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

Martin R. Albrecht

Nancy, March 30, 2011

Outline

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.)
- implements asymptotically fast multiplication [ABH10]
- implements asymptotically fast decomposition [AP10]
- implements 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

M4RM [ADKF70] I

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

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

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

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

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

of k rows each. We have:

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

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

begin
    $C \longleftarrow$ create an $m \times n$ matrix with all entries 0;
    $k \longleftarrow \lfloor \log n \rfloor$;
    for $0 \leq i < (\ell/k)$ **do**
        // **create table of $2^k - 1$ linear combinations**
        $T \leftarrow$ MAKETABLE$(B, i \times k, 0, k)$;
        for $0 \leq j < m$ **do**
            // **read index for table T**
            $id \longleftarrow$ READBITS$(A, j, i \times k, k)$;
            add row id from T to row j of C;
    return C;
end

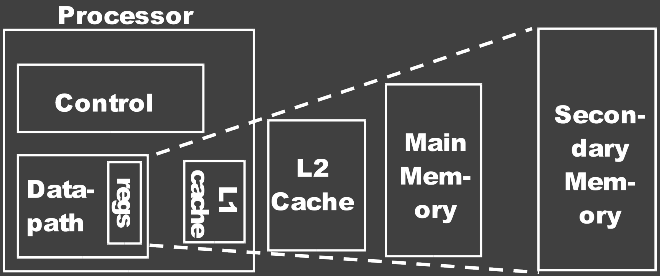
Algorithm 1: M4RM

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 [Bhu99]



Memory	Regs	L1	L2	Ram	Swap
Speed (ns)	0.5	2	6	10^2	10^7
Cost (cycles)	1	4	14	200	$2 \cdot 10^7$
Size	$4 \cdot$ 64-bit	64k	1-4M	1G	100G

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
            // we load each row j to take care of 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**
      $T \leftarrow \textsc{MakeTable}(B, i \times k, 0, k)$;
      for $0 \leq s < b_s$ **do**
        $j \longleftarrow start \times b_s + s$;
        $id \longleftarrow \textsc{ReadBits}(A, j, i \times k, k)$;
        add row id from T to row j of C;
  return C;
end

Cache Friendly M4RM III

Matrix Dimensions	Plain	Cache Friendly
$10,000 \times 10,000$	4.141	2.866
$16,384 \times 16,384$	16.434	12.214
$20,000 \times 20,000$	29.520	20.497
$32,000 \times 32,000$	86.153	82.446

Table: Strassen-Winograd with different base cases on 64-bit Linux, 2.33Ghz Core 2 Duo

$t > 1$ 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 ⟵ create an m × n matrix with all entries 0;
    for 0 ≤ i < (ℓ/(2k)) do
        T_0 ← MakeTable(B, i × 2k, 0, k);
        T_1 ← MakeTable(B, i × 2k + k, 0, k);
        for 0 ≤ j < m do
            id_0 ⟵ ReadBits(A, j, i × 2k, k);
            id_1 ⟵ ReadBits(A, j, i × 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
```

$t > 1$ Gray Code Tables III

Matrix Dimensions	$t = 1$	$t = 2$	$t = 8$
$10,000 \times 10,000$	4.141	1.982	1.599
$16,384 \times 16,384$	16.434	7.258	6.034
$20,000 \times 20,000$	29.520	14.655	11.655
$32,000 \times 32,000$	86.153	49.768	44.999

Table: Strassen-Winograd with different base cases on 64-bit Linux, 2.33Ghz Core 2 Duo

Results: Multiplication

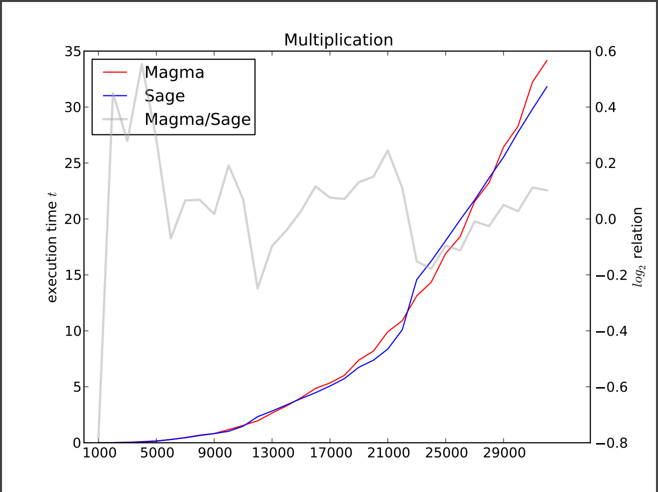


Figure: 2.66 Ghz Intel i7, 4GB RAM

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

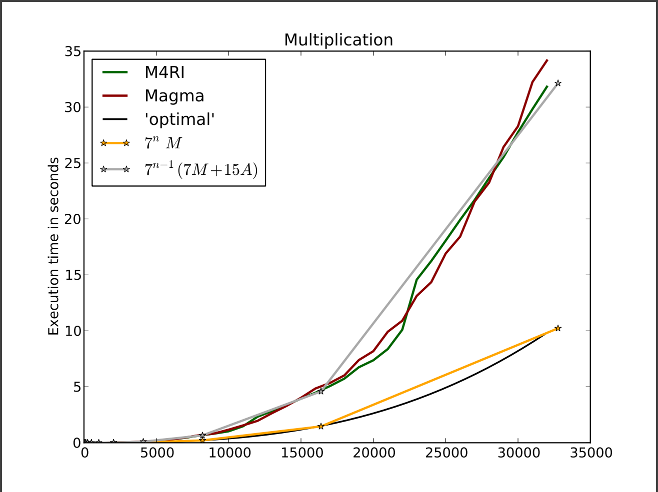


Figure: 2.66 Ghz Intel i7, 4GB RAM

Outline

M4RI

Introduction

Multiplication

Elimination

M4RIE

Introduction

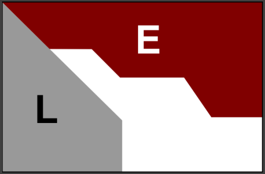
Travolta Tables

Karatsuba Multiplication

Results

Final

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.

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

Write

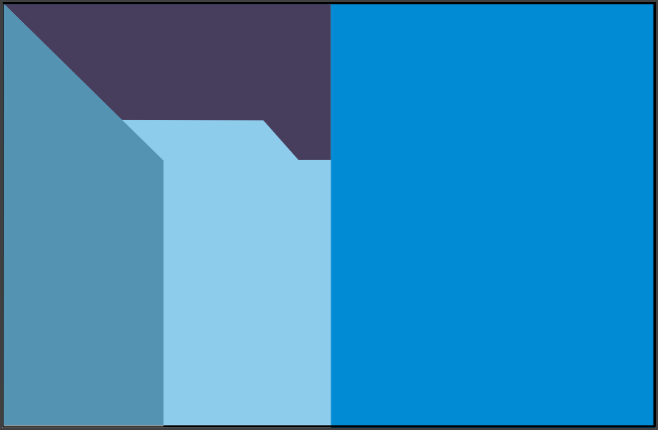
$$A = \begin{pmatrix} A_W & A_E \end{pmatrix} = \begin{pmatrix} A_{NW} & A_{NE} \\ A_{SW} & A_{SE} \end{pmatrix}$$

Main steps:

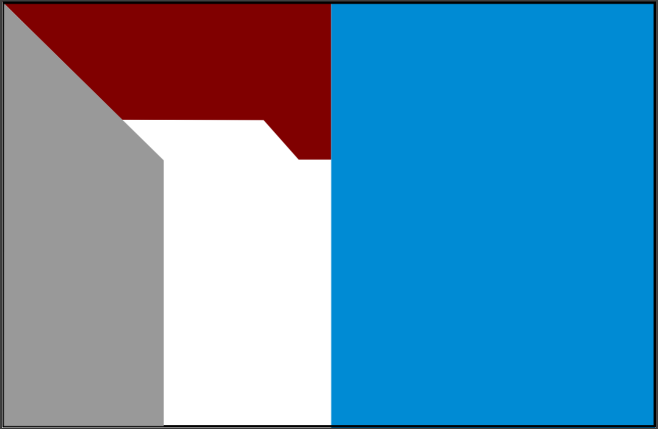
1. Call PLE on A_W
2. Apply row permutation to A_E
3. $L_{NW} \leftarrow$ the lower left triangular matrix in A_{NW}
4. $A_{NE} \leftarrow L_{NW}^{-1} \times A_{NE}$
5. $A_{SE} \leftarrow A_{SE} + A_{SW} \times A_{NE}$
6. Call PLE on A_{SE}
7. Apply row permutation to A_{SW}
8. Compress L

Visualisation

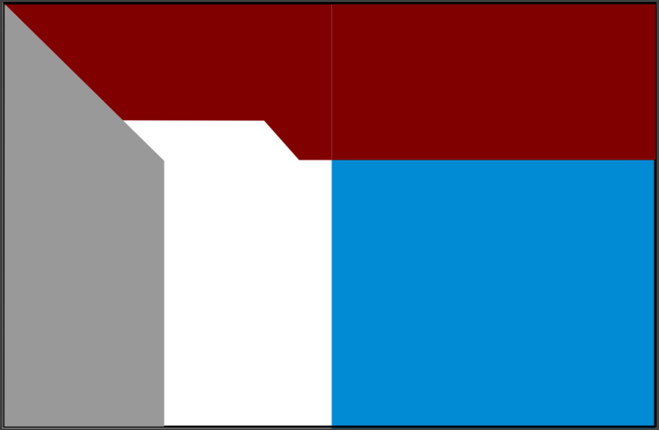
Visualisation



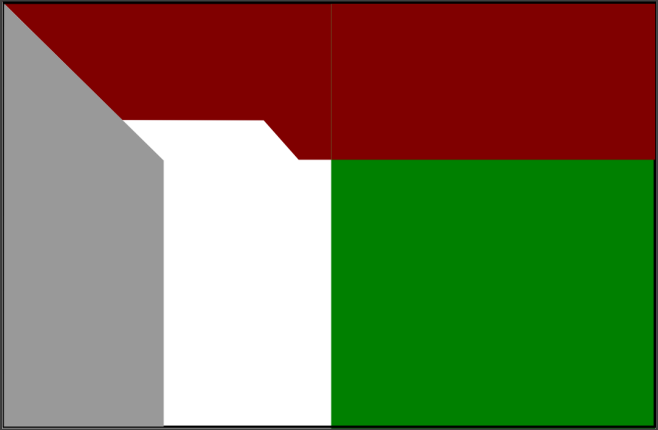
Visualisation



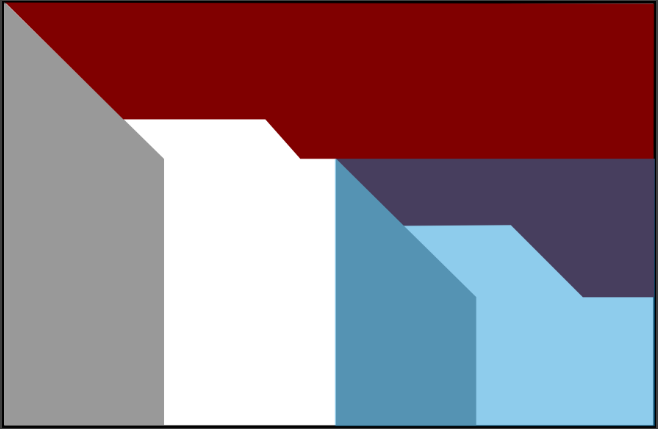
Visualisation



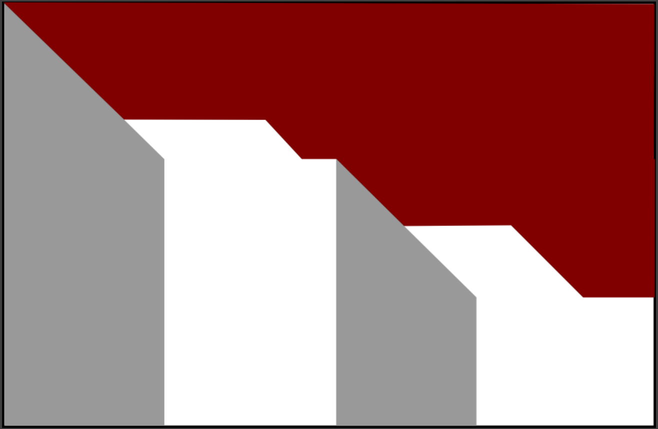
Visualisation



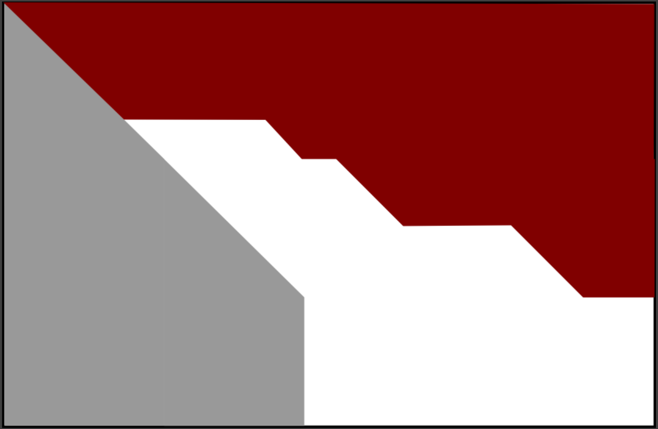
Visualisation



Visualisation



Visualisation



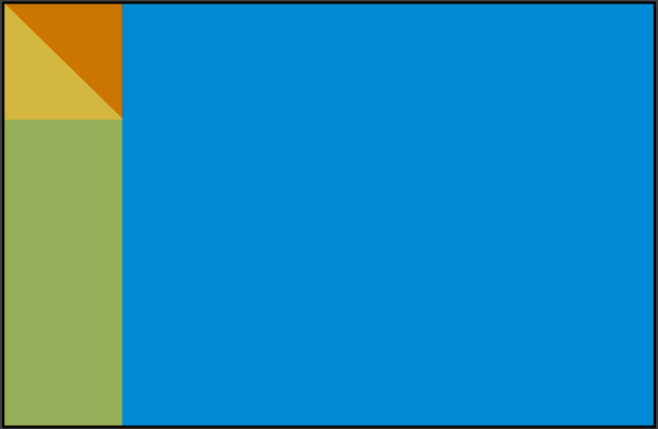
Block Iterative PLE Decomposition I

We need an efficient base case for PLE Decomposition

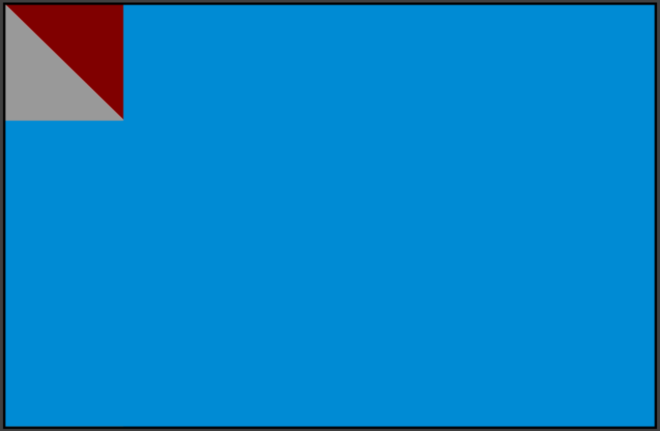
- block recursive PLE decomposition gives rise to a block iterative PLE decomposition
- choose blocks of size $k = \log n$ and use M4RM for the "update" multiplications
- this gives a complexity $\mathcal{O}(n^3/\log n)$
- this is an alternative way of looking at the M4RI algorithm or its PLE decomposition equivalent ("MMPF")
- M4RI is more cache friendly than straight block iterative PLE decomposition, so we adapt it PLE using M4RI idea

Visualisation

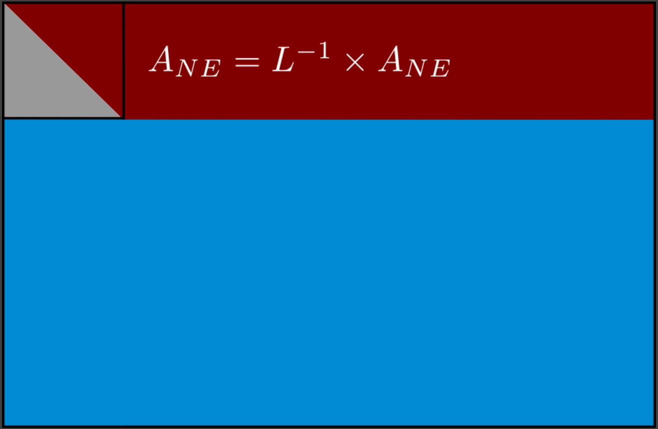
Visualisation



Visualisation

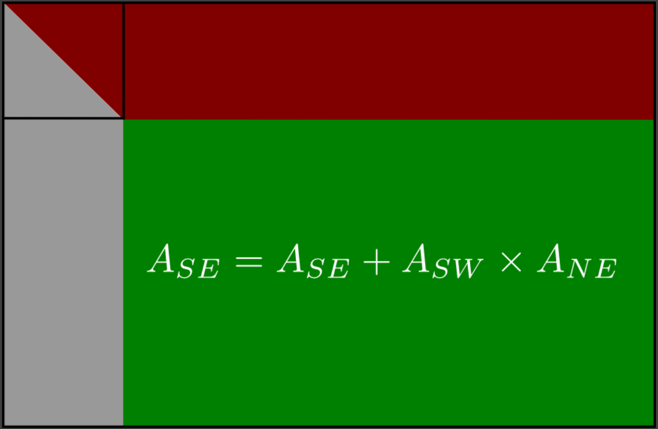


Visualisation

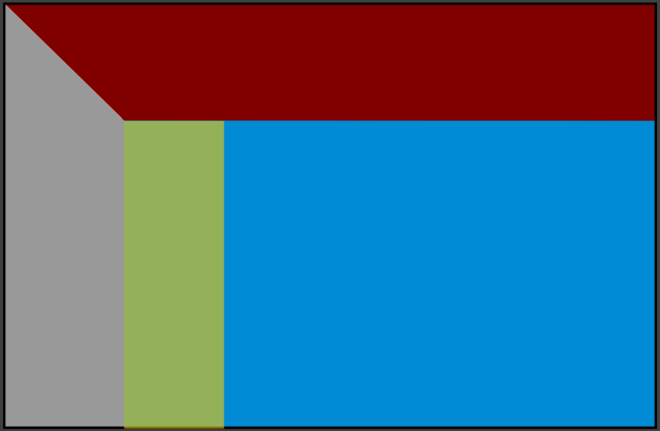


Visualisation

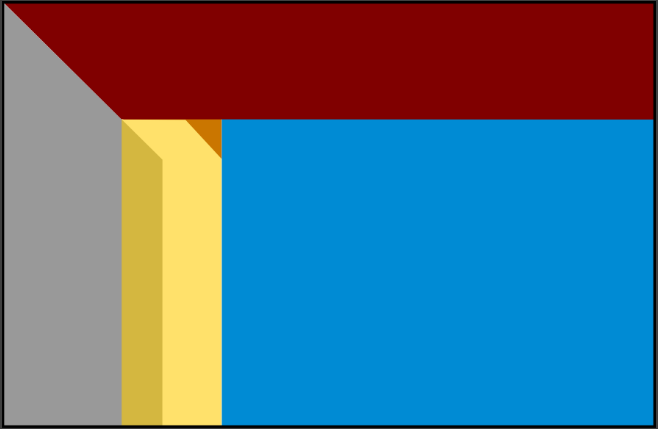
Visualisation



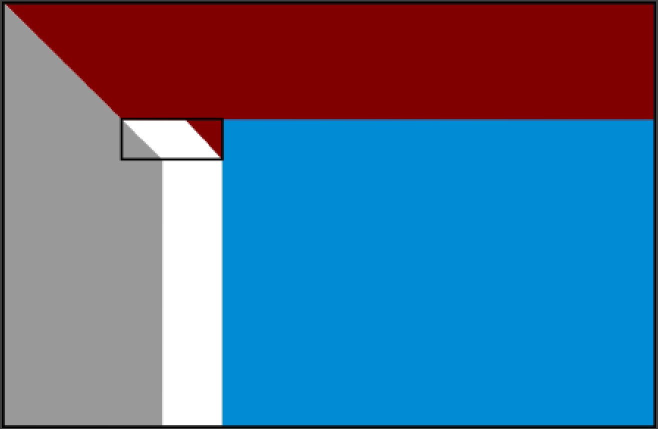
Visualisation



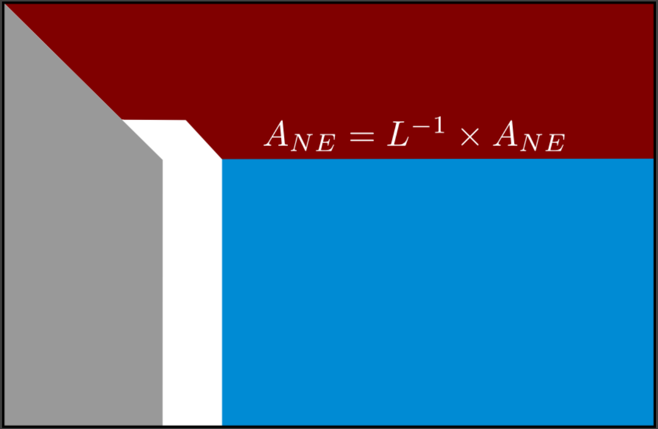
Visualisation



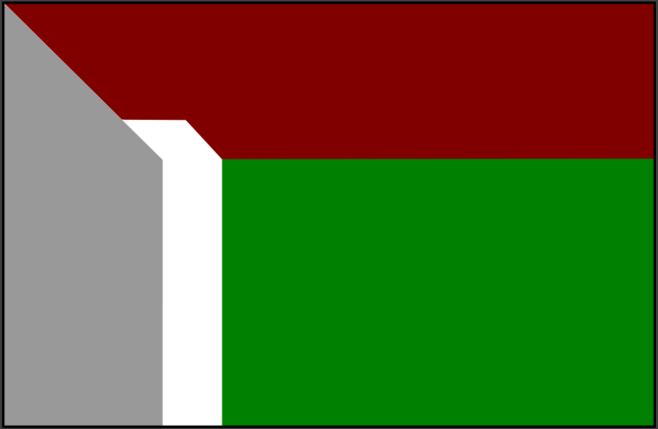
Visualisation



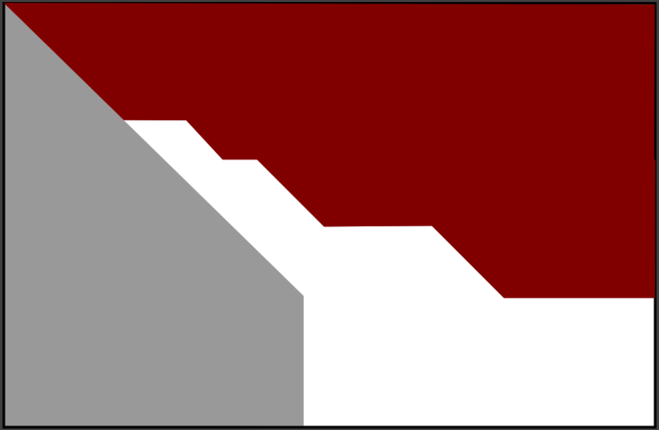
Visualisation



Visualisation



Visualisation



Results: Reduced Row Echelon Form

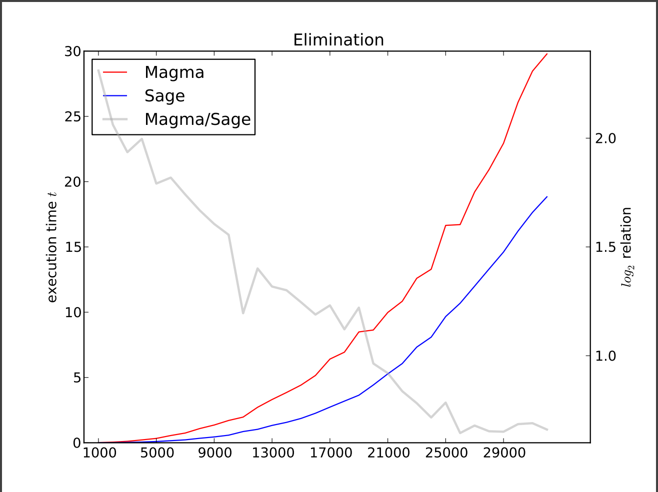


Figure: 2.66 Ghz Intel i7, 4GB RAM

Results: Row Echelon Form

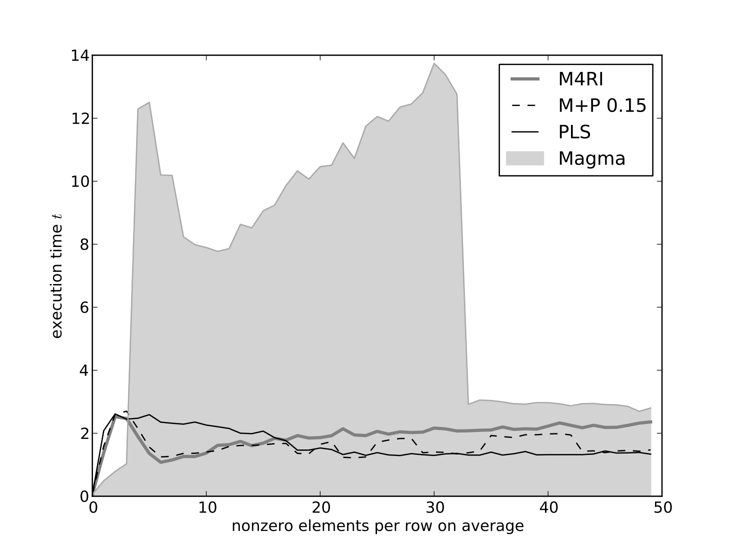
Using one core we can compute the echelon form of a $500,000 \times 500,000$ dense random matrix over $\mathbb{F}_2$ in

$$9711.42 \text{ seconds } = 2.7 \text{ hours.}$$

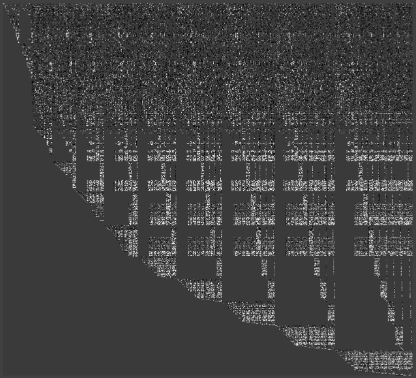
Using four cores decomposition we can compute the echelon form of a random dense $500,000 \times 500,000$ matrix in

$$3806.28 \text{ seconds } = 1.05 \text{ hours.}$$

Work-in-Progress: Sensitivity to Sparsity

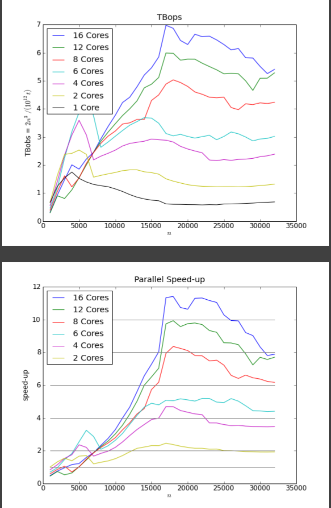


Work-in-Progress: Gröbner Basis Linear Algebra

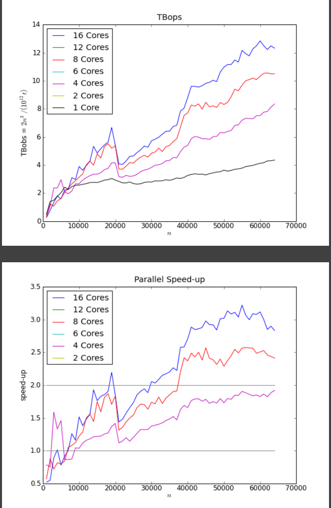


			64-bit Debian/GNU Linux, 2.6Ghz Opteron)				
Problem	Matrix Dimension	Density	Magma 2.15-10	M4RI 20100324	PLS 20100324	M+P 0.15 20100429	M+P 0.20 20100429
HFE 25	$12,307 \times 13,508$	0.076	4.57s	3.28s	3.45s	**3.03s**	3.21s
HFE 30	$19,907 \times 29,323$	0.067	33.21s	**23.72s**	25.42s	23.84s	25.09s
HFE 35	$29,969 \times 55,800$	0.059	278.58s	126.08s	159.72s	154.62s	**119.44s**
MXL	$26,075 \times 26,407$	0.185	76.81s	23.03s	19.04s	**17.91s**	18.00s

Work-in-Progress: Multi-core Support



M4RI BOpS & Speed-up



PLE BOpS & Speed-up

Outline

M4RI

- Introduction
- Multiplication
- Elimination

M4RIE

- Introduction
- Travolta Tables
- Karatsuba Multiplication
- Results

Final

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 *s*
Sage 4.6	36.53
NTL 5.4.2	15.33
Magma 2.15	0.87
LinBox over $\mathbb{F}_{251}$	0.17
this work	0.24

Table: Row echelon form of a dense $1,000 \times 1,000$ matrix over $\mathbb{F}_{2^8}$.

Note

Our code is not asymptotically fast yet.

The M4RIE Library

- 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 fast echelon forms
- Linux, Mac OS X (x86 and PPC), OpenSolaris (Sun Studio Express) and Windows (Cygwin)

`http://m4ri.sagemath.org`

Representation of Elements

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

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

with f irreducible, $e = \deg(f)$ and $f(\alpha) = 0$ in the algebraic closure of $\mathbb{F}_2$.

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 $\sum_{i=0}^{e-1} a_i 2^i$.

We pack several of those bitstrings into one machine word:

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

Outline

The idea I

In our representation:

- Scaling a row is expensive, we need to deal with each element individually.
- Adding two rows is cheap, we can deal with words directly: XOR.

Thus, we prefer additions over multiplications.

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**
      $C_j \longleftarrow C_j + A_{j,i} \times B_i$;
  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**
            $C_j \longleftarrow C_j + A_{j,i} \times B_i$; // **cheap**
    return C;
end

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;
end

The idea V

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

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

The idea VI

```
begin
    Input: A – m × n matrix
    Input: B – n × k matrix
    for 0 ≤ i < m do
        for 0 ≤ j < 2^e do
            T_j ⟵ j × B_i;
        for 0 ≤ j < n do
            x ⟵ A_{j,i};
            C_j ⟵ C_j + T_x;
    return C;
end
```

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

Gaussian elimination

```
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

M4RI

Introduction

Multiplication

Elimination

M4RIE

Introduction

Travolta Tables

Karatsuba Multiplication

Results

Final

The idea

- Rewrite matrices over $\mathbb{F}_{2^e}$ as a list of e matrices over $\mathbb{F}_2$ which contain the coefficients for each degree of α.
- Instead of considering matrices of polynomials, we can consider polynomials of matrices.

Tomas J. Boothby and Robert Bradshaw.
Bitslicing and the Method of Four Russians Over Larger Finite Fields.
CoRR, abs/0901.1413, 2009.

An example I

- 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 $\mathbb{F}_2$.

An example II

e	Travolta CPU time	# of $\mathbb{F}_2$ mults	naive	Karatsuba
2	0.664s	9.222	4	3
3	1.084s	16.937	9	
4	1.288s	17.889	16	9
5	3.456s	50.824	25	
6	4.396s	64.647	36	
7	6.080s	84.444	49	
8	11.117s	163.471	64	27
9	37.842s	556.473	81	
10	47.143s	620.261	100	

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

Outline

Results: Multiplication I

We only implemented Karatsuba for $\mathbb{F}_{2^2}$ so far.

n	Travolta	Karatsuba	$3 \cdot \mathbb{F}_2$ time	Travolta	Karatsuba	$3 \cdot \mathbb{F}_2$ time
	2.66Ghz i7			Opteron 8439 SE		
1000	0.012s	0.012s	0.012s	0.020s	0.020s	0.060s
2000	0.068s	0.052s	0.024s	0.140s	0.070s	0.030s
3000	0.224s	0.136s	0.096s	0.470s	0.200s	0.150s
4000	0.648s	0.280s	0.336s	1.120s	0.480s	0.390s
5000	1.144s	0.520s	0.432s	2.090s	0.870s	0.690s
6000	1.952s	0.984s	1.008s	3.490s	1.500s	1.260s
7000	3.272s	1.444s	1.632s	5.440s	2.270s	1.950s
8000	4.976s	2.076s	2.484s	8.050s	3.230s	2.850s
9000	6.444s	2.784s	2.628s	10.710s	4.560s	4.140s
10000	8.761s	3.668s	3.528s	14.580s	5.770s	5.190s

Table: Multiplication of $n \times n$ matrices over $\mathbb{F}_{2^2}$.

Results: Multiplication II

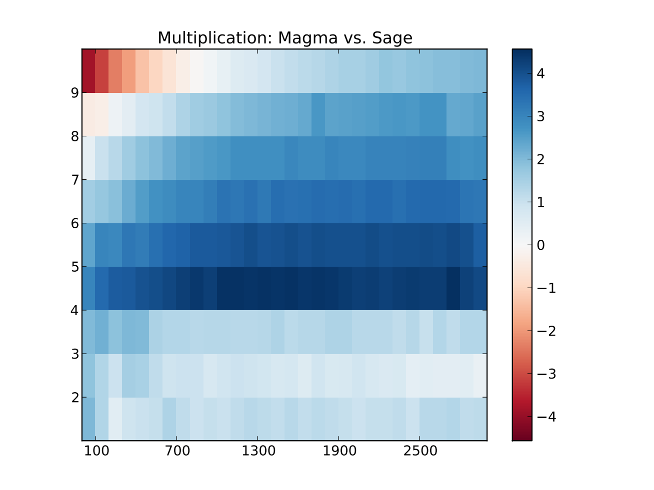


Figure: 2.66 Ghz Intel i7, 4GB RAM

Results: Reduced Row Echelon Forms

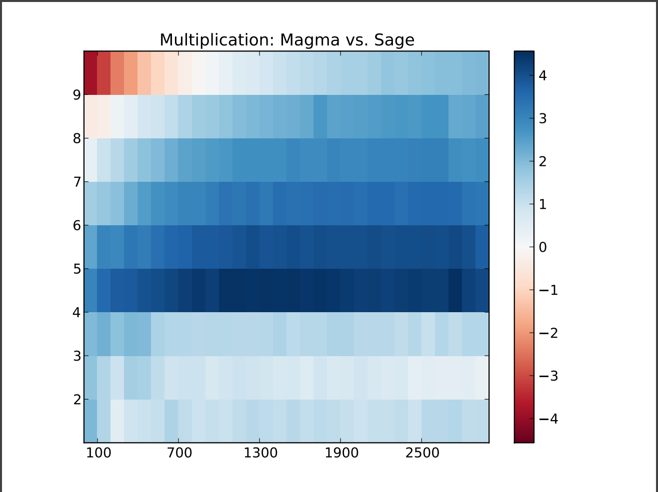


Figure: 2.66 Ghz Intel i7, 4GB RAM

Fin

- Martin Albrecht, Gregory V. Bard, and William Hart.
  Algorithm 898: Efficient multiplication of dense matrices over GF(2).
  ACM Transactions on Mathematical Software, 37(1):14 pages, January 2010.
  pre-print available at `http://arxiv.org/abs/0811.1714`.
- 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.
- Martin R. Albrecht and Clément Pernet.
  Efficient decomposition of dense matrices over GF(2).
  `arXiv:1006.1744v1`, Jun 2010.

- L.N. Bhuyan.
  CS 161 Ch 7: Memory Hierarchy Lecture 22, 1999.
  Available at
  http://www.cs.ucr.edu/~bhuyan/cs161/LECTURE22.ppt.
- Volker Strassen.
  Gaussian elimination is not optimal.
  Nummerische Mathematik, 13:354–256, 1969.