The M4RI & M4RIE libraries for linear algebra over $\mathbb{F}_2$ and small extensions

Martin R. Albrecht

Sage/FLINT Days, 19.12.2011, Warwick (UK)

Outline

- M4RI
  - Multiplication
  - Elimination
  - Projects
- M4RIE
  - Introduction
  - Newton-John Tables
  - Karatsuba Multiplication
  - Results

Outline

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

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

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

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

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

```
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;
end
```

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;
end
```

Cache Friendly M4RM II

begin

  $C \longleftarrow$ create an $m \times n$ matrix with all entries 0;

  for $0 \leq start < m/b_s$ **do**

    for $0 \leq i < (\ell/k)$ **do**

      // **we regenerate T for each block**

      $T \leftarrow \text{MakeTable}(B, i \times k, 0, k)$;

      for $0 \leq s < b_s$ **do**

        $j \longleftarrow start \times b_s + s$;

        $id \longleftarrow \text{ReadBits}(A, j, i \times k, k)$;

        add row id from T to row j of C;

  return C;

end

$t > 1$ 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;
end

Results: Multiplication

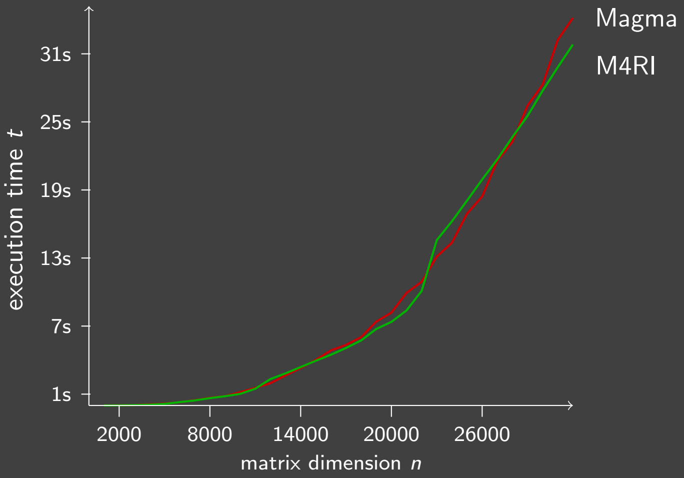


Figure: 2.66 Ghz Intel i7, 4GB RAM

Small Matrices

M4RI is efficient for large matrices, but not necessarily for small matrices.

	Thomé	M4RI
transpose	4.5097 μs	0.6352 μs
copy	0.2019 μs	0.2674 μs
add	0.2533 μs	0.2921 μs
mul	0.2535 μs	0.4472 μs

Table: 64×64 matrices (`matops.c`)

Note

One performance bottleneck is that our matrix structure is much more complicated than Emmanuel's.

Results: Multiplication Revisited

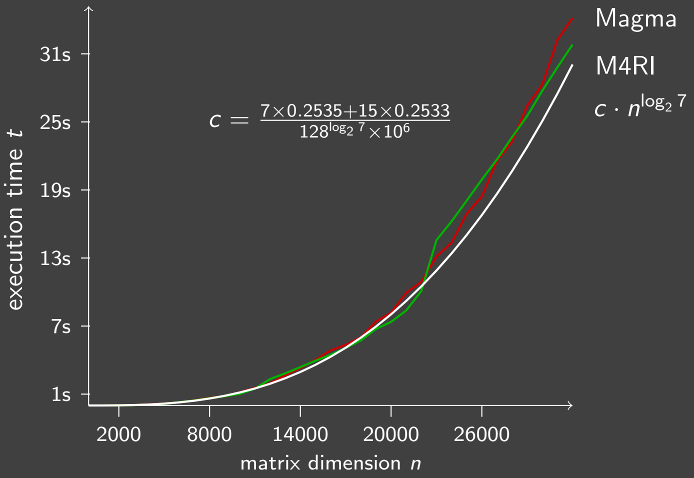
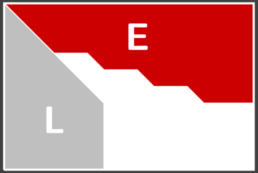


Figure: 2.66 Ghz Intel i7, 4GB RAM

Outline

- M4RI
  - Multiplication
  - Elimination
  - Projects
- M4RIE
  - Introduction
  - Newton-John Tables
  - Karatsuba Multiplication
  - Results

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.
Fast gaussian elimination and the PLE decomposition.
in preparation, 30 pages, 2011.

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

Write

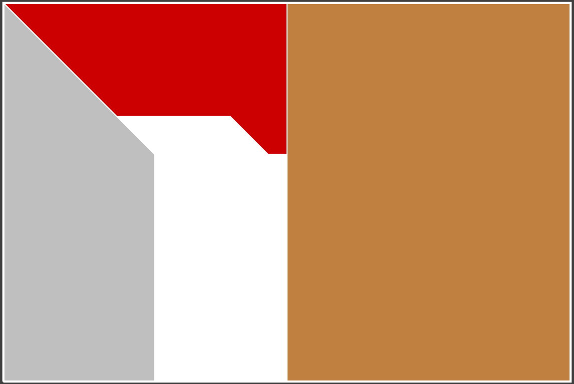
$$A = \begin{pmatrix} A_W & A_E \end{pmatrix} = \begin{pmatrix} A_{NW} & A_{NE} \\ A_{SW} & A_{SE} \end{pmatrix}$$

Main steps:

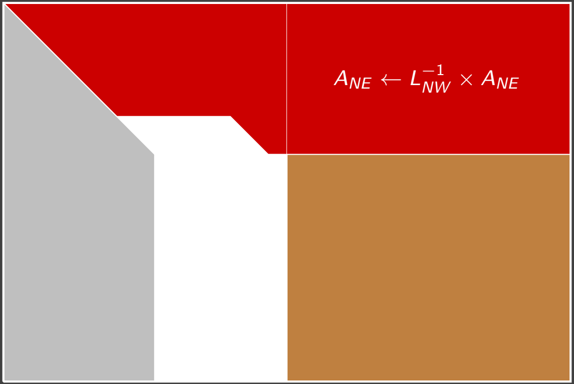
1. Call PLE on A_W
2. Apply row permutation to A_E
3. $L_{NW} \leftarrow$ the lower left triangular matrix in A_{NW}
4. $A_{NE} \leftarrow L_{NW}^{-1} \times A_{NE}$
5. $A_{SE} \leftarrow A_{SE} + A_{SW} \times A_{NE}$
6. Call PLE on A_{SE}
7. Apply row permutation to A_{SW}
8. Compress L

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

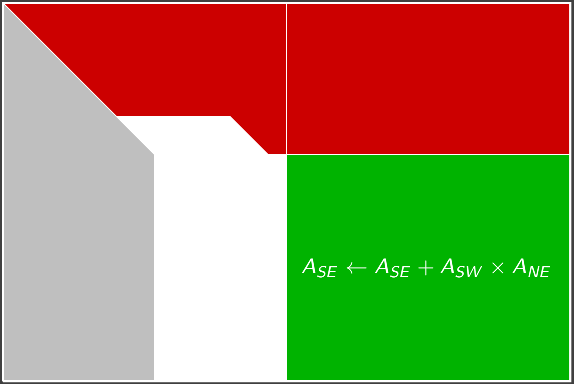
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ III



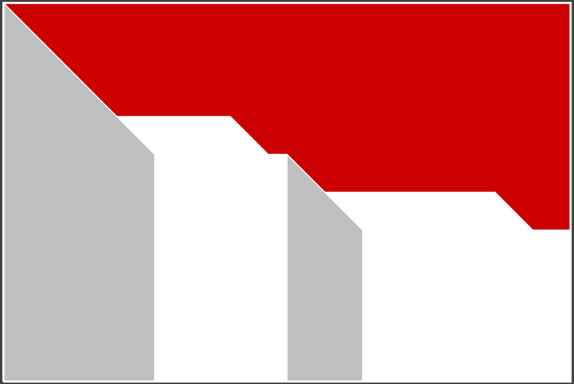
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ IV



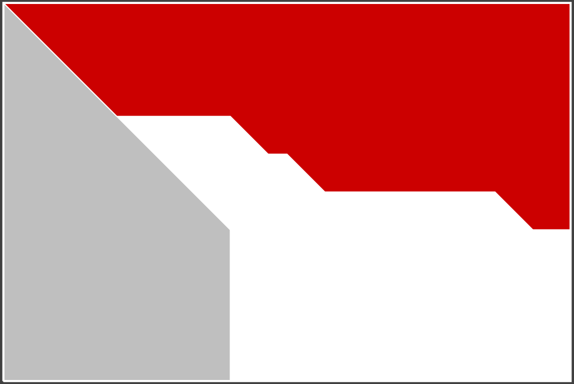
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ V



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



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

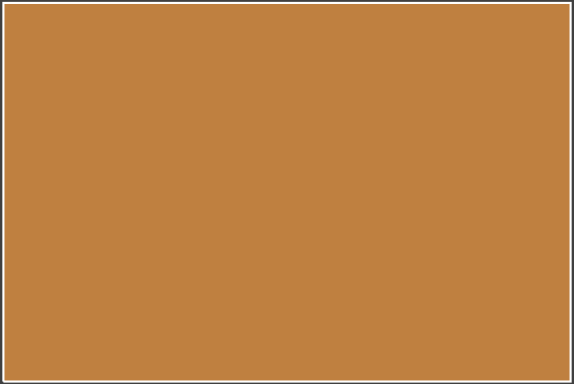


Block Iterative PLE Decomposition I

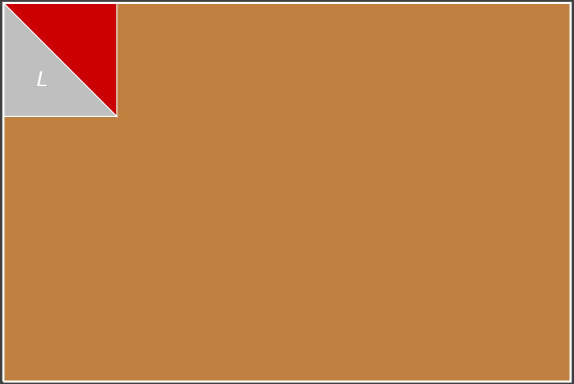
We need an efficient base case for PLE Decomposition

- 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 $\mathcal{O}\left(n^3 / \log n\right)$

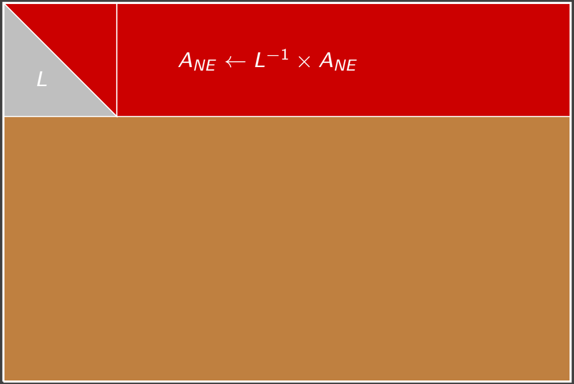
Block Iterative PLE Decomposition II



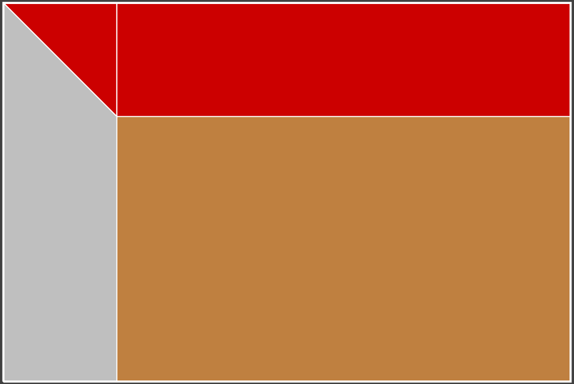
Block Iterative PLE Decomposition III



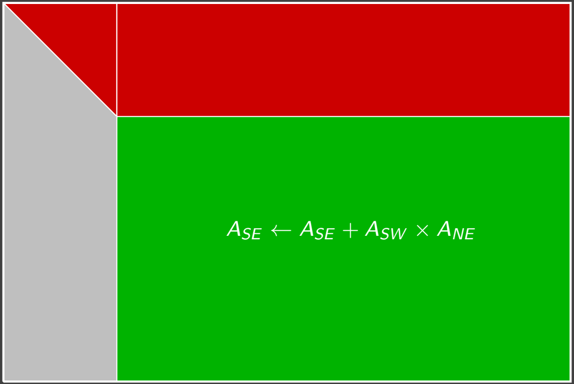
Block Iterative PLE Decomposition IV



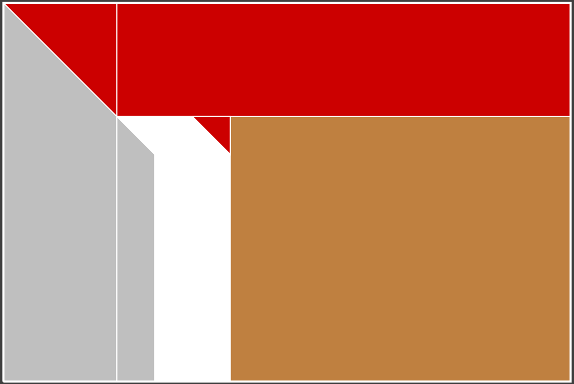
Block Iterative PLE Decomposition V



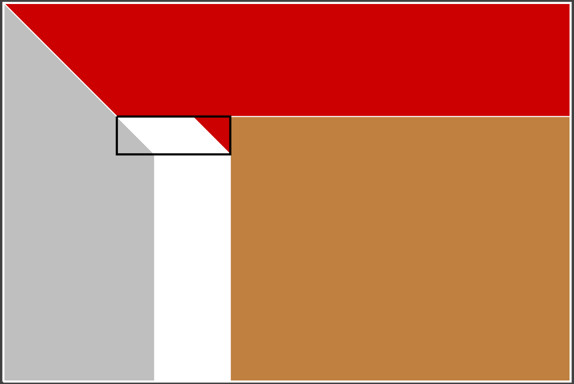
Block Iterative PLE Decomposition VI



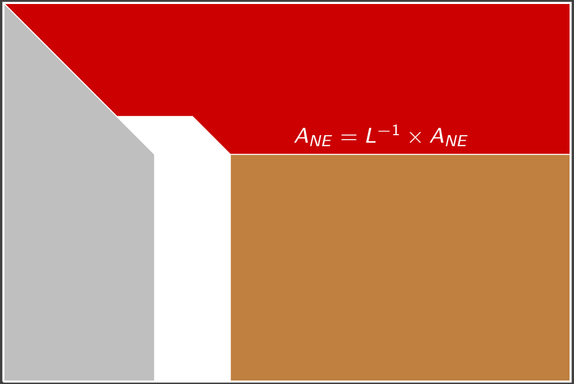
Block Iterative PLE Decomposition VII



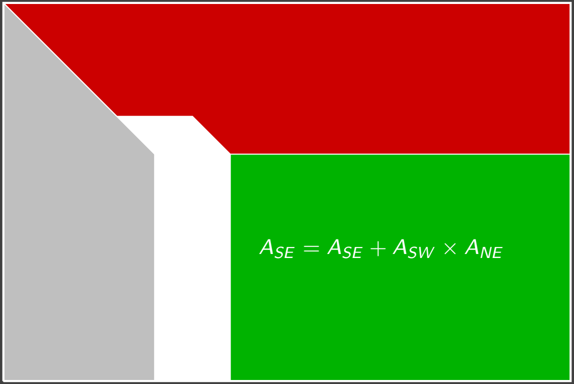
Block Iterative PLE Decomposition VIII



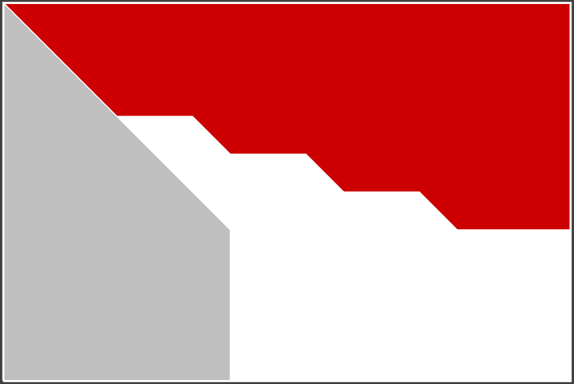
Block Iterative PLE Decomposition IX



Block Iterative PLE Decomposition X



Block Iterative PLE Decomposition XI



Results: Reduced Row Echelon Form

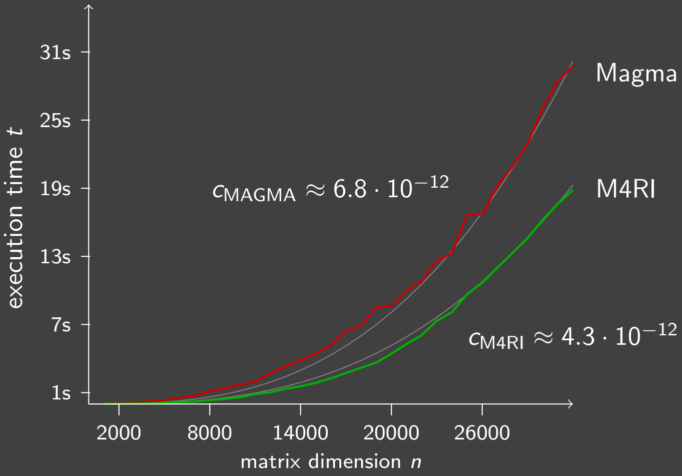


Figure: 2.66 Ghz Intel i7, 4GB RAM

Results: 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

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

Anybody got a 256GB RAM machine idlying around so that we can try $1,000,000 \times 1,000,000$ which should take about 20 hours on a single CPU? You know, for science!

Outline

- M4RI
  - Multiplication
  - Elimination
  - Projects
- M4RIE
  - Introduction
  - Newton-John Tables
  - Karatsuba Multiplication
  - Results

Sensitivity to Sparsity

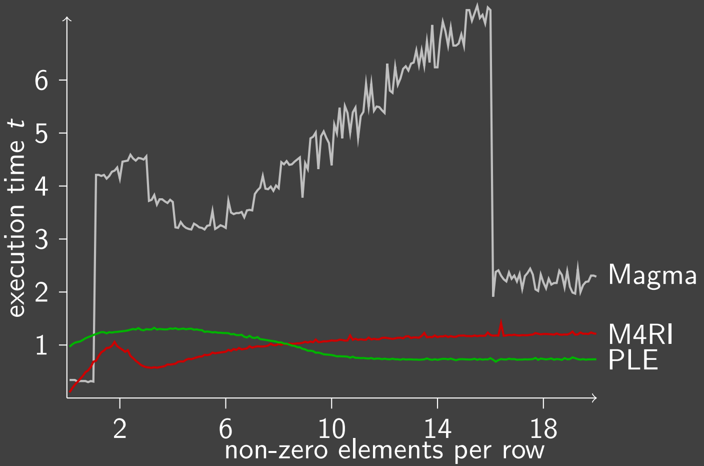
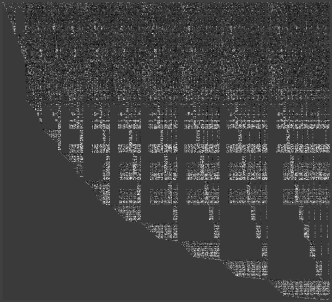


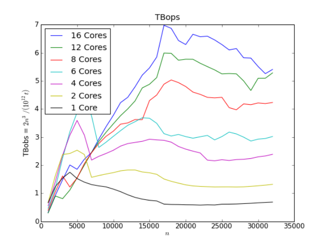
Figure: Gaussian elimination of $10,000 \times 10,000$ matrices on Intel 2.33GHz Xeon E5345 comparing Magma 2.17-12 and M4RI 20111004.

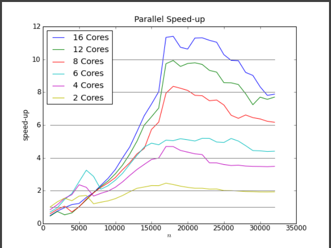
Linear Algebra for Gröbner Basis



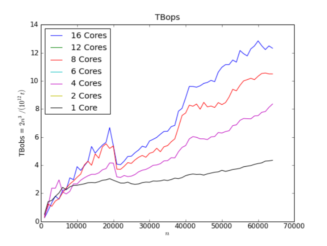
Problem	matrix dimensions	density	PLE	M4RI	GB
HFE 25 matrix 5 (5.1M)	12307 x 13508	0.07600	1.03	0.59	0.81
HFE 30 matrix 5 (16M)	19907 x 29323	0.06731	4.79	2.70	4.76
HFE 35 matrix 5 (37M)	29969 x 55800	0.05949	19.33	9.28	19.51
Mutant matrix (39M)	26075 x 26407	0.18497	5.71	3.98	2.10
random n=24, m=26 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

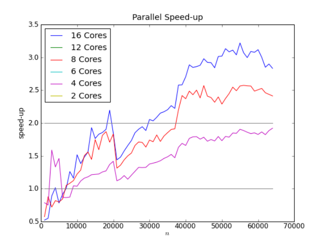
Multi-core Support





M4RI BOpS & Speed-up





PLE BOpS & Speed-up

GF(2) on GFX

Tabelle 3.12: Zeiten auf der GeForce GTX 295 und GeForce GTX 480.

Matrixgröße	GeForce GTX 295	GeForce GTX 480
9.984 x 10.240	0,9 Sek.	1,2 Sek.
16.384 x 16.384	2,47 Sek.	2,9 Sek.
20.000 x 20.480	4,63 Sek.	4,63 Sek.
32.000 x 32.768	13,3 Sek.	12,2 Sek.
64.000 x 65.536	-	70,74 Sek.

Tabelle 3.13: Zeiten auf der CPU [6].

Matrix Dimension	M4RI/M4RI 20090105[1]	M4RI/M4RI 20100817[2]
10.000 x 10.000	1,532	1,050
16.384 x 16.384	6,597	3,890
20.000 x 20.000	12,031	7,250
32.000 x 32.000	40,768	22,560
64.000 x 64.000	241,017	124,480

[1] 64- bit Debian/GNU Linux, 2.33 Ghz Core2Duo (Macbook Pro, 2nd. Gen.)
[2] 64- bit Debian/GNU Linux, 2.6 Ghz Intel i7 (Macbook Pro 6,2)

Denise Demirel
Effizientes Lösen linearer Gleichungssysteme über GF(2) mit GPUs
Diplomarbeit, TU Darmstadt, September 2010

Outline

M4RI

- Multiplication
- Elimination
- Projects

M4RIE

- Introduction
- Newton-John Tables
- Karatsuba Multiplication
- Results

Motivation I

Your NTL patch worked perfectly for me first try. I tried more benchmarks (on Pentium-M 1.8Ghz):

```
[...] //these are for GF(2^8), malb
sage: n=1000; m=ntl.mat_GF2E(n,n,[ ntl.GF2E_random() for i in xrange(n^2) ])
sage: time m.echelon_form()
1000
Time: CPU 29.72 s, Wall: 43.79 s
```

This is pretty good; vastly better than what's was in SAGE by default, and way better than PARI. Note that MAGMA is much faster though (nearly 8 times faster):

```
[...]
 > n := 1000; A := MatrixAlgebra(GF(2^8),n)![Random(GF(2^8)) : i in [1..n^2]];
 > time E := EchelonForm(A);
Time: 3.440
```

MAGMA uses (1) [. . .] and (2) a totally different algorithm for computing the echelon form. [. . .] As far as I know, the MAGMA method is not implemented anywhere in the open source world. But I'd love to be wrong about that... or even remedy that.

– W. Stein in 01/2006 replying to my 1st non-trivial patch to Sage

Motivation II

The situation has not improved much in **2011**:

System	Time in *ms*
Sage 4.7.2	97,000
NTL 5.4.2	85,000
LinBox SVN + patches	460
GAP 4.412	210
Magma 2.15	13
this work	5.5

Table: Product of two dense $1,000 \times 1,000$ matrix over $\mathbb{F}_{2^2}$.

...an older version of our code will be in Sage 4.8.

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;
end

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;
end

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;
end

The idea IV

```
Input: A – m × n matrix
Input: B – n × k matrix
begin
    for 0 ≤ i < m do
        for 0 ≤ j < n do
            C_j ⟵ C_j + A_{j,i} × B_i; // expensive
    return C;
end
```

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;

end

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

Gaussian elimination & PLE decomposition

```
Input: A – m × n matrix
begin
    r ⟵ 0;
    for 0 ≤ j < n do
        for r ≤ 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 ⟵ multiplication table for row r of A;
            for r + 1 ≤ k < m do
                x ⟵ A_{k,j};
                A_k ⟵ A_k + T_x;
            r ⟵ r + 1;
        return r;
end
```

Outline

The idea

- Consider $\mathbb{F}_{2^2}$ with the primitive polynomial $f = x^2 + x + 1$.
- We want to compute $C = AB$.
- 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

Results: Multiplication I

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

Results: Multiplication II

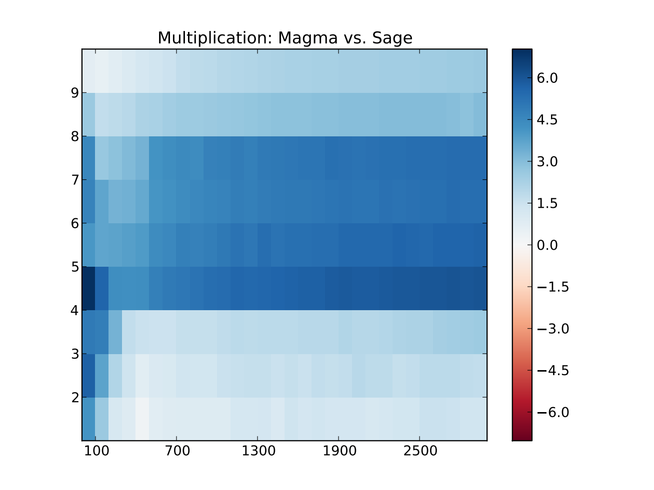


Figure: 2.66 Ghz Intel i7, 4GB RAM

Results: Reduced Row Echelon Forms I

e	Magma 2.15-10	GAP 4.4.12	M4RIE 6b24b839a46f
2	6.040s	162.658s	3.310s
3	14.470s	442.522s	5.332s
4	60.370s	502.672s	6.330s
5	659.030s	N/A	10.511s
6	685.460s	N/A	13.078s
7	671.880s	N/A	17.285s
8	840.220s	N/A	20.247s
9	1630.380s	N/A	260.774s
10	1631.350s	N/A	291.298s

Table: Elimination of $10,000 \times 10,000$ matrices

Results: Reduced Row Echelon Forms II

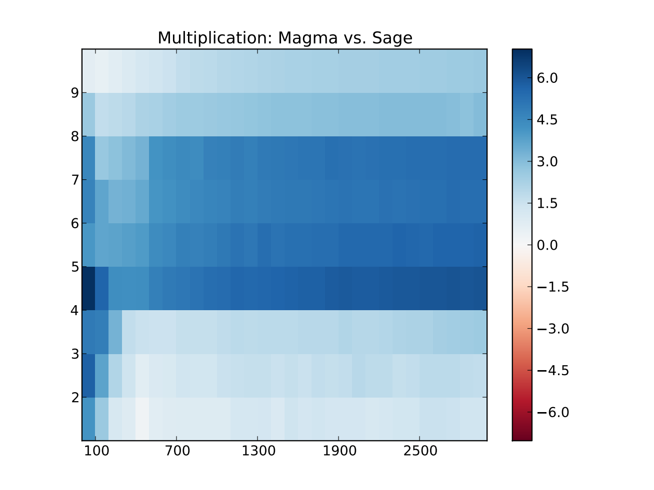


Figure: 2.66 Ghz Intel i7, 4GB RAM

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.

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.