IMPROVED PARALLEL GAUSSIAN ELIMINATION FOR GRÖBNER BASIS COMPUTATIONS IN FINITE FIELDS

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

October 01, 2015

University of Kaiserslautern

TABLE OF CONTENTS

LINEAR ALGEBRA FOR GRÖBNER BASIS COMPUTATIONS

USING LINEAR ALGEBRA TO COMPUTE GRÖBNER BASES

- Algorithms like Faugère's **F4** compute Gröbner bases via isolating the tasks of *searching for reducers* and *performing the reduction*.

USING LINEAR ALGEBRA TO COMPUTE GRÖBNER BASES

- 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.

USING LINEAR ALGEBRA TO COMPUTE GRÖBNER BASES

- 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.

USING LINEAR ALGEBRA TO COMPUTE GRÖBNER BASES

- 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.

USING LINEAR ALGEBRA TO COMPUTE GRÖBNER BASES

- 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.

IDEA BY FAUGÈRE & LACHARTRE

Specialize Linear Algebra for reduction steps in GB computations.

IDEA BY FAUGÈRE & LACHARTRE

Specialize Linear Algebra for reduction steps in GB computations.

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

IDEA BY FAUGÈRE & LACHARTRE

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.

IDEA BY FAUGÈRE & LACHARTRE

Specialize Linear Algebra for reduction steps in GB computations.

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

Try to exploit underlying GB structure.

IDEA BY FAUGÈRE & LACHARTRE

Specialize Linear Algebra for reduction steps in GB computations.

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

Try to exploit underlying GB structure.

IDEA BY FAUGÈRE & LACHARTRE

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rcl}
\text{S-pair} & \left\{ & \begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{matrix} \\
\text{S-pair} & \left\{ & \begin{matrix} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{matrix} \\
\text{reducer} & \leftarrow & 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 3 \quad 1
\end{array}
$$

Try to exploit underlying GB structure.

Specialize Linear Algebra for reduction steps in GB computations.

$$
\begin{array}{rcl}
\text{S-pair} & \left\{ & \begin{matrix} 1 & 3 & 0 & 0 & 7 & 1 & 0 \\ 1 & 0 & 4 & 1 & 0 & 0 & 5 \end{matrix} \\
\text{S-pair} & \left\{ & \begin{matrix} 0 & 1 & 6 & 0 & 8 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 7 & 0 \end{matrix} \\
\text{reducer} & \leftarrow & 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 3 \quad 1
\end{array}
$$

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.

FAUGÈRE-LACHARTRE IDEA

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

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

Pivot row

2nd step: Sort pivot and non-pivot rows

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

Pivot row

Non-Pivot row

2nd step: Sort pivot and non-pivot rows

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

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

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

3rd step: Reduce lower left part to zero

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

4th step: Reduce lower right part

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

4th step: Reduce lower right part

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

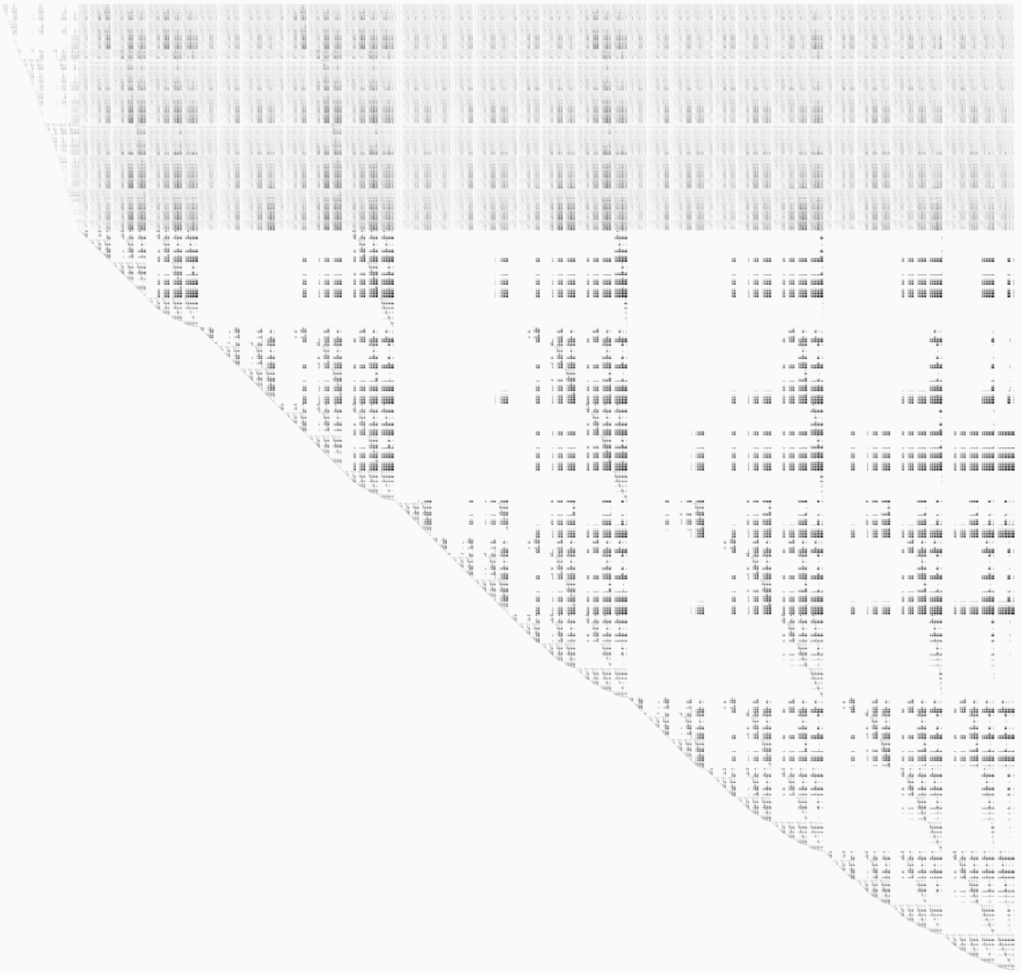
4th step: Reduce lower right part

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

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



WHAT OUR MATRICES LOOK LIKE

Some data about the matrix:

- **F4** computation of homogeneous KATSURA-12, degree 6 matrix

WHAT OUR MATRICES LOOK LIKE

Some data about the matrix:

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

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%)

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: $21,182 \times 22,207$

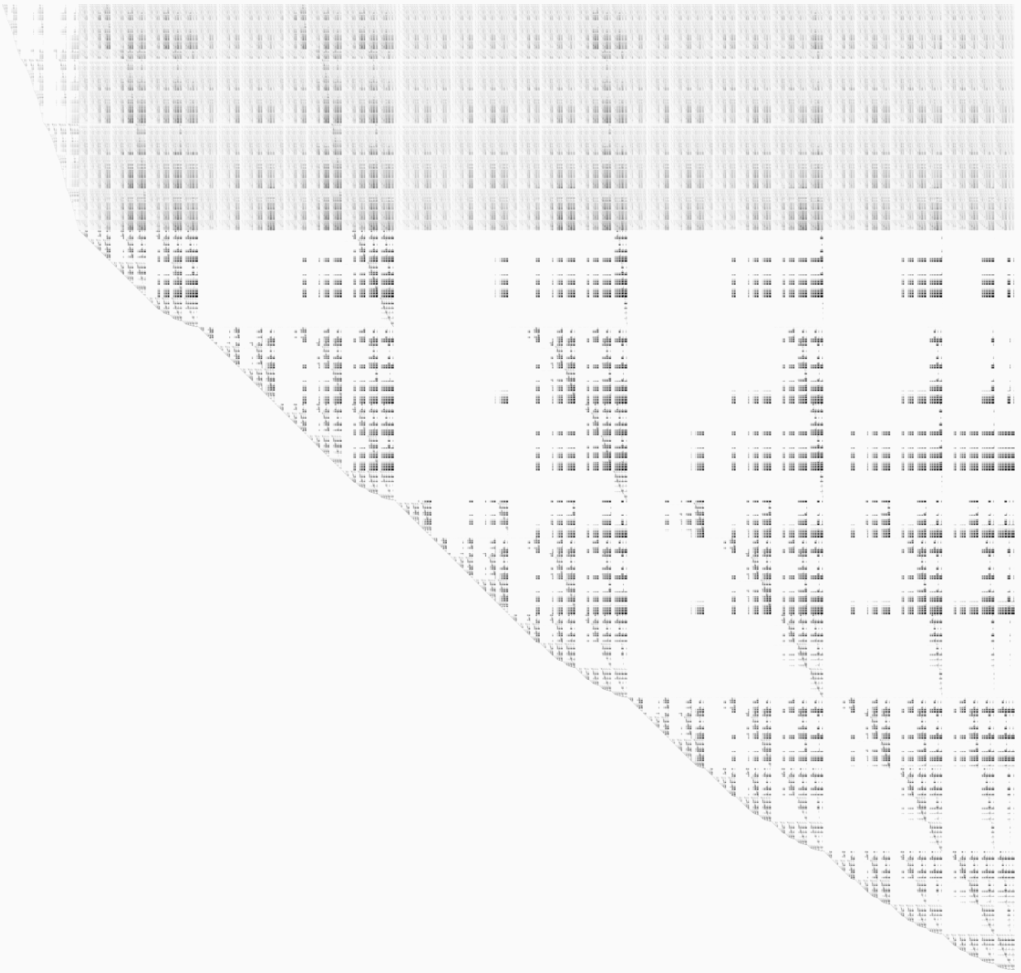
WHAT OUR MATRICES LOOK LIKE

Some data about the matrix:

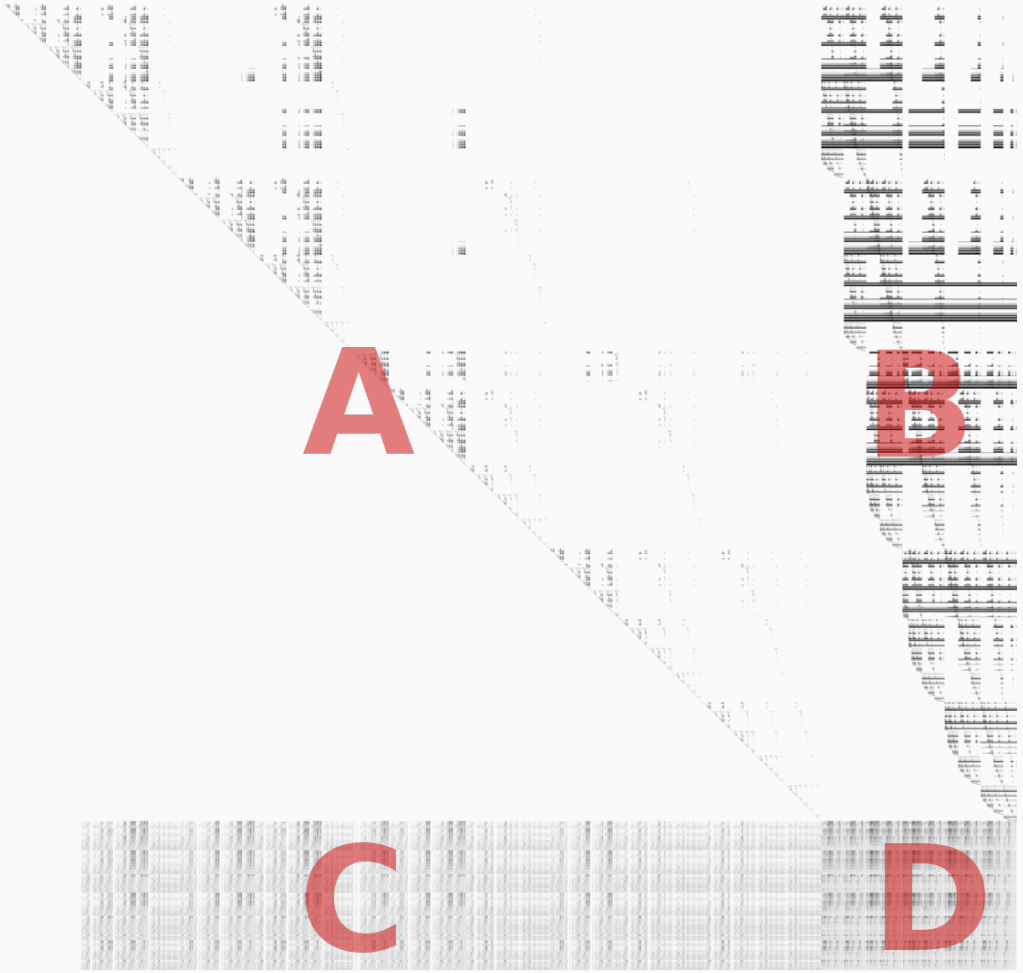
- **F4** computation of homogeneous KATSURA-12, degree 6 matrix
- Size 55MB
- $24,006,869$ nonzero elements (density: 5%)
- Dimensions:

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

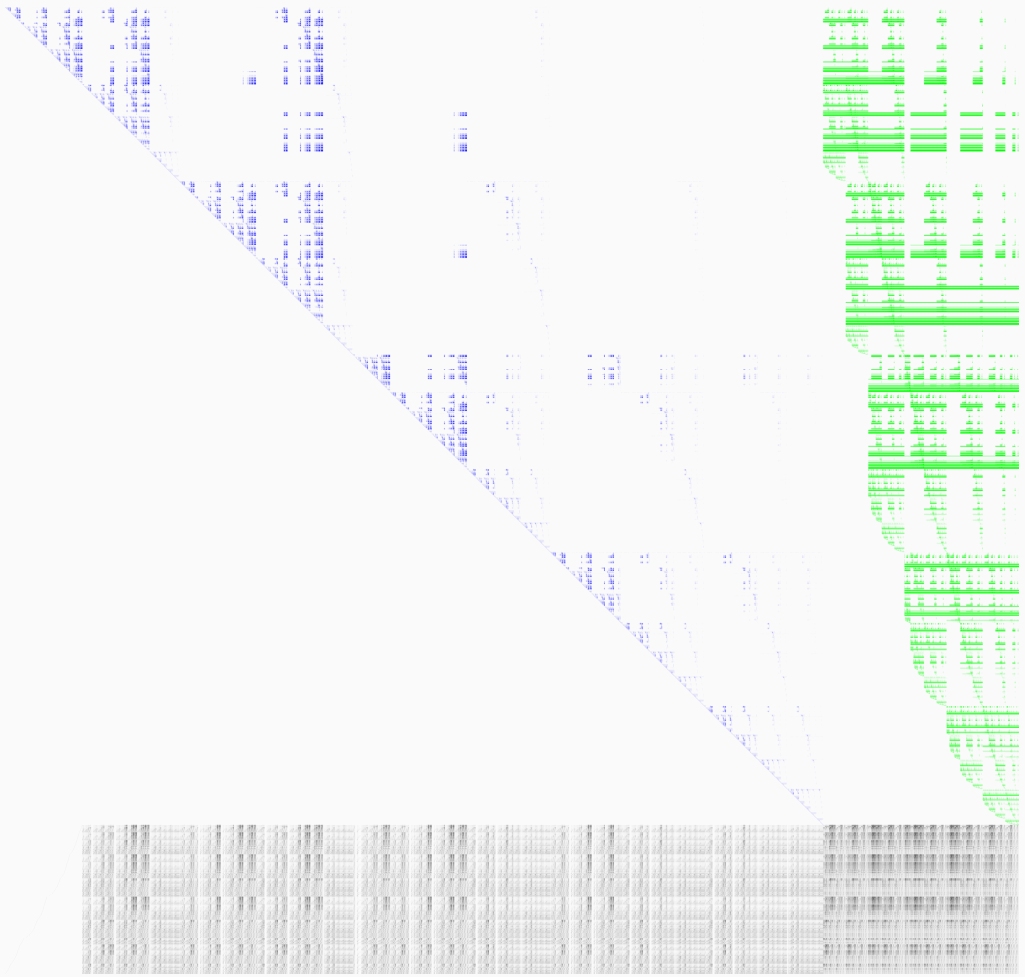
WHAT OUR MATRICES LOOK LIKE



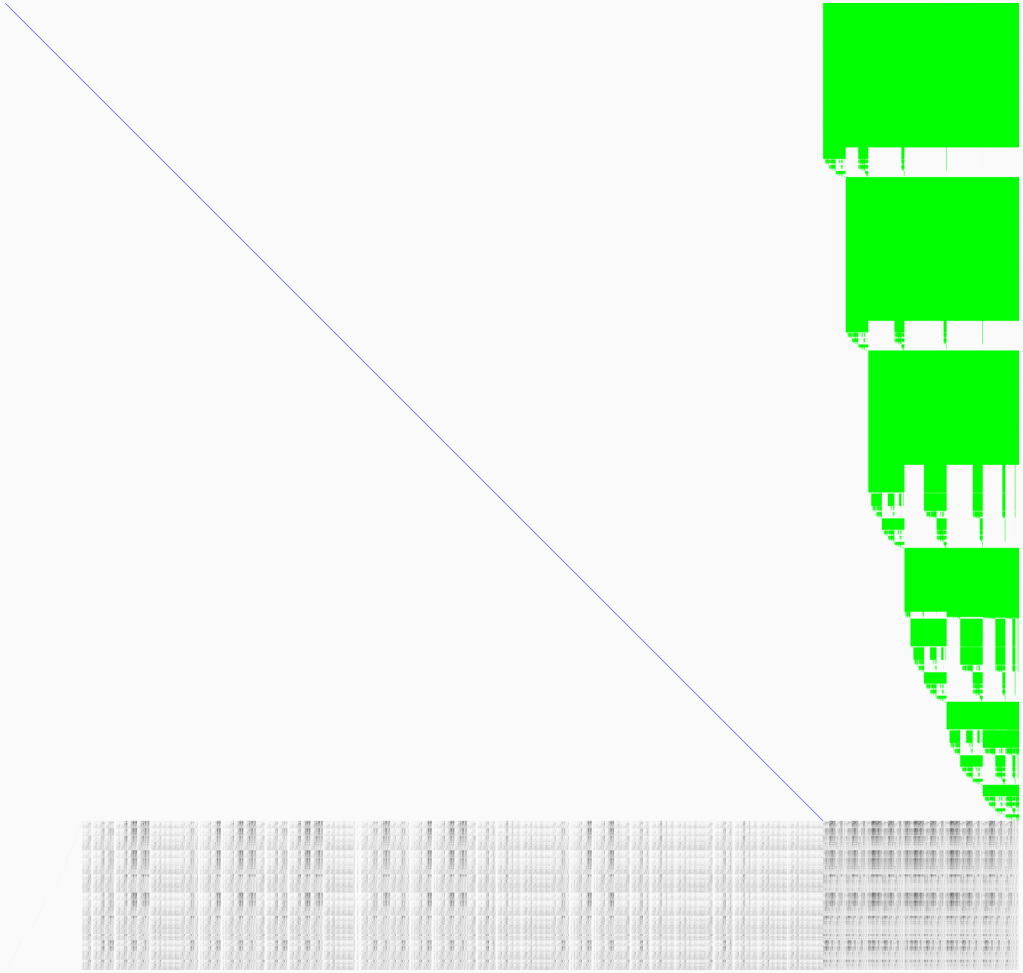
WHAT OUR MATRICES LOOK LIKE



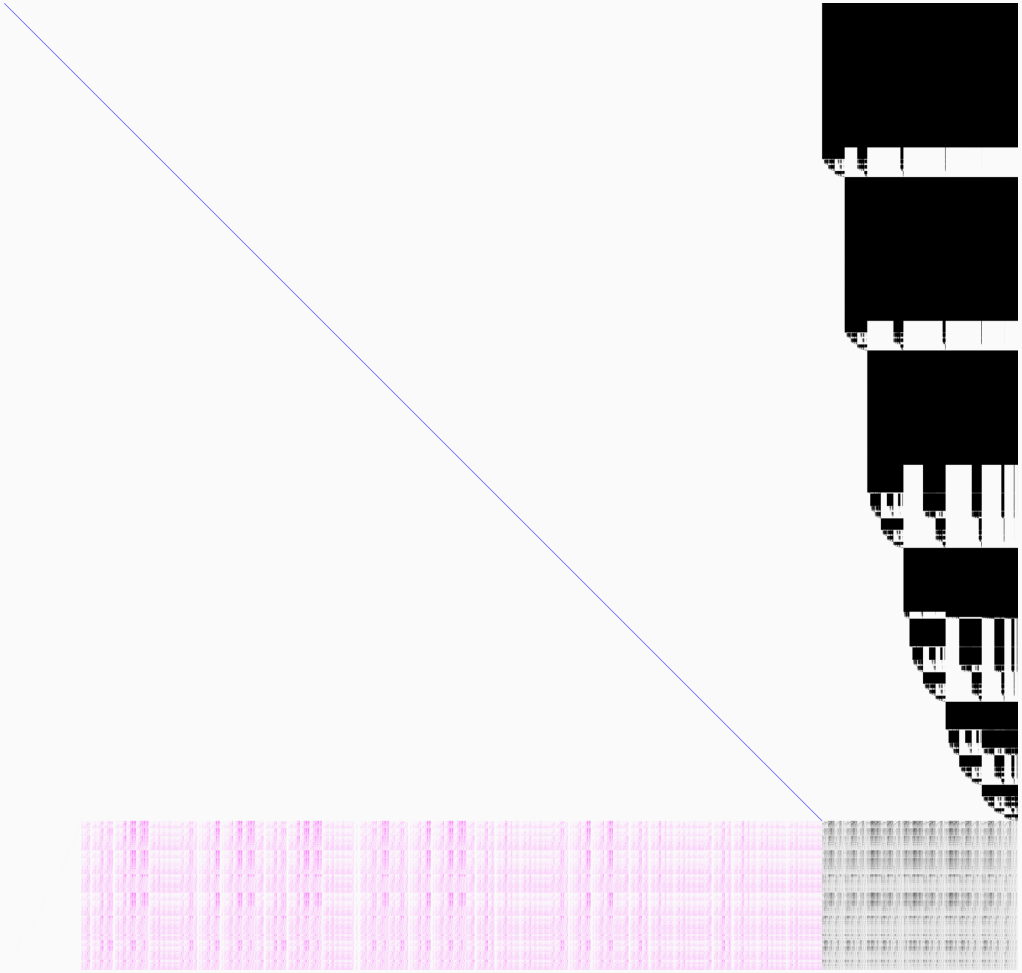
HYBRID MATRIX MULTIPLICATION $A^{-1}B$



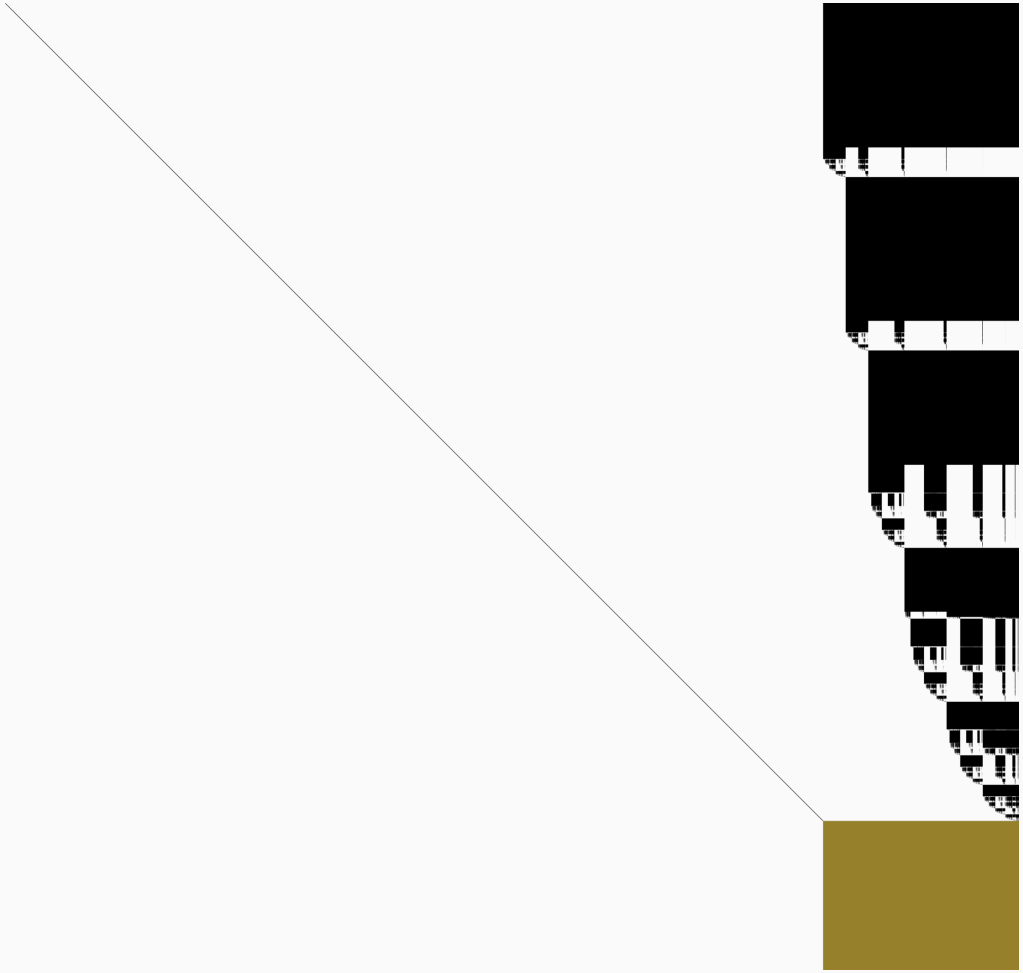
HYBRID MATRIX MULTIPLICATION $A^{-1}B$



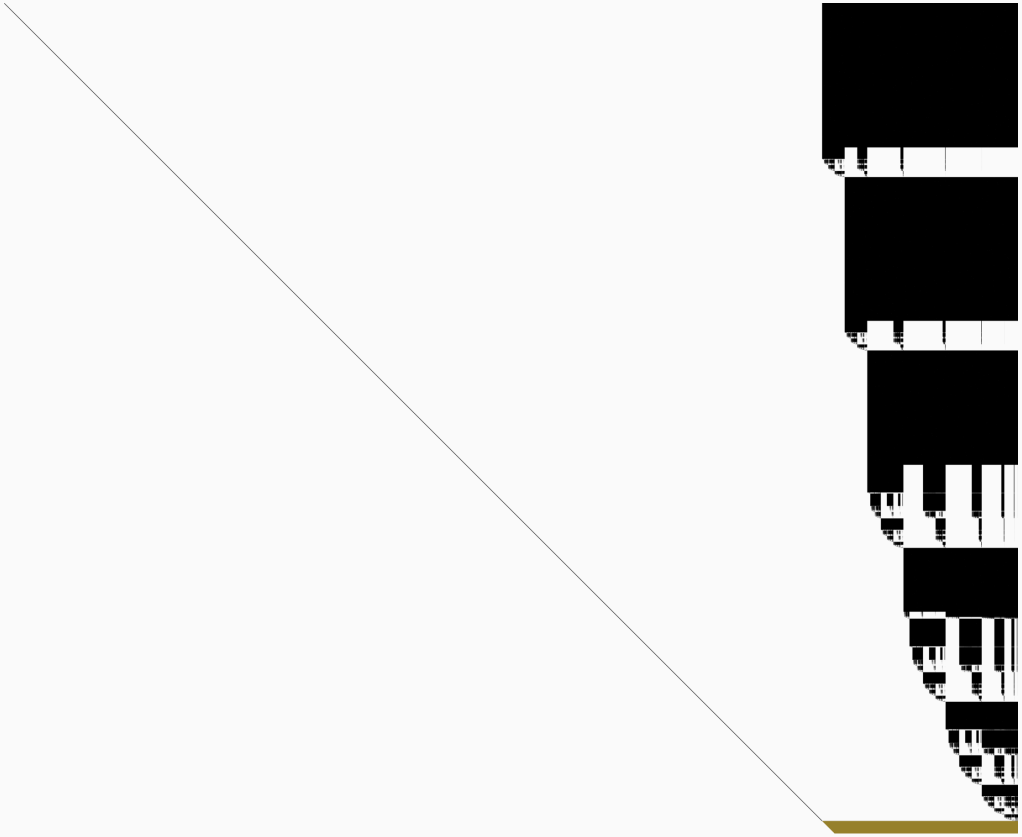
REDUCE C TO ZERO



GAUSSIAN ELIMINATION ON D



NEW INFORMATION



FEATURES OF GBLA

LIBRARY OVERVIEW

- **Open source** library written in **plain C**.

LIBRARY OVERVIEW

- **Open source** library written in **plain C**.
- Specialized linear algebra for GB computations.

LIBRARY OVERVIEW

- **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).

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.

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.

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.

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`

EXPLOITING BLOCK STRUCTURES IN MATRICES

Matrices from GB computations have **nonzero entries** often **grouped in blocks**.

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

EXPLOITING BLOCK STRUCTURES IN MATRICES

Matrices from GB computations have **nonzero entries** often **grouped in blocks**.

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

EXPLOITING BLOCK STRUCTURES IN MATRICES

Matrices from GB computations have **nonzero entries** often **grouped in blocks**.

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

EXPLOITING BLOCK STRUCTURES IN MATRICES

Matrices from GB computations have **nonzero entries** often **grouped in blocks**.

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

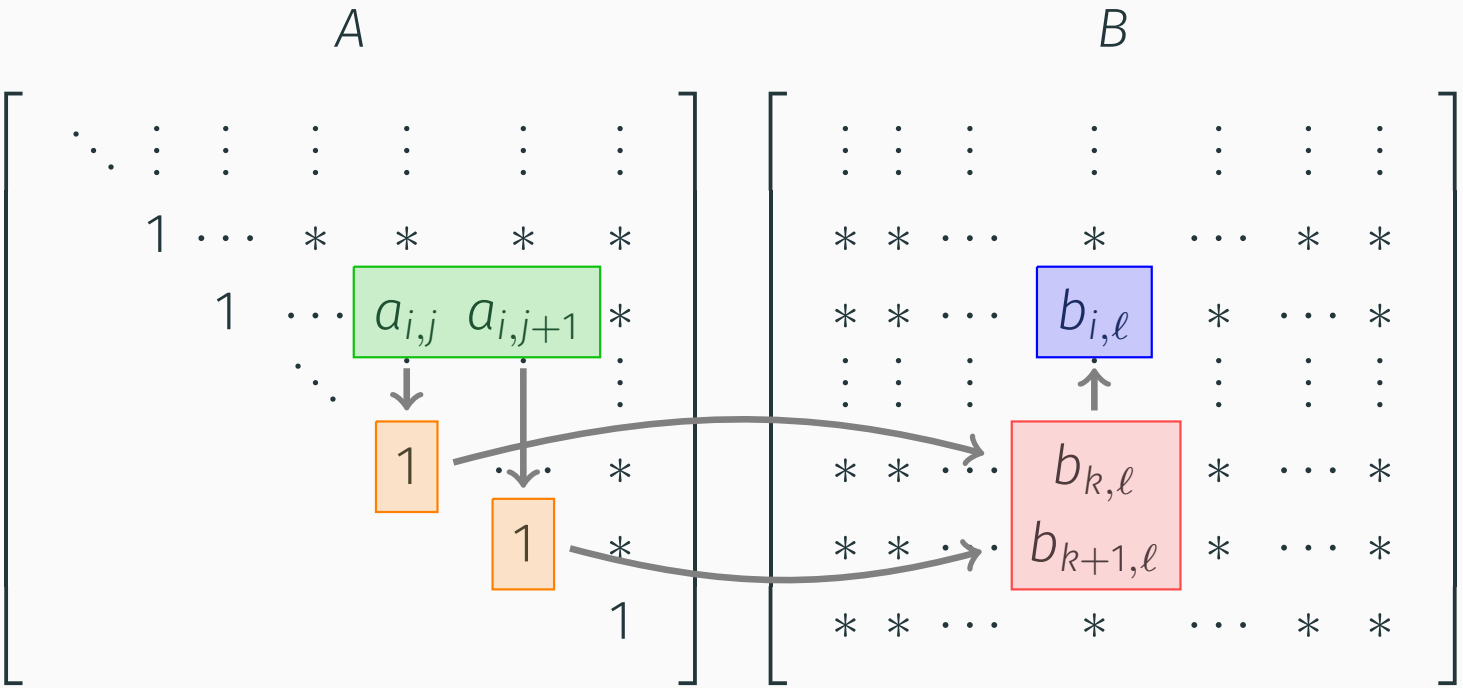
- Can be used to optimize **AXPY** and **TRSM** operations in FL reduction.
- Horizontal pattern taken care of canonically.

EXPLOITING BLOCK STRUCTURES IN MATRICES

Matrices from GB computations have **nonzero entries** often **grouped in blocks**.

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

MULTILINE DATA STRUCTURE – AN EXAMPLE

Consider the following two rows:

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

MULTILINE DATA STRUCTURE – AN EXAMPLE

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

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

MULTILINE DATA STRUCTURE – AN EXAMPLE

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

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

MULTILINE DATA STRUCTURE – AN EXAMPLE

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

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

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.

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 $A^{-1}B$.

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.

GBLA MATRIX FORMATS

- Matrices are pretty sparse, but structured.

GBLA MATRIX FORMATS

- 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.

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.

Table 1: 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`	# entries in matrix
`uint32_t`	`nnz`	`cols`	column index of entry
`uint32_t`	`m`	`rows`	length of rows

Table 2: 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 3: 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 3: 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

New format vs. **Old format**

Table 3: 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

New format vs. **Old format**

- 1/3rd of memory usage.

Table 3: 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

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

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

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

GBLA VS. FAUGÈRE-LACHARTRE

All timings in seconds.

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

GBLA VS. MAGMA V2.20-10

All timings in seconds.

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

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

[1] Reconstruction fails due to memory consumption

OUTLOOK

DIFFERENT APPROACHES

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

DIFFERENT APPROACHES

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

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**.

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**.

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**.
- First steps exploiting **heterogeneous CPU/GPU platforms** for GBLA.

REFERENCES

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

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

Buchberger, B.
Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, 1985.
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.
A new efficient algorithm for computing Gröbner bases (F4), 1999.
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.
On an installation of Buchberger's algorithm, 1988.
Journal of Symbolic Computation

THANK YOU!

COMMENTS? QUESTIONS?