Algorithms & Techniques for Dense Linear Algebra over Small Finite Fields

Martin R. Albrecht
(martinralbrecht+summerschool@googlemail.com)

POLSYS Team, UPMC, Paris, France

ECrypt II PhD Summer School

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

The M4RI Library

- available under the GPL Version 2 or later (GPLv2+)
- provides basic arithmetic (addition, equality testing, stacking, augmenting, sub-matrices, randomisation, etc.)
- asymptotically fast multiplication
- asymptotically fast elimination
- some multi-core support
- Linux, Mac OS X (x86 and PPC), OpenSolaris (Sun Studio Express) and Windows (Cygwin)

`http://m4ri.sagemath.org`

$\mathbb{F}_2$

- field with two elements.
- logical bitwise XOR is addition.
- logical bitwise AND is multiplication.
- 64 (128) basic operations in at most one CPU cycle
- ... arithmetic rather cheap

		$\oplus$	$\odot$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Memory access is the expensive operation, not arithmetic.

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

Gray Codes

The Gray code [Gra53], named after Frank Gray and also known as reflected binary code, is a numbering system where two consecutive values differ in only one digit.

Gray Code Examples

0
1

0	**0**	$\Downarrow$
0	**1**	
1	1	
1	0	$\Uparrow$

0	**0**	**0**
0	**0**	**1**
0	**1**	**1**
0	**1**	**0**
1	1	0
1	1	1
1	0	1
1	0	0

0	**0**	**0**	**0**
0	**0**	**0**	**1**
0	**0**	**1**	**1**
0	**0**	**1**	**0**
0	**1**	**1**	**0**
0	**1**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**0**	**0**
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

Applications

Gray codes are used in various applications where all vectors over small finite fields need to be enumerated, such as:

- matrix multiplication;
- fast exhaustive search of Boolean polynomial systems;
- cube attacks on Grain-128.

Gray codes are a pretty basic part of the cryptographer's toolkit because they allow to reduce the cost of enumerating all vectors over $\mathbb{F}_2$ of length n from $n2^n - 1$ to $2^n - 1$.

Outline

$\mathbb{F}_2$

Gray Codes

Multiplication

Elimination

$\mathbb{F}_p$

$\mathbb{F}_{2^e}$

Precomputation Tables

Karatsuba Multiplication

Performance

$\mathbb{F}_p[x]$

M4RM [ADKF70] I

Consider $C = A \cdot B$ (A is $m \times \ell$ and B is $\ell \times n$).

A can be divided into ℓ/k vertical "stripes"

$$A_0 \dots A_{(\ell-1)/k}$$

of k columns each. B can be divided into ℓ/k horizontal "stripes"

$$B_0 \dots B_{(\ell-1)/k}$$

of k rows each. We have:

$$C = A \cdot B = \sum_{0}^{(\ell-1)/k} A_i \cdot B_i.$$

M4RM [ADKF70] II

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, A_0 = \begin{pmatrix} \mathbf{1} & \mathbf{1} \\ 0 & 0 \\ \mathbf{1} & \mathbf{1} \\ 0 & 1 \end{pmatrix}$$

$$A_1 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{pmatrix}, B_0 = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, B_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$$A_0 \cdot B_0 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 1 & 0 \end{pmatrix}, A_1 \cdot B_1 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

M4RM: Algorithm $\mathcal{O}\left(n^3/\log n\right)$

```
begin
    C ⟵ create an m × n matrix with all entries 0;
    k ⟵ ⌊log n⌋;
    for 0 ≤ i < (ℓ/k) do
        // create table of 2^k − 1 linear combinations
        T ← MakeTable(B, i × k, 0, k);
        for 0 ≤ j < m do
            // read index for table T
            id ⟵ ReadBits(A, j, i × k, k);
            add row id from T to row j of C;
    return C;
```

Algorithm 1: M4RM

Strassen-Winograd [Str69] Multiplication

- fastest known pratical algorithm
- complexity: $\mathcal{O}\left(n^{\log_2 7}\right)$
- linear algebra constant: $\omega = \log_2 7$
- M4RM can be used as base case for small dimensions

$\rightarrow$ optimisation of this base case

Cache Friendly M4RM I

```
begin
    C ⟵ create an m × n matrix with all entries 0;
    for 0 ≤ i < (ℓ/k) do
        // this is cheap in terms of memory access
        T ← MakeTable(B, i × k, 0, k);
        for 0 ≤ j < m do
            // for each load of row j we take care of only k bits
            id ⟵ ReadBits(A, j, i × k, k);
            add row id from T to row j of C;
    return C;
```

Cache Friendly M4RM II

```
begin
    C ⟵ create an m × n matrix with all entries 0;
    for 0 ≤ start < m/b_s do
        for 0 ≤ i < (ℓ/k) do
            // we regenerate T for each block
            T ← MakeTable(B, i × k, 0, k);
            for 0 ≤ s < b_s do
                j ⟵ start × b_s + s;
                id ⟵ ReadBits(A, j, i × k, k);
                add row id from T to row j of C;
    return C;
```

$t > 1$ Gray Code Tables I

- actual arithmetic is quite cheap compared to memory reads and writes
- the cost of memory accesses greatly depends on where in memory data is located
- try to fill all of L1 with Gray code tables.
- Example: $k = 10$ and 1 Gray code table $\rightarrow$ 10 bits at a time. $k = 9$ and 2 Gray code tables, still the same memory for the tables but deal with 18 bits at once.
- The price is one extra row addition, which is cheap if the operands are all in cache.

$t > 1$ Gray Code Tables II

begin
    $C \longleftarrow$ create an $m \times n$ matrix with all entries 0;
    for $0 \leq i < (\ell/(2k))$ **do**
        $T_0 \leftarrow \textsc{MakeTable}(B, i \times 2k, 0, k)$;
        $T_1 \leftarrow \textsc{MakeTable}(B, i \times 2k + k, 0, k)$;
        for $0 \leq j < m$ **do**
            $id_0 \longleftarrow \textsc{ReadBits}(A, j, i \times 2k, k)$;
            $id_1 \longleftarrow \textsc{ReadBits}(A, j, i \times 2k + k, k)$;
            add row id_0 from T_0 and row id_1 from T_1 to row j of C;
    return C;

Performance: Multiplication

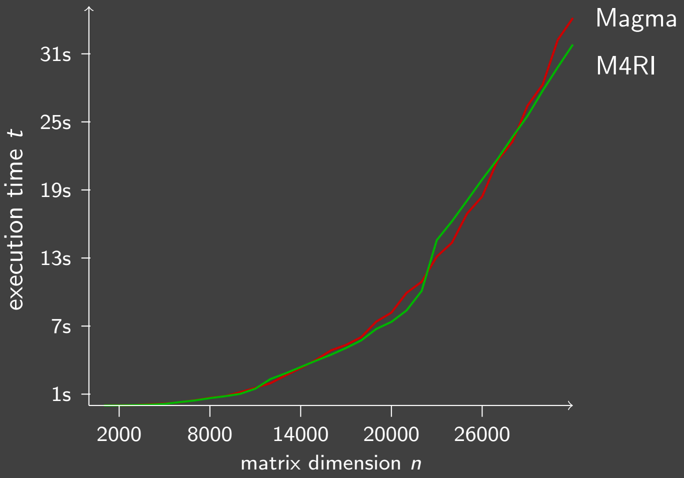
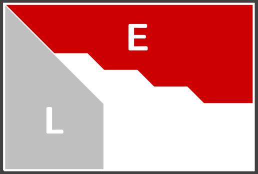


Figure: 2.66 Ghz Intel i7, 4GB RAM

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

PLE Decomposition I



Definition (PLE)

Let A be a $m \times n$ matrix over a field K. A PLE decomposition of A is a triple of matrices P, L and E such that P is a $m \times m$ permutation matrix, L is a unit lower triangular matrix, and E is a $m \times n$ matrix in row-echelon form, and

$$A = PLE.$$

PLE decomposition can be in-place, that is L and E are stored in A and P is stored as an m-vector.

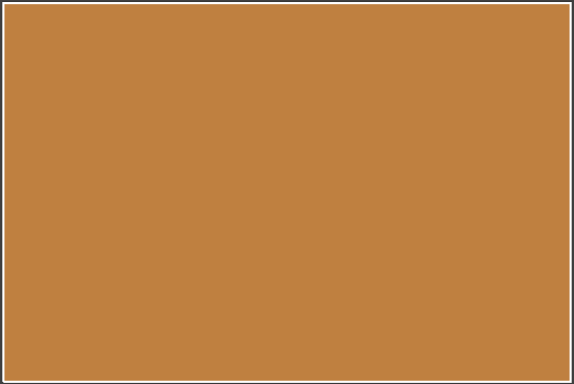
PLE Decomposition II

From the PLE decomposition we can

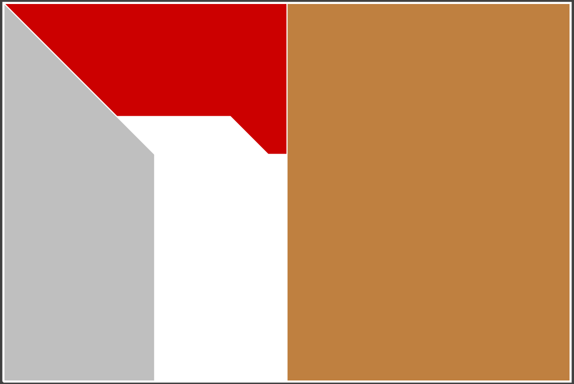
- read the rank r,
- read the row rank profile (pivots),
- compute the null space,
- solve $y = Ax$ for x and
- compute the (reduced) row echelon form.

C.-P. Jeannerod, C. Pernet, and A. Storjohann.
Rank-profile revealing Gaussian elimination and the CUP matrix decomposition.
arXiv:1112.5717, 35 pages, 2012.

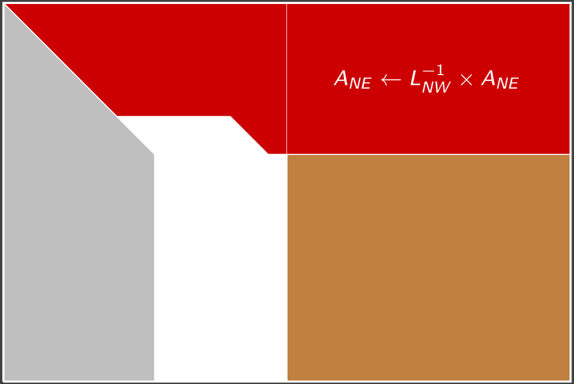
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ I



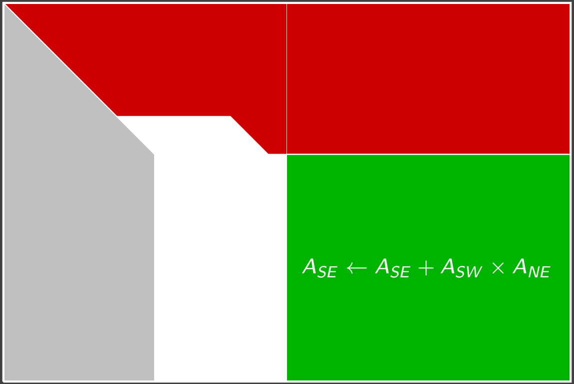
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ II



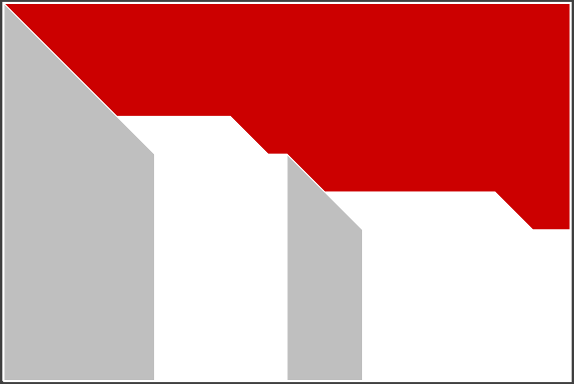
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ III



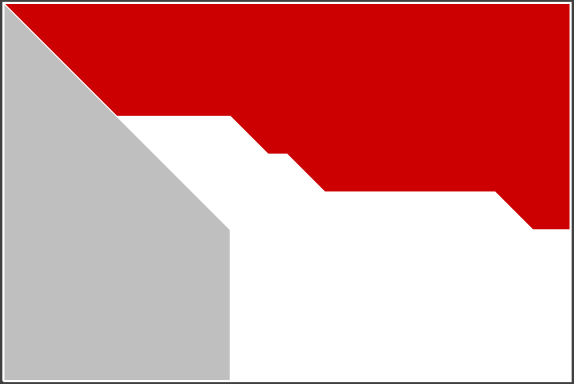
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ IV



Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ V



Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ VI



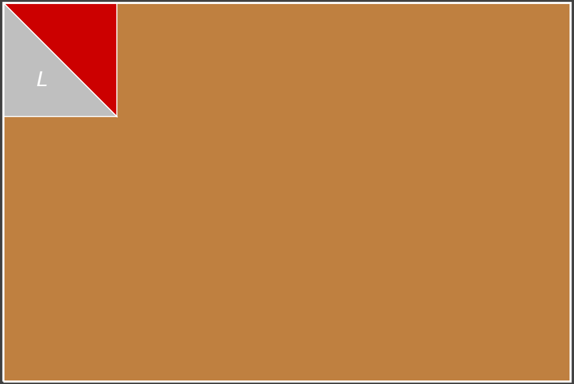
Block Iterative PLE Decomposition I

We need an efficient base case for PLE Decomposition

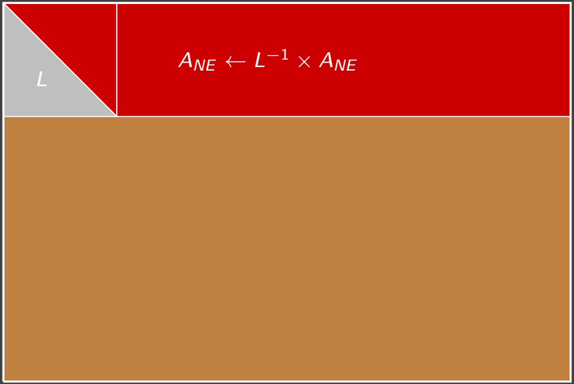
- block recursive PLE decomposition gives rise to a block iterative PLE decomposition
- choose blocks of size $k = \log n$ and use M4RM for the "update" multiplications
- this gives a complexity $\mathcal{O}\left(n^3/\log n\right)$

Block Iterative PLE Decomposition II

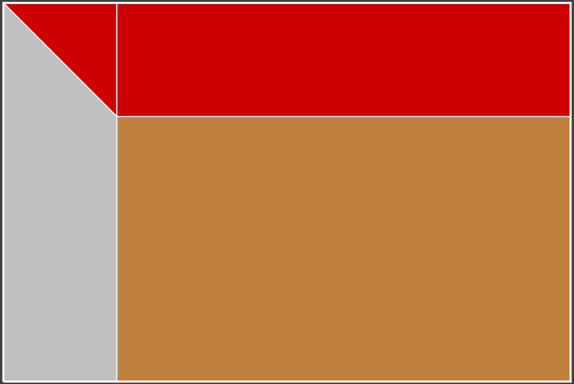
Block Iterative PLE Decomposition III



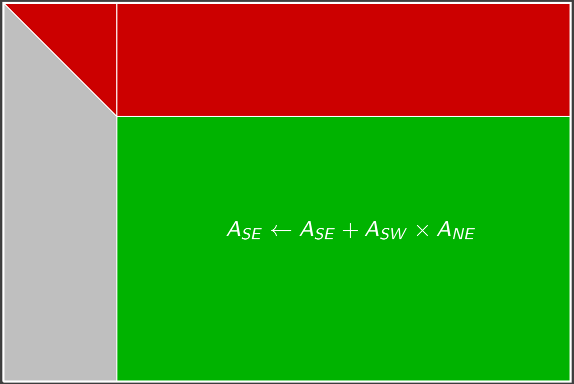
Block Iterative PLE Decomposition IV



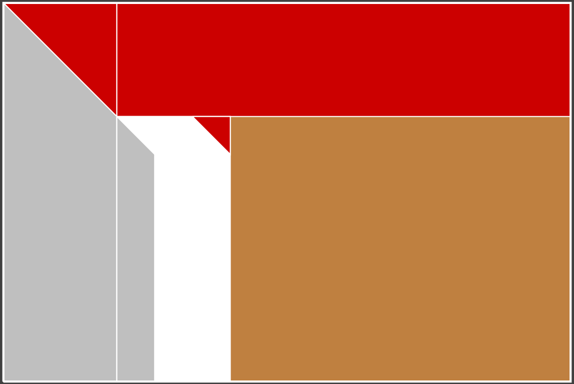
Block Iterative PLE Decomposition V



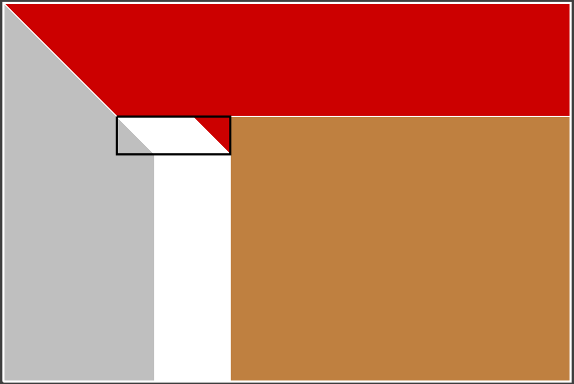
Block Iterative PLE Decomposition VI



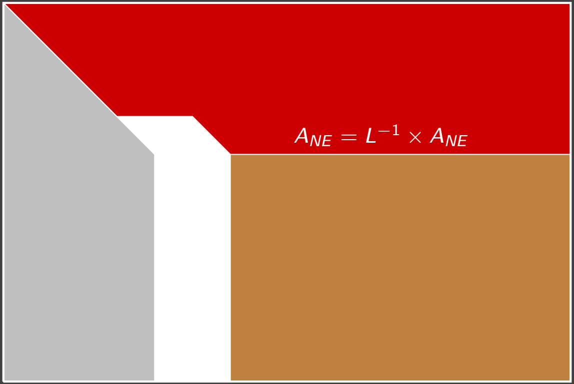
Block Iterative PLE Decomposition VII



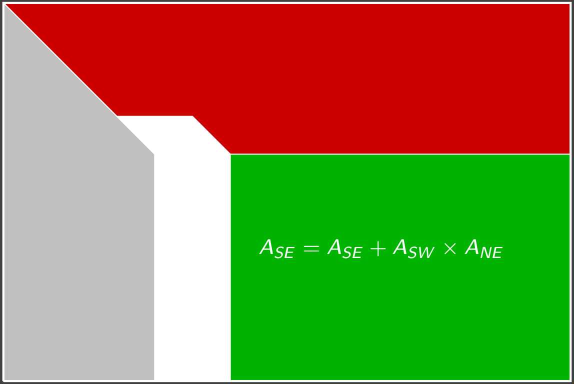
Block Iterative PLE Decomposition VIII



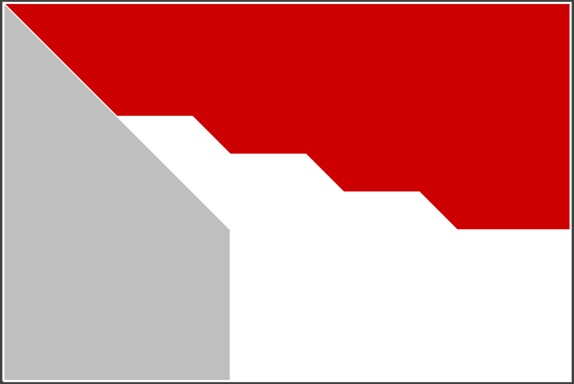
Block Iterative PLE Decomposition IX



Block Iterative PLE Decomposition X



Block Iterative PLE Decomposition XI



Performance: Reduced Row Echelon Form

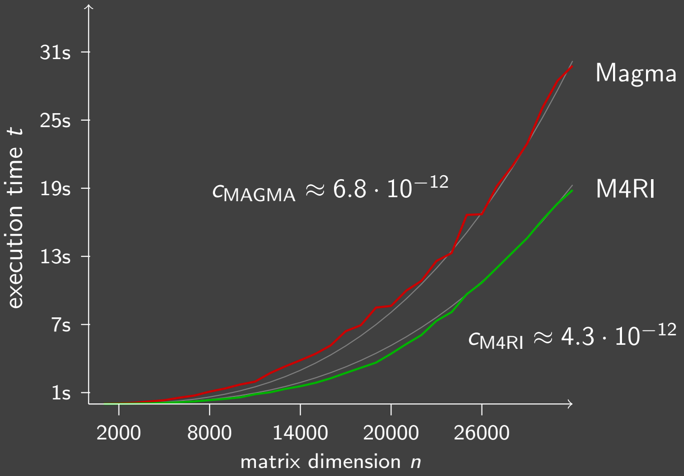


Figure: 2.66 Ghz Intel i7, 4GB RAM

Performance: Row Echelon Form

Using one core – on sage.math – we can compute the echelon form of a $500,000 \times 500,000$ dense random matrix over $\mathbb{F}_2$ in

$$9711 \text{ seconds } = 2.7 \text{ hours } (c \approx 10^{-12}).$$

Using four cores decomposition we can compute the echelon form of a random dense $500,000 \times 500,000$ matrix in

$$3806 \text{ seconds } = 1.05 \text{ hours.}$$

Caveat: Sensitivity to Sparsity

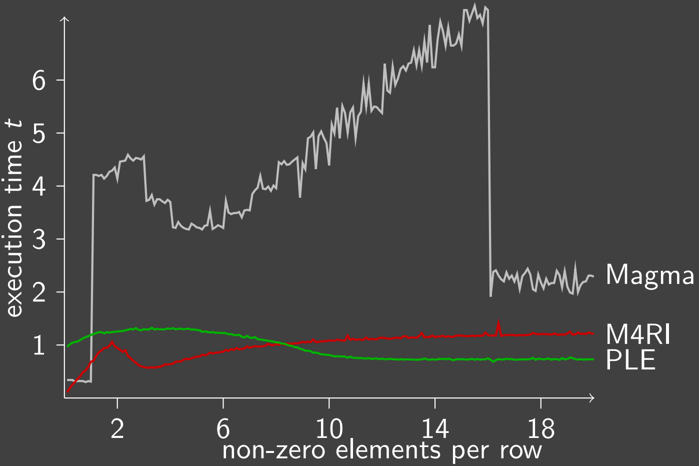
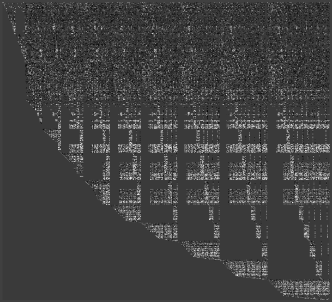


Figure: Gaussian elimination of $10,000 \times 10,000$ matrices on Intel 2.33GHz Xeon E5345 comparing Magma 2.17-12 and M4RI 20111004.

Caveat: Linear Algebra for Gröbner Basis



Problem	matrix dimensions	density	PLE	M4RI	GB
HFE 25 matrix 5 (5.1M)	12307 x 13508	0.07600	1.03	0.59	0.81
HFE 30 matrix 5 (16M)	19907 x 29323	0.06731	4.79	2.70	4.76
HFE 35 matrix 5 (37M)	29969 x 55800	0.05949	19.33	9.28	19.51
Mutant matrix (39M)	26075 x 26407	0.18497	5.71	3.98	2.10
random n=24, m=26 matrix 3 (30M)	37587 x 38483	0.03832	20.69	21.08	19.36
random n=24, m=26 matrix 4 (24M)	37576 x 32288	0.04073	18.65	28.44	17.05
SR(2,2,2,4) compressed, matrix 2 (328K)	5640 x 14297	0.00333	0.40	0.29	0.18
SR(2,2,2,4) compressed, matrix 4 (2.4M)	13665 x 17394	0.01376	2.18	3.04	2.04
SR(2,2,2,4) compressed, matrix 5 (2.8M)	11606 x 16282	0.03532	1.94	4.46	1.59
SR(2,2,2,4) matrix 6 (1.4M)	13067 x 17511	0.00892	1.90	2.09	1.38
SR(2,2,2,4) matrix 7 (1.7M)	12058 x 16662	0.01536	1.53	1.93	1.66
SR(2,2,2,4) matrix 9 (36M)	115834 x 118589	0.00376	528.21	578.54	522.98

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

$p < 2^{23}$

- For medium sized primes your best bet is LinBox or more precisely FFLAS/FFPACK (C++ libraries).
- It reduces computations $\bmod p$ to computations with floating point numbers.
- On top of that it implements asymptotically fast techniques (Strassen, PLE, ...).

http://www.linalg.org/

p very small: Packing

- If p is small, you can pack several entries into one machine word. If there is enough zero padding these remain independent.
- There exists code to do this by the LinBox people but it's not in LinBox (yet).

p very small: Slicing

If $p \in (3, 5, 7)$ you can bit-slice your entries and implement the boolean circuit to perform arithmetic on machine words. If your prime has k-bits and you want to represent n elements, you'd represent your elements as k bitstrings of length n.

Example

Represent $\mathbb{F}_3$ as $0 : [0, 0], 1 : [1, 0], -1 : [1, 1]$. To add two elements $[x_0, x_1]$ and $[y_0, y_1]$ compute: $s \leftarrow x_0 \oplus y_1, t \leftarrow x_1 \oplus y_0$ and return $[s \wedge t, (s \oplus x_1) \vee (t \oplus y_1)]$.

Unfortunately, there is no ready-made library available yet which implements this (but there is some proof-of-concept code by Tom Boothby).

Outline

$\mathbb{F}_2$

- Gray Codes
- Multiplication
- Elimination

$\mathbb{F}_p$

$\mathbb{F}_{2^e}$

- Precomputation Tables
- Karatsuba Multiplication
- Performance

$\mathbb{F}_p[x]$

The M4RIE Library

- handles $\mathbb{F}_{2^e}$ for $2 \leq e \leq 10$; $e \leq 16$ planned.
- available under the GPL Version 2 or later (GPLv2+)
- provides basic arithmetic (addition, equality testing, stacking, augmenting, sub-matrices, randomisation, etc.)
- implements asymptotically fast multiplication
- implements asymptotically fast elimination
- Linux, Mac OS X (x86 and PPC), OpenSolaris, and Windows (Cygwin)

`http://m4ri.sagemath.org`

Representation of Elements I

Elements in $\mathbb{F}_{2^e} \cong \mathbb{F}_2[x]/f$ can be written as

$$a_0\alpha^0 + a_1\alpha^1 + \cdots + a_{e-1}\alpha^{e-1}.$$

We identify the bitstring $a_0, \ldots, a_{e-1}$ with

- the element $\sum_{i=0}^{e-1} a_i\alpha^i \in \mathbb{F}_{2^e}$ and
- the integer $\sum_{i=0}^{e-1} a_i 2^i$.

In the datatype `mzed_t` we pack several of those bitstrings into one machine word:

$$a_{0,0,0}, \ldots, a_{0,0,e-1},\ a_{0,1,0}, \ldots, a_{0,1,e-1},\ \ldots,\ a_{0,n-1,0}, \ldots, a_{0,n-1,e-1}.$$

Additions are cheap, scalar multiplications are expensive.

Representation of Elements II

- Instead of representing matrices over $\mathbb{F}_{2^e}$ as matrices over polynomials we may represent them as polynomials with matrix coefficients.
- For each degree we store matrices over $\mathbb{F}_2$ which hold the coefficients for this degree.
- The data type `mzd_slice_t` for matrices over $\mathbb{F}_{2^e}$ internally stores e-tuples of M4RI matrices, i.e., matrices over $\mathbb{F}_2$.

Additions are cheap, scalar multiplications are expensive.

Representation of Elements III

$$\begin{aligned} A &= \begin{pmatrix} \alpha^2+1 & \alpha \\ \alpha+1 & 1 \end{pmatrix} \\ &= \begin{bmatrix} \square 101 & \square 010 \\ \square 011 & \square 001 \end{bmatrix} \\ &= \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \right) \end{aligned}$$

Figure: 2×2 matrix over $\mathbb{F}_8$

Outline

The idea I

Input: A – $m \times n$ matrix
Input: B – $n \times k$ matrix

begin
  for $0 \leq i < m$ **do**
    for $0 \leq j < n$ **do**
      $C_j \longleftarrow C_j + A_{j,i} \times B_i$;
  return C;

The idea II

Input: A – $m \times n$ matrix
Input: B – $n \times k$ matrix

begin
  for $0 \leq i < m$ **do**
    for $0 \leq j < n$ **do**
      $C_j \longleftarrow C_j + A_{j,i} \times B_i$; // **cheap**
  return C;

The idea III

Input: A – $m \times n$ matrix
Input: B – $n \times k$ matrix

begin
  for $0 \leq i < m$ **do**
    for $0 \leq j < n$ **do**
      $C_j \longleftarrow C_j + A_{j,i} \times B_i$; // **expensive**
  return C;

The idea IV

Input: A – $m \times n$ matrix
Input: B – $n \times k$ matrix

begin
    for $0 \leq i < m$ **do**
        for $0 \leq j < n$ **do**
            $C_j \longleftarrow C_j + A_{j,i} \times B_i$; // **expensive**
    return C;

But there are only 2^e possible multiples of B_i.

The idea V

begin

    Input: A – $m \times n$ matrix

    Input: B – $n \times k$ matrix

    for $0 \leq i < m$ **do**

        for $0 \leq j < 2^e$ **do**

            $T_j \longleftarrow j \times B_i$;

        for $0 \leq j < n$ **do**

            $x \longleftarrow A_{j,i}$;

            $C_j \longleftarrow C_j + T_x$;

    return C;

$m \cdot n \cdot k$ additions, $m \cdot 2^e \cdot k$ multiplications.

Gaussian elimination & PLE decomposition

Input: A – $m \times n$ matrix

begin

    $r \longleftarrow 0$;
    for $0 \leq j < n$ **do**
        for $r \leq i < m$ **do**
            if $A_{i,j} = 0$ **then** **continue**;
            rescale row i of A such that $A_{i,j} = 1$;
            swap the rows i and r in A;
            $T \longleftarrow$ multiplication table for row r of A;
            for $r + 1 \leq k < m$ **do**
                $x \longleftarrow A_{k,j}$;
                $A_k \longleftarrow A_k + T_x$;
            $r \longleftarrow r + 1$;
        return r;

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

The idea

- Consider $\mathbb{F}_{2^2}$ with the primitive polynomial $f = x^2 + x + 1$.
- We want to compute $C = A \cdot B$.
- Rewrite A as $A_0x + A_1$ and B as $B_0x + B_1$.
- The product is

$$C = A_0B_0x^2 + (A_0B_1 + A_1B_0)x + A_1B_1.$$

- Reduction modulo f gives

$$C = (A_0B_0 + A_0B_1 + A_1B_0)x + A_1B_1 + A_0B_0.$$

- This last expression can be rewritten as

$$C = ((A_0 + A_1)(B_0 + B_1) + A_1B_1)x + A_1B_1 + A_0B_0.$$

Thus this multiplication costs 3 multiplications and 4 adds over $\mathbb{F}_2$.

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
- $\mathbb{F}_p$
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance
- $\mathbb{F}_p[x]$

Performance: Multiplication

e	Magma 2.15-10	GAP 4.4.12	SW-NJ	SW-NJ/ M4RI	[Mon05]	Bitslice	Bitslice/ M4RI
1	0.100s	0.244s	–	1	1	0.071s	1.0
2	1.220s	12.501s	0.630s	8.8	3	0.224s	3.1
3	2.020s	35.986s	1.480s	20.8	6	0.448s	6.3
4	5.630s	39.330s	1.644s	23.1	9	0.693s	9.7
5	94.740s	86.517s	3.766s	53.0	13	1.005s	14.2
6	89.800s	85.525s	4.339s	61.1	17	1.336s	18.8
7	82.770s	83.597s	6.627s	93.3	22	1.639s	23.1
8	104.680s	83.802s	10.170s	143.2	27	2.140s	30.1

Table: Multiplication of $4,000 \times 4,000$ matrices over $\mathbb{F}_{2^e}$

Performance: Reduced Row Echelon Forms

e	Magma 2.15-10	GAP 4.4.12	LinBox (mod p) 1.1.6	M4RIE 6b24b839a46f
2	6.04s	162.65s	49.52s	3.31s
3	14.47s	442.52s	49.92s	5.33s
4	60.37s	502.67s	50.91s	6.33s
5	659.03s	N/A	51.20s	10.51s
6	685.46s	N/A	51.61s	13.08s
7	671.88s	N/A	53.94s	17.29s
8	840.22s	N/A	64.24s	20.25s
9	1630.38s	N/A	76.18s	260.77s
10	1631.35s	N/A	76.45s	291.30s

Table: Elimination of $10,000 \times 10,000$ matrices on 2.66Ghz i7

Outline

Prime-slicing

- The idea of bitsliced Karatsuba multiplication can be trivially extended to $\mathbb{F}_{p^e}$ and $\mathbb{F}_p[x]$ for $p > 2$.
- That is, we represent $(\mathbb{F}_p[x])^{m \times n}$ as $\mathbb{F}_p^{m \times n}[x]$ and
- use non-commutative Karatsuba-style formulas for multiplications in $\mathbb{F}_p[x]$.

Finding Formulas: Evaluation-Interpolation Schemes I

$f, g \in \mathbb{F}_{2^e}$, we

- consider them as polynomials $f(x), g(x)$ in $\mathbb{F}_2[x]$;
- evaluate those polynomials on sufficiently many points (possibly over some extension of $\mathbb{F}_2$),
- perform pointwise multiplication and
- interpolate $(f \cdot g)(x)$ from those points.

Finding Formulas: Evaluation-Interpolation Schemes II

Example: We multiply $f, g \in \mathbb{F}_{2^3}$, i.e., we are searching for

$$h(x) = f(x) \cdot g(x).$$

We compute $h(x)$ mod $p(x)$ where $\deg(p(x)) > \deg(h(x))$ such that $h(x) \bmod p(x) = h(x)$ and set

$$p(x) = (x + \infty) \cdot (x) \cdot (x + 1) \cdot (x^2 + x + 1).$$

That is, we compute modulo the factors of $p(x)$ and reconstruct the result using the Chinese remainder theorem. Multiplication modulo $(x + c)$ costs one in $\mathbb{F}_2$, modulo $x^2 + x + 1$ it costs 3 in $\mathbb{F}_2$. The total cost is 6 multiplications in $\mathbb{F}_2$.

Finding Formulas: Evaluation-Interpolation Schemes III

We can improve this strategy.

Example: We consider $f, g \in \mathbb{F}_{2^{11}}$. Instead of computing the solution modulo the product of **irreducible** polynomials

$$\begin{aligned} p(x) &= (x+\infty)\cdot(x)\cdot(x+1)\cdot(x^3+x+1)\cdot(x^3+x^2+1)\cdot \\ &\quad (x^4+x+1)\cdot(x^4+x^3+1)\cdot(x^4+x^3+x^2+x+1) \end{aligned}$$

with cost $3+2\cdot 6+3\cdot 9=42$, we compute modulo

$$\begin{aligned} p(x) &= (x+\infty)\cdot(\mathbf{x^2})\cdot(\mathbf{x+1})^2\cdot(x^2+x+1)\cdot(x^3+x+1)\cdot \\ &\quad (x^3+x^2+1)\cdot(x^4+x+1)\cdot(x^4+x^3+1). \end{aligned}$$

This only costs $1+3\cdot 3+2\cdot 6+2\cdot 9=40$ multiplications over $\mathbb{F}_2$.

Finding Formulas: Evaluation-Interpolation Schemes IV

How to find a good $p(x)$ for some degree e? $\Rightarrow$ We express this as a mixed integer linear program.

Let c be a table holding costs of polynomial multiplication, such that c_d is the cost of multiplying two polynomials modulo some polynomial of degree d: $c_0 = 0, c_1 = 1, c_2 = 3, \ldots$

Also, let

$$G_p(d) := \frac{1}{d} \sum_{d_i | d} \mu(d/d_i) p^{d_i}$$

be the function which returns the number of irreducible polynomials of degree d over $\mathbb{F}_p$.

Finding Formulas: Evaluation-Interpolation Schemes V

We want to minimize the function

$$1 + \sum_{d=1}^{\lceil \log_2(2e) \rceil} c_d n_d \qquad (1)$$

where n_d are number of degree d factors ($+1$ for $x + \infty$).

Our n_d must satisfy $\deg(p(x)) \geq 2e - 1$

$$\sum_{i=1}^{\lceil \log_2(2e) \rceil} n_d \cdot d \geq 2e - 2. \qquad (2)$$

Finding Formulas: Evaluation-Interpolation Schemes VI

We also have

$$0 \leq \sum_{i \in D(d)} n_i \leq \sum_{i \in D(d)} G_p(i) \tag{3}$$

for $1 \leq d \leq \lceil \log_2(2e) \rceil$ where $D(d)$ is defined as:

$$D(d) = \begin{cases} \{d\} & \text{if } d \text{ is odd} \\ \{d\} \cup D(d/2) & \text{else} \end{cases}$$

Minimizing (1) under the constraints (2) and (3), returns a $p(x)$ given by n_i.

This is a **very simple** mixed integer linear program and solving it for very large e is easy.

Finding Formulas: Evaluation-Interpolation Schemes VII

Adding a trick about field embeddings we get the follwing table.

e	$\mathbb{F}_2$	$\mathbb{F}_3$	$\mathbb{F}_{17}$	$\mathbb{F}_{39}$	$\mathbb{F}_{251}$
10	33	27	20	19	19
100	532	454	290	279	199
1000	6430	5455	3844	2997	2873
10000	71425	62845	43543	39217	29873
100000	755554	679861	474276	434007	355494

Table: Upper bounds on mul. in $\mathbb{F}_p$ for $f \cdot g \in \mathbb{F}_{p^e}$.

Note

There are sometimes better bounds known in the literature, the point here is that we can compute explicit formulas quickly.

Fin

V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev.
On economical construction of the transitive closure of a directed graph.
Dokl. Akad. Nauk., 194(11), 1970.
(in Russian), English Translation in Soviet Math Dokl.

Frank Gray.
Pulse code communication, March 1953.
US Patent No. 2,632,058.

Peter L. Montgomery.
Five, six, and seven-term Karatsuba-like formulae.
IEEE Trans. on Computers, 53(3):362–369, 2005.

Volker Strassen.
Gaussian elimination is not optimal.
Nummerische Mathematik, 13:354–256, 1969.