Q-adic Floating-point Transform revisited: arithmetic over small extension field via floating point routines

Jean-Guillaume Dumas

Q-adic Floating-point Transform revisited: arithmetic over small extension field via floating point routines

Jean-Guillaume Dumas *

October 2, 2007

Abstract

We present an algorithm to perform arithmetic operations over small extension field via numerical routines. The idea is to convert the X-adic representation of modular polynomials, with X an indeterminate, to a q-adic representation where q is a prime power larger than the field characteristic. With some control on the different involved sizes it is then possible to perform some of the q-adic arithmetic directly with floating point operators. Depending also on the number of performed numerical operations one can then convert back to the q-adic or X-adic representation and eventually mod out high residues. In this note we present a new version of both conversions: more tabulations and a way to reduce the number of divisions involved in the process are presented.

1 Introduction

The FFLAS/FFPACK project has demonstrated the need of a wrapping of the numerical routines for efficient small finite field linear algebra [2, 3].

A conversion between a modular representation of prime fields and floating points used exactly is natural. It uses the homomorphism to the integers. Now for extension fields (isomorphic to polynomials over a prime field) such a conversion is not direct. In [2] we propose to transform the polynomials into a q-adic representation where q is a prime power larger than the field characteristic. We call this transformation QFT for Q-adic Floating-point Transform. With some care, in particular on the size of q, it is possible to map the operations in the extension field into the floating point arithmetic realization of this q-adic representation and convert back using an inverse QFT.

In this note we still use this scheme but propose some implantation improvements: we propose to use a tabulated Zech logarithm for the QFT and give a

*Laboratoire J. Kuntzmann, Université de Grenoble. UMR CNRS 5224, BP 53X, F38041 Grenoble, France, Jean-Guillaume.Dumas@imag.fr

trick to reduce the number of machine divisions involved in the inverse. Therefore we recall in section 2 the previous conversion algorithm. We then present our new ideas in section 3 and end the presentation by a comparison and an application to fast matrix multiplication.

2 A first algorithm

We want now to use our ATLAS based implementation with non-prime fields. The requirements are to be able to produce a coherent representation of $\mathtt{GF}(p^k)$ with `double` precision numbers.

2.1 A $q-$adic representation

We follow here the presentation of [2] of the idea of [5]: we go back to the polynomial arithmetic but in a $q-$adic way, with q a sufficiently big prime or power of a single prime.

Suppose that $a = \sum_{i=0}^{k-1} \alpha_i X^i$ and $b = \sum_{i=0}^{k-1} \beta_i X^i$ are two elements of $\mathtt{GF}(p^k)$ represented by $\mathbb{Z}/\mathrm{p}\mathbb{Z}[X]/Q$. One can perform the polynomial multiplication ab via $q-$adic numbers. Indeed, by setting $\tilde{a} = \sum_{i=0}^{k-1} \alpha_i q^i$ and $\tilde{b} = \sum_{i=0}^{k-1} \beta_i q^i$, the product is computed in the following manner (we suppose that $\alpha_i = \beta_i = 0$ for $i > k-1$):

$$\widetilde{ab} = \sum_{j=0}^{2k-2} \left(\sum_{i=0}^{j} \alpha_i \beta_{j-i} \right) q^j \tag{1}$$

Now if q is large enough, the coefficient of q^i will not exceed q. In this case, it is possible to evaluate a and b as floating point numbers, compute the product of these evaluations, and convert back to finite field element, via a $q-$adic reconstruction, divisions by p and a division by Q. Several floating point operations can even be accumulated before performing the inverse transformation. Depending on the size of q the results can still remain exact.

Theorem 1. *[2] Let m be the number of available mantissa bits within the machine floating point numbers and n be the number of floating points products $v_1.v_2$ accumulated, e.g. in a dot product, before the re-conversion. If*

$$q > nk(p-1)^2 \text{ and } (2k-1)\log[2](q) < m,$$

then Algorithm 2.1 is correct.

It is shown on [2, Figures 5 & 6] that this wrapping is already a pretty good way to obtain high speed linear algebra over some small extension fields. This requires the following optimizations.

2.2 Implementation optimizations

For the performance, a first naïve implementation would only give limited speed-up as the conversion cost is then very expensive. However, some simple optimizations were proposed in [2]:

Algorithm 2.1 Dot product over Galois fields via $q-$adic conversions to floating point numbers

Require: a field $\mathtt{GF}(p^k)$ represented as polynomials mod p and mod Q, for Q a degree k irreducible polynomial over $\mathbb{Z}/\mathrm{p}\mathbb{Z}$.
Require: Two elements v_1 and v_2 of $\mathtt{GF}(p^k)$ each, as polynomials.
Require: a prime power q.
Ensure: $R \in \mathtt{GF}(p^k)$, with $R = v_1.v_2$.

Polynomial to $q-$adic conversion

1: Set $\widetilde{v_1}$ and $\widetilde{v_2}$ to the floating point vectors of the evaluations at q of the elements of v_1 and v_2. {Using e.g. Horner's formula}

One computation

2: Compute $\tilde{r} = \widetilde{v_1}\widetilde{v_2}$

Building the solution

3: $\tilde{r} = \sum_{i=0}^{2k-2} \widetilde{\mu_i} q^i$. {Using radix conversion, see e.g. [4, Algorithm 9.14]}
4: For each i, set $\mu_i = \widetilde{\mu_i} \mod p$
5: set $R = \sum_{i=0}^{2k-2} \mu_i X^i \mod Q$

1. The prime power q can be chosen to be a power of 2. Then the Horner like evaluation of the polynomials at q (line 1 of algorithm 2.1) is just a left shift. One can then compute this shift with exponent manipulations in floating point arithmetic and use then native C++ << operator as soon as values are within the 32 bits range, or use the native C++ << on 64 bits when available.

2. Some *sparse* primitive polynomials modulo p can be chosen to build $\mathtt{GF}(p^k)$. Then the division (line 5 of algorithm 2.1) can be simplified. The idea is to consider primitive tri or penta-nomials. In this case, we proved that less than $5k$ field operations were required to perform the final reconstruction.

With those optimization we were able to reach high peak performance, quite close to those obtained with prime fields, namely 420 Mop/s on a PIII, 735 MHz, and more than 500 Mop/s on a 64-bit DEC alpha 500 MHz. This is roughly 20 percent below the pure floating point performance and 15 percent below the prime field implementation.

3 Improvements

In this paper we propose to improve all the conversion steps of the previous algorithm in order to approach the performance of the prime field wrapping also for several extension fields:

1. Replace the Horner evaluation of the polynomials, to form the q-adic expansion, by a single table lookup recovering directly the floating point representation.

2. Replace the radix conversion by a single floating point division and some "à la Montgomery" reductions.

3. Replace the polynomial division by a single field operation.

Suppose the internal representation of the extension field is already by Zech logarithms and uses conversion tables from polynomial to index representations. See e.g. [1] for more details. Then the overall memory required by these new tables only doubles.

More precisely, we propose the following algorithm 3.1:

Algorithm 3.1 Fast Dot product over Galois fields via QFT and QFT inverse

---

Require: a field $\mathtt{GF}(p^k)$ with elements represented as exponents of a generator of the field.
Require: Two elements v_1 and v_2 of $\mathtt{GF}(p^k)$.
Require: a prime power q.
Ensure: $R \in \mathtt{GF}(p^k)$, with $R = v_1.v_2$.

Tabulated $q-$adic conversion

{Use conversion tables from exponent to floating point evaluation}

1: Set $\widetilde{v_1}$ and $\widetilde{v_2}$ to the floating point vectors of the evaluations at q of the elements of v_1 and v_2.

The floating point computation

2: Compute $\tilde{r} = \widetilde{v_1}\widetilde{v_2}$;

Computing a radix decomposition

3: $r = \lfloor \tilde{r} \rfloor$;
4: $rop = \left\lfloor \frac{\tilde{r}}{p} \right\rfloor$;
5: **for** $i = 0$ to $2k-2$ **do**
6: $\quad u_i = \left\lfloor \frac{r}{q^i} \right\rfloor - p \left\lfloor \frac{rop}{q^i} \right\rfloor$;
7: **end for**

Tabulated radix conversion to exponents of the generator

{μ_i is such that $\mu_i = \widetilde{\mu_i} \mod p$ for $\tilde{r} = \sum_{i=0}^{2k-2} \widetilde{\mu_i} q^i$}

8: Set $L = representation(\sum_{i=0}^{k-2} \mu_i X^i)$.
9: Set $H = representation(X^{k-1} \times \sum_{i=k-1}^{2k-2} \mu_i X^{i-k+1})$.

Reduction in the field

10: Return $R = H + L \in \mathtt{GF}(p^k)$;

We prove next that in algorithm 3.1, u_i satisfies $u_i = \sum_{j=i}^{2k-2} \mu_j q^{j-i} \mod p$. Therefore the representations of $\sum \mu_i X^i$ in the field can be precomputed and stored in a table where the indexing will be made by $(u_0, \ldots, u_{k-1})$ and $(u_{k-1}, \ldots, u_{2k-2})$ and not by the μ_i's[1]. Note also that the representation of X^{k-1} can be just

[1] In practice, indexing by a t-uple of integers mod p is made by evaluating at p, as $\sum u_i p^i$

$k-1$ if the irreducible polynomial used to build $\mathtt{GF}(p^k)$ is primitive and X has been chosen as the generator. Finally note that when q is a power of 2 division by q^i and flooring is just a right shift.

Theorem 2. *Algorithm 3.1 is correct.*

We first need the following lemma:

Lemma 1. *For r, a, $b \in \mathbb{N}$,*

$$\left\lfloor \frac{\lfloor \frac{r}{b} \rfloor}{a} \right\rfloor = \left\lfloor \frac{\lfloor \frac{r}{a} \rfloor}{b} \right\rfloor$$

Proof. We proceed by splitting the possible values of r into intervals $kab \leq r < (k+1)ab$. Then $kb \leq \frac{r}{a} < (k+1)b$ and since kb is an integer we also have that $kb \leq \lfloor \frac{r}{a} \rfloor < (k+1)b$. Thus $k \leq \frac{\lfloor \frac{r}{a} \rfloor}{b} < k+1$ and $\left\lfloor \frac{\lfloor \frac{r}{a} \rfloor}{b} \right\rfloor = k$. Obviously the same is true for the left hand side which proves the lemma. □

Proof *of theorem 2.* First we need to prove that $0 \leq u_i < p$. By definition of the truncation, we have $\frac{r}{q^i} - 1 < \left\lfloor \frac{r}{q^i} \right\rfloor \leq \frac{r}{q^i}$ and $\frac{r}{pq^i} - 1 - \frac{1}{q^i} < \left\lfloor \frac{rop}{q^i} \right\rfloor \leq \frac{r}{pq^i}$. Thus $-1 < u_i < p + \frac{p}{q^i}$, which is $0 \leq u_i \leq p$ since u_i is an integer. We now consider the possible case $u_i = p$ and show that it does not happen. $u_i = p$ means that $\left\lfloor \frac{r}{q^i} \right\rfloor = p(1 + \left\lfloor \frac{rop}{q^i} \right\rfloor) = pg$. This means that $r < pgq^i + q^i$. So that in turns $rop \leq \frac{r}{p} < gq^i + \frac{q^i}{p}$. Thus $\frac{rop}{q^i} < g + \frac{1}{p}$ so that $\left\lfloor \frac{rop}{q^i} \right\rfloor = g$. but then from the definition of g we have that $g = g - 1$ which is absurd. Therefore $0 \leq u_i \leq p-1$.

Second we show that $u_i = \sum_{j=i}^{2k-2} \mu_j q^{j-i} \mod p$. Well $u_i = \left\lfloor \frac{r}{q^i} \right\rfloor - p \left\lfloor \frac{\lfloor \frac{\tilde{r}}{p} \rfloor}{q^i} \right\rfloor$ and thus lemma 1 gives that $u_i = \left\lfloor \frac{r}{q^i} \right\rfloor - p \left\lfloor \frac{\lfloor \frac{\tilde{r}}{q^i} \rfloor}{p} \right\rfloor$. The latter is $u_i = \left\lfloor \frac{\tilde{r}}{q^i} \right\rfloor \mod p$. Now, since $\tilde{r} = \sum_{j=0}^{2k-2} \widetilde{\mu_j} q^j$, we have that $\left\lfloor \frac{\tilde{r}}{q^i} \right\rfloor = \sum_{j=i}^{2k-2} \widetilde{\mu_j} q^{j-i}$. Therefore, as $\mu_j = \widetilde{\mu_j} \mod p$, the equality is proven.

There remains to prove that it is possible to compute L and H from the u_i. From the equality above, we see that $\mu_{2k-2} = u_{2k-2}$ and $\mu_i = u_i - qu_{i+1} \mod p$, for $i = 0..(2k-3)$. Therefore a p^k elements precomputed table, indexed by $(u_0, \ldots, u_{k-1})$, can provide the representation of

$$L = \sum_{i=0}^{k-2} (u_i - qu_{i+1} \mod p) X^i.$$

Another table with p^k elements, indexed by $(u_{k-1}, \ldots, u_{2k-2})$, can provide the representation of

$$H = u_{2k-2} X^{2k-2} + \sum_{i=k-1}^{2k-3} (u_i - qu_{i+1} \mod p) X^i.$$

Finally $R = X^{k-1} \times \sum_{i=k-1}^{2k-2} \mu_i X^{i-k+1} + \sum_{i=0}^{k-2} \mu_i X^i$ needs to be reduced modulo the irreducible polynomial used to build the field. But, if we are given the representations of H and L in the field, R is then equal to their addition inside the field, directly using the internal representations. □

Table 1 gives the respective complexities of the two presented algorithms

	Shift & axpy	Access	Division	Reduction	Memory
Algorithm 2.1	$4k-2$	0	$2k-1$	$5k$	$3p^k$
Algorithm 3.1	$4k-2$	4	1	3	$6p^k$

Table 1: Complexity of the back and forth conversion between extension field and floating point numbers

4 Conclusion

Figure 1 shows only the speed of the conversion after the floating point operations. The log scales prove that for q ranging from 2^1 to 2^{26} (on a 32 bit Pentium IV) our new implantation is two to three times faster than the previous one.

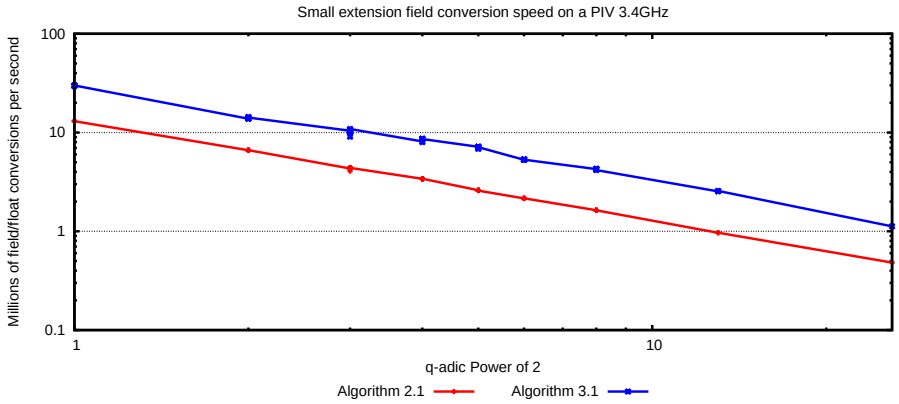

Figure 1: Small extension field conversion speed on a PIV 3.4GHz

Furthermore, these improvements e.g. allow the extension field routines to reach the speed of 7500 millions of GF(9) operations per second (on a XEON, 3.6 GHz) as shown on figure 2. This represent a reduction from the 15 percent overhead of the previous implementation to less than 6 percent now, when compared to GF(11).

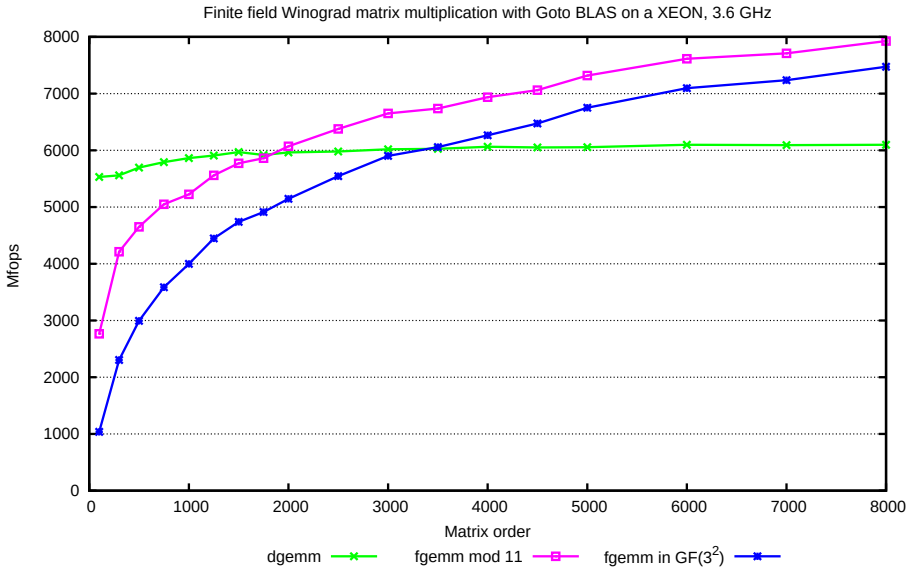

Figure 2: Finite field Winograd matrix multiplication with Goto BLAS on a XEON, 3.6 GHz

References

[1] Jean-Guillaume Dumas. Efficient dot product over finite fields. In Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *Proceedings of the seventh International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*, pages 139–154. Technische Universität München, Germany, July 2004.

[2] Jean-Guillaume Dumas, Thierry Gautier, and Clément Pernet. Finite field linear algebra subroutines. In Teo Mora, editor, *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, Lille, France*, pages 63–74. ACM Press, New York, July 2002.

[3] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. FFPACK: Finite field linear algebra package. In Jaime Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain*, pages 119–126. ACM Press, New York, July 2004.

[4] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999.

[5] B. David Saunders. Personal communication, 2001.