend is to employ *mod p*-based cryptosystems (see Section I). Nevertheless, one may note that the FPGA implementations of the Tate pairing primitive use a wide portion of a somewhat larger FPGA device. Moreover, while for processors is it possible to obtain area estimates, the measurement of the physical area of FPGA implementations is widely dependent on CLB interconnection and pin layout. Therefore, the CLB count of an FPGA implementation gives no clue on the actual area occupied by the design. For the proposed implementation, coprocessors are based on a DSPFabric chip using a 7 mm^2 die for 64 nodes, which is a mean figure with respect to the range of possibilities illustrated in the experimental evaluation.

A comparison can still be given with a high-end embedded processor such as the 32-bit StrongARM, which is reported to execute the same pairing computation in over 60 million cycles [17] at 206 MHz, while an FPGA implementation [33] requires about 6300 cycles at 15 MHz. The time $\times$ area optimum of our exploration requires 1 million cycle, on a processor that can be clocked up to 400 MHz. Our solution is, as expected, midway between the pure software and pure hardware solutions, though its performances position it nearer to the hardware solution.

With respect to competitor technologies, tiled architectures using the proposed methodology give the following advantages: smooth scalability (tiled architectures provide excellent scalability properties w.r.t. standard VLIW or superscalar architectures) and quick development cycle (almost as fast as software development).

VII. Concluding Remarks

In this paper, we propose a novel programming model for tiled architectures, suitable for computationally intensive public key cryptographic algorithms. Our proposal is supported by a case study on the DSPFabric reconfigurable architecture, focusing on the implementation of the Tate pairing primitive. Results prove that large amounts of parallelism can be exploited, yielding speedups of more than one order of magnitude with respect to state of the art software implementations. Finally, note that tiled architectures have a rather contained power consumption with a uniform distribution. Such feature is a competitive edge w.r.t. FPGA implementations, which are known to have poor performances from this point of view. As a future development, the methodology developed in this work could be fully automated, by designing a dedicated programming language and its compiler tool-chain and integrating the scheduling algorithm within the compiler back-end.

Acknowledgment

The authors wish to thank Joël Cambonie of STMicroelectronics for his help with the details of the DSPfabric architecture.

References

[1] B. Pfitzmann and M. Waidner, “Asymmetric fingerprinting for larger collusions.” in *ACM Conference on Computer and Communications Security*, 1997, pp. 151–160.

[2] ——, “Anonymous fingerprinting.” in *EUROCRYPT*, 1997, pp. 88–102.

[3] K. Kurosawa and Y. Desmedt, “Optimum traitor tracing and asymmetric schemes.” in *EUROCRYPT*, 1998, pp. 145–157.

[4] A. Joux, “A one round protocol for tripartite diffie-hellman,” in *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*. London, UK: Springer-Verlag, 2000, pp. 385–394.

[5] D. Boneh and M. K. Franklin, “Identity-based encryption from the weil pairing,” in *CRYPTO ’01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 2001, pp. 213–229.

[6] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” in *ASIACRYPT ’01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2001, pp. 514–532.

[7] A. Joux, “The weil and tate pairings as building blocks for public key cryptosystems,” in *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*. London, UK: Springer-Verlag, 2002, pp. 20–32.

[8] R. Dutta, R. Barua, and P. Sarkar, “Pairing-based cryptographic protocols: A survey,” Cryptology ePrint Archive, Report 2005/64, 2004.

[9] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, “Efficient algorithms for pairing-based cryptosystems,” in *CRYPTO ’02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 2002, pp. 354–368.

[10] S. D. Galbraith, K. Harrison, and D. Soldera, “Implementing the tate pairing,” in *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*. London, UK: Springer-Verlag, 2002, pp. 324–337.

[11] P. C. van Oorschot and M. J. Wiener, “Parallel collision search with cryptanalytic applications,” *Journal of Cryptology*, vol. 12, no. 1, pp. 1–28, March 1999, online Date Thursday, February 19, 2004.

[12] D. Freeman, “Constructing pairing-friendly elliptic curves with embedding degree 10,” Cryptology ePrint Archive, Report 2006/026, 2006.

[13] P. Barreto and M. Naehrig, “Pairing-friendly elliptic curves of prime order,” 2005.

[14] M. Scott, “Scaling security in pairing-based protocols,” Cryptology ePrint Archive, Report 2005/139, 2005.

[15] M. Scott and P. S. L. M. Barreto, “Compressed pairings.” in *CRYPTO*, ser. Lecture Notes in Computer Science, M. K. Franklin, Ed., vol. 3152. Springer, 2004, pp. 140–156.

[16] V. Miller, “Short programs for functions on curve,” Unpublished manuscript, 1986, avaliable at http://crypto.stanford.edu/miller/miller.pdf.

[17] M. Scott, “Computing the tate pairing.” in *CT-RSA*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 3376. Springer, 2005, pp. 293–304.

[18] M. B. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, “Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams,” in *ISCA ’04: Proceedings of the 31st annual international symposium on Computer architecture*. Washington, DC, USA: IEEE Computer Society, 2004, p. 2.

[19] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "Wavescalar," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, p. 291.

[20] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, N. Ranganathan, D. Burger, S. W. Keckler, R. G. McDonald, and C. R. Moore, "Trips: A polymorphous architecture for exploiting ilp, tlp, and dlp," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 62–93, 2004.

[21] M. B. Taylor and W. Lee, "Scalar operand networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 145–162, 2005, member-Saman P. Amarasinghe and Member-Anant Agarwal.

[22] J. Cambonie, "A hierarchical reconfigurable computer architecture," Patent.

[23] B. R. Rau, "Iterative modulo scheduling: an algorithm for software pipelining loops," in *MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture*. New York, NY, USA: ACM Press, 1994, pp. 63–74.

[24] B. R. Rau, M. S. Schlansker, and P. P. Tirumalai, "Code generation schema for modulo scheduled loops," in *MICRO 25: Proceedings of the 25th annual international symposium on Microarchitecture*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 158–169.

[25] G. Desoli, "Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach," Hewlett-Packard Laboratories, Tech. Rep. HPL-98-13, Feb 1998.

[26] M. M. Fernandes, J. Llosa, and N. Topham, "Distributed modulo scheduling," in *HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 1999, p. 130.

[27] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. Amarasinghe, "Space-time scheduling of instruction-level parallelism on a raw machine," in *ASPLOS-VIII: Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM Press, 1998, pp. 46–57.

[28] V. S. Lapinskii, M. F. Jacome, and G. A. D. Veciana, "Cluster assignment for high-performance embedded vliw processors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 3, pp. 430–454, 2002.

[29] M. Chu, K. Fan, and S. Mahlke, "Region-based hierarchical operation partitioning for multicluster processors," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, NY, USA: ACM Press, 2003, pp. 300–311.

[30] M. Scott, N. Costigan, and W. Abdulwahab, "Implementing cryptographic pairings on smartcards," Cryptology ePrint Archive, Report 2006/144, 2006.

[31] S. Kwon, "Efficient tate pairing computation for elliptic curves over binary fields." in *ACISP*, ser. Lecture Notes in Computer Science, C. Boyd and J. M. G. Nieto, Eds., vol. 3574. Springer, 2005, pp. 134–145.

[32] P. S. L. M. Barreto, S. Galbraith, C. O. hEigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," Cryptology ePrint Archive, Report 2004/375, 2004.

[33] T. Kerins, W. P. Marnane, E. M. Popovici, and P. S. L. M. Barreto, "Efficient hardware for the tate pairing calculation in characteristic three." in *CHES*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 412–426.

[34] T. Kerins, E. M. Popovici, and W. P. Marnane, "Algorithms and architectures for use in fpga implementations of identity based encryption schemes." in *FPL*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds., vol. 3203. Springer, 2004, pp. 74–83.

[35] G. Bertoni, L. Breveglieri, P. Fragneto, and G. Pelosi, "Parallel hardware architectures for the cryptographic tate pairing." in *ITNG*. IEEE Computer Society, 2006, pp. 186–191.

[36] X. Lai and J. L. Massey, "A proposal for a new block encryption standard." in *EUROCRYPT*, 1990, pp. 389–404.

[37] R. Ronan, C. O'Eigeartaigh, C. C. Murphy, M. Scott, T. Kerins, and W. P. Marnane, "An embedded processor for a pairing-based cryptosystem." in *ITNG*. IEEE Computer Society, 2006, pp. 192–197.

[38] *Third International Conference on Information Technology: New Generations (ITNG 2006), 10-12 April 2006, Las Vegas, Nevada, USA*. IEEE Computer Society, 2006.