

Compressed Modular Matrix Multiplication

Jean-Guillaume Dumas* Laurent Fousse* Bruno Salvy†

<http://www.orcca.on.ca/conferences/mica2008>

Abstract

Matrices of integers modulo a small prime can be compressed by storing several entries into a single machine word. Modular addition is performed by addition and possibly subtraction of a word containing several times the modulus. We show how modular multiplication can also be performed. In terms of arithmetic operations, the gain over classical matrix multiplication is equal to the number of integers that are stored inside a machine word. The gain in actual speed is also close to that number.

First, modular dot product can be performed via an integer multiplication by the reverse integer. Modular multiplication by a word containing a single residue is also possible. We give bounds on the sizes of primes and matrices for which such a compression is possible. We also make explicit the details of the required compressed arithmetic routines and show some practical performance.

Keywords : Kronecker substitution ; Finite field ; Modular Polynomial Multiplication ; REDQ (simultaneous modular reduction) ; DQT (Discrete Q -adic Transform) ; FQT (Fast Q -adic Transform).

1 Introduction

Compression of matrices over fields of characteristic 2 is classically made via the binary representation of machine integers and has numerous uses in number theory [1, 10]. The need for efficient matrix computations over *very small* finite fields of characteristic other than 2 arises in particular in graph theory (adjacency matrices), see, e.g., [11] or [12].

The FFLAS/FFPACK project has demonstrated the efficiency that is gained by wrapping cache-aware BLAS routines for efficient linear algebra over small finite fields [4, 5, 2]. The conversion between a modular representation of prime fields of any (small) characteristic and floating points can be performed via the homomorphism to the integers. For extension fields, the elements are naturally represented as polynomials over prime fields. In [3] it is proposed to transform these polynomials into a Q -adic representation where Q is an integer larger than the characteristic of the field. This transformation is called DQT for Discrete Q -adic Transform, it is a form of Kronecker substitution [7, §8.4]. With some care, in particular on the size of Q , it is possible to map the polynomial operations into the floating point arithmetic realization of this Q -adic representation and convert back using an inverse DQT.

In this work, we propose to use this fast polynomial arithmetic within machine words to compress matrices over very small finite fields. This is achieved by storing groups of $d + 1$ entries of the matrix into one floating point number each, where d is a parameter to be maximized depending on the cardinality of the finite field and the size of the matrices. First, we show in Section 2 how a dot product of vectors of size $d + 1$ can be recovered from a single machine word multiplication. This extends to matrix multiplication by compressing both matrices first. Then we propose in Section 3 an alternative matrix multiplication using multiplication

*Laboratoire J. Kuntzmann, Université de Grenoble, umr CNRS 5224. BP 53X, 51, rue des Mathématiques, F38041 Grenoble, France. {Jean-Guillaume.Dumas,Laurent.Fousse}@imag.fr. This work is supported in part by the French *Agence Nationale pour la Recherche* (ANR Safescale).

†Algorithms Project, INRIA Rocquencourt, 78153 Le Chesnay. France. Bruno.Salvy@inria.fr. This work is supported in part by the French *Agence Nationale pour la Recherche* (ANR Gecko).

of a compressed word by a single residue. This operation also requires a simultaneous modular reduction, which is called REDQ in [3] where its efficient implementation is described.

In general, the prime field, the size of matrices and the available mantissa are given. This gives some constraints on the possible choices of Q and d . In both cases anyway, we show that these compression techniques represent a speed-up factor of up to the number $d + 1$ of residues stored in the compressed format. We conclude in Section 5 with a comparison of the techniques.

2 Q-adic compression or Dot product via polynomial multiplication

2.1 Modular dot product via machine word multiplication

If $a = [a_0, \dots, a_d]$ and $b = [b_0, \dots, b_d]$ are two vectors with entries in $\mathbb{Z}/p\mathbb{Z}$, their dot product $\sum a_i b_i$ is the coefficient of degree d in the product of polynomials $a(X) = \sum_{i=0}^d a_{d-i} X^i$ and $b(X) = \sum_{i=0}^d b_i X^i$. The idea here, as in [3], is to replace X by an integer Q , usually a power of 2 in order to speed up conversions. Thus the vectors of residues a and b are stored respectively as $\bar{b} = \sum_{i=0}^d b_i Q^i$ and the *reverse* $\bar{a} = \sum_{i=0}^d a_{d-i} Q^i$.

For instance, for $d = 2$, the conversion is performed by the following compression:

```
double& init3( double& r, const double u, const double v, const double w) {
    // _dQ is a floating point storage of Q
    r=u; r*=_dQ; r+=v; r*=_dQ; return r+=w;
}
```

2.2 Compressed Matrix Multiplication

We first illustrate the idea for 2×2 matrices and $d = 1$. The product

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

is recovered from

$$\begin{bmatrix} Qa + b \\ Qc + d \end{bmatrix} \times [e + Qg \quad f + Qh] = \begin{bmatrix} * + (ae + bg)Q + *Q^2 & * + (af + bh)Q + *Q^2 \\ * + (ce + dg)Q + *Q^2 & * + (cf + dh)Q + *Q^2 \end{bmatrix},$$

where the character $*$ denotes other coefficients.

In general, A is an $m \times k$ matrix to be multiplied by a $k \times n$ matrix B , the matrix A is first compressed into a $m \times \left\lceil \frac{k}{d+1} \right\rceil$ **CompressedRowMatrix**, CA , and B is transformed into a $\left\lceil \frac{k}{d+1} \right\rceil \times n$ **CompressedColumnMatrix**, CB . The compressed matrices are then multiplied and the result can be extracted from there. This is depicted on Fig. 1

In terms of number of arithmetic operations, the matrix multiplication $CA \times CB$ can save a *factor of* $d + 1$ over the multiplication of $A \times B$ as shown on the 2×2 case above.

The computation has three stages: compression, multiplication and extraction of the result. The compression and extraction are less demanding in terms of asymptotic complexity, but can still be noticeable for moderate sizes. For this reason, compressed matrices are often reused and it is more informative to distinguish the three phases in an analysis. This is done in Section 5 (Table 2), where the actual matrix multiplication algorithm is also taken into account.

Partial compression Note that the last column of CA and the last row of B might not have $d + 1$ elements if $d + 1$ does not divide k . Thus one has to artificially append some zeroes to the converted values. On \bar{b} this means just do nothing. On the reversed \bar{a} this means multiplying by Q several times.

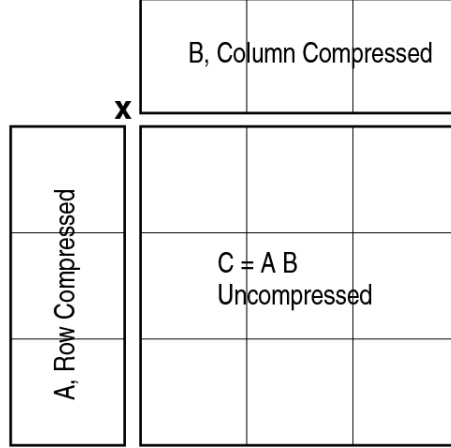


Figure 1: Compressed Matrix Multiplication (CMM)

2.3 Delayed reduction and lower bound on Q

For the results to be correct the inner dot product must not exceed Q . With a positive modular representation mod p (i.e. integers from 0 to $p - 1$), this means that we demand that the inequality $(d + 1)(p - 1)^2 < Q$ holds. Moreover, it is possible to save more time by using delayed reductions on the intermediate results, i.e., accumulating several products $\bar{a}\bar{b}$ before any modular reduction. It is thus possible to perform matrix multiplications with common dimension k as long as:

$$\frac{k}{d+1}(d+1)(p-1)^2 = k(p-1)^2 < Q. \quad (1)$$

2.4 Available mantissa and upper bound on Q

If the product $\bar{a}\bar{b}$ is performed with floating point arithmetic we just need that the coefficient of degree d fits in the β bits of the mantissa. Writing $\bar{a}\bar{b} = c_H Q^d + c_L$, we see that this implies that c_H , and only c_H , must remain smaller than 2^β . It can then be recovered exactly by multiplication of $\bar{a}\bar{b}$ with the correctly precomputed and rounded inverse of Q^d as shown e.g., in [3, Lemma 2].

With delayed reduction this means that

$$\sum_{i=0}^d \frac{k}{d+1} (i+1)(p-1)^2 Q^{d-i} < 2^\beta.$$

Using Eq. (1) shows that this is ensured if

$$Q^{d+1} < 2^\beta. \quad (2)$$

Thus a single reduction has to be made at the end of the dot product as follows:

```
Element& init( Element& rem, const double dp) const {
    double r = dp;
    // Multiply by the inverse of Q^d with correct rounding
    r *= _inverseQto_d;
    // Now we just need the part less than Q=2^t
    unsigned long r1( static_cast<unsigned long>(r) );
    r1 &= _QMINUSONE;
    // And we finally perform a single modular reduction
    r1 %= _modulus;
```

```

    return rem = static_cast<Element>(r1);
}

```

Note that one can avoid the multiplication by the inverse of Q when Q is a power of 2, say 2^t : by adding $Q^{2^{d+1}}$ to the final result one is guaranteed that the $t(d+1)$ high bits represent exactly the $d+1$ high coefficients. On the one hand, the floating point multiplication is then replaced by an addition. On the other hand, this doubles the size of the dot product and thus reduces by a factor of $\sqrt[2]{2}$ the largest possible dot product size k .

2.5 Results

On Figure 2 we compare our compression algorithm to the numerical double floating point matrix multiplication `dgemm` of GotoBlas [8] and to the `fgemm` modular matrix multiplication of the FFLAS-LinBox library [4]. For the latter we show timings using `dgemm` and also `sgemm` over single floating points.

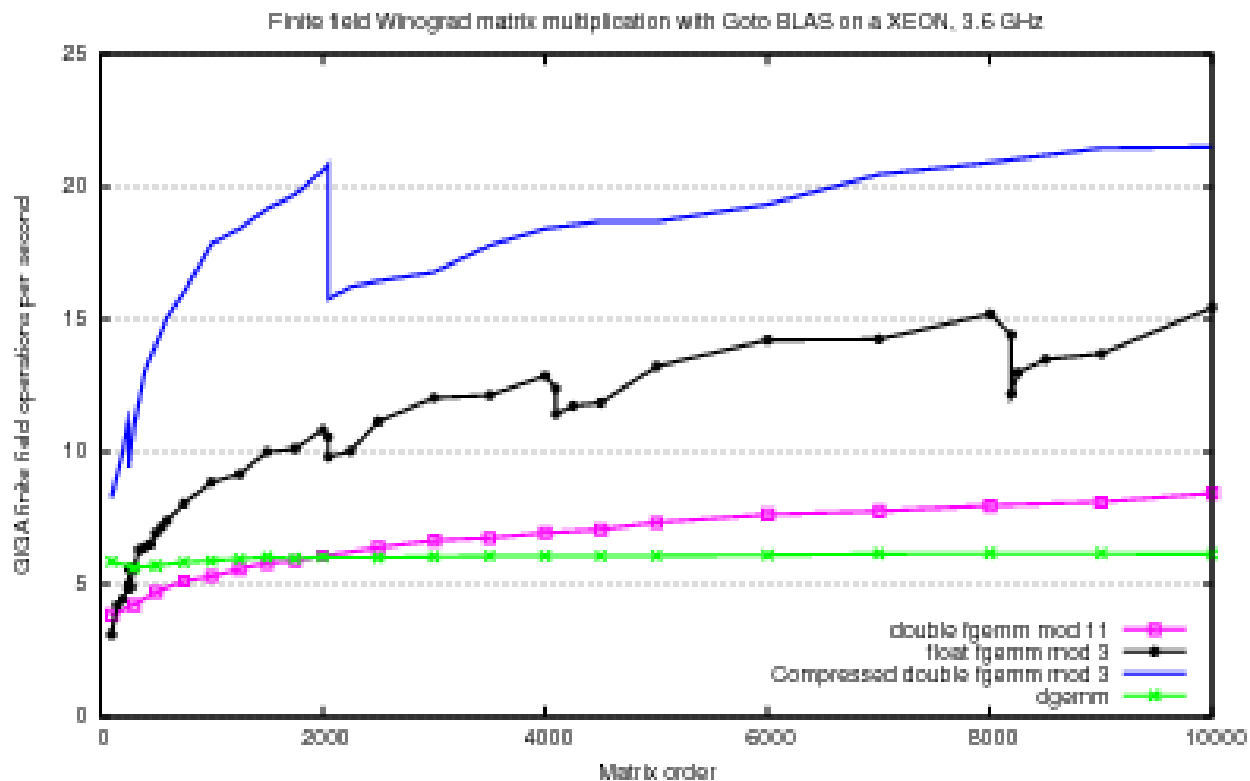


Figure 2: Compressed matrix multiplication compared with `dgemm` (the floating point double precision matrix multiplication of GotoBlas) and `fgemm` (the exact routine of FFLAS) with double or single precision.

This figure shows that the compression ($d+1$) is very effective for small primes: the gain over the double floating point routine is quite close to d .

Observe that the curve of `fgemm` with underlying arithmetic on single floats oscillates and drops sometimes. Indeed, the matrix begins to be too large and modular reductions are now required between the recursive matrix multiplication steps. Then the floating point BLAS¹ routines are used only when the submatrices are small enough. One can see the subsequent increase in the number of classical arithmetic steps on the drops around 2048, 4096 and 8192.

¹<http://www.tacc.utexas.edu/resources/software/>

Compression	2	3..4	5..8	8	7	6	5	4	3
Degree d	1	5	9	7	6	5	4	3	2
Q-adic	2 ³	2 ⁴	2 ⁵	2 ⁶	2 ⁷	2 ⁸	2 ¹⁰	2 ¹³	2 ¹⁷
Dimensions	2	≤ 4	≤ 8	≤ 16	≤ 32	≤ 64	≤ 256	≤ 2048	≤ 32768

Table 1: Compression factors for different common matrix dimensions modulo 3, with 53 bits of mantissa and Q a power of 2.

On Table 1, we show the compression factors modulo 3, with Q a power of 2 to speed up conversions. For a dimension $n \leq 256$ the compression is at a factor of five and the time to perform a matrix multiplication is less than a hundredth of a second. Then from dimensions from 257 to 2048 one has a factor of 4 and the times are roughly 16 times the time of the four times smaller matrix. The next stage, from 2048 to 32768 is the one that shows on Figure 1.

Figure 2 shows the dramatic impact of the compression dropping from 4 to 3 between $n = 2048$ and $n = 2049$. It would be interesting to compare the multiplication of 3-compressed matrices of size 2049 with a decomposition of the same matrix into matrices of sizes 1024 and 1025, thus enabling 4-compression also for matrices larger than 2048, but with more modular reductions.

3 Right or Left Compressed Matrix Multiplication

Another way of performing compressed matrix multiplication is to multiply an uncompressed $m \times k$ matrix to the right by a row-compressed $k \times \frac{n}{d+1}$ matrix. We illustrate the idea on 2×2 matrices:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e + Qf \\ g + Qh \end{bmatrix} = \begin{bmatrix} (ae + bg) + Q(af + bh) \\ (ce + dg) + Q(cf + dh) \end{bmatrix}$$

The general case is depicted on Fig. 3, center. This is called Right Compressed Matrix Multiplication. Left Compressed Matrix Multiplication is obtained by transposition.

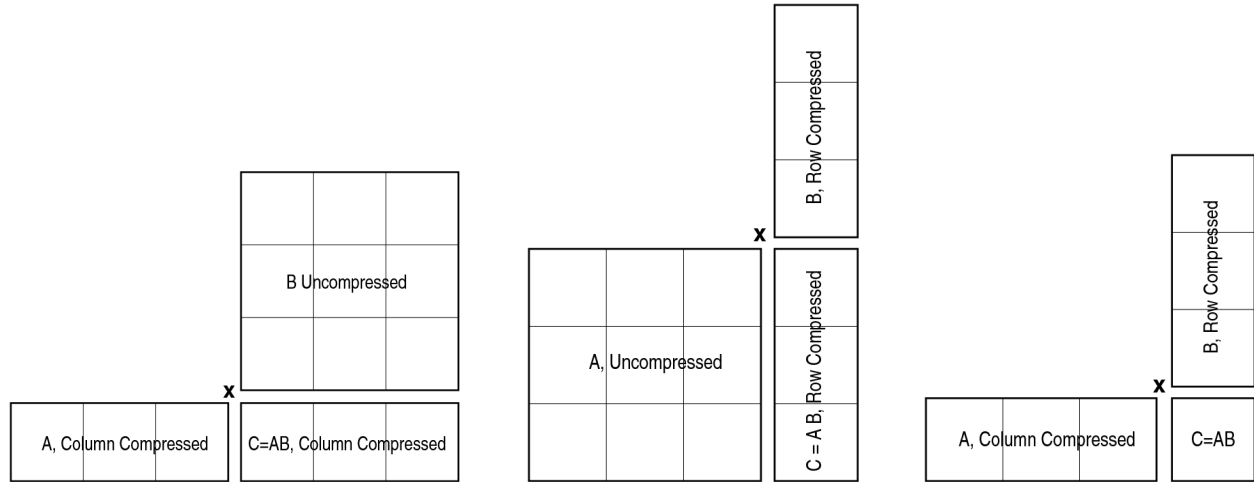


Figure 3: Left, Right and Full Compressions

Here also Q and d must satisfy Eqs. (1) and (2).

The major difference with the Compressed Matrix Multiplication lies in the reductions. Indeed, now one needs to reduce simultaneously the $d + 1$ coefficients of the polynomial in Q in order to get the results. This simultaneous reduction can be made by the REDQ algorithm of [3, Algorithm 2].

When working over compressed matrices CA and CB , a first step is to uncompress CA , which has to be taken into account when comparing methods. Thus the whole right compressed matrix multiplication is the following algorithm

$$A = \text{Uncompress}(CA); CC = A \times CB; \text{REDQ}(CC) \quad (3)$$

4 Full Compression

It is also possible to compress simultaneously both dimensions of the matrix product (see Fig. 3, right). This is achieved by using polynomial multiplication with two variables Q and Θ . Again, we start by an example in dimension 2:

$$[a + Qc \quad b + Qd] \times \begin{bmatrix} e + \Theta f \\ g + \Theta h \end{bmatrix} = [(ae + bg) + Q(ce + dg) + \Theta(af + bh) + Q\Theta(cf + dh)]$$

More generally, let d_q be the degree in Q and d_θ be the degree in Θ . Then, the dot product is:

$$\begin{aligned} a \cdot b &= \left[\sum_{i=0}^{d_q} a_{i0} Q^i, \dots, \sum_{i=0}^{d_q} a_{in} Q^i \right] \times \left[\sum_{j=0}^{d_\theta} b_{0j} \Theta^j, \dots, \sum_{j=0}^{d_\theta} b_{nj} \Theta^j \right], \\ &= \sum_{l=0}^k \left(\sum_{i=0}^{d_q} a_{il} \right) \left(\sum_{j=0}^{d_\theta} b_{lj} \right) Q^i \Theta^j = \sum_{i=0}^{d_q} \sum_{j=0}^{d_\theta} \left(\sum_{l=0}^k a_{il} b_{lj} \right) Q^i \Theta^j. \end{aligned}$$

In order to guarantee that all the coefficients can be recovered independently, Q must still satisfy Eq. (1) but then Θ must satisfy an additional constraint:

$$Q^{d_q+1} \leq \Theta \quad (4)$$

This imposes restrictions on d_q and d_θ :

$$Q^{(d_q+1)(d_\theta+1)} < 2^\beta \quad (5)$$

5 Comparison

In Table 2, we summarize the differences of the algorithms presented on Figures 1 and 3. As usual, the exponent ω denotes the exponent of matrix multiplication. Thus, $\omega = 3$ for the classical matrix multiplication, while $\omega < 3$ for faster matrix multiplications, as used in [6, §3.2]. For products of rectangular matrices, we use the classical technique of first decomposing the matrices into square blocks and then using fast matrix multiplication on those blocks.

Compression Factor The costs in Table 2 are expressed in terms of a *compression factor* e , that we define as

$$e := \left\lfloor \frac{\beta}{\log_2(Q)} \right\rfloor,$$

where, as above, β is the size of the mantissa and Q is the integer chosen according to Eqs. (1) and (2), except for Full Compression where the more constrained Eq. (5) is used.

Thus the degree of compression for the first three algorithms is just $d = e - 1$, while it becomes only $d = \sqrt{e} - 1$ for the full compression algorithm (with equal degrees $d_q = d_\theta = d$ for both variables Q and Θ).

Algorithm	Operations	Reductions	Conversions
CMM	$\mathcal{O}\left(mn\left(\frac{k}{e}\right)^{\omega-2}\right)$	$m \times n$ REDC	$\frac{1}{e}mn$ INIT _e
Right Comp.	$\mathcal{O}\left(mk\left(\frac{n}{e}\right)^{\omega-2}\right)$	$m \times \frac{n}{e}$ REDQ _e	$\frac{1}{e}mn$ EXTRACT _e
Left Comp.	$\mathcal{O}\left(nk\left(\frac{m}{e}\right)^{\omega-2}\right)$	$\frac{m}{e} \times n$ REDQ _e	$\frac{1}{e}mn$ EXTRACT _e
Full Comp.	$\mathcal{O}\left(k\left(\frac{mn}{e}\right)^{\frac{\omega-1}{2}}\right)$	$\frac{m}{\sqrt{e}} \times \frac{n}{\sqrt{e}}$ REDQ _e	$\frac{1}{e}mn$ INIT _e

Table 2: Number of arithmetic operations for the different algorithms

Analysis In terms of asymptotic complexity, the cost in number of arithmetic operations is dominated by that of the product (column Operations in the table), while reductions and conversions are linear in the dimensions. This is well reflected in practice. For example, with algorithm CMM on matrices of sizes $10,000 \times 10,000$ it took 92.75 seconds to perform the matrix multiplication modulo 3 and 0.25 seconds to convert the resulting matrix. This is less than 0.3%. For 250×250 matrices it takes less than 0.0028 seconds to perform the multiplication and roughly 0.00008 seconds for the conversions. There, the conversions account for 3% of the time.

In the case of rectangular matrices, the second column of Table 2 shows that one should choose the algorithm depending on the largest dimension: CMM if the common dimension k is the largest, Right Compression if n is the largest and Left Compression if m dominates. The gain in terms of arithmetic operations is $e^{\omega-2}$ for the first three variants and $e^{\frac{\omega-1}{2}}$ for full compression. This is not only of theoretical interest but also of practical value, since the compressed matrices are then less rectangular. This enables more locality for the matrix computations and usually results in better performance. Thus, even if $\omega = 3$, i.e., classical multiplication is used, these considerations point to a source of speed improvement.

The full compression algorithm seems to be the best candidate for locality and use of fast matrix multiplication; however the compression factor is an integer, depending on the flooring of either $\frac{\beta}{\log_2(Q)}$ or $\sqrt{\frac{\beta}{\log_2(Q)}}$. Thus there are matrix dimensions for which the compression factor of e.g., the right compression will be larger than the square of the compression factor of the full compression. There the right compression will have some advantage over the full compression.

If the matrices are square ($m = n = k$) or if $\omega = 3$, the products all become the same, with similar constants implied in the $O()$, so that apart from locality considerations, the difference between them lies in the time spent in reductions and conversions. Since the REDQ_e reduction is faster than e classical reductions [3], and since INIT_e and EXTRACT_e are roughly the same operations, the best algorithm would then be one of the Left, Right or Full compression. Further work would include implementing the Right or Full compression and comparing the actual timings of conversion overhead with that of algorithm CMM.

References

- [1] Don Coppersmith. Solving linear equations over $GF(2)$: block Lanczos algorithm. *Linear Algebra and its Applications*, 192:33–60, October 1993.
- [2] 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.
- [3] Jean-Guillaume Dumas. Q-adic transform revisited. In David Jeffrey, editor, *Proceedings of the 2008 International Symposium on Symbolic and Algebraic Computation, Hagenberg, Austria*. ACM Press, New York, July 2008.

- [4] 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.
- [5] 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.
- [6] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Dense linear algebra over prime fields. *ACM Transactions on Mathematical Software*, 2008. to appear.
- [7] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999.
- [8] Kazushige Goto and Robert van de Geijn. On reducing TLB misses in matrix multiplication. Technical Report TR-2002-55, University of Texas, November 2002. FLAME working note #9.
- [9] Chao H. Huang and Fred J. Taylor. A memory compression scheme for modular arithmetic. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(6):608–611, December 1979.
- [10] Erich Kaltofen and Austin Lobo. Distributed matrix-free solution of large sparse linear systems over finite fields. In A.M. Tentner, editor, *Proceedings of High Performance Computing 1996, San Diego, California*. Society for Computer Simulation, Simulation Councils, Inc., April 1996.
- [11] John P. May, David Saunders, and Zhendong Wan. Efficient matrix rank computation with application to the study of strongly regular graphs. In Christopher W. Brown, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation, Waterloo, Canada*, pages 277–284. ACM Press, New York, July 29 – August 1 2007.
- [12] Guobiao Weng, Weisheng Qiu, Zeying Wang, and Qing Xiang. Pseudo-Paley graphs and skew Hadamard difference sets from presemifields. *Designs, Codes and Cryptography*, 44(1-3):49–62, 2007.