Parallel Gröbner Basis Algorithms over Finite Fields

Christian Eder and Jean-Charles Faugère

University of Kaiserslautern

October 14, 2016

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

$$\operatorname{spol}(f, g) := \frac{\lambda}{\operatorname{lt}(f)} f - \frac{\lambda}{\operatorname{lt}(g)} g.$$

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

$$\operatorname{spol}(f, g) := \frac{\lambda}{\operatorname{lt}(f)} f - \frac{\lambda}{\operatorname{lt}(g)} g.$$

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 $f, g \in G$: $\operatorname{spol}(f, g) \xrightarrow{G} 0$.

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 $P \leftarrow \{\text{spol}\left(f_i, f_j\right) \mid f_i, f_j \in G, i > j\}$

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 $P \leftarrow \{\mathrm{spol}\left(f_i, f_j\right) \mid f_i, f_j \in G, i > j\}$
4. Choose $p \in P$, $P \leftarrow P \setminus \{p\}$

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 $P \leftarrow \{\mathrm{spol}\left(f_i, f_j\right) \mid f_i, f_j \in G, i > j\}$
4. Choose $p \in P$, $P \leftarrow P \setminus \{p\}$
   (a) If $p \xrightarrow{G} 0$ ▶▶ **no new information**
   Go on with the next element in P.
   (b) If $p \xrightarrow{G} q \neq 0$ ▶▶ **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

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 $P \neq \emptyset$:

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) $P_d \leftarrow$ **Select** (P), $P \leftarrow P \setminus P_d$
   (c) $L_d \leftarrow \{af, bg \mid (af, bg) \in P_d\}$
   (d) $L_d \leftarrow$ **Symbolic Preprocessing**(L_d, G)
   (e) $F_d \leftarrow$ **Reduction**(L_d, G)
   (f) for $h \in F_d$:
      - If $\mathrm{lt}(h) \notin L(G)$ (all other h are "useless"):
        - $P \leftarrow P \cup \{$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 **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.

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 $\lambda\, \mathrm{lt}\,(g) = m$
   $\triangleright\ F \leftarrow F \cup \{\lambda g\}$
4. Return F

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 $\lambda\, \mathrm{lt}\,(g) = m$
   $\triangleright\, F \leftarrow F \cup \{\lambda g\}$
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:

$$\begin{array}{rcl} \text{columns} & \hat{=} & \text{monomials (sorted by monomial order } <) \\ \text{rows} & \hat{=} & \text{coefficients of polynomials in } L \end{array}$$

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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 **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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:

$$\begin{aligned} T(L_1) &= \{ab, b^2, bc, ad, bd, cd \quad \} \\ L_1 &= \{f_3, bf_4 \quad \} \end{aligned}$$

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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:

$$\begin{aligned} T(L_1) &= \{ab, b^2, bc, ad, bd, cd \quad \} \\ L_1 &= \{f_3, bf_4 \quad \} \end{aligned}$$

$b^2 \notin L(G)$,

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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:

$$\begin{aligned} T(L_1) &= \{ab, b^2, bc, ad, bd, cd \quad \} \\ L_1 &= \{f_3, bf_4 \quad \} \end{aligned}$$

$b^2 \notin L(G)$, $bc \notin L(G)$,

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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:

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

$b^2 \notin L(G)$, $bc \notin L(G)$, $d\,\mathrm{lt}\,(f_4) = ad$,

$\mathcal{R} = \mathbb{Q}[a, b, c, d]$, $<$ denotes DRL and we use the normal selection strategy for **Select** (P).
$I = \langle f_1, \ldots, f_4 \rangle$, where

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

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:

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

$b^2 \notin L(G)$, $bc \notin L(G)$, $d\,\mathrm{lt}\,(f_4) = ad$, all others also $\notin L(G)$,

Now reduction:
Convert polynomial data L_1 to Macaulay Matrix M_1

$$\begin{array}{c} \\ df_4 \\ f_3 \\ bf_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} ab & b^2 & bc & ad & bd & cd & d^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Now reduction:
Convert polynomial data L_1 to Macaulay Matrix M_1

$$\begin{array}{c} \\ df_4 \\ f_3 \\ bf_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} ab & b^2 & bc & ad & bd & cd & d^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Gaussian Elimination of M_1:

$$\begin{array}{c} \\ df_4 \\ f_3 \\ bf_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} ab & b^2 & bc & ad & bd & cd & d^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 2 & 0 & 1 \end{pmatrix} \end{array}$$

Convert matrix data back to polynomial structure F_1:

$$\begin{array}{c} \\ df_4 \\ f_3 \\ bf_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} ab & b^2 & bc & ad & bd & cd & d^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 2 & 0 & 1 \end{pmatrix} \end{array}$$

$$F_1 = \left\{ \underbrace{ad + bd + cd + d^2}_{f_5}, \underbrace{ab + bc - bd - d^2}_{f_6}, \underbrace{b^2 + 2bd + d^2}_{f_7} \right\}$$

Convert matrix data back to polynomial structure F_1:

$$\begin{array}{c} \\ df_4 \\ f_3 \\ bf_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} ab & b^2 & bc & ad & bd & cd & d^2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 2 & 0 & 1 \end{pmatrix} \end{array}$$

$$F_1 = \left\{ \underbrace{ad + bd + cd + d^2}_{f_5}, \underbrace{ab + bc - bd - d^2}_{f_6}, \underbrace{b^2 + 2bd + d^2}_{f_7} \right\}$$

$\mathrm{lt}\,(f_5)\,, \mathrm{lt}\,(f_6) \in L(G)$, so

$\mathbf{G} \leftarrow \mathbf{G} \cup \{\mathbf{f_7}\}$.

Next round:

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

Next round:

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

We can simplify the computations:

$$\mathrm{lt}\,(bcf_4) = abc = \mathrm{lt}\,(cf_6).$$

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

$$\Longrightarrow\ L_2 = \{f_2, cf_6\}$$

Next round:

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

We can simplify the computations:

$$\mathrm{lt}\,(bcf_4) = abc = \mathrm{lt}\,(cf_6).$$

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

$$\Longrightarrow\ L_2 = \{f_2, cf_6\}$$

Symbolic preprocessing:

$$\begin{array}{rcl} T(L_2) & = & \{abc, bc^2, abd, acd, bcd, cd^2\} \\ L_2 & = & \{f_2, cf_6, \quad \} \end{array}$$

Next round:

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

We can simplify the computations:

$$\mathrm{lt}\,(bcf_4) = abc = \mathrm{lt}\,(cf_6)\,.$$

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

$$\Longrightarrow\ L_2 = \{f_2, cf_6\}$$

Symbolic preprocessing:

$$\begin{array}{rcl} T(L_2) & = & \{abc, bc^2, abd, acd, bcd, cd^2\} \\ L_2 & = & \{f_2, cf_6, \quad \} \end{array}$$

$bc^2 \notin L(G)$,

Next round:

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

We can simplify the computations:

$$\mathrm{lt}\,(bcf_4) = abc = \mathrm{lt}\,(cf_6).$$

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

$$\Longrightarrow\ L_2 = \{f_2, cf_6\}$$

Symbolic preprocessing:

$$\begin{array}{rcl} T(L_2) & = & \{abc, bc^2, abd, acd, bcd, cd^2\} \\ L_2 & = & \{f_2, cf_6, \quad \} \end{array}$$

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

Next round:

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

We can simplify the computations:

$$\mathrm{lt}\,(bcf_4) = abc = \mathrm{lt}\,(cf_6).$$

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

$$\Longrightarrow\ L_2 = \{f_2, cf_6\}$$

Symbolic preprocessing:

$$\begin{array}{rcl} T(L_2) & = & \{abc, bc^2, abd, acd, bcd, cd^2\} \\ L_2 & = & \{f_2, cf_6, \quad\} \end{array}$$

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

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.
$\Rightarrow$ Reuse rows that are reduced but not "in" G.

Idea

Replace $u \cdot f$ by $(wv) \cdot g$ where $vg \in F_i$ for a previous reduction step.
$\Rightarrow$ Reuse rows that are reduced but not "in" G.

Note

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

Idea

Replace $u \cdot f$ by $(wv) \cdot g$ where $vg \in F_i$ for a previous reduction step.
$\Rightarrow$ Reuse rows that are reduced but not "in" G.

Note

- Tries to reuse all rows from old matrices.
  $\Rightarrow$ We need to keep them in memory.
- We also simplify generators of S-pairs, as we have done in our example: $(f_2, bcf_4) \Longrightarrow (f_2, cf_6)$.
- 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:

$$\begin{aligned} T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2 \qquad\qquad \} \\ L_2 &= \{f_2, cf_6 \qquad\qquad \} \end{aligned}$$

$bc^2 \notin L(G)$,

Symbolic preprocessing - now with simplify:

$$\begin{aligned} T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2 \qquad\qquad \} \\ L_2 &= \{f_2, cf_6 \qquad\qquad \} \end{aligned}$$

$bc^2 \notin L(G)$, $abd = \mathrm{lt}\,(bf_5)$,

Symbolic preprocessing - now with simplify:

$$\begin{aligned} T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d \quad \} \\ L_2 &= \{f_2, cf_6, bf_5 \quad \} \end{aligned}$$

$bc^2 \notin L(G)$, $abd = \mathrm{lt}\,(bf_5)$,

Symbolic preprocessing - now with simplify:

$$\begin{aligned} T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d, \dots\} \\ L_2 &= \{f_2, cf_6, bf_5, cf_5, df_7\} \end{aligned}$$

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

Symbolic preprocessing - now with simplify:

$$\begin{aligned} T(L_2) &= \{abc, bc^2, abd, acd, bcd, cd^2, b^2d, c^2d, \dots\} \\ L_2 &= \{f_2, cf_6, bf_5, cf_5, df_7\} \end{aligned}$$

$bc^2 \notin L(G)$, $abd = \mathrm{lt}\,(bf_5)$, 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.

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Specialize Linear Algebra for reduction steps in GB computations.

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Try to exploit underlying GB structure.

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rl}
\text{S-pair} & \left\{\begin{array}{ccccccc} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{array}\right. \\
\text{S-pair} & \left\{\begin{array}{ccccccc} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{array}\right. \\
\text{reducer} & \leftarrow \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{array}
\end{array}
$$

Try to exploit underlying GB structure.

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rl}
\text{S-pair} & \left\{ \begin{array}{ccccccc} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{array} \right. \\
\text{S-pair} & \left\{ \begin{array}{ccccccc} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{array} \right. \\
\text{reducer} & \leftarrow \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{array}
\end{array}
$$

Try to exploit underlying GB structure.

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rl}
\text{S-pair} & \left\{\begin{array}{ccccccc} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{array}\right. \\
\text{S-pair} & \left\{\begin{array}{ccccccc} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{array}\right. \\
\text{reducer} & \leftarrow \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{array}
\end{array}
$$

Try to exploit underlying GB structure.

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rl}
\text{S-pair} & \left\{ \begin{array}{ccccccc} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{array} \right. \\
\text{S-pair} & \left\{ \begin{array}{ccccccc} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{array} \right. \\
\text{reducer} & \leftarrow \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{array}
\end{array}
$$

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

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

1st step: Sort pivot and non-pivot columns

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Pivot column

1st step: Sort pivot and non-pivot columns

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Pivot column

1st step: Sort pivot and non-pivot columns

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Pivot column Non-Pivot column

1st step: Sort pivot and non-pivot columns

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix}$$

Pivot column Non-Pivot column

1st step: Sort pivot and non-pivot columns

$$\begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \\ 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{matrix} \longrightarrow \left[\begin{matrix} 1 & 3 & 7 \\ 1 & 0 & 0 \\ 0 & 1 & 8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \;\middle|\; \begin{matrix} 0 & 0 & 1 & 0 \\ 4 & 1 & 0 & 5 \\ 6 & 0 & 0 & 9 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 1 \end{matrix}\right]$$

Pivot column Non-Pivot column

2nd step: Sort pivot and non-pivot rows

$$\begin{array}{ccc|cccc}
1 & 3 & 7 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 8 & 6 & 0 & 0 & 9 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1
\end{array}$$

2nd step: Sort pivot and non-pivot rows

$$\begin{array}{ccc|cccc} 1 & 3 & 7 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 4 & 1 & 0 & 5 \\ 0 & 1 & 8 & 6 & 0 & 0 & 9 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 1 & 0 & 0 & 3 & 1 \end{array}$$

Pivot row

2nd step: Sort pivot and non-pivot rows

$$\begin{array}{ccc|cccc} 1 & 3 & 7 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 4 & 1 & 0 & 5 \\ 0 & 1 & 8 & 6 & 0 & 0 & 9 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 1 & 0 & 0 & 3 & 1 \end{array}$$

Pivot row

Non-Pivot row

2nd step: Sort pivot and non-pivot rows

$$\begin{array}{ccc|cccc} 1 & 3 & 7 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 4 & 1 & 0 & 5 \\ 0 & 1 & 8 & 6 & 0 & 0 & 9 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 1 & 0 & 0 & 3 & 1 \end{array}$$

Pivot row

Non-Pivot row

2nd step: Sort pivot and non-pivot rows

$$
\begin{array}{ccc|cccc}
1 & 3 & 7 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 8 & 6 & 0 & 0 & 9 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1
\end{array}
\longrightarrow
\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
1 & 3 & 7 & 0 & 0 & 1 & 0 \\
0 & 1 & 8 & 6 & 0 & 0 & 9
\end{array}
$$

Pivot row

Non-Pivot row

3rd step: Reduce lower left part to zero

$$\begin{array}{ccc|cccc} 1 & 0 & 0 & 4 & 1 & 0 & 5 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 1 & 0 & 0 & 3 & 1 \\ \hline 1 & 3 & 7 & 0 & 0 & 1 & 0 \\ 0 & 1 & 8 & 6 & 0 & 0 & 9 \end{array}$$

3rd step: Reduce lower left part to zero

$$
\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
1 & 3 & 7 & 0 & 0 & 1 & 0 \\
0 & 1 & 8 & 6 & 0 & 0 & 9
\end{array}
\longrightarrow
\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 10 & 3 & 10 \\
0 & 0 & 0 & 6 & 0 & 2 & 1
\end{array}
$$

4th step: Reduce lower right part

$$\begin{array}{ccc|cccc} 1 & 0 & 0 & 4 & 1 & 0 & 5 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 1 & 0 & 0 & 3 & 1 \\ \hline 0 & 0 & 0 & 7 & 10 & 3 & 10 \\ 0 & 0 & 0 & 6 & 0 & 2 & 1 \end{array}$$

4th step: Reduce lower right part

$$
\left[\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 10 & 3 & 10 \\
0 & 0 & 0 & 6 & 0 & 2 & 1
\end{array}\right]
\longrightarrow
\left[\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 0 & 6 & 3 \\
0 & 0 & 0 & 0 & 4 & 1 & 5
\end{array}\right]
$$

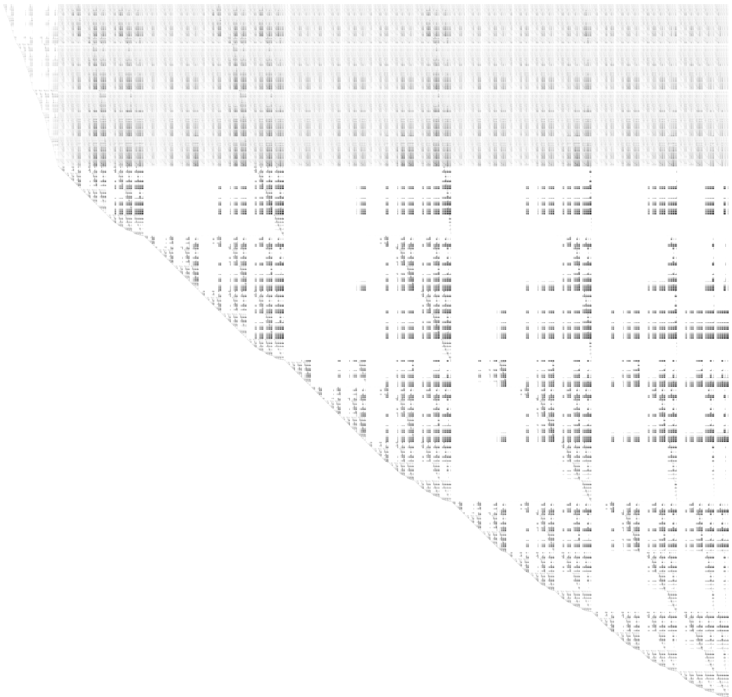
4th step: Reduce lower right part

$$
\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 10 & 3 & 10 \\
0 & 0 & 0 & 6 & 0 & 2 & 1
\end{array}
\longrightarrow
\begin{array}{ccc|cccc}
1 & 0 & 0 & 4 & 1 & 0 & 5 \\
0 & 1 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 1 & 0 & 0 & 3 & 1 \\
\hline
0 & 0 & 0 & 7 & 0 & 6 & 3 \\
0 & 0 & 0 & 0 & 4 & 1 & 5
\end{array}
$$

5th step: Remap columns and get new polynomials for GB out of lower right part.

“Real world” matrices?

What our matrices look like



Characteristics of this matrix

- **F4** computation of homogeneous Katsura-12, degree 6 matrix
- Size 55MB

Characteristics of this matrix

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

Characteristics of this matrix

- **F4** computation of homogeneous Katsura-12, degree 6 matrix
- Size 55MB
- $24,006,869$ nonzero elements (density: 5%)
- Dimensions:

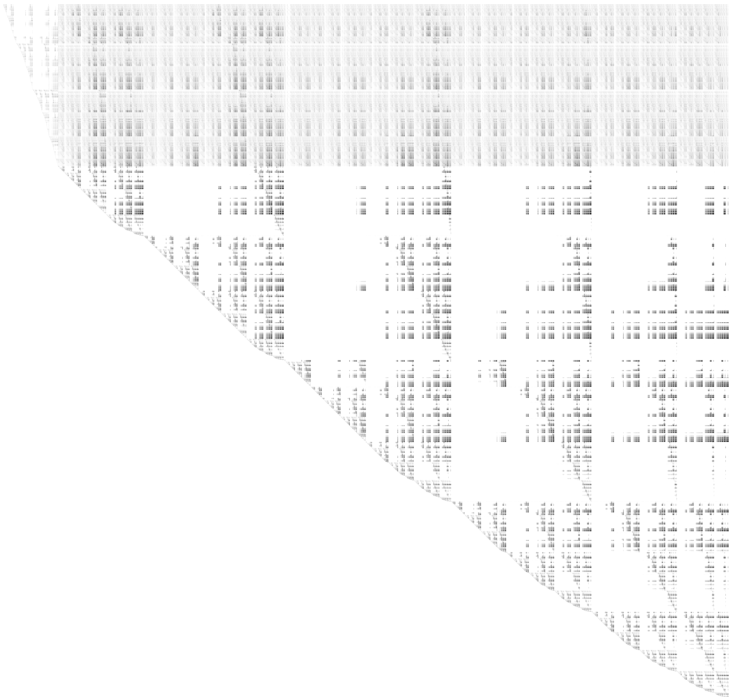
  full matrix: $21,182 \times 22,207$

Characteristics of this matrix

- **F4** computation of homogeneous Katsura-12, degree 6 matrix
- Size 55MB
- $24,006,869$ nonzero elements (density: 5%)
- Dimensions:

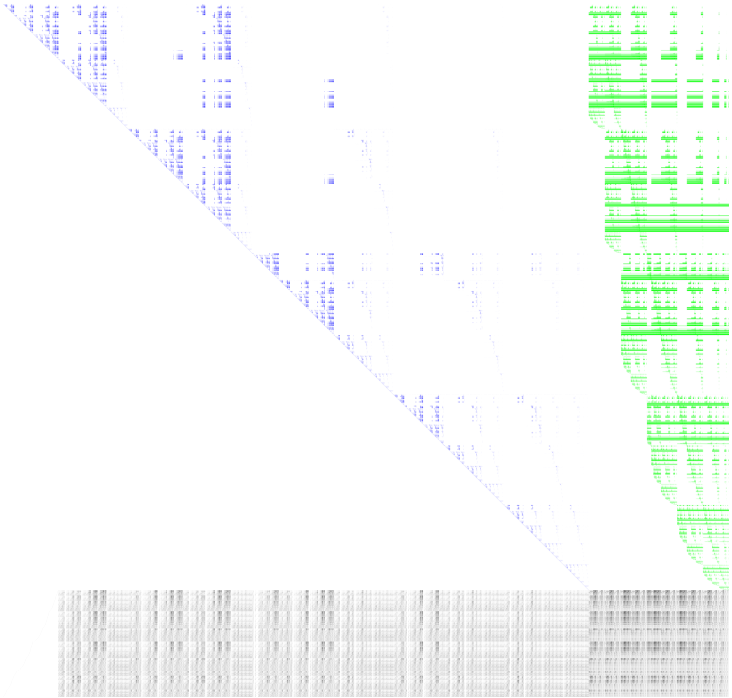
full matrix:	$21,182$	$\times$	$22,207$	
upper-left:	$17,915$	$\times$	$17,915$	known pivots
lower-left:	$3,267$	$\times$	$17,915$	
upper-right:	$17,915$	$\times$	$4,292$	
lower-right:	$3,267$	$\times$	$4,292$	new information

What our matrices look like

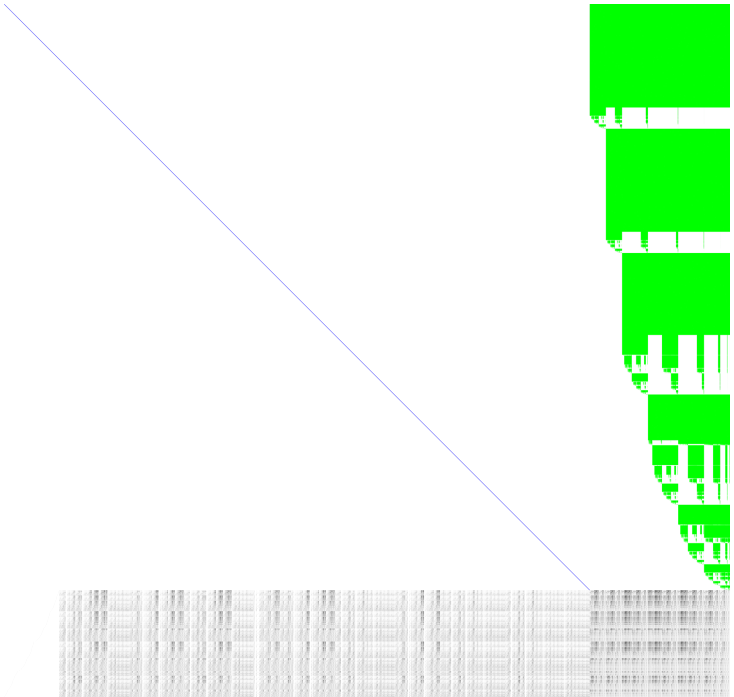


What our matrices look like

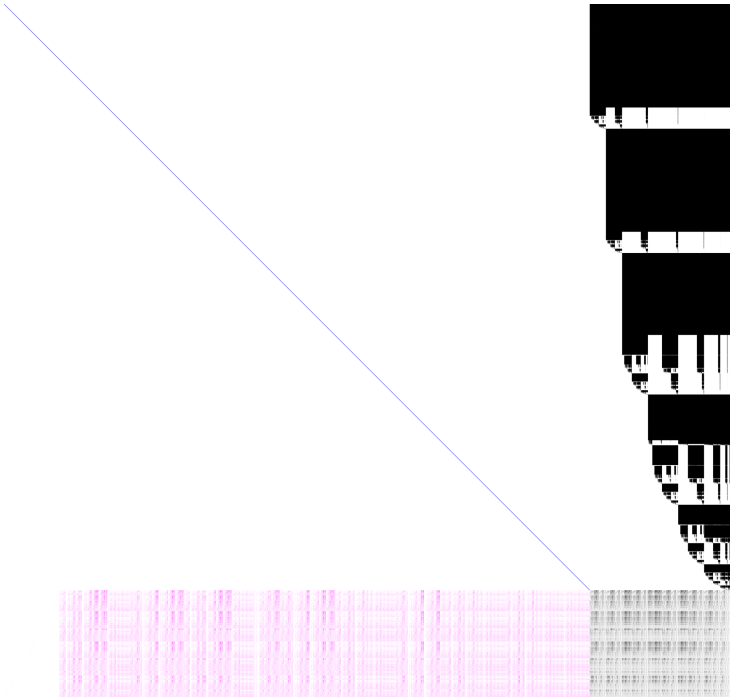
Hybrid Matrix Multiplication $A^{-1}B$



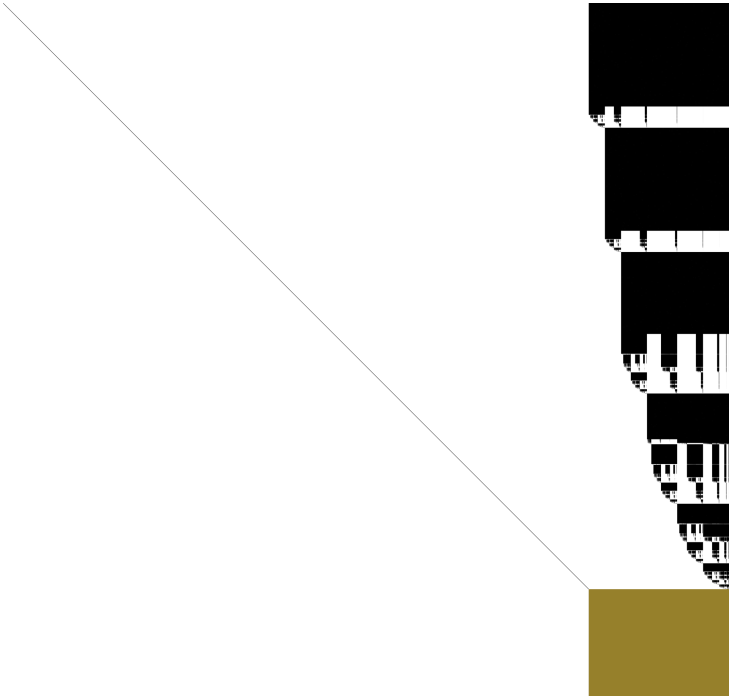
Hybrid Matrix Multiplication $A^{-1}B$



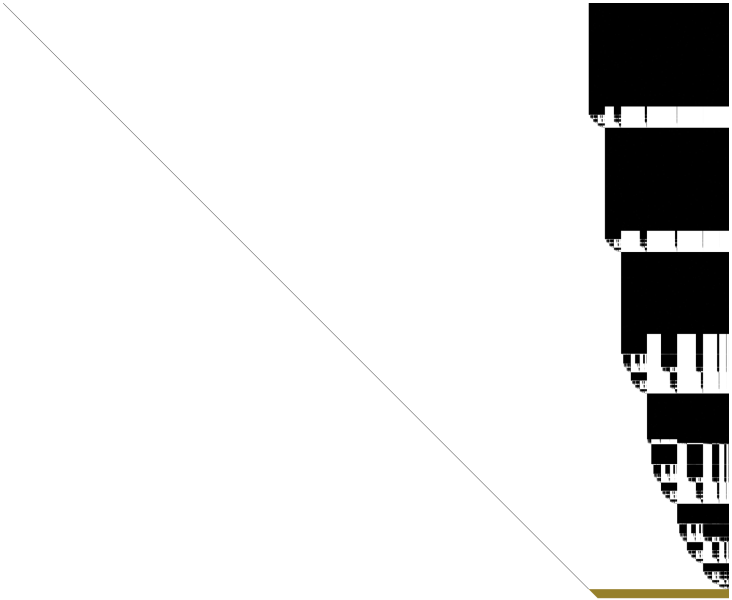
Reduce C to zero



Gaussian Elimination on D



New information



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.

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).

- **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.

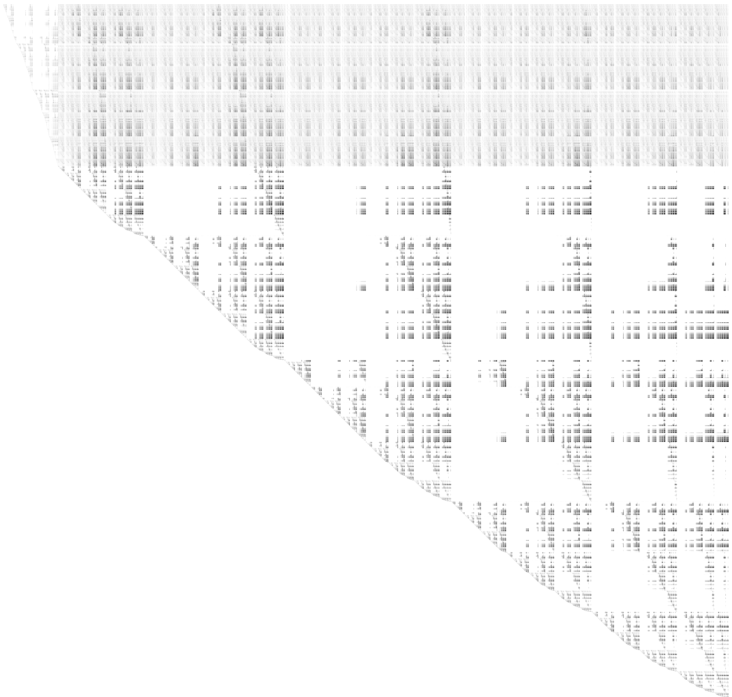
- **Open source** library written in **plain C**.
- Specialized linear algebra for GB computations.
- **Parallel implementation** (OpenMP), scaling “nicely” up to 32 cores.
- Works over finite fields for 16-bit primes (at the moment).
- **Several strategies** for splicing and reduction.
- Includes **converter**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

- **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
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 $m_{i,j} \neq 0$ 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 $m_{i+1,j} \neq 0$.

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**.

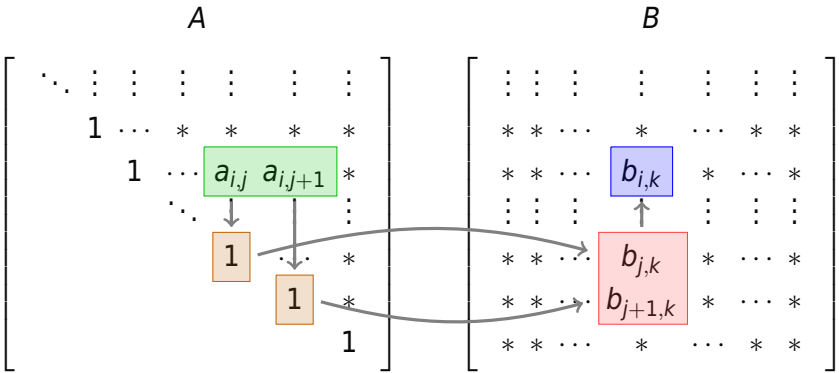
Horizontal Pattern If $m_{i,j} \neq 0$ then often $m_{i,j+1} \neq 0$.
Vertical Pattern If $m_{i,j} \neq 0$ then often $m_{i+1,j} \neq 0$.

- 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:

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

Consider the following two rows:

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

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

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

Consider the following two rows:

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

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

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

A multiline vector representation of `r1` and `r2` is given by

$$\begin{aligned} \texttt{ml.val} &= [\ 2\ \ 1\ \ 3\ \ 7\ \ 1\ \ 0\ \ 4\ \ 3\ \ 0\ \ 1\ \ 5\ \ 2\ ], \\ \texttt{ml.pos} &= [\ 0\ \ 1\ \ 3\ \ 4\ \ 5\ \ 6\ ]. \end{aligned}$$

Consider the following two rows:

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

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

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

A multiline vector representation of `r1` and `r2` is given by

$$\begin{aligned} \texttt{ml.val} &= [\ 2\ \ 1\ \ 3\ \ 7\ \ 1\ \ 0\ \ 4\ \ 3\ \ 0\ \ 1\ \ 5\ \ 2\ ], \\ \texttt{ml.pos} &= [\ 0\ \ 1\ \ 3\ \ 4\ \ 5\ \ 6\ ]. \end{aligned}$$

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

- 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 $A^{-1}B$.

- 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?

- 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

- 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.

- 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.

- 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.

- 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: Old matrix format (legacy version)

Size	Length	Data	Description
`uint32_t`	1	`b`	version number
`uint32_t`	1	`m`	# rows
`uint32_t`	1	`n`	# columns
`uint32_t`	1	`p`	prime / field characteristic
`uint64_t`	1	`nnz`	# nonzero entries
`uint16_t`	`nnz`	`data`	entry in matrix
`uint32_t`	`nnz`	`cols`	column index of entry
`uint32_t`	`m`	`rows`	length of rows

Table: New matrix format (compressing `data` and `cols`)

Size	Length	Data	Description
`uint32_t`	1	`b`	version number + information for data type of `pdata`
`uint32_t`	1	`m`	# rows
`uint32_t`	1	`n`	# columns
`uint32_t`	1	`p`	prime / field characteristic
`uint64_t`	1	`nnz`	# nonzero entries
`uint16_t`	`nnz`	`data`	several rows are of type $x_i f_j$
`uint32_t`	`nnz`	`cols`	can be compressed for consecutive elements
`uint32_t`	`m`	`rows`	length of rows
`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

Table: New matrix format (compressing `data` and `cols`)

Size	Length	Data	Description
`uint32_t`	1	`b`	version number + information for data type of `pdata`
`uint32_t`	1	`m`	# rows
`uint32_t`	1	`n`	# columns
`uint32_t`	1	`p`	prime / field characteristic
`uint64_t`	1	`nnz`	# nonzero entries
`uint16_t`	`nnz`	`data`	several rows are of type $x_i f_j$
`uint32_t`	`nnz`	`cols`	can be compressed for consecutive elements
`uint32_t`	`m`	`rows`	length of rows
`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: Storage and time efficiency of the new format

Matrix	Size old	Size new	`gzip`ped old	`gzip`ped new	Time old	Time new
`F4-kat14-mat9`	2.3GB	0.74GB	1.2GB	0.29GB	230s	66s
`F5-kat17-mat10`	43GB	12GB	24GB	5.3GB	4419s	883s

Table: Storage and time efficiency of the new format

Matrix	Size old	Size new	gzipped old	gzipped new	Time old	Time new
F4-kat14-mat9	2.3GB	0.74GB	1.2GB	0.29GB	230s	66s
F5-kat17-mat10	43GB	12GB	24GB	5.3GB	4419s	883s

New format vs. **Old format**

Table: Storage and time efficiency of the new format

Matrix	Size old	Size new	gzipped old	gzipped new	Time old	Time new
F4-kat14-mat9	2.3GB	0.74GB	1.2GB	0.29GB	230s	66s
F5-kat17-mat10	43GB	12GB	24GB	5.3GB	4419s	883s

New format vs. **Old format**

- 1/3rd of memory usage.

Table: Storage and time efficiency of the new format

Matrix	Size old	Size new	gzipped old	gzipped new	Time old	Time new
F4-kat14-mat9	2.3GB	0.74GB	1.2GB	0.29GB	230s	66s
F5-kat17-mat10	43GB	12GB	24GB	5.3GB	4419s	883s

New format vs. **Old format**

- 1/3rd of memory usage.
- 1/4th of memory usage when compressed with gzip.

Table: Storage and time efficiency of the new format

Matrix	Size old	Size new	gzipped old	gzipped new	Time old	Time new
F4-kat14-mat9	2.3GB	0.74GB	1.2GB	0.29GB	230s	66s
F5-kat17-mat10	43GB	12GB	24GB	5.3GB	4419s	883s

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.

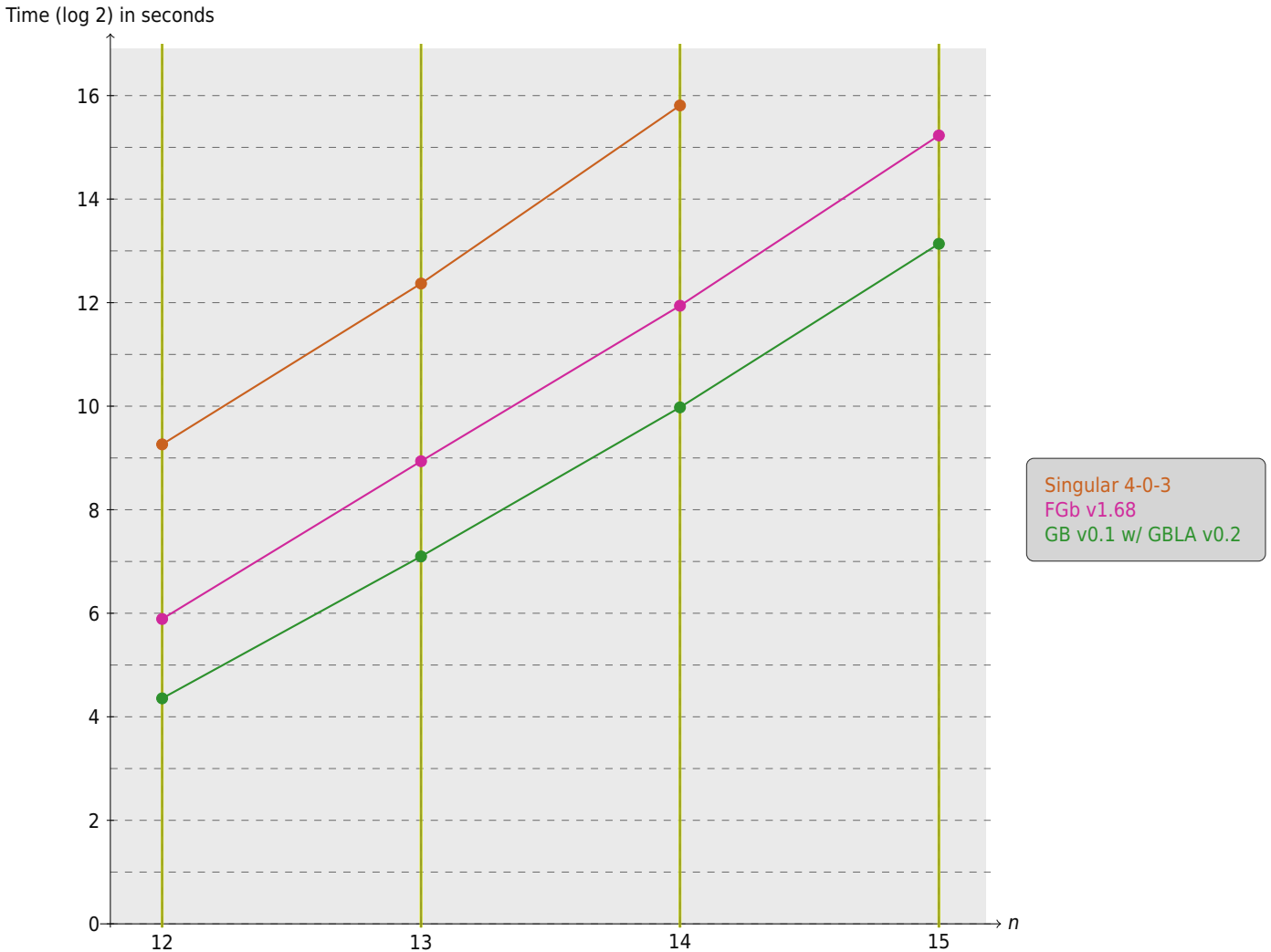
- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **DRL** and **LEX** monomial ordering.
- Several **strategies for simplification**.

- **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**.

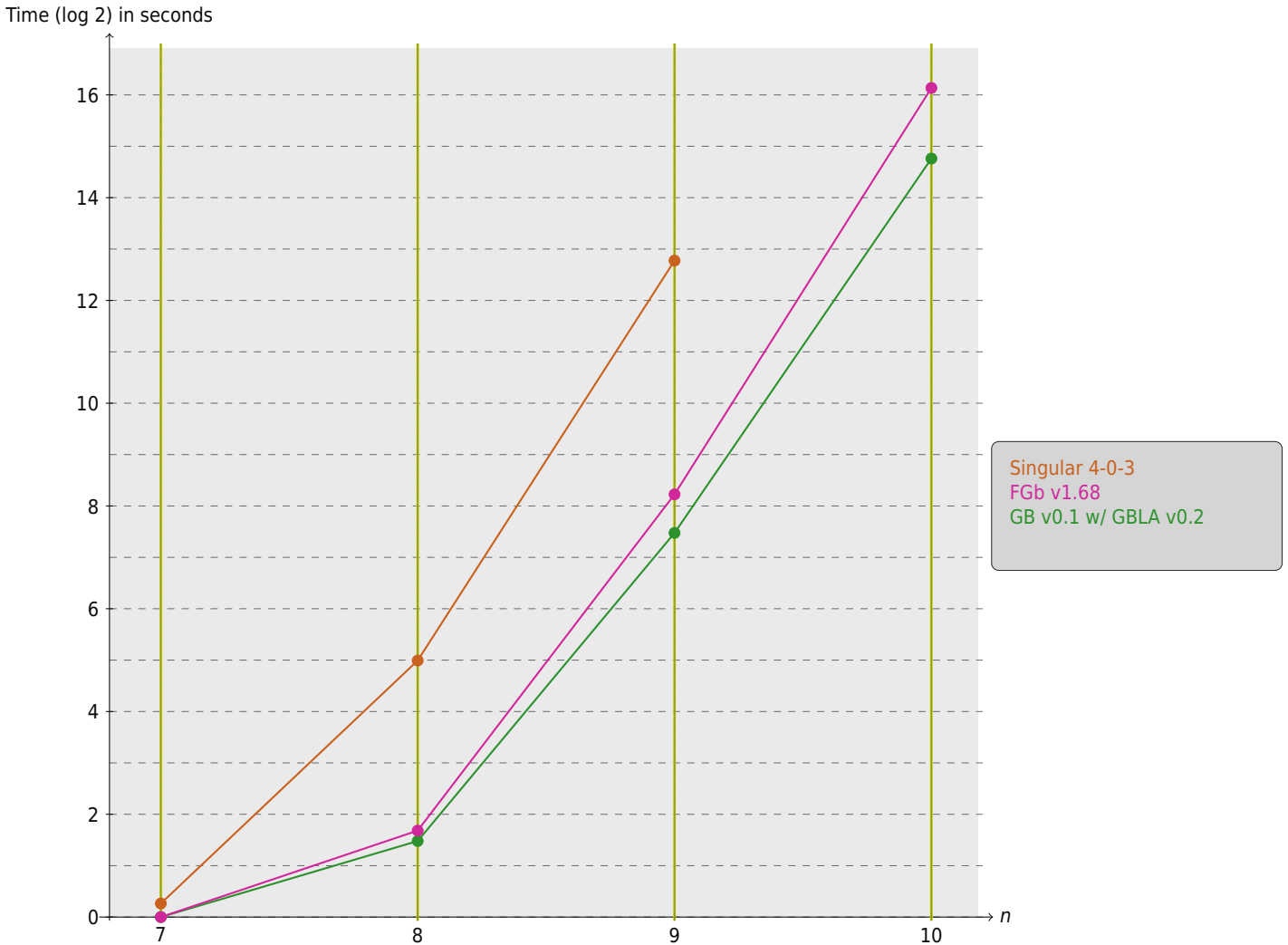
- **Open source** library written in **plain C**.
- Uses **GBLA** for linear algebra part.
- **Parallel matrix construction** based on OpenMP.
- Works over finite fields (restrictions due to GBLA).
- **DRL** and **LEX** monomial ordering.
- Several **strategies for simplification**.
- Available as alpha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

`https://www.github.com/ederc/gb`

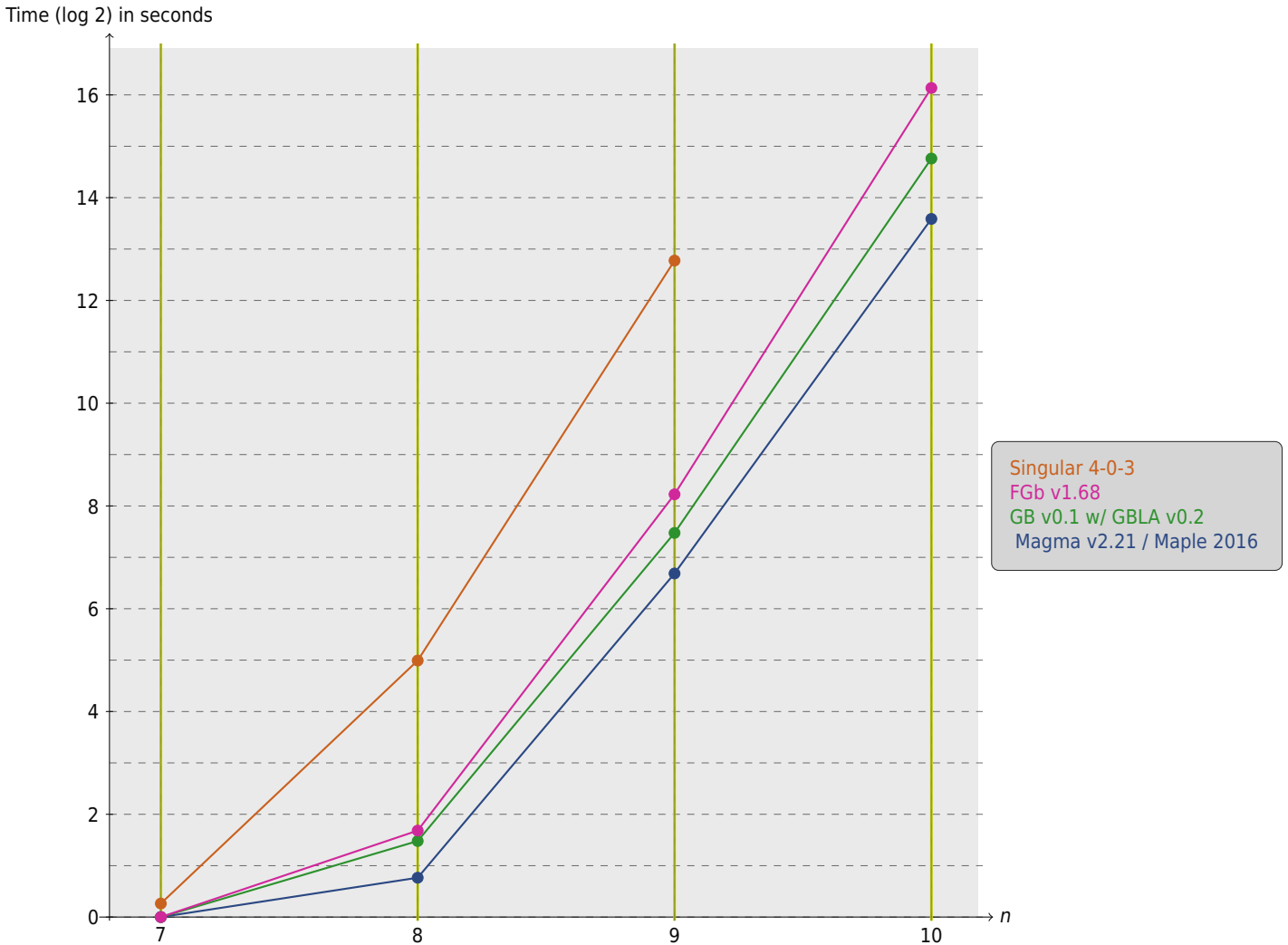
Katsura–n w.r.t. DRL (single-threaded)



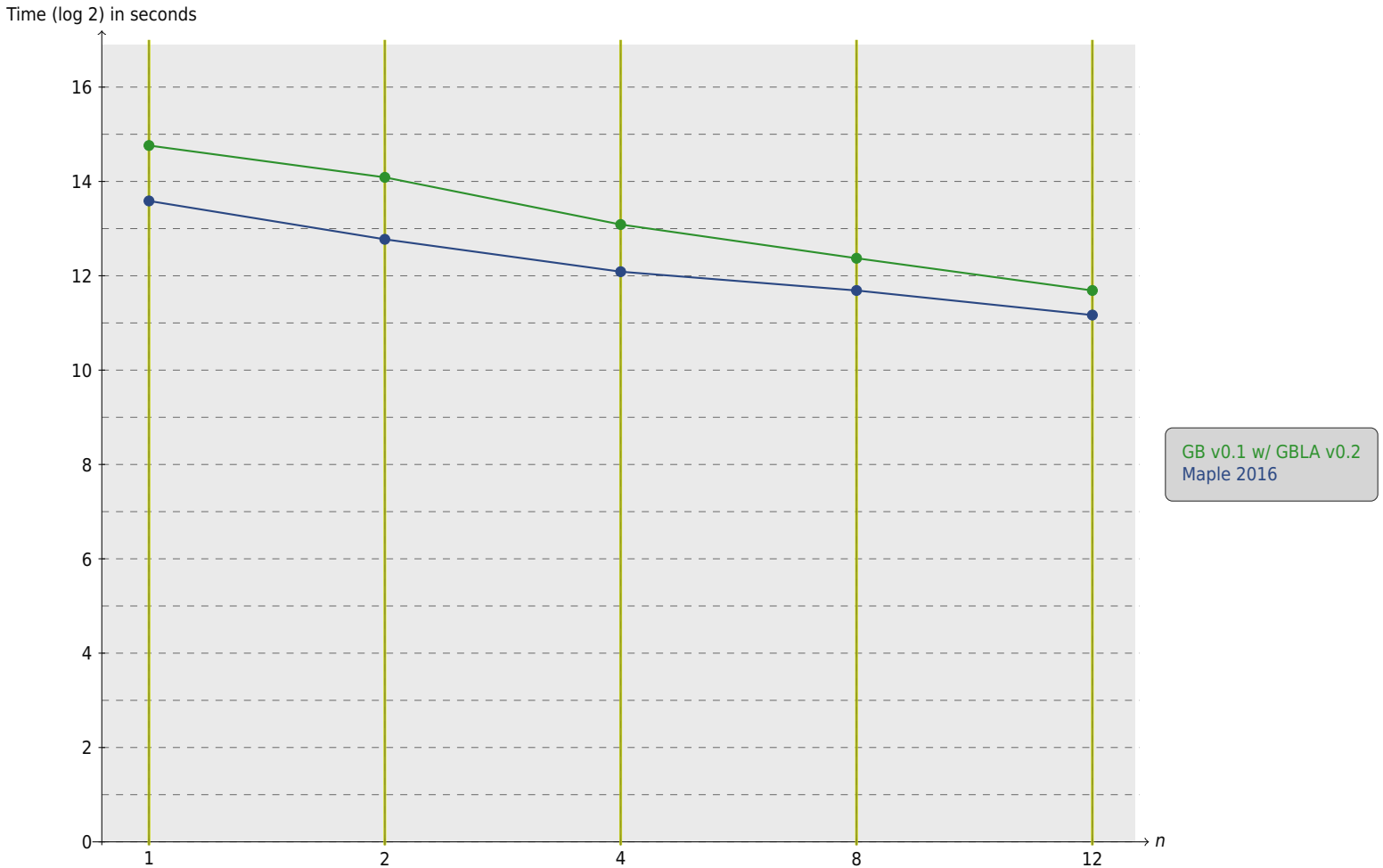
Cyclic–*n* w.r.t. DRL (single-threaded)



Cyclic–n w.r.t. DRL (single-threaded)



Cyclic–10 w.r.t. DRL (n-threads)



- **v0.3** of GBLA.

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

- **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.

- **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.

- **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.

- **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.

- **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**.

[1] Boyer, B. and Eder, C. and Faugère, J.-C. and Lachartre, S. and Martani, F. *GBLA - Gröbner Basis Linear Algebra Package*, 2016. Proceedings of the 2016 International Symposium on Symbolic and Algebraic Computation

[2] Buchberger, B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, 1965. PhD thesis, Universtiy of Innsbruck, Austria

[3] Buchberger, B. *A criterion for detecting unnecessary reductions in the construction of Gröbner bases*, 1979. EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation

[4] Buchberger, B. *Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory*, 1985. Multidimensional Systems Theory, D. Reidel Publication Company

[5] Eder, C. and Faugère, J.-C. *A survey on signature-based Gröbner basis algorithms*, 2016. Journal of Symbolic Computation

[6] Faugère, J.-C. *A new efficient algorithm for computing Gröbner bases (F4)*, 1999. Journal of Pure and Applied Algebra

[7] 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

[8] 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

[9] Gebauer, R. and Möller, H. M. *On an installation of Buchberger's algorithm*, 1988. Journal of Symbolic Computation

Thank you!

Questions? Comments?