Pushing the Limits of ECM

Joppe W. Bos[1], Thorsten Kleinjung[1], Arjen K. Lenstra[1], and Peter L. Montgomery[2]

[1] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

Abstract. This paper describes our implementation of phase one of the elliptic curve method on the Cell processor and reports on actual record factors obtained. Our implementation uses a new and particularly efficient variable radix multiplication of independent interest.
Keywords: Elliptic curve method, Integer factorization, Cell processor, PlayStation 3, Mersenne numbers

1 Introduction

The study of theoretical and practical aspects of integer factorization algorithms is a subject of continuing interest for the security assessment of various common public-key cryptosystems. Factorization milestones are therefore carefully monitored by many applied cryptographers. Of these milestones, the ones that set new records for general integer factorization count the number of bits of the number factored and thus have direct implications for the size of RSA moduli. They are of greatest cryptographic interest. Examples are the factorizations of a 512-bit RSA modulus [13] and, most recently, of a 768-bit one [18], both obtained using the *number field sieve* (NFS, [21]).

NFS is an outgrowth of a method that is now referred to as *special* NFS (SNFS), the invention of which was inspired by the strong desire of its inventor (John Pollard) to factor the ninth Fermat number $F_9 = 2^{2^9} + 1$ in 1990. SNFS applies only to composites of a special form (such as F_9) and is several orders of magnitude faster than NFS applied to RSA moduli of the same size. Nevertheless, the successful factorization of F_9 (indeed in 1990 [22]) was interpreted by many as the writing on the wall for 512-bit RSA moduli: it took NFS almost a decade to realize the perceived threat. SNFS records are also measured by the size of the number factored. They still serve as warning signals. The current SNFS record factorization of $2^{1039} - 1$ from [1] does not directly affect the security of 1024-bit RSA moduli, but neither does it inspire any confidence in their continued usage.

The third category of integer factorization milestones, records obtained using the *elliptic curve method* (ECM, [23]), belong more to the realm of recreational mathematics. ECM, as further explained below, can only be expected to outperform NFS and SNFS in the presence of relatively small factors. ECM records measure the size of the second largest factor or cofactor found. Regular RSA moduli are known not to have small factors and NFS would factor them much

faster than ECM. Numbers of special form may have factors of any kind, so for those ECM stands a chance to beat SNFS. It is indeed for such special numbers that armies of factoring buffs are in a friendly worldwide competition to try and squeeze the most out of ECM. Lists of annual and all-time records are carefully maintained [11,29] and achievements are enthusiastically discussed on a variety of discussion forums. Although to outsiders it may look a bit like growing grass, with years without visible growth except for a thickening of the undergrowth, this makes it all the more exciting when something happens: after a dry-spell of more than three years, the August 2006 67-digit record was beaten in December 2009 by a 68-digit ECM factor. Not the long anticipated 70-digit factor yet, but progress at last. In this paper we report on the latest, and even more recent, ECM record, a 73-digit factor found in March 2010. That this unusually large jump is not a fluke was shown by a slightly smaller, second 73-digit factor, found in April 2010. We also report on how we happened into the ECM record business and, more importantly, on the carry-less arithmetic operations that were used and that are of independent interest.

None of the ECM factors reported on the various record lists has much cryptographic significance. The only cryptographic implication of ECM records that we are aware of has arguably rather weak practical importance, namely their consequence for the RSA variant known as *RSA multiprime*. To gain a speedup by a factor of r^2 or $\frac{r^2}{4}$ for the private operation in vanilla RSA or CRT-RSA, respectively, one may select RSA moduli (of appropriate size to be out of reach of NFS) that consist of the product of $r > 2$ primes of about the same size. Here r must be chosen in such a way that ECM has a sufficiently low probability to find the resulting relatively small prime factors. Our ECM record affirms that 1024-bit RSA moduli with $r \geq 4$ should be avoided [20] and may give RSA multiprime practitioners some guidance how large r may be chosen. We suspect, however, that most cryptographers are more interested in new ECM records because of their mild entertainment value.

The limited cryptographic significance of ECM records does not imply that ECM *performance* is cryptographically irrelevant as well. Indeed, this area has seen a flurry of recent activity [26,25,16,15,6,5]. In [6], 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 to reach the next NFS factoring milestone. The memory-hungry NFS sieving step generates large quantities of fairly small, say 100- to 200-bit composites that must be factored. That task requires little memory and is therefore best outsourced to cheap devices, so the siever can stay sieving and all resources are used in a cost-conscious fashion.

This type of ECM work naturally follows from a line of research that is of central interest to practical cryptology: fast arithmetic in finite fields and groups of elliptic curves on any type of inexpensive device, ranging from game consoles and GPUs to FPGAs and ASICs. Results affect not just the feasibility of cryptographic applications but may also have cryptanalytic impact. For instance, the very substantial amount of computing required for parallelized Pollard rho exper-

iments to assess the security of elliptic curve cryptosystems may cost-effectively be harvested from any combination of devices as mentioned above [2].

Similarly, the new ECM work presented in this paper was not pursued in its own right. Instead it resulted from the conflux of two independent lines of work, both cryptographically relevant, neither inspired by ECM, but nevertheless resulting in entertaining ECM results. This story is told in Section 2, along with a comparison with relevant previous work. Section 3 gives some background on the Cell processor (on which we ran ECM) and describes its architecture. The newly developed arithmetic to run ECM on the Cell processor is presented in Section 4 and, finally, ECM on the Cell is described in Section 5, including our new ECM records.

2 Background

As part of a potential (S)NFS project, we need a list of hard to factor composites of the form $2^M - 1$ for exponents M in the range from 1 100 to 1 400. Because SNFS factorization of each of these numbers requires on the order of 200 to tens of thousands of years of computing on a single core[3] the easy cases must be weeded out from the current list of not fully factored candidates. That is typically done using an ECM effort commensurate with the expected SNFS effort: the larger the exponent, the harder one should try first with ECM.

ECM primer. ECM attempts to factor a composite using a number of independent trials. Each 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 find a factor of up to 65 decimal digits with probability at least $\frac{e-1}{e} \approx 0.632$ (when present), 24 000 ECM trials with $B_1 = 3 \cdot 10^9$ and $B_2 \approx 10^{14}$ suffice. For the same bounds, 110 000 trials suffice to find a 70-digit factor (when present) with the same probability. On a single core, phase one for an ECM trial with M around 1 100 takes on the order of six hours, phase two takes about one hour[4] requiring many GBytes of RAM. For one of our candidates 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.

ECM needs many additions, subtractions, and multiplications modulo the number being factored. Modular inversions are as much as possible avoided. Phase one for any number of trials can easily be run in parallel in *Single Instruction Multiple Data* (SIMD) fashion. During a large scale ECM effort, overall

[3] All performance figures for regular processors will be for a single core of a 2.2 GHz Athlon 2427 or of a 2.66 GHz Core2. These are good representatives of currently popular processors with, for the rough figures presented, comparable performance. If k cores are used, the wall clock time is reduced by a factor of k, due to the embarrassingly parallel nature of our applications.

[4] A generic composite of comparable size take about twice as long.

throughput is, within reason, more important than latency per trial: being able to process four simultaneous trials in a day is better, on the same platform, than processing one trial every eight hours.

Finding large factors using ECM requires luck and persistence, but too much of either is not advisable: if one's coveted ECM result is deemed to be easier to find using other methods than by means of ECM, then it will not be accepted on record lists. See [11] for the criterion.

Running ECM on the target numbers. Given the "popularity" of numbers of the form 2^M-1, they have, since the invention of ECM in 1985, been subjected to substantial ECM efforts by others. Although we would be surprised if factors of 55 digits or less would still show up, we have no way to estimate the probability that one of our candidate numbers still has a factor of, say, 65 digits. Thus, we decided on an ECM effort down our entire list of candidates, aiming to find all factors of up to, roughly, 65 digits (as that size is near the top range of what seemed doable, cf. [11,29]), to lower the probability that any remaining number later turns out to be an *ECM miss*: a number that could have been factored easier using ECM. Since this was not a research effort but a simple production run, we decided to use an off-the-shelf ECM package. Given its easy availability, ease-of-use, excellent track-record, and ability to take advantage of the special form of our numbers, we opted for the GMP-ECM package [31]. Other packages may be faster, but we were not familiar with them [4].

Given the sizes of our server-clusters it would take several years to complete an adequate number of ECM trials for all our target numbers. It would be a waste of resources, because most of the time RAM would be underutilized. Worse, it would be a misallocation of resources, since the server-clusters are meant for (and fully occupied by) a variety of research projects and are not intended for production runs.

By coincidence, the cluster of 215 Sony PlayStation 3 (PS3) game consoles that we have access to had just finished a large cryptanalytic project and was not working on anything that was considered to be urgent. Thus it was decided to port phase one of GMP-ECM to the PS3 cluster, while running phase two, which requires more memory than available on a PS3, on regular servers. This required experimentation with the many different PS3-specific arithmetic packages that we had developed over the years to find out which one would suit GMP-ECM best. We achieved decent performance right away, and were lucky to stumble upon a 63-digit prime factor (of $2^{1187}-1$), not close to an ECM record but encouraging nevertheless – and proving that conducting a thorough ECM search makes sense. Closer study was then made of the arithmetic required for the exponents M in our range of interest, resulting in the twice faster arithmetic described in Section 4 and the ECM records reported in Section 5.

Although we are happy with any ECM factor that we find in this way, the goal of the ECM project is to get a list of numbers that we failed to factor using ECM so they can be used for our SNFS experiments. Only after the latter experiments have been concluded can we tell how successful our ECM trials have been – we can only hope that no factors of 65 digits or less were missed by ECM.

The arithmetic package that we developed, however, is of interest in its own right and may find other applications elsewhere as well.

Note that it has not been our goal to improve the ECM package that we put on top of our enhanced arithmetic. There is every reason to expect that improvements reported over GMP-ECM [31,30] that are based on different elliptic curve arithmetic or representations, such as, for instance, described and implemented in [3,4], apply to our overall performance figures as well. Besides these general purpose implementations that target arbitrary sized numbers, there is a growing interest in ECM as a cofactor factorization method for NFS applied to a 1024-bit RSA modulus. In such applications, ECM is typically applied to composites in the 200-bit range: see [26,25,16,15] for implementations on reconfigurable hardware such as field-programmable gate arrays and [6,5] for GPUs. In [5] the Cell architecture is covered as well.

3 The Cell processor and its architecture

Although it was introduced already over four years ago, the Cell processor as embedded in the PS3[5] is still a relatively inexpensive and flexible source of compute power. We have exploited this for a variety of cryptanalytic projects. Examples include exhaustive key searches or password cracking, chosen prefix collisions for the cryptographic hash function MD5 [27], the creation of a rogue Certification Authority certificate [28], the solution of an elliptic curve discrete logarithm problem over a 112-bit prime field [8], and the implementation of arithmetic in an elliptic curve group over a degree-130 binary extension field [9].

The Cell processor is quite different from 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. It not only helps us to take advantage of the Cell's inexpensive processing power, it also helps to prepare ourselves for future generations of processors.

The Cell's main processing power comes from eight *Synergistic Processing Units* (SPUs). They run independently from each other at 3.192GHz, each working on its own 256 kilobyte of fast local memory (the *Local Store*) for instructions and data and their own 128 registers of 128 bits each. The latter allow SIMD operations on sixteen 8-bit, eight 16-bit, or four 32-bit integers. There are many boolean operations, but integer multiplication is limited to several 4-way SIMD $16 \times 16 \rightarrow 32$-bit multipliers including a multiply-and-add. There is no $32 \times 32 \rightarrow 64$-bit or $64 \times 64 \rightarrow 128$-bit multiplier. The SPU has an odd and even pipeline: per clock cycle it can dispatch one odd and one even instruction, assuming the instructions are independent. Because the SPU lacks smart branch prediction, branching is best avoided. The Cell also has a *Power Processing Element* (PPE), a dual-threaded 64-bit processor with a 128-bit AltiVec/VMX SIMD unit.

[5] Early versions of the PS3 allow access to the Cell processor via Sony's hypervisor. This feature has been disabled in current versions.

When running Linux, six SPUs can be used (one is disabled, and one is reserved by the hypervisor[6]). We have access to a cluster of 215 PS3s, comprising 1290 accessible SPUs. Thus, given the low memory requirement of phase one of ECM, in principle 1290 phase one ECM trials can be run independently in parallel, without any need for synchronization. However, as mentioned, our performance measure is overall throughput, while latency per ECM trial is mostly irrelevant. Given the SIMD-parallelizability of phase one of ECM and the 4-way SIMD nature of the SPUs, it may thus pay off to run four trials in SIMD-parallel fashion per SPU. This would result in $4 \cdot 1290 = 5160$ parallel phase one ECM trials. As shown below, this is indeed what we did.

For some applications, multiple SIMD processes may even be interleaved, filling both pipelines to increase throughput, while possibly increasing per-process latency. For our ECM trials below we did not do so but took advantage of interleaving in another manner.

4 Arithmetic modulo $2^M - 1$ on the SPU

Any ECM package will have to rely, one way or another, on arithmetic modulo the number to be factored. In this section we describe the SPU-arithmetic that we developed for arithmetic modulo $2^M - 1$, for M well beyond one thousand. As pointed out above our approach aims to optimize overall throughput.

Given the SPU's instruction set, it turned out to be most advantageous to use a 4-way SIMD approach (as already alluded to above), implying that always four integers modulo $2^M - 1$ are operated upon simultaneously. This is achieved as follows. Each 128-bit SPU register r is partitioned into four 32-bit *words* r_1, r_2, r_3, and r_4. The different words r_i of a single register r contain bits of different integers modulo $2^M - 1$, but word r_i of register r and word v_i of register v may both contain bits of the same integer modulo $2^M - 1$: if s registers are thought of as being stacked horizontally on top of each other, four different integers modulo $2^M - 1$ are represented using the four disjoint parallel 32-bit wide vertical columns of height s, respectively. It follows that s must be chosen such that $32s \geq M$. If $t < 32$ bits are used per word – as will frequently be done below – then it must be the case that $ts \geq M$.

Addition and subtraction in 4-way SIMD fashion on a pair of 4-tuples of integers modulo $2^M - 1$ in radix 2^t-representation, with each 4-tuple represented by a stack of s registers of 128-bits as described above (for some s), is done by applying s additions or subtractions to the matching pairs of registers (one from each stack), combined with a moderate amount of fiddling around with carries. The reduction modulo $2^M - 1$ most of the time affects just two of the radix-2^t digits, with probability on the order of 2^{-2t-1} that more digits are affected (in which case it causes a slight stall for the other three calculations in the 4-tuple). This can be made to work quickly for any reasonable value of t such as $t = 32$.

Multiplication is more challenging and is described in the remainder of this section. A 4-tuple of $2M$-bit numbers (containing the products of a pair of 4-

[6] This may change, see [17].

tuples of M-bit integers) can in principle be reduced modulo $2^M - 1$ by means of a few of the above 4-tuple additions and subtractions modulo $2^M - 1$. In our implementaton the reductions modulo $2^M - 1$ are folded into the final part of the multiplications, as further elaborated upon below.

4.1 Carry-less M-bit$\times M$-bit $\rightarrow 2M$-bit multiplication on the SPU

We give an outline of our method to multiply two M-bit numbers on the SPU. It should be understood that our multiplication is done in 4-way SIMD fashion, so it always operates on a pair of 4-tuples of M-bit integers and produces a 4-tuple containing the respective $2M$-bit products. Assume that $M < 13 \cdot 96 = 1248$, i.e., in the lower part of our range of interest. More generally one would put $M < u \cdot v$ with $v \cdot (2^{u-1})^2 < 2^{31}$. This would also accommodate larger M-values.

Let a, b be two M-bit numbers to be multiplied. Assuming a and b are in radix-2^{32} representation (39 words of 32 bits for each), we first calculate their **signed 12-bit radix-2^{13} representation**: $P_a(X) = \sum_{i=0}^{95} a_i X^i \in \mathbf{Z}[X]$ with $a_i \in [-2^{12}, 2^{12})$ such that $P_a(2^{13}) = a$, and similarly for b. The coefficients of the polynomial $P(X) = P_a(X) \cdot P_b(X) = \sum_{i=0}^{190} p_i X^i$ satisfy $|p_i| \leq 96 \cdot (2^{12})^2 < 2^{31}$, so that $P(X)$ can be computed modulo 2^{32}, and $P(2^{13})$ contains a signed 31-bit radix-2^{13} representation of the product $a \cdot b$. 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, resulting in 27 pairs of polynomials $(P_a^{(j)}(X), P_b^{(j)}(X))$ of degree ≤ 11, for $j = 1, 2, \ldots, 27$ (in the more general case we can do $16 - u$ levels of Karatsuba multiplication). This leads to 27 independent polynomial multiplications $Q^{(j)}(X) = P_a^{(j)}(X) P_b^{(j)}(X)$, done using carry-less schoolbook multiplication. The polynomial $P(X)$ is then obtained by carry-less additions and subtractions of the appropriate $Q^{(j)}(X)$'s. The desired product $a \cdot b$ can be obtained by converting $P(2^{13})$ to its radix-2^{32} representation. Below, however, $P(X)$ is used to find directly the radix-2^{32} representation of the M-bit number $(a \cdot b) \bmod (2^M - 1)$.

Rationale. The idea to avoid carries by using a small radix is not new (cf. [14] and [19, Section 4.6]). In [5], for instance, signed 12-bit radix-2^{13} representation as above is used along with the SPU's $16 \times 16 \rightarrow 32$-bit multiplication instruction, so that all additions done during a single schoolbook multiplication are carry-less, requiring normalization to the proper radix-2^{13} representation only at the end of the multiplication. For our application on the same platform (the SPU of the PS3), however, it turned out to be advantageous to transform the $2M$-bit product, in unnormalized radix-2^{13} representation (resulting from carry-less additions: as explained above involving signed 31-bit digits), to the regular radix-2^{32} representation of its remainder modulo $2^M - 1$, before transforming that representation to a signed 12-bit radix-2^{13} representation again. At first glance this looks rather awkward, but not only does the intermediate 32-bit

representation allow fast addition and subtraction modulo $2^M - 1$ (as often required in elliptic curve arithmetic), the conversion to radix 2^{32} can be made to conveniently absorb the reduction modulo $2^M - 1$. The various conversions are sufficiently fast that overall a speedup is achieved compared to any other approach that we fully implemented.

Below we describe some relevant implementation details, including the two radix conversion methods required. Our implementations are particularly suited to the SPU, but the approach may have broader applicability.

4.2 Conversion from radix-2^{32} to signed 12-bit radix-2^{13}

When converting 4-tuples of M-bit numbers from their regular radix-2^{32} representation to signed-digit radix-2^{13} representation, we store the resulting 4-tuples of two signed 12-bit integers in a 4-tuple of 32-bit words, one in the higher and one in the lower 16-bit part. This halves the number of registers required to represent 4-tuples of M-bit integers while at several places speeding up the computation by a factor of two.

Let $C_0 = 2^{12} \cdot \sum_{i=0}^{95} 2^{13i}$ be a precomputed constant, given in its radix-2^{32} representation. The polynomial $P_a(X) = \sum_{i=0}^{95} a_i X^i$ is determined by first adding C_0 to a, by extracting (using masks and shifts) the regular radix-2^{13} representation $\sum_{i=0}^{95} \tilde{a}_i \cdot 2^{13i}$ of the sum $a + C_0$, and by subtracting C_0 again by putting $a_i = \tilde{a}_i - 2^{12}$. The addition of C_0 requires carries, as usual, but the other two steps can mostly be parallelized into independent steps and run twice faster if two 13-bit or two signed 12-bit pieces are packed into a single 32-bit word.

4.3 Conversion from signed 31-bit radix-2^{13} to radix-2^{32} modulo $2^M - 1$

The conversion from the resulting signed 31-bit radix-2^{13} representation of the product to its radix-2^{32} representation is combined with the reduction of the product modulo $2^M - 1$. Thus, with $P(X) = \sum_{i=0}^{190} p_i X^i$ as above, we compute the M-bit number $c = \sum_{j=0}^{38} c_j \cdot 2^{32j} \equiv P(2^{13}) \bmod (2^M - 1)$. We use the following precomputed constants:

- $C_1 \equiv -2^{31} \cdot \sum_{i=0}^{190} 2^{13i} \bmod (2^M - 1)$, $0 \le C_1 < 2^M - 1$.
- Integers k_i, l_i and m_i such that

$$13i = m_i M + 32 l_i + k_i \quad \text{with } 0 \le 32 l_i + k_i < M \text{ and } 0 \le k_i < 32,$$

  for $0 \le i < 191$. Note that $m_i \in \{0, 1, 2\}$ because $M > 827$.

Given these values, the conversion is done in four steps:

1. Compute $\tilde{p}_i = p_i + 2^{31}$, which can be done in parallel for $0 \le i < 191$. As a result $0 \le \tilde{p}_i < 2^{32}$ for $0 \le i < 191$. Note that $(\sum_{i=0}^{190} \tilde{p}_i \cdot 2^{13i}) + C_1 \equiv P(2^{13}) \bmod (2^M - 1)$.

2. Left shift $\tilde{p}_i$ over k_i bits and right shift $\tilde{p}_i$ over $32 - k_i$ bits, to obtain d_i, e_i such that

$$\tilde{p}_i \cdot 2^{13i} \equiv d_i \cdot 2^{32l_i} + e_i \cdot 2^{32(l_i+1)} \bmod (2^M - 1), \quad \text{for } 0 \leq i < 191.$$

   This can be done in parallel for $0 \leq i < 191$.
3. For $0 \leq j < 39$ compute

$$\tilde{c}_j = \sum_{i:l_i=j} d_i + \sum_{i:l_i+1=j} e_i \tag{1}$$

   while propagating carries to $\tilde{c}_{j+1}$. This can be done partially in parallel. Set $\tilde{c}_{39} = \sum_{i:l_i=38} e_i$ (including the carries from $\tilde{c}_{38}$). Note that reduction moduli $2^M - 1$ is effected by disregarding m_i and grouping together identical d_i-values and identical e_i-values. As a result, $\tilde{c} = \sum_{j=0}^{39} \tilde{c}_j \cdot 2^{32j}$ satisfies $\tilde{c} + C_1 \equiv c \bmod (2^M - 1)$.
4. Calculate $c \equiv \tilde{c} + C_1 \bmod (2^M - 1)$. Although the numbers are slightly bigger, this calculation is in principle the same as regular addition modulo $2^M - 1$.

4.4 Optimizations

Swapping even for odd instructions. Modular arithmetic mostly relies on the SPU's arithmetic instructions, which are even pipeline instructions. Following the approach from [24,10] 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 decreases the overall runtime. 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 i precomputed integers $\tilde{k}_i$, $\tilde{l}_i$, and $\tilde{m}_i$ such that

$$13i + 1 = \tilde{m}_i M + 32\tilde{l}_i + \tilde{k}_i \quad \text{with } 0 \leq 32\tilde{l}_i + \tilde{k}_i < M \text{ and } 0 \leq \tilde{k}_i < 32$$

instead of k_i, l_i, and m_i, as defined earlier. The ten remaining sums need to be doubled before they are added to the corresponding squared input coefficient. Each doubling can conveniently be done by a single even pipeline shift instruction. As suggested above, however, a doubling can be performed by four odd pipeline instructions. The ten remaining doublings could thus be squeezed in the odd pipeline, including all load and storage overheads[7]. As a result, all required doublings came for free.

[7] Interestingly, all 21 doublings would not have fit in the odd pipeline.

Table 1: SPU cycle counts for 4-SIMD multiplications and squarings modulo $2^{1193} - 1$

$a \cdot b \bmod (2^{1193} - 1)$					$a^2 \bmod (2^{1193} - 1)$		
instructions		cycles	calculation of		instructions		cycles
even	odd				even	odd	
708	722	752	$P_a(X)$ and $P_b(X)$	$P_a(X)$	354	361	376
3389	1137	3905	$Q^{(j)}$ for $1 \le j \le 27$		2107	2055	2130
1138	1078	1163	$P(X)$ and (d_i, e_i) for $0 \le i < 191$		1139	1086	1171
906	907	936	$\tilde{c}_j$ for $0 \le j < 39$ and c		900	905	931
6141	3844	6756	total		4500	4407	4608
		6971	measured				4814

Conversion to radix-2^{32}. The computation of d_i and e_i requires a shift by k_i and $32 - k_i$, respectively, for $0 \le i < 191$, for a total of 384 even pipeline shift instructions. If $k_i \equiv 0 \bmod 8$, each can be replaced by a single odd pipeline byte reordering instruction (or by no instruction if $k_i = 0$). Shift counts bigger than 8 can be replaced by three odd pipeline instructions.

***M*-dependent optimization.** The summation $\sum_{i:l_i+1=j} e_i$, for all j, from Eq. (1) does not exceed 2^{32} for most M since e_i is obtained by a right shift over $32 - k_i > 0$ bits and the shift amounts usually differ. Thus, for such M, these summations do not generate carries.

We have written a program that generates SPU code for each value of M, with the applicable C_0, C_1, k_i, l_i, m_i (and $\tilde{k}_i$, $\tilde{l}_i$, $\tilde{m}_i$) hard-coded and incorporating all optimizations mentioned. The resulting code and its performance thus depends on the value of M used, with a slight variation between different M-values. Representative instruction and cycle counts for 4-SIMD multiplication and squaring modulo $2^{1193} - 1$ on a single SPU are given in Table 1. Because $\frac{78}{144} \cdot 3905 \approx 2115$, the 2130 cycles required for the calculation of the $Q^{(j)}$'s while squaring is very close to what one would expect based on the 3905 cycles required while multiplying.

Overall speed. The figures in Table 1 count the number of cycles required per SPU for four simultaneous modular multiplications. Because six SPUs are available per PS3, a single PS3 can perform roughly 11 million multiplications modulo $2^M - 1$, or roughly 16 million modular squarings, per second. This may be compared to 209 million and 138 million multiplications modulo 192-bit and 224-bit special moduli, respectively, as reported for a single PS3 in [7], i.e., roughly a 13-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. At worst our numbers will be reduced to about 3.7 million and 5.3 million, respectively (but we expect that an actual implementation can be made to work substantially faster). This may be compared to roughly 102 million modular multiplications for generic moduli

Table 2: SPU effort for 4-SIMD phase one ECM trials for $2^{1193}-1$ with $B_1 = 3 \cdot 10^9$

operation	cycles per call	number of calls	time
multiplication	6971	26 193 284 192	15.89h
squaring	4814	13 358 576 558	5.60h
addition-subtraction	268	18 990 126 989	0.44h
addition	≈ 180	523 868 924	0.01h
subtraction	≈ 180	523 868 924	0.01h
		total	21.95h

in the 200-bit range, as reported for a single PS3 in [5], i.e., a 28-fold slowdown for 6-fold bigger generic moduli.

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

For a single phase one ECM trial with phase one bound $B_1 = 3 \cdot 10^9$ (cf. Section 2) GMP-ECM needs 6 155 419 355 additions and 523 868 924 doublings in an elliptic curve group. With the elliptic curve arithmetic used by GMP-ECM, an elliptic curve group addition requires four multiplications, two squarings and three addition-subtractions (i.e., $(c, d) = (a+b, a-b)$); a doubling requires three multiplications, two squarings, one addition-subtraction, one addition and one subtraction. All arithmetic is modulo $2^M - 1$. Table 2 lists the resulting total operation counts per phase one trial with $B_1 = 3 \cdot 10^9$ and, given the SPU's 3.192GHz clock speed, estimates the wall-clock time for an SPU to complete four phase one ECM trials. The measured wall-clock time was 22.03h, i.e., less than 5 minutes more than the estimate in Table 2. With more than 24 phase one ECM trials per day, a single PS3 is at least competitive with current quadcore desktops.

With six SPUs per PS3 and 215 PS3, we can process $4 \times 6 \times 215 = 5160$ phase one ECM trials in about 22 hours. With the number of trials from Section 2, phase one for a 65-digit search takes less than four and a half days and less than three weeks for a 70-digit search. 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 less than 4 and 18 days, respectively, on a 56 node cluster (with two hexcore processors per node) that we have access to: 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 about three months of sustained calculations for several M-values, four 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 $M = 1163$, and a 66-digit factor for $M = 1073$. The 241-bit, 73-digit prime factor of $2^{1181} - 1$ is the current ECM record. The factor was found after somewhat more than 30 000 phase one trials at approximately the 8 800th corresponding phase two trial, implying that we were quite lucky finding it. It was found for $\sigma = 4\,000\,027\,779$ (cf. [31]) with elliptic curve group order factoring into primes at most B_1 with the exception of one prime between B_1 and B_2:

$$2^4 \cdot 3^2 \cdot 13 \cdot 23 \cdot 61 \cdot 379 \cdot 13\,477 \cdot 272\,603 \cdot 12\,331\,747 \cdot 19\,481\,797 \cdot$$

$$125\,550\,349 \cdot 789\,142\,847 \cdot 1\,923\,401\,731 \cdot 10\,801\,302\,048\,203.$$

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 for $\sigma = 3\,000\,085\,158$ and group order

$$2^2 \cdot 3^2 \cdot 5 \cdot 23 \cdot 1\,429 \cdot 28\,229 \cdot 139\,133 \cdot 249\,677 \cdot 389\,749 \cdot 15\,487\,861 \cdot$$

$$47\,501\,591 \cdot 111\,707\,179 \cdot 431\,421\,191 \cdot 13\,007\,798\,103\,359.$$

So far our example number $2^{1193} - 1$ stubbornly resisted all ECM efforts to be factored. For the numbers $2^M - 1$ that we fail to factor using ECM, such as for $M = 1193$, our efforts will result in a reasonable degree of confidence that they will not have a prime factor of 60 digits or less. Although we hope, during our continuing efforts, 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. Table 3 lists the number of ECM trials performed with bounds as above and results obtained so far, with ck and pk denoting a k-digit composite and prime, respectively.

6 Conclusion

For integers M in the range from 1100 to 1400 we presented our PS3 implementation of multiplication of M-bit integers, processing 24 such multiplications in parallel on a single PS3, 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 focussed on application of our arithmetic to elliptic curve factoring, as a preparatory step for a potential (S)NFS factoring project. This led to the two largest factors found using ECM so far, beating the previous ECM record by 5 digits and, according to [12], finding 73-digit ECM factors almost two years ahead of schedule.

Acknowledgements. This work was supported by the Swiss National Science Foundation under grant numbers 200021-119776 and 206021-117409 and by EPFL DIT. Paul Zimmermann kindly provided us with the elliptic curve group orders.

Table 3: Targeted M-values and status of ECM trials (as of June 18, 2010).

M	targeted composite	completed number of trials		result
		phase one	phase two	
1007	c254	24 600	24 600	
1031	c299	24 600	24 600	
1073	c281	$\approx$ 25 000	1 460	p66 · p215
1081	c251	24 480	24 480	
1111	c278	24 434	24 434	
1117	c332	$\approx$ 25 000	0	
1123	c321	24 720	24 720	
1129	c327	24 732	24 732	
1135	c225	in progress		
1139	c313	25 056	25 056	
1147	c263	50 076	50 076	
1151	c242	in progress		
1153	c331	50 032	50 032	
1159	c309	50 032	50 032	
1163	c318	$\approx$ 50 000	47 768	p73 · p246
1171	c342	43 592	43 592	
1177	c255	$\approx$ 25 000	16 908	in progress
1181	c291	$\approx$ 30 000	8 808	p73 · p218
1187	c266	$\approx$ 15 000	7 871	p63 · p204
1193	c355	142 162	142 162	
1199	c314	50 141	50 141	

References

1. K. Aoki, J. Franke, T. Kleinjung, A. K. Lenstra, and D. A. Osvik. A kilobit special number field sieve factorization. In *Asiacrypt 2007*, volume 4833 of *LNCS*, pages 1–12, 2007.
2. D. V. Bailey, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, H.-C. Chen, C.-M. Cheng, G. van Damme, G. de Meulenaer, L. J. D. Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A. V. Herrewege, and B.-Y. Yang. Breaking ECC2K-130. Cryptology ePrint Archive, Report 2009/541, 2009. `http://eprint.iacr.org/`.
3. 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/`.
4. D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. EECM: ECM using Edwards curves. `http://eecm.cr.yp.to/`, 2010.
5. 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`.
6. 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.

7. J. W. Bos. High-performance modular multiplication on the Cell processor. In *WAIFI 2010*, volume 6087 of *LNCS*, pages 7–24, 2010.
8. 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`.
9. 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.
10. J. W. Bos and D. Stefan. Performance analysis of the SHA-3 candidates on exotic multi-core architectures. In *CHES 2010*, LNCS, 2010. To appear.
11. R. P. Brent. Large factors found by ECM. `http://wwwmaths.anu.edu.au/~brent/ftp/champs.txt`.
12. R. P. Brent. Recent progress and prospects for integer factorisation algorithms. In *COCOON*, volume 1858 of *LNCS*, pages 3–22, 2000.
13. S. Cavallar, B. Dodson, A. K. Lenstra, W. M. Lioen, P. L. Montgomery, B. Murphy, H. J. J. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. C. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of a 512-bit RSA modulus. In *Eurocrypt 2000*, volume 1807 of *LNCS*, pages 1–18, 2000.
14. R. Crandall and B. Fagin. Discrete weighted transforms and large-integer arithmetic. *Math. of Comp.*, 62(205):305–324, 1994.
15. 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.
16. 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.
17. G. Hotz. Here's your silver platter. `http://geohotps3.blogspot.com/2010/01/heres-your-silver-platter.html`.
18. T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thom'e, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. Factorization of a 768-bit RSA modulus. In *Crypto 2010*, LNCS, 2010. To appear.
19. D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
20. A. K. Lenstra. Unbelievable security: Matching AES security using public key systems. In *Asiacrypt 2001*, volume 2248 of *LNCS*, pages 67–86, 2001.
21. 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.
22. A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number. *Math. of Comp.*, 61(203):319–349, 1993.
23. H. W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649–673, 1987.
24. 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.
25. 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.

26. M. Simka, 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.
27. 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/`.
28. 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.
29. P. Zimmermann. 50 large factors found by ECM. `http://www.loria.fr/~zimmerma/records/top50.html`.
30. P. Zimmermann and B. Dodson. 20 years of ECM. In *ANTS*, volume 4076 of *LNCS*, pages 525–542, 2006.
31. P. Zimmermann et al. GMP-ECM (elliptic curve method for integer factorization). `https://gforge.inria.fr/projects/ecm/`, 2010.