Programming Highly Parallel Reconfigurable Architectures for Public-Key Cryptographic Applications

***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. 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.**

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.

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 computationally intensive cryptographic pairings, which are common in modern public key algorithms.

A cryptographic pairing is a bilinear map between two groups $\mathbf{G}_1$, $\mathbf{G}_2$ in which 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 resolve several open problems in cryptography such as, one-round three-way key exchange [15], identity-based encryption [6], and short digital signatures [7]. For ulterior deepenings on the protocols that make use of pairing primitives we send back to [11], [16] and their further references.The Weil and Tate pairings on elliptic curves over finite fields represents 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 [4], [14]. The pairing is considered to be secure if taking 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 [34]. 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 [2], [13], [28] 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 [4], [14], [29]. The cryptographic usage of the Tate pairing involves the application of Miller's Algorithm [22] 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 [27]. 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 a 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.

The rest of this paper is organized as follows. Section II introduces tiled architectures and their interconnection structure. Section III outlines the proposed programming model. Section IV provides an experimental evaluation of the proposed programming model. Finally, Section V draws the conclusions and suggests future research directions.

II. 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 [33], Wavescalar [31] and TRIPS [26], 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* [32]. 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.

Tiled architectures aim at addressing critical problems in high performance processor design, especially design complexity and manufacturing fault rates, by replacing complex processors with smaller and simpler replicated processing elements. The typical applications range from general purpose (for high-end tiled architectures with private data caches for each node) to DSP (for more compact designs, with centralized data cache or streaming data access). Tiled architecture fill a niche between the static general purpose and DSP processors, and the FPGA-based reconfigurable systems. They often expose a degree of reconfigurability in the scalar operand network, allowing the communications among clusters to be tailored to suit the application.

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 architectures belongs to the family of Scalar Operand Networks (SON), and can be characterized by the AsTrO taxonomy [32], 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 [8], 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 itself inside each cluster, again with an array of 4-issue processing elements, connected together by multiplexers with capacity M. The last

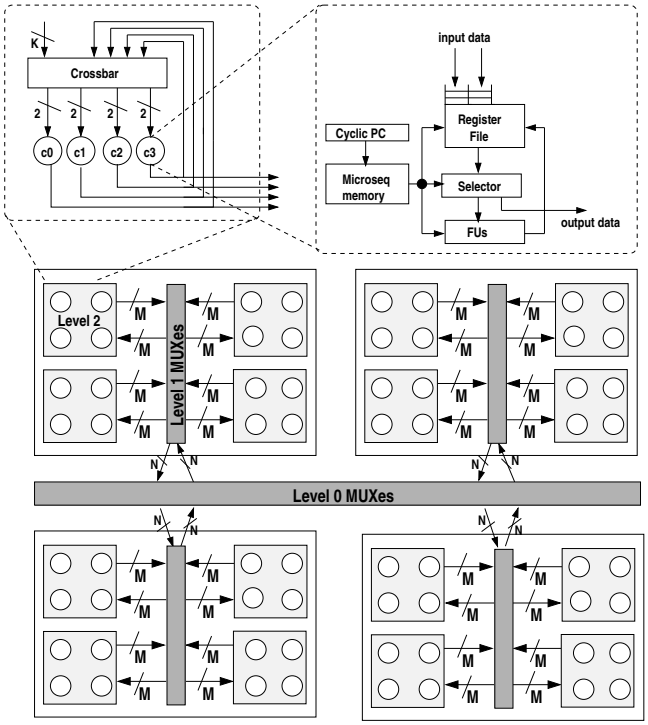

Fig. 1. A DSPFabric architecture with 64 nodes

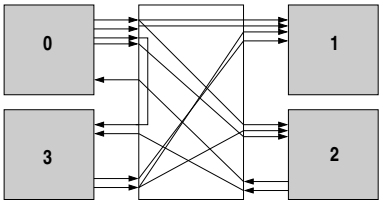

Fig. 2. A feasible interconnection among clusters sets. Assuming the output and input capacities of 4 wires, cluster set 0 and 1 have saturated their available output and input wires, respectively

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-clusters 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 have no 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 in the requesting cluster register file.

III. 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.

A. Compiler Techniques for Tiled Architectures

Tiled architectures are specifically designed for the execution of computationally intensive kernels of multimedia architecture. A typical scenario is to employ such machines as innermost loop accelerators – implemented as coprocessors and coupled with the central processing system.

Multimedia applications spend most of their execution time in few kernel algorithms, i.e. Inverse Cosine Discrete Transform, interpolation and deblocking filters. These loops are characterized by largerly 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.

The compiler is typically driven by in-code pragmas, which select the loops to map onto the multiclustered 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 behaviour 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 [9], [10], [12], [20], [21].

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].

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	

TABLE II

IMPLEMENTATION OF THE 32×32 MULTIPLIER AS A SOFTWARE 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	

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 unschedulable code.

When obeserved at a high level, the Tate pairing algorithm is a single loop that presents high parallelism at the level of operations between very long words. These operations, if written in a high level source code as C, will appear as loops over the length of the operands, exposing an internal degree of parallelism. The only way to exploit the high level parallelism is to completely unroll all the internal loops into the outermost one, and then try to apply the modulo scheduling pass over the whole 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 one 300000 nodes for a 512bit Tate pairing implementation.

We propose in this paper a novel compilation approach, which allows to exploit the available parallelism, decoupling the problem in 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 [3], [19], [30] 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 [17], [18]. 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 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 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 makes for a very regular structure that can be easily mapped to the configurable connections between computational nodes, as each node needs to synchronize only with its neighbours.

Each software functional 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 functional 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 functional units by means of a list-based scheduling algorithm [5].

C. Case Study: Modular Arithmetics

The goal of this Section is to describe the design of a basic multiprecision 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 III.1.

Table I shows the basic schema for a modular adder. Each column of the table represent 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 carry and with no carry. The table considers the case of only three words multiprecision operands, but the extension to larger sizes is straightforward.

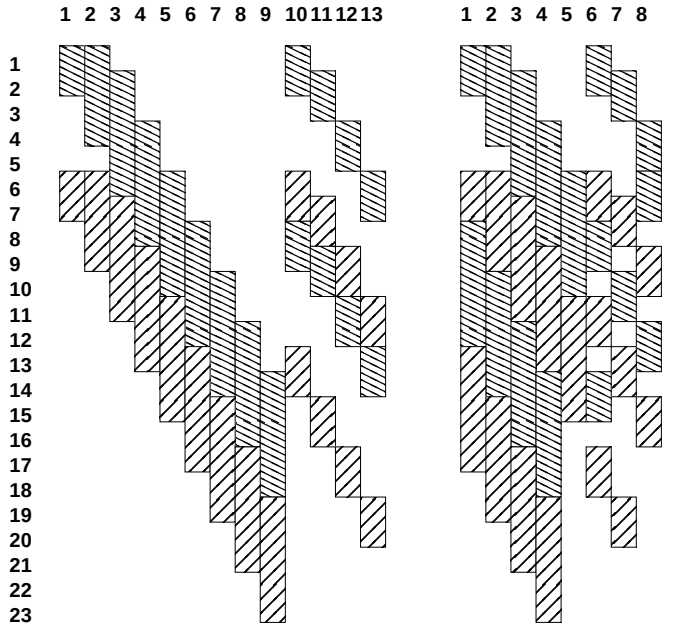

Fig. 3. Time/space scheduling of a 128-bit modular addition: the darker shaded areas represent the non-modular adder (as described in Table I), while the lighter shaded areas implement the modulo operation

Figure 3 shows how the adder can be 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 word-by-vector multiplication is obtained by juxtaposing 32 bit multipliers, followed by a multiprecision (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 III.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.

Algorithm III.1: Montgomery multiplier core loop

```
1 x ← 0
2 for i ← 0 to n + 1 do
3     if N'_0 x_0 mod b ≠ 0 then
4         x ← x + tN
5     x ← x >> w + A_i B
6 return x
```

Note that the composition of the larger function unit takes into account the shape of the scheduled code of the component 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 nodes number of the dataflow graph.

D. High-level Scheduling

Given the software functional units described in Section III-C, in order to implement a public key cryptographic primitive, we need to encode it in terms of the software functional units. Then, we can explore the possible high-level schedules by means of automatic scheduling tools, such as those presented in [5].

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 emphatized, following an ASAP scheduling policy with no 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 [27] is based, includes a conditional branch that is only taken when the scan of binary expansion of the scalar r (see Section I) meets a 1.

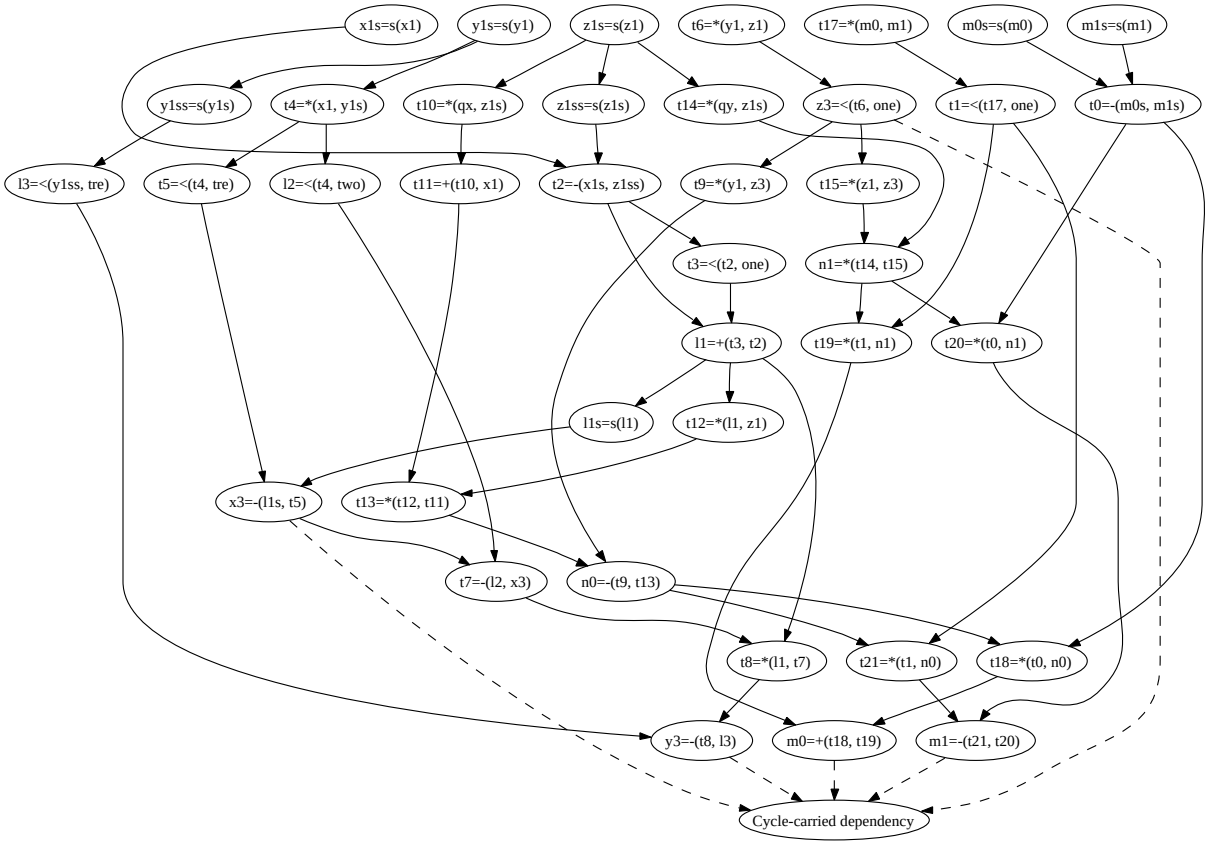

Fig. 4. High-level scheduling of the dataflow graph for the doubling step of the Tate pairing algorithm [27]. $<$ is the modular left shift operator, $+/-$ are the modular adder/subtracter, and $*$ is the Montgomery multiplier

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 has no instructions to execute) would cause the predicated code to negatively affect the performance.

IV. 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 complexity data for the simpler units, while Tables IV and V show the complexity of two different implementations of the Montgomery multiplier. Analytically, these data 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 that splits the A_iB word-by-vector multiplication in Algorithm III.1 to execute it in parallel with tN and the subsequent addition, to reduce the number of processors used.

TABLE III

Complexity of 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
$x \pm y \bmod m$	$2n + 6$	8
$x \cdot y \bmod m$	$(n + 1)(2n + 19)$	$2n$
$x << z \bmod m$	$2n + 2$	8

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

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

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

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

TABLE V

EXECUTION TIME AND TIME/AREA PRODUCT FOR THE SOFTWARE 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

of the Montgomery multipliers employed, as described in Tables IV and V.

Figure 5 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 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.

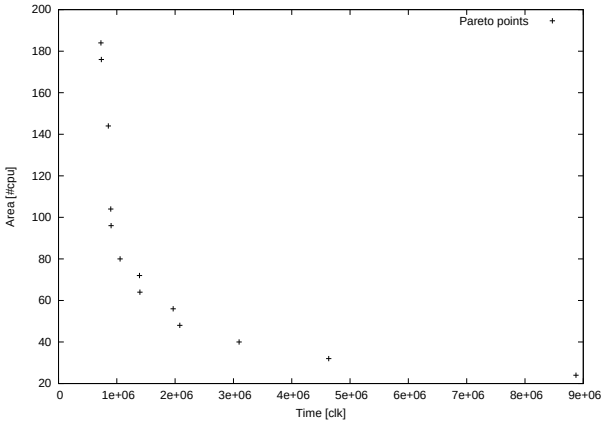

Fig. 5. Pareto frontier for the area/latency tradeoff as a multiobjective goal function

Note that the Pareto frontier shown in Figure 5 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.

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 [17], [25] are based on different arithmetic, while the current trend is to employ *mod p*-based cryptosystems (see Section I). 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 based on DSPfabric size at 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.

On the other hand, a comparison can 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 [27].

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

V. CONCLUDING REMARKS

In this paper, we propose a novel programming model for tiled architectures, suitable for computationally intensive public key cryptographic applications.

Our proposal is supported by a case study on the DSPfabric reconfigurable tiled architecture, focusing on the implementation of the Tate pairing primitive, which is at the core of all identity based cryptographic protocols.

Results prove that large amounts of parallelism can be extracted and exploited, yielding speedups of one order of magnitude with respect to state of the art software implementations.

As a future development, the metodology developed in this work could be fully automated, by designing a dedicated programming language and its compiler toolchain and integrating the scheduling algorithm within the compiler backend.

REFERENCES

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

[2] P. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order, 2005. `http://eprint.iacr.org/2005/133.pdf`.

[3] Paulo S. L. M. Barreto, Steven Galbraith, Colm O hEigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. `http://eprint.iacr.org/`.

[4] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, London, UK, 2002. Springer-Verlag.

[5] Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, and Gerardo Pelosi. Parallel hardware architectures for the cryptographic tate pairing. In *ITNG* [1], pages 186–191.

[6] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229, London, UK, 2001. Springer-Verlag.

[7] Dan Boneh, Ben Lynn, and Hovav 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*, pages 514–532, London, UK, 2001. Springer-Verlag.

[8] Joel Cambonie. A hierarchical reconfigurable computer architecture. Patent (application number 05112850).

[9] Michael Chu, Kevin Fan, and Scott 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*, pages 300–311, New York, NY, USA, 2003. ACM Press.

[10] Giuseppe Desoli. Instruction Assignment for Clustered VLIW DSP Compilers: A New Approach. Technical Report HPL-98-13, Hewlett-Packard Laboratories, Feb 1998.

[11] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2005/64, 2004. `http://eprint.iacr.org/2004/64.pdf`.

[12] Marcio Merino Fernandes, Josep Llosa, and Nigel Topham. Distributed modulo scheduling. In *HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture*, page 130, Washington, DC, USA, 1999. IEEE Computer Society.

[13] David Freeman. Constructing pairing-friendly elliptic curves with embedding degree 10. Cryptology ePrint Archive, Report 2006/026, 2006. `http://eprint.iacr.org/2006/026.pdf`.

[14] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.

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

[16] Antoine 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*, pages 20–32, London, UK, 2002. Springer-Verlag.

[17] Tim Kerins, William P. Marnane, Emanuel M. Popovici, and Paulo S. L. M. Barreto. Efficient hardware for the tate pairing calculation in characteristic three. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2005.

[18] Tim Kerins, Emanuel M. Popovici, and William P. Marnane. Algorithms and architectures for use in fpga implementations of identity based encryption schemes. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *FPL*, volume 3203 of *Lecture Notes in Computer Science*, pages 74–83. Springer, 2004.

[19] Soonhak Kwon. Efficient tate pairing computation for elliptic curves over binary fields. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2005.

[20] Viktor S. Lapinskii, Margarida F. Jacome, and Gustavo A. De Veciana. Cluster assignment for high-performance embedded vliw processors. *ACM Trans. Des. Autom. Electron. Syst.*, 7(3):430–454, 2002.

[21] Walter Lee, Rajeev Barua, Matthew Frank, Devabhaktuni Srikrishna, Jonathan Babb, Vivek Sarkar, and Saman 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*, pages 46–57, New York, NY, USA, 1998. ACM Press.

[22] Victor Miller. Short programs for functions on curve. Unpublished manuscript, 1986. Avaliable at `http://crypto.stanford.edu/miller/miller.pdf`.

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

[24] B. Ramakrishna Rau, Michael 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*, pages 158–169, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[25] Robert Ronan, Colm O'Eigeartaigh, Colin C. Murphy, Michael Scott, Tim Kerins, and William P. Marnane. An embedded processor for a pairing-based cryptosystem. In *ITNG* [1], pages 192–197.

[26] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Nitya Ranganathan, Doug Burger, Stephen W. Keckler, Robert G. McDonald, and Charles R. Moore. Trips: A polymorphous architecture for exploiting ilp, tlp, and dlp. *ACM Trans. Archit. Code Optim.*, 1(1):62–93, 2004.

[27] Michael Scott. Computing the tate pairing. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2005.

[28] Michael Scott. Scaling security in pairing-based protocols. Cryptology ePrint Archive, Report 2005/139, 2005. `http://eprint.iacr.org/2005/139.pdf`.

[29] Michael Scott and Paulo S. L. M. Barreto. Compressed pairings. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2004.

[30] Michael Scott, Neil Costigan, and Wesam Abdulwahab. Implementing cryptographic pairings on smartcards. Cryptology ePrint Archive, Report 2006/144, 2006. `http://eprint.iacr.org/`.

[31] Steven Swanson, Ken Michelson, Andrew Schwerin, and Mark Oskin. Wavescalar. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 291, Washington, DC, USA, 2003. IEEE Computer Society.

[32] Michael Bedford Taylor and Walter Lee. Scalar operand networks. *IEEE Trans. Parallel Distrib. Syst.*, 16(2):145–162, 2005. Member-Saman P. Amarasinghe and Member-Anant Agarwal.

[33] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant 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*, page 2, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, March 1999. Online Date Thursday, February 19, 2004.