The LinBox library

Clément PERNET & the LinBox group

CAT Workshop,
Aug 29, 2009
Introduction

Exact linear algebra:

- over $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}_p, \text{GF}(p^k)$.
- matrix-multiply, solve, rank, det, echelon, charpoly, Smith-Normal-Form, ...
- dense, sparse, blackbox matrices
Introduction

Exact linear algebra:

- over $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}_p, \text{GF}(p^k)$.
- matrix-multiply, solve, rank, det, echelon, charpoly, Smith-Normal-Form, ...
- dense, sparse, blackbox matrices

Growing applicative demand

- CAT: Homology of simplicial complexes
- Number Theory: computing modular forms,
- Crypto: NFS, DLP Groebner bases, ...
- Graph Theory: closure, spectrum, ...
- High precision approximate linear algebra
- ... (Mathematics is the art of reducing any problem to linear algebra [W. Stein])
Software solutions for exact computations

Specialized libraries

finite fields:  NTL, Givaro, Lidia, ...
integers:     GMP, MPIR
polynomials:  NTL, Givaro, zn_poly ...
# Software solutions for exact computations

## Specialized libraries

- **finite fields:** NTL, Givaro, Lidia, ...
- **integers:** GMP, MPIR
- **polynomials:** NTL, Givaro, zn_poly ...

## End-user level softwares

- Maple, Mathematica, MuPad, ... (closed source)
- Sage, Pari, Maxima, ... (open source)
# Software solutions for exact computations

## Specialized libraries

<table>
<thead>
<tr>
<th>Finite fields:</th>
<th>NTL, Givaro, Lidia, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers:</td>
<td>GMP, MPIR</td>
</tr>
<tr>
<td>Polynomials:</td>
<td>NTL, Givaro, zn_poly ...</td>
</tr>
</tbody>
</table>

## End-user level softwares

- Maple, Mathematica, MuPad, ... (closed source)
- Sage, Pari, Maxima, ... (open source)

Linear Algebra ?
Outline

1. Organization and design

2. Algorithmic models
   - Black box matrices
   - Dense matrices
   - Sparse matrices
   - Lifting over the integers

3. Evolution and perspectives
Outline

1. Organization and design

2. Algorithmic models
   - Black box matrices
   - Dense matrices
   - Sparse matrices
   - Lifting over the integers

3. Evolution and perspectives
A generic middleware

- uses basic implementations from specialized libraries (GMP, Givaro, NTL, BLAS...)
- Optional libraries used in a Plug & Play manner
- Interfaces to top-level softwares (Maple, Sage, GAP)
The LinBox project, facts

Joint NFS-NSERC-CNRS project.

**US:** U. of Delaware, North Carolina State U.
**Canada:** U. of Waterloo, U. of Calgary,
**France:** Grenoble U., INRIA (Lyon, Grenoble)
The LinBox project, facts

Joint NFS-NSERC-CNRS project.

**US:** U. of Delaware, North Carolina State U.

**Canada:** U. of Waterloo, U. of Calgary,

**France:** Grenoble U., INRIA (Lyon, Grenoble)

A LGPL source library:
- 125 000 lines of C++ code
- about 5 active developers

Available online: [http://linalg.org](http://linalg.org)

**Google groups:** linbox-devel, linbox-use

Distributed in Debian and Sage
Design of LinBox v1

Features:

**Solutions**
- rank
- det
- minpoly
- charpoly
- solve
- positive definiteness
- Smith normal form
Design of LinBox v1

Features:

### Solutions
- rank
- det
- minpoly
- charpoly
- solve
- positive definiteness
- Smith normal form

### Domains of computation
- $\mathbb{Z}_p, \mathbb{F}_q$
- $\mathbb{Z}$

### Matrices
- Dense
- Sparse
- Blackbox
Genericity

- **Domain wrt. element representations:**
  ```cpp
template <class Element>
class Modular<Element>;
```

- **Matrix wrt. domains:**
  ```cpp
template <class Field>
class DenseMatrix<Field>;
```

- **Algorithms wrt. matrices:**
  ```cpp
template <class Matrix>
unsigned long rank (unsigned long & r,
const Matrix & A);
```
Interface

Field/Ring Plug & Play interface

- Common interface with Givaro
  
  Modular<int> F(11);
  int x, y, z;
  F.init(x, 2);
  F.init(y, 13);
  F.mul(z, x, y);

- Wraps NTL, Lidia, Givaro implementations
- Proper floating point based implementations for dense computations

BLAS

Compliant with the standard C-BLAS interface

- GotoBLAS, ATLAS, MKL, GSL, ...
Structure of the library

Choosing the algorithm from the type of matrix and domain (automated by default)

Solutions
- det
- rank
...

Component implementation
- NTL::ZZp
- Toeplitz
...

Template specialization / Traits

Algorithms
- Wiedemann
- LU
- ...

Notes:
- Algorithms
- Solutions
- Component implementation
- Choosing the algorithm from the type of matrix and domain (automated by default)

Library organization:
- Introduction
- Organization and design
- Algorithmic models
- Evolution and perspectives
Several levels of use

- Web servers: http://www.linalg.org
Several levels of use

- **Web servers**: [http://www.linalg.org](http://www.linalg.org)
- **Executables**: `charpoly MyMatrix 65521`
Several levels of use

- **Web servers:** [http://www.linalg.org](http://www.linalg.org)
- **Executables:** `$ charpoly MyMatrix 65521$
- **Call to a solution:**
  ```
  NTL::ZZp F(65521);
  Toeplitz<NTL::ZZp> A(F);
  Polynomial<NTL::ZZp> P;
  charpoly (P, A);
  ```
Several levels of use

- **Web servers:** [http://www.linalg.org](http://www.linalg.org)
- **Executables:** `$ charpoly MyMatrix 65521$
- **Call to a solution:**
  ```cpp
  NTL::ZZp F(65521);
  Toeplitz<NTL::ZZp> A(F);
  Polynomial<NTL::ZZp> P;
  charpoly (P, A);
  ```
- **Calls to specific algorithms**
Outline

1. Organization and design

2. Algorithmic models
   - Black box matrices
   - Dense matrices
   - Sparse matrices
   - Lifting over the integers

3. Evolution and perspectives
Black box linear algebra
Black box linear algebra

- Matrices viewed as linear operators
- Algorithms based on matrix-vector apply only $\Rightarrow$ cost $E(n)$

$$A \in K^{n \times m}$$

$v \in K^m \xrightarrow{} Av \in K^n$
Black box linear algebra

- Matrices viewed as linear operators
- Algorithms based on matrix-vector apply only $\Rightarrow$ cost $E(n)$

![Diagram of linear operator](image)

**Structured matrices:** Fast apply (e.g. $E(n) = O(n \log n)$)

**Sparse matrices:** Fast apply and no fill-in
Black box linear algebra

- Matrices viewed as linear operators
- Algorithms based on matrix-vector apply only $\Rightarrow$ cost $E(n)$

\[ A \in K^{n \times m} \]
\[ v \in K^m \quad \Rightarrow \quad Av \in K^n \]

Structured matrices: Fast apply (e.g. $E(n) = O(n \log n)$)
Sparse matrices: Fast apply and no fill-in

$\Rightarrow$

- Iterative methods
- No access to coefficients, trace, no elimination
- Matrix multiplication $\Rightarrow$ Black-box composition
Black box linear algebra

**Minimal polynomial:** [Wiedemann 86]
- ⇒ adapts numerical iterative Krylov/Lanczos methods
- ⇒ $O\left(dE(n) + n^2\right)$ operations
Black box linear algebra

**Minimal polynomial:** [Wiedemann 86]
- adapts numerical iterative Krylov/Lanczos methods
- $O\left(dE(n) + n^2\right)$ operations

**Rank, Det, Solve:** [Kaltofen & Saunders 90, Chen & Al. 02]
- reduced to minimal polynomial and preconditioners
- $O^\sim\left(nE(n) + n^2\right)$ operations
  where $E(n)$: cost of applying the matrix to a vector
Black box linear algebra

**Minimal polynomial:** [Wiedemann 86]
⇒ adapts numerical iterative Krylov/Lanczos methods
⇒ $\mathcal{O} \left( dE(n) + n^2 \right)$ operations

**Rank, Det, Solve:** [Kaltofen & Saunders 90, Chen& Al. 02]
⇒ reduced to minimal polynomial and preconditioners
⇒ $\mathcal{O}^\sim \left( nE(n) + n^2 \right)$ operations

where $E(n)$ : cost of applying the matrix to a vector

**Smith Normal Form:** [Dumas & Al. 02] cf. J-G. Dumas talk
Dense matrices: **FFLAS–FFPACK**

Building block: **matrix multiplication over word-size finite field**

**Principle:**
- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)

**Graph:**

```
<table>
<thead>
<tr>
<th>Dimension</th>
<th>Vitesse (Mops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td>2000</td>
<td>2500</td>
</tr>
<tr>
<td>3000</td>
<td>3500</td>
</tr>
<tr>
<td>4000</td>
<td>4500</td>
</tr>
<tr>
<td>5000</td>
<td>5500</td>
</tr>
<tr>
<td>6000</td>
<td>7000</td>
</tr>
<tr>
<td>8000</td>
<td>9000</td>
</tr>
<tr>
<td>10000</td>
<td></td>
</tr>
</tbody>
</table>
```

**Comparison:**
- BLAS dgemm (in IR)
- FFLAS fgemm (in \(\mathbb{Z}/65521\) \(\mathbb{Z}\))
Dense matrices: FFLAS–FFPACK

Building block: matrix multiplication over word-size finite field

Principle:
- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- Cache tuning

⇒ rely on the existing BLAS
Dense matrices: **FFLAS–FFPACK**

**Building block:** matrix multiplication over word-size finite field

**Principle:**
- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- Cache tuning

⇒ rely on the existing BLAS
Dense matrices: **FFLAS–FFPACK**

**Building block:** matrix multiplication over word-size finite field

**Principle:**
- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- Cache tuning

⇒ rely on the existing BLAS

![Graph showing performance comparison between BLAS and FFLAS on an Opteron 2.4 GHz with 4 GB RAM. The x-axis represents the dimension of the matrices, and the y-axis represents the number of Mflops. The graph compares the performance of the classical multiplication in \( \mathbb{Z}/65521\mathbb{Z} \) on a P4, 3.4 GHz processor with different implementations.](image-url)
Dense matrices: **FFLAS–FFPACK**

Building block: **matrix mutlip. over word-size finite field**

Principle:
- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- Cache tuning

⇒ rely on the existing BLAS

- Sub-cubic algorithm (Winograd)
Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.
Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

⇒ Block algorithm with multiple cascade structures

\[
\begin{pmatrix}
X_{i-1,i-1} \\
X_i \\
\end{pmatrix} =
\begin{pmatrix}
V_i \\
\end{pmatrix}
^{-1}
\begin{pmatrix}
B_{i-1,i-1} \\
B_i \\
\end{pmatrix}
\]
Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

⇒ Block algorithm with multiple cascade structures

\[
X_{i,i-1} = \ldots \cdot X_i = \ldots = U^{-1}_i V X_i B_{1..i-1} \cdot B_i_{1..i-1}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>5000</th>
<th>10 000</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRSM</td>
<td>1,66</td>
<td>1,33</td>
<td>1,24</td>
<td>1,12</td>
<td>1,01</td>
</tr>
<tr>
<td>LQUP</td>
<td>2,00</td>
<td>1,56</td>
<td>1,43</td>
<td>1,18</td>
<td>1,07</td>
</tr>
<tr>
<td>INVERSE</td>
<td>1.62</td>
<td>1.32</td>
<td>1.15</td>
<td>0.86</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Characteristic Polynomial:

\[
\begin{array}{c|ccc}
   n & 500 & 5000 & 15 000 \\
\hline
   \text{LinBox} & 0.91s & 4m44s & 2h20m \\
   \text{magma-2.13} & 1.27s & 15m32s & 7h28m \\
\end{array}
\]
Sparse Matrices

Two approaches:

**Blackbox:**
- No fill-in,
- \( E(n) = \mathcal{O}(\#\text{non-zero-elt}) \)

**Sparse elimination:**
- local pivoting strategies
- switch to dense elimination when too much fill-in
Lifting over the integers

**Multimodular reconstruction**
- scalars and vectors
- early termination (with user-specified probability of success)
- or deterministic (e.g. Hadamard’s bound)

**$p$-adic lifting**
- **dense matrices:** Dixon’s lifting with LU decomposition
- **blackbox/sparse matrices:** no inverse nor LU can be computed
  - Wiedemann lifter
  - block-Wiedemann lifter
  - block-Hankel lifter
Outline

1. Organization and design

2. Algorithmic models
   - Black box matrices
   - Dense matrices
   - Sparse matrices
   - Lifting over the integers

3. Evolution and perspectives
Block Krylov projections

- **Wiedemann algorithm:** scalar projections of $A^i$ for $i = 0..2d$:
  
  \[ u^T v, u^T Av, \ldots, u^T A^{2d/k} v \]  
  such that $u, v$ are $n \times 1$

- **Block Wiedemann:** $k \times k$ dense projections of $A^i$ for $i = 1..2d/k$
  
  \[ U^T V, U^T AV, \ldots, U^T A^{2d/k} V, \]  
  such that $U, V$ are $n \times k$
Block Krylov projections

- **Wiedemann algorithm**: scalar projections of $A^i$ for $i = 0..2d$:

  $$u^T v, u^T Av, \ldots, u^T A^{2d/k} v$$ such that $u, v$ are $n \times 1$

- **Block Wiedemann**: $k \times k$ dense projections of $A^i$ for $i = 1..2d/k$

  $$U^T V, U^T AV, \ldots, U^T A^{2d/k} V$$, such that $U, V$ are $n \times k$

- Building block of the most recent algorithmic advances
- In practice: better balance efficiency between Blackbox and dense methods
Packed matrices over small finite fields

**GF(2): M4RI [Albrecht, Bard & Al.]**

- Packed representation of elements:
  \[ \text{long long} \equiv \text{GF}(2)^{64} \]
- Greasing technique: tables, and Gray codes
- SSE2 support and cache friendliness
- sub-cubic matrix arithmetic
Packed matrices over small finite fields

**GF(2):** M4RI [Albrecht, Bard & Al.]
- Packed representation of elements:
  \[ \text{long long} \equiv \text{GF}(2)^{64} \]
- Greasing technique: tables, and Gray codes
- SSE2 support and cache friendliness
- sub-cubic matrix arithmetic

**GF(3, 5, 7):** similar projects [Bradshaw, Boothby]

**GF(p), p < 2^8:** Kronecker substitution [Dumas 2008]

\[ (a_1, a_2, a_3) \rightarrow a_1 X^2 + a_2 X + a_2 \rightarrow a_1 \alpha^2 + a_2 \alpha + a_2 \]

integer on 64 bits
Packed matrices over small finite fields

**GF(2):** M4RI [Albrecht, Bard & Al.]
- Packed representation of elements:
  \[ \text{long long} \equiv \text{GF}(2)^{64} \]
- Greasing technique: tables, and Gray codes
- SSE2 support and cache friendliness
- sub-cubic matrix arithmetic

**GF(3, 5, 7):** similar projects [Bradshaw, Boothby]

**GF(p), p < 2^8:** Kronecker substitution [Dumas 2008]

\[(a_1, a_2, a_3) \rightarrow a_1X^2 + a_2X + a_2 \rightarrow a_1\alpha^2 + a_2\alpha + a_2\]

integer on 64 bits

\[\Rightarrow\text{Matrices are no longer containers of field elements}\]
Evolution and perspectives

LinBox 2.0 in the radar: major rewrite of the library

- Clean up and simplify existing code
- Unify the usage block-Krylov/Wiedemann
- Redesign dense matrices (enabling packing for small finite fields)
- Support for new architecture framework: GPU, GPU/CPU, multi-core, grid computing...
  ⇒ Workstealing and adaptive scheduling libs: Cilk, Kaapi
- New algorithms...