From BLAS routines to finite field exact linear algebra solutions

Pascal Giorgi

Laboratoire de l'Informatique du Parallélisme (Arenaire team)
ENS Lyon - CNRS - INRIA - UCBL
France

Main goals

Solve Linear Algebra problems exactly using BLAS routines

Implementation in LinBox library

- Focus on finite fields
- Use matrix multiplication and BLAS routines as kernel
- Fast exact linear algebra routines (triangular solver, LSP factorization)

Finite field computations via BLAS routines

Main idea [Dumas-Gautier-Pernet 02]

Convert data from finite field to double
Direct call to BLAS
Convert back the result

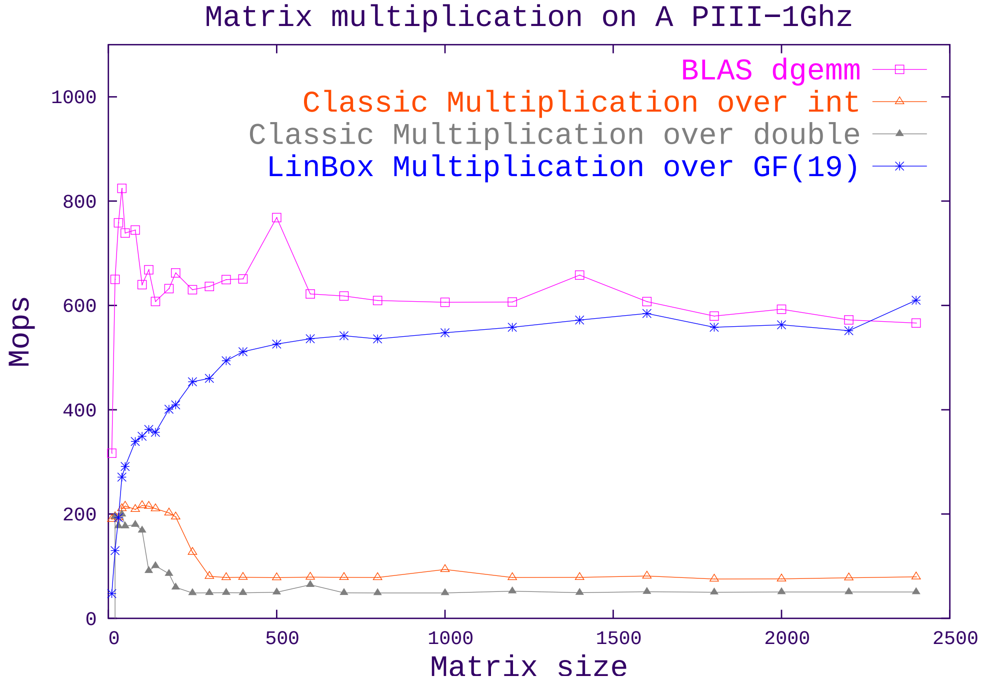
- Only one reduction
- Certify data hold over **53** bits
- Use division-free BLAS routines only

Illustration with matrix multiplication

- $(\mathbf{m}$ by $\mathbf{k}$ matrix$) \times (\mathbf{k}$ by $\mathbf{n}$ matrix$)$ over $\mathbf{GF(p)}$.

$$\text{certificate}: \quad \mathbf{k(p-1)^2 < 2^{53}}$$

- Performances with $\mathbf{m} = \mathbf{n} = \mathbf{k}$ and $\mathbf{p} = \mathbf{19}$:



- Even better with Strassen-Winograd algorithm [Dumas-Gautier-Pernet 02]

Our extension to a triangular solver with matrix r.h.s.

Problem:

- Certificate is $\mathbf{p^k < 2^{53}}$ instead of $\mathbf{kp^2 < 2^{53}}$ for matrix multiplication
  Direct call to BLAS only is too restrictive (only small matrices)

Solution:

- Use a block recursive algorithm
- Decrease matrix size until certification
- Use BLAS-based matrix multiplication routine to reconstruct the solution

Block recursive algorithm

- Solve $\mathbf{AX} = \mathbf{B}$ over $\mathbf{GF(p)}$
- While no certification

$$\begin{bmatrix} A_1 & A_2 \\ & A_3 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

$$\begin{cases} \text{solve } \mathbf{A_3X_2 = B_2} & \text{recursive call} \\ \mathbf{B_1} \leftarrow \mathbf{B_1} - \mathbf{A_2X_2} & \text{BLAS-based MM} \\ \text{solve } \mathbf{A_1X_1 = B_1} & \text{recursive call} \end{cases}$$

- Now, how to solve small (certified) linear systems?

Solving a certified triangular linear system

- Certified as soon as $(\mathbf{p}-\mathbf{1})\mathbf{p}^{\mathbf{m}-\mathbf{1}} < \mathbf{2}^{\mathbf{53}}$ ($\mathbf{m}$ = row dimension of small system $\mathbf{AX} = \mathbf{B}$)

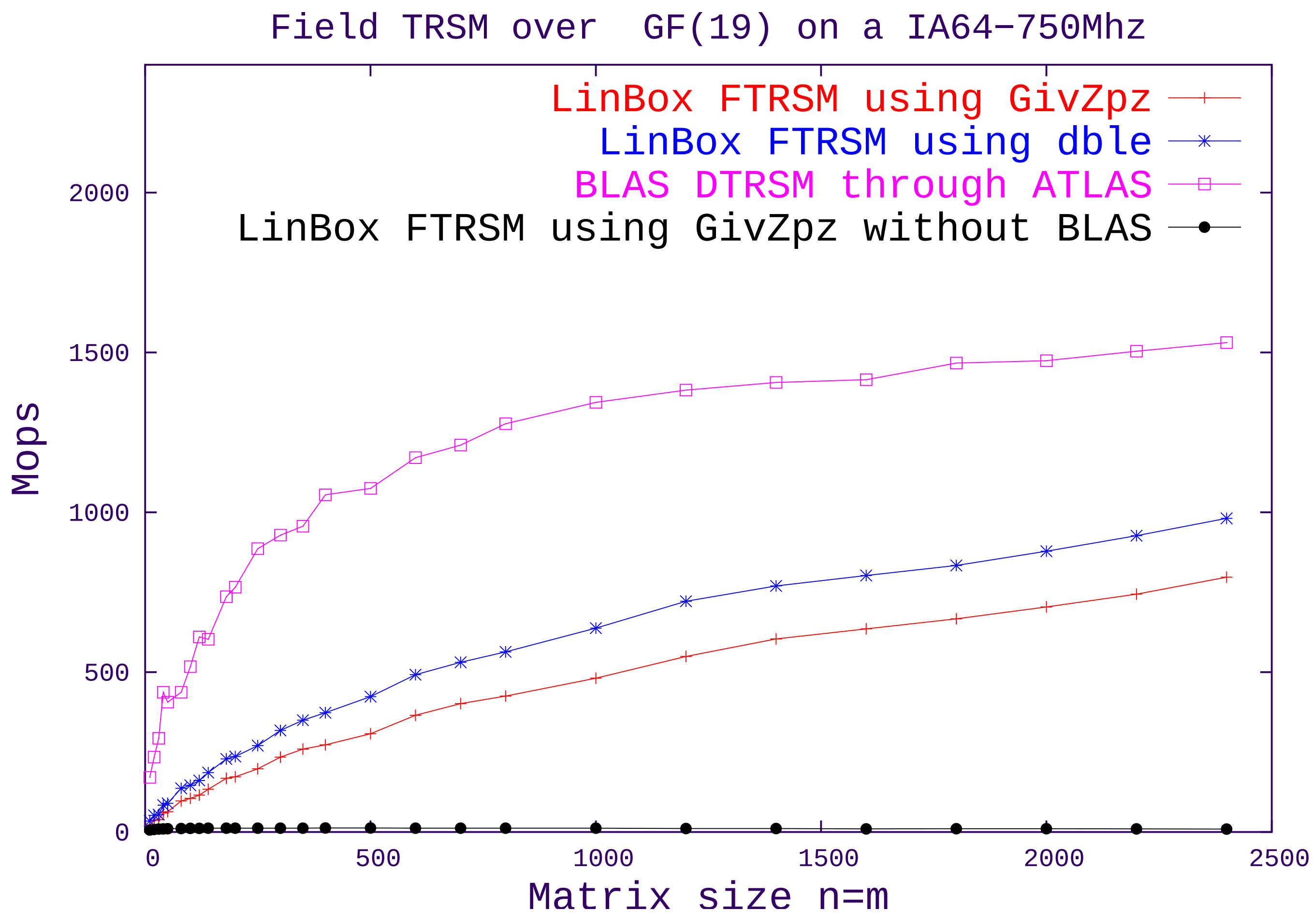
- Let $\mathbf{A} = \mathbf{UD}$ over $\mathbf{GF}(\mathbf{p})$

  with $\mathbf{D}$ a diagonal matrix and $\mathbf{U}$ a unit upper triangular matrix

  Solve $\mathbf{UY} = \mathbf{B}$ using the *dtrsm* BLAS routine

- Return $\mathbf{X} = \mathbf{D}^{-1}\mathbf{Y}$ over $\mathbf{GF}(\mathbf{p})$

Performances over $\mathbf{GF(19)}$ on Intel Itanium



In summary, we have just seen

- BLAS-based matrix multiplication
- BLAS-based triangular solver with matrix r.h.s.

Now, let us see

- LSP factorization

What is LSP factorization?

- **LSP** matrix factorization [Bini-Pan 94]

  L - lower triangular matrix with 1's on the main diagonal

  S - reduces to an u.t. matrix with nonzero diagonal entries when zero rows deleted

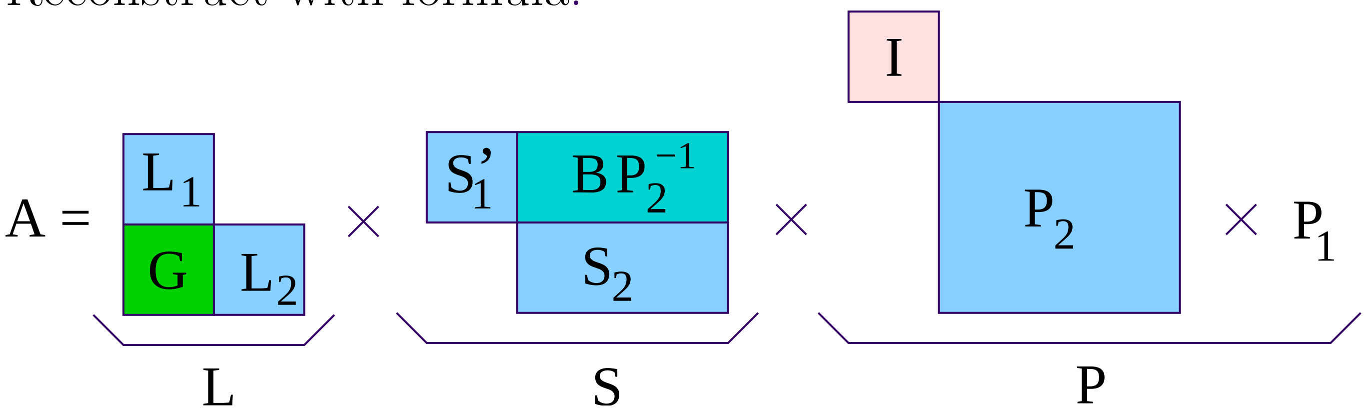
  P - permutation matrix

- Exemple over $\mathbf{GF(7)}$:

$$\begin{bmatrix} 1 & 3 & 5 & 2 & 4 & 6 \\ 3 & 2 & 1 & 6 & 5 & 4 \\ 3 & 3 & 6 & 0 & 1 & 2 \\ 5 & 3 & 3 & 6 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ 3 & 1 & & \\ 1 & & 1 & \\ 1 & & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 1 & 5 & 6 & 2 & 4 \\ & & & & & \\ & 2 & 1 & 3 & 5 & 4 \\ & & 1 & 1 & 5 & 3 \end{bmatrix} \times \begin{bmatrix} & 1 & & & & \\ 1 & & & & & \\ & & 1 & & & \\ & & & & & 1 \\ & & & 1 & & \\ & & & & 1 & \end{bmatrix}$$

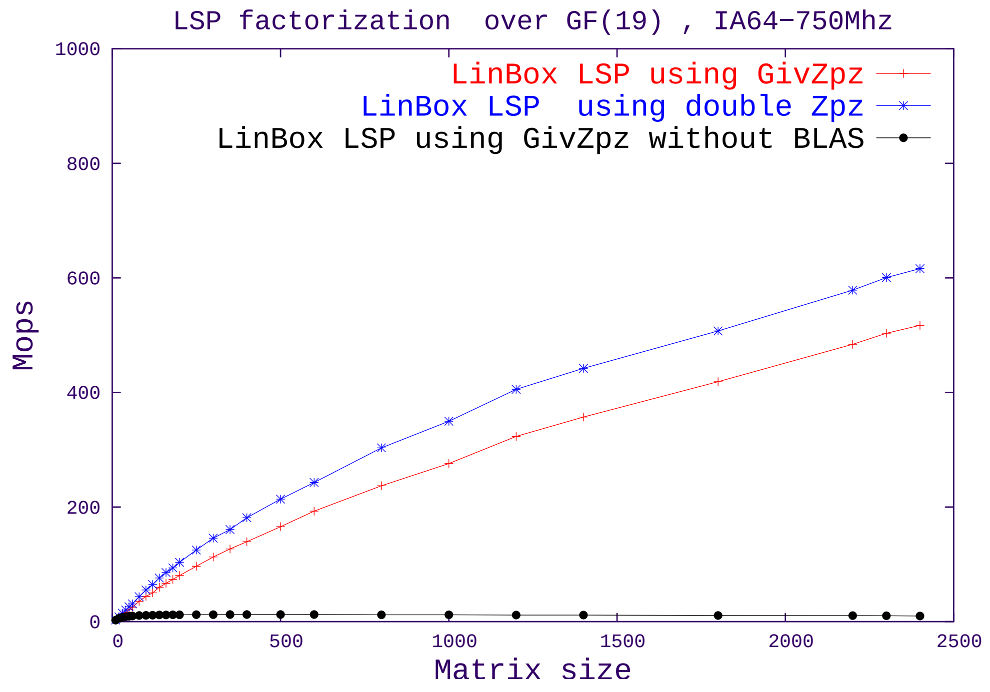
LSP factorization via matrix multiplication

- Recursive algorithm [Ibarra-Moran-Hui 82]:
- Partition $\mathbf{A} = \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \end{bmatrix}$ and factor $\mathbf{A_1} = \mathbf{L_1 S_1 P_1}$
- Partition $\mathbf{S_1} = [\mathbf{S_1'}\ \mathbf{B}]$ and $\mathbf{A_2 P_1^{-1}} = [\mathbf{C}\ \mathbf{D}]$
- Solve $\mathbf{G S'_1} = \mathbf{C}$ and factor $\mathbf{D} - \mathbf{G B} = \mathbf{L_2 S_2 P_2}$
- Reconstruct with formula:



Our implementation

- Solve $\mathbf{G}\mathbf{S}_1' = \mathbf{C}$ by using BLAS-based triangular solver
- Compute $\mathbf{D} - \mathbf{G}\mathbf{B}$ by using BLAS-based matrix multiplication
- Performances over $\mathbf{GF(19)}$ on Intel Itanium:



In summary, we have just seen

- BLAS-based matrix multiplication
- BLAS-based triangular solver with matrix r.h.s.
- BLAS-based LSP factorization

Now, let us see

- Availability in LinBox and application to minimal matrix polynomial

Integration in LinBox library

- FFLAS package [Dumas-Gautier-Pernet 02]

- Fast BLAS-based triangular solver (BLAS-like interface)

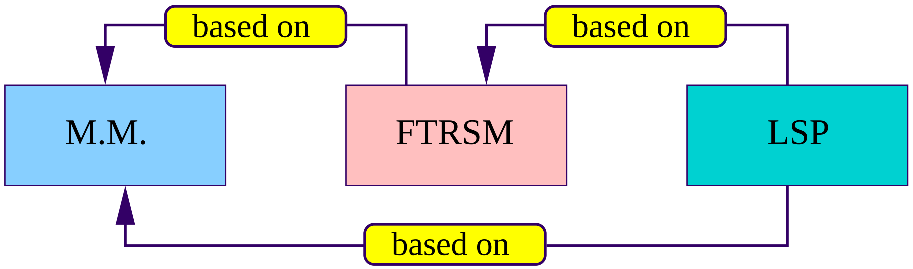
```
template < class Field >
        void Field_trsm(  const Field& F,
                          int m, int n,
                          typename Field::Element * B, int ldb,
                          typename Field::Element * A, int lda,
                          typename Field::Element * X, int ldx,
                          Triangular tr,
                          Unitary un,
                          Side si);
```

- Fast BLAS-based LSP factorization

- Genericity over the domain $\mathbf{GF(p)}$ and dense matrices (with BLAS-like storage)

Features of our LinBox implementation routines

- Easier to implement higher level algorithms based on matrix multiplication
- Speeding up matrix multiplication $\Longrightarrow$ faster routines



- Example: computation of minimal matrix polynomial

Minimal matrix polynomial $\mathbf{\Pi_A(x)}$ of a square matrix $\mathbf{A}$

- Important algorithm in LinBox (Krylov/Lanczos approach)

Two main steps of the algorithm:

- Compute the first terms of $\mathbf{UV}, \mathbf{UAV}, \mathbf{UA^2V}, \ldots$

- Deduce $\mathbf{\Pi_A(x)}$ by computing a matrix approximant of $\begin{bmatrix} \sum_{\mathbf{i}} (\mathbf{UA^iV})\mathbf{x^i} & 0 \\ 0 & \mathbf{I} \end{bmatrix}$

  [Turner's PhD Thesis 02]

Matrix approximant algorithm

- Beckermann-Labahn's algorithm via matrix multiplication [Giorgi-Jeannerod-Villard 03]

- Iterative algorithm which computes approximant $\mathbf{M(x)}$ s.t.

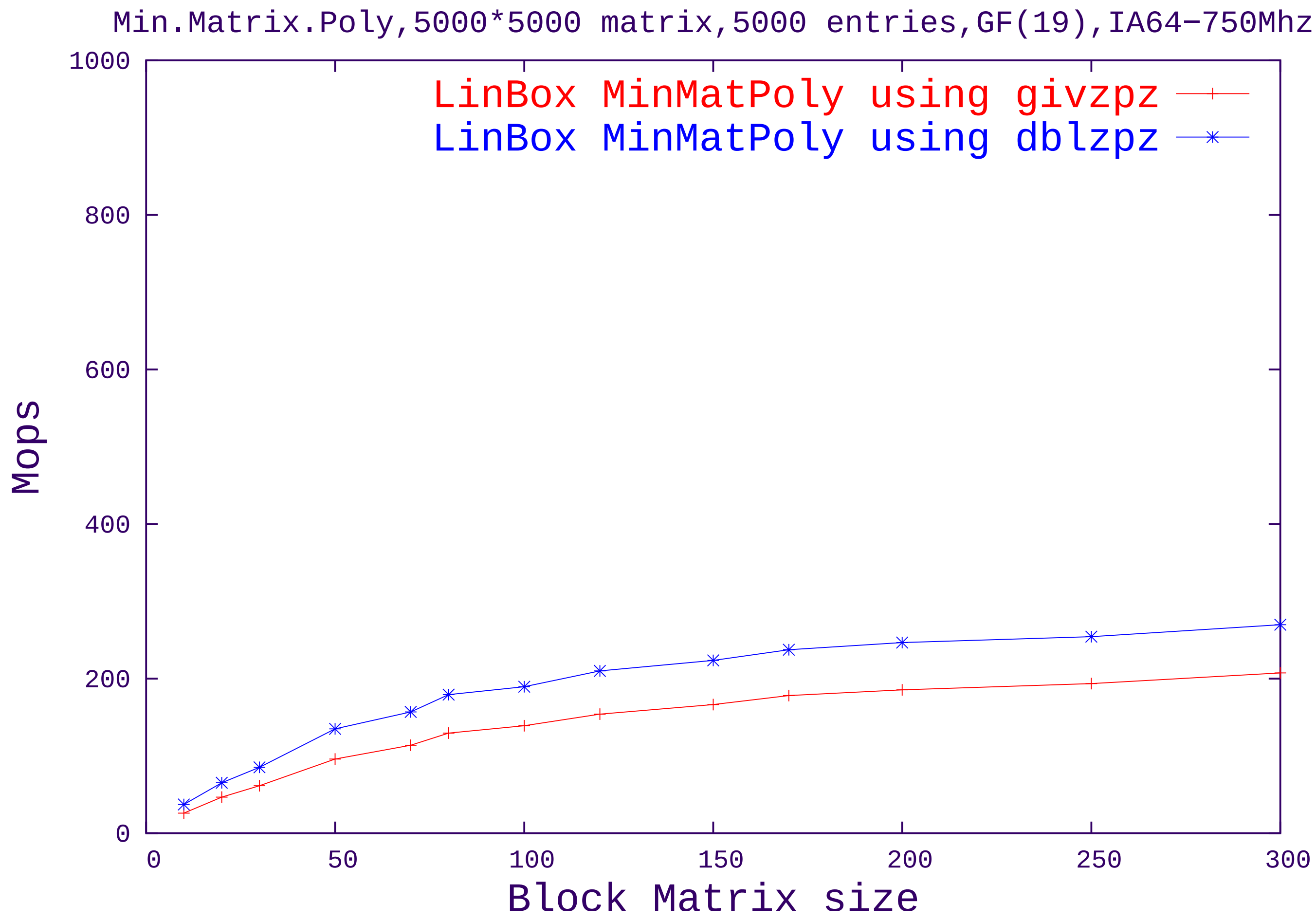
$$\mathbf{M(x)F(x)} = \mathbf{O(x^{\sigma})} \text{ in } \boldsymbol{\sigma} \text{ steps}$$

- Main operations involved at step **k**

  - **k** calls to matrix multiplication
  - **1** call to **LSP** factorization
  - **k** calls to triangular system solving

- We have used our LinBox BLAS-based routines to implement this algorithm

First performances over $\mathbf{GF(19)}$ on Intel Itanium



Conclusion and future work

- Significant improvement for some linear algebra problems over $\mathbf{GF(p)}$
- Implementation in LinBox library [www.linalg.org]
- Extension to sparse matrices
- Extension to other algorithms using matrix multiplication