Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings

Patrick Longa

Microsoft Research, USA
plonga@microsoft.com

Abstract. We propose a novel approach that generalizes interleaved modular multiplication algorithms to the computation of sums of products over large prime fields. This operation has widespread use and is at the core of many cryptographic applications. The method reformulates the widely used lazy reduction technique, crucially avoiding the need for storage and computation of "double-precision" operations. Moreover, it can be easily adapted to the different methods that exist to compute modular multiplication, producing algorithms that are significantly more efficient and memory-friendly. We showcase the performance of the proposed approach in the computation of multiplication over an extension field $\mathbb{F}_{p^k}$, and demonstrate its impact with a record-breaking implementation for bilinear pairings: a full optimal ate pairing over the popular BLS12-381 curve is computed in under half a millisecond on a 3.2GHz Intel Coffee Lake processor, which is about 1.40× faster than the state-of-the-art.

Keywords: Sum of products · prime fields · extension fields · bilinear pairings · BLS12-381 · efficient computation.

1 Introduction

Take two sets of integers $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ all defined over a certain finite field $\mathbb{F}_p$ with large prime characteristic p. The sum of their products, namely, the computation $c = \sum_{i=0}^{t-1} \pm a_i \cdot b_i \bmod p$ is a fundamental operation that is at the core of various types of arithmetic over $\mathbb{F}_p$, from matrix multiplication and multiplication over polynomial rings to elliptic curve arithmetic. Of special interest is that this operation has wide use in the form of multiplication over extension fields $\mathbb{F}_{p^k}$, which is at the heart of several cryptographic schemes such as elliptic curves defined over extension fields [35], bilinear pairings on ordinary curves [62] and supersingular isogeny-based schemes [47].

In this work, we propose a new approach that computes a sum of products over $\mathbb{F}_p$ by interleaving intermediate products with the modular reduction step, in similar fashion to classical interleaved modular multiplication algorithms (§1). Crucially, it departs from algorithms using the well-known lazy reduction

technique [65, 69], eliminating the excessive growth of intermediate values and the need of computing "double-precision" arithmetic. The method can be easily adapted to existing algorithmic variants that fit different application profiles, for software and hardware platforms. We show that some of these variants are especially efficient for software implementation, exhibiting strong synergy with computer architectures that leverage the simplicity of schoolbook-like multiplication. The final result is a streamlined implementation with very constrained memory use.

The target. To showcase the potential of the new method, we apply it in the context of bilinear pairings; concretely, optimal ate pairings over the BLS12-381 curve. Bilinear pairings are at the core of a myriad of elegant cryptographic schemes such as identity-based cryptosystems [19, 24, 28, 29] and non-interactive zero-knowledge proof systems [44, 45]. More recently, pairings have attracted great interest because of their novel use in blockchain technology like Zcash [6] and Ethereum 2.0 [26]. Unfortunately, recent advances in the computation of discrete logarithms over extension fields, as those used in pairing-based cryptography [10, 50], have forced an increase in parameter sizes [9, 60]. This, for example, motivated the design of BLS12-381, a new pairing-friendly curve that is conjectured to provide 128 bits of security and is widely used in blockchain protocols [23] (see [64] for details on a standardization effort in the IETF CFRG). The larger parameter sizes, added to the fact that pairing operations were already computationally expensive, increase the need of developing new methods that improve performance.

It is important to note that other operations beside multiplication over extension fields are possible targets for the new method (e.g., elliptic curve arithmetic operations with the form $AB + CD$ over a finite field of large prime characteristic). However, since we expect a larger gain in the case of arithmetic over extension fields, we use this setting in the context of pairings for illustration purposes. Note that other worthwhile applications include post-quantum supersingular isogeny-based protocols such as SQISign [40][1]. Also, although wider in generality, the method is analyzed in the context of Montgomery multiplication [61], which is the most widely used approach for modular multiplication.

Computing sums of products and the case of multiplication over $\mathbb{F}_{p^k}$. There are two main approaches to optimize multiplication over $\mathbb{F}_{p^k}$. On one hand, algebraic transformations are used to reduce the required number of underlying field multiplications. A well-known example of this case is Karatsuba multiplication [49].

The second approach consists in minimizing the number of modular reductions using the so-called *lazy reduction* technique. Lazy reduction, which goes

[1] Other very attractive applications where the method would have had a significant impact include SIDH [47] and SIKE [5]; see App. B. Unfortunately, these protocols were very recently proven insecure in a series of papers starting with [27].

back to at least [69], is an extensively used optimization that has been applied in a wide variety of scenarios [4,30,54,57,65]. The basic principle is very simple: products are computed and left unreduced. A modular reduction is only applied at the very end of the computation, right after the summation of the intermediate, "double-precision" values. This elimination of reductions is highly effective, especially for primes for which the reduction routine is roughly as expensive as the integer multiplication part. If we assume the use of Montgomery multiplication for computing the summation of t n-digit products, the cost is reduced from $t(2n^2 + n)$ multiplies to only $n^2(t + 1) + n$.

In the context of cryptographic pairings, Scott [65] was the first to apply lazy reduction to Karatsuba multiplication in the computation of multiplication over $\mathbb{F}_{p^2}$. Later, the technique was extended to the full towering and curve arithmetic by Aranha et al. [3] (see also [58]). Given an extension field $\mathbb{F}_{p^k} = \mathbb{F}_p[x]/(x^k - \omega)$ with $\omega \in \mathbb{F}_p$, a multiplication in $\mathbb{F}_{p^k}$ exploiting lazy reduction can be performed with only k reductions modulo p, in contrast to the k^2 reductions that would be required with a conventional multiplication.

Many works have exploited the technique, especially in the context of multiplication over $\mathbb{F}_{p^2}$, without too much change in the basic approach [3,7,18,20,21,36,67]. In fact, despite the availability of efficient interleaved modular multiplication algorithms [38], the combination of lazy reduction with these faster algorithms has been long seen as an impossibility. As consequence, lazy reduction has been strictly limited to use non-interleaved modular multiplications.

The main drawback of this traditional approach using lazy reduction is that it forces the storage and computation with intermediate results of double precision. Beside this adding pressure on the memory usage, the optimization of double-precision arithmetic might require specialized routines that increase the complexity of implementations significantly [2,3]. In contrast, the proposed method limits the growth of intermediate results and gets rid of double-precision arithmetic. The simplest variants use schoolbook internally, eliminating further the additional storage demanded by Karatsuba. For small and medium-sized primes, the significant reduction in memory friction pays off in terms of speed, even in the cases where the use of multiplications is higher, as we show in §4.

Outline. We start by giving some preliminary background on algorithmic aspects of Montgomery multiplication and extension field arithmetic in §2. Then, we describe the details of our method in §3, together with an exhaustive classification of its different algorithmic variants. This section also includes a preliminary cost analysis. In §4, we present our case study targeting optimal ate pairings over BLS12-384, and describe suitable algorithmic variants and their efficient implementation using the RELIC library. Finally, we discuss the impact of this work and potential future developments in §5. Appendix A discusses additional algorithmic variants using Karatsuba, and Appendix B describes how to adapt the approach to SIKE, which in turn allows us to evaluate the improvements in speed performance and memory usage for different prime sizes.

2 Preliminaries

2.1 Montgomery multiplication

A well-known and widely-used method to implement modular multiplication is due to Montgomery [61]. This method introduces a significant speedup in the computation of modular reduction by replacing expensive long divisions by simple divisions with powers of two.

To carry out a Montgomery multiplication, field elements are represented in the so-called Montgomery domain. Let $R = 2^N$ and $p' = -p^{-1} \bmod R$, where $N = n \cdot w$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$ and w is the computer wordsize. For two field elements a and b, their Montgomery representation is given by $\tilde{a} = aR \bmod p$ and $\tilde{b} = bR \bmod p$, respectively. If it holds that $\tilde{a}\tilde{b} < pR$, the Montgomery residue $c = \tilde{a}\tilde{b}R^{-1} \bmod p = abR \bmod p$ is then computed as

$$c = (\tilde{a}\tilde{b} + (\tilde{a}\tilde{b}p' \bmod 2^N) \cdot p)/2^N. \tag{1}$$

The final result $a \cdot b$ can then be easily obtained by dividing by the value R. If one assumes that conversions to/from the Montgomery domain are amortized by executing a long series of modular multiplications, the cost of a Montgomery reduction (i.e., without the integer multiplication part) is given by (n^2+n) w-bit multiplications.

Interleaved and non-interleaved modular multiplication There are two general approaches for implementing modular multiplication, which are determined by how the integer multiplication and reduction parts are "glued" together. In an *interleaved* or *non-separated* modular multiplication, both pieces are computed in an intertwined fashion, while the *non-interleaved* or *separated* version computes the full integer multiplication first and then proceeds to carry out the reduction part separately. In applications dealing with sums of products over large prime fields (e.g., for multiplying over extension fields like in pairings or supersingular isogeny-based protocols), the use of non-interleaved modular multiplication has become the *de facto* approach, as it enables a straightforward application of lazy reduction and other advanced implementation techniques [3, 4, 18, 20, 21, 36, 54, 65, 67, 69].

Radix-r Montgomery multiplication. Straight implementations of Eq. (1) demand the use of a high number of registers since the full inputs are processed in a single pass using the full modulus. Let r be a certain radix, typically a power of two, in which operands and the modulus are represented (e.g., an operand a is represented as $(a_{t-1}, \ldots, a_1, a_0)_r$). A more implementation-friendly approach proposed by Dussé and Kaliski Jr. [38] processes the computation one digit at a time reducing with r at each iteration, in what is called the radix-r Montgomery reduction. An *interleaved* computation of a radix-r Montgomery multiplication of two Montgomery elements $\tilde{a}$ and $\tilde{b}$ then proceeds by fixing $p' = -p^{-1} \bmod r$

(assuming that the modulus is a prime p of bitlength l), initializing c to 0, and computing $t = \lceil l/\log_2 r \rceil$ iterations doing

$$c = \big(c + \tilde{a}_i\tilde{b} + ((c + \tilde{a}_i\tilde{b})p' \bmod r) \cdot p\big)/r, \tag{2}$$

for $i = 0, \ldots, t-1$.

In this work, we adopt a generalization of the original radix-r Montgomery multiplication by setting $r = 2^{Bw}$, where $B \in \mathbb{Z}$ and $0 < B \leq n$ and, as before, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$ and w is the computer wordsize[2]. In this case, Eq. (2) runs for $\lceil n/B \rceil$ iterations. This generalization lifts the restriction that the bitlength of the radix r should match the computer wordsize w of a given platform, as originally assumed in [38]. Note that the original radix-r Montgomery multiplication corresponds to the case $B = 1$.

As we will show in §3, the flexibility introduced by B in the definition of the radix allows for a comprehensive generalization that captures many implementation variants of Montgomery multiplication exploiting either the "operand-scanning" form (a.k.a. schoolbook method), the "product-scanning" form (a.k.a. Comba method [31]), the Karatsuba technique [49], and their different combinations. To the best of our knowledge, several of the arising variants are novel.

2.2 Sums of products over large prime fields

Let $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ be two sets of elements all belonging to a certain field $\mathbb{F}_p$ of large prime characteristic p. We define a "sum of products" as a computation with the form

$$c = \sum_{i=0}^{t-1} \pm a_i \cdot b_i \bmod p$$

This operation can be found at the core of many cryptologic computations, the most notable of which is perhaps multiplication over extension fields with large prime characteristic. Hence, we use this operation to illustrate the impact of the proposed algorithms (see §3).

Let's recall the simplest scenario for a multiplication of two elements $a, b \in \mathbb{F}_{p^k}$ modulo an irreducible polynomial with the form $f = x^k - \omega$, where ω is a primitive element in $\mathbb{F}_p^*$ and $k|(p-1)$. The polynomial multiplication is then given by

$$\begin{aligned} c(x) = a(x)b(x) &= \Big(\sum_{i=0}^{k-1} a_i x^i\Big)\Big(\sum_{i=0}^{k-1} b_i x^i\Big) \\ &\equiv c_{k-1}x^{k-1} + \sum_{i=0}^{k-2}(c_i + \omega c_{i+k})x^k \qquad (\bmod\ f(x)), \end{aligned}$$

[2] This generalization is similar to the description by Bos and Friedberger [20, Section 3.2], but without limiting to a special-form prime.

where

$$c_j = \sum_{i=0}^{j} a_i b_{j-i} \bmod p.$$

It is a widespread practice to optimize the implementation for computing each c_j using an "accumulation and reduction" strategy, most commonly known as lazy reduction. This technique effectively reduces the number of modular reductions to only one (or k, for the full polynomial multiplication). Note that, as in most practical scenarios, we assume that ω has small coefficients that make a multiplication by it relatively cheap.

As mentioned before, this use of lazy reduction has some drawbacks, the most critical of which being the need of extra storage and computing with intermediate results of double precision. As we show in §4 (see also App. B), this issue makes implementations slower and less memory-friendly for small and medium-sized primes.

The rise of fast multiplication over $\mathbb{F}_{p^2}$. The most basic "sum of products" operation underlying several cryptographic schemes is multiplication over a quadratic extension field $\mathbb{F}_{p^2}$. Modern examples of these schemes include bilinear pairings on ordinary elliptic curves over prime fields and supersingular isogeny-based protocols.

For illustrative purposes, let's use the common construction fixing $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ for $i^2 - \beta = 0$, where β is a small integer in absolute value. Two main approaches are known to realize the multiplication in this case.

Take two elements $a = (a_0 + a_1 i)$ and $b = (b_0 + b_1 i) \in \mathbb{F}_{p^2}$. The first method to compute the multiplication $a \cdot b$ in $\mathbb{F}_{p^2}$ is the straightforward schoolbook method which computes it as

$$a \cdot b = (a_0 b_0 + a_1 b_1 \beta) + (a_0 b_1 + a_1 b_0) i$$

The second approach is Karatsuba multiplication, which computes the same operation as

$$a \cdot b = (a_0 b_0 + a_1 b_1 \beta) + \big((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1\big) i$$

If we assume that $\beta = -1$ as in most efficient instantiations, the operation count for multiplication using schoolbook is of four modular multiplications and two modular additions/subtractions, while the one for Karatsuba is of three modular multiplications and five modular additions/subtractions.

Efficient implementation of the arithmetic over $\mathbb{F}_{p^2}$ attracted lots of interest from the cryptographic community around the mid-2000's, contributing to a remarkable effort aimed at reducing the high cost of computing cryptographic pairings. In 2007, Scott [65] was the first to apply lazy reduction to Karatsuba multiplication in the context of pairings, changing the cost of one $\mathbb{F}_{p^2}$ multiplication to three integer multiplications, two modular reductions, two additions

and three double-precision additions/subtractions. This approach was later perfectioned by Beuchat et al. [18] and Aranha et al. [3] with the use of some aggressive optimization techniques such as the avoidance of modular corrections and carry-handling elimination, all in the context of the computation of optimal ate pairings [68] over a 254-bit Barreto-Naehrig (BN) curve [15].

The algorithm for multiplication in $\mathbb{F}_{p^2}$ combining all these optimizations for the optimal case with $\beta = -1$ is shown in Algorithm 1. Integer operations without modular correction or reduction are represented as $\times, +$ or $-$. The only operation that requires a modular correction is the subtraction in line 4 that is represented as $\ominus$. Double-precision operands are represented in uppercase, while single-precision operands are in lowercase.

Algorithm 1 Multiplication in $\mathbb{F}_{p^2}$ using Karatsuba and lazy reduction

Input: $a = (a_0 + a_1 i)$ and $b = (b_0 + b_1 i) \in \mathbb{F}_{p^2}$
Output: $c = a \cdot b = (c_0 + c_1 i) \in \mathbb{F}_{p^2}$

1: $T_0 \leftarrow a_0 \times b_0$
2: $T_1 \leftarrow a_1 \times b_1$
3: $t_0 \leftarrow a_0 + a_1$
4: $t_1 \leftarrow b_0 + b_1$
5: $T_2 \leftarrow t_0 \times t_1$
6: $T_3 \leftarrow T_0 + T_1$
7: $T_2 \leftarrow T_2 - T_3$
8: $T_0 \leftarrow T_0 \ominus T_1$
9: $c_0 \leftarrow T_2 \bmod p$
10: $c_1 \leftarrow T_0 \bmod p$
11: **return** $C = (c_0 + c_1 i)$

The case of higher-degree field extensions. For pairings, high-degree extension field arithmetic represents the main building block and, therefore, its efficient implementation becomes crucial. To this end, it has been recommended to implement it as a tower of extensions built with suitable irreducible polynomials [52], following a similar development for optimal extension fields [8]. For example, the following tower scheme to represent $\mathbb{F}_{p^{12}}$ is widely used in many implementations [1, 3, 18, 42]:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, with β a non-square.
- $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[s]/(s^2 - \xi)$, with ξ a non-square.
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, with ξ a non-cube.
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[t]/(t^3 - s)$ or $\mathbb{F}_{p^6}[w]/(w^2 - v)$ or $\mathbb{F}_{p^2}[w]/(\omega^6 - \xi)$, with ξ a non-square and non-cube.

The use of a tower field allows to write modularized implementations, in which each layer can be easily optimized using algebraic transformations like Karatsuba to reduce the number of modular multiplications.

In 2011, Aranha et al. [3] pushed the performance limits further by extending the use of lazy reduction to the full tower scheme, in order to minimize the use of modular reductions. Concretely, they showed that this optimization, when applied above the $\mathbb{F}_{p^2}$ arithmetic up to the $\mathbb{F}_{p^{12}}$ layer, injects an 11%-17% speedup on a variety of x64 processors. Nevertheless, this extended lazy

reduction technique comes at a price. It requires additional specialized routines to perform the double-precision arithmetic, which increase the complexity and memory footprint of the implementation.

In §4, we discuss how one can improve performance and memory use for prime sizes of practical relevance with a new approach that we present next.

3 The Proposed Method

We describe the proposed method in the context of Montgomery multiplication, which is arguably one of the most relevant scenarios. See §5 for a discussion on other potential applications.

Let $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ be two sets of integers with $a_i, b_i \in [0, p)$ for $i = 0, \ldots, (t-1)$ and $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize. From now on, we make the assumption that inputs a_i and b_i are already in the Montgomery domain.

From a general perspective, the new approach essentially consists in performing a *merged* computation of the operation $c = \sum_{i=0}^{t-1} \pm a_i \cdot b_i \bmod p$ using *interleaved* radix-r Montgomery multiplication[3], that is, initializing $c = 0$ and executing $\lceil l/\log_2 r \rceil = \lceil n/B \rceil$ iterations doing

$$c = \Big(c + \sum_{i=0}^{t-1} a_{i,j} b_i + \big((c + \sum_{i=0}^{t-1} a_{i,j} b_i) p' \bmod r\big) \cdot p\Big)/r, \tag{3}$$

for $j = 0, \ldots, \lceil n/B \rceil - 1$, where $p' = -p^{-1} \bmod r$, and each integer a_i is represented in radix-r representation as $(a_{i,\lceil n/B \rceil - 1}, \ldots, a_{i,1}, a_{i,0})_r$. As stated in §2.1, the radix r is adopted in the generalized form $r = 2^{Bw}$, where $B \in \mathbb{Z}$ and $0 < B \leq n$. In the following, we call each digit in this radix representation a "B-digit".

We remark that Eq. (3) is presented in a general form for simplicity purposes. Next, we provide a more detailed description that covers the wide diversity of variants that can be derived from the approach.

At a high-level, we can classify the different variants by the method that is used to implement the top layer in the computation of the multiplications in Eq. (3). Thus, we can distinguish operand-scanning (or schoolbook), product-scanning (or Comba), and Karatsuba variants. In the remainder, we mostly focus on the first case which brings very fast computations to the software platform class that we target in this work. We comment that product-scanning and Karatsuba variants, such as those described in Appendix A, might be useful in other scenarios, e.g., for hardware implementations (see discussion in §5).

Remark 1. The result of a Montgomery reduction is upper bounded by $2p$ when its input is in the range $[0, pR)$. Hence, a conditional subtraction is needed to

[3] As explained before, the non-interleaved or separated case is used with the standard lazy reduction technique.

bring the result to $[0, p)$. However, this operation can be avoided if we perform arithmetic over a redundant representation (e.g., over $\mathbb{Z}_{2p}$). For example, if operands are kept in the range $[0, 2p)$ such that the result of a multiplication is guaranteed to be $c = a \cdot b < 4p^2 \leq pR$ (i.e., it should hold that $R \geq 4p$), then the result of the Montgomery reduction is still going to be bounded by $2p$ but we will no longer require the modular correction. A simple correction is going to be required at the very end of the computations to bring the final result to the canonical range $[0, p)$. In the following, all the algorithms assume the use of this redundant representation to avoid the final conditional subtraction.

3.1 Operand-scanning method

For this method, the computation flow at the top layer follows the operand-scanning or schoolbook form. That is, for each multiplication, a B-digit from the radix-r representation of a given multiplier is first multiplied with the full value of the multiplicand before proceeding to the next computation. For the remainder, we refer to this operation as *B-digit* $\times$ *row* multiplication.

We distinguish two main approaches, depending on whether the inner multiplications $a_{i,j}b_i$ from Eq. (3) are interleaved with the multiplications with the prime p or not. We adopt the naming convention from [53] and call the approaches *finely integrated* if we do the former case (i.e., with interleaved inner multiplications), and *coarsely integrated*, otherwise.

Coarsely integrated variants. The merged "sum of products" algorithm using a coarsely integrated strategy is displayed in Algorithm 2. The construction of the algorithm easily follows from Eq. (3) when $n \bmod B = 0$. We still need to manage the cases in which $n \bmod B \neq 0$ (i.e., the digit-size of the most significant B-digit is strictly smaller than that of a B-digit). This is done in lines 6–9, where the computations are adjusted to the right digit size.

As can be seen, the *B-digit* $\times$ *row* multiplications corresponding to $a_{i,j}b_i$ (line 3) are interleaved with those for the multiplications with p' and p (lines 4 and 5) at each iteration of the for-loop.

There are multiple ways in which the inner multiply-and-accumulate operations $\sum a_{i,j} \cdot b_i$ and $u + q \cdot p$ can be realized. We classify these variants according to the chosen value B as follows:

- Case with $B = 1$: one is setting $r = 2^w$ and all the inner computations essentially become straight *digit* $\times$ *row* multiplications. This is the analogous version of "Improvement 2" from [38], called coarsely integrated operand scanning (CIOS) in [53].
- Case with $B > 1, B \neq n$: the inner computations work on "blocks" of digits and, hence, each *B-digit* $\times$ *B-digit* multiplication can be implemented in either schoolbook, Comba or Karatsuba style (or any combination of these in a *multi-level* fashion for sufficiently large primes).
- Case with $B = n$: this is essentially the original lazy reduction technique.

Algorithm 2 Merged sums of products using radix-r interleaved Montgomery multiplication in coarsely integrated form.

Input: integers $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ s.t. $a_i, b_i \in [0, 2p)$ for $i = 0, \ldots, (t-1)$ and $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; the radix $r = 2^{Bw}$ s.t. $B \in \mathbb{Z}$ and $0 < B \leq n$, and the Montgomery constant $p' = -p^{-1} \bmod r$. Integers are represented in radix r, e.g., $a_i = (a_{i,\lceil n/B \rceil - 1}, \ldots, a_{i,1}, a_{i,0})_r$.
Output: the Montgomery residue $c = \sum_{i=0}^{t-1} a_i b_i \cdot R^{-1} \bmod p$ s.t. $c \in [0, 2p)$.

1: $u \leftarrow 0$
2: **for** $j = 0$ **to** $\lfloor n/B \rfloor - 1$ **do**
3: $\quad u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \cdot b_i$
4: $\quad q \leftarrow u \cdot p' \bmod 2^{Bw}$
5: $\quad u \leftarrow (u + q \cdot p)/2^{Bw}$
6: **if** $n \bmod B \neq 0$ **then**
7: $\quad u \leftarrow u + \sum_{i=0}^{t-1} a_{i,\lceil n/B \rceil - 1} \cdot b_i$
8: $\quad q \leftarrow u \cdot p' \bmod 2^{(n \bmod B)w}$
9: $\quad u \leftarrow (u + q \cdot p)/2^{(n \bmod B)w}$
10: **return** $c \leftarrow u$

Finely integrated variants. The merged "sum of products" algorithm using a finely integrated strategy is displayed in Algorithm 3. In this case, note that multiplications are performed *B-digit* by *B-digit*, interleaving those corresponding to $a_{i,k} b_{i,j}$ (lines 3 and 7) with those for the multiplications with p_k (lines 5 and 8). Note that we manage the case with $n \bmod B \neq 0$ as described before.

Similar to the coarsely integrated form, there are multiple ways in which the inner multiply-and-accumulate operations can be realized. Again, we classify these variants according to the value B as follows:

- Case with $B = 1$: one is setting $r = 2^w$ and all the inner computations become simple *digit* $\times$ *digit* multiplications, where those corresponding to input operands are interleaved with those with the prime. This is the analogous version of the finely integrated operand scanning (FIOS) method from [53].
- Case with $B > 1, B \neq n$: the inner computations work on "blocks" of digits and, hence, each *B-digit* $\times$ *B-digit* multiplication can be implemented in either schoolbook, Comba or Karatsuba style (or any combination of these in a *multi-level* fashion for sufficiently large primes).
- Case with $B = n$: this is essentially the original lazy reduction technique.

Selecting a variant. Picking a specific variant depends on both the modulus size (see the next subsection) and the targeted platform. Generally speaking, the coarsely integrated variant (Algorithm 2) should be the preferred option in most software platforms in which schoolbook works well and the availability of general purpose registers (GPRs) is sufficient to support a full *B-digit* $\times$ *row* multiplication with minimal interaction with the memory. On the other hand, the inner for-loop of the finely integrated variant (Algorithm 3) consists of a

Algorithm 3 Merged sums of products using radix-r interleaved Montgomery multiplication in finely integrated form.

Input: integers $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ s.t. $a_i, b_i \in [0, 2p)$ for $i = 0, \ldots, (t-1)$ and $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; the radix $r = 2^{Bw}$ s.t. $B \in \mathbb{Z}$ and $0 < B \leq n$, and the Montgomery constant $p' = -p^{-1} \bmod r$. Integers are represented in radix r, e.g., $a_i = (a_{i,\lceil n/B \rceil - 1}, \ldots, a_{i,1}, a_{i,0})_r$.

Output: the Montgomery residue $c = \sum_{i=0}^{t-1} a_i b_i \cdot R^{-1} \bmod p$ s.t. $c \in [0, 2p)$.

$u \leftarrow 0$
for $j = 0$ **to** $\lfloor n/B \rfloor - 1$ **do**
    $u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \cdot b_{i,j}$
    $q \leftarrow u \cdot p' \bmod 2^{Bw}$
    $u \leftarrow (u + q \cdot p_0)/2^{Bw}$
    for $k = 1$ **to** $\lceil n/B \rceil - 1$ **do**
        $u \leftarrow u + 2^{(k-1)Bw} \cdot \sum_{i=0}^{t-1} a_{i,k} \cdot b_{i,j}$
        $u \leftarrow u + 2^{(k-1)Bw} \cdot q \cdot p_k$
if $n \bmod B \neq 0$ **then**
    $u \leftarrow u + \sum_{i=0}^{t-1} a_{i,0} \cdot b_{i,\lceil n/B \rceil - 1}$
    $q \leftarrow u \cdot p' \bmod 2^{(n \bmod B)w}$
    $u \leftarrow (u + q \cdot p_0)/2^{(n \bmod B)w}$
    for $k = 1$ **to** $\lceil n/B \rceil - 1$ **do**
        $u \leftarrow u + 2^{(kB - n \bmod B)w} \cdot \sum_{i=0}^{t-1} a_{i,k} \cdot b_{i,\lceil n/B \rceil - 1}$
        $u \leftarrow u + 2^{(kB - n \bmod B)w} \cdot q \cdot p_k$
return $c \leftarrow u$

bunch of multiplications that are independent from each other and, hence, can be executed in parallel on, e.g., an FPGA. See §5 for an extended discussion on other uses for the algorithms.

Regarding the selection of the value B, setting $B > 1$ might make it easier to alleviate memory use for relatively large primes, especially in the case of Algorithm 3. For small and medium size primes[4], it appears that setting $B = 1$ hits the right balance between the size of intermediate results and the number of GPRs available on x64 platforms.

Finally, we mentioned before that for the internal multiplications it is possible to use either schoolbook, Comba, Karatsuba, or any combination of these in a multi-layer implementation. To be efficient, Karatsuba would require a B-digit consisting of a sufficiently large number of limbs. And between schoolbook and Comba, the former is typically preferable when a given platform supports efficient carry-saving instructions such as `mulx` or versatile multiply-and-add (MULADD) instructions (see §5).

[4] We use a loose definition here: a prime should be well above 500 bits long to be considered "large", but this varies with the computer wordsize (the smaller the wordsize the lower the threshold to consider that a prime is large).

3.2 Cost analysis

Except for the variants that could use Karatsuba at the lower levels of their computations (which would only be the case for relatively large primes), the complexity of the proposed algorithms is quadratic in terms of multiplication instructions. For t products, it is easy to see that they require $n^2(t+1)+n$ digit multiplications, which is precisely the complexity of standard lazy reduction when the products are done in schoolbook or Comba-style. This means that lazy reduction in conjunction with a subquadratic multiplication like Karatsuba-schoolbook or Karatsuba-Comba (KCM) [43] is theoretically cheaper in terms of multiplications.

Nevertheless, the new method can achieve a superior performance in practice for small and medium-size primes since it enables streamlined implementations with much less friction with memory. Moreover, the schoolbook variants can reach exceptional speed by exploiting their synergy with carry-preserving instructions.

To see this, let's run a comparative analysis with one of the most promising variants for software platforms, namely, a merged sum of products using radix-r interleaved Montgomery multiplication in coarsely integrated form (Algorithm 2). We assume $B = 1$ in the case of an $\mathbb{F}_{p^2}$ multiplication, and set an x64 processor as the target. We compare against state-of-the-art implementations of multiplication over $\mathbb{F}_{p^2}$, which essentially use variants of Algorithm 1 [2, 37].

First, if we perform a theoretical analysis in line with, e.g., [53], which counts the number of instructions executing a multiplication (mul), addition/subtraction (add), memory load (read) and memory store (write), the cost of one $\mathbb{F}_{p^2}$ multiplication using Algorithm 1 is approximately given by

$$\begin{aligned}
\text{cost} &= \text{cost}_{\text{line 1}} + \text{cost}_{\text{line 2}} + \ldots + \text{cost}_{\text{line 10}} \\
&= 2 \times \left(3n^2/4 \text{ muls} + (3n^2 + 4n + 2) \text{ adds} + (3n^2/2 + 15n/2 + 1) \text{ reads} + (3n^2/4 + 11n/2 + 1) \text{ writes}\right) + 2 \times \left(n \text{ adds} + 2n \text{ reads} + n \text{ writes}\right) + \left(3n^2/4 \text{ muls} + (3n^2 + 4n + 2) \text{ adds} + (3n^2/2 + 15n/2 + 1) \text{ reads} + (3n^2/4 + 11n/2 + 1) \text{ writes}\right) \\
&\quad + 2 \times \left(2n \text{ adds} + 4n \text{ reads} + 2n \text{ writes}\right) + \left(3n \text{ adds} + 2n \text{ reads} + 5n \text{ writes}\right) \\
&\quad + 2 \times \left((n^2 + n) \text{ muls} + 4n^2 \text{ adds} + (2n^2 + 2n) \text{ reads} + n^2 \text{ writes}\right) \\
&= (17n^2/4 + 2n) \text{ muls} + (17n^2 + 21n + 6) \text{ adds} + (17n^2/2 + 83n/3 + 3) \text{ reads} + (17n^2/4 + 49n/2 + 3) \text{ writes}, \qquad (4)
\end{aligned}$$

where the multiplications (lines 1, 2 and 5, Alg. 1) are assumed to be computed using Karatsuba at the top level and schoolbook underneath, and the reductions (lines 9 and 10, Alg. 1) are assumed to be computed with radix-r interleaved Montgomery multiplication in operand scanning form (schoolbook).

Now, if we do a similar cost analysis for Algorithm 2, the cost for the sum of t products is given by

$$\begin{aligned}
\text{cost} = (tn^2 + n^2 + n) \text{ muls} + (4tn^2 + 4n^2 + 2tn) \text{ adds} + (2tn^2 + 2n^2 + 2tn + 2n) \text{ reads} \\
+ (tn^2 + n^2 + 2tn) \text{ writes}.
\end{aligned}$$

And thus, the cost for a full $\mathbb{F}_{p^2}$ multiplication, consisting of two sums of products with $t = 2$, is given by

$$\text{cost} = (6n^2 + 2n)\ \text{muls} + (24n^2 + 8n)\ \text{adds} + (12n^2 + 12n)\ \text{reads} + (6n^2 + 8n)\ \text{writes}. \quad (5)$$

If we compare costs (4) and (5), standard lazy reduction appears to beat the new method solidly for almost every operation type. However, this analysis ignores key practical considerations, as we discuss below.

Let's now perform a more practical analysis based on an actual implementation of the $\mathbb{F}_{p^2}$ multiplication using a 384-bit prime (primes of this size are relevant in pairing and isogeny-based applications; see §4 and App. B). For the implementations, we consider the use of carry-preserving instructions like `mulx` and `adx`, which are supported by all modern Intel and AMD processors.

In this case, the cost when using the standard lazy reduction technique (Alg. 1) reduces to

$$\text{cost} = (17n^2/4 + 2n)\ \text{muls} + (17n^2/2 + 55n/2 - 9)\ \text{adds} + (17n^2/4 + 40n)\ \text{reads} + (2n^2 + 47n/2)\ \text{writes}. \quad (6)$$

And the cost of the new method using Algorithm 2 reduces to

$$\text{cost} = (6n^2 + 2n)\ \text{muls} + (12n^2 + 6n)\ \text{adds} + (6n^2 + 6n)\ \text{reads} + 2n\ \text{writes}. \quad (7)$$

As can be seen, in practice the memory access costs are greatly reduced thanks to the use of the general purpose registers (GPRs). Likewise, the use of carry-preserving instructions reduces the number of addition instructions significantly. While this happens across both algorithms, the improvement is much more pronounced for the new method, especially in the case of memory writes. This highlights the streamlined nature of the proposed approach, which permits to eliminate many memory accesses.

The above can be clearly observed in Figure 1, which displays the total number of instructions (multiplications, additions/subtractions and memory reads and writes) for different prime sizes, using the theoretical analysis and the analysis based on practical implementations. For the practical cases, the results for the different bitlengths were extrapolated from the 384-bit prime implementation. We remark that this analysis is obviously imperfect but still reveals meaningful information about the performance of the different approaches.

An important observation not visible in Figure 1 due to the extrapolation is that, in practice, the number of GPRs is limited and starts to force more memory accesses as the prime bitlength passes certain threshold. This reduces the relative speedup of the new method for large primes, as the multiplication elimination via Karatsuba becomes increasingly attractive. As can be seen in the analysis on the SIKE primes (see App. B), the speed superiority of our method over standard lazy reduction starts to vanish around the 610-bit mark on an x64 platform (note, however, that the memory savings remain stable and even increase slightly with the prime bitlength).

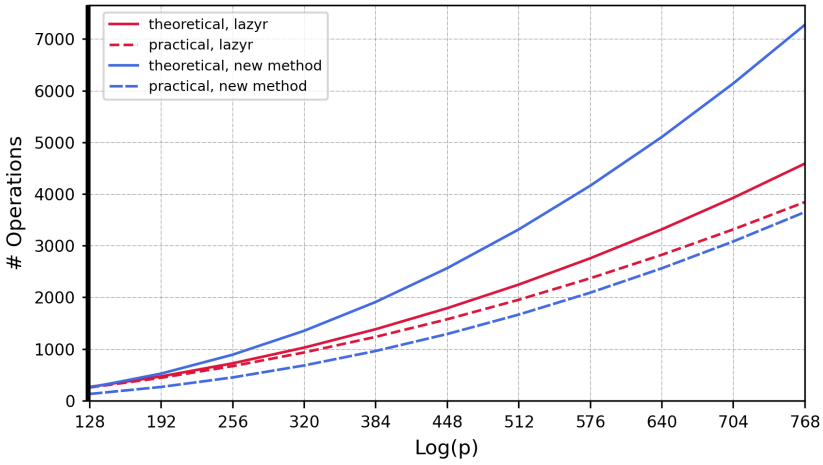

Fig. 1: Comparison of instruction counts between the proposed method (Algorithm 2, $B = 1$) and the standard lazy reduction method (`lazyr`, Algorithm 1) for computing a full multiplication over $\mathbb{F}_{p^2}$. The counts cover all the instructions executing multiplications, additions, subtractions and memory accesses. The theoretical counts closely follow similar counts from [43, 53], while the practical counts are based on actual implementations over a 384-bit prime exploiting carry-preserving instructions (results for other bitlengths are obtained by extrapolation).

4 Case Study: optimal ate pairing over BLS12-381

Since the seminal papers by Sakai et al. [63] on identity-based non-interactive authentication key agreement and by Joux [48] on tripartite one-round key agreement, bilinear pairings have become a powerful tool in the design of a myriad of novel cryptographic schemes such as identity-based cryptosystems [19, 24, 28, 29] and non-interactive zero-knowledge proof systems [44, 45].

One critical drawback of pairings is their relatively expensive running time. This motivated an extraordinary effort by the research community to improve efficiency on several fronts, including the construction of pairing-friendly elliptic curves [12, 25, 41, 66], the development and improvement of the algorithms for the Miller loop and final exponentiation [11, 13, 14, 46, 68], the optimization of the explicit formulas for the curve arithmetic [3, 33, 34], and the design and optimization of towering schemes of extension fields $\mathbb{F}_{p^k}$ [3, 17, 52]. Readers are referred to [1] for a modern take on the design and implementation of pairings.

Here, we focus on the optimization of the arithmetic over $\mathbb{F}_{p^k}$ using the proposed algorithms. To illustrate the performance gains, we target a modern and popular pairing-friendly curve, namely BLS12-381 [23], using an optimal ate pairing instantiation [68]. BLS12-381, proposed by Bowe [23], is an elliptic curve from the Barreto-Lynn-Scott (BLS) family [12] that targets the 128-bit security level and is undergoing a standardization effort in the IETF CFRG [64].

Most notoriously, this curve is widely used in zero-knowledge proofs and digital signatures in blockchain applications like Zcash [6] and Ethereum 2.0 [26][5].

BLS12-381 is defined by the curve $E(\mathbb{F}_p) : y^2 = x^3 + 4$, with embedding degree $k = 12$. Relevant to our analysis is that, in practice, the arithmetic implementation over $\mathbb{F}_{p^k}$ is realized via the following towering scheme:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, where $\beta = -1$.
- $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[s]/(s^2 - \xi)$, where $\xi = 1 + i$.
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$.
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[t]/(t^3 - s)$ or $\mathbb{F}_{p^6}[w]/(w^2 - v)$ or $\mathbb{F}_{p^2}[w]/(\omega^6 - \xi)$.

Given that for pairings, generic Montgomery multiplication (i.e., variants that do not exploit any special form in the prime) is known to provide the best performance in software, straight implementations of the variants discussed in §3 are relevant in this case. More specifically, variants that exploit the synergy between schoolbook algorithms and carry-preserving instructions are expected to outperform other approaches. Thus, we observed that the implementation of multiplication over $\mathbb{F}_{p^2}$ can be efficiently carried out using the interleaved radix-r Montgomery multiplication variant in coarsely integrated form (Algorithm 2), with $B = 1$ to make full use of schoolbook.

Without loss of generalization, we discuss next the implementation options for the upper layers in the targeted tower field.

The case of multiplication over $\mathbb{F}_{p^6}$. There are multiple choices to implement multiplication over $\mathbb{F}_{p^6}$. For example, it can be implemented on top of the $\mathbb{F}_{p^2}$ arithmetic layer using our method for the multiplication over $\mathbb{F}_{p^2}$ and Karatsuba at the $\mathbb{F}_{p^6}$ level with or without lazy reduction. Or it could be implemented using the proposed method by seeing $\mathbb{F}_{p^6}$ as a direct extension of $\mathbb{F}_{p^2}$ and expressing the operations down to the base field, as discussed next.

Let $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$ as in the towering scheme above. And let $a = a_0 + a_1 v + a_2 v^2$ be an element in $\mathbb{F}_{p^6}$, where $a_i = (a_{i,0}, a_{i,1}) \in \mathbb{F}_{p^2}$ for $i = \{0, 1, 2\}$. Then, the multiplication $c = (c_0, c_1, c_2) = a \cdot b$ in $\mathbb{F}_{p^6}$ is given by

$$
\begin{aligned}
c_{0,0} &= a_{0,0}b_{0,0} - a_{0,1}b_{0,1} + a_{1,0}b_{2,0} - a_{1,1}b_{2,1} + a_{2,0}b_{1,0} - a_{2,1}b_{1,1} - a_{1,0}b_{2,1} - a_{1,1}b_{2,0} - a_{2,0}b_{1,1} \\
&\quad - a_{2,1}b_{1,0}. \\
&= a_{0,0}b_{0,0} - a_{0,1}b_{0,1} + a_{1,0}(b_{2,0} - b_{2,1}) - a_{1,1}(b_{2,0} + b_{2,1}) + a_{2,0}(b_{1,0} - b_{1,1}) - a_{2,1}(b_{1,0} + b_{1,1}). \\
c_{0,1} &= a_{0,0}b_{0,1} + a_{0,1}b_{0,0} + a_{1,0}b_{2,1} + a_{1,1}b_{2,0} + a_{2,0}b_{1,1} + a_{2,1}b_{1,0} + a_{1,0}b_{2,0} - a_{1,1}b_{2,1} + a_{2,0}b_{1,0} \\
&\quad - a_{2,1}b_{1,1}. \\
&= a_{0,0}b_{0,1} + a_{0,1}b_{0,0} + a_{1,0}(b_{2,0} + b_{2,1}) + a_{1,1}(b_{2,0} - b_{2,1}) + a_{2,0}(b_{1,0} + b_{1,1}) + a_{2,1}(b_{1,0} - b_{1,1}). \\
c_{1,0} &= a_{0,0}b_{1,0} - a_{0,1}b_{1,1} + a_{1,0}b_{0,0} - a_{1,1}b_{0,1} + a_{2,0}b_{2,0} - a_{2,1}b_{2,1} - a_{2,0}b_{2,1} - a_{2,1}b_{2,0}. \\
&= a_{0,0}b_{1,0} - a_{0,1}b_{1,1} + a_{1,0}b_{0,0} - a_{1,1}b_{0,1} + a_{2,0}(b_{2,0} - b_{2,1}) - a_{2,1}(b_{2,0} + b_{2,1}). \\
c_{1,1} &= a_{0,0}b_{1,1} + a_{0,1}b_{1,0} + a_{1,0}b_{0,1} + a_{1,1}b_{0,0} + a_{2,0}b_{2,1} + a_{2,1}b_{2,0} + a_{2,0}b_{2,0} - a_{2,1}b_{2,1}. \\
&= a_{0,0}b_{1,1} + a_{0,1}b_{1,0} + a_{1,0}b_{0,1} + a_{1,1}b_{0,0} + a_{2,0}(b_{2,0} + b_{2,1}) + a_{2,1}(b_{2,0} - b_{2,1}). \\
c_{2,0} &= a_{0,0}b_{2,0} - a_{0,1}b_{2,1} + a_{1,0}b_{1,0} - a_{1,1}b_{1,1} + a_{2,0}b_{0,0} - a_{2,1}b_{0,1}. \\
c_{2,1} &= a_{0,0}b_{2,1} + a_{0,1}b_{2,0} + a_{1,0}b_{1,1} + a_{1,1}b_{1,0} + a_{2,0}b_{0,1} + a_{2,1}b_{0,0}.
\end{aligned}
\tag{8}
$$

[5] In fact, the main motivation for the design of BLS12-381 was its use for Zcash's zk-SNARK proofs.

After regrouping common coefficients and assuming that the four values $(b_{1,0} + b_{1,1}), (b_{1,0} - b_{1,1}), (b_{2,0} + b_{2,1})$ and $(b_{2,0} - b_{2,1})$ are pre-calculated, one can apply either Algorithm 2 or Algorithm 3 with a cost of $6 \times 6 = 36$ multiplications in the base field.

Note that, in contrast to a generic sum of products, the polynomial multiplication modulo f offers opportunities to eliminate some multiplications using Karatsuba. For example, in the term $c_{0,1}$ one could compute $a_{0,0}b_{0,1} + a_{0,1}b_{0,0}$ as $(a_{0,0} + a_{0,1})(b_{0,0} + b_{0,1}) - a_{0,0}b_{0,0} - a_{0,1}b_{0,1}$ with only *one* base field multiplication, using intermediate values from $c_{0,0}$. However, these replacements should be applied with care, since they break the algorithm flow (recall that inner multiplications are interleaved with reduction computations) and increase memory usage, potentially neglecting any savings obtained by eliminating multiplications. Ultimately, the benefit of combining Karatsuba with the proposed algorithms might depend on the target platform (see Appendix A for details on another Karatsuba variant).

The case of multiplication over $\mathbb{F}_{p^{12}}$. Similarly in this case, we can leverage the multiplications over $\mathbb{F}_{p^2}$ or over $\mathbb{F}_{p^6}$ discussed above, in combination with Karatsuba with or without lazy reduction at the $\mathbb{F}_{p^{12}}$ layer. But we can also do the computation by writing the full polynomial multiplication down to the base field level. Recall that $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[w]/(\omega^6 - \xi)$, where $\xi = 1 + i$. It is straightforward to determine that, in this case, we need to compute *twelve* terms each consisting of a sum of *twelve* products (assuming the pre-calculation of ten values, similarly to multiplication over $\mathbb{F}_{p^6}$). Similar comments apply to the possibility of exploiting Karatsuba to products in adjacent terms.

Performance results. To evaluate the proposed algorithms, we have integrated our implementations to the RELIC cryptographic library, version 0.5.0 [2]. This library contains, to our knowledge, some of the most efficient open-source implementations of pairings. In particular, it applies the generalized lazy reduction to the full extension field and elliptic curve arithmetic, as proposed in [3].

In our experiments, we use a 3.4GHz Intel Core i7-6700 (Skylake) processor with TurboBoost disabled to follow standard practice. Compilation was carried out using clang v6.0.1 with the command `clang -O3`. Memory stack usage was obtained using `valgrind`[6] and `massif-cherrypick`[7].

Table 1 compares RELIC's implementation of the extension field multiplications for BLS12-381 against the various options that we discussed for our algorithms. We use the following notation to specify a given strategy: first, we indicate up to which layer an algorithm is applied, followed by the approach taken for the upper layers (if any). For the latter, we have two options: for the upper layers, one can use either straight Karatsuba (called "Karat") or Karatsuba with lazy reduction (called "Karat + lazyr"). For example, if the table indicates "Alg. 2 over $\mathbb{F}_{p^6}$. Karat + lazyr over $\mathbb{F}_{p^{12}}$", it means that we use Algorithm 2

[6] https://valgrind.org/
[7] https://github.com/lnishan/massif-cherrypick

to implement multiplication up to the $\mathbb{F}_{p^6}$ layer, with the upper layer over $\mathbb{F}_{p^{12}}$ implemented with a formula that exploits Karatsuba with lazy reduction. In all the cases, we set $B = 1$ for Algorithm 2, which gives optimal performance on the targeted x64 platform. As noted before, this schoolbook-like algorithmic variant implementing an interleaved modular multiplication in coarsely integrated form fully exploits the availability of carry-preserving instructions. We comment that, at least on the targeted processor, the algorithm should achieve similar performance for small values of B, as long as an increase in the radix size does not put additional pressure on the register usage. For example, in our experiments, we obtained similar results for $B = 1$ and $B = 2$.

In terms of speed, Table 1 reveals that the full use of the new method solidly beats the state-of-the-art implementations up to the $\mathbb{F}_{p^6}$ layer. For the $\mathbb{F}_{p^{12}}$ multiplication, the fastest mark is achieved by using the implementation over $\mathbb{F}_{p^6}$ and implementing the upper layer over $\mathbb{F}_{p^{12}}$ using Karatsuba. This is due to the fact that at certain threshold Karatsuba starts to outperform schoolbook algorithms when multiplications get eliminated at a sufficiently faster rate. Interestingly enough, we do not require the use of lazy reduction because a basic implementation based on Karatsuba already achieves optimal performance. This is the consequence of minimizing the cost of reduction through the proposed approach, and this greatly reduces the complexity of the implementation. Note that we also obtain a significant gain in the computation of squaring over $\mathbb{F}_{p^2}$. This is the result of replacing the non-interleaved Montgomery multiplication available in [2] by a faster interleaved version, given that we were not limited anymore to the old algorithmic selection that exploited lazy reduction.

Most notably, the proposed method reduces significantly the use of memory, achieving savings in the range 43%-78% for different extension field operations and with increasing savings for higher extension degrees. The remarkable difference is mainly due to the elimination of double-precision operations and the streamlined nature of our algorithms. Looking at the different options for $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ multiplication, one can see that those that avoid lazy reduction and implement the full arithmetic using Algorithm 2 minimize the use of memory. For example, although slightly slower, the use of Algorithm 2 over $\mathbb{F}_{p^{12}}$ represents the most memory-friendly option for multiplication over $\mathbb{F}_{p^{12}}$.

Finally, Table 1 also reports the cycle counts for the full pairing computation using an optimal ate instantiation: on a 3.4GHz Intel Core i7-6700 Skylake machine the computation of our fastest implementation option is performed in ~ 674 μsec. As an additional data point, the same computation on a 3.2GHz Intel Core i7-8700 Coffee Lake machine is carried out in ~ 491 μsec.[8] (compare to the 688 μsec. obtained by running the implementation from RELIC on the same platform). We remark that our implementation is much simpler and more memory-friendly, and still achieves a speedup of up to 1.40× over the state-of-

[8] Some fun trivia: the reported BLS12-384 implementation runs a 128-bit secure pairing in under half a millisecond, which is just slightly faster than the speed record mark hit by the BN254 pairing almost 11 years ago [3], before new attacks emerged and pushed field sizes up.

Table 1: Comparison of the speed performance (in terms of clock cycles) and memory stack usage (in terms of bytes) between the state-of-the-art implementation of an optimal ate pairing over BLS12-381 and its extension field arithmetic [2] and the optimized implementation using the method proposed in this work. The target platform is a 3.4GHz Intel Core i7-6700 (Skylake) processor.

Reference	**Strategy**	**Speed**		**Memory**	
		cc	%	bytes	%
	$\mathbb{F}_{p^2}$ mul				
RELIC [2]	Separated mul/rdc. Karat + lazyr	566	-	1,920	-
This work	Alg. 2 over $\mathbb{F}_{p^2}$	**357**	-37%	**1,104**	-43%
	$\mathbb{F}_{p^2}$ sqr				
RELIC [2]	Separated mul/rdc	451	-	1,824	-
This work	Interleaved mul/rdc	**273**	-40%	**976**	-46%
	$\mathbb{F}_{p^6}$ mul				
RELIC [2]	Separated mul/rdc. Karat + lazyr	3,376	-	6,320	-
This work	Alg. 2 over $\mathbb{F}_{p^2}$. Karat over $\mathbb{F}_{p^6}$	2,695	-20%	2,416	-62%
This work	Alg. 2 over $\mathbb{F}_{p^2}$. Karat + lazyr over $\mathbb{F}_{p^6}$	2,961	-12%	6,320	-0%
This work	Alg. 2 over $\mathbb{F}_{p^6}$	**2,344**	-31%	**2,104**	-67%
	$\mathbb{F}_{p^{12}}$ mul				
RELIC [2]	Separated mul/rdc. Karat + lazyr	10,061	-	16,040	-
This work	Alg. 2 over $\mathbb{F}_{p^2}$. Karat over $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$	7,845	-22%	4,240	-74%
This work	Alg. 2 over $\mathbb{F}_{p^2}$. Karat + lazyr over $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$	8,800	-13%	16,040	-0%
This work	Alg. 2 over $\mathbb{F}_{p^6}$. Karat over $\mathbb{F}_{p^{12}}$	**7,841**	-22%	3,928	-76%
This work	Alg. 2 over $\mathbb{F}_{p^6}$. Karat + lazyr over $\mathbb{F}_{p^{12}}$	8,114	-19%	13,784	-14%
This work	Alg. 2 over $\mathbb{F}_{p^{12}}$	8,315	-17%	**3,544**	-78%
	Pairing				
RELIC [2]	Separated mul/rdc. Karat + lazyr	3.15×10^6	-	23,198	-
This work	Alg. 2 over $\mathbb{F}_{p^6}$. Karat over $\mathbb{F}_{p^{12}}$	$\mathbf{2.29 \times 10^6}$	-27%	12,752	-45%
This work	Alg. 2 over $\mathbb{F}_{p^{12}}$	2.30×10^6	-27%	**11,792**	-49%

the-art on an x64 processor. In addition, the table also includes another implementation option in which the full $\mathbb{F}_{p^{12}}$ multiplication uses Algorithm 2. This implementation saves up to 49% in memory usage in the pairing computation while almost achieving top speed performance.

5 Impact to Other Scenarios and Future Work

The simple but effective approach that we have proposed in this work changes the paradigm which the implementation of extension field arithmetic has long relied upon. This immediately impacts the software implementation of cryptographic schemes such as those based on bilinear pairings and supersingular isogenies. Moreover, we also expect the approach to influence the development of efficient techniques and implementations for other software platforms, constrained devices and hardware architectures, not only because of the potential speed gains

but also (and maybe more critically) because of the significant savings in memory usage. We discuss below a few possibilities for some representative platforms.

Software platforms. In platforms with a limited number of registers there is a risk of high memory access costs. Hence, a streamlined, schoolbook-like algorithm like Algorithm 2 that minimizes memory friction and reduces the use of certain operations such as additions (e.g., when there is support for carry-preserving instructions) achieves high performance on modern x64 platforms. Alternative methods based on Karatsuba are expected to become attractive at relatively large prime sizes, when the reduction in multiplications compensates for the bumpier algorithmic flow with higher number of memory accesses and additions.

We expect a similar (if not better) situation with vectorized implementations using the recent AVX-512 vector instructions available in some Intel processors. For example, the optional extension "Integer Fused Multiply and Add" (IFMA) includes MULADD instructions that perform up to eight 52-bit multiplications followed by accumulations with 64-bit values [32]. Future work could involve studying the performance of the proposed method using the operand and product-scanning forms, in combination with different vectorization strategies.

Similar comments apply to implementations using the ARM NEON vector engine [55]. In this case, there is access to powerful, high-throughput MULADD instructions that perform up to two 32-bit multiplications followed by accumulations with 64-bit values. Thus, these instructions would favor an algorithmic variant of Eq. (3) in product-scanning form. In the case of multiplication over $\mathbb{F}_{p^2}$, for example, the 2-way NEON execution naturally adapts to perform the two-term computation (i.e., the operations $c_0 = a_0b_0 + a_1b_1\beta$ and $c_1 = a_0b_1 + a_1b_0$) in parallel.

For the case of scalar implementations on 64-bit ARMv8 processors, the relatively high cost of multiplication instructions might make the case for standard Karatsuba with lazy reduction. However, memory accesses are also expensive, which would favor a more streamlined execution as in the proposed algorithms. This requires actual experimentation to determine which algorithm would be optimal.

Constrained platforms. For these devices, the potential reduction in memory use is particularly relevant. A popular platform in this computing category is ARM Cortex-M4. For this case, one can exploit the powerful, *one-cycle* MULADD instructions available in the DSP extension. These instructions can perform a 32-bit multiplication plus 64-bit accumulation, or a 32-bit multiplication plus two 32-bit accumulations. The low cost of multiplication, added to the potential of eliminating the overhead from addition instructions in a schoolbook-like computation (e.g., see [56, §3.4]), makes our approach using Algorithm 2 a perfect fit.

Hardware platforms. On hardware platforms like FPGAs, there are two important metrics to consider when choosing and designing an algorithm: flexibility

to parallelize independent tasks, and the area-time (AT) cost. With this taken into account, our finely-integrated variant (i.e., Algorithm 3) provides an *optimal* level of parallelization at the algorithmic level without adding excessive area overhead. Note that Karatsuba and lazy reduction are not ideal in many cases because they introduce overheads and storage requirements that might hurt one or both of the hardware metrics mentioned above. Karatsuba permits to subdivide computations in multiple, smaller multiplies that can be done in parallel, but the extra circuitry can neglect a good AT trade-off on pipelined architectures. In contrast, using hardware adaptations of Algorithm 3 would naturally enable a parallel computation of up to $(t+1)$ products (note that all the products in lines 7 and 8 in the inner for-loop are independent from each other).

For applications that can deal with the extra overhead and look for a reduction in the number of multiplications, we discuss two Karatsuba variants with subquadratic complexity in Appendix A.

Other applications. At the core, the proposed method gains in efficiency by eliminating modular reductions. Therefore, it is natural to conclude that primes that support a very fast reduction (e.g., Mersenne or pseudo-Mersenne primes) would not gain a significant advantage.

Finally, we comment that other reduction algorithms in the literature could exploit the method advantageously. For example, this is the case of Barrett reduction [16] and its interleaved version [51, §2.1], which can be extended to support merged sum of products with a unified modular reduction, as done in this work.

References

1. D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini. The realm of the pairings. In *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2013.
2. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
3. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2011.
4. R. M. Avanzi. Aspects of hyperelliptic curves over large prime fields in software implementations. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2004.
5. R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Hutchinson, A. Jalali, K. Karabina, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik. Supersingular Isogeny Key Encapsulation (SIKE), 2017–2022. Specification available at https://sike.org.
6. E. B.-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy – SP 2014*, pages 459–474. IEEE Computer Society, 2014.

7. J.-C. Bajard and S. Duquesne. Montgomery-friendly primes and applications to cryptography. *J. Cryptogr. Eng.*, 11(4):399–415, 2021.
8. S. Baktir and B. Sunar. Optimal tower fields. *IEEE Trans. Computers*, 53(10):1231–1243, 2004.
9. R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *J. Cryptol.*, 32(4):1298–1336, 2019.
10. R. Barbulescu, P. Gaudry, and T. Kleinjung. The tower number field sieve. In *Advances in Cryptology – ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 31–55. Springer, 2015.
11. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.
12. P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks – SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.
13. P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 2003.
14. P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptol.*, 17(4):321–334, 2004.
15. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC 2006*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
16. P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1986.
17. N. Benger and M. Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In *International Workshop on the Arithmetic of Finite Fields – WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2010.
18. J.-L. Beuchat, J. E. González-Díaz, S. M., E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.
19. D. Boneh and M.K. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
20. J. W. Bos and S. Friedberger. Fast arithmetic modulo 2^{x} $p^{y} \pm 1$. In *IEEE Symposium on Computer Arithmetic – ARITH 2017*, pages 148–155. IEEE Computer Society, 2017.
21. J. W. Bos and S. Friedberger. Faster modular arithmetic for isogeny-based crypto on embedded devices. *J. Cryptogr. Eng.*, 10(2):97–109, 2020.
22. J. W. Bos and P. L. Montgomery. Montgomery arithmetic from a software perspective. *Chapter 2 of Topics in Computational Number Theory Inspired by Peter L. Montgomery*, pages 10–39, 2017.
23. S. Bowe. BLS12-381: New zk-SNARK elliptic curve construction, 2017. https://electriccoin.co/blog/new-snark-curve/.

24. X. Boyen and B. Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2006.
25. F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. *Des. Codes Cryptogr.*, 37(1):133–141, 2005.
26. J. Buck. Ethereum upgrade Byzantium is live, verifies first ZK-Snark proof, 2017. https://cointelegraph.com/news/ethereum-upgrade-byzantium-is-live-verifies-first-zk-snark-proof.
27. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. Cryptology ePrint Archive, Report 2022/975, 2022.
28. J. C. Cha and J. H. Cheon. An identity-based signature from gap Diffie-Hellman groups. In *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2003.
29. L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241, 2007.
30. R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. Xiaoxu Yao. FPGA implementation of pairings using residue number system and lazy reduction. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 421–441. Springer, 2011.
31. P. G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, 1990.
32. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, 2021.
33. C. Costello, H. Hisil, C. Boyd, J. M. González Nieto, and K. Koon-Ho Wong. Faster pairings on special Weierstrass curves. In *Pairing-Based Cryptography – Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2009.
34. C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. In *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 224–242. Springer, 2010.
35. C. Costello and P. Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In *Advances in Cryptology – ASIACRYPT 2015*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.
36. C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *LNCS*, pages 572–601. Springer, 2016.
37. C. Costello, P. Longa, and M. Naehrig. SIDH Library. https://github.com/Microsoft/PQCrypto-SIDH, 2016–2022.
38. S. R. Dussé and B. S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 1991.
39. Armando Faz-Hernández, Julio López Hernandez, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Trans. Computers*, 67(11):1622–1636, 2018.
40. L. De Feo, D. Kohel, A. Leroux, C. Petit, and B. Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In *Advances in Cryptology – ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.

41. D. Freeman. Constructing pairing-friendly elliptic curves with embedding degree 10. In *Algorithmic Number Theory – ANTS-VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 452–465. Springer, 2006.
42. C. C. F. Pereira Geovandro, M. A. Simplício Jr., M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *J. Syst. Softw.*, 84(8):1319–1326, 2011.
43. J. Großschädl, R. M. Avanzi, E. Savas, and S. Tillich. Energy-efficient software implementation of long integer modular arithmetic. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2005.
44. J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.
45. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.
46. F. Hess, N. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.
47. D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography – PQCrypto 2011*, volume 7071 of *LNCS*. Springer, 2011.
48. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Algorithm Number Theory Symposium – ANTS IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.
49. A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Doklady Akad. Nauk SSSR*, (145):293–294, 1962. Translation in Physics-Doklady 7, 595-596, 1963.
50. T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *Lecture Notes in Computer Science*, pages 543–571. Springer, 2016.
51. M. Knezevic, F. Vercauteren, and I. Verbauwhede. Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods. *IEEE Trans. Computers*, 59(12):1715–1721, 2010.
52. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In *International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.
53. Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *Micro, IEEE*, 16(3):26–33, 1996.
54. C. H. Lim and H. S. Hwang. Fast implementation of elliptic curve arithmetic in GF(p^n). In *Workshop on Practice and Theory in Public Key Cryptography – PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 405–421. Springer, 2000.
55. ARM Limited. NEON programmer's guide, v1.0. https://developer.arm.com/documentation/den0018/a/?lang=en, 2013.
56. Z. Liu, P. Longa, G. C. C. F. Pereira, O. Reparaz, and H. Seo. Four$\mathbb{Q}$ on embedded devices with strong countermeasures against side-channel attacks. *IEEE Trans. Dependable Secur. Comput.*, 17(3):536–549, 2020.
57. Z. Liu, H. Seo, A. Castiglione, K.-K. R. Choo, and H. Kim. Memory-efficient implementation of elliptic curve cryptography for the Internet-of-Things. *IEEE Trans. Dependable Secur. Comput.*, 16(3):521–529, 2019.

58. P. Longa. *High-speed elliptic curve and pairing-based cryptography*. PhD thesis, University of Waterloo, 2011.
59. P. Longa, W. Wang, and J. Szefer. The cost to break SIKE: A comparative hardware-based analysis with AES and SHA-3. In *Advances in Cryptology – CRYPTO 2021*, volume 12827 of *Lecture Notes in Computer Science*, pages 402–431. Springer, 2021.
60. A. Menezes, P. Sarkar, and S. Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *Paradigms in Cryptology – Mycrypt 2016*, volume 10311 of *Lecture Notes in Computer Science*, pages 83–108. Springer, 2016.
61. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):pp. 519–521, 1985.
62. N. El Mrabet and M. Joye. Guide to pairing-based cryptography. Chapman & Hall/CRC Cryptography and Network Security Series (CRC Press, 2017).
63. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security – SCIS 2000, Japan*, 2000.
64. Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby. Pairing-Friendly Curves, 2021. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-pairing-friendly-curves-10.
65. M. Scott. Implementing cryptographic pairings. In *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 177–196. Springer, 2007.
66. M. Scott and P. S. L. M. Barreto. Generating more MNT elliptic curves. *Des. Codes Cryptogr.*, 38(2):209–217, 2006.
67. T. Unterluggauer and E. Wenger. Efficient pairings and ECC for embedded systems. In *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2014.
68. F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.
69. D. Weber and T. F. Denny. The solution of McCurley's discrete log challenge. In *Advances in Cryptology – (CRYPTO '98)*, volume 1462 of *LNCS*, pages 458–471. Springer, 1998.

A Other algorithmic variants

In this section, we discuss two variants using Karatsuba that could be amenable for certain applications looking to reduce the number of multiplies.

The first one was already mentioned in §4 and applies to polynomial multiplication in general: for certain sums of two products, it is possible to use products from adjacent terms to convert them to a Karatsuba multiplication and save a multiplication (see the subsection "The case of multiplication over $\mathbb{F}_{p^6}$"). Since the proposed algorithms interleave integer multiplications and reduction products, some care has to be taken into account to avoid excessive use of storage.

Below, we propose another option that interleaves Karatsuba multiplication with the radix-r Montgomery reduction. The new algorithm is shown in Algorithm 4 using 2-way Karatsuba.

Let's focus on the simple case with $t = 1$, i.e., a standard modular multiplication $a \cdot b \bmod p$ (the sum-of-products case easily follows). The basic idea of the algorithm is to first split operands in two halves (for a 2-way Karatsuba) using the generalized radix $r = 2^{Bw} = 2^{\lceil n/2 \rceil}$, such that operands are represented as $(a_1, a_0)_{2^{\lceil n/2 \rceil}}$. Then, from Eq. (2) and proceeding in product-scanning form, we have that:

$$\begin{aligned} u &= (a_0 b_0 + q_0 p_0)/r, \text{ with } q_0 = a_0 b_0 p' \bmod r \\ &= (u + a_1 b_0 + a_0 b_1 + q_0 p_1 + q_1 p_0)/r, \text{ with } q_1 = (u + a_1 b_0 + a_0 b_1 + q_0 p_1) p' \bmod r \\ &= u + a_1 b_1 + q_1 p_1. \end{aligned}$$

Finally, we simply replace the intermediate computation $(a_1 b_0 + a_0 b_1)$ by $(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$. For B odd we proceed as other cases in this work and adjust the operations to the right digit size (lines 7 and 8 in Alg. 4).

We comment that other variants are possible and easily follow, such as for example an interleaved, 3-way Karatsuba-Montgomery multiplication that can be derived in a similar way with a splitting of operands in three parts. It is also possible to derive a SIKE-friendly version. In this case, one can conveniently set the radix to $r = 2^{e_2}$ for a prime $p = 2^{e_2} \cdot 3^{e_3} - 1$ and eliminate the multiplications by p'.

Algorithm 4 Merged sums of products using Karatsuba in a radix-r interleaved Montgomery multiplication.

Input: integers $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ s.t. $a_i, b_i \in [0, 2p)$ for $i = 0, \ldots, (t-1)$ and $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; the radix $r = 2^{Bw}$ s.t. $B = \lceil n/2 \rceil$, and the Montgomery constant $p' = -p^{-1} \bmod r$. Integers are represented in radix r, e.g., $a_i = (a_{i,1}, a_{i,0})_r$.
Output: the Montgomery residue $c = \sum_{i=0}^{t-1} a_i b_i \cdot R^{-1} \bmod p$ s.t. $c \in [0, 2p)$.

1: $u_0 \leftarrow \sum_{i=0}^{t-1} a_{i,0} \cdot b_{i,0}$
2: $q_0 \leftarrow u_0 \cdot p' \bmod 2^{Bw}$
3: $u \leftarrow (u_0 + q_0 \cdot p_0)/2^{Bw}$
4: $u_1 \leftarrow \sum_{i=0}^{t-1} a_{i,1} \cdot b_{i,1}$
5: $u_0 \leftarrow \left(\sum_{i=0}^{t-1} (a_{i,0} + a_{i,1}) \cdot (b_{i,0} + b_{i,1})\right) - u_0 - u_1$
6: $u \leftarrow u + u_0 + q_0 \cdot p_1$
7: $q_1 \leftarrow u \cdot p' \bmod 2^{(B-n \bmod 2)w}$
8: $u \leftarrow (u + q_1 \cdot p_0)/2^{(B-n \bmod 2)w}$
9: $u \leftarrow u + 2^{(n \bmod 2)w} \cdot (u_1 + q_1 \cdot p_1)$
10: **return** $c \leftarrow u$

B Case Study: SIKE

In this section, we discuss how to adapt the proposed method to SIKE [5]. Even though this protocol has been recently proven insecure [27], our techniques can be adapted to other similar settings, such as SQISign [40]. Moreover, the presented

analysis also allows us to evaluate the improvements in speed performance and memory usage that can be achieved for different prime sizes.

A common feature of supersingular isogeny-based schemes is that computations are performed on elliptic curves defined over $\mathbb{F}_{p^2}$, which makes arithmetic over this extension field the main building block of their implementations. This, in turn, makes the schemes an attractive target for optimization using the proposed algorithms. In the remainder, we focus on SIKE's underlying field arithmetic. For complete details about the protocol, the readers are referred to [5].

SIKE, as well as SIDH, uses a special prime with the form $p = 2^{e_2} \cdot 3^{e_3} - 1$, where $2^{e_2} \approx 3^{e_3}$ for integers e_2 and e_3. This special form allows further optimizations to the radix-r Montgomery reduction, as first exploited by Costello et al. [36] in similar fashion to the techniques used for the so-called Montgomery-friendly primes (e.g., see [22, §3.2]). Later, Bos and Friedberger [20] suggested the use of a radix greater than the computer wordsize, with the generalization $r = 2^{Bw}$. Faz-Hernandez et al. [39] then showed that this idea indeed worked well on certain x64 processors for which there is support for the carry-preserving instructions `mulx` and `adx`[9].

Similarly to previous optimizations, all the algorithms presented in §3 can be adapted to exploit the special prime form in SIDH and SIKE. On one hand, we eliminate all the multiplications by p' by noting that $p' = -p^{-1} \bmod 2^{Bw} \equiv 1$ (this holds by requiring that $0 < B \leq z$, where z is the number of 0 w-bit digits of the value $p + 1$). Secondly, the multiplications with the prime p are replaced by multiplications with the much smaller value $\hat{p} = (p+1)/2^{zw}$. As shown below, intermediate results are easily corrected without extra costs.

Algorithm 5 is the analogous version of Algorithm 2 using the optimizations above. Taking Eq. (3) as starting point, the core computation in Algorithm 5 is derived as follows

$$\begin{aligned} c &= (u + (up' \bmod r) \cdot p)/r = (u + (u \bmod 2^{Bw}) \cdot p)/2^{Bw} \\ &= (2^{Bw}\lfloor u/2^{Bw} \rfloor + u \bmod 2^{Bw} \cdot (p+1))/2^{Bw} \\ &= \lfloor u/2^{Bw} \rfloor + 2^{(z-B)w} u \bmod 2^{bw} \cdot (p+1)/2^{zw} \\ &= \lfloor u/2^{Bw} \rfloor + 2^{(z-B)w} q \cdot \hat{p}, \end{aligned}$$

where u accumulates the value $\sum_{i=0}^{t-1} a_{i,j} \cdot b_i$. As in the generic algorithms from §3, we manage the case with $n \bmod B \neq 0$ by adjusting the computations for the size-reduced digit at the most significant position (lines 6–9).

Adapting other algorithms to the SIKE primes, such as Algorithm 3 and the variants in Appendix A, easily follows.

Cost analysis. As noted in §3, the proposed algorithms get a significant advantage by eliminating double-precision operations and vastly reducing the number

[9] Carry-preserving instructions enhance the performance of schoolbook-like multiplication which, in turn, achieves better performance on modern x64 processors when using $B > 1$ in a radix-r Montgomery reduction.

Algorithm 5 Merged sums of products using radix-r Montgomery reduction in coarsely integrated form for a prime with the form $p = 2^{e_2} \cdot 3^{e_3} - 1$.

Input: integers $(a_0, a_1, \ldots, a_{t-1})$ and $(b_0, b_1, \ldots, b_{t-1})$ s.t. $a_i, b_i \in [0, 2p)$ for $i = 0, \ldots, (t-1)$ and $0 \leq \sum_{i=0}^{t-1} a_i b_i < pR$, where $R = 2^{nw}$, $p = 2^{e_2} \cdot 3^{e_3} - 1$, $n = \lceil l/w \rceil$, $l = \lceil \log_2 p \rceil$, and w is the computer wordsize; $z = \lfloor e_2/w \rfloor$, $\hat{p} = (p+1)/2^{zw}$, and the radix $r = 2^{Bw}$ s.t. $B \in \mathbb{Z}$ and $0 < B \leq z$. Integers are represented in radix r, e.g., $a_i = (a_{i,\lceil n/B \rceil - 1}, \ldots, a_{i,1}, a_{i,0})_r$.

Output: the Montgomery residue $c = \sum_{i=0}^{t-1} a_i b_i \cdot R^{-1} \bmod p$ s.t. $c \in [0, 2p)$.

$u \leftarrow 0$
for $j = 0$ **to** $\lfloor n/B \rfloor - 1$ **do**
    $u \leftarrow u + \sum_{i=0}^{t-1} a_{i,j} \cdot b_i$
    $q \leftarrow u \bmod 2^{Bw}$
    $u \leftarrow \lfloor u/2^{Bw} \rfloor + 2^{(z-B)w} q \cdot \hat{p}$
if $n \bmod B \neq 0$ **then**
    $u \leftarrow u + \sum_{i=0}^{t-1} a_{i,\lceil n/B \rceil - 1} \cdot b_i$
    $q \leftarrow u \bmod 2^{(n \bmod B)w}$
    $u \leftarrow \lfloor u/2^{(n \bmod B)w} \rfloor + 2^{(z-n \bmod B)w} q \cdot \hat{p}$
return $c \leftarrow u$

of memory accesses. If we assume that Karatsuba is not employed in the lower layers of Algorithm 5 (or the other variants), their costs in terms of multiplications are given by n^2 w-bit multiplications for the integer multiplication part and $n(n - \lfloor e_2/w \rfloor)$ w-bit multiplications for the Montgomery reduction part. The cost of reduction can be brought down further for some special primes for which the shifting technique by Bos and Friedberger [20] applies. Specifically, if it holds that $e_2 \bmod w > \lceil \log p \rceil \bmod w$, one can shift the value $\hat{p}$ in order to increase the number of 0 digits (i.e., z) and, thus, trade multiplications for shifting operations in the computation of $q \cdot \hat{p}$. Afterwards, the result can be easily corrected with a shift in the opposite direction. In this case, the cost of the specialized Montgomery reduction is of $(n \cdot \lceil \log 3^{e_3}/w \rceil)$ w-bit multiplications. However, in many cases the replacement of multiplications by other (traditionally assumed) cheap instructions ends up increasing the cost of the full computation (if the cost of a shift instruction is too close to that of a multiplication, any additional overhead can negate the potential gains). We verified that, on the targeted x64 platform, the proposed method works better without the shifting optimization.

To evaluate the performance of the proposed approach, we implemented Algorithm 5 with the Round 3 SIKE parameters p434, p503 and p610, which are intended for the NIST security levels 1, 2 and 3, respectively [5]. We also evaluate the SIKE prime p377 recently proposed by Longa et al. [59], which is shown to match NIST's level 1 using alternative cost models. We compare against the performance of the state-of-the-art implementations of the same SIKE primes from the SIDH library v3.4 [37] and from [59][10]. These libraries implement the

[10] Our algorithms have been integrated to version 3.5 of the SIDH library, available at https://github.com/microsoft/PQCrypto-SIDH.

integer multiplication in two layers using Karatsuba (upper layer) and schoolbook (lower layer). The reduction part is implemented using a non-interleaved radix-r Montgomery reduction with $B > 1$, specialized to SIDH/SIKE primes as described above (see [39, Alg. 6]). As is standard, these libraries use lazy reduction for the multiplication over $\mathbb{F}_{p^2}$, following Algorithm 1.

Table 2 presents the performance comparison (in terms of clock cycles) for the $\mathbb{F}_{p^2}$ multiplication on an x64 processor, specifically, a 3.4GHz Intel Core i7-6700 (Skylake) processor. All the implementations in the comparison are written in assembly language, and were compiled and tested on the same platform using clang v6.0.1 with the command `clang -O3`. The table also includes a detailed instruction count of all the implementations, including multiplications, additions, subtractions and other instructions. The columns "read" and "write" present counts of all the corresponding instructions that require a memory access operation. We remark that the total instruction counts are provided as an additional data point only and should not be considered to follow actual performance with high-precision. Especially in the case of the targeted platform, its superscalar, deeply pipelined microarchitecture makes extremely difficult to extract performance data from a straight instruction count. Nevertheless, it can provide relevant information for a first-order comparison of the different algorithms.

Firstly, we observe that all our implementations achieve much better speed performance, even though they require a higher number of multiplication instructions. This is due to the significant reduction of other operations, especially of those requiring read/write memory accesses. Another related aspect is that at certain threshold the operand sizes become too large and the lack of enough general purpose registers forces the use of many more memory access instructions. This can be observed for the largest prime under analysis, i.e., p610, which precisely returns the lowest speedup. In the rest of the cases, the speedup goes from $\sim 1.17\times$ up to $1.31\times$, with the speedup increasing as the size of the prime decreases. This is consistent with the results from existing literature that show that Karatsuba becomes more profitable as sizes grow (see §3). Nevertheless, we demonstrate that, for the case of a quadratic extension field, schoolbook can still be much faster for primes up to around 500 bits.

As a side note, we point out that the relative speed improvement for p377 and p503 is slightly lower than expected because the implementations from [37] and [59] benefit from the shifting technique on the targeted x64 platform. As we noted before, the proposed method does not seem to benefit from this technique when the cost of a shift instruction is too close to that of a multiplication.

Table 2 also summarizes the stack memory usage corresponding to each parameter set. The figures were obtained using `valgrind` and `massif-cherrypick`, as in §4. As can be seen, our approach achieves a significant reduction in memory consumption that is well above 35%. This is obtained consistently across the different parameter sets, with even a slight increase in the savings as the prime size goes up. This is consistent with the memory analysis in §4, although for pairings the savings are even greater for higher degree extension fields when the use of the towering-based approach is minimized.

Table 2: Comparison of instruction counts, speed performance (in terms of clock cycles) and memory stack usage (in terms of bytes) between the proposed method (Algorithm 5, $B = 1$) and the state-of-the-art implementations of multiplication over $\mathbb{F}_{p^2}$ for the SIDH and SIKE protocols [5,37,59]. The target platform is a 3.4GHz Intel Core i7-6700 (Skylake) processor. The comparison covers the Round 3 SIKE primes p434, p503 and p610, and also the prime p377 proposed in [59]. The instruction counts cover all the instructions executing multiplications, additions, subtractions and memory accesses. The column "others" includes any other additional instructions such as logical and shift instructions.

Reference	**Instruction count**							**Speed**		**Memory**	
	read	write	mul	add	others	total	%	cc	%	bytes	%
p377											
[59]	360	146	117	438	190	1,251	-	352	-	1,096	-
This work	227	18	192	412	68	917	-26.7%	**269**	-23.6%	**712**	-35.0%
p434											
[37]	492	196	179	589	164	1,620	-	440	-	1,224	-
This work	292	21	252	537	74	1,176	-27.4%	**341**	-22.5%	**784**	-35.9%
p503											
[37]	568	225	208	677	236	1,914	-	514	-	1,320	-
This work	366	24	336	709	90	1,525	-20.3%	**440**	-14.4%	**832**	-37.0%
p610											
[37]	903	368	345	1,019	182	2,817	-	762	-	1,536	-
This work	715	170	500	1,090	117	2,592	-8.0%	**734**	-3.7%	**952**	-38.0%

To assess the overall performance improvement of SIKE using the proposed method, we integrated our implementations of the quadratic extension field arithmetic to the SIDH library, version 3.4 [37]. This included both multiplication and squaring over $\mathbb{F}_{p^2}$. For the latter, we replaced the non-interleaved Montgomery multiplication available in [37, 59] by a faster interleaved version, given that we were not constrained anymore by the old algorithmic selection that exploited lazy reduction. All our implementations are written in constant time, i.e., there are no secret memory accesses and no secret data branches. Therefore, they offer protection against timing and cache attacks at the software level.

The results on a 3.4GHz Intel Core i7-6700 (Skylake) processor are summarized in Table 3. Following standard practice, we disabled TurboBoost during the tests. Compilation was carried out using clang v6.0.1 with the command `clang -O3`.

As can be seen, we achieve speedups of 1.30×, 1.30×, 1.19× and 1.06× for SIKEp377, SIKEp434, SIKEp503 and SIKEp610, respectively. As expected, the largest speedups correspond to the smaller parameter sets, with a significant improvement still observed up to the $\sim$ 500-bit parameter, i.e., SIKEp503.

Table 3: Performance comparison (in terms of clock cycles) between the state-of-the-art implementation of SIKE and its arithmetic over $\mathbb{F}_{p^2}$ [5, 37] and the optimized implementation using the method proposed in this work (Algorithm 5, $B = 1$). Performance (in cycles) is reported for multiplication and squaring over $\mathbb{F}_{p^2}$ and (in million of cycles) for SIKE's key generation (Gen), encapsulation (Enc) and decapsulation (Dec). The target platform is a 3.4GHz Intel Core i7-6700 (Skylake) processor. The comparison covers the Round 3 SIKE parameters sets 1 (SIKEp434), 2 (SIKEp503) and 3 (SIKEp610), and also the alternative level 1 parameter set SIKEp377 [59].

NIST sec level	$\log p$	**SIDH library v3.4 [37]**					**This work**				
		Speed (cc)		Speed ($\times 10^6$ cc)			Speed (cc)		Speed ($\times 10^6$ cc)		
		$\mathbb{F}_{p^2}$mul	$\mathbb{F}_{p^2}$sqr	Gen	Enc	Dec	$\mathbb{F}_{p^2}$mul	$\mathbb{F}_{p^2}$sqr	Gen	Enc	Dec
1 †	377	352	258	3.9	7.3	7.2	269	187	3.0	5.6	5.6
1	434	440	322	5.9	9.7	10.3	341	232	4.6	7.4	7.9
2	503	514	382	8.2	13.5	14.4	440	300	6.9	11.4	12.1
3	610	762	570	14.9	27.3	27.4	734	484	14.0	25.7	25.8

† Alternative parameter set for NIST Level 1 based on the analysis by Longa et al. [59].