IMPROVED PARALLEL GAUSSIAN ELIMINATION FOR GRÖBNER BASIS COMPUTATIONS IN FINITE FIELDS

Brice Boyer, Christian Eder, Jean-Charles Faugère, Sylvain Lachartre and Fayssal Martani

October 01, 2015

University of Kaiserslautern
TABLE OF CONTENTS

1. Linear Algebra for Gröbner basis computations
2. Features of GBLA
3. Some benchmarks
4. Outlook
LINEAR ALGEBRA FOR GRÖBNER BASIS COMPUTATIONS
• Algorithms like Faugère’s **F4** compute Gröbner bases via isolating the tasks of *searching for reducers* and *performing the reduction*. 
• Algorithms like Faugère’s \textbf{F4} compute Gröbner bases via isolating the tasks of searching for reducers and performing the reduction.

• Taking a subset of \textit{S-pairs} a \textit{symbolic preprocessing} is performed.
• Algorithms like Faugère’s F4 compute Gröbner bases via isolating the tasks of searching for reducers and performing the reduction.
• Taking a subset of S-pairs a symbolic preprocessing is performed.
• Out of this data a matrix \( M \) is generated: Its rows correspond to polynomials, its columns represent all appearing monomials in the given order.
• Algorithms like Faugère’s F4 compute Gröbner bases via isolating the tasks of searching for reducers and performing the reduction.
• Taking a subset of S-pairs a symbolic preprocessing is performed.
• Out of this data a matrix $M$ is generated: Its rows correspond to polynomials, its columns represent all appearing monomials in the given order.
• Performing Gaussian Elimination on $M$ corresponds to reducing the chosen subset of S-pairs at once.
• Algorithms like Faugère’s F4 compute Gröbner bases via isolating the tasks of searching for reducers and performing the reduction.

• Taking a subset of S-pairs a symbolic preprocessing is performed.

• Out of this data a matrix $M$ is generated: Its rows correspond to polynomials, its columns represent all appearing monomials in the given order.

• Performing Gaussian Elimination on $M$ corresponds to reducing the chosen subset of S-pairs at once.

• New data for the Gröbner basis can then be read off the reduced matrix: Restore corresponding rows as polynomials.
Specialize **Linear Algebra** for reduction steps in GB computations.
Specialize **Linear Algebra** for reduction steps in GB computations.

\[
\begin{align*}
1 & 3 & 0 & 0 & 7 & 1 & 0 \\
1 & 0 & 4 & 1 & 0 & 0 & 5 \\
0 & 1 & 6 & 0 & 8 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 0 & 0 & 1 & 3 & 1 \\
\end{align*}
\]
Specialize **Linear Algebra** for reduction steps in GB computations.

```
1 3 0 0 7 1 0
1 0 4 1 0 0 5
0 1 6 0 8 0 1
0 1 0 0 0 7 0
0 0 0 0 1 3 1
```

Try to exploit underlying GB structure.
Specialize **Linear Algebra** for reduction steps in GB computations.

\[
\begin{align*}
\text{S-pair} &\quad \begin{cases} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\
1 & 0 & 4 & 1 & 0 & 0 & 5 \\
0 & 1 & 6 & 0 & 8 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
\end{cases} \\
\text{reducer} &\quad \leftarrow 0 & 0 & 0 & 0 & 1 & 3 & 1
\end{align*}
\]

Try to exploit underlying GB structure.
Specialize **Linear Algebra** for reduction steps in GB computations.

```
S-pair
{ 1 3 0 0 7 1 0
  1 0 4 1 0 0 5
  0 1 6 0 8 0 1
  0 1 0 0 0 7 0
}
reducer ← 0 0 0 0 1 3 1
```

Try to exploit underlying GB structure.
Specialize **Linear Algebra** for reduction steps in GB computations.

Try to exploit underlying GB structure.
Specialize **Linear Algebra** for reduction steps in GB computations.

Try to exploit underlying GB structure.

**Main idea**

Do a static **reordering before** the Gaussian Elimination to achieve a better initial shape. **Invert the reordering afterwards.**
1st step: Sort pivot and non-pivot columns

1 3 0 0 7 1 0
1 0 4 1 0 0 5
0 1 6 0 8 0 1
0 1 0 0 0 7 0
0 0 0 0 1 3 1
1st step: Sort pivot and non-pivot columns

```
1 3 0 0 7 1 0
1 0 4 1 0 0 5
0 1 6 0 8 0 1
0 1 0 0 0 7 0
0 0 0 0 1 3 1
```

Pivot column
1st step: Sort pivot and non-pivot columns

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pivot column
1st step: Sort pivot and non-pivot columns

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
1st step: Sort pivot and non-pivot columns
1st step: Sort pivot and non-pivot columns

Pivot column

Non-Pivot column
2nd step: Sort pivot and non-pivot rows

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>7</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
2nd step: Sort pivot and non-pivot rows

1 3 7 0 0 1 0
1 0 0 4 1 0 5
0 1 8 6 0 0 9
0 1 0 0 0 7 0
0 0 1 0 0 3 1

Pivot row
2nd step: Sort pivot and non-pivot rows

<table>
<thead>
<tr>
<th>1 3 7</th>
<th>0 0 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0</td>
<td>4 1 0 5</td>
</tr>
<tr>
<td>0 1 8</td>
<td>6 0 0 9</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 7 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 3 1</td>
</tr>
</tbody>
</table>

Pivot row | Non-Pivot row
2nd step: Sort pivot and non-pivot rows

Pivot row

Non-Pivot row
### 2nd step: Sort pivot and non-pivot rows

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>7</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
### 3rd step: Reduce lower left part to zero

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>
3rd step: Reduce lower left part to zero

\[
\begin{array}{cccccc}
1 & 0 & 0 & 4 & 1 & 0 \ 5 \\
0 & 1 & 0 & 0 & 0 & 7 \ 0 \\
0 & 0 & 1 & 0 & 0 & 3 \ 1 \\
1 & 3 & 7 & 0 & 0 & 1 \ 0 \\
0 & 1 & 8 & 6 & 0 & 0 \ 9 \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{cccccc}
1 & 0 & 0 & 4 & 1 & 0 \ 5 \\
0 & 1 & 0 & 0 & 0 & 7 \ 0 \\
0 & 0 & 1 & 0 & 0 & 3 \ 1 \\
0 & 0 & 0 & 7 & 10 & 3 \ 10 \\
0 & 0 & 0 & 6 & 0 & 2 \ 1 \\
\end{array}
\]
**4th step:** Reduce lower right part

\[
\begin{array}{cccc}
1 & 0 & 0 & 4 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 7 \\
0 & 0 & 0 & 6 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 5 \\
0 & 7 & 0 \\
0 & 3 & 1 \\
7 & 10 & 3 \\
6 & 0 & 2 \\
\end{array}
\]
4th step: Reduce lower right part
4th step: Reduce lower right part

\[
\begin{array}{cccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 10 & 3 & 10 \\
0 & 0 & 0 & 6 & 0 & 2 & 1 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{cccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 0 & 6 & 3 \\
0 & 0 & 0 & 0 & 4 & 1 & 5 \\
\end{array}
\]

5th step: Remap columns and get new polynomials for GB out of lower right part.
SO, WHAT DO “REAL WORLD” MATRICES FROM GB COMPUTATIONS LOOK LIKE?
WHAT OUR MATRICES LOOK LIKE
Some data about the matrix:

- **F4** computation of homogeneous **KATSURA-12**, degree 6 matrix
Some data about the matrix:

- **F4** computation of homogeneous *KATSURA-12*, degree 6 matrix
- Size 55MB
Some data about the matrix:

- \textbf{F4} computation of homogeneous \textsc{Katsura-12}, degree 6 matrix
- Size 55MB
- 24,006,869 nonzero elements (density: 5%)
Some data about the matrix:

• **F4** computation of homogeneous **KATSURA-12**, degree 6 matrix
• Size 55MB
• 24,006,869 nonzero elements (density: 5%)
• Dimensions:
  
  full matrix: 21,182 × 22,207
WHAT OUR MATRICES LOOK LIKE

Some data about the matrix:

- **F4** computation of homogeneous KATSURA-12, degree 6 matrix
- Size 55MB
- 24,006,869 nonzero elements (density: 5%)
- Dimensions:
  - full matrix: 21,182 × 22,207
  - upper-left: 17,915 × 17,915 - known pivots
  - lower-left: 3,267 × 17,915
  - upper-right: 17,915 × 4,292
  - lower-right: 3,267 × 4,292 - new information
WHAT OUR MATRICES LOOK LIKE
WHAT OUR MATRICES LOOK LIKE
HYBRID MATRIX MULTIPLICATION $A^{-1}B$
HYBRID MATRIX MULTIPLICATION $A^{-1}B$
REDUCE C TO ZERO
FEATURES OF GBLA
- **Open source** library written in **plain C**.
• Open source library written in plain C.

• Specialized linear algebra for GB computations.
- **Open source** library written in **plain C**.
- Specialized linear algebra for GB computations.
- **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.

[http://hpac.imag.fr/gbla](http://hpac.imag.fr/gbla)
• Open source library written in plain C.
• Specialized linear algebra for GB computations.
• Parallel implementation (OpenMP), scaling “nicely” up to 32 cores.
• Works over finite fields for 16-bit primes (at the moment).
• **Open source** library written in plain C.
• Specialized linear algebra for GB computations.
• **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.
• Works over finite fields for 16-bit primes (at the moment).
• **Several strategies** for splicing and reduction.
• **Open source** library written in **plain C**.
• Specialized linear algebra for GB computations.
• **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.
• Works over finite fields for 16-bit primes (at the moment).
• **Several strategies** for splicing and reduction.
• Includes **converter** from and to our dedicated matrix format.

[http://hpac.imag.fr/gbla](http://hpac.imag.fr/gbla)
• **Open source** library written in **plain C**.
• Specialized linear algebra for GB computations.
• **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.
• Works over finite fields for 16-bit primes (at the moment).
• Several strategies for splicing and reduction.
• Includes **converter** from and to our dedicated matrix format.
• Access to **huge matrix database**: > 500 matrices, > 280GB of data.
• **Open source** library written in plain C.
• Specialized linear algebra for GB computations.
• **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.
• Works over finite fields for 16-bit primes (at the moment).
• **Several strategies** for splicing and reduction.
• Includes **converter** from and to our dedicated matrix format.
• Access to **huge matrix database**: > 500 matrices, > 280GB of data.

http://hpac.imag.fr/gbla
Matrices from GB computations have **nonzero entries** often **grouped** in blocks.

**Horizontal Pattern**  If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$. 
Matrices from GB computations have nonzero entries often grouped in blocks.

**Horizontal Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$. 

**Vertical Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i+1,j} \neq 0$. 

Can be used to optimize AXPY and TRSM operations in FL reduction.  
Horizontal pattern taken care of canonically.  
Need to take care of vertical pattern.
Matrices from GB computations have **nonzero entries** often **grouped** in blocks.

**Horizontal Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$.

**Vertical Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i+1,j} \neq 0$.

- Can be used to optimize **AXPY** and **TRSM** operations in FL reduction.
Matrices from GB computations have **nonzero entries** often grouped in blocks.

**Horizontal Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$.

**Vertical Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i+1,j} \neq 0$.

- Can be used to optimize **AXPY** and **TRSM** operations in FL reduction.
- Horizontal pattern taken care of canonically.
Matrices from GB computations have **nonzero entries** often grouped in blocks.

**Horizontal Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$.

**Vertical Pattern**  
If $m_{i,j} \neq 0$ then often $m_{i+1,j} \neq 0$.

- Can be used to optimize **AXPY** and **TRSM** operations in FL reduction.
- Horizontal pattern taken care of canonically.
- Need to take care of vertical pattern.
Exploiting horizontal and vertical patterns in the TRSM step.
Consider the following two rows:

$$\begin{align*}
\mathbf{r}_1 &= [ 2 \ 3 \ 0 \ 1 \ 4 \ 0 \ 5 ], \\
\mathbf{r}_2 &= [ 1 \ 7 \ 0 \ 0 \ 3 \ 1 \ 2 ].
\end{align*}$$
Consider the following two rows:

\[
\begin{align*}
\mathbf{r}_1 & = [2 \ 3 \ 0 \ 1 \ 4 \ 0 \ 5 ], \\
\mathbf{r}_2 & = [1 \ 7 \ 0 \ 0 \ 3 \ 1 \ 2 ].
\end{align*}
\]

A sparse vector representation of the two rows would be given by

\[
\begin{align*}
\mathbf{r}_1\.\text{val} & = [2 \ 3 \ 1 \ 4 \ 5 ], \\
\mathbf{r}_1\.\text{pos} & = [0 \ 1 \ 3 \ 4 \ 6 ], \\
\mathbf{r}_2\.\text{val} & = [1 \ 7 \ 3 \ 1 \ 2 ], \\
\mathbf{r}_2\.\text{pos} & = [0 \ 1 \ 4 \ 5 \ 6 ].
\end{align*}
\]
Consider the following two rows:

\[
\begin{align*}
r1 &= [2 \ 3 \ 0 \ 1 \ 4 \ 0 \ 5], \\
r2 &= [1 \ 7 \ 0 \ 0 \ 3 \ 1 \ 2].
\end{align*}
\]

A sparse vector representation of the two rows would be given by

\[
\begin{align*}
r1.val &= [2 \ 3 \ 1 \ 4 \ 5], \\
r1.pos &= [0 \ 1 \ 3 \ 4 \ 6], \\
r2.val &= [1 \ 7 \ 3 \ 1 \ 2], \\
r2.pos &= [0 \ 1 \ 4 \ 5 \ 6].
\end{align*}
\]

A multiline vector representation of \(r1\) and \(r2\) is given by

\[
\begin{align*}
ml.val &= [2 \ 1 \ 3 \ 7 \ 1 \ 0 \ 4 \ 3 \ 0 \ 1 \ 5 \ 2], \\
ml.pos &= [0 \ 1 \ 3 \ 4 \ 5 \ 6].
\end{align*}
\]
Consider the following two rows:

\[
\begin{align*}
  r1 &= [2 \ 3 \ 0 \ 1 \ 4 \ 0 \ 5], \\
  r2 &= [1 \ 7 \ 0 \ 0 \ 3 \ 1 \ 2].
\end{align*}
\]

A sparse vector representation of the two rows would be given by

\[
\begin{align*}
  r1.val &= [2 \ 3 \ 1 \ 4 \ 5], \\
  r1.pos &= [0 \ 1 \ 3 \ 4 \ 6], \\
  r2.val &= [1 \ 7 \ 3 \ 1 \ 2], \\
  r2.pos &= [0 \ 1 \ 4 \ 5 \ 6].
\end{align*}
\]

A multiline vector representation of \( r1 \) and \( r2 \) is given by

\[
\begin{align*}
  ml.val &= [2 \ 1 \ 3 \ 7 \ 1 \ 0 \ 4 \ 3 \ 0 \ 1 \ 5 \ 2], \\
  ml.pos &= [0 \ 1 \ 3 \ 4 \ 5 \ 6].
\end{align*}
\]
• Number of initially known pivots (i.e. # rows of $A$ and $B$) is large compared to # rows of $C$ and $D$. 
NEW ORDER OF OPERATIONS

- Number of initially known pivots (i.e. # rows of $A$ and $B$) is large compared to # rows of $C$ and $D$.
- Most time of FL reduction is spent in TRSM step $A^{-1}B$. 

25
Number of initially known pivots (i.e. # rows of A and B) is large compared to # rows of C and D.

Most time of FL reduction is spent in TRSM step $A^{-1}B$.

Only interested in $D$ resp. rank of $M$?
NEW ORDER OF OPERATIONS

• Number of initially known pivots (i.e. # rows of A and B) is large compared to # rows of C and D.
• Most time of FL reduction is spent in TRSM step $A^{-1}B$.
• Only interested in $D$ resp. rank of $M$?

Change order of operations.
NEW ORDER OF OPERATIONS

- Number of initially known pivots (i.e. # rows of $A$ and $B$) is large compared to # rows of $C$ and $D$.
- Most time of FL reduction is spent in TRSM step $A^{-1}B$.
- Only interested in $D$ resp. rank of $M$?

Change order of operations.

1. Reduce $C$ directly with $A$ (store corresponding data in $C$).
• Number of initially known pivots (i.e. # rows of $A$ and $B$) is large compared to # rows of $C$ and $D$.
• Most time of FL reduction is spent in TRSM step $A^{-1}B$.
• Only interested in $D$ resp. rank of $M$?

Change order of operations.

1. Reduce $C$ directly with $A$ (store corresponding data in $C$).
2. Carry out corresponding operations from $B$ to $D$ using updated $C$. 
NEW ORDER OF OPERATIONS

• Number of initially known pivots (i.e. # rows of A and B) is large compared to # rows of C and D.
• Most time of FL reduction is spent in TRSM step $A^{-1}B$.
• Only interested in D resp. rank of M?

Change order of operations.

1. Reduce C directly with A (store corresponding data in C).
2. Carry out corresponding operations from B to D using updated C.
3. Reduce D.
Matrices are pretty sparse, but structured.
• Matrices are pretty sparse, but structured.
• GBLA supports two matrix formats, both use binary format.

<table>
<thead>
<tr>
<th>Size</th>
<th>Length</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>b</td>
<td>version number</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>m</td>
<td># rows</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>n</td>
<td># columns</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>p</td>
<td>prime / field characteristic</td>
</tr>
<tr>
<td>uint64_t</td>
<td>1</td>
<td>nnz</td>
<td># nonzero entries</td>
</tr>
<tr>
<td>uint16_t</td>
<td>nnz</td>
<td>data</td>
<td># entries in matrix</td>
</tr>
<tr>
<td>uint32_t</td>
<td>nnz</td>
<td>cols</td>
<td>column index of entry</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>rows</td>
<td>length of rows</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Matrices are pretty sparse, but structured.
• GBLA supports **two matrix formats**, both use binary format.
• GBLA includes a converter between the two supported formats and can also dump to Magma matrix format.
Matrices are pretty sparse, but structured.

GBLA supports **two matrix formats**, both use binary format.

GBLA includes a converter between the two supported formats and can also dump to Magma matrix format.

**Table 1:** Old matrix format (legacy version)

<table>
<thead>
<tr>
<th>Size</th>
<th>Length</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint32_t</code></td>
<td>1</td>
<td>b</td>
<td>version number</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>1</td>
<td>m</td>
<td># rows</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>1</td>
<td>n</td>
<td># columns</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>1</td>
<td>p</td>
<td>prime / field characteristic</td>
</tr>
<tr>
<td><code>uint64_t</code></td>
<td>1</td>
<td>nnz</td>
<td># nonzero entries</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td><code>nnz</code></td>
<td>data</td>
<td># entries in matrix</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td><code>nnz</code></td>
<td>cols</td>
<td>column index of entry</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>m</td>
<td>rows</td>
<td>length of rows</td>
</tr>
</tbody>
</table>
## Table 2: New matrix format (compressing data and cols)

<table>
<thead>
<tr>
<th>Size</th>
<th>Length</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>b</td>
<td>version number + information for data type of pdata</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>m</td>
<td># rows</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>n</td>
<td># columns</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>p</td>
<td>prime / field characteristic</td>
</tr>
<tr>
<td>uint64_t</td>
<td>1</td>
<td>nnz</td>
<td># nonzero entries</td>
</tr>
<tr>
<td>uint16_t</td>
<td>nnz</td>
<td>data</td>
<td>several rows are of type $x;f_j$</td>
</tr>
<tr>
<td>uint32_t</td>
<td>nnz</td>
<td>cols</td>
<td>can be compressed for consecutive elements</td>
</tr>
<tr>
<td>uint32_t</td>
<td>m</td>
<td>rows</td>
<td>length of rows</td>
</tr>
<tr>
<td>uint32_t</td>
<td>m</td>
<td>pmap</td>
<td>maps rows to pdata</td>
</tr>
<tr>
<td>uint64_t</td>
<td>1</td>
<td>k</td>
<td>size of compressed colid</td>
</tr>
<tr>
<td>uint64_t</td>
<td>k</td>
<td>colid</td>
<td>compression of columns:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single column entry masked via $(1 &lt;&lt; 31)$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$s$ consecutive entries starting at column $c$ are stored as “$c s”</td>
</tr>
<tr>
<td>uint32_t</td>
<td>1</td>
<td>pnb</td>
<td># polynomials</td>
</tr>
<tr>
<td>uint64_t</td>
<td>1</td>
<td>pnnz</td>
<td># nonzero coefficients in polynomials</td>
</tr>
<tr>
<td>uint32_t</td>
<td>pnb</td>
<td>prow</td>
<td>length of polynomial / row representation</td>
</tr>
<tr>
<td>xinty_t</td>
<td>pnnz</td>
<td>pdata</td>
<td>coefficients of polynomials</td>
</tr>
</tbody>
</table>
### GBLA MATRIX FORMATS – COMPARISON

**Table 3:** Storage and time efficiency of the new format

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size old</th>
<th>Size new</th>
<th>gzipped old</th>
<th>gzipped new</th>
<th>Time old</th>
<th>Time new</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4-kat14-mat9</td>
<td>2.3GB</td>
<td>0.74GB</td>
<td>1.2GB</td>
<td>0.29GB</td>
<td>230s</td>
<td>66s</td>
</tr>
<tr>
<td>F5-kat17-mat10</td>
<td>43GB</td>
<td>12GB</td>
<td>24GB</td>
<td>5.3GB</td>
<td>4419s</td>
<td>883s</td>
</tr>
</tbody>
</table>

• 1/3rd of memory usage.
• 1/4th of memory usage when compressed with gzip.
• Compression 5 times faster.
Table 3: Storage and time efficiency of the new format

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size old</th>
<th>Size new</th>
<th>gzipped old</th>
<th>gzipped new</th>
<th>Time old</th>
<th>Time new</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4-kat14-mat9</td>
<td>2.3GB</td>
<td>0.74GB</td>
<td>1.2GB</td>
<td>0.29GB</td>
<td>230s</td>
<td>66s</td>
</tr>
<tr>
<td>F5-kat17-mat10</td>
<td>43GB</td>
<td>12GB</td>
<td>24GB</td>
<td>5.3GB</td>
<td>4419s</td>
<td>883s</td>
</tr>
</tbody>
</table>

New format vs. Old format
Table 3: Storage and time efficiency of the new format

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size old</th>
<th>Size new</th>
<th>gzipped old</th>
<th>gzipped new</th>
<th>Time old</th>
<th>Time new</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4-kat14-mat9</td>
<td>2.3GB</td>
<td>0.74GB</td>
<td>1.2GB</td>
<td>0.29GB</td>
<td>230s</td>
<td>66s</td>
</tr>
<tr>
<td>F5-kat17-mat10</td>
<td>43GB</td>
<td>12GB</td>
<td>24GB</td>
<td>5.3GB</td>
<td>4419s</td>
<td>883s</td>
</tr>
</tbody>
</table>

New format vs. Old format

• 1/3rd of memory usage.
Table 3: Storage and time efficiency of the new format

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size old</th>
<th>Size new</th>
<th>gzipped old</th>
<th>gzipped new</th>
<th>Time old</th>
<th>Time new</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4-kat14-mat9</td>
<td>2.3GB</td>
<td>0.74GB</td>
<td>1.2GB</td>
<td>0.29GB</td>
<td>230s</td>
<td>66s</td>
</tr>
<tr>
<td>F5-kat17-mat10</td>
<td>43GB</td>
<td>12GB</td>
<td>24GB</td>
<td>5.3GB</td>
<td>4419s</td>
<td>883s</td>
</tr>
</tbody>
</table>

New format vs. Old format

- 1/3rd of memory usage.
- 1/4th of memory usage when compressed with gzip.
**Table 3:** Storage and time efficiency of the new format

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size old</th>
<th>Size new</th>
<th>gzipped old</th>
<th>gzipped new</th>
<th>Time old</th>
<th>Time new</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4-kat14-mat9</td>
<td>2.3GB</td>
<td>0.74GB</td>
<td>1.2GB</td>
<td>0.29GB</td>
<td>230s</td>
<td>66s</td>
</tr>
<tr>
<td>F5-kat17-mat10</td>
<td>43GB</td>
<td>12GB</td>
<td>24GB</td>
<td>5.3GB</td>
<td>4419s</td>
<td>883s</td>
</tr>
</tbody>
</table>

**New format vs. Old format**

- 1/3rd of memory usage.
- 1/4th of memory usage when compressed with gzip.
- Compression 4 – 5 times faster.
SOME BENCHMARKS
All timings in seconds.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>FL Implementation</th>
<th>GBLA v0.1</th>
<th>GBLA v0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix/Threads:</td>
<td>1 16 32</td>
<td>1 16 32</td>
<td>1 16 32</td>
</tr>
<tr>
<td>F5-kat13-mat5</td>
<td>16.7 2.7 2.3</td>
<td>14.5 2.02 1.87</td>
<td>14.5 1.73 1.61</td>
</tr>
<tr>
<td>F5-kat13-mat6</td>
<td>27.3 4.15 4.0</td>
<td>23.9 3.08 2.65</td>
<td>25.9 3.03 2.28</td>
</tr>
<tr>
<td>F5-kat14-mat7</td>
<td>139 17.4 16.6</td>
<td>142 13.4 10.6</td>
<td>122 11.2 8.64</td>
</tr>
<tr>
<td>F5-kat14-mat8</td>
<td>181 24.95 23.1</td>
<td>177 16.9 12.7</td>
<td>158 14.7 10.5</td>
</tr>
<tr>
<td>F5-kat15-mat7</td>
<td>629 61.8 55.6</td>
<td>633 55.1 38.2</td>
<td>553 46.3 30.7</td>
</tr>
<tr>
<td>F5-kat16-mat6</td>
<td>1,203 110 83.3</td>
<td>1,147 98.7 69.9</td>
<td>988 73.9 49.0</td>
</tr>
<tr>
<td>F5-mr-9-10-7-mat3</td>
<td>591 70.8 71.3</td>
<td>733 57.3 37.9</td>
<td>747 52.8 33.2</td>
</tr>
<tr>
<td>F5-cyclic-10-mat20</td>
<td>2,589 274 209</td>
<td>2,074 171 152</td>
<td></td>
</tr>
<tr>
<td>F5-cyclic-10-sym-mat17</td>
<td>2,463 465 405</td>
<td>2,391 275 245</td>
<td></td>
</tr>
</tbody>
</table>
# GBLA vs. Magma v2.20-10

All timings in seconds.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Magma</th>
<th>GBLA v0.1</th>
<th>GBLA v0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix/Threads:</td>
<td>1</td>
<td>16 32</td>
<td>16 32</td>
</tr>
<tr>
<td>F4-kat12-mat9</td>
<td>11.2</td>
<td>11.4 1.46 1.60</td>
<td>11.3 1.40 1.40</td>
</tr>
<tr>
<td>F4-kat13-mat2</td>
<td>0.94</td>
<td>1.18 0.38 0.61</td>
<td>1.11 0.26 0.33</td>
</tr>
<tr>
<td>F4-kat13-mat3</td>
<td>9.33</td>
<td>11.0 1.70 3.10</td>
<td>8.51 1.07 1.13</td>
</tr>
<tr>
<td>F4-kat13-mat9</td>
<td>168</td>
<td>165 16.0 11.8</td>
<td>114 9.74 6.83</td>
</tr>
<tr>
<td>F4-kat14-mat8</td>
<td>2,747</td>
<td>2,545 207 165</td>
<td>1,338 104 65.8</td>
</tr>
<tr>
<td>F4-kat15-mat7</td>
<td>10,345</td>
<td>9,514 742 537</td>
<td>4,198 298 195</td>
</tr>
<tr>
<td>F4-kat15-mat8</td>
<td>13,936</td>
<td>12,547 961 604</td>
<td>6,508 470 283</td>
</tr>
<tr>
<td>F4-kat15-mat9</td>
<td>24,393</td>
<td>22,247 1,709 1,256</td>
<td>10,923 779 450</td>
</tr>
<tr>
<td>F4-rand16-d2-2-mat6</td>
<td>4,902</td>
<td>375 219</td>
<td>3,054 224 133</td>
</tr>
<tr>
<td>F4-rand16-d2-3-mat8</td>
<td>48,430</td>
<td>3,473 2,119</td>
<td>26,533 1,782 1,027</td>
</tr>
<tr>
<td>F4-rand16-d2-3-mat9</td>
<td>6,956</td>
<td>4,470</td>
<td>3,214 1,776</td>
</tr>
<tr>
<td>F4-rand16-d2-3-mat10^1</td>
<td>9,691</td>
<td>6,223</td>
<td>3,820 1,972</td>
</tr>
</tbody>
</table>

**Note** that Magma generates slightly bigger matrices for the given examples.

^1Reconstruction fails due to memory consumption
OUTLOOK
• Optimizing GBLA for **floating point** and **32-bit unsigned int arithmetic**.
• Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.

• Connect GBLA to Singular to get a tentative F4.
• Optimizing GBLA for **floating point** and **32-bit unsigned int arithmetic**.

• Connect GBLA to **Singular** to get a tentative **F4**.

• Creation of a new open source plain C library **GBTOOLS**.
Different Approaches

- Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
- Connect GBLA to Singular to get a tentative F4.
- Creation of a new open source plain C library GBTOOLS.
- Deeper investigation on parallelization on networks.
• Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
• Connect GBLA to Singular to get a tentative F4.
• Creation of a new open source plain C library GBTOOLS.
• Deeper investigation on parallelization on networks.
• First steps exploiting heterogeneous CPU/GPU platforms for GBLA.
Buchberger, B. 
PhD thesis, University of Innsbruck, Austria

Buchberger, B. 
EUROSAM ’79, An International Symposium on Symbolic and Algebraic Manipulation

Buchberger, B. 
Multidimensional Systems Theory, D. Reidel Publication Company

Eder, C. and Faugère, J.-C. 
*A survey on signature-based Groebner basis algorithms*, 2014. 
http://arxiv.org/abs/1404.1774

Faugère, J.-C. 
Journal of Pure and Applied Algebra

Faugère, J.-C. 
*A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*, 2002. 
Proceedings of the 2002 international symposium on Symbolic and algebraic computation

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

Gebauer, R. and Möller, H. M. 
Journal of Symbolic Computation
THANK YOU!
COMMENTS? QUESTIONS?