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

**M4RI**
- Multiplication
- Elimination
- Projects

**M4RIE**
- Introduction
- Newton-John Tables
- Karatsuba Multiplication
- Results
Consider $C = A \cdot B$ ($A$ is $m \times \ell$ and $B$ is $\ell \times n$).

$A$ can be divided into $\ell/k$ vertical “stripes”

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

of $k$ columns each. $B$ can be divided into $\ell/k$ horizontal “stripes”

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

of $k$ rows each. We have:

$$C = A \cdot B = \sum_{0}^{(\ell-1)/k} A_i \cdot B_i.$$
\[ A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \quad A_0 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ A_1 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad B_0 = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \quad B_1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \]

\[ A_0 \cdot B_0 = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \quad A_1 \cdot B_1 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \]
M4RM: Algorithm $O\left(\frac{n^3}{\log n}\right)$

1 begin
2 \hspace{1em} C \leftarrow \text{create an } m \times n \text{ matrix with all entries 0;}
3 \hspace{1em} k \leftarrow \lfloor \log n \rfloor;
4 \hspace{1em} \textbf{for } 0 \leq i < (\ell / k) \textbf{ do}
5 \hspace{2em} \text{ // create table of } 2^k - 1 \text{ linear combinations}
6 \hspace{3em} T \leftarrow \text{MakeTable}(B, i \times k, 0, k);
7 \hspace{2em} \textbf{for } 0 \leq j < m \textbf{ do}
8 \hspace{3em} \text{ // read index for table } T
9 \hspace{4em} \text{id} \leftarrow \text{ReadBits}(A, j, i \times k, k);
10 \hspace{4em} \text{add row id from } T \text{ to row } j \text{ of } C;
11 \hspace{1em} \text{return } C;
12 end

Algorithm 1: M4RM
Strassen-Winograd [Str69] Multiplication

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

$\rightarrow$ optimisation of this base case
begin
  \( C \leftarrow \) create an \( m \times n \) matrix with all entries 0;
  for \( 0 \leq i < \ell/k \) do
    // this is cheap in terms of memory access
    \( T \leftarrow \text{MAKE_TABLE}(B, i \times k, 0, k); \)
    for \( 0 \leq j < m \) do
      // for each load of row \( j \) we take care of only \( k \) bits
      \( id \leftarrow \text{READBITS}(A, j, i \times k, k); \)
      add row \( id \) from \( T \) to row \( j \) of \( C \);
  end
return \( C \);
Cache Friendly M4RM II

1. \( \text{begin} \)
2. \( C \leftarrow \text{create an } m \times n \text{ matrix with all entries 0;} \)
3. for \( 0 \leq \text{start} < m/b_s \) do
4. for \( 0 \leq i < (\ell/k) \) do
   5. \( \text{// we regenerate } T \text{ for each block} \)
   6. \( T \leftarrow \text{MAKE\_TABLE}(B, i \times k, 0, k); \)
   7. for \( 0 \leq s < b_s \) do
      8. \( j \leftarrow \text{start} \times b_s + s; \)
      9. \( id \leftarrow \text{READ\_BITS}(A, j, i \times k, k); \)
      10. add row \( id \) from \( T \) to row \( j \) of \( C \);
11. \( \text{return } C; \)
12. \( \text{end} \)
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

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

for $0 \leq i < (\ell/(2k))$ do

$T_0 \leftarrow \text{MakeTable}(B, i \times 2k, 0, k)$;

$T_1 \leftarrow \text{MakeTable}(B, i \times 2k + k, 0, k)$;

for $0 \leq j < m$ do

$id_0 \leftarrow \text{ReadBits}(A, j, i \times 2k, k)$;

$id_1 \leftarrow \text{ReadBits}(A, j, i \times 2k + k, k)$;

add row $id_0$ from $T_0$ and row $id_1$ from $T_1$ to row $j$ of $C$;

return $C$;

end
Results: Multiplication

<table>
<thead>
<tr>
<th>Matrix Dimension $n$</th>
<th>Execution Time $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1s</td>
</tr>
<tr>
<td>8000</td>
<td>7s</td>
</tr>
<tr>
<td>14000</td>
<td>13s</td>
</tr>
<tr>
<td>20000</td>
<td>19s</td>
</tr>
<tr>
<td>26000</td>
<td>25s</td>
</tr>
</tbody>
</table>

Figure: 2.66 Ghz Intel i7, 4GB RAM
Small Matrices

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Thomé (μs)</th>
<th>M4RI (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>transpose</td>
<td>4.5097</td>
<td>0.6352</td>
</tr>
<tr>
<td>copy</td>
<td>0.2019</td>
<td>0.2674</td>
</tr>
<tr>
<td>add</td>
<td>0.2533</td>
<td>0.2921</td>
</tr>
<tr>
<td>mul</td>
<td>0.2535</td>
<td>0.4472</td>
</tr>
</tbody>
</table>

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

\[ c = \frac{7 \times 0.2535 + 15 \times 0.2533}{128^{\log_2 7} \times 10^6} \]

\[ c \cdot n^{\log_2 7} \]

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.
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.
Block Recursive PLE Decomposition $O(n^\omega)$ 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 $O(n^{\omega})$ II
Block Recursive PLE Decomposition $O(n^\omega)$ III
Block Recursive PLE Decomposition $O(n^\omega)$ IV

$A_{NE} \leftarrow L_{NW}^{-1} \times A_{NE}$
Block Recursive PLE Decomposition $O(n^\omega)$ V

$A_{SE} \leftarrow A_{SE} + A_{SW} \times A_{NE}$
Block Recursive PLE Decomposition $O(n^\omega)$ VI
Block Recursive PLE Decomposition $O(n^\omega)$ VII
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 $\mathcal{O}(n^3 / \log n)$
Block Iterative PLE Decomposition II
Block Iterative PLE Decomposition III
\[ A_{NE} \leftarrow L^{-1} \times A_{NE} \]
$A_{SE} \leftarrow A_{SE} + A_{SW} \times A_{NE}$
Block Iterative PLE Decomposition VII
Block Iterative PLE Decomposition VIII
$A_{NE} = L^{-1} \times A_{NE}$
$A_{SE} = A_{SE} + A_{SW} \times A_{NE}$
Block Iterative PLE Decomposition XI
Results: Reduced Row Echelon Form

Magma

\[ t \approx 1s \]
\[ t \approx 7s \]
\[ t \approx 13s \]
\[ t \approx 19s \]
\[ t \approx 25s \]
\[ t \approx 31s \]

M4RI

\[ t \approx 2000 \]
\[ t \approx 8000 \]
\[ t \approx 14000 \]
\[ t \approx 20000 \]
\[ t \approx 26000 \]

\[ c_{MAGMA} \approx 6.8 \cdot 10^{-12} \]
\[ c_{M4RI} \approx 4.3 \cdot 10^{-12} \]

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 idling 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

![Graph showing execution time vs. non-zero elements per row for Gaussian elimination of 10,000 x 10,000 matrices on Intel 2.33GHz Xeon E5345 comparing Magma 2.17-12 and M4RI 20111004.]

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

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

M4RI BOpS & Speed-up

PLE BOpS & Speed-up
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):*

```python
[...] //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):*

```python
[...] 
> 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
The situation has not improved much in 2011:

<table>
<thead>
<tr>
<th>System</th>
<th>Time in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sage 4.7.2</td>
<td>97,000</td>
</tr>
<tr>
<td>NTL 5.4.2</td>
<td>85,000</td>
</tr>
<tr>
<td>LinBox SVN + patches</td>
<td>460</td>
</tr>
<tr>
<td>GAP 4.412</td>
<td>210</td>
</tr>
<tr>
<td>Magma 2.15</td>
<td>13</td>
</tr>
<tr>
<td>this work</td>
<td>5.5</td>
</tr>
</tbody>
</table>

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 $\sum_{i=0}^{e-1} a_i2^i$.

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.
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

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

Figure: 2 \times 2 matrix over \( \mathbb{F}_8 \)
Outline

M4RI
  Multiplication
  Elimination
  Projects

M4RIE
  Introduction
  Newton-John Tables
  Karatsuba Multiplication
  Results
The idea is:

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   \hspace{1em} $C_j \leftarrow C_j + A_{j; i} \times B_{i;}$
5  return $C$
6 end
The idea II

**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. \[ C_j \leftarrow C_j + A_{j,i} \times B_i; \quad \text{// cheap} \]
5. return $C$;
6. **end**
The idea III

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             $C_j \leftarrow C_j + A_{j,i} \times B_i$; // expensive
5     return $C$; 
6 end
The idea IV

\textbf{Input:} \( A - m \times n \) matrix

\textbf{Input:} \( B - n \times k \) matrix

\begin{verbatim}
begin
  for 0 \leq i < m do
    for 0 \leq j < n do
      C_j \leftarrow C_j + A_{j,i} \times B_i; \quad \text{// expensive}
  return C;
end
\end{verbatim}

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
        $T_j \leftarrow j \times B_i$;
    end
    for $0 \leq j < n$ do
        $x \leftarrow A_{j,i}$;
        $C_j \leftarrow C_j + T_x$;
    end
return $C$;
end

$m \cdot n \cdot k$ additions, $m \cdot 2^e \cdot k$ multiplications.
Gaussian elimination & PLE decomposition

Input: $A$ – $m \times n$ matrix

begin

\[ r \leftarrow 0; \]

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

\State return $r$;

end
Outline

M4RI
- Multiplication
- Elimination
- Projects

M4RIE
- Introduction
- Newton-John Tables
- Karatsuba Multiplication
- Results
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

M4RI
  Multiplication
  Elimination
  Projects

M4RIE
  Introduction
  Newton-John Tables
  Karatsuba Multiplication
  Results
Results: Multiplication I

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

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

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

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.

Peter L. Montgomery.
Five, six, and seven-term Karatsuba-like formulae.

Volker Strassen.
Gaussian elimination is not optimal.