The M4RI and M4RIE libraries – with a bias towards Gröbner basis computations.

Martin R. Albrecht (martinralbrecht@googlemail.com)

POLSYS Team, UPMC, Paris, France

Efficient Linear Algebra for Gröbner Basis Computations, Kaiserslautern, Germany

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
  - Gröbner Basis Computations
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance

Outline

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

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	**0**	⇓
0	0	**1**	
1	1	1	
	1	0	⇑

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

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"

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

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

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

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

```
1 begin
2     C ⟵ create an m × n matrix with all entries 0;
3     k ⟵ ⌊log n⌋;
4     for 0 ≤ i < (ℓ/k) do
          // create table of 2^k − 1 linear combinations
5         T ← MakeTable(B, i × k, 0, k);
6         for 0 ≤ j < m do
              // read index for table T
7             id ⟵ ReadBits(A, j, i × k, k);
8             add row id from T to row j of C;
9     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

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

Cache Friendly M4RM II

```
1  begin
2      C ⟵ create an m × n matrix with all entries 0;
3      for 0 ≤ start < m/b_s do
4          for 0 ≤ i < (ℓ/k) do
               // we regenerate T for each block
5              T ← MakeTable(B, i × k, 0, k);
6              for 0 ≤ s < b_s do
7                  j ⟵ start × b_s + s;
8                  id ⟵ ReadBits(A, j, i × k, k);
9                  add row id from T to row j of C;
10     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

```
1  begin
2      C ⟵ create an m × n matrix with all entries 0;
3      for 0 ≤ i < (ℓ/(2k)) do
4          T_0 ← MakeTable(B, i × 2k, 0, k);
5          T_1 ← MakeTable(B, i × 2k + k, 0, k);
6          for 0 ≤ j < m do
7              id_0 ⟵ ReadBits(A, j, i × 2k, k);
8              id_1 ⟵ ReadBits(A, j, i × 2k + k, k);
9              add row id_0 from T_0 and row id_1 from T_1 to row j of C;
10     return C;
```

Performance: Multiplication

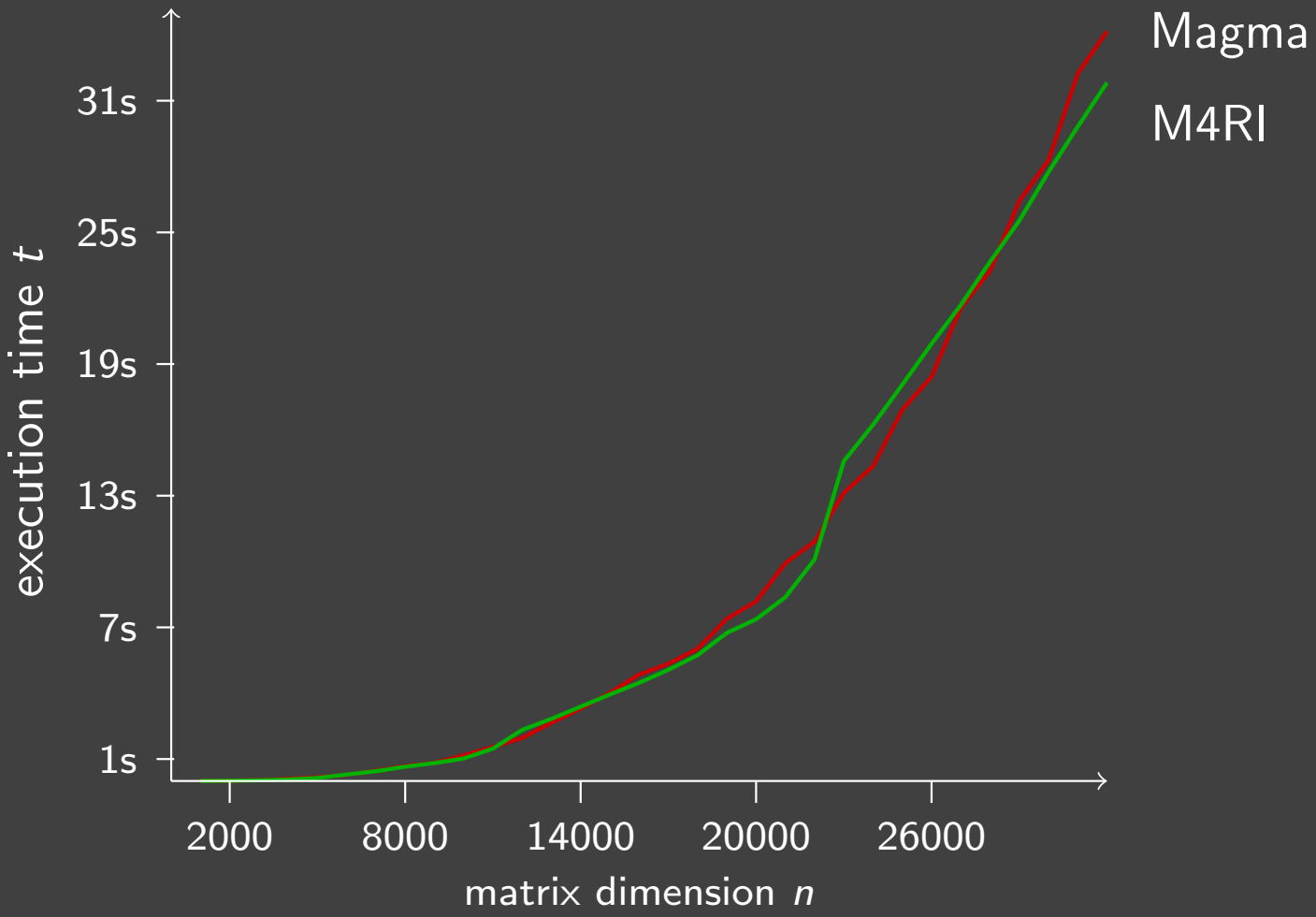


Figure: 2.66 Ghz Intel i7, 4GB RAM

Work-in-Progress: Small Matrices

M4RI is efficient for large matrices, but not so for small matrices. But there is work under way by Carlo Wood to fix this.

	Emmanuel	M4RI
transpose	4.9064 μs	5.3642 μs
copy	0.2019 μs	0.2674 μs
add	0.2533 μs	**0.7503** 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

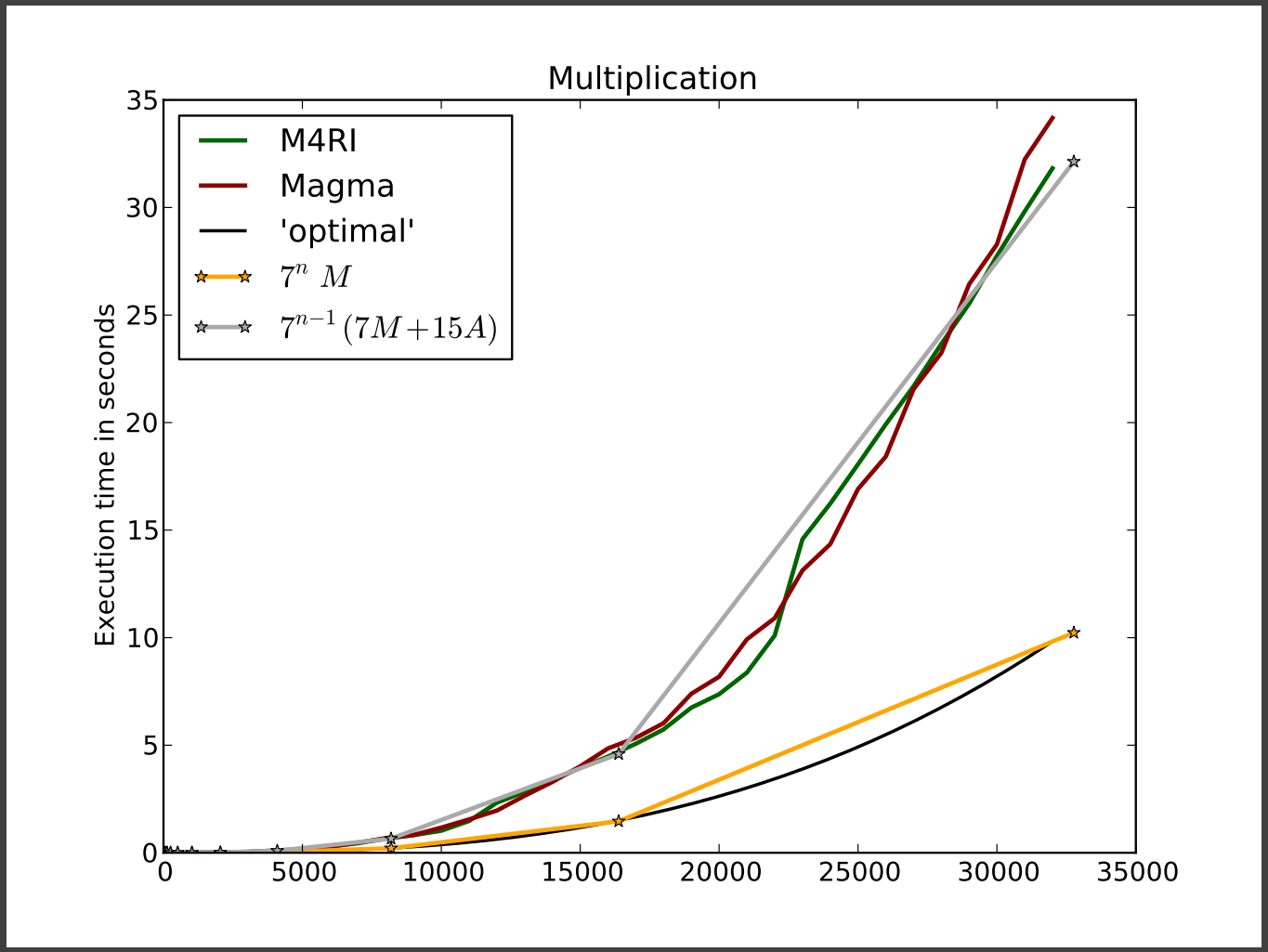
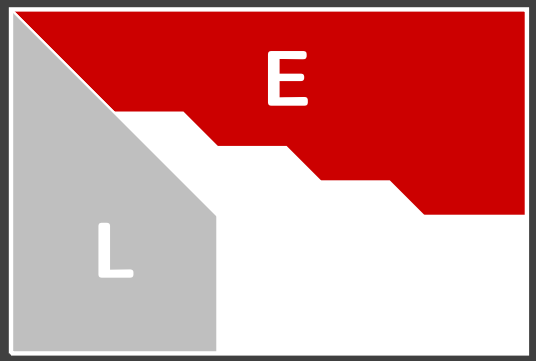


Figure: 2.66 Ghz Intel i7, 4GB RAM

Outline

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

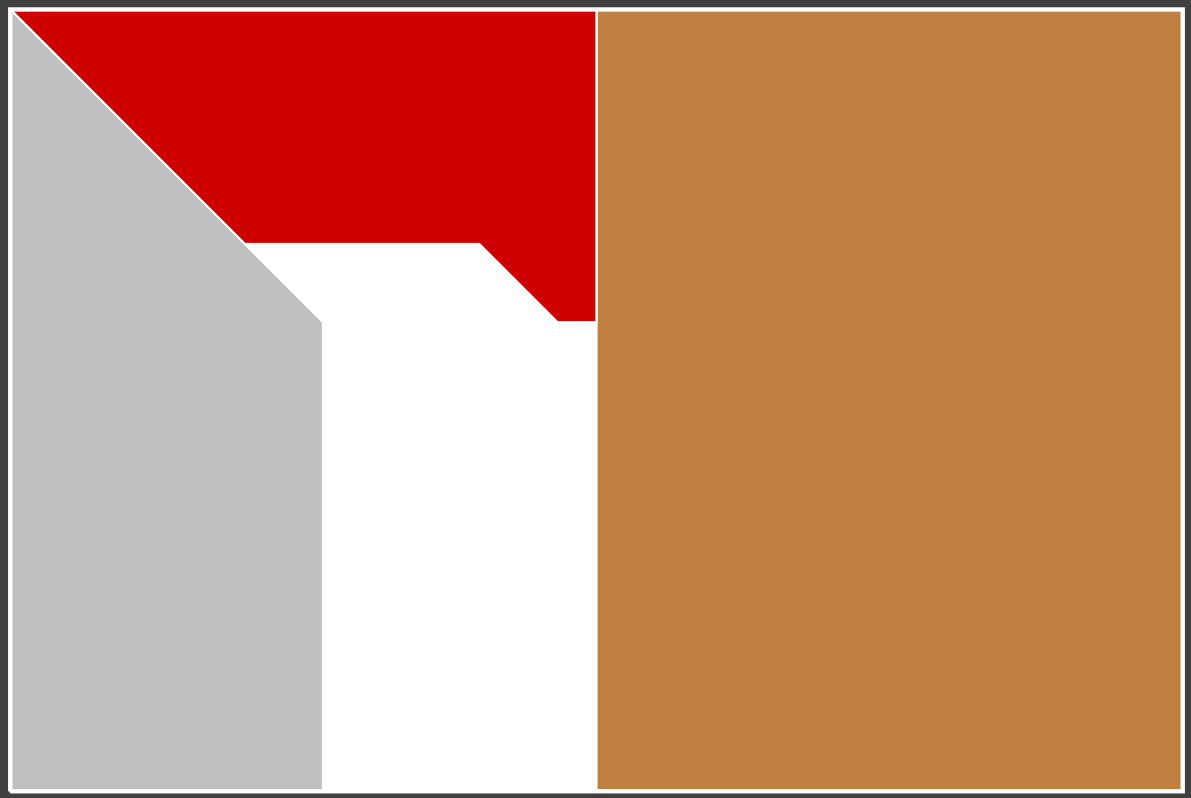
From the PLE decomposition we can

- 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.

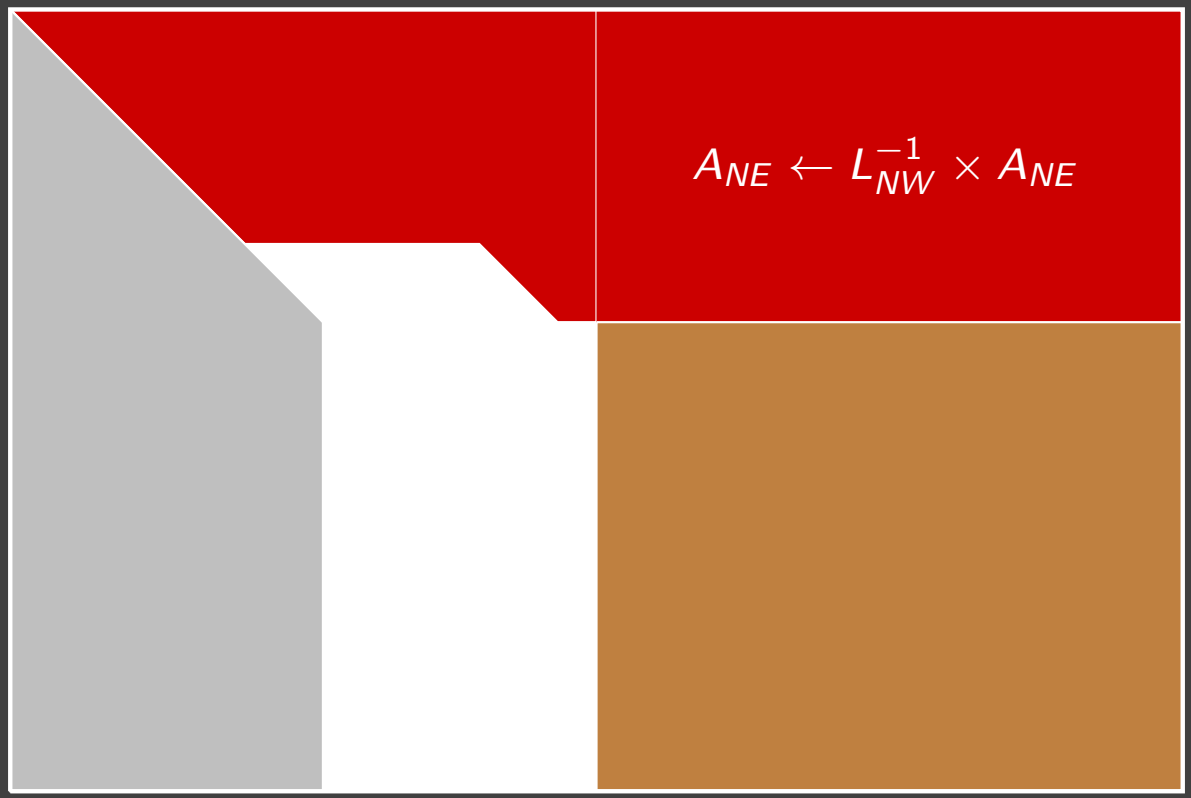
C.-P. Jeannerod, C. Pernet, and A. Storjohann.
Rank-profile revealing Gaussian elimination and the CUP matrix decomposition.
arXiv:1112.5717, 35 pages, 2012.

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

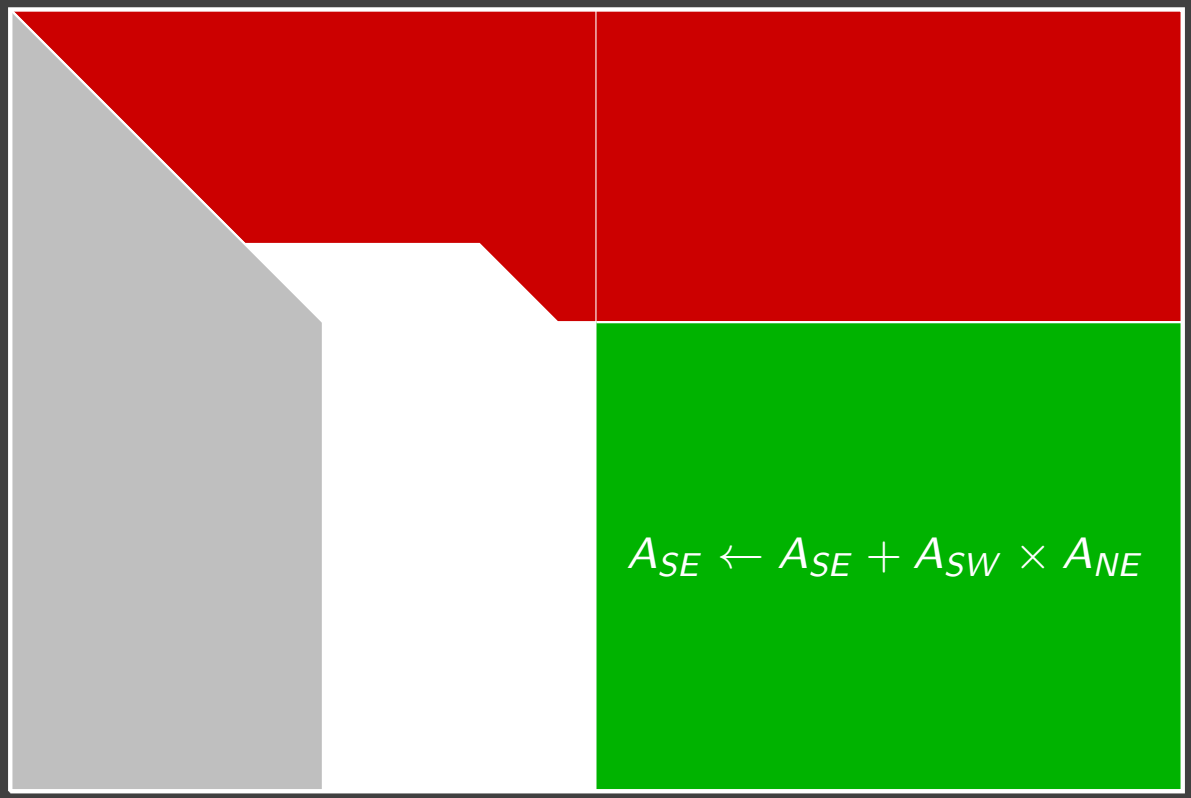
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ II



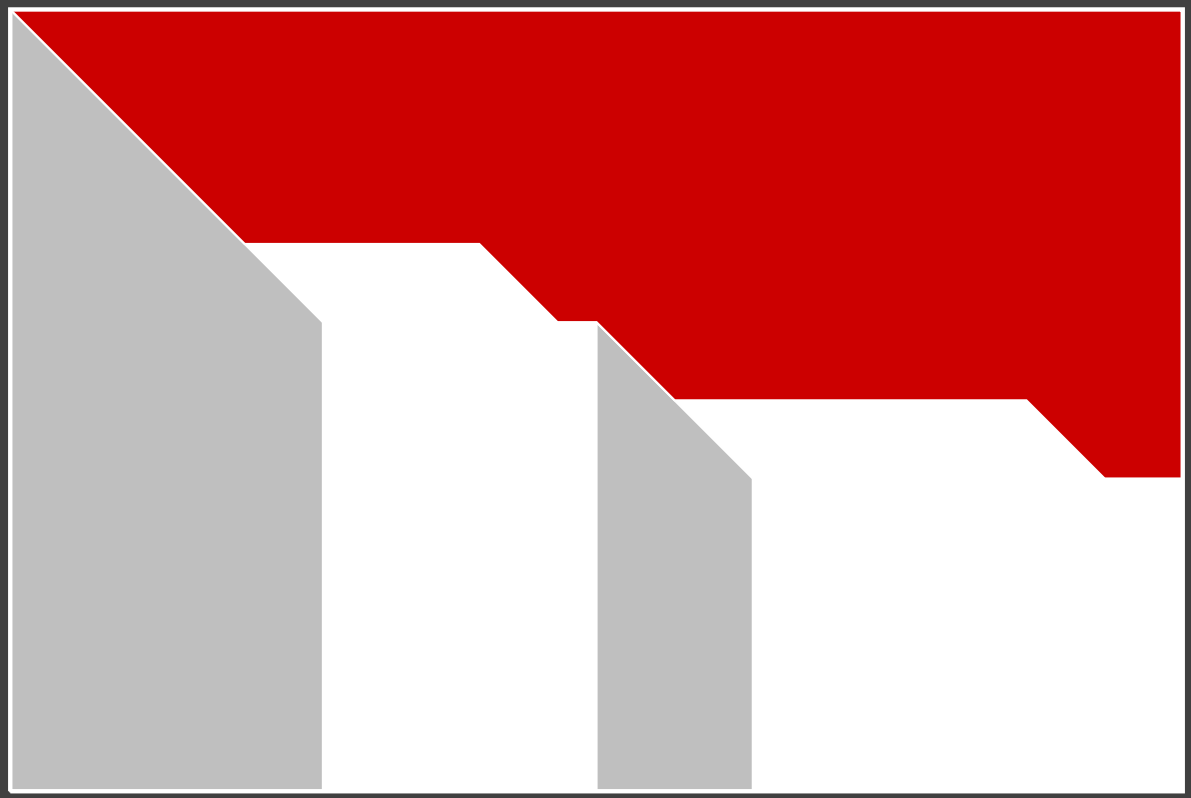
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ III



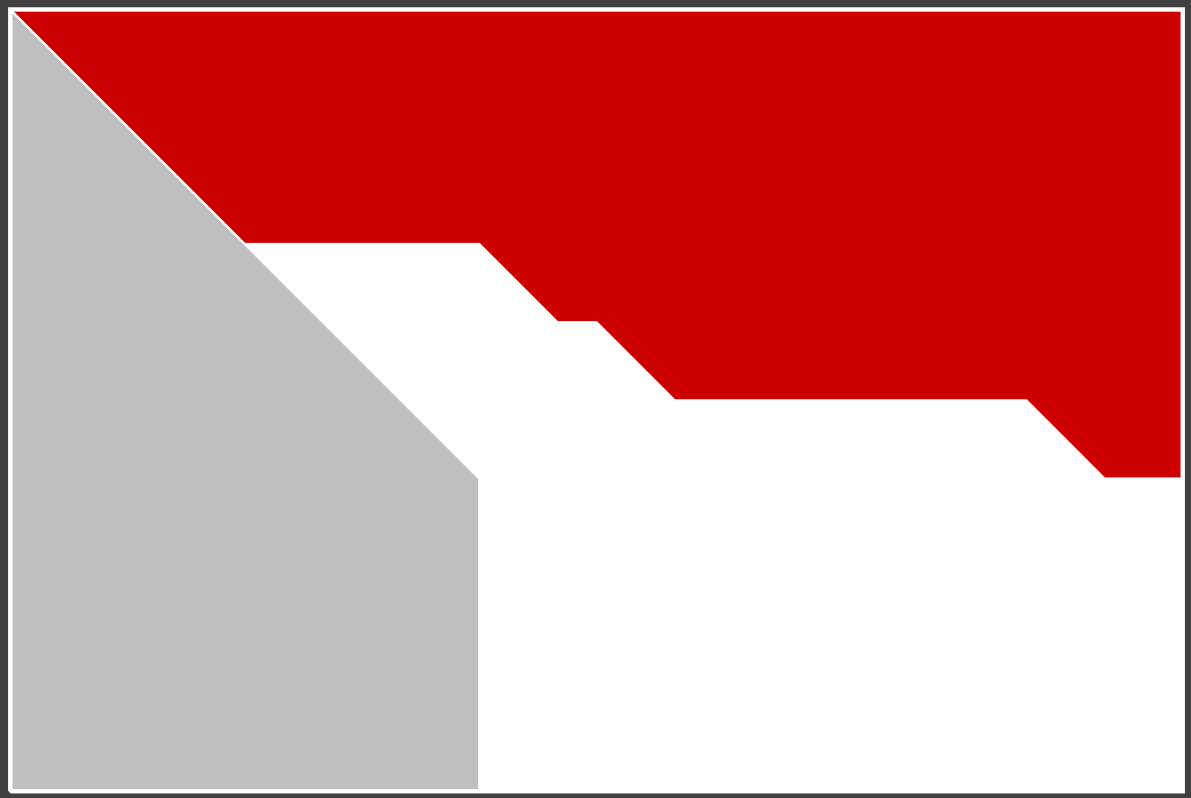
Block Recursive PLE Decomposition $\mathcal{O}(n^\omega)$ IV



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



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

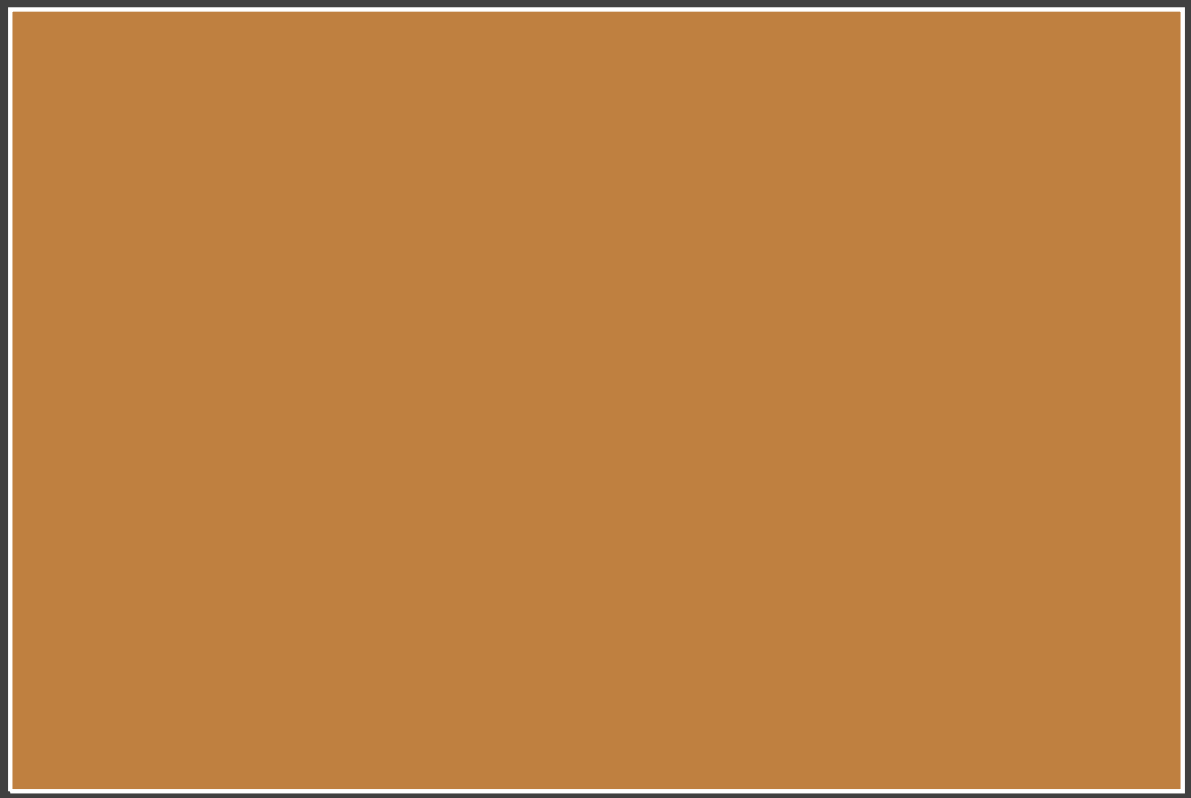


Block Iterative PLE Decomposition I

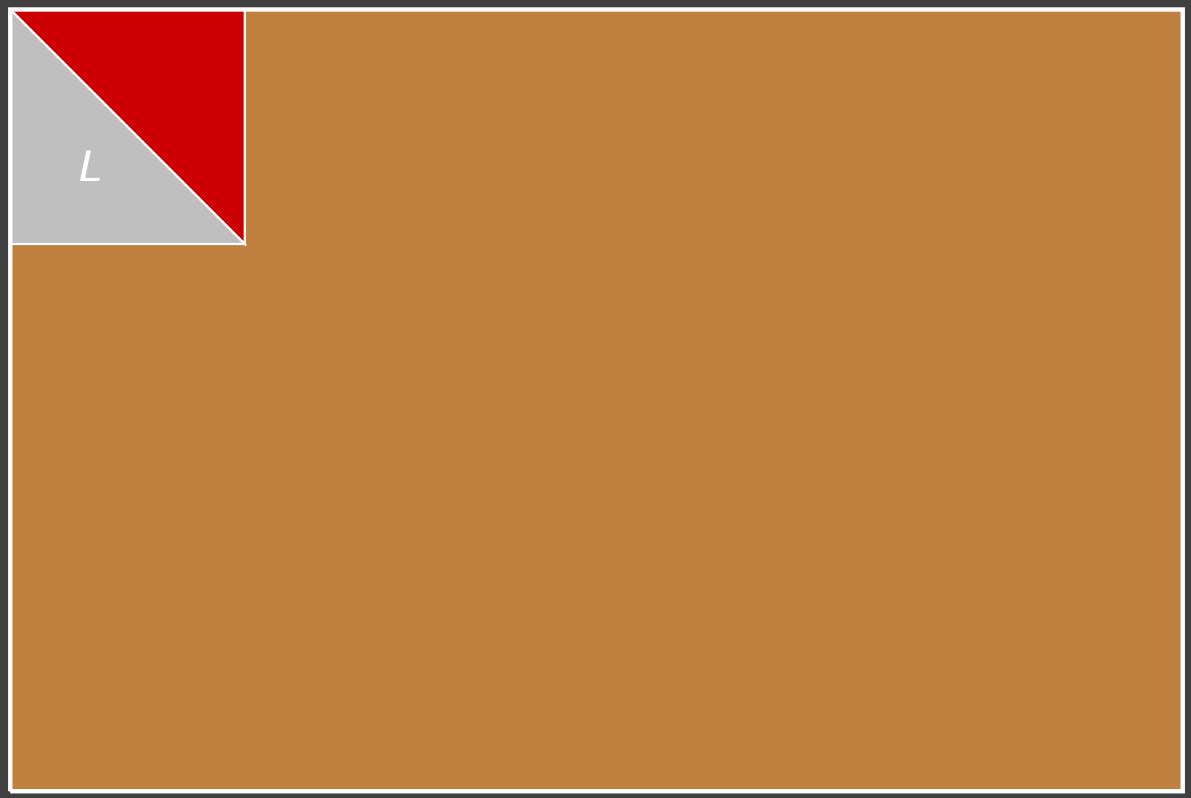
We need an efficient base case for PLE Decomposition

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

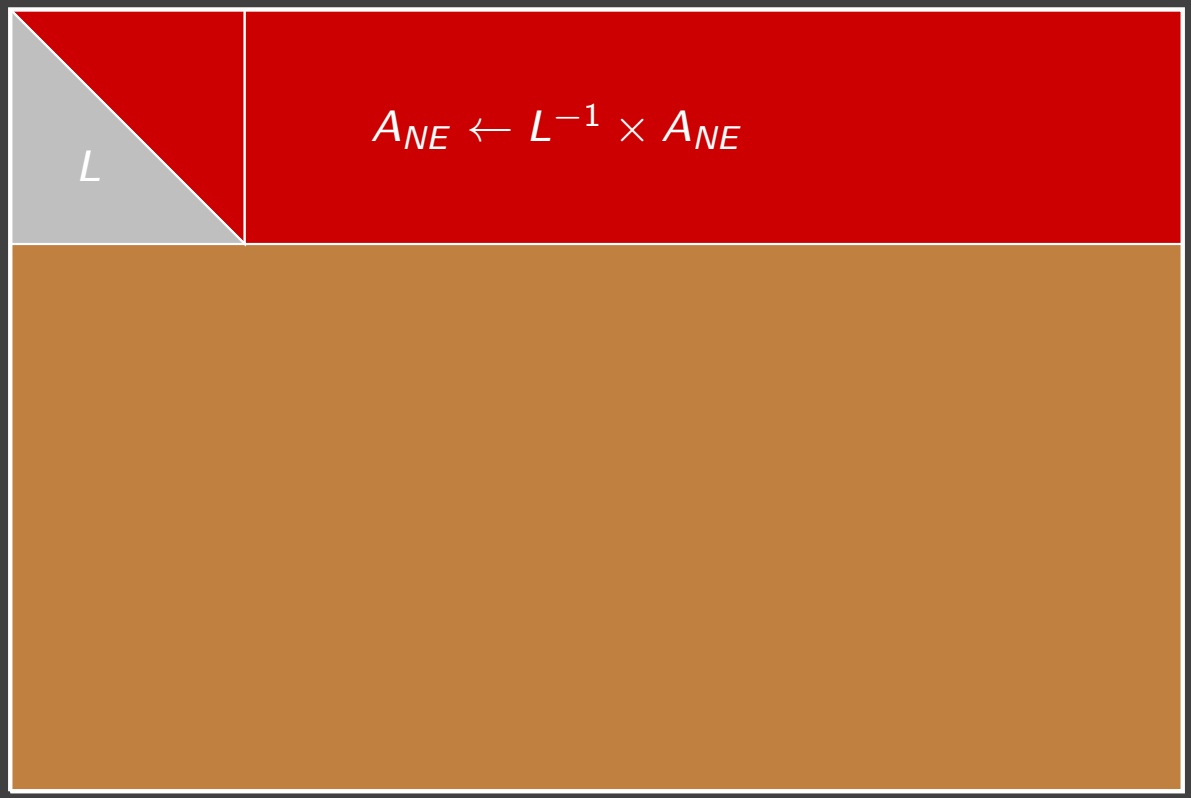
Block Iterative PLE Decomposition II



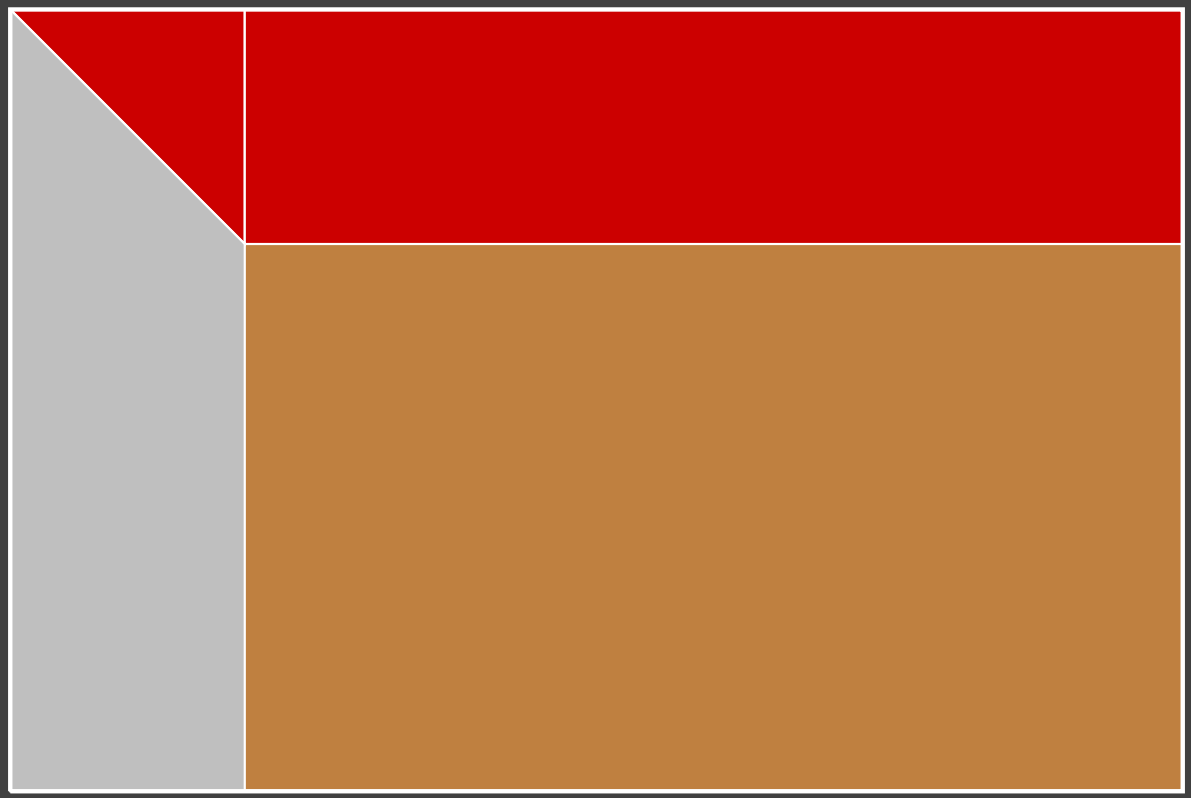
Block Iterative PLE Decomposition III



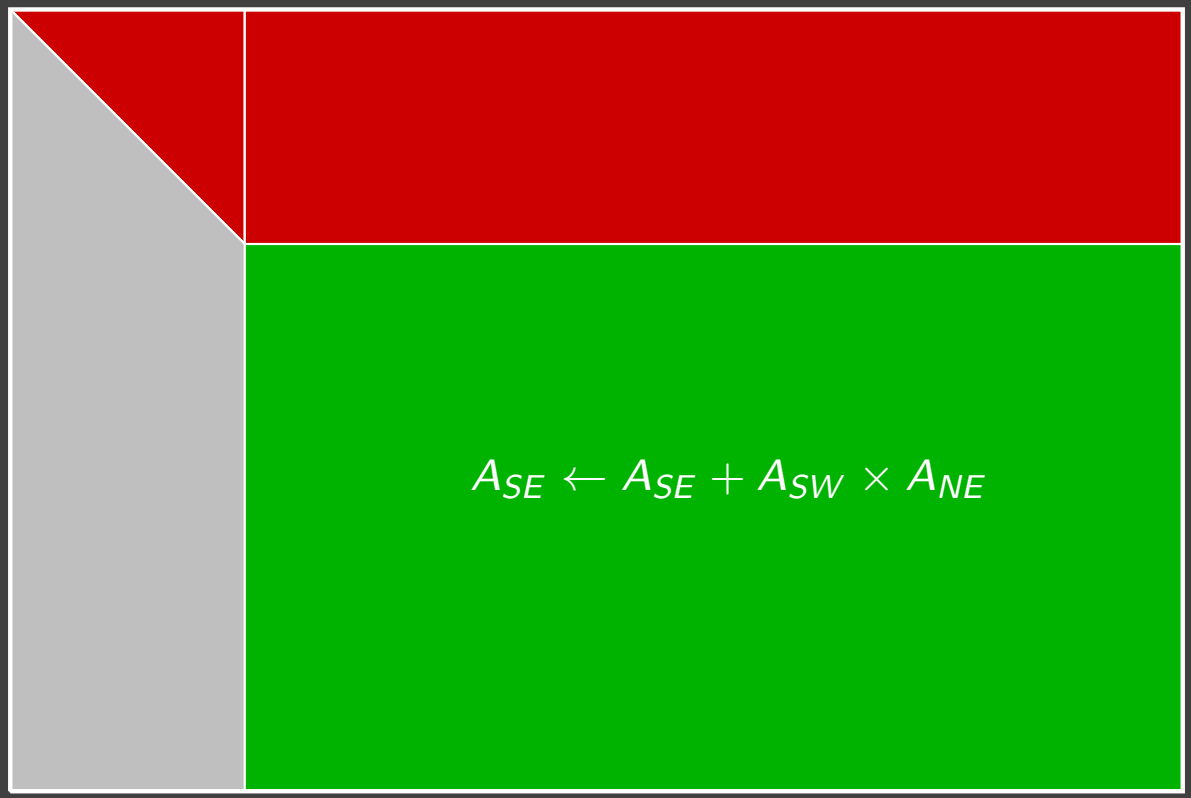
Block Iterative PLE Decomposition IV



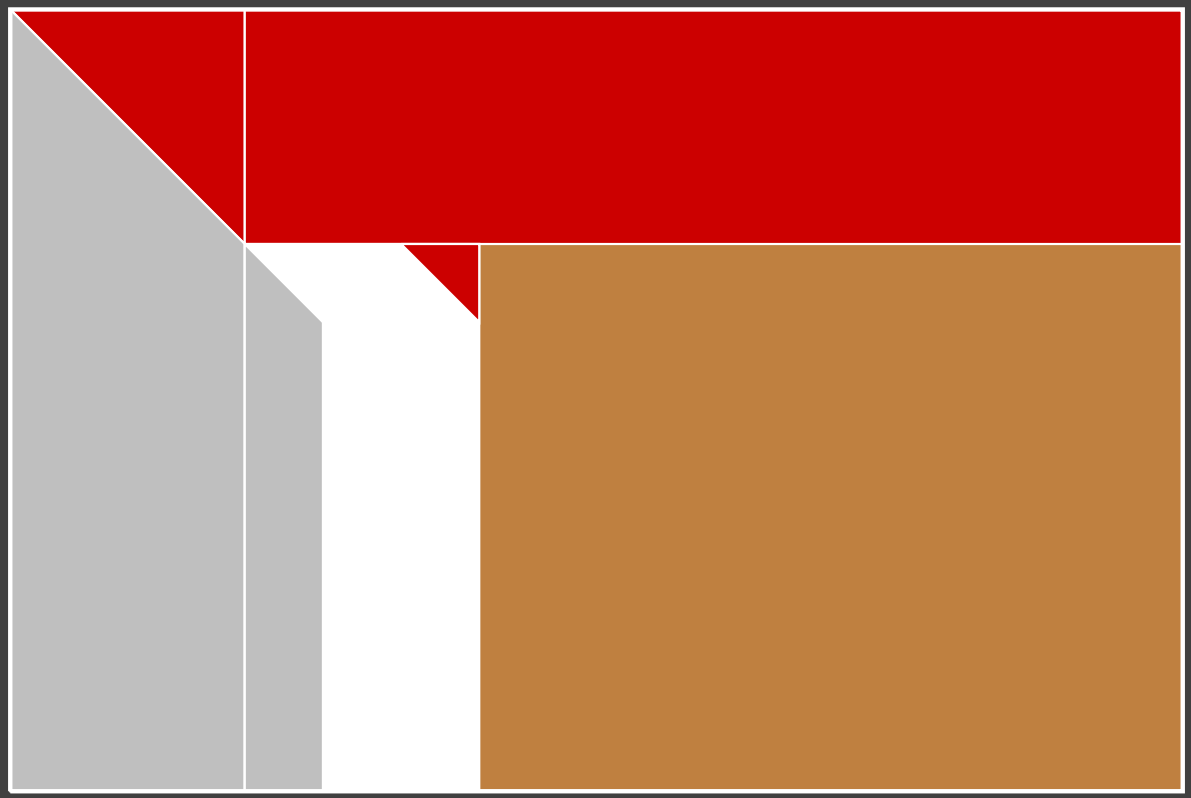
Block Iterative PLE Decomposition V



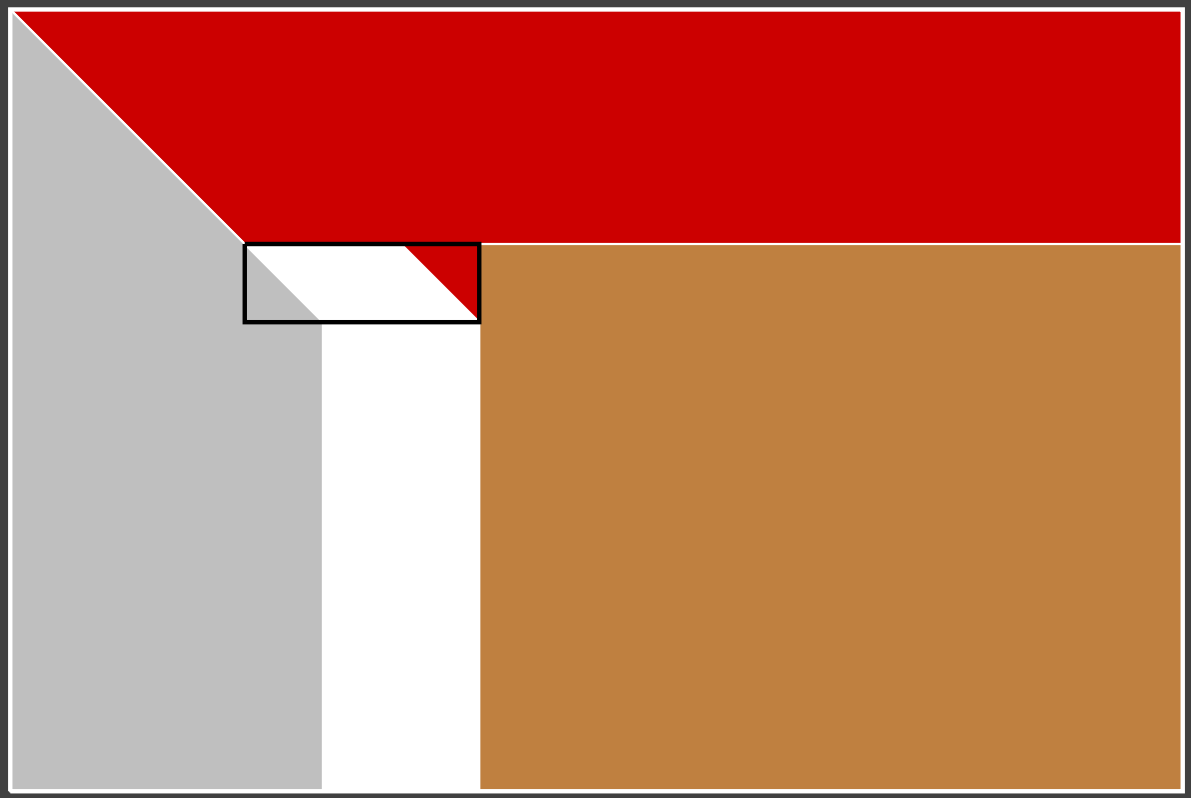
Block Iterative PLE Decomposition VI



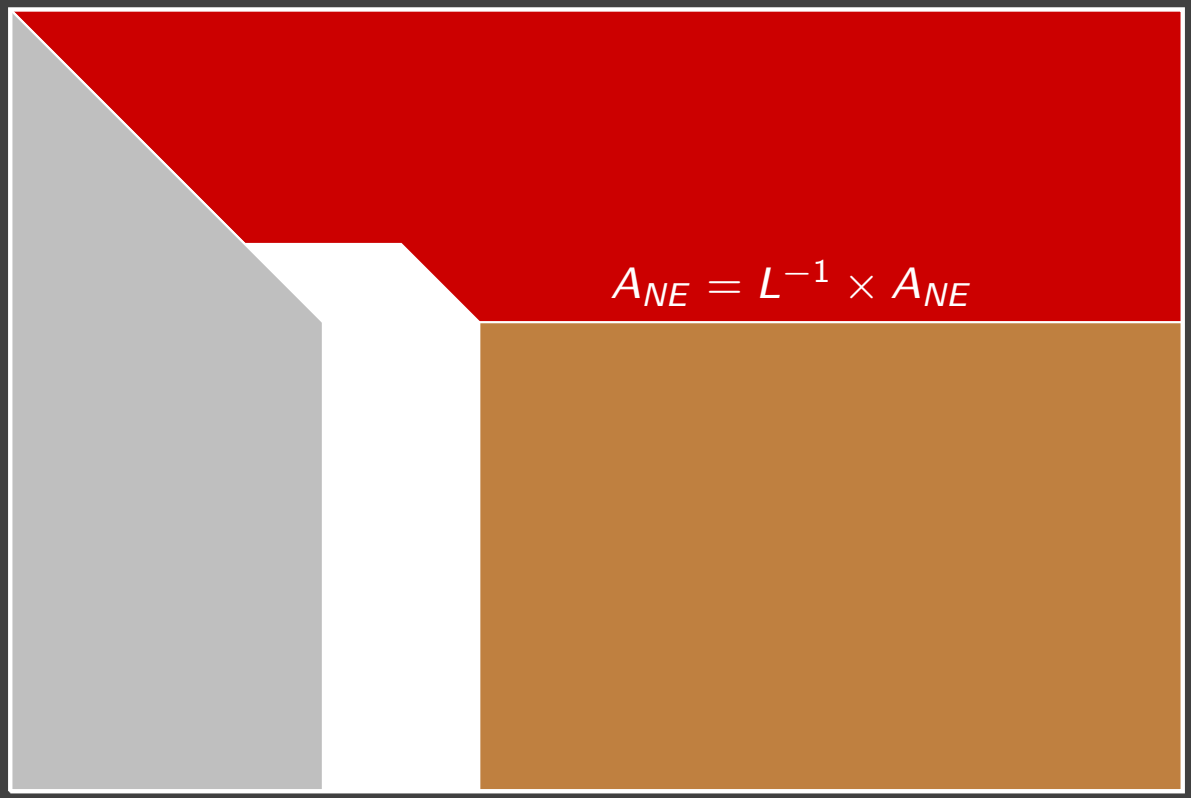
Block Iterative PLE Decomposition VII



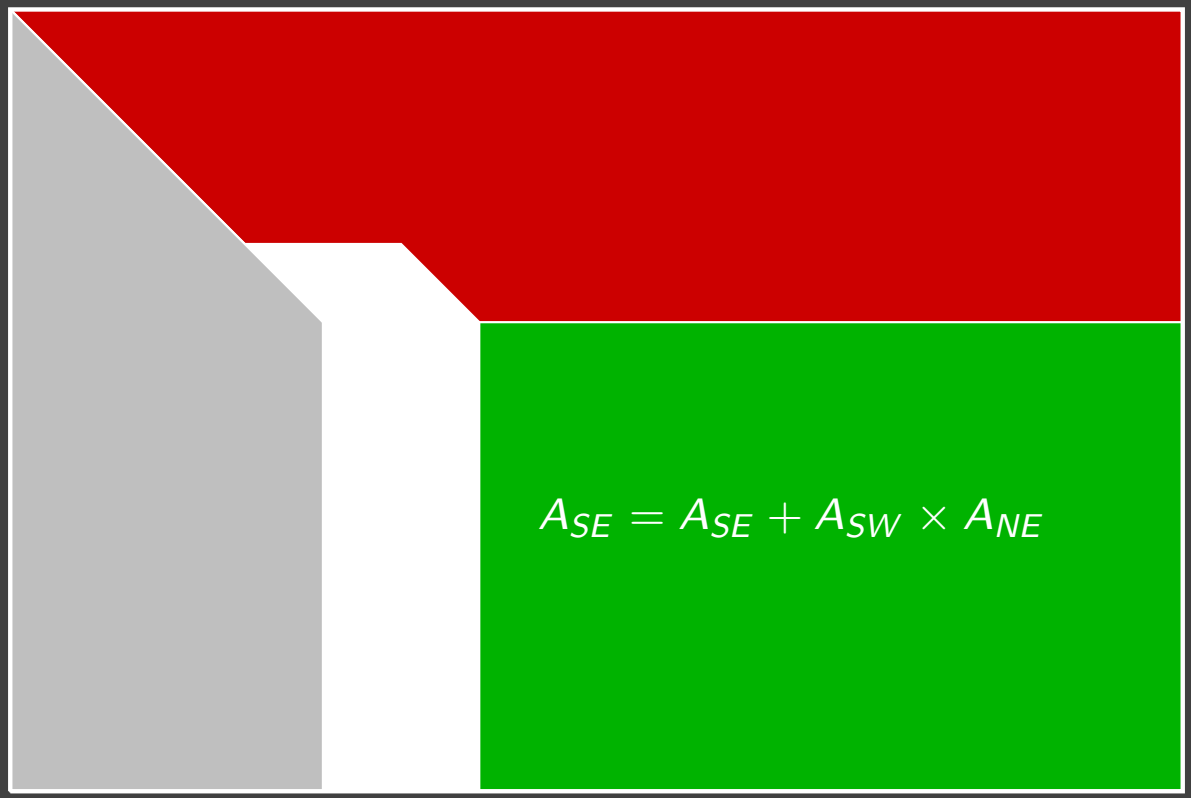
Block Iterative PLE Decomposition VIII



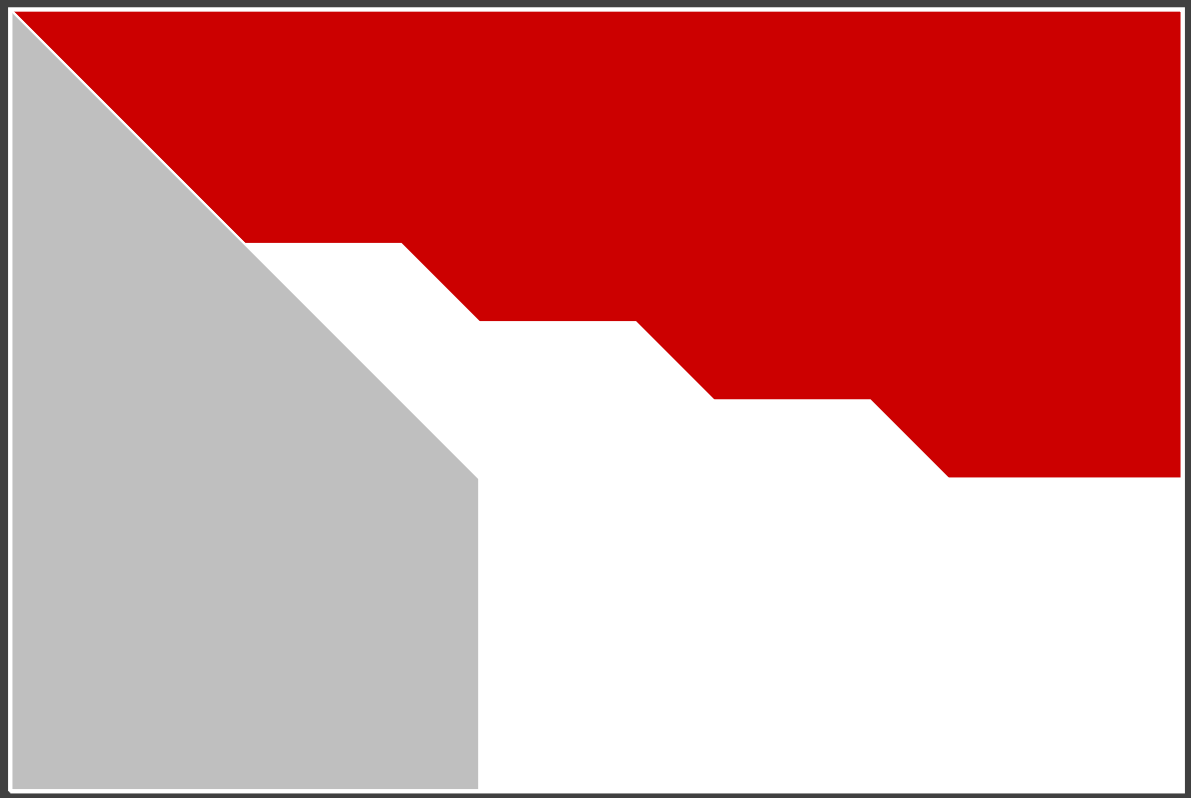
Block Iterative PLE Decomposition IX



Block Iterative PLE Decomposition X



Block Iterative PLE Decomposition XI



Performance: Reduced Row Echelon Form

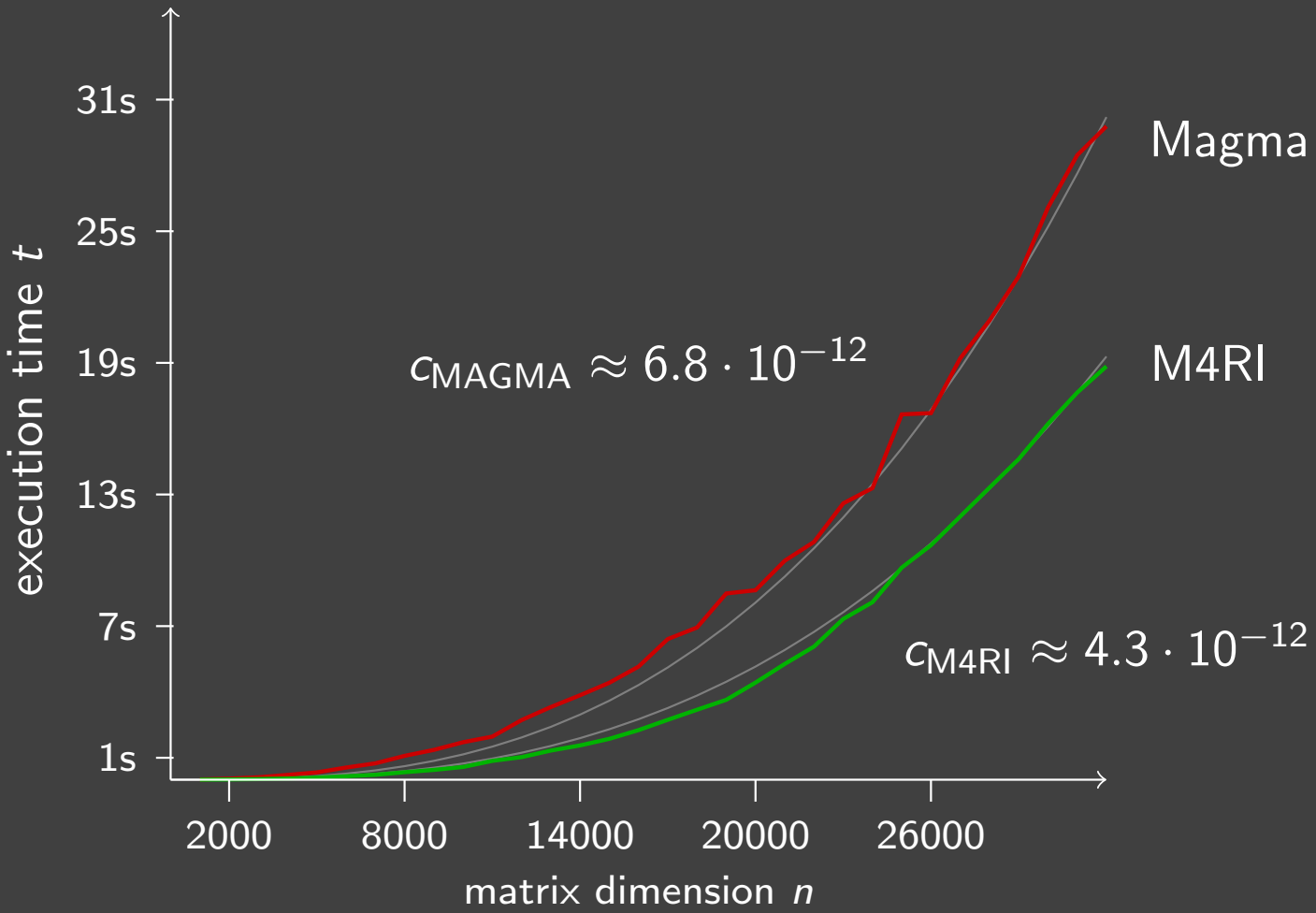


Figure: 2.66 Ghz Intel i7, 4GB RAM

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

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

Outline

Caveat: Sensitivity to Sparsity I

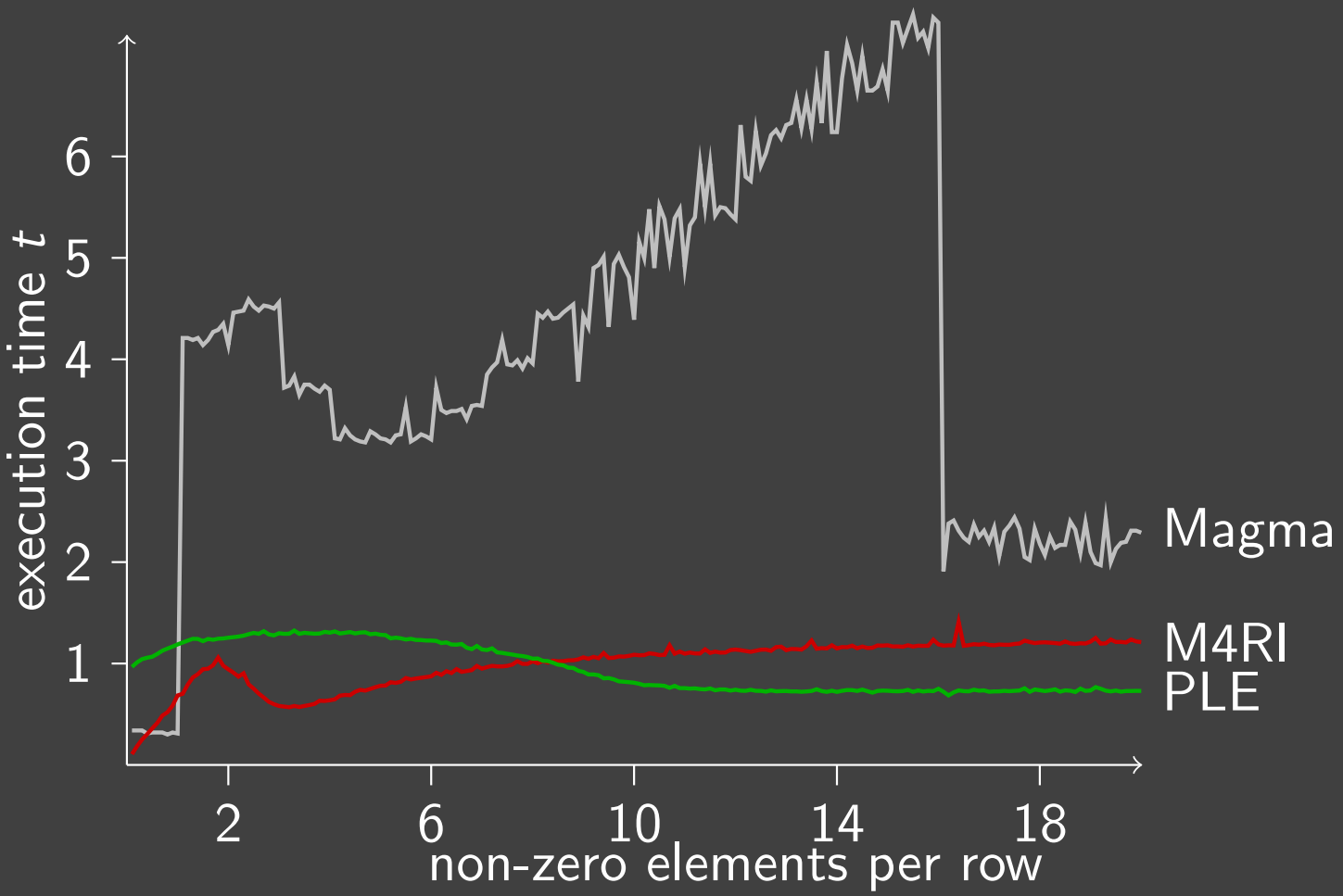


Figure: Gaussian elimination of $10,000 \times 10,000$ matrices on Intel 2.33GHz Xeon E5345 comparing Magma 2.17-12 and M4RI 20111004.

Caveat: Sensitivity to Sparsity II

Why is that?

1. These algorithms are designed to be *rank sensitive* and they swap rows and columns. The latter is quite expensive.
2. Searching for a pivot often requires to deal with single bits instead of complete words.
3. We loose cache efficiency by doing less operations on those rows already in cache.

New Project I

- New repository for code optimised for Gröbner basis computations,
- supports timing of three algorithms (PLE, M4RI, GB) and
- reads 1-bit PNG images during execution,
- plotting of intermediate matrices.

New Project II

```
groebner

REQUIRED
  filename (1-bit PNG or JCF format)

OPTIONAL
  -w filename     write the processed matrix to a 1-bit PNG
  -c level        compression level for output
  -e algo         echelonize using algorithm 'algo'
  -r              compute the rank
  -v              write intermediate matrices when executing
                  gb algorithm
```

https://bitbucket.org/malb/m4ri-groebner

Algorithm I

The implemented algorithm is very simple variant of [FL10, BD07]

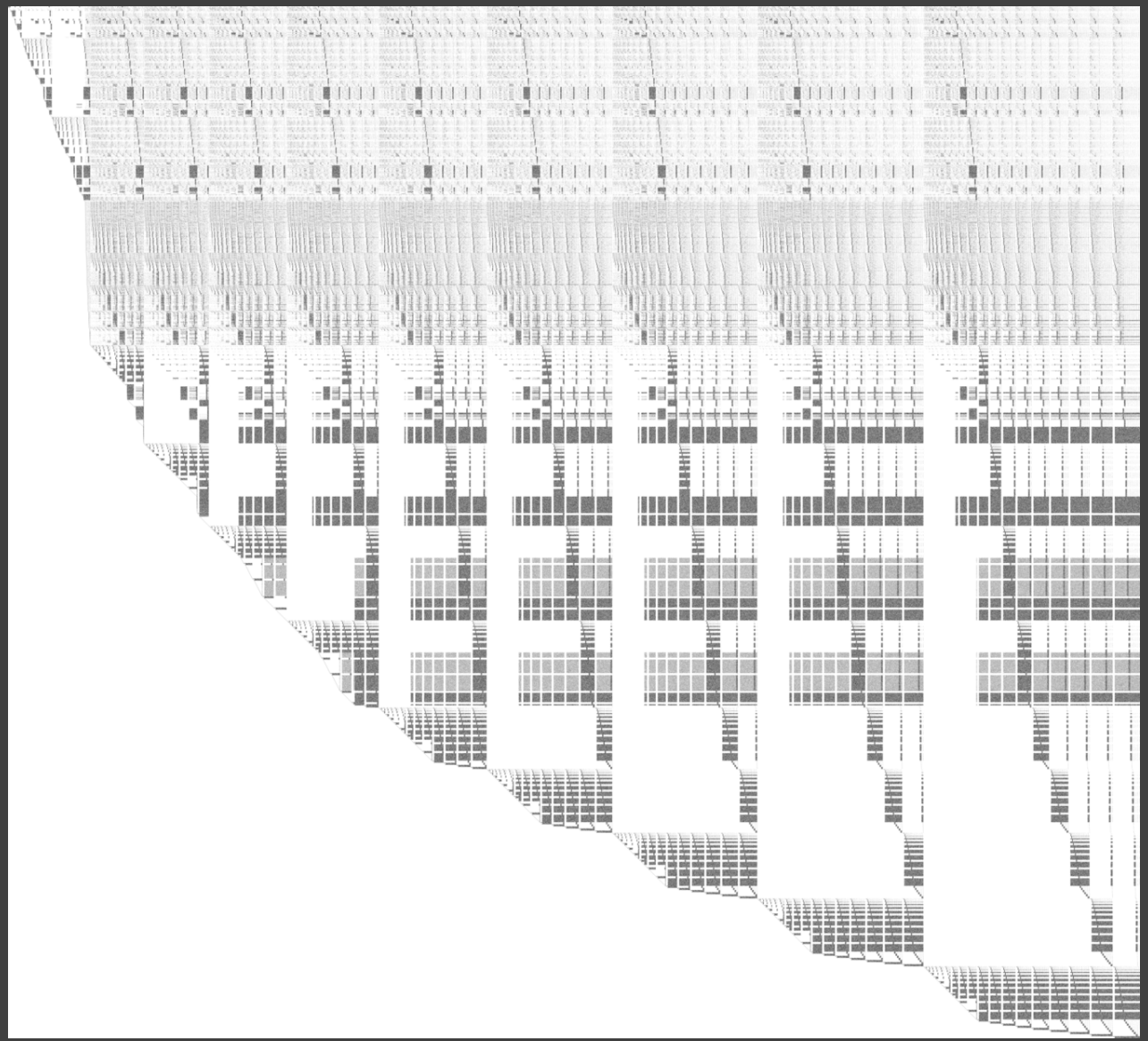
J.-C. Faugère and S. Lachartre.
Parallel Gaussian Elimination for Gröbner bases computations in finite fields.
In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 89–97, 2010.

M. Brickenstein and A. Dreyer.
PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials.
In *Electronic Proceedings of MEGA 2007*, 2007.

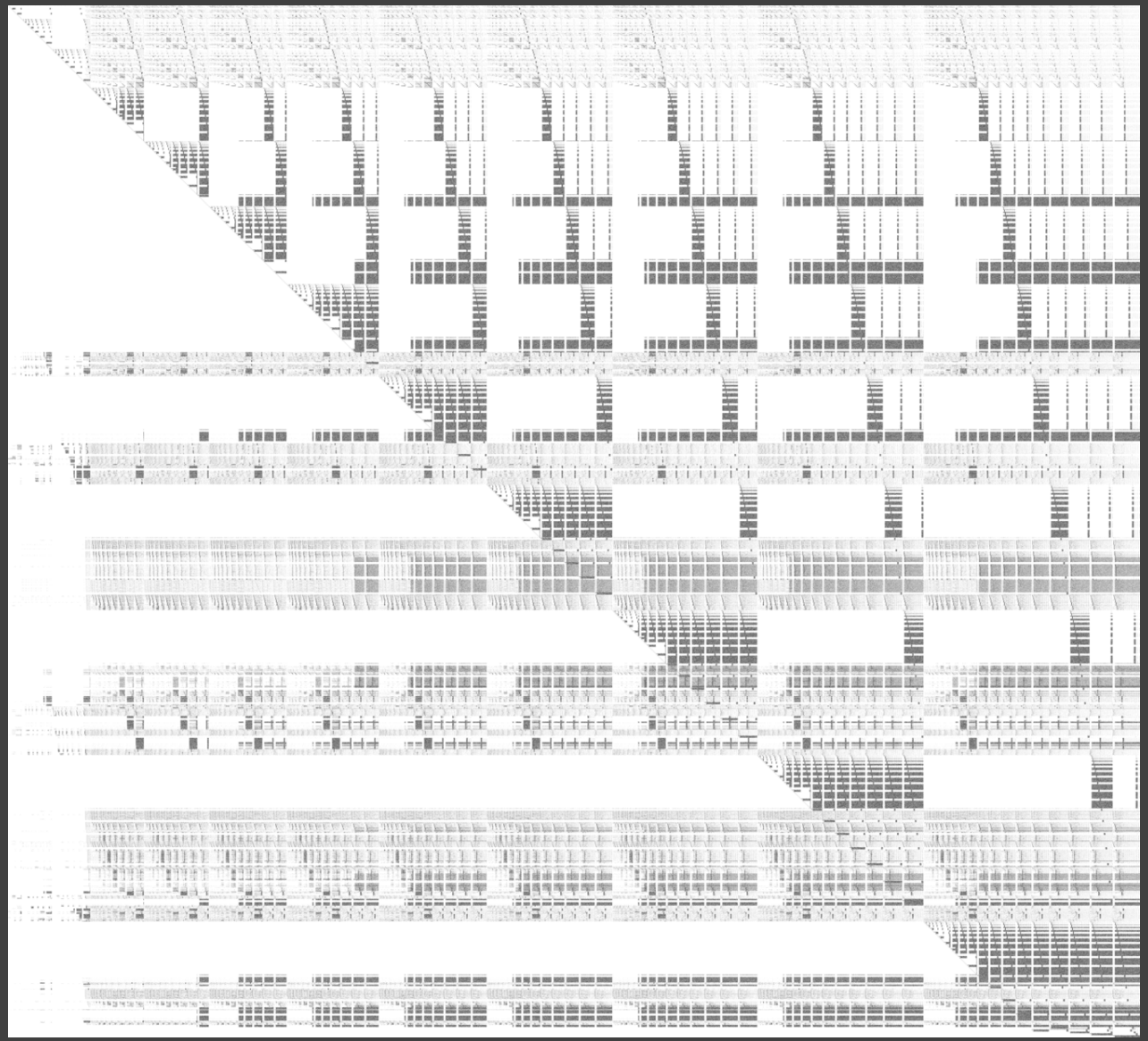
Algorithm II

```
mzp_t *Q = mzp_init(n);
rci_t r0 = mzd_analyse_gb(A, Q);

for(rci_t i=0; i<r0; i++) {
  mzd_row_swap(A, i, Q->values[i]);
}
for(rci_t i=r0; i<MIN(m,n); i++) {
 if (Q->values[i] >= r0 && Q->values[i] < MIN(m,n))
   mzd_row_swap(A, i, Q->values[i]);
}

mzd_t *ANW = mzd_init_window(A,  0,  0, r0, r0);
mzd_t *ASW = mzd_init_window(A, r0,  0,  m, r0);
mzd_t *ANE = mzd_init_window(A,  0, r0, r0,  n);
mzd_t *ASE = mzd_init_window(A, r0, r0,  m,  n);

mzd_trsm_upper_left(ANW, ANE, 0);
mzd_addmul(ASE, ASW, ANE, 0);

mzd_set_ui(ANW,1);
mzd_set_ui(ASW,0);

r0 += mzd_echelonize_pluq(ASE, 0);
```

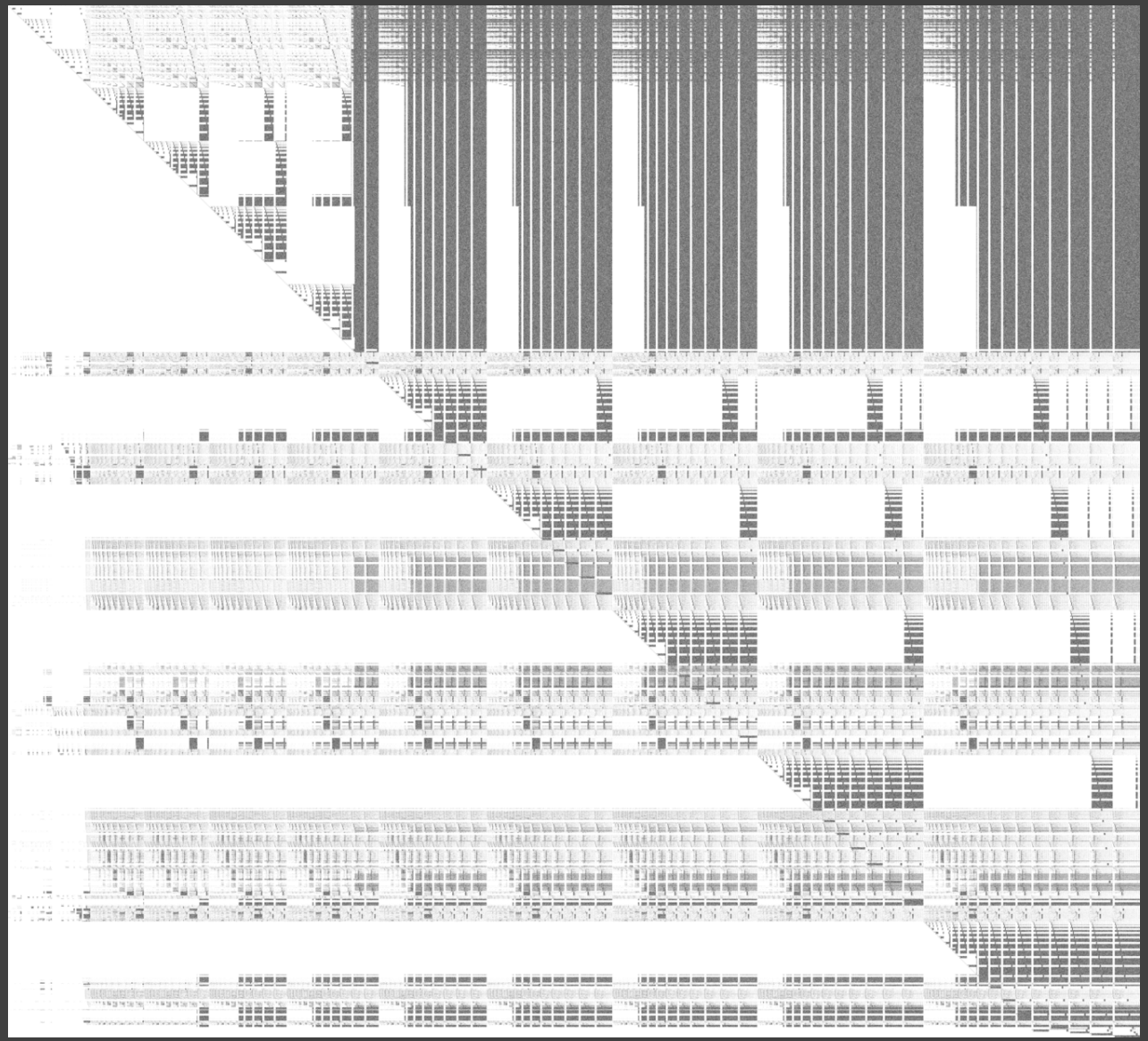
Example I



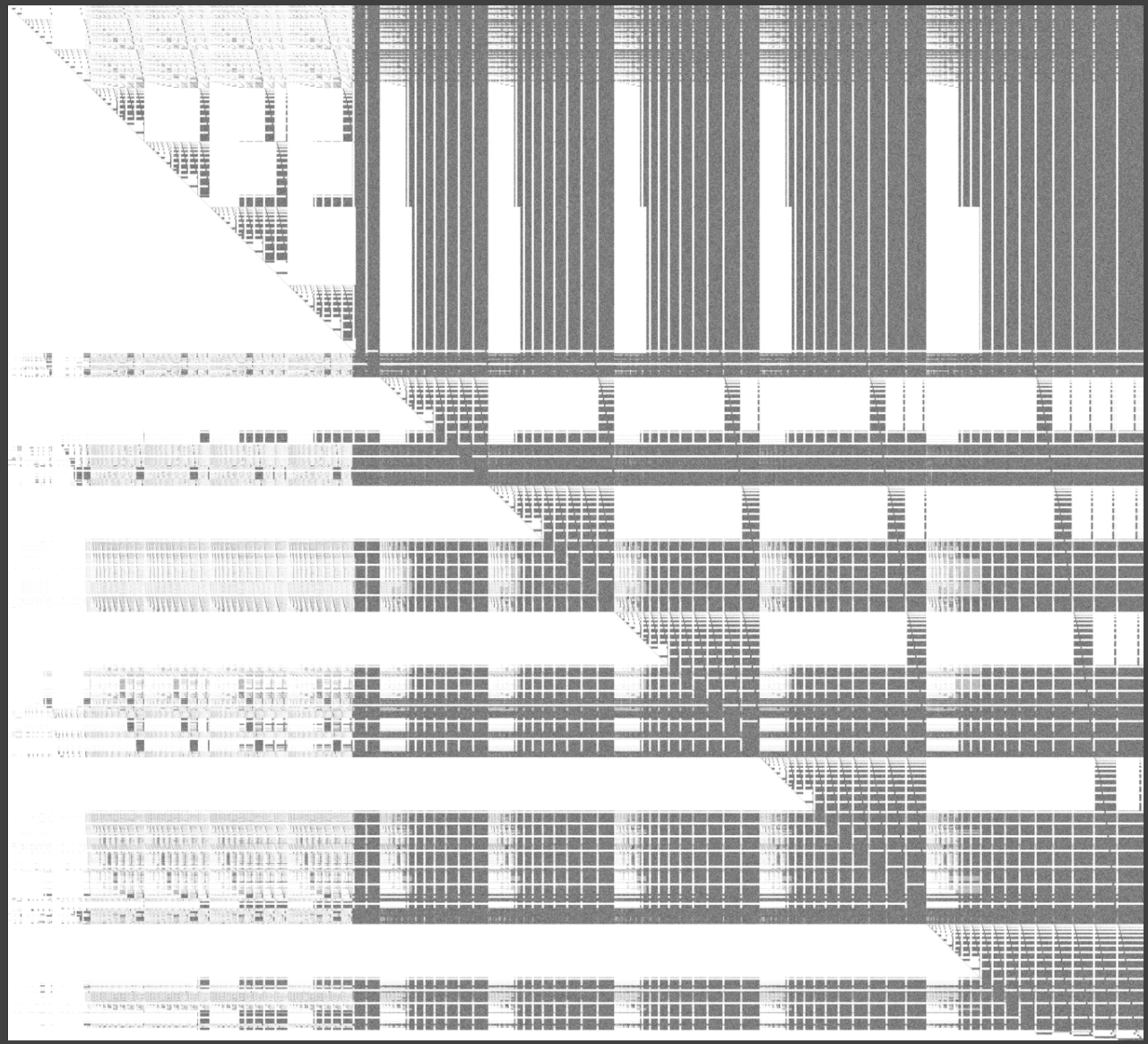
Example II



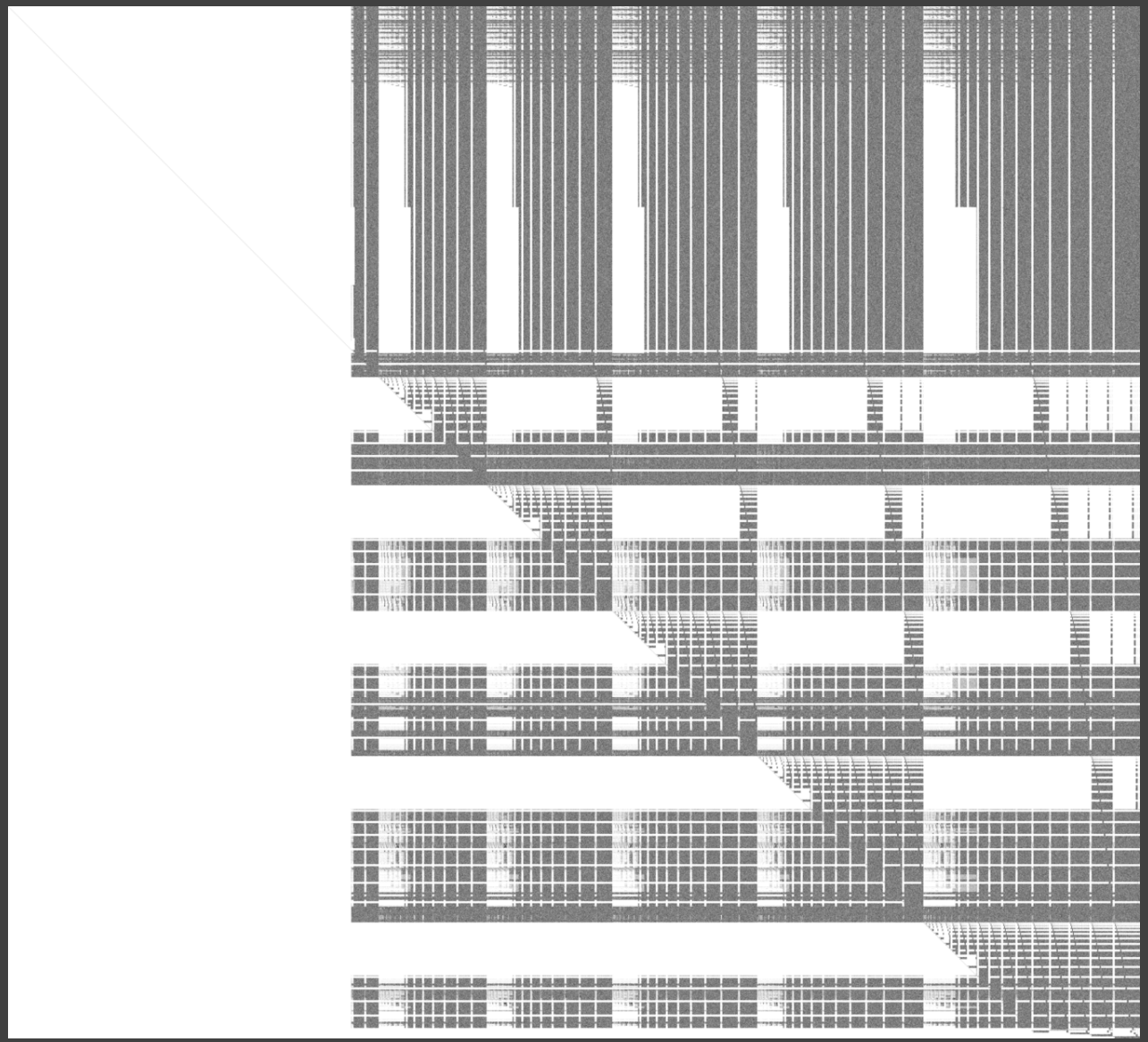
Example III



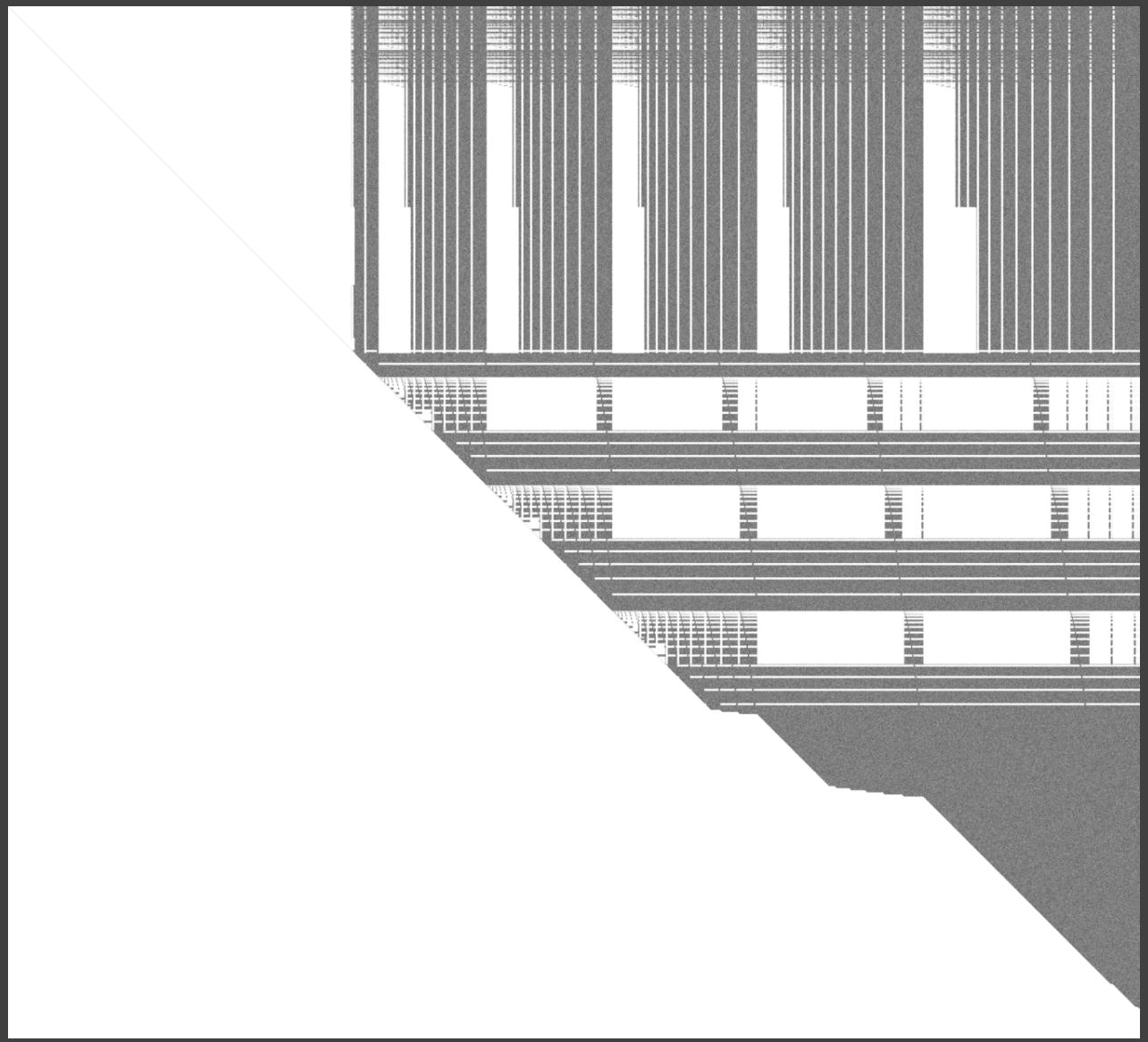
Example IV



Example V



Example VI



Results I

problem	m	n	dens.	PLE	M4RI	GB	LELA
HFE 25	12307	13508	0.076	1.0	0.5	0.8	0.7
HFE 30	19907	29323	0.067	4.7	2.7	4.7	5.4
HFE 35	29969	55800	0.059	19.3	9.2	19.5	21.8
Mutant	26075	26407	0.184	5.7	3.9	2.1	10.9
n=24, m=26	37587	38483	0.038	20.6	21.0	19.3	10.1
n=24, m=26	37576	32288	0.040	18.6	28.4	17.0	4.5
SR(2,2,2,4) c	5640	14297	0.003	0.4	0.2	0.1	1.0
SR(2,2,2,4) c	13665	17394	0.013	2.1	3.0	2.0	4.0
SR(2,2,2,4) c	11606	16282	0.035	1.9	4.4	1.5	2.2
SR(2,2,2,4)	13067	17511	0.008	1.9	2.0	1.3	3.4
SR(2,2,2,4)	12058	16662	0.015	1.5	1.9	1.6	2.7
SR(2,2,2,4)	115834	118589	0.003	528.2	578.5	522.9	95.0

LELA does reduced row echelon forms, M4RI does not.

Outline

- $\mathbb{F}_2$
  - Gray Codes
  - Multiplication
  - Elimination
  - Gröbner Basis Computations
- $\mathbb{F}_{2^e}$
  - Precomputation Tables
  - Karatsuba Multiplication
  - Performance

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
1 **begin**
2 **for** $0 \leq i < m$ **do**
3 **for** $0 \leq j < n$ **do**
4 $C_j \longleftarrow C_j + A_{j,i} \times B_i$;
5 **return** C;

The idea II

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

```
1 begin
2     for 0 ≤ i < m do
3         for 0 ≤ j < n do
4             C_j ⟵ C_j + A_{j,i} × B_i; // cheap
5     return C;
```

The idea III

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

```
1 begin
2     for 0 ≤ i < m do
3         for 0 ≤ j < n do
4             C_j ⟵ C_j + A_{j,i} × B_i; // expensive
5     return C;
```

The idea IV

Input: A – $m \times n$ matrix
Input: B – $n \times k$ matrix
1 **begin**
2 **for** $0 \leq i < m$ **do**
3 **for** $0 \leq j < n$ **do**
4 $C_j \longleftarrow C_j + A_{j,i} \times B_i$; // **expensive**
5 **return** C;

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

The idea V

```
1 begin
      Input: A – m × n matrix
      Input: B – n × k matrix
2     for 0 ≤ i < m do
3         for 0 ≤ j < 2^e do
4             T_j ⟵ j × B_i;
5         for 0 ≤ j < n do
6             x ⟵ A_{j,i};
7             C_j ⟵ C_j + T_x;
8     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

1 **begin**
2 $r \longleftarrow 0$;
3 **for** $0 \leq j < n$ **do**
4 **for** $r \leq i < m$ **do**
5 **if** $A_{i,j} = 0$ **then continue**;
6 rescale row i of A such that $A_{i,j} = 1$;
7 swap the rows i and r in A;
8 $T \longleftarrow$ multiplication table for row r of A;
9 **for** $r + 1 \leq k < m$ **do**
10 $x \longleftarrow A_{k,j}$;
11 $A_k \longleftarrow A_k + T_x$;
12 $r \longleftarrow r + 1$;
13 **return** r;

Outline

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 $A_0 x + A_1$ and B as $B_0 x + B_1$.
- The product is

$$C = A_0 B_0 x^2 + (A_0 B_1 + A_1 B_0)x + A_1 B_1.$$

- Reduction modulo f gives

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

- This last expression can be rewritten as

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

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

Outline

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

Sensitivity to Sparsity

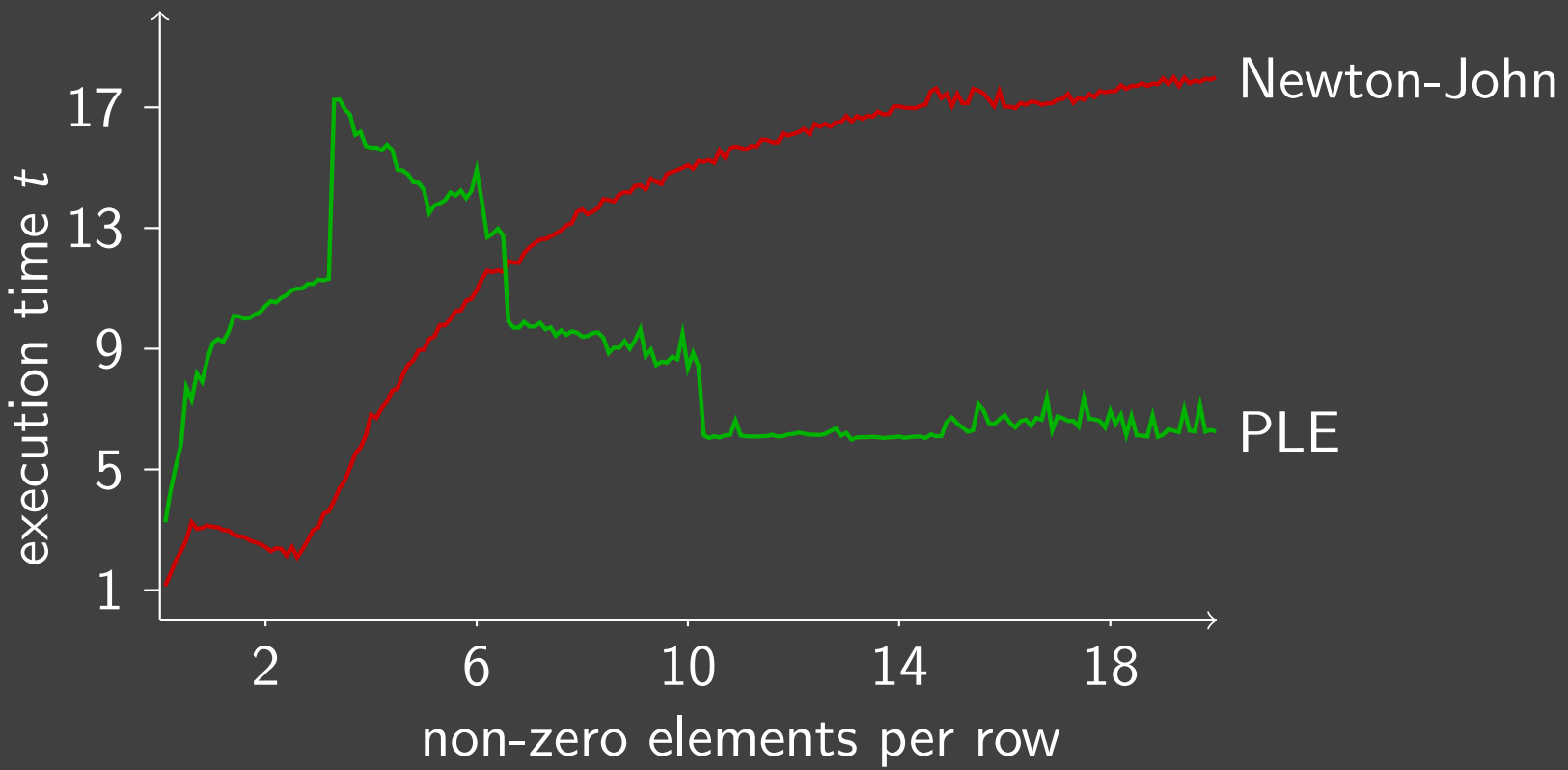
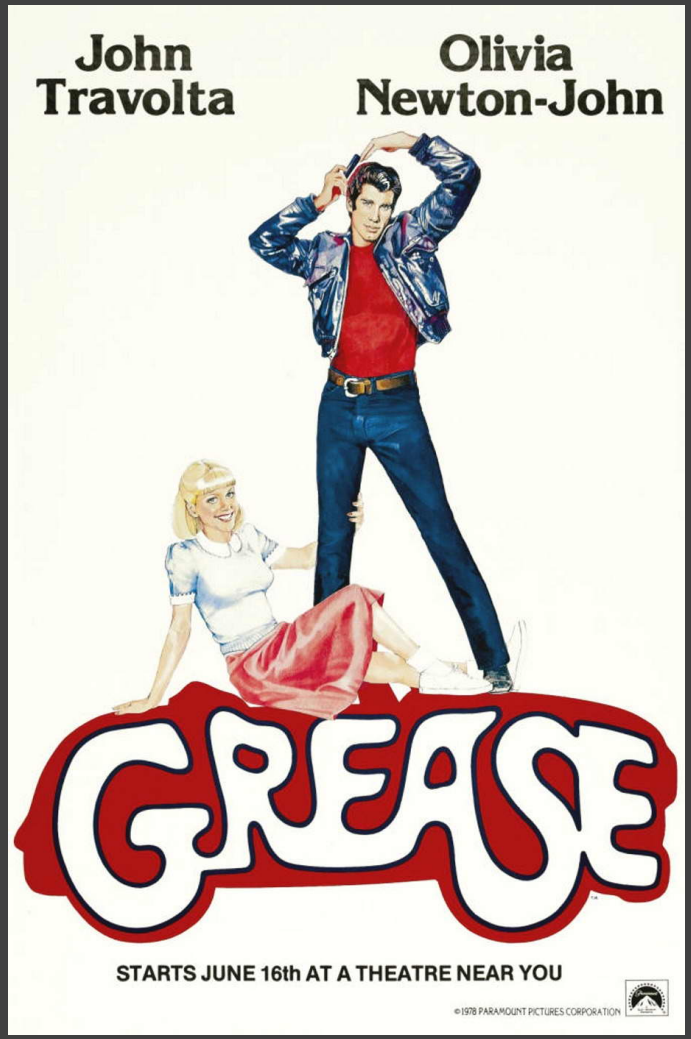


Figure: Gaussian elimination of $10,000 \times 10,000$ matrices over $\mathbb{F}_{16}$ on 2.67GHz Intel Core i7 M 620.

Fin


John Travolta
Olivia Newton-John
GREASE
STARTS JUNE 16th AT A THEATRE NEAR YOU
©1978 PARAMOUNT PICTURES CORPORATION


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.


Michael Brickenstein and Alexander Dreyer.
PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials.
In *Electronic Proceedings of MEGA 2007*, 2007.
Available at `http://www.ricam.oeaw.ac.at/mega2007/electronic/26.pdf`.


Jean-Charles Faugère and Sylvain Lachartre.
Parallel Gaussian Elimination for Gröbner bases computations in finite fields.
In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 89–97, 2010.


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.