HAL
open science

On Newton-Raphson iteration for multiplicative inverses modulo prime powers

Jean-Guillaume Dumas

To cite this version:

Jean-Guillaume Dumas. On Newton-Raphson iteration for multiplicative inverses modulo prime powers. IEEE Transactions on Computers, 2014, 63 (8), pp.2106-2109. 10.1109/TC.2013.94 . hal-00736701v5

HAL Id: hal-00736701

https://hal.science/hal-00736701v5

Submitted on 14 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On Newton-Raphson iteration for multiplicative inverses modulo prime powers

Jean-Guillaume Dumas*

May 14, 2018

Abstract

We study algorithms for the fast computation of modular inverses. Newton-Raphson iteration over p-adic numbers gives a recurrence relation computing modular inverse modulo p^m, that is logarithmic in m. We solve the recurrence to obtain an explicit formula for the inverse. Then we study different implementation variants of this iteration and show that our explicit formula is interesting for small exponent values but slower for large exponent, say of more than 700 bits. Overall we thus propose a hybrid combination of our explicit formula and the best asymptotic variants. This hybrid combination yields then a constant factor improvement, also for large exponents.

1 Introduction

The multiplicative inverse modulo a prime power is fundamental for the arithmetic of finite rings, for instance at the initialization phase of Montgomery's integer multiplication (see, e.g., [6, 1] and references therein). It is also used, e.g., to compute homology groups in algebraic topology for image pattern recognition [4], mainly to improve the running time of algorithms working modulo prime powers. Those can be used for the computation of the local Smith normal form [5, 7], for instance in the context of algebraic topology: there linear algebra modulo p^e can reveal torsion coefficients and inverses are required for pivoting in Gaussian elimination or minimal polynomial synthesis (see, e.g., [4, algorithm LRE] or [10]).

Classical algorithms to compute a modular inverse uses the extended Euclidean algorithm and Newton-Raphson iteration over p-adic fields, namely Hensel lifting [9]. Arazi and Qi in [1] lists also some variants adapted to the binary characteristic case that cut the result in lower and higher bits.

In the following, we give another proof of Arazi and Qi's logarithmic formula using Hensel lifting. Then we derive an explicit formula for the inverse that

*Université Grenoble Alpes; Laboratoire Jean Kuntzmann, (umr CNRS 5224); 700 avenue centrale, IMAG - CS 40700, F-38058 Grenoble, France. Jean-Guillaume.Dumas@univ-grenoble-alpes.fr

generalizes to any prime power. Finally, we study the respective performance of the different algorithms both asymptotically and in practice and introduce a hybrid algorithm combining the best approaches.

2 Hensel's lemma modulo p^m

For the sake of completeness, we first give here Hensel's lemma and its proof from Newton-Raphson's iteration (see e.g. [2, Theorem 7.7.1] or [3, §4.2] and references therein).

Lemma 1 (Hensel)**.** *Let p be a prime number, $m \in \mathbb{N}$, $f \in \mathbb{Z}[X]$ and $r \in Z$ such that $f(r) = 0 \mod p^m$. If $f'(r) \neq 0 \mod p^m$ and*

$$t = -\frac{f(r)}{p^m} f'(r)^{-1},$$

then $s = r + tp^m$ satisfies $f(s) = 0 \mod p^{2m}$.

Proof. Taylor expansion gives that $f(r+tp^m) = f(r)+tp^m f'(r)+O(p^{2m})$. Thus if $t = -\frac{f(r)}{p^m} f'(r)^{-1}$, the above equation becomes $f(s) = 0 \mod p^{2m}$. □

3 Inverse modulo 2^m

Now, in the spirit of [8], we apply this lemma to the inverse function

$$F_a(x) = \frac{1}{ax} - 1 \tag{1}$$

3.1 Arazi and Qi's formula

We denote by an under-script $_L$ (resp. $_H$) the lower (resp. higher) part in binary format for an integer. From Equation (1) and Lemma 1 modulo 2^i, if $r = a^{-1} \mod 2^i$, then we immediately get

$$t = -\frac{\frac{1}{ax} - 1}{2^i} \left(-\frac{1}{ax^2}\right)^{-1}.$$

In other words $t = \frac{1-ar}{2^i} r \mod 2^i$. Now let $a = b + 2^i a_H \mod 2^{2i}$ so that we also have $r = b^{-1} \mod 2^i$ and hence $rb = 1 + 2^i\alpha$ with $0 \leq \alpha < 2^i$. Thus $ar = br + 2^i r a_H = 1 + 2^i(\alpha + r a_H)$ which shows that

$$t = -(\alpha + r a_H) r \equiv -\left((rb)_H + (r a_H)_L\right) r \mod 2^i \tag{2}$$

The latter is exactly [1, Theorem 1] and yields the following Algorithm 1, where the lower and higher parts of integers are obtained via masking and shifting.

Lemma 2. *Algorithm 1 requires $13\lfloor \log_2(m) \rfloor + 1$ arithmetic operations.*

Algorithm 1 Arazi&Qi Quadratic Modular inverse modulo 2^m

Input: $a \in \mathbb{Z}$ odd and $m \in N$.
Output: $U \equiv a^{-1} \mod 2^m$.

1: $U = 1$;
2: **for** $(i = 1;\ i < m;\ i <<= 1)$ **do**
3: $\quad b = a \ \&\ (2^i - 1)$; $\{b = a \mod 2^i\}$
4: $\quad t_1 = U * b;\ t_1 >>= i$; $\{(rb)_H\}$
5: $\quad c = (a >> i) \ \&\ (2^i - 1)$; $\{a_H\}$
6: $\quad t_2 = (U * c) \ \&\ (2^i - 1)$; $\{(ra_H)_L\}$
7: $\quad t_1 += t_2$;
8: $\quad t_1 *= U;\ t_1 \ \&= (2^i - 1)$; $\{-t\}$
9: $\quad t_1 = 2^i - t_1$; $\{t\}$
10: $\quad t_1 <<= i$; $\{t2^i\}$
11: $\quad U \mid= t_1$; $\{r + t2^i\}$
12: **end for**
13: $U \ \&= (2^m - 1)$; $\{r \mod 2^m\}$
14: **return** U;

3.2 Recurrence formula

Another view of Newton-Raphson's iteration is to create a recurrence. Equation (1) gives

$$\begin{aligned} U_{n+1} &= U_n - \frac{\frac{1}{aU_n} - 1}{-\frac{1}{aU_n^2}} = U_n - (aU_n - 1)U_n \\ &= U_n(2 - aU_n) \end{aligned} \tag{3}$$

This yields the loop of Algorithm 2, for the computation of the inverse, see e.g. [9] or [3, §2.4].

Lemma 3. *Algorithm 2 is correct and requires* $6\lceil \log_2(m) \rceil + 2$ *arithmetic operations.*

Proof. The proof of correctness is natural in view of the Hensel lifting. First $U_0 = a^{-1} \mod p$. Second, by induction, suppose $a \cdot U_n \equiv 1 \mod p^k$. Then $aU_n = 1 + \lambda p^k$ and $aU_{n+1} = aU_n(2 - aU_n) = (1 + \lambda p^k)(2 - 1 - \lambda p^k) = (1 - \lambda^2 p^{2k} \equiv 1 \mod p^{2k})$. Finally $U_n \equiv a^{-1} \mod p^{2^n}$. □

Remark 1. *We present this algorithm for computations modulo* p^m *but its optimization modulo a power of* 2 *is straightforward: replace the modular operations of for instance lines 5, 11 etc. by a binary masking:* $x \ \&= (2^i - 1)$.

Remark 2. *It is important to use a squaring in line 3. Indeed squaring can be faster than multiplication, in particular in the arbitrary precision setting [12]. In the case of Algorithm 2, the improvement over an algorithm of the form* $temp = 2 - a * U; temp\% = p^m; U* = temp; U\% = p^m;$ *is of about* 30%.

Algorithm 2 Hensel Quadratic Modular inverse

Input: $p \in \mathbb{Z}$ a prime, $a \in \mathbb{Z}$ coprime to p, and $m \in N$.
Output: $U \equiv a^{-1} \mod p^m$.

$U = a^{-1} \mod p$; $\{\text{extended gcd}\}$
for $(i = 2;\ i < m;\ i <<= 1)$ **do**
  $temp = U * U$; $\{U_n^2\}$
  $temp *= a$; $\{aU_n^2\}$
  $temp \,\%= p^i$; $\{temp \mod p^i\}$
  $U <<= 1$; $\{2U_n\}$
  $U -= temp$; $\{U_n(2 - aU_n)\}$
end for
$temp = U * U$; $\{U_n^2\}$
$temp *= a$; $\{aU_n^2\}$
$temp \,\%= p^m$; $\{temp \mod p^m\}$
$U <<= 1$; $\{2U_n\}$
$U -= temp$; $\{U_n(2 - aU_n)\}$
$U \,\%= p^m$; $\{U \mod p^m\}$
return U;

Remark 3. *Note that for Algorithms 1 and 2, a large part of the computation occur during the last iteration of the loop when 2^i is closest to 2^m. Therefore, a recursive version cutting in halves will be more efficient in practice since the latter will be exactly done at $i = m/2$ instead of at the largest power of 2 lower than m. Moreover this improvement will take place at each recursion level. We thus give in the following the recursive version for Formula (3), the one for a recursive version of Arazi&Qi is in the same spirit.*

Algorithm 2' Recursive Hensel

Input: $p \in \mathbb{Z}$ a prime, $a \in \mathbb{Z}$ coprime to p, and $m \in N$.
Output: $r \equiv a^{-1} \mod p^m$.

if $m == 1$ **then return** $a^{-1} \mod p$; **end if** $\{\text{ext. gcd}\}$
$h = \lceil \frac{m}{2} \rceil$
$b = a \,\% p^h$; $\{b = a \mod p^h\}$
$r =$RecursiveHensel(p, b, h);
$temp = r * r$; $\{r^2\}$
$temp *= a$; $\{ar^2\}$
$temp \,\%= p^m$; $\{temp \mod p^m\}$
$r <<= 1$; $\{2r\}$
$r -= temp$; $\{r(2 - ar)\}$
$r \,\%= p^m$; $\{r \mod p^m\}$
return r;

3.3 Factorized formula

We now give an explicit formula for the inverse by solving the preceding recurrence relation, first in even characteristic.

We denote by $H_n = aU_n$ a new sequence, that satisfies $H_{n+1} = H_n(2 - H_n)$. With $H_0 = a$ we get $H_1 = a(2-a) = 2a - a^2 = 1 - (a-1)^{2^1}$, by induction, supposing that $H_n = 1 - (a-1)^{2^i}$, we get

$$\begin{aligned} H_{n+1} &= \left(1 - (a-1)^{2^n}\right)\left(2 - 1 + (a-1)^{2^n}\right) \\ &= 1^2 - \left((a-1)^{2^n}\right)^2 = 1 - (a-1)^{2^{n+1}} \end{aligned}$$

Using the remarkable identity, this in turn yields:

$$H_n = a(2-a)\prod_{i=1}^{n-1}\left(1 + (a-1)^{2^i}\right);$$

therefore, with $U_0 = 1$ and $U_1 = 2 - a$ we have that

$$U_n = (2-a)\prod_{i=1}^{n-1}\left(1 + (a-1)^{2^i}\right) \tag{4}$$

The latter equation gives immediately rise to the following Algorithm 3.

Algorithm 3 Explicit Quadratic Modular inverse modulo 2^m

Input: $a \in \mathbb{Z}$ odd and $m \in N$.
Output: $U \equiv a^{-1} \mod 2^m$.

```
Let s and t be such that t is odd and a = 2^s t + 1;
U = 2 - a;
amone = a - 1;
for (i = 1; i < m/s; i <<= 1) do
    amone *= amone;                          {square: (a-1)^{2^i}}
    amone &= (2^m - 1);                      {(a-1)^{2^i} mod 2^m}
    U *= (amone + 1);
    U &= (2^m - 1);                          {U mod 2^m}
end for
return U;
```

Lemma 4. *Algorithm 3 is correct and requires* $5\lfloor \log_2(\frac{m}{s})\rfloor + 2$ *arithmetic operations.*

Proof. Modulo 2^m, a is invertible if and only if a is odd, so that $a = 2^s t + 1$ and therefore, using Formula (4), we get $aU_n = H_n = 1 - (a-1)^{2^n} = 1 - (2^s t)^{2^n} \equiv 1 \mod 2^{s2^n}$. Thus, $U_{\lceil \log_2(\frac{m}{s})\rceil} \mod 2^m \equiv a^{-1} \mod 2^m$. □

There are two major points to remark with this variant:

1. It performs fewer operations than previous algorithms.
2. It must compute with the full p-adic development (modulo operations are made modulo 2^m and not 2^i.

Therefore we will see that this algorithm has a worse asymptotic complexity but is very efficient in practice for small exponents.

3.4 Generalization modulo any prime power

The formula generalizes directly for any prime power:

Theorem 1. *Let p be a prime number, a coprime to p and $b = a^{-1} \mod p$ is the inverse of a modulo p. Let also V_n be the following sequence:*

$$\begin{cases} V_0 & = b \equiv a^{-1} \mod p, \\ V_n & = b(2-ab)\prod_{i=1}^{n-1}\left(1+(ab-1)^{2^i}\right) \end{cases} \tag{5}$$

Then $V_n \equiv a^{-1} \mod p^{2^n}$.

Proof. The proof is similar to that of Lemma 3 and follows also from Hensel's lemma. From the analogue of Equation (4), we have $a \cdot V_n = 1-(ab-1)^{2^n}$. Now as $a \cdot b = 1+\lambda p$, by the definition of b we have $a \cdot V_n = 1-(ab-1)^{2^n} = 1-(\lambda p)^{2^n} \equiv 1 \mod p^{2^n}$. □

4 Complexity analysis over arbitrary precision

We provide here the equivalents of the complexity results of the previous section but now for arbitrary precision: the associated binary complexity bounds for the different algorithms are given here with classical arithmetic operations on integer (i.e. without fast variants like Karatsuba or DFT). We thus now suppose that masking and shifting as well as addition are linear and that multiplication is quadratic ($\mathcal{O}\left(2m^2\right)$ operations to multiply to elements of size m).

Lemma 5. *Using classical arithmetic, Algorithm 1 requires*

$$\mathcal{O}\left(2m^2+10m\right) \quad \textit{binary operations.}$$

Proof. Following the algorithm, the complexity bound becomes

$$\mathcal{O}\left(m+\sum_{j=1}^{\log_2(m)-1} 3\cdot 2(2^j)^2+1(2^j)\right) = \mathcal{O}\left(2m^2+10m\right).$$

□

Lemma 6. *Using classical arithmetic in even characteristic modulo 2^m, Algorithm 2 requires*

$$\mathcal{O}\left(\frac{16}{3}m^2 + 9m\right) \quad \textit{binary operations.}$$

Proof. Following the algorithm, the complexity bound becomes

$$\mathcal{O}\left(2 \cdot 2m^2 + 5m + \sum_{j=2}^{\log_2(m)-1} 2 \cdot 2(2^j)^2 + 4(2^j)\right) = \mathcal{O}\left(\frac{16}{3}m^2 + 9m\right).$$

□

Lemma 7. *Using classical arithmetic, Algorithm 3 requires*

$$\mathcal{O}\left(\left(4m^2 + 2m\right)\lfloor \log_2(m) \rfloor\right) \quad \textit{binary operations.}$$

Proof. Similarly, here we have

$$\mathcal{O}\left(\sum_{j=1}^{\log_2(m)-1} 2 \cdot 2m^2 + 2m\right) = \mathcal{O}\left(\left(4m^2 + 2m\right)\log_2(m)\right).$$

□

5 Experimental comparisons

The point of the classical Newton-Raphson algorithms (as well as Arazi and Qi's variant) is that it works with modular computations of increasing sizes, whereas the explicit formula requires to work modulo the highest size from the beginning. On the one hand we show next that this gives an asymptotic advantage the recurring relations. On the other hand, in practice, the explicit formula enables much faster performance for say cryptographic sizes. All experiments have been done on an Intel Xeon W3530, 2.8 GHz, running linux debian[1]

5.1 Over word-size integers

Using word-size integers, the many masking and shifting required by recurring relations do penalize the performance, where the simpler Algorithm 3 is on average 26% faster on a standard desktop PC, as shown on Figure 1. Differently, Arazi and Qi's variant suffers from the manipulations required to extract the low and high parts of integers.

[1]The source code for the experiments is available on http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Software/InvModTwoK. It uses GMP version 5.0.5 (http://gmplib.org), with givaro-3.7.2 C++ wrappers (http://givaro.forge.imag.fr)

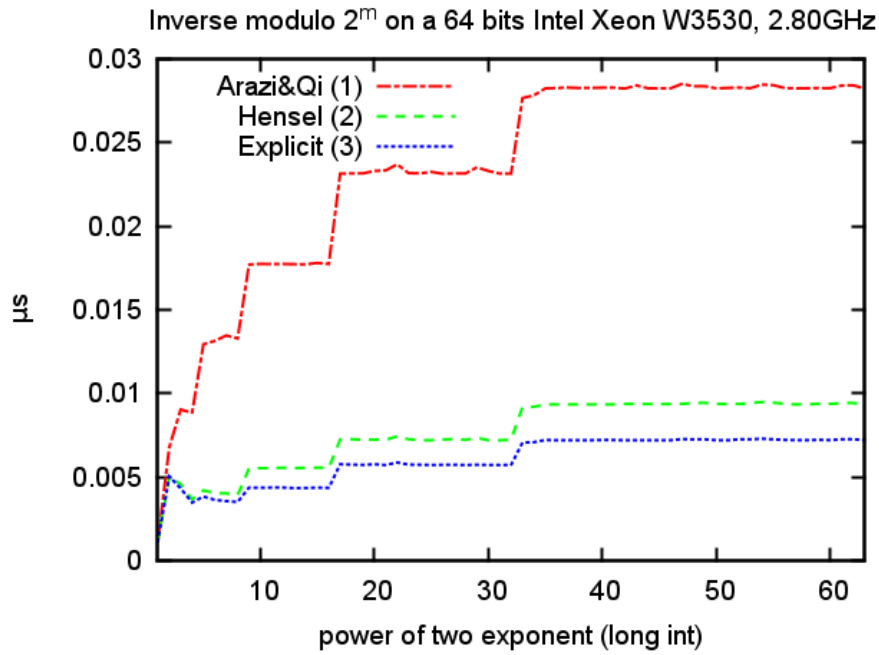

Figure 1: Modular inverse on 64 bits machine words

5.2 Over arbitrary precision arithmetic

From Lemma 7, we see that the explicit formula adds a logarithmic factor, asymptotically. In practice, Figure 2 shows that using GMP[1], the asymptotic behavior of Algorithm 1 becomes predominant only for integers with more than 1200 bits. For the Newton-Raphson iteration the asymptotic behavior of Algorithm 2 becomes predominant even sooner, for integers with about 640 bits. Below that size, Algorithm 3 is better.

Remark 4. *Now, in [9, 8], the recurrence relation from (3), is extended*[2] *to* $X_{n+1} = \frac{1-(1-aX_n)^r}{a}$ *for a fixed* r*. This allows a faster convergence, by a factor of* $\log_2(r)$*.*

Unfortunately the price to pay is to compute a r*-th power at each iteration (instead of a single square), which could be done, say by recursive binary squaring, but at a price of* $\log_2(r)$ *squarings and between* 0 *and* $\log_2(r)$ *multiplications. Overall there would be no improvement in the running time.*

[2]this is to be compared with explicit Formula (4), $V_{n+1} = V_n(1 + (1 - ab)^{2^n})$, where the computation is done with the first inverse *modp* (recall that $b \equiv a^{-1} \mod p$), where in the classical setting the computation is done with the inverse so far: X_n

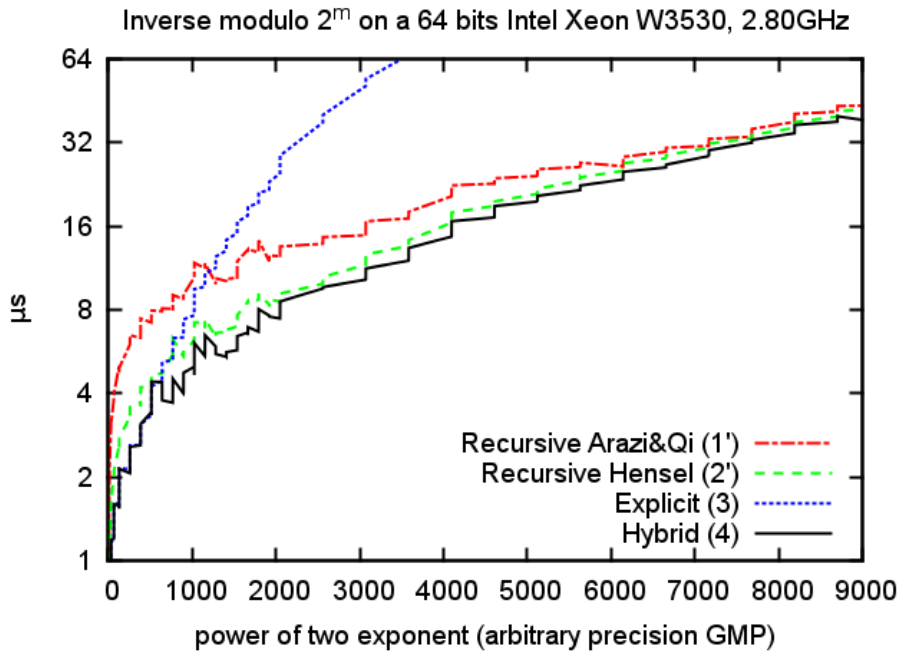

Figure 2: Modular inverse on arbitrary precision integers

5.3 Hybrid algorithm

With the thresholds of Figure 2 and from the previous algorithms, we can then use a classical hybrid strategy which is better than all of them everywhere: for small exponents it uses the explicit formula of Algorithm 3; then for larger exponents:

1. it starts to compute the inverse recursively at half the initial exponent;
2. then, to lift the inverse modulo the double exponent, it uses the classical Hensel formula of Equation (3), and switches to Arazi&Qi formula of Equation (2), only for exponents larger than 9000 bits.

Actually, the lift switches back to Hensel formula after 10^6 bits: indeed on the used computer quasi linear multiplication via FFT comes into play in GMP and the analysis of Section 4 is not relevant anymore. The obtained algorithm is on average 21% times faster than any other direct lifting alone as shown with the curve (4) of Figure 2 (recall that on Figure 2 ordinates are presented in a logarithmic scale) and also on the ratios of Figure 3.

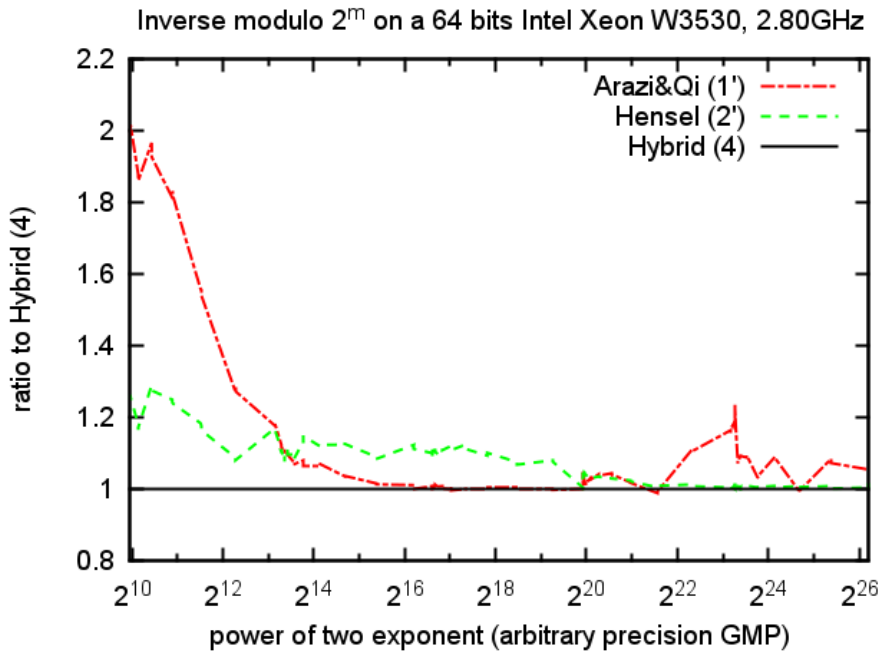

Figure 3: Ratios of modular inverse lifting algorithms over the hybrid method

6 Conclusion

We have studied different variants of Newton-Raphson's iteration over p-adic numbers to compute the inverse modulo a prime power. We have shown that a new explicit formula can be up 26% times faster in practice than the recursive variants for small exponents. Asymptotically, though, the latter formula suffers from a supplementary logarithmicfactor in the power (or a doubly logarithmic factor in the prime power) that makes it slower for large arbitrary precision integers. However, using each one of the best two algorithms in their respective regions of efficiency, we were able to use a hybrid strategy with improved performance of 21% on average at any precision.

More studies are to be made for the respective behavior of the algorithms in odd characteristic. Indeed there bit masking is replaced by extended euclidean algorithms variants and their respective performance could be different.

Acknowledgment

Many thanks to Christoph Walther, who found two typographic mistakes in the published version of Algorithm 2' (2^h where p^h was correct, line 3 and p^h where p^m was correct, line 7) [11].

References

[1] O. Arazi and Hairong Qi. On calculating multiplicative inverses modulo 2^m. *IEEE Transactions on Computers*, 57(10):1435–1438, October 2008.

[2] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory: Efficient Algorithms*. MIT press, 1996.

[3] Richard P. Brent and Paul Zimmermann. *Modern computer arithmetic*, volume 18 of *Cambridge monographs on applied and computational mathematics*. Cambridge University Press, Cambridge, UK, 2011.

[4] Jean-Guillaume Dumas, Frank Heckenbach, B. David Saunders, and Volkmar Welker. Computing simplicial homology based on efficient Smith normal form algorithms. In Michael Joswig and Nobuki Takayama, editors, *Algebra, Geometry and Software Systems*, pages 177–206. Springer, 2003.

[5] Jean-Guillaume Dumas, B. David Saunders, and Gilles Villard. On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computation*, 32(1/2):71–99, July–August 2001.

[6] Stephen R. Dussé and Burton S. Kaliski Jr. A cryptographic library for the Motorola DSP56000. In *EUROCRYPT '90, Denmark, May 21-24, 1990, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 230–244, 1990.

[7] Mustafa Elsheikh, Mark Giesbrecht, Andy Novocin, and B. David Saunders. Fast computation of Smith forms of sparse matrices over local rings. In Joris van der Hoeven and Mark van Hoeij, editors, *ISSAC'2012, Proceedings of the 2012 ACM International Symposium on Symbolic and Algebraic Computation, Grenoble, France*, pages 146–153, July 2012.

[8] Michael Knapp and Christos Xenophontos. Numerical analysis meets number theory: using root finding methods to calculate inverses mod p^n. *Applicable Analysis and Discrete Mathematics*, 4(1):23–31, 2010.

[9] E.V. Krishnamurthy and Venu K. Murthy. Fast iterative division of p-adic numbers. *IEEE Transactions on Computers*, C-32(4):396–398, April 1983.

[10] James A. Reeds and Neil J. A. Sloane. Shift-register synthesis (modulo m). *SIAM Journal on Computing*, 14(3):505–513, August 1985.

[11] Christoph Walther. Formally verified Montgomery multiplication. In *30th International Conference on Computer Aided Verification (CAV'18), Oxford, UK*, 2018.

[12] Dan Zuras. More on squaring and multiplying large integers. *IEEE Transactions on Computers*, 43(8):899–908, August 1994.