Efficient SIMD arithmetic modulo a Mersenne number

Joppe W. Bos, Thorsten Kleinjung, Arjen K. Lenstra
EPFL IC LACAL
Station 14, CH-1015 Lausanne, Switzerland
Peter L. Montgomery
Microsoft Research, One Microsoft Way,
Redmond, WA 98052, USA

***Abstract*—This paper describes carry-less arithmetic operations modulo an integer $2^M - 1$ in the thousand-bit range, targeted at single instruction multiple data platforms and applications where overall throughput is the main performance criterion. Using an implementation on a cluster of PlayStation 3 game consoles a new record was set for the elliptic curve method for integer factorization.**

***Keywords*-Mersenne number, Single Instruction Multiple Data, Cell processor, Elliptic curve method, Integer factorization**

I. Introduction

Numbers of a special form often allow faster modular arithmetic operations than generic moduli. This is exploited in a variety of applications and has led to a substantial body of literature on the subject of fast special arithmetic. Speeding up calculations using special moduli was already proposed in the mid 1960s by Merrill [40] in the setting of residue number systems (RNS) [25]. Other applications range from speeding up fast Fourier transform based multiplication [19], enhancing the performance of digital signal processing [54], [50], [23], to faster elliptic curve cryptography (ECC; [32], [41]), such as in [3].

Another application area of special moduli is in factorization attempts of so-called *Cunningham numbers*, numbers of the form $b^n \pm 1$ for $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers. This long term factorization project, originally reported in the Cunningham tables [21] and still continuing in [15], has a long and distinguished record of inspiring algorithmic developments and large-scale computational projects [34], [42], [14], [46], [37], [13]. Factorizations from [15] with $b = 2$ are used in formal correctness proofs of floating point division methods [27]. Several of these developments [36] turned out to be applicable beyond special form moduli, and are relevant for security assessment of various common public-key cryptosystems.

This paper concerns efficient arithmetic modulo a Mersenne number, an integer of the form $2^M - 1$. These numbers, and a larger family of numbers called generalized Mersenne numbers [51], [17], [1], have found many arithmetic applications ranging from number theoretic transforms [12] to cryptography. In the latter they are used to run calculations concurrently using RNS [2] or to improve the speed of finite field arithmetic in ECC based schemes [51], [55]. The great internet Mersenne prime search project [26] is based on an implementation of the Lucas-Lehmer primality test [39], [33] for Mersenne numbers in the many million bit range. Hence, efficient arithmetic modulo a Mersenne number is a widely studied subject, not just of interest in its own right but with many applications.

Our interest in arithmetic modulo a Mersenne number was triggered by a potential (special) number field sieve (NFS) project [36], for which we need a list of composites dividing $2^M - 1$ for exponents M in the range from 1000 to 1200. The Cunningham tables contain at least 20 composite Mersenne numbers (or composite factors thereof) in the desired range that have not been fully factored yet. It may be expected that some of these composites are not suitable candidates for our list because they can be factored faster using the elliptic curve method (ECM) for integer factorization [38] than by means of special NFS (SNFS). The only way to find out if ECM is indeed preferable, is by subjecting each candidate to an extensive ECM effort (which, though it may be substantial, is small compared to the effort that would be required by SNFS): only candidates that ECM failed to factor should be included on the list.

The efficiency of ECM factoring attempts relies on the efficiency of integer arithmetic modulo the number being factored. Given the need to do extensive ECM pre-testing for at least 20 composite Mersenne numbers, we developed arithmetic operations modulo a Mersenne number suitable for implementation of ECM on the platform that we intended to use for the calculations: the Cell processor as found in the Sony PlayStation 3 game console. Because each ECM effort consists of a large number of independent attempts that can be executed in *single instruction multiple data* (SIMD) mode and because each core of the Cell processor can be interpreted as a 4-way SIMD environment, our arithmetic modulo a Mersenne number is geared towards SIMD implementation. It is described in Section III after a brief description of the Cell architecture in Section II. Although our implementations were written for the Cell processor, our methods apply to any type of SIMD platform, including graphics cards. Section IV sketches ECM, our Cell processor implementation, and lists some of our ECM

results, including a new ECM record factorization.

While the new ECM factorizations removed some of the easy cases from our list of candidate Mersenne numbers, the further practical implications of ECM records are limited to their consequence for two variants of the RSA cryptosystem [47], namely *RSA multiprime* [47] and *unbalanced RSA* [48]. The former gains a speedup by a factor of r^2 or $\frac{r^2}{4}$ for the private operation in vanilla RSA or CRT-RSA, respectively, by selecting RSA moduli (of appropriate size to be out of reach of NFS) consisting of the product of $r > 2$ primes of about the same size. In unbalanced RSA, the RSA modulus has two factors as usual, but one is chosen much smaller than the other. In these variants, r and the smallest factor must be chosen in such a way that ECM has a sufficiently low probability to find the resulting relatively small prime factor(s). Our ECM findings affirm that 1024-bit RSA moduli with $r \geq 4$ should be avoided [35] and may give practitioners of these variants some guidance how small the factors may be chosen.

II. The Cell processor and its architecture

The Cell processor, the main processor of the PS3 and thus mainly targeted at the gaming market, is a powerful general purpose processor. On the first generation PS3s it can be accessed using Sony's hypervisor, a feature that has been disabled in current versions. This made the PS3 a relatively inexpensive and also flexible source of processing power, as witnessed by a variety of cryptanalytic projects: chosen prefix collisions for the cryptographic hash function MD5 [52], [53], the solution of a 112-bit prime field elliptic curve discrete logarithm problem [9], and implementation of elliptic curve group arithmetic over a degree-130 binary extension field [10].

The architecture of the Cell processor is quite different from that of regular server or desktop processors. Taking full advantage of it requires designing new software. It is worthwhile doing so, because architectures similar to the Cell's will soon be mainstream [44]. It not only helps us to take advantage of the Cell's inexpensive processing power, it also helps to prepare for future generations of processors. See Section IV-A for the rationale why the Cell processor was chosen as the platform for our ECM attempts.

The Cell has a *Power Processing Element* (PPE), a dual-threaded Power architecture-based 64-bit processor with access to a 128-bit AltiVec/VMX SIMD unit. Its main processing power, however, comes from eight *Synergistic Processing Elements* (SPEs). When running Linux, six SPEs can be used: one is disabled, and one is reserved by the hypervisor. It is conceivable that this last one becomes accessible too [28]. Each SPE runs independently from the others at 3.192GHz, using its own 256 kilobyte of fast local memory for instructions and data. It has 128 registers of 128 bits each, allowing SIMD operations on sixteen 8-bit, eight 16-bit, or four 32-bit integers. An SPE has no $32 \times 32 \rightarrow 64$-bit or $64 \times 64 \rightarrow 128$-bit integer multiplier, but has several 4-way SIMD $16 \times 16 \rightarrow 32$-bit integer multipliers including multiply-and-add instructions.

There is an odd and an even pipeline: in ideal circumstances an SPE can dispatch one odd and one even instruction per clock cycle. Most arithmetic instructions are even. Because the SPE lacks smart branch prediction, branching is best avoided (as usual in SIMD). Multiple SIMD processes may be interleaved, filling both pipelines to increase throughput, while possibly increasing per process latency. Here we took advantage of interleaving in another manner.

The Cell processor has also been made available to the supercomputing community by placing two Cell chips in a single blade server. They come with more memory than in the PS3 and on each Cell all eight SPEs are accessible. For high-performing blade servers a newer derivative of the Cell, the PowerXCell 8i, offers enhanced double-precision floating-point capabilities. Due to their significantly higher price these compute nodes come at a price performance ratio quite different from the relatively inexpensive PS3.

III. Arithmetic modulo $2^M - 1$ on the SPE

In this section we describe the SPE-arithmetic that we developed for arithmetic modulo $N = 2^M - 1$, for M in the range from 1000 to 1200 (allowing larger values as well). Assume that $M < 13 \cdot 96 - 2 = 1246$ (larger M-values can be accommodated by putting $M < u \cdot v - 2$ with $v \cdot (2^{u-1})^2 < 2^{31}$). Our approach aims to optimize overall throughput as opposed to minimize per process latency. Two variants are presented: a first approach where addition and subtraction are fast at the cost of a radix conversion before and after the multiplication, and an alternative approach where radix conversions are avoided at the cost of slower addition and subtraction. This second variant turns out to be faster for our ECM application. In applications with a different balance between the various operations the first approach could be preferable, so it is described as well. All our methods are particularly suited to SPE-implementation, but the approach may have broader applicability.

For $k \in \mathbf{Z}_{>0}$ a *k-bit integer* is an integer w with $0 \leq w < 2^k$. A *signed k-bit integer* is an integer w with $-2^{k-1} \leq w < 2^{k-1}$. For $r \in \mathbf{Z}_{>1}$ a *radix-r representation* of an integer z with $0 \leq z < r^s$ is a sequence of *radix-r digits* $(w_j)_{j=0}^{s-1}$ such that $z = \sum_{j=0}^{s-1} w_j r^j$ and $w_j \in \mathbf{Z}_{\geq 0}$. It is unique if $0 \leq w_j < r$ for $0 \leq j < s$. If $2^k \geq r$, a *signed k-bit radix-r representation* of z is a sequence $(w_j)_{j=0}^{s}$ of signed k-bit integers such that $z = \sum_{j=0}^{s} w_j r^j$. We use *signed radix-$2^k$ representation* for signed k-bit radix-2^k representation.

A. Related work

In [18] an SPE implementation is presented using arithmetic modulo the special prime $2^{255}-19$ introduced in [3]. SPE-arithmetic modulo a special prime is used in [9] to solve a 112-bit elliptic curve discrete logarithm problem on Cell processors. The SPE-performance of generic versus generalized Mersenne moduli is compared in [8]. SPE-arithmetic for moduli in the 200-bit range is presented in [6], [16]; on PS3s the former is more than twice faster than the latter. Different approaches to implement arithmetic over a binary extension field on SPEs are stated in [10].

Our usage of a small radix to avoid carries (cf. below) is not new [20], [31, Section 4.6], [6]. In [6] signed radix-2^{13} representation is used along with the SPE's $16\times 16\to 32$-bit multiplication instruction to develop fast multiplication modulo 195-bit moduli. All additions done during a single schoolbook multiplication are carry-less, requiring normalization to radix-2^{13} representation only at the end of the multiplication.

B. Representation of 4-tuples of integers modulo N

On the SPE it is advantageous to operate on four integers modulo N simultaneously, in 4-way SIMD fashion. Each 128-bit SPE register is interpreted as being partitioned into four 32-bit *words*. With s 128-bit registers thought to be stacked on top of each other, where $32s\geq M$, four different integers modulo N can be represented using four disjoint parallel columns, each consisting of s words: denoting the ith word of the jth register by w_{ij} for $i\in\{1,2,3,4\}$ and $j=0,1,\ldots,s-1$, the sequence $(w_{ij})_{j=0}^{s-1}$ is interpreted as the radix-2^{32} representation of the $32s$-bit integer $\sum_{j=0}^{s-1}w_{ij}2^{32i}$. More generally, for any $t\leq 32$ of one's choice, the sequence $(w_{ij})_{j=0}^{s-1}$ may represent the integer $\sum_{j=0}^{s-1}w_{ij}2^{ti}$ whose value depends on the interpretation of the words w_{ij}: as an unnormalized radix-2^t representation if the w_{ij} are interpreted as non-negative integers (normalized and unique if $w_{ij}<2^t$ as well), and as a signed k-bit radix-2^t representation, for some $k\leq 32$, if the w_{ij} are interpreted as signed k-bit integers.

It should be understood that the integer operations described below are always carried out in 4-way SIMD fashion on the SPE.

C. Addition and subtraction modulo N

Addition and subtraction in 4-way SIMD fashion on a pair of 4-tuples of integers modulo N in radix-2^t representation, with each 4-tuple represented by a stack of s registers of 128-bits (where $ts\geq M$), is done by applying s additions or subtractions to the matching pairs of registers (one from each stack), combined with a moderate number of carry propagations. The reduction modulo N most of the time affects just two of the radix-2^t digits, with probability $2^{-1-t-(M \bmod t)}$ that more digits are affected (in which case it causes a slight stall for the other three calculations in the 4-tuple).

For $t=32$ the SPE's built-in carry generation instructions are used, for smaller t-values somewhat more work needs to be done. For completeness (and future reference, cf. Step 5 in Section III-G), we describe the calculation of $c=a+b \bmod N$ and $d=a-b \bmod N$ (so-called *addition-subtraction* of a and b) given the signed radix-2^{13} representations $a=\sum_{j=0}^{95}a_j2^{13j}$ and $b=\sum_{j=0}^{95}b_j2^{13j}$. The following 5 steps are carried out:

1) Let $a'_j=a_j+2^{12}$ for $0\leq j<96$.
2) Set $c_j=a'_j+b_j$ and $d_j=a'_j-b_j$ for $0\leq j<96$.
3) Let the initial value of the carry τ be 0. For $j=0$ to 95 in succession first replace τ by $\tau+c_j$, next replace c_j by $\tau \bmod 2^{13}$, and finally replace τ by $\lfloor\tau/2^{13}\rfloor$. The resulting τ is a carry corresponding to $\tau\cdot 2^{13\cdot 96}$; modulo N this carry is taken care of by adding $\tau\cdot 2^{\alpha}$ to c_β (for $\gamma=13\cdot 96-M$, $\beta=\lfloor\gamma/13\rfloor$ and $\alpha=\gamma-13\beta\in[0,12]$) followed by a few more carry propagations. If there is still a carry which occurs rarely, use a more expensive function.
4) Repeat the previous step with c replaced by d.
5) Set $c_j=c_j-2^{12}$ and $d_j=d_j-2^{12}$ for $0\leq j<96$.

Steps 1, 2, and 5 allow arbitrary parallelization. Table I lists SPE clock cycle counts for the addition operations modulo $2^{1193}-1$: it can be seen that for signed radix-2^{13} representation they are more than twice slower than for radix-2^{32} representation.

D. Multiplication modulo N using radix conversions

Given a pair of 4-tuples of M-bit integers, the four pairwise products result in a 4-tuple of $2M$-bit integers. The four reductions modulo N can in principle be done by means of a few of the above 4-tuple additions and subtractions modulo N. Here we present our first approach that uses two different radix representations, thereby making it possible to take advantage of the fast radix-2^{32} addition and subtraction modulo N. In Section III-F another approach is described that is based on signed radix-2^{13} representation.

The multiplication modulo N of two M-bit integers a and b given by their radix-2^{32} representations, each using 39 words of 32 bits, proceeds in three steps that are described in more detail in sections III-D1 through III-D3. The steps are:

1) conversion of inputs a and b to signed radix-2^{13} representation;
2) carry-less calculation of the $2M$-bit product $a\cdot b$ in signed 32-bit radix-2^{13} representation;
3) reduction modulo N and conversion to radix-2^{32} representation of the $2M$-bit product $a\cdot b$, resulting in $c=a\cdot b \bmod N\in\{0,1,\ldots,N-1\}$.

The following sections describe the steps in more detail.

1) Conversion of inputs to signed radix-2^{13} representation: Given the radix-2^{32} representation of the precomputed constant $C_0 = 2^{12} \cdot \sum_{j=0}^{95} 2^{13j}$, first calculate the radix-2^{32} representation of $a + C_0$, in the usual way requiring carries. Next, using masks and shifts, extract the radix-2^{13} representation $(\tilde{a})_{j=0}^{95}$ of $a + C_0$, and finally subtract C_0 again by calculating $a_j = \tilde{a}_j - 2^{12}$, for $j = 0, 1, \ldots, 95$ (because $a_{96} = 0$ for our choice of M, it is dropped). The last two steps allow various straightforward parallelizations and run twice faster (while requiring fewer registers) if two 13-bit chunks are packed into a single 32-bit word. Applying the same method to b, we find signed radix-2^{13} representations of the inputs, below regarded as polynomials $P_a(X) = \sum_{j=0}^{95} a_j X^j$, $P_b(X) = \sum_{j=0}^{95} b_j X^j \in \mathbf{Z}[X]$ with $P_a(2^{13}) = a$ and $P_b(2^{13}) = b$.

2) Carry-less calculation of the $2M$-bit product in signed 32-bit radix-2^{13} representation: The product polynomial $P(X) = P_a(X)P_b(X) = \sum_{j=0}^{190} p_j X^j$ corresponds to the carry-less product calculation of a and b as represented by $(a_j)_{j=0}^{95}$ and $(b_j)_{j=0}^{95}$, respectively. Its coefficients satisfy $|p_j| \leq 96 \cdot (2^{12})^2 < 2^{31}$, which allows computation modulo 2^{32}, resulting in a signed 32-bit radix-2^{13} representation $(p_j)_{j=0}^{190}$ of the product $a \cdot b = P(2^{13})$. If $M < 13 \cdot w$ with $w < 96$, the degree of $P(X)$ will be at most $2w - 2 < 190$, which leads to savings here and in the description below.

The polynomial $P(X)$ is calculated using three levels of Karatsuba multiplication [30] (but see Section III-F2 for the possibility to use more levels), resulting in 27 pairs of polynomials $(P_a^{(k)}(X), P_b^{(k)}(X))$ of degree ≤ 11, for $k = 1, 2, \ldots, 27$ (in the more general case where $M < u \cdot v - 2$ we would use $16 - u$ levels). This leads to 27 independent polynomial multiplications $Q^{(k)}(X) = P_a^{(k)}(X)P_b^{(k)}(X)$, done using carry-less schoolbook multiplications. The polynomial $P(X)$ is then obtained by carry-less additions and subtractions of the appropriate $Q^{(k)}(X)$'s.

3) Reduction modulo N and conversion to radix-2^{32} representation of the $2M$-bit product: Given a signed 32-bit radix-2^{13} representation $(p_j)_{j=0}^{190}$ of the $2M$-bit product $a \cdot b$, regarded as the polynomial $P(X) = \sum_{j=0}^{190} p_j X^j$ with $P(2^{13}) = a \cdot b$, the radix-2^{32} representation $(c_i)_{i=0}^{38}$ of the M-bit number $c \equiv P(2^{13}) \bmod N$ is calculated. We use the following precomputed constants:

- $C_1 \equiv -2^{31} \cdot \sum_{j=0}^{190} 2^{13j} \bmod N$, $0 \leq C_1 < N$.
- Integers k_j, l_j and m_j such that

$$13j = m_j M + 32l_j + k_j$$
$$\text{with } 0 \leq 32l_j + k_j < M \text{ and } 0 \leq k_j < 32,$$

for $0 \leq j < 191$. Note that $m_j \in \{0, 1, 2\}$ because $M > 827$ (and $M < 1246$).

Given these values, the following four steps are carried out, the correctness of which easily follows by inspection:

1) For $0 \leq j < 191$, compute $\tilde{p}_j = p_j + 2^{31}$ (this allows arbitrary parallelization), so that $0 \leq \tilde{p}_j < 2^{32}$. As a result $(\sum_{j=0}^{190} \tilde{p}_j \cdot 2^{13j}) + C_1 \equiv P(2^{13}) \bmod N$.
2) For $0 \leq j < 191$, left shift $\tilde{p}_j$ over k_j bits and right shift $\tilde{p}_j$ over $32 - k_j$ bits, to obtain d_j, e_j such that
$$\tilde{p}_j \cdot 2^{13j} \equiv d_j \cdot 2^{32l_j} + e_j \cdot 2^{32(l_j+1)} \bmod N$$
(this again allows arbitrary parallelization).
3) Let $v_0 = 0$. For $0 \leq i < 39$, let
$$u_i = \sum_{j:l_j=i} d_j + \sum_{j:l_j+1=i} e_j, \tag{1}$$
(where the indices j can be precomputed) and compute
$$\tilde{c}_i = (v_i + u_i) \bmod 2^{32} \in \{0, 1, \ldots, 2^{32} - 1\},$$
$$v_{i+1} = \lfloor (v_i + u_i)/2^{32} \rfloor$$
(this allows partial parallelization). Finally, compute $\tilde{c}_{39} = v_{39} + \sum_{j:l_j=38} e_j$.
Using Eq. (1), reduction moduli N is effected by disregarding m_j and grouping together identical d_j-values and identical e_j-values. As a result, $(\tilde{c}_i)_{i=0}^{39}$ is the radix-2^{32} representation of a number $\tilde{c}$ with $\tilde{c} + C_1 \equiv c \bmod N$.
4) Calculate $c \equiv \tilde{c} + C_1 \bmod N$. Although the numbers are slightly bigger, this calculation is in principle the same as regular addition modulo N.

E. Optimizations

Swapping even for odd instructions. Modular arithmetic mostly relies on the SPE's arithmetic instructions, which are even pipeline instructions. Following the approach from [43], [11] one may replace an even instruction by one or more odd ones with the same effect. Although this may increase the latency for the functionality of each replaced even instruction and the number of instructions, balancing the counts of even and odd instructions often increases the throughput. This method was used throughout our implementation. Examples are sketched below.

Modular squaring. When squaring polynomials of degree at most 11, half of the mixed products, i.e., $\frac{12^2-12}{2} = 66$ multiplications, can be saved by doubling their resulting 21 sums (as the top elements are zero). Of these sums, the eleven for coefficients of odd degree can be doubled for free during the conversion to radix-2^{32}, by using for odd j precomputed integers $\tilde{k}_j$, $\tilde{l}_j$, and $\tilde{m}_j$ such that

$$13j + 1 = \tilde{m}_j M + 32\tilde{l}_j + \tilde{k}_j$$
$$\text{with } 0 \leq 32\tilde{l}_j + \tilde{k}_j < M \text{ and } 0 \leq \tilde{k}_j < 32,$$

instead of k_j, l_j, and m_j, as defined earlier. The ten remaining sums need to be doubled before they are added to the corresponding squared input coefficient. Each doubling can be done by a single even pipeline addition. However, a doubling can also be performed by four odd pipeline instructions (or two doublings in six odd pipeline instructions). The ten remaining doublings could thus be squeezed in the

odd pipeline, including all load and storage overheads (all 21 doublings would not have fit in the odd pipeline). As a result, all doublings required for squaring came for free.

Conversion to radix-2^{32}. The computation of d_j and e_j requires a shift by k_j and $32 - k_j$, respectively, for $0 \leq j < 191$, for a total of 382 even pipeline shift instructions. If $k_j \equiv 0 \bmod 8$, each shift can be replaced by a single odd pipeline byte reordering instruction (or by no instruction if $k_j = 0$). Shift counts bigger than 8 can be replaced by three odd pipeline instructions.

M-dependent optimization. For $0 \leq j < 191$ and most M we have $\sum_{j:l_j+1=i} e_j < 2^{32}$, since e_j is obtained by a right shift over $32 - k_j > 0$ bits and the shift amounts usually differ. Thus, for such M the second summation in Eq. (1) does not generate carries.

We have written a program that generates SPE code for each value of M, with the applicable C_0, C_1, k_j, l_j, m_j, $\tilde{k}_j$, $\tilde{l}_j$, and $\tilde{m}_j$ hard-coded and including all optimizations mentioned so far. The resulting code thus depends on the value of M used, with slightly varying performance between different M-values. Representative instruction and cycle counts for 4-way SIMD multiplication and squaring modulo $2^{1193} - 1$ on a single SPE are given in Table I. Because $\frac{78}{144} \cdot 3905 \approx 2115$, the 2130 cycles required for the calculation of the $Q^{(k)}$'s while squaring is very close to what one would expect based on the 3905 cycles required for multiplication.

F. Further speedups

Initial estimates indicated that the advantage of speed of the radix-2^{32} additions would outweigh the disadvantage of the conversion (in Section III-D1) to signed radix-2^{13} representations required for the carry-less product calculation. Only after the code based on the methods described above had been used for about nine months (obtaining the results as reported in Section IV) and two further improvements had been developed, this issue was revisited. The two improvements, in sections III-F1 and III-F2, apply to the first approach as well. The alternative version of the method from Section III-D3 that normalizes (and reduces) the signed 32-bit radix-2^{13} product to its signed radix-2^{13} representation (as opposed to converting and reducing the product to radix-2^{32} representation, as in Section III-D3) is presented in Section III-G.

1) Using $C_1 \equiv 0 \bmod N$ in Section III-D3: Let $\gamma = 13 \cdot 191 + 18 - M$, $\beta = \lfloor \gamma/13 \rfloor$ and $\alpha = \gamma - 13\beta$. To get non-negative $\tilde{p}_j$'s in the first step of Section III-D3, it suffices to put $\tilde{p}_0 = p_0 + 2^{31}$, $\tilde{p}_j = p_j + 2^{31} - 2^{18}$ for $1 \leq j < 191$, and next to replace $\tilde{p}_\beta$ by $\tilde{p}_\beta - 2^\alpha$ to make sure that the sum of all values added to $\sum_{j=0}^{190} p_j 2^{13j}$ telescopes to zero modulo N. Here we use that $p_j \geq -96(2^{12})(2^{12} - 1) > -2^{31} + 2^{19} > -2^{31} + 2^{18}$ and that $-2^{31} + 2^{19} > -2^{31} + 2^{18} + 2^\alpha$ (or $-2^{31} + 2^{19} > -2^{31} + 2^\alpha$ if $\beta = 0$). Thus C_1 in Section III-D3 is replaced by a value that is zero modulo N. This saves an addition (by C_1) in the final calculation of c in the fourth step of Section III-D3.

2) Karatsuba multiplication with multiply-and-add: A more substantial improvement is obtained by noting that for 26 out of the 27 k-values in Section III-D2 the coefficients of the polynomials $P_a^{(k)}(X)$ and $P_b^{(k)}(X)$ are signed 15-bit integers. Therefore, for these k another level of Karatsuba multiplication can be used for the calculation of $Q^{(k)}(X)$, while taking advantage of the SPE's multiply-and-add instructions. Some details are described below.

Let e, e', f, f' be four polynomials of degree $n - 1$. To multiply the two polynomials $e + e'X^n$ and $f + f'X^n$ of degree $2n - 1$, calculate $g = e - e'$ and $h = f' - f$ (note the asymmetry). Defining $ef = U + U'X^n$, $e'f' = V + V'X^n$ and $gh = W + W'X^n$, we have to calculate $(e + e'X^n)(f + f'X^n) = U + (U' + W + U + V)X^n + (V + W' + U' + V')X^{2n} + V'X^{3n}$. This is done by calculating (using multiply-and-add when relevant) U and U' in n^2 operations, next $U' + V$ and V' using another n^2 operations, $U' + V + U$ (n additions) and $U' + V + V'$ ($n - 1$ additions), and finally $U' + V + U + W$ and $U' + V + V' + W'$ using n^2 operations.

In this way this final level of Karatsuba multiplication requires $3n^2 + 4n - 1$ operations, which can be reduced to $3n^2 + 3n - 1$ if g and h can be calculated twice as fast, as in our case. With $n = 6$ this becomes 125 operations for the calculation of each of the 26 $Q^{(k)}(X)$'s to which this applies; the 27th one can be done in 144 operations, for a total of 3394 even instructions to calculate all $Q^{(k)}(X)$'s. For $n = 3$ we get $3n^2 + 3n - 1 = 35 < 6^2$, but the remaining parts of the 12-to-6-Karatsuba step take more than 20 operations, so more than $3 \times 35 + 20 = 125$ operations per $Q^{(k)}(X)$.

Improving the method from Section III-D using sections III-F1 and III-F2 would lead to a speedup of slightly less than 10% for modular multiplication and a much smaller speedup for modular squaring. We have not used this improvement as it led to only a small speedup of the ECM application. Instead we combined the improvements with the method presented in Section III-G below as it was expected (and turned out) to lead to a more substantial speedup for the ECM application.

G. Multiplication modulo N using signed radix-2^{13}

Multiplication modulo N with inputs and output in signed radix-2^{13} representation (and thus relatively slow addition operations) is obtained from the description in Section III-D by omitting the conversion in Section III-D1, keeping Section III-D2 in place (possibly improved as described in Section III-F2), and by replacing Section III-D3 by the reduction and normalization step described below.

1) Reduction modulo N and normalization to signed radix-2^{13} representation of the $2M$-bit product: Given a

Table I: SPE cycle counts for 4-way SIMD operations modulo $2^{1193} - 1$.

instructions even	instructions odd	cycles	**measured**		instructions even	instructions odd	cycles	**measured**
$a + b$ or $a - b$					$a + b$ and $a - b$			
120	117	144	**180**	radix-2^{32}	222	180	235	**268**
301	296	332	**363**	signed radix-2^{13}	553	394	571	**645**
$a \cdot b$				**original, radix 2^{32} inputs and output (Section III-D)**	a^2			
708	722	752		$P_a(X), P_b(X)$, and $P_a^{(k)}(X)$, $P_b^{(k)}(X)$ for $1 \le k \le 27$ / $P_a(X)$ and $P_a^{(k)}(X)$ for $1 \le k \le 27$	354	361	376	
3889	1137	3905		$Q^{(k)}(X)$ for $1 \le k \le 27$	2107	2055	2130	
1138	1078	1163		$P(X)$ and (d_j, e_j) for $0 \le j < 191$	1139	1086	1171	
906	907	936		$\tilde{c}_i$ for $0 \le i < 39$ and c	900	905	931	
6641	3844	6756	**6971**	total	4500	4407	4608	**4814**
$a \cdot b$				**signed radix-2^{13} inputs and output (sections III-F, III-G)**	a^2			
3622	1510	3637		$P_a^{(k)}(X)$, $P_b^{(k)}(X)$, and $Q^{(k)}(X)$ for $1 \le k \le 27$	2220	1921	2243	
1292	1172	1308		$P(X)$, steps 1, 2 and part of steps 3, 4 of Section III-G1	1299	1264	1340	
544	508	568		Steps 5, 6 and remainder of steps 3, 4 of Section III-G1	544	508	568	
5458	3190	5513	**5666**	total	4063	3693	4151	**4306**

signed 32-bit radix-2^{13} representation $(p_j)_{j=0}^{190}$ of the $2M$-bit product $a \cdot b$, regarded as the polynomial $P(X) = \sum_{j=0}^{190} p_j X^j$ with $P(2^{13}) = a \cdot b$, the signed radix-2^{13} representation $(c_j)_{j=0}^{95}$ of the M-bit number $c \equiv P(2^{13}) \bmod N$ is calculated.

1) Compute $(\tilde{p}_j)_{j=0}^{190}$ as described in Section III-F1.
2) For $0 \le j < 96$ replace $\tilde{p}_j$ by $\tilde{p}_j + 2^{12}$. (All additions in steps 1 and 2 are combined at a total cost of 191 even addition instructions for steps 1 and 2.)
3) For $96 \le j < 191$ let p'_j and p''_j be words such that $\tilde{p}_j = p'_j + p''_j 2^{16}$ and $0 \le p'_j, p''_j < 2^{16}$, and replace p'_j by $p'_j 2^{k'_j}$ and p''_j by $p''_j 2^{k''_j}$ using odd instructions, where $13j = m'_j M + 13\ell'_j + k'_j$ and $13j + 16 = m''_j M + 13\ell''_j + k''_j$.
4) For $96 \le j < 191$ replace $\tilde{p}_{\ell'_j}$ by $\tilde{p}_{\ell'_j} + p'_j$ and $\tilde{p}_{\ell''_j}$ by $\tilde{p}_{\ell''_j} + p''_j$ using a total of 190 even instructions. (No overflow occurs because $p'_j, p''_j \le 2^{28}$ and $p_j < (j+1)2^{24}$ for $0 \le j < 96$.)
5) Perform Step 3 of the addition-subtraction method in Section III-C with c (consisting of halfwords) replaced by $\tilde{p}$ (consisting of words). The carry τ can become as big as $2^{19} - 1$.
6) For $0 \le j < 96$ calculate the halfword $c_j = \tilde{p}_j - 2^{12}$.

Steps 1, 2, 3, 4, and 6 allow arbitrary parallelization. The resulting SPE clock cycle counts are listed in Table I.

H. Comparison with other SPE implementations

Because an SPE runs at 3.192GHz and six are available per PS3, it follows from Table I that a single PS3 can perform 13.5 (17.8) million multiplications (squarings) modulo N per second. This may be compared to 182 million and 138 million multiplications modulo 192-bit and 224-bit special moduli, respectively, as reported for a single PS3 in [8], i.e., less than an 11-fold slowdown for 5-fold bigger special moduli.

For generic moduli the same carry-less Karatsuba-based multiplication applies, but the reduction becomes more cumbersome. We expect we can do much better than the basic approach which would reduce our performance by a factor of at most three. Compared to the roughly 102 million modular multiplications for generic moduli in the 200-bit range, as reported for a single PS3 in [6], we would get at worst a 20-fold slowdown for 6-fold bigger generic moduli.

IV. Application to ECM

A. Background on ECM

ECM [38] attempts to factor a composite using a number of independent trials. The success probability per trial grows with the effort spent per trial, but decreases with the size of the smallest factor. Overall, the expected factorization effort for ECM (i.e., number of trials times effort per trial) grows subexponentially with the size of the smallest factor. For (S)NFS the effort does not depend on the size of the factor(s) but just on the size of the number being factored. For RSA moduli with two factors of about equal size, NFS is expected to be much faster than ECM. If there may be a relatively small factor (such as for composites of the form $2^M - 1$), ECM may be more efficient than (S)NFS.

Each ECM trial consists of two phases, phase one with bound B_1, which is compute intensive but requires little memory, followed by a memory-hungry phase two with bound B_2. Depending on the number of trials and the two bounds, the probability can be estimated that a factor up to a specific size, if present, will be found. To have probability at least $\frac{e-1}{e} \approx 0.632$ to find a factor of up to 65 decimal digits (when present), 24 000 ECM trials with $B_1 = 3 \cdot 10^9$ and $B_2 \approx 10^{14}$ suffice [58]. For the same bounds and success probability, 110 000 trials suffice to find a 70-digit factor (when present). Before our work the largest prime factor ever found using ECM had 68 decimal digits [56].

Using the GMP-ECM package [58], [57] on a single core of a 2.2GHz Athlon 2427, phase one for an ECM trial for 2^M-1 with M around 1100 takes on the order of six hours, phase two takes about one hour requiring many GBytes of RAM (for generic composites of comparable size each phase takes about twice as long; more precise timings are presented in Table IV in Section IV-C below). For each composite of the form $2^M - 1$ with $1000 \leq M \leq 1200$ this implies about 20 core years for an ECM attempt to find a 65-digit factor, and about 90 core years for a 70-digit one. This should be compared to an SNFS effort ranging from on the order or 70 ($M \approx 1000$) to several thousand ($M \approx 1200$) core years. Thus, the larger M, the harder we should first try with ECM, commensurate with the expected SNFS effort and the probability that a candidate has a small factor.

Each ECM trial performs a particular sequence of additions, subtractions, and multiplications modulo the number being factored. Modular inversions can mostly be avoided. Phase one can easily be run in parallel in SIMD fashion for *any* number of trials. During a large scale ECM effort, overall throughput of trials is, within reason, a more important performance measure than latency per trial: for instance, being able to process four trials simultaneously in one day is better than processing (on the same platform) one trial every eight hours.

Rationale to use Cell processors for ECM on 2^M-1. Factoring numbers of the form 2^M-1 is a "popular" activity [15] and hunting for relatively small factors is not hard given several freely available ECM packages. Nevertheless, given the efforts involved, we considered it likely that several of the unfactored composites 2^M-1 with $1000 \leq M \leq 1200$ have a factor that can be found more economically by ECM than by SNFS. Given our research interest in the ones that cannot (relatively) easily be factored by ECM, we decided on an ECM effort down our list of at least 20 candidates, aiming to find all factors of up to, roughly, 65 digits. Since it was meant to be a simple production run, we chose to use the off-the-shelf GMP-ECM package, because it is free, easy to use, has an excellent track-record, and can take advantage of the special form of the number 2^M-1. Other packages may be faster, but we were not familiar with them [5].

The overall computation requires at least $20 \times 20 = 400$ core years and can in principle be done on regular server-clusters. But that would be a waste of resources, because about $\frac{6}{7}$th of the time is spent in phase one, which requires little memory thereby underutilizing the available RAM.

We also have access to a cluster of 215 PS3s, and thus to 215 Cell processors comprising a total of 1290 SPEs with only little memory per SPE. It could therefore be more economical for us to use those SPEs to do all phase one calculations, and to do the relatively small phase two effort whenever servers with adequate RAM would otherwise be idle. To test this we ported phase one of GMP-ECM to the SPE, trying a variety of home-grown SPE-specific arithmetic packages (which were already known to outperform [29]). In the course of these early experiments we stumbled upon a 63-digit prime factor (of $2^{1187}-1$). This showed that conducting a thorough ECM search indeed makes sense, and stimulated development of the much faster SPE-arithmetic modulo 2^M-1 described in Section III.

Table II: SPE effort for 4-way SIMD phase one ECM trials for $N = 2^{1193} - 1$, $B_1 = 3 \cdot 10^9$ (where "cpc" = "cycles per call").

operation mod N	number of calls	radix-2^{32}		signed radix-2^{13}	
		cpc	hours	cpc	hours
$a \cdot b$	26 193 284 192	6971	15.89	5666	12.92
a^2	13 358 576 558	4814	5.60	4306	5.00
$a+b$, $a-b$	18 990 126 989	268	0.44	645	1.12
$a+b$	523 868 924	180	0.01		
$a-b$	523 868 924	180	0.01		
		total	21.95		19.05

It was not our goal to improve the ECM package that we put on top of our enhanced arithmetic. It is likely that improvements reported over GMP-ECM that are based on different elliptic curve arithmetic or representations, such as, for instance, described and implemented in [4], [5], apply to our overall performance figures as well.

ECM on the Cell processor to support (S)NFS. Although ECM factorizations have little cryptographic significance, this does not imply that ECM *performance* is cryptographically irrelevant as well. In [7], for instance, it is observed that high performance ECM implementations on relatively inexpensive devices (given their computational power, such as on graphics cards (GPUs)), may be helpful for future (S)NFS projects. A particularly memory-hungry step of (S)NFS, *sieving*, generates large quantities of fairly small (100- to 200-bit) composites that must be factored. That task requires little memory and is therefore best outsourced to cheap devices, so sieving is not interrupted and all resources are used in a cost-conscious fashion. This area has seen a flurry of recent activity: see [49], [45], [24], [22] for implementations on reconfigurable hardware such as field-programmable gate arrays and [7], [6] for GPUs. In [6] the Cell architecture is covered as well.

B. ECM on the Cell applied to $2^M - 1$

Table II lists the numbers of modular arithmetic operations carried out by phase one of a single ECM trial with bound $B_1 = 3 \cdot 10^9$ (cf. Section IV-A) when using GMP-ECM. When run on an SPE, four phase one trials are run simultaneously. With the operations from Section III, their cycle counts (cf. Table I), and the SPE's 3.192GHz clock speed, this leads to an estimated time of less than 22 hours on a single SPE to complete four phase one ECM trials with bound $B_1 = 3 \cdot 10^9$ using our first approach from Section III-D, and a more than 10% speedup when using the approach from Section III-G along with the improvements

from Section III-F. The measured wall-clock times are slightly larger than the estimates. For applications where additions play a more important role the method from Section III-D may outperform the method from Section III-G (where both methods are enhanced using Section III-F).

With six SPEs per Cell processor and 215 Cell processors in the PS3-cluster, $4 \times 6 \times 215 = 5160$ phase one ECM trials can be processed in less than 20 hours. With 24 000 trials (cf. Section IV-A), phase one for a 65-digit search takes less than four days; phase one for the 110 000 trials for a 70-digit search takes two and a half weeks. Using our multi-core adaptation of phase two of GMP-ECM, the corresponding phase two calculations (with $B_2 = 103\,971\,375\,307\,818$) take the same time when using 4 cores per node on a 56-node cluster (with two hexcore processors per node): each trial takes 15 minutes on 4 cores, using at most 16 GBytes of RAM. Thus, the efforts of the two clusters involved in our calculations are well matched.

After nine months of sustained calculations for several M-values (using the slower approach from Section III-D), seven new factors have been found, in the following order: a 63-digit factor for $M = 1187$, the 73-digit factor

$$1\,808\,422\,353\,177\,349\,564\,546\,512\,035\,512\,530\,001$$
$$279\,481\,259\,854\,248\,860\,454\,348\,989\,451\,026\,887$$

for $M = 1181$, another 73-digit factor,

$$1\,042\,816\,042\,941\,845\,750\,042\,952\,206\,680\,089\,794$$
$$415\,014\,668\,329\,850\,393\,031\,910\,483\,526\,456\,487,$$

for $M = 1163$, a 66-digit factor for $M = 1073$, a 63-digit factor for $M = 1051$, a 68-digit factor for $M = 1139$, and a 70-digit factor for $M = 1237$. The 241-bit, 73-digit prime factor of $2^{1181} - 1$ is the current ECM record, beating the previous record by 5 digits. The factor was found after somewhat more than 25 000 phase one trials at approximately the 8800th corresponding phase two trial, implying that we were quite lucky finding it. Less, but still considerable luck was involved in finding the second 73-bit factor (a bit smaller at 240 bits): it was found after about 50 000 ECM trials. So far our example number $2^{1193} - 1$ stubbornly resisted all ECM efforts to be factored after running 142 162 ECM trials on it. For the numbers $2^M - 1$ that we fail to factor using ECM, such as (so far) for $M = 1193$, our efforts will result in a reasonable degree of confidence that they will not have a prime factor of 65 digits or less. Only for $M = 1051$ and $M = 1237$ did we find composite cofactors: for $M = 1051$ the attempt was continued and the 63-factor was indeed re-found where it could be predicted (once it had been found), but the c248 cofactor remained unfactored.

Table III lists all results obtained using the slower approach from Section III-D, with ck and pk denoting a k-digit composite and prime, respectively. For exponents $M \in [1000, 1125]$ ($M \in [1126, 1200]$) not stated in Table III roughly 25 000 (50 000) ECM trials have been completed with bounds as above without finding a factor.

Table III: Factors found of $2^M - 1$ using ECM on the Cell with the arithmetic described in Section III-D of this paper, and with $B_1 = 3 \cdot 10^9$ and $B_2 \approx 10^{14}$.

M	targeted composite	completed number of trials		result
		phase one	phase two	
1051	c310	23 136	9 186	p63 · c248
1073	c281	24 504	1 460	p66 · p215
1139	c313	49 080	35 490	p68 · p246
1163	c318	50 152	47 768	p73 · p246
1181	c291	25 393	8 808	p73 · p218
1187	c266	15 089	9 860	p63 · p204
1237	c373	71 556	70 809	p70 · c303

Table IV: Time to complete 24 phase one ECM trials.

processor	GHz	cores	hours	
			Mersenne	generic
Intel Xeon E5430	2.66	8	23.70	43.13
Intel Core i7 920	2.67	4	46.28	83.52
Intel Core2 Quad Q9550	2.83	4	47.26	85.93
Intel Core2 Quad Q6700	2.66	4	48.80	86.45
AMD Phenom 9500	2.22	4	38.48	65.75
AMD Opteron 1381	2.50	4	33.78	58.46
PlayStation 3	3.19	6	19.20	

Although we hope, during our continuing efforts using the faster approach from sections III-F and III-G, not to miss factors up to the 65-digit range, with ECM one can never be sure. Should we wish to find out, using SNFS is probably the best option. Using the improved arithmetic we have so far found one factorization: for $M = 961$ we found that c254 = p61 · p193 after 1190 curves with $B_1 = 10^9$ and $B_2 = 25\,427\,965\,563\,016$.

C. Comparison between Cell and regular processors.

A single PS3 processes 24 phase one ECM trials for $2^{1193} - 1$ in 19.2 hours. To put this number in perspective, we did the same computation using GMP-ECM 6.3 powered by GMP 5.0.1 (both the latest versions) using all cores on a variety of processors, with optimal multiplication parameters obtained using the tune-up script, and while taking advantage of the special Mersenne-arithmetic available in GMP-ECM. Table IV lists the results. It can be seen that for this application a single PS3 outperforms several common 4-core platforms by a factor of more than 2. On a per-core basis, and accounting for the ratio in clock-speeds, our special 4-way SPE Mersenne arithmetic turns out to about $\frac{5}{4}$ times more effective than the regular Mersenne arithmetic from GMP-ECM 6.3 when run on Intel processors, despite the fact that the SPE does not have 64-bit or 32-bit integer multiplications. The lack of such multipliers is, however, clearly to the SPE's disadvantage when comparing it to the AMD processor with its much faster (than Intel) integer multiplication.

V. Conclusion

For integers M in the range from 1000 to 1200 we presented our Cell processor implementation of multiplication of M-bit integers, processing 24 such multiplications in parallel on a single PlayStation 3 game console, and used it to obtain efficient multiplication modulo $2^M - 1$. The ideas underlying our implementation apply to many arithmetic contexts of cryptologic relevance, such as elliptic curve cryptosystems and cryptanalysis thereof.

We focused on application of our arithmetic to elliptic curve factoring, as a preparatory step for a potential (S)NFS factoring project. This led to the three largest factors found using ECM so far.

Acknowledgements. Much appreciated incisive comments by the reviewers helped improve the focus of this paper. This work was supported by the Swiss National Science Foundation under grant numbers 200021-119776, 200020-132160 and 206021-117409 and by EPFL DIT. Paul Zimmermann kindly provided us with the elliptic curve group orders. *Acknowledgements.*

References

[1] J.-C. Bajard, L. Imbert, and T. Plantard. Modular number systems: Beyond the Mersenne family. In *Selected Areas in Cryptography*, volume 3357 of *LNCS*, pages 159–169. Springer, 2004.

[2] J.-C. Bajard, N. Meloni, and T. Plantard. Efficient RNS bases for cryptography. In *IMACS'05 : World Congress: Scientific Computation Applied Mathematics and Simulation*, 2005.

[3] D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *PKC 2006*, volume 3958 of *LNCS*, pages 207–228, 2006.

[4] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. ECM using Edwards curves. Cryptology ePrint Archive, Report 2008/016, 2008. http://eprint.iacr.org/.

[5] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. EECM: ECM using Edwards curves. http://eecm.cr.yp.to/, 2010.

[6] D. J. Bernstein, H.-C. Chen, M.-S. Chen, C.-M. Cheng, C.-H. Hsiao, T. Lange, Z.-C. Lin, and B.-Y. Yang. The billion-mulmod-per-second PC. In *Workshop record of SHARCS'09*, pages 131–144, 2009. http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf.

[7] D. J. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, and B.-Y. Yang. ECM on graphics cards. In *Eurocrypt 2009*, volume 5479 of *LNCS*, pages 483–501, 2009.

[8] J. W. Bos. High-performance modular multiplication on the Cell processor. In *WAIFI 2010*, volume 6087 of *LNCS*, pages 7–24, 2010.

[9] J. W. Bos, M. E. Kaihara, and P. L. Montgomery. Pollard rho on the PlayStation 3. In *Workshop record of SHARCS'09*, pages 35–50, 2009. http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf.

[10] J. W. Bos, T. Kleinjung, R. Niederhagen, and P. Schwabe. ECC2K-130 on Cell CPUs. In *Africacrypt 2010*, volume 6055 of *LNCS*, pages 225–242, 2010.

[11] J. W. Bos and D. Stefan. Performance analysis of the SHA-3 candidates on exotic multi-core architectures. In *CHES 2010*, volume 6225 of *LNCS*, pages 279–293, 2010.

[12] S. Boussakta and A. Holt. New transform using the mersenne numbers. *Vision, Image and Signal Processing, IEE Proceedings* -, 142(6):381–388, December 1995.

[13] R. P. Brent. Factorization of the tenth Fermat number. *Mathematics of Computation*, 68(225):429–451, 1999.

[14] R. P. Brent and J. M. Pollard. Factorization of the eighth Fermat number. *Mathematics of Computation*, 36(154):627–630, 1981.

[15] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr. Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers. American Mathematical Society, First edition, 1983, Second edition, 1988, Third edition, 2002. Electronic book available at: http://homes.cerias.purdue.edu/~ssw/cun/index.html, 1983.

[16] H.-C. Chen, C.-M. Cheng, S.-H. Hung, and Z.-C. Lin. Integer number crunching on the cell processor. *International Conference on Parallel Processing*, pages 508–515, 2010.

[17] J. Chung and M. A. Hasan. More generalized mersenne numbers. In *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 335–347. Springer, 2003.

[18] N. Costigan and P. Schwabe. Fast elliptic-curve cryptography on the Cell broadband engine. In *Africacrypt 2009*, volume 5580 of *LNCS*, pages 368–385, 2009.

[19] R. Crandall and B. Fagin. Discrete weighted transforms and large-integer arithmetic. *Mathematics of Computation*, 62(205):305–324, 1994.

[20] R. Crandall and B. Fagin. Discrete weighted transforms and large-integer arithmetic. *Math. of Comp.*, 62(205):305–324, 1994.

[21] A. J. C. Cunningham and H. J. Woodall. Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers. Frances Hodgson, London, 1925.

[22] G. de Meulenaer, F. Gosset, G. M. de Dormale, and J.-J. Quisquater. Integer factorization based on elliptic curve method: Towards better exploitation of reconfigurable hardware. In *FCCM 2007*, pages 197–206. IEEE Computer Society, 2007.

[23] V. Dimitrov, T. Cooklev, and B. Donevsky. Generalized Fermat-Mersenne number theoretic transform. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 41(2):133–139, February 1994.

[24] K. Gaj, S. Kwon, P. Baier, P. Kohlbrenner, H. Le, M. Khaleeluddin, and R. Bachimanchi. Implementing the elliptic curve method of factoring in reconfigurable hardware. In *CHES 2006*, volume 4249 of *LNCS*, pages 119–133, 2006.

[25] H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, 8:140–147, 1959.

[26] GIMPS Home Page. The great internet mersenne prime search. http://www.merssene.org, 2010.

[27] J. Harrison. Isolating critical cases for reciprocals using integer factorization. In *IEEE Symposium on Computer Arithmetic (Arith-16)*, pages 148–157, 2003.

[28] G. Hotz. Here's your silver platter. http://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/.

[29] IBM. Multi-precision math library. Example Library API Reference. Available at https://www.ibm.com/developerworks/power/cell/documents.html.

[30] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. Number 145 in Proceedings of the USSR Academy of Science, pages 293–294, 1962.

[31] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[32] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[33] D. H. Lehmer. An extended theory of Lucas' functions. *Annals of Mathematics*, 31(3):419–448, 1930.

[34] D. N. Lehmer. Hunting big game in the theory of numbers. Scripta Mathematica, March 1933.

[35] A. K. Lenstra. Unbelievable security: Matching AES security using public key systems. In *Asiacrypt 2001*, volume 2248 of *LNCS*, pages 67–86, 2001.

[36] A. K. Lenstra and H. W. Lenstra, Jr. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.

[37] A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number. *Mathematics of Computation*, 61(203):319–349, 1993.

[38] H. W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649–673, 1987.

[39] E. Lucas. Théorie des fonctions numériques simplement périodiques. *American Journal of Mathematics*, 1(2):184–196, 1878.

[40] R. D. Merrill. Improving digital computer performance using residue number theory. *Electronic Computers, IEEE Transactions on*, EC-13(2):93–101, April 1964.

[41] V. S. Miller. Use of elliptic curves in cryptography. In *Crypto 1985*, volume 218 of *LNCS*, pages 417–426, 1986.

[42] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29(129):183–205, 1975.

[43] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright. Fast software AES encryption. In *FSE 2010*, volume 6147 of *LNCS*, pages 75–93, 2010.

[44] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design (4th Edition): The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[45] J. Pelzl, M. Šimka, T. Kleinjung, M. Drutarovský, V. Fischer, and C. Paar. Area-time efficient hardware architecture for factoring integers with the elliptic curve method. *Information Security, IEE Proceedings on*, 152(1):67–78, oct 2005.

[46] J. M. Pollard. The lattice sieve. pages 43–49 in [36].

[47] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[48] A. Shamir. CryptoBytes Technical Newsletter. ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n3.pdf.

[49] M. Šimka, J. Pelzl, T. Kleinjung, J. Franke, C. Priplata, C. Stahlke, M. Drutarovský, and V. Fischer. Hardware factorization based on elliptic curve method. In *FCCM 2005*, pages 107–116. IEEE Computer Society, 2005.

[50] A. Skavantzos and P. Rao. New multipliers modulo $2^n - 1$. *IEEE Transactions on Computers*, 41:957–961, 1992.

[51] J. A. Solinas. Generalized Mersenne numbers. Technical Report CORR 99-39, Centre for Applied Cryptographic Research, University of Waterloo, 1999.

[52] M. Stevens, A. K. Lenstra, and B. de Weger. Predicting the winner of the 2008 US presidential elections using a Sony PlayStation 3. http://www.win.tue.nl/hashclash/Nostradamus/.

[53] M. Stevens, A. Sotirov, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO*, volume 5677 of *LNCS*, pages 55–69, 2009.

[54] F. Taylor. Large moduli multipliers for signal processing. *Circuits and Systems, IEEE Transactions on*, 28(7):731–736, July 1981.

[55] U.S. Department of Commerce and National Institute of Standards and Technology. Digital Signature Standard (DSS). See http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf, 2009.

[56] yoyo@home and M. Thompson. Found gmp-ecm top50 factor. http://www.loria.fr/~zimmerma/records/p68, 2009.

[57] P. Zimmermann and B. Dodson. 20 years of ECM. In *ANTS*, volume 4076 of *LNCS*, pages 525–542, 2006.

[58] P. Zimmermann et al. GMP-ECM (elliptic curve method for integer factorization). https://gforge.inria.fr/projects/ecm/, 2010.