Parallel Gröbner Basis Algorithms over Finite Fields

Christian Eder and Jean-Charles Faugère

University of Kaiserslautern

October 14, 2016
1. Gröbner Bases and Buchberger’s Algorithm

2. Faugère’s F4 Algorithm

3. Specialized Linear Algebra for Gröbner Basis Computation

4. GBLA – A Gröbner Basis Linear Algebra Library

5. GB – A Gröbner Basis Library

6. Some benchmarks

7. Outlook
S-polynomials
Let \( f \neq 0, g \neq 0 \in \mathcal{R} \) and let \( \lambda = \text{lcm} (\text{lt} (f), \text{lt} (g)) \) be the least common multiple of \( \text{lt} (f) \) and \( \text{lt} (g) \). The **S-polynomial** between \( f \) and \( g \) is given by

\[
\text{spol} (f, g) := \frac{\lambda}{\text{lt} (f)} f - \frac{\lambda}{\text{lt} (g)} g.
\]
**S-polynomials**

Let \( f \neq 0, g \neq 0 \in \mathcal{R} \) and let \( \lambda = \text{lcm}(\text{lt}(f), \text{lt}(g)) \) be the least common multiple of \( \text{lt}(f) \) and \( \text{lt}(g) \). The **S-polynomial** between \( f \) and \( g \) is given by

\[
\text{spol}(f, g) := \frac{\lambda}{\text{lt}(f)} f - \frac{\lambda}{\text{lt}(g)} g.
\]

---

**Buchberger’s criterion [2]**

Let \( I = \langle f_1, \ldots, f_m \rangle \) be an ideal in \( \mathcal{R} \). A finite subset \( G \subset \mathcal{R} \) is a **Gröbner basis for** \( I \) if \( G \subset I \) and for all \( f, g \in G \) : \( \text{spol}(f, g) \to 0 \).
Buchberger’s algorithm

**Input:** Ideal $I = \langle f_1, \ldots, f_m \rangle$

**Output:** Gröbner basis $G$ for $I$

1. $G \leftarrow \emptyset$
2. $G \leftarrow G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P \leftarrow \{\text{spol} (f_i, f_j) \mid f_i, f_j \in G, i > j\}$
**Buchberger’s algorithm**

**Input:** Ideal $I = \langle f_1, \ldots, f_m \rangle$

**Output:** Gröbner basis $G$ for $I$

1. $G \leftarrow \emptyset$
2. $G \leftarrow G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P \leftarrow \{\text{spol} (f_i, f_j) \mid f_i, f_j \in G, i > j\}$
4. Choose $p \in P$, $P \leftarrow P \setminus \{p\}$
Buchberger’s algorithm

**Input:** Ideal \( I = \langle f_1, \ldots, f_m \rangle \)

**Output:** Gröbner basis \( G \) for \( I \)

1. \( G \leftarrow \emptyset \)
2. \( G \leftarrow G \cup \{f_i\} \) for all \( i \in \{1, \ldots, m\} \)
3. Set \( P \leftarrow \{\text{spol}(f_i, f_j) \mid f_i, f_j \in G, i > j\} \)
4. Choose \( p \in P, P \leftarrow P \setminus \{p\} \)
   
   (a) If \( p \not\xrightarrow{G} 0 \) \quad \textbf{no new information}
       Go on with the next element in \( P \).

   (b) If \( p \xrightarrow{G} q \neq 0 \) \quad \textbf{new information}
       Build new S-pair with \( q \) and add them to \( P \).
       Add \( q \) to \( G \).
       Go on with the next element in \( P \).

5. When \( P = \emptyset \) we are done and \( G \) is a Gröbner basis for \( I \).
Faugère’s F4 algorithm
Faugère’s F4 algorithm

**Input:** Ideal \( I = \langle f_1, \ldots, f_m \rangle \)

**Output:** Gröbner basis \( G \) for \( I \)

1. \( G \leftarrow \emptyset \)
2. \( G \leftarrow G \cup \{f_i\} \) for all \( i \in \{1, \ldots, m\} \)
3. Set \( P \leftarrow \{(af, bg) \mid f, g \in G\} \)
4. \( d \leftarrow 0 \)
5. while \( P \neq \emptyset \):

...
Faugère’s F4 algorithm

**Input:** Ideal $I = \langle f_1, \ldots, f_m \rangle$

**Output:** Gröbner basis $G$ for $I$

1. $G \leftarrow \emptyset$
2. $G \leftarrow G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P \leftarrow \{(af, bg) \mid f, g \in G\}$
4. $d \leftarrow 0$
5. while $P \neq \emptyset$:
   (a) $d \leftarrow d + 1$
   (b) $P_d \leftarrow \text{Select}(P)$, $P \leftarrow P \setminus P_d$
   (c) $L_d \leftarrow \{af, bg \mid (af, bg) \in P_d\}$
   (d) $L_d \leftarrow \text{Symbolic Preprocessing}(L_d, G)$
   (e) $F_d \leftarrow \text{Reduction}(L_d, G)$
   (f) for $h \in F_d$:
      ▶ If $\text{lt}(h) \notin L(G)$ (all other $h$ are “useless”):
         ▷ $P \leftarrow P \cup \{\text{new pairs with } h\}$
         ▷ $G \leftarrow G \cup \{h\}$
6. Return $G$
1. Select a subset $P_d$ of $P$, not only one element.
2. Do a symbolic preprocessing:
   Search and store reducers, but do not reduce.
3. Do a full reduction of $P_d$ at once:
   Reduce a subset of $\mathcal{R}$ by a subset of $\mathcal{R}$.
1. Select a subset $P_d$ of $P$, not only one element.
2. Do a symbolic preprocessing:
   Search and store reducers, but do not reduce.
3. Do a full reduction of $P_d$ at once:
   Reduce a subset of $\mathcal{R}$ by a subset of $\mathcal{R}$

If $\textbf{Select}(P)$ selects only one pair F4 is just Buchberger’s algorithm.
Usually one chooses the normal selection strategy, i.e. all pairs of lowest degree.
Symbolic preprocessing

**Input:** $L, G$ finite subsets of $\mathcal{R}$

**Output:** a finite subset of $\mathcal{R}$

1. $F \leftarrow L$
2. $D \leftarrow L(F)$ (S-pairs already reduce lead terms)
3. while $T(F) \neq D$:
   (a) Choose $m \in T(F) \setminus D$, $D \leftarrow D \cup \{m\}$.
   (b) If $m \in L(G) \Rightarrow \exists g \in G$ and $\lambda \in \mathcal{R}$ such that $\lambda \text{lt}(g) = m$
      $F \leftarrow F \cup \{\lambda g\}$
4. Return $F$
Symbolic preprocessing

**Input:** $L, G$ finite subsets of $\mathcal{R}$

**Output:** a finite subset of $\mathcal{R}$

1. $F \leftarrow L$

2. $D \leftarrow L(F)$ (S-pairs already reduce lead terms)

3. while $T(F) \neq D$:
   
   (a) Choose $m \in T(F) \setminus D$, $D \leftarrow D \cup \{m\}$.
   
   (b) If $m \in L(G) \Rightarrow \exists g \in G$ and $\lambda \in \mathcal{R}$ such that $\lambda \text{lt}(g) = m$
   
   $\triangleright F \leftarrow F \cup \{\lambda g\}$

4. Return $F$

We optimize this soon!
Input: $L$ finite subsets of $\mathcal{R}$
Output: a finite subset of $\mathcal{R}$

1. $M \leftarrow$ Macaulay matrix of $L$
2. $M \leftarrow$ Gaussian Elimination of $M$ (Linear algebra)
3. $F \leftarrow$ polynomials from rows of $M$
4. Return $F$
**Input:** $L$ finite subsets of $\mathcal{R}$

**Output:** a finite subset of $\mathcal{R}$

1. $M \leftarrow$ Macaulay matrix of $L$
2. $M \leftarrow$ Gaussian Elimination of $M$ (Linear algebra)
3. $F \leftarrow$ polynomials from rows of $M$
4. Return $F$

**Macaulay matrix:**

- **columns** \( \equiv \) monomials (sorted by monomial order \(<\) )
- **rows** \( \equiv \) coefficients of polynomials in $L$
\[ R = \mathbb{Q}[a, b, c, d], < \text{ denotes DRL and we use the normal selection strategy for } \textbf{Select}(P). \]
\[ I = \langle f_1, \ldots, f_4 \rangle, \text{ where} \]
\[ f_1 = abcd - 1, \]
\[ f_2 = abc + abd + acd + bcd, \]
\[ f_3 = ab + bc + ad + cd, \]
\[ f_4 = a + b + c + d. \]
Example: Cyclic-4

\( \mathcal{R} = \mathbb{Q}[a, b, c, d] \), \(<\) denotes DRL and we use the normal selection strategy for \textbf{Select}(P).

\( I = \langle f_1, \ldots, f_4 \rangle \), where

\[
\begin{align*}
  f_1 &= abcd - 1, \\
  f_2 &= abc + abd + acd + bcd, \\
  f_3 &= ab + bc + ad + cd, \\
  f_4 &= a + b + c + d.
\end{align*}
\]

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf_4)\} \), thus \( L_1 = \{f_3, bf_4\} \).
\( \mathcal{R} = \mathbb{Q}[a, b, c, d], < \) denotes DRL and we use the normal selection strategy for \textbf{Select}(P).

\( l = \langle f_1, \ldots, f_4 \rangle \), where

\begin{align*}
    f_1 &= abcd - 1, \\
    f_2 &= abc + abd + acd + bcd, \\
    f_3 &= ab + bc + ad + cd, \\
    f_4 &= a + b + c + d.
\end{align*}

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf_4)\} \), thus \( L_1 = \{f_3, bf_4\} \).

Let us do symbolic preprocessing:

\[
\begin{align*}
    T(L_1) &= \{ab, b^2, bc, ad, bd, cd\} \\
    L_1 &= \{f_3, bf_4\}
\end{align*}
\]
\( R = \mathbb{Q}[a, b, c, d], < \) denotes DRL and we use the normal selection strategy for \textbf{Select} \((P)\).

\( I = \langle f_1, \ldots, f_4 \rangle \), where

\[
\begin{align*}
    f_1 &= abcd - 1, \\
    f_2 &= abc + abd + acd + bcd, \\
    f_3 &= ab + bc + ad + cd, \\
    f_4 &= a + b + c + d.
\end{align*}
\]

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf_4)\} \), thus \( L_1 = \{f_3, bf_4\} \).

Let us do \textit{symbolic preprocessing}:

\[
\begin{align*}
    T(L_1) &= \{ab, b^2, bc, ad, bd, cd\} \\
    L_1 &= \{f_3, bf_4\}.
\end{align*}
\]

\( b^2 \not\in L(G) \).
Example: Cyclic-4

\[ \mathcal{R} = \mathbb{Q}[a, b, c, d], < \text{ denotes DRL and we use the normal selection strategy for } \textbf{Select}(P). \]

\[ l = \langle f_1, \ldots, f_4 \rangle, \text{ where} \]

\[
\begin{align*}
f_1 &= abcd - 1, \\
f_2 &= abc + abd + acd + bcd, \\
f_3 &= ab + bc + ad + cd, \\
f_4 &= a + b + c + d.
\end{align*}
\]

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf_4)\} \), thus \( L_1 = \{f_3, bf_4\} \).

Let us do symbolic preprocessing:

\[
\begin{align*}
T(L_1) &= \{ab, b^2, bc, ad, bd, cd\} \\
L_1 &= \{f_3, bf_4\}
\end{align*}
\]

\( b^2 \notin L(G), bc \notin L(G), \)
\( \mathcal{R} = \mathbb{Q}[a, b, c, d], < \) denotes DRL and we use the normal selection strategy for \textbf{Select}(P).

\( I = \langle f_1, \ldots, f_4 \rangle \), where

- \( f_1 = abcd - 1 \),
- \( f_2 = abc + abd + acd + bcd \),
- \( f_3 = ab + bc + ad + cd \),
- \( f_4 = a + b + c + d \).

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf_4)\} \), thus \( L_1 = \{f_3, bf_4\} \).

Let us do \textit{symbolic preprocessing}:

\[
\begin{align*}
T(L_1) &= \{ab, b^2, bc, ad, bd, cd, d^2\} \\
L_1 &= \{f_3, bf_4, df_4\}
\end{align*}
\]

\( b^2 \not\in L(G), bc \not\in L(G), d \text{ if } (f_4) = ad, \)
\( R = \mathbb{Q}[a, b, c, d], < \) denotes DRL and we use the normal selection strategy for \textbf{Select}(P).

\( I = \langle f_1, \ldots, f_4 \rangle \), where

\[
\begin{align*}
  f_1 & = abcd - 1, \\
  f_2 & = abc + abd + acd + bcd, \\
  f_3 & = ab + bc + ad + cd, \\
  f_4 & = a + b + c + d.
\end{align*}
\]

We start with \( G = \{f_4\} \) and \( P_1 = \{(f_3, bf)\} \), thus \( L_1 = \{f_3, bf\} \). Let us do symbolic preprocessing:

\[
\begin{align*}
  T(L_1) & = \{ab, b^2, bc, ad, bd, cd, d^2\} \\
  L_1 & = \{f_3, bf, df\}
\end{align*}
\]

\( b^2 \notin L(G), bc \notin L(G), d \text{lt}(f_4) = ad, \) all others also \( \notin L(G) \),
Now **reduction**: Convert polynomial data $L_1$ to Macaulay Matrix $M_1$

\[
\begin{align*}
\text{df}_4 & \begin{pmatrix} ab & b^2 & bc & ad & bd & cd & d^2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \\
\text{f}_3 & \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \\
\text{bf}_4 & \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}
\end{align*}
\]
Now **reduction**:
Convert polynomial data $L_1$ to Macaulay Matrix $M_1$

\[
\begin{align*}
    df_4 & \begin{pmatrix} ab & b^2 & bc & ad & bd & cd & d^2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \\
    f_3 & \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \\
    bf_4 & \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}
\end{align*}
\]

**Gaussian Elimination of $M_1$:**

\[
\begin{align*}
    df_4 & \begin{pmatrix} ab & b^2 & bc & ad & bd & cd & d^2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \\
    f_3 & \begin{pmatrix} 1 & 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \\
    bf_4 & \begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 0 & 1 \end{pmatrix}
\end{align*}
\]
Convert matrix data back to polynomial structure $F_1$:

$$
\begin{pmatrix}
\begin{array}{ccccccc}
ab & b^2 & bc & ad & bd & cd & d^2 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & -1 & 0 & -1 \\
0 & 1 & 0 & 0 & 2 & 0 & 1 \\
\end{array}
\end{pmatrix}
$$

$$
F_1 = \left\{ \frac{\begin{array}{c}
ad + bd + cd + d^2 \\
\end{array}}{f_5}, \frac{\begin{array}{c}
ab + bc - bd - d^2 \\
\end{array}}{f_6}, \frac{\begin{array}{c}
b^2 + 2bd + d^2 \\
\end{array}}{f_7} \right\}
$$
Convert matrix data back to polynomial structure $F_1$:

\[
\begin{pmatrix}
ab & b^2 & bc & ad & bd & cd & d^2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & -1 & 0 & -1 \\
0 & 1 & 0 & 0 & 2 & 0 & 1 \\
\end{pmatrix}
\]

\[
F_1 = \left\{\frac{ad + bd + cd + d^2}{f_5}, \frac{ab + bc - bd - d^2}{f_6}, \frac{b^2 + 2bd + d^2}{f_7}\right\}
\]

\[\text{lt} (f_5), \text{lt} (f_6) \in L(G), \text{ so } G \leftarrow G \cup \{f_7\}.\]
Next round:

\[ G = \{ f_4, f_7 \}, \; P_2 = \{ (f_2, bcf_4) \}, \; L_2 = \{ f_2, bcf_4 \} \]
Next round:

\[ G = \{ f_4, f_7 \}, \quad P_2 = \{ (f_2, bcf_4) \}, \quad L_2 = \{ f_2, bcf_4 \}. \]

We can simplify the computations:

\[ \text{lt}(bcf_4) = abc = \text{lt}(cf_6). \]

\( f_6 \) possibly better reduced than \( f_4 \). (\( f_6 \) is not in \( G \! \))

\[ \Rightarrow L_2 = \{ f_2, cf_6 \} \]
Next round:

\[ G = \{f_4, f_7\}, \ P_2 = \{(f_2, bcf_4)\}, \ L_2 = \{f_2, bcf_4\}. \]

We can simplify the computations:

\[ \text{lt} (bcf_4) = abc = \text{lt} (cf_6). \]

\(f_6\) possibly better reduced than \(f_4\). \((f_6\) is not in \(G!\))

\[ \implies L_2 = \{f_2, cf_6\} \]

Symbolic preprocessing:

\[
\begin{align*}
T(L_2) & = \{abc, bc^2, abd, acd, bcd, cd^2\} \\
L_2 & = \{f_2, cf_6\}
\end{align*}
\]
Example: Cyclic-4

Next round:

\[ G = \{f_4, f_7\}, \quad P_2 = \{(f_2, bcf_4)\}, \quad L_2 = \{f_2, bcf_4\}. \]

We can simplify the computations:

\[ \text{lt}(bcf_4) = abc = \text{lt}(cf_6). \]

\( f_6 \) possibly better reduced than \( f_4 \). (\( f_6 \) is not in \( G \))

\[ \Rightarrow \quad L_2 = \{f_2, cf_6\} \]

Symbolic preprocessing:

\[
\begin{align*}
T(L_2) & = \{abc, bc^2, abd, acd, bcd, cd^2\} \\
L_2 & = \{f_2, cf_6, \} \\
bc^2 & \notin L(G),
\end{align*}
\]
Next round:

\[ G = \{f_4, f_7\}, \ P_2 = \{(f_2, bcf_4)\}, \ L_2 = \{f_2, bcf_4\}. \]

We can simplify the computations:

\[ \text{lt}(bcf_4) = abc = \text{lt}(cf_6). \]

\( f_6 \) possibly better reduced than \( f_4 \). \((f_6 \text{ is not in } G!\)

\[ \implies L_2 = \{f_2, cf_6\} \]

Symbolic preprocessing:

\[
\begin{align*}
T(L_2) & = \{abc, bc^2, abd, acd, bcd, cd^2\} \\
L_2 & = \{f_2, cf_6, \}
\end{align*}
\]

\( bc^2 \notin L(G), \ abd = \text{lt}(bdf_4), \text{ but also } abd = \text{lt}(bf_5)! \)
Next round:

\[ G = \{f_4, f_7\}, \ P_2 = \{(f_2, bcf_4)\}, \ L_2 = \{f_2, bcf_4\}. \]

We can simplify the computations:

\[ \text{lt}(bcf_4) = abc = \text{lt}(cf_6). \]

\(f_6\) possibly better reduced than \(f_4\). (\(f_6\) is not in \(G!\))

\[ \implies L_2 = \{f_2, cf_6\} \]

Symbolic preprocessing:

\[
T(L_2) = \{abc, bc^2, abd, acd, bcd, cd^2\} \\
L_2 = \{f_2, cf_6\} 
\]

\(bc^2 \notin L(G), \ abd = \text{lt}(bdf_4), \) but also \(abd = \text{lt}(bf_5)\)!

Let us investigate this in more detail.
**Idea**
Replace $u \cdot f$ by $(wv) \cdot g$ where $vg \in F_i$ for a previous reduction step.
⇒ Reuse rows that are reduced but not “in” $G$. 

Note ▶ T ries to reuse all rows from old matrices.
▶ We also simplify generators of S-pairs, as we have done in our example: $(f_2; bcf_4) = (f_2; cf_6)$.
▶ One can also choose “better” reducers by other properties, not only “last reduced one”.

In our example: Choose $bf_5$ as reducer, not $bdf_4$. 

Interlude – Simplify
Idea
Replace $u \cdot f$ by $(wv) \cdot g$ where $vg \in F_i$ for a previous reduction step.
⇒ Reuse rows that are reduced but not “in” $G$.

Note
- Tries to reuse all rows from old matrices.
  ⇒ We need to keep them in memory.

- We also simplify generators of S-pairs, as we have done in our example: $(f_2, bcf_4) \implies (f_2, cf_6)$.

- One can also choose “better” reducers by other properties, not only “last reduced one”.

14 / 50
**Idea**
Replace $u \cdot f$ by $(wv) \cdot g$ where $vg \in F_i$ for a previous reduction step.
⇒ Reuse rows that are reduced but not “in” $G$.

**Note**
- Tries to reuse all rows from old matrices.
  ⇒ We need to keep them in memory.
- We also simplify generators of S-pairs, as we have done in our example: $(f_2, bcf_4) \implies (f_2, cf_6)$.
- One can also choose “better” reducers by other properties, not only “last reduced one”.

In our example:
Choose $bf_5$ as reducer, not $bdf_4$. 
Symbolic preprocessing - now with simplify:

\[
\begin{align*}
T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2\} \\
L_2 &= \{f_2, cf_6\}
\end{align*}
\]

\[bc^2 \notin L(G),\]
Symbolic preprocessing - now with simplify:

\[
T(L_2) = \{ abc, bc^2, abd, acd, bcd, cd^2 \} \\
L_2 = \{ f_2, cf_6 \}
\]

\( bc^2 \notin L(G) \), \( abd = \text{lt} (bf_5) \)
Symbolic preprocessing - now with simplify:

\[ T(L_2) = \{ abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d \} \]
\[ L_2 = \{ f_2, cf_6, bf_5 \} \]

\( bc^2 \notin L(G) \), \( abd = \text{lt}(bf_5) \).
Symbolic preprocessing - now with simplify:

\[
T(L_2) = \{abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d, \ldots \}
\]
\[
L_2 = \{f_2, cf_6, bf_5, cf_5, df_7\}
\]

\[bc^2 \notin L(G), abd = \text{lt}(bf_5),\] and so on.
Symbolic preprocessing - now with simplify:

\[ T(L_2) = \{ abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d, \ldots \} \]
\[ L_2 = \{ f_2, cf_6, bf_5, cf_5, df_7 \} \]

\( bc^2 \notin L(G) \), \( abd = \text{lt} (bf_5) \), and so on.

Now try to exploit the special structure of the Macaulay matrices.
Specialized Linear Algebra for Gröbner Basis Computation
Specialize Linear Algebra for reduction steps in GB computations.
Specialize Linear Algebra for reduction steps in GB computations.

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

\[
\begin{align*}
1 & \ 3 \ 0 \ 0 \ 7 \ 1 \ 0 \\
1 & \ 0 \ 4 \ 1 \ 0 \ 0 \ 5 \\
0 & \ 1 \ 6 \ 0 \ 8 \ 0 \ 1 \\
0 & \ 1 \ 0 \ 0 \ 0 \ 7 \ 0 \\
0 & \ 0 \ 0 \ 0 \ 1 \ 3 \ 1 \\
\end{align*}
\]

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

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

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

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

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

<table>
<thead>
<tr>
<th>S-pair</th>
<th>1 3 0 0 7 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 0 4 1 0 0 5</td>
</tr>
<tr>
<td>S-pair</td>
<td>0 1 6 0 8 0 1</td>
</tr>
<tr>
<td></td>
<td>0 1 0 0 7 0</td>
</tr>
<tr>
<td>reducer</td>
<td>← 0 0 0 0 1 3 1</td>
</tr>
</tbody>
</table>

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

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

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

```
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  | Non-Pivot column
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

\[
\begin{array}{cccc}
1 & 3 & 0 & 0 \\
1 & 0 & 4 & 1 \\
0 & 1 & 6 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{array}
\quad \quad \quad \quad
\begin{array}{cccc}
1 & 3 & 7 & 0 \\
1 & 0 & 0 & 4 \\
0 & 1 & 8 & 6 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{array}
\]

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

\[
\begin{align*}
1 & 3 & 7 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 8 & 6 & 0 & 0 & 9 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\end{align*}
\]
**2nd step:** Sort pivot and non-pivot rows

\[
\begin{align*}
&1 \ 3 \ 7 \ 0 \ 0 \ 1 \ 0 \\
&1 \ 0 \ 0 \ 4 \ 1 \ 0 \ 5 \\
&0 \ 1 \ 8 \ 6 \ 0 \ 0 \ 9 \\
&0 \ 1 \ 0 \ 0 \ 0 \ 7 \ 0 \\
&0 \ 0 \ 1 \ 0 \ 0 \ 3 \ 1 \\
\end{align*}
\]
**2nd step**: Sort pivot and non-pivot rows

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

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

Pivot row
Non-Pivot row
3rd step: Reduce lower left part to zero

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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
4th step: Reduce lower right part

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

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

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

**5th step**: Remap columns and get new polynomials for GB out of lower right part.
“Real world” matrices?
What our matrices look like
What our matrices look like

**Characteristics of this matrix**

- **F4** computation of homogeneous *Katsura-12*, degree 6 matrix
- Size 55MB
What our matrices look like

Characteristics of this matrix

- **F4** computation of homogeneous *Katsura-12*, degree 6 matrix
- Size 55MB
- 24,006,869 nonzero elements (density: 5%)
What our matrices look like

Characteristics of this matrix

- **F4** computation of homogeneous *Katsura-12*, degree 6 matrix
- Size 55MB
- 24,006,869 nonzero elements (density: 5%)
- Dimensions:
  - full matrix: $21,182 \times 22,207$
What our matrices look like

Characteristics of this 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
Gaussian Elimination on D
GBLA – A Gröbner Basis Linear Algebra Library
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.
- **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.

[[http://hpac.imag.fr/gbla]]
[[https://www.github.com/ederc/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.
- **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.
Library Overview

- **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
https://www.github.com/ederc/gbla
Exploiting block structures in GB matrices
Matrices from GB computations have nonzero entries often grouped in blocks.

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

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 \( m_{ij} \neq 0 \) then often \( m_{i,j+1} \neq 0 \).

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

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

**Vertical Pattern**  If \( m_{ij} \neq 0 \) 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 $m_{ij} \neq 0$ then often $m_{ij+1} \neq 0$.

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

**Vertical Pattern**  If $m_{ij} \neq 0$ 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 block structures in GB matrices

Exploiting horizontal and vertical patterns in the TRSM step.
Consider the following two rows:

\[
\begin{align*}
  r1 &= [2 \ 3 \ 0 \ 1 \ 4 \ 0 \ 5], \\
  r2 &= [1 \ 7 \ 0 \ 0 \ 3 \ 1 \ 2].
\end{align*}
\]
Multiline data structure – an example

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

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

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

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

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

A multiline vector representation of \( \mathbf{r}_1 \) and \( \mathbf{r}_2 \) is given by

\[
\begin{align*}
\text{ml}: \text{val} &= [ 2 \ 1 \ 3 \ 7 \ 1 \ 0 \ 4 \ 3 \ 0 \ 1 \ 5 \ 2 ], \\
\text{ml}: \text{pos} &= [ 0 \ 1 \ 3 \ 4 \ 5 \ 6 ].
\end{align*}
\]
New order of operations

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

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 $D$.

This leads to reduced matrices that keep $A$ and $B$ untouched, i.e. the computation has a smaller memory footprint.
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 $A^{-1}B$. 

This leads to reduced matrices that keep $A$ and $B$ untouched, i.e. the computation has a smaller memory footprint.
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 $M$?
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 $C$).
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 $C$. 
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 $D$. 
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 $D$.

This leads to reduced matrices that keep $A$ and $B$ untouched, i.e. the computation has a **smaller memory footprint**.
Matrices are pretty sparse, but structured.
- Matrices are pretty sparse, but structured.
- GBLA supports two matrix formats, both use binary format.
GBLA Matrix formats

- 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>
<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>b</td>
<td></td>
<td>version number</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>m</td>
<td></td>
<td># rows</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>n</td>
<td></td>
<td># columns</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>p</td>
<td></td>
<td>prime / field characteristic</td>
</tr>
<tr>
<td><code>uint64_t</code></td>
<td>nnz</td>
<td>data</td>
<td># nonzero entries</td>
</tr>
<tr>
<td><code>uint16_t</code></td>
<td></td>
<td>data</td>
<td>entry in matrix</td>
</tr>
<tr>
<td><code>uint32_t</code></td>
<td>nnz</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>
## GBLA Matrix formats

Table: 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 $x_i f_j$</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: Single column entry masked via $(1 &lt;&lt; 31)$; $s$ consecutive entries starting at column $c$ are stored as &quot;c s&quot;</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

<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 ofpdata</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 $x_i f_j$</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></td>
<td>length of rows</td>
</tr>
</tbody>
</table>

| uint32_t   | m      | pmap | maps rows to pdata                                                          |
| uint64_t   | 1      | k    | size of compressed colid                                                    |
| uint64_t   | k      | colid| compression of columns:
|            |        |     | Single column entry masked via ($1 << 31$);
|            |        |     | $s$ consecutive entries starting at column $c$ are stored as “$c s”       |
| uint32_t   | 1      | pnb  | # polynomials                                                               |
| uint64_t   | 1      | pnnz | # nonzero coefficients in polynomials                                       |
| uint32_t   | pnb    | prow | length of polynomial / row representation                                   |
| xinty_t    | pnnz   | pdata| coefficients of polynomials                                                 |

Approximately 1/3rd of memory usage compared to the old 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>

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

- New format = 3rd of memory usage.
- New format = 4th of memory usage when compressed with gzip.
- Compression 5 times faster.
### Table: 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.
## GBLA Matrix formats – Comparison

### Table: 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.
**GBLA Matrix formats – Comparison**

<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.
GB – A Gröbner Basis Library
- **Open source** library written in **plain C**.
- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Open source** library written in plain C.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **DRL** and **LEX** monomial ordering.
Library Overview

- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **DRL** and **LEX** monomial ordering.
- Several **strategies for simplification**.
Library Overview

- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **DRL** and **LEX** monomial ordering.
- Several strategies for simplification.
- Available as alpha version in **Singular after 4-0-3 release**.
▸ **Open source** library written in **plain C**.

▸ Uses **GBLA** for linear algebra part.

▸ **Parallel matrix construction** based on OpenMP.

▸ Works over finite fields (restrictions due to GBLA).

▸ **DRL** and **LEX** monomial ordering.

▸ Several **strategies for simplification**.

▸ Available as alpha version in **Singular after 4-0-3 release**.

https://www.github.com/ederc/gb
Katsura-\( n \) w.r.t. DRL (single-threaded)

![Graph showing Time (log 2) in seconds for Katsura-\( n \) w.r.t. DRL (single-threaded).](image)

- Singular 4-0-3
- FGb v1.68
- GB v0.1 w/ GBLA v0.2

Time (log 2) in seconds:

- 0
- 2
- 4
- 6
- 8
- 10
- 12
- 14
- 16

\( n \) values:

- 12
- 13
- 14
- 15

\( \log_2 \) scale on the y-axis.
Cyclic-n w.r.t. DRL (single-threaded)

Time (log 2) in seconds

- Singular 4-0-3
- FGb v1.68
- GB v0.1 w/ GBLA v0.2

Magma v2.21 / Maple 2016

45 / 50
Cyclic-$n$ w.r.t. DRL (single-threaded)

Time (log 2) in seconds

7 8 9 10

2 4 6 8 10 12 14 16

Singular 4-0-3
FGB v1.68
GB v0.1 w/ GBLA v0.2
Magma v2.21 / Maple 2016
Cyclic-10 w.r.t. DRL (n-threads)

Time (log 2) in seconds

GB v0.1 w/ GBLA v0.2
Maple 2016
Next steps

▶ v0.3 of GBLA.
Next steps

- v0.3 of GBLA.
- Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
Next steps

▶ **v0.3** of GBLA.

▶ Optimizing GBLA for **floating point** and **32-bit unsigned int arithmetic**.

▶ **Multi-modular computation** in GB.
Next steps

- v0.3 of GBLA.
- Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
- Multi-modular computation in GB.
- Parallel hashing in GB.
Next steps

- **v0.3** of GBLA.
- Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
- Multi-modular computation in GB.
- Parallel hashing in GB.
- Better handling of memory bounds in GB/GBLA.
Next steps

► **v0.3** of GBLA.

► Optimizing GBLA for **floating point** and **32-bit unsigned int arithmetic**.

► **Multi-modular computation** in GB.

► **Parallel hashing** in GB.

► **Better handling of memory bounds** in GB/GBLA.

► **FGLM** in GB for zero-dimensional system solving.
Next steps

▶ **v0.3** of GBLA.
▶ Optimizing GBLA for **floating point** and **32-bit unsigned int arithmetic**.
▶ **Multi-modular computation** in GB.
▶ **Parallel hashing** in GB.
▶ **Better handling of memory bounds** in GB/GBLA.
▶ **FGLM** in GB for zero-dimensional system solving.
▶ First steps exploiting **heterogeneous CPU/GPU platforms** for GBLA.
Next steps

- **v0.3** of GBLA.
- Optimizing GBLA for floating point and 32-bit unsigned int arithmetic.
- Multi-modular computation in GB.
- Parallel hashing in GB.
- Better handling of memory bounds in GB/GBLA.
- FGLM in GB for zero-dimensional system solving.
- First steps exploiting heterogeneous CPU/GPU platforms for GBLA.
- Deeper investigation on parallelization on networks.


Thank you!
Questions? Comments?