Fast exact linear algebra: LinBox

Clément PERNET

SAGE Days 6,
November 11, 2007
Outline

**LinBox: an overview**
- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

**Memory efficient implementations**
- In-place eliminations
- Fast matrix multiplication

**Linear algebra over big integers**
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
LinBox: an overview

Principles
Organisation of the library
Dense computations
BlackBox computations

Memory efficient implementations
In-place eliminations
Fast matrix multiplication

Linear algebra over big integers
LinBox

A generic middleware

Sage  GAP  Maple  CoCoA

Finite fields
  NTL
  Givaro
  ...

BLAS
  ATLAS
  GOTO
  ...

GMP

LinBox

Givaro  NTL  GAP  Maple  CoCoA

GAP  Maple  CoCoA

Sage
The LinBox project, facts

Joint NSF–CNRS project.
- U. of Delaware, North Carolina State U.
- U. of Waterloo, U. of Calgary,
- Laboratoires LJK, ID (Grenoble), LIP (Lyon)
The LinBox project, facts

Joint NSF–CNRS project.
- U. of Delaware, North Carolina State U.
- U. of Waterloo, U. of Calgary,
- Laboratoires LJK, ID (Grenoble), LIP (Lyon)

A LGPL source library:
- 122 000 lines of C++ code
- 5-10 active developers
Brief LinBox history

1998 Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002 v0.1 first beta release
Brief LinBox history

1998  Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002  v0.1 first beta release

March 2003  Introduction of FFLAS–FFPACK dense linear algebra kernel

1998  Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002  v0.1 first beta release

March 2003  Introduction of FFLAS–FFPACK dense linear algebra kernel


Feb 2007  Wrapped in Sage
Brief LinBox history

1998  Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002  v0.1 first beta release

March 2003  Introduction of FFLAS–FFPACK dense linear algebra kernel


Feb 2007  Wrapped in Sage

April 2007  creation of the public google-groups
Brief LinBox history

1998 Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002 v0.1 first beta release

March 2003 Introduction of \texttt{FFLAS-FFPACK} dense linear algebra kernel


Feb 2007 Wrapped in Sage

April 2007 creation of the public google-groups

April 2007 first patch from M Abshoff

Oct 2007 v1.1.4
Brief LinBox history

1998  Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002  v0.1 first beta release

March 2003  Introduction of FFLAS-FFPACK dense linear algebra kernel


Feb 2007  Wrapped in Sage

April 2007  creation of the public google-groups

April 2007  first patch from M Abshoff

Oct 2007  v1.1.4

2008  Towards a major change of interface (simpler) v2.0
Fast exact linear algebra, LinBox
Clément Pernet

Introduction
LinBox: an overview

Principles
Organisation of the library
Dense computations
BlackBox computations

Memory efficient implementations
In-place eliminations
Fast matrix multiplication

Linear algebra over big integers

Solutions
- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness
LinBox-1.0

Solutions
- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation
- Finite fields
  - \( \mathbb{Z} \)
LinBox-1.0

**Solutions**
- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

**Domains of computation**
- Finite fields
  - \( \mathbb{Z} \)

**Matrices**
- Sparse, structured
- Dense
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
A design for genericity

Field/Ring interface

- Shared interface with Givaro
- Wraps NTL, Givaro implementations, using archetype or envelopes
- Proper implementations, suited for dense computations
A design for genericity

Field/Ring interface

- Shared interface with Givaro
- Wraps NTL, Givaro implementations, using archetype or envelopes
- Proper implementations, suited for dense computations

Matrix interface

- Sparse, Structured, Dense: BlackBox apply
- Dense matrix interface: several levels of abstraction
Structure of the library

- **Solutions**
  - det
  - rank
  - ...

Specifying the method, domain

- **Algorithms**
  - Wiedmann
  - LU
  - ...

Specifying the component implementation

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

**Fast exact linear algebra, LinBox**

- Clément Pernet

**Introduction**

- LinBox: an overview
  - Principles
  - Organisation of the library
    - Dense computations
    - BlackBox computations
  - Memory efficient implementations
    - In-place eliminations
    - Fast matrix multiplication
  - Linear algebra over big integers
Several levels of use

- **Web servers**: http://www.linalg.org
- **Executables**: $ charpoly MyMatrix 65521
Several levels of use

- **Web servers**: [http://www.linalg.org](http://www.linalg.org)
- **Executables**: $\text{charpoly MyMatrix 65521}$
- **Call to a solution**:
  
  ```cpp
  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:**

```c++
NTL::ZZp F(65521);
Toeplitz<NTL::ZZp> A(F);
Polynomial<NTL::ZZp> P;
charpoly (P, A);
```
► **Calls to specific algorithms**
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**
- **Hack with components**
Outline

LinBox: an overview

Principles
Organisation of the library
Dense computations
BlackBox computations

Memory efficient implementations
In-place eliminations
Fast matrix multiplication

Linear algebra over big integers
Dense computations: FFLAS–FFPACK

Building block:

matrix multiplication over word-size finite field

Principle:

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

Production matriciel : BLAS vs FFLAS Opteron 2.4Ghz 4Go RAM
BLAS dgemm (dans IR)
FFLAS fgemm (dans Z/65521 Z)
Dense computations: \texttt{FFLAS–FFPACK}

Building block:

\emph{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 computations:** 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 computations:** **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 computations: **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
- Sub-cubic algorithm (Winograd)

![Graph showing performance comparison between BLAS and FFLAS](image-url)
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
Design of other dense routines

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

⇒ Block algorithm with multiple cascade structures

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1000</td>
<td>2000</td>
<td>3000</td>
<td>5000</td>
</tr>
<tr>
<td>TRSM</td>
<td>( \text{ftrsm} )</td>
<td>1.66</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td></td>
<td>( \text{dtrsm} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LQUP</td>
<td>( \text{lqup} )</td>
<td>2.00</td>
<td>1.56</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>( \text{dgetrf} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVERSE</td>
<td>( \text{inverse} )</td>
<td>1.62</td>
<td>1.32</td>
<td>1.15</td>
</tr>
<tr>
<td></td>
<td>( \text{dgetrf+dgetri} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.
**Fact**

\[ O \left( n^{\omega} \right) \text{ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.} \]

Practical algorithm:
Characteristic polynomial

**Fact**

$O(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm:

<table>
<thead>
<tr>
<th>$n$</th>
<th>LinBox</th>
<th>magma-2.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.91s</td>
<td>1.27s</td>
</tr>
<tr>
<td>5000</td>
<td>4m44s</td>
<td>15m32s</td>
</tr>
<tr>
<td>15000</td>
<td>2h20m</td>
<td>7h28m</td>
</tr>
</tbody>
</table>
Characteristic polynomial

Fact

\[ O(n^\omega) \] Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm:

<table>
<thead>
<tr>
<th>( n )</th>
<th>500</th>
<th>5000</th>
<th>15 000</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinBox</td>
<td>0.91s</td>
<td>4m44s</td>
<td>2h20m</td>
</tr>
<tr>
<td>magma-2.13</td>
<td>1.27s</td>
<td>15m32s</td>
<td>7h28m</td>
</tr>
</tbody>
</table>

- Frobenius normal form as well
- Transformation in \( O(n^\omega \log \log n) \)
Outline

LinBox: an overview
- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations
- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers
BlackBox computations

Goal: computation with very large sparse or structured matrices.
▶ No explicit representation of the matrix,
▶ Only operation: application of a vector
▶ Efficient algorithms
▶ Efficient preconditioners: Toeplitz, Hankel, Butterfly, ...
BlackBox computations

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector
BlackBox computations

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector
- Efficient algorithms
- Efficient preconditioners: Toeplitz, Hankel, Butterfly, ...
Block projection algorithms

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

⇒ Balance efficiency between BlackBox and dense computations
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
Memory efficient dense linear algebra

Structure of dense algorithms: reduction to \texttt{matmul}
Memory efficient dense linear algebra

Structure of dense algorithms: reduction to \texttt{matmul}

Approach:

1. Memory efficient reductions to \texttt{matmul} (ideally in-place)

2. Reduce extra memory requirements for \texttt{matmul}
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $O(n^3)$
Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $\mathcal{O}(n^3)$

- Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $\mathcal{O}(n^\omega)$
Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $O(n^3)$

- Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $O(n^\omega)$

- Post-Strassen, non singular:
  Bunch, Hopcroft, 74 : $A = LUP$, in $O(n^\omega)$
Triangular decompositions

► Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $O(n^3)$

► Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $O(n^\omega)$

► Post-Strassen, non singular:
  Bunch, Hopcroft, 74 : $A = LUP$, in $O(n^\omega)$

► Post-Strassen any rank profile:
  Ibarra, Moran, Hui 82 : $A = LSP$, in $O(n^\omega)$
  Ibarra, Moran, Hui 82 : $A = LQUP$, in $O(n^\omega)$
LSP-LQUP decompositions

\[ A = LSP \]
LSP-LQUP decompositions

\[ A = LSP \]

\[ A = LQUP \]
LSP-LQUP decompositions

\[ A = LSP \]

\[ A = LQUP \]
The LSP algorithm

1. Split A Row-wise

\[
\begin{bmatrix}
\text{A1} \\
\text{A2}
\end{bmatrix}
\]
The LSP algorithm

1. Split $A$ Row-wise
2. Recursive call on $A_1$
The LSP algorithm

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1} \text{ (trsm)}$
The LSP algorithm

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1} (\text{trsm})$
4. $H \leftarrow A_{22} - G \times V (\text{matmul})$
The LSP algorithm

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1} (\text{trsm})$
4. $H \leftarrow A_{22} - G \times V (\text{matmul})$
5. Recursive call on $H$
The LQUP decomposition

1. Split $A$ Row-wise

\[ A = \begin{bmatrix} A_1 & A_2 \end{bmatrix} \]
The LQUP decomposition

1. Split $A$ Row-wise
2. Recursive call on $A_1$
The LQUP decomposition

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1} (\text{trsm})$
The LQUP decomposition

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1} \text{ (trsm)}$
4. $H \leftarrow A_{22} - G \times V \text{ (matmul)}$
The LQUP decomposition

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1}$ (trsm)
4. $H \leftarrow A_{22} - G \times V$ (matmul)
5. Recursive call on $H$
The LQUP decomposition

1. Split $A$ Row-wise
2. Recursive call on $A_1$
3. $G \leftarrow A_{21} U_1^{-1}$ (trsm)
4. $H \leftarrow A_{22} - G \times V$ (matmul)
5. Recursive call on $H$
6. Row permutations
LSP-LQUP decompostions

Choice of the LQUP decomposition as a building block:

- in-place compact storage
- in-place computation
LSP-LQUP decompositions

Choice of the LQUP decomposition as a building block:

- in-place compact storage
- in-place computation
- Permutation $Q$ describes the row rank profile of $A$
LSP-LQUP decompositions

Choice of the LQUP decomposition as a building block:

- **in-place** compact storage
- **in-place** computation
- Permutation $Q$ describes the row rank profile of $A$
- Rank sensitive computation time: $O(mnr^{\omega-2})$

![Graph showing comparison between Block algorithm and Iterative algorithm](image)

$n=3000$, PIII−1.6Ghz, 512Mb RAM
Echelon forms

Row Echelon Form \[ XA = E \]

Column Echelon Form \[ AY = C \]
Echelon forms

Row Echelon Form  \[ XA = E \]

Column Echelon Form  \[ AY = C \]
Echelon forms

Row Echelon Form \[ XA = E \]

Column Echelon Form \[ AY = C \]

Property (Link with LQUP)

\[ C = LQ \begin{bmatrix} I_r \\ 0 \end{bmatrix}, \quad Y = P^T \begin{bmatrix} U_1 & U_2 \\ 0 & I_{n-r} \end{bmatrix}^{-1} \]
From LQUP to Column Echelon

Additional operations:

\[ -U^{-1}U_2 \text{ trsm } \text{(triangular system solve) in-place} \]
From LQUP to Column Echelon

Additional operations:

- $U^{-1}U_2$ \texttt{trsm} (triangular system solve) \textit{in-place}
- $U_1^{-1}$: \texttt{trtri} (triangular inverse)
From LQUP to Column Echelon

Additional operations:

\[-U^{-1}U_2 \text{ trsm (triangular system solve) in-place}\]

\[U_1^{-1}: \text{ trtri (triangular inverse)}\]

TRTRI: triangular inverse

\[
\begin{bmatrix}
U_1 & U_2 \\
U_3
\end{bmatrix}^{-1} =
\begin{bmatrix}
U_1^{-1} & -U_1^{-1}U_2U_3^{-1} \\
U_3^{-1}
\end{bmatrix}
\]

1: if $n = 1$ then
2: $U \leftarrow U^{-1}$
3: else
4: $U_2 \leftarrow U_3^{-1}U_2$ TRSM
5: $U_2 \leftarrow -U_2U_3^{-1}$ TRSM
6: $U_1 \leftarrow U_1^{-1}$ TRTRI
7: $U_3 \leftarrow U_3^{-1}$ TRTRI
8: end if
From LQUP to Column Echelon

Additional operations:

- $-U^{-1}U_2 \text{ trsm (triangular system solve) in-place}$
- $U_1^{-1}: \text{ trtri (triangular inverse) in-place}$

**TRTRI**: triangular inverse

\[
\begin{bmatrix} U_1 & U_2 \\ U_3 \end{bmatrix}^{-1} = \begin{bmatrix} U_1^{-1} & -U_1^{-1}U_2U_3^{-1} \\ & U_3^{-1} \end{bmatrix}
\]

1: if $n = 1$ then
2: $U \leftarrow U^{-1}$
3: else
4: $U_2 \leftarrow U_3^{-1}U_2$ \hspace{1cm} TRSM
5: $U_2 \leftarrow -U_2U_3^{-1}$ \hspace{1cm} TRSM
6: $U_1 \leftarrow U_1^{-1}$ \hspace{1cm} TRTRI
7: $U_3 \leftarrow U_3^{-1}$ \hspace{1cm} TRTRI
8: end if
Reduced Echelon forms

Row Reduced Echelon Form  \[XA = E\]
Reduced Echelon forms

Row Reduced Echelon Form  \[ XA = E \]

Column Reduced Echelon Form  \[ AY = C \]
From Echelon to Reduced Echelon

Again reduces to:

\[ U^{-1}X: \text{TRSM, in-place} \]

\[ U^{-1}: \text{TRTRI, in-place} \]
From Echelon to Reduced Echelon

Again reduces to:

\[ U^{-1}X: \text{TRSM, in-place} \]
\[ U^{-1}: \text{TRTRI, in-place} \]
\[ UL: \text{TRTRM,} \]
From Echelon to Reduced Echelon

Again reduces to:

\[ U^{-1} X: \text{ TRSM, in-place} \]

\[ U^{-1}: \text{ TRTRI, in-place} \]

\[ UL: \text{ TRTRM,} \]

**TRTRM: triangular triangular multiplication**

\[
\begin{bmatrix}
U_1 & U_2 \\
0 & U_3
\end{bmatrix}
\begin{bmatrix}
L_1 \\
L_2
\end{bmatrix}
= 
\begin{bmatrix}
U_1 L_1 + U_2 L_2 & U_2 L_3 \\
U_3 L_2 & U_3 L_3
\end{bmatrix}
\]

1: \( X_1 \leftarrow U_1 L_1 \)  
2: \( X_1 \leftarrow X_1 + U_2 L_2 \)  
3: \( X_2 \leftarrow U_2 L_3 \)  
4: \( X_3 \leftarrow U_3 L_2 \)  
5: \( X_4 \leftarrow U_3 L_3 \)

TRTRM  
MM  
TRTRM  
TRMM  
TRTRM
From Echelon to Reduced Echelon

Again reduces to:

\( U^{-1}X: \) TRSM, \textit{in-place}  
\( U^{-1}: \) TRTRI, \textit{in-place}  
\( UL: \) TRTRM, \textit{in-place}  

\textbf{TRTRM: triangular triangular multiplication}

\[
\begin{bmatrix}
  U_1 & U_2 \\
  U_3 &
\end{bmatrix}
\begin{bmatrix}
  L_1 \\
  L_2 & L_3
\end{bmatrix}
= 
\begin{bmatrix}
  U_1 L_1 + U_2 L_2 & U_2 L_3 \\
  U_3 L_2 & U_3 L_3
\end{bmatrix}
\]

1: \( X_1 \leftarrow U_1 L_1 \)  
2: \( X_1 \leftarrow X1 + U_2 L_2 \)  
3: \( X_2 \leftarrow U_2 L_3 \)  
4: \( X_3 \leftarrow U_3 L_2 \)  
5: \( X_4 \leftarrow U_3 L_3 \)
Example

A has full rank and generic rank profile.
Example

$A$ has full rank and generic rank profile.

$LQUP$ decomposition

$$A = LU$$
Example

$A$ has full rank and generic rank profile.

$LQUP$ decomposition \hspace{1cm} Echelon

$AU^{-1} = L$
Example

A has full rank and generic rank profile.

\[ A(U^{-1}L^{-1}) = I \]
Summary

Global scheme of reductions to LQUP decomposition. Ensures:

- in-place computations
Summary

Global scheme of reductions to LQUP decomposition. Ensures:

- in-place computations
- rank sensitive $O(r\omega^{-2}n^2)$ computation time
Global scheme of reductions to LQUP decomposition. Ensures:

- in-place computations
- rank sensitive $O(r^2 \omega^{-2} n^2)$ computation time
- increases modularity
Time complexity

These reductions are “efficient” with regard to the constant $C_3$ where $\mathcal{O}(n^3) = C_3 n^3$: 
These reductions are “efficient” with regard to the constant $C_3$ where $O(n^3) = C_3 n^3$:

<table>
<thead>
<tr>
<th></th>
<th>$L, U, P$</th>
<th>$L, S, P$</th>
<th>$L, Q, U, P$</th>
<th>Echelon</th>
<th>Red Echelon</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>2/3</td>
<td>2/3</td>
<td>2/3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
These reductions are “efficient” with regard to the constant $C_3$ where $O(n^3) = C_3 n^3$:

<table>
<thead>
<tr>
<th></th>
<th>L, U, P</th>
<th>L, S, P</th>
<th>L, Q, U, P</th>
<th>Echelon</th>
<th>Red Echelon</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>2/3</td>
<td>2/3</td>
<td>2/3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>rank profile</td>
<td>X</td>
<td>2/3</td>
<td>2/3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Echelon form</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Red Echelon</td>
<td>X</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>in place</td>
<td>V</td>
<td>X</td>
<td>V</td>
<td>V</td>
<td>X</td>
</tr>
</tbody>
</table>
These reductions are “efficient” with regard to the constant $C_3$ where $O(n^3) = C_3 n^3$:

<table>
<thead>
<tr>
<th></th>
<th>$L, U, P$</th>
<th>$L, S, P$</th>
<th>$L, Q, U, P$</th>
<th>Echelon</th>
<th>Red Echelon</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>$2/3$</td>
<td>$2/3$</td>
<td>$2/3$</td>
<td>$1$</td>
<td>$2$</td>
</tr>
<tr>
<td>rank profile</td>
<td>$X$</td>
<td>$2/3$</td>
<td>$2/3$</td>
<td>$1$</td>
<td>$2$</td>
</tr>
<tr>
<td>Echelon form</td>
<td>$X$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$X$</td>
</tr>
<tr>
<td>Red Echelon in place</td>
<td>$X$</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
</tr>
<tr>
<td></td>
<td>$V$</td>
<td>$X$</td>
<td>$V$</td>
<td>$V$</td>
<td>$X$</td>
</tr>
</tbody>
</table>
Experiments

Fast exact linear algebra, LinBox
Clément Pernet

Introduction
LinBox: an overview
Principles
Organisation of the library
Dense computations
BlackBox computations
Memory efficient implementations
In-place eliminations
Fast matrix multiplication
Linear algebra over big integers

Experiments

/test-invert 65521 A1000 1 518,996,125,000 bytes x ms

/test-redechelon 65521 A1000 1 280,663,687,500 bytes x ms
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
Strassen-Winograd algorithm

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix},
\]

**8 additions:**
\[
\begin{align*}
S_1 & \leftarrow A_{21} + A_{22} \\
S_2 & \leftarrow S_1 - A_{11} \\
S_3 & \leftarrow A_{11} - A_{21} \\
S_4 & \leftarrow A_{12} - S_2
\end{align*}
\begin{align*}
T_1 & \leftarrow B_{12} - B_{11} \\
T_2 & \leftarrow B_{22} - T_1 \\
T_3 & \leftarrow B_{22} - B_{12} \\
T_4 & \leftarrow T_2 - B_{21}
\end{align*}
\]

**7 recursive multiplications:**
\[
\begin{align*}
P_1 & \leftarrow A_{11} \times B_{11} \\
P_2 & \leftarrow A_{12} \times B_{21} \\
P_3 & \leftarrow S_4 \times B_{22} \\
P_4 & \leftarrow A_{22} \times T_4 \\
P_5 & \leftarrow S_1 \times T_1 \\
P_6 & \leftarrow S_2 \times T_2 \\
P_7 & \leftarrow S_3 \times T_3
\end{align*}
\]

**7 final additions:**
\[
\begin{align*}
U_1 & \leftarrow P_1 + P_2 \\
U_2 & \leftarrow P_1 + P_6 \\
U_3 & \leftarrow U_2 + P_7 \\
U_4 & \leftarrow U_2 + P_5 \\
U_5 & \leftarrow U_4 + P_3 \\
U_6 & \leftarrow U_3 - P_4 \\
U_7 & \leftarrow U_3 + P_5
\end{align*}
\]

**The result is the matrix:**
\[
C = \begin{bmatrix}
U_1 & U_5 \\
U_6 & U_7
\end{bmatrix}
\]
Tasks dependencies

Extra temporary blocks required
Pebble game to minimize their number

[Huss-Ledermann & Al. 96]
Tasks dependencies

- Extra temporary blocks required
- Pebble game to minimize their number
  [Huss-Ledermann & Al. 96]
Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$
- $C \leftarrow A \times B + C$
Reducing memory requirements

Dealing with 2 kind of computations:

1. $C \leftarrow A \times B$  
   - 2 temporaries $\Rightarrow 2/3n^2$

2. $C \leftarrow A \times B + C$  
   - 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].
Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ 
  2 temporaries $\Rightarrow 2/3n^2$

- $C \leftarrow A \times B + C$ 
  3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions
Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$  
  2 temporaries  $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$  
  3 temporaries  $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

- Inputs can be overwritten  
  $Cn^{2.8} + \epsilon n^{2.8}$
Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ with 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ with 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

- Inputs can be overwritten
- Add a few pre-additions
Reducing memory requirements

Dealing with 2 kind of computations:

1. $C \leftarrow A \times B$  
   2 temporaries $\Rightarrow 2/3n^2$
2. $C \leftarrow A \times B + C$  
   3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

1. Inputs can be overwritten $Cn^{2.8} + \epsilon n^{2.8}$
2. Add a few pre-additions $Cn^{2.8} + \epsilon n^{2.8}$
3. Cascading with classical algorithm $Cn^{2.8} + \epsilon n^{2.8}$

Fast exact linear algebra, LinBox
Clément Pernet

Introduction
LinBox: an overview
Principles
Organisation of the library
Dense computations
BlackBox computations
Memory efficient implementations
In-place eliminations
Fast matrix multiplication
Linear algebra over big integers
## Results

### Adding pre-additions:

<table>
<thead>
<tr>
<th>#</th>
<th>operation</th>
<th>loc.</th>
<th>#</th>
<th>operation</th>
<th>loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C_{22} = C_{22} - C_{12}$</td>
<td>$C_{22}$</td>
<td>13</td>
<td>$P_3 = S_4 B_{22} + C_{12}$</td>
<td>$C_{12}$</td>
</tr>
<tr>
<td>2</td>
<td>$C_{21} = C_{21} - C_{22}$</td>
<td>$C_{21}$</td>
<td>14</td>
<td>$P_1 = A_{11} B_{11}$</td>
<td>$X_1$</td>
</tr>
<tr>
<td>3</td>
<td>$C_{12} = C_{12} - C_{22}$</td>
<td>$C_{12}$</td>
<td>15</td>
<td>$U_2 = P_6 + P_1$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>4</td>
<td>$S_1 = A_{21} + A_{22}$</td>
<td>$X_1$</td>
<td>16</td>
<td>$P_2 = A_{12} B_{21} + C_{11}$</td>
<td>$C_{11}$</td>
</tr>
<tr>
<td>5</td>
<td>$T_1 = B_{12} - B_{11}$</td>
<td>$X_2$</td>
<td>17</td>
<td>$U_1 = P_1 + P_2$</td>
<td>$C_{11}$</td>
</tr>
<tr>
<td>6</td>
<td>$P_5 = S_1 T_1 + C_{12}$</td>
<td>$C_{12}$</td>
<td>18</td>
<td>$U_5 = U_2 + C_{12}$</td>
<td>$C_{12}$</td>
</tr>
<tr>
<td>7</td>
<td>$S_2 = S_1 - A_{11}$</td>
<td>$X_1$</td>
<td>19</td>
<td>$S_3 = A_{11} - A_{21}$</td>
<td>$X_1$</td>
</tr>
<tr>
<td>8</td>
<td>$T_2 = B_{22} - T_1$</td>
<td>$X_2$</td>
<td>20</td>
<td>$T_3 = B_{22} - B_{12}$</td>
<td>$X_2$</td>
</tr>
<tr>
<td>9</td>
<td>$P_6 = S_2 T_2 + C_{21}$</td>
<td>$C_{21}$</td>
<td>21</td>
<td>$U_3 = P_7 + U_2 = S_3 T_3 + U_2$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>10</td>
<td>$S_4 = A_{12} - S_2$</td>
<td>$X_1$</td>
<td>22</td>
<td>$U_7 = U_3 + C_{22}$</td>
<td>$C_{22}$</td>
</tr>
<tr>
<td>11</td>
<td>$T_4 = T_2 - B_{21}$</td>
<td>$X_2$</td>
<td>23</td>
<td>$U_6 = U_3 - P_4 = -A_{12} T_4 + U_3$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>12</td>
<td>$C_{22} = P_5 + C_{22}$</td>
<td>$C_{22}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $\downarrow$ | $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (3 pre-adds) |
Results

Overwriting inputs:

<table>
<thead>
<tr>
<th>#</th>
<th>operation</th>
<th>loc.</th>
<th>#ř</th>
<th>operation</th>
<th>loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C_{21} = C_{21} - C_{22}$</td>
<td>$C_{21}$</td>
<td>13</td>
<td>$P_{4} = A_{22} T_{4} + \beta C_{21}$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>2</td>
<td>$C_{22} = C_{22} - C_{12}$</td>
<td>$C_{22}$</td>
<td>14</td>
<td>$P_{2} = A_{12} B_{21} + \beta C_{11}$</td>
<td>$C_{11}$</td>
</tr>
<tr>
<td>3</td>
<td>$S_{3} = A_{11} - A_{21}$</td>
<td>$X$</td>
<td>15</td>
<td>$P_{1} = A_{11} B_{11}$</td>
<td>$B_{21}$</td>
</tr>
<tr>
<td>4</td>
<td>$T_{3} = B_{22} - B_{12}$</td>
<td>$Y$</td>
<td>16</td>
<td>$U_{1} = P_{1} + P_{2}$</td>
<td>$C_{11}$</td>
</tr>
<tr>
<td>5</td>
<td>$P_{7} = S_{3} T_{3} + \beta C_{22}$</td>
<td>$C_{22}$</td>
<td>17</td>
<td>$P_{6} = S_{2} T_{2}$</td>
<td>$A_{12}$</td>
</tr>
<tr>
<td>6</td>
<td>$S_{1} = A_{21} + A_{22}$</td>
<td>$A_{21}$</td>
<td>18</td>
<td>$U_{2} = P_{1} + P_{6}$</td>
<td>$C_{12}$</td>
</tr>
<tr>
<td>7</td>
<td>$T_{1} = B_{12} - B_{11}$</td>
<td>$B_{12}$</td>
<td>19</td>
<td>$U_{4} = U_{2} + P_{5}$</td>
<td>$C_{12}$</td>
</tr>
<tr>
<td>8</td>
<td>$S_{2} = S_{1} - A_{11}$</td>
<td>$X$</td>
<td>20</td>
<td>$U_{3} = U_{2} + P_{7}$</td>
<td>$C_{22}$</td>
</tr>
<tr>
<td>9</td>
<td>$T_{2} = B_{22} - T_{1}$</td>
<td>$Y$</td>
<td>21</td>
<td>$U_{7} = U_{3} + P_{5}$</td>
<td>$C_{22}$</td>
</tr>
<tr>
<td>10</td>
<td>$P_{5} = S_{1} T_{1} + \beta C_{12}$</td>
<td>$C_{12}$</td>
<td>22</td>
<td>$U_{6} = U_{3} - P_{4}$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>11</td>
<td>$S_{4} = A_{12} - S_{2}$</td>
<td>$A_{21}$</td>
<td>23</td>
<td>$P_{3} = S_{4} B_{22}$</td>
<td>$A_{12}$</td>
</tr>
<tr>
<td>12</td>
<td>$T_{4} = T_{2} - B_{21}$</td>
<td>$B_{12}$</td>
<td>24</td>
<td>$U_{5} = U_{4} + P_{3}$</td>
<td>$C_{12}$</td>
</tr>
</tbody>
</table>

$\rightarrow C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (2 pre-adds)
Results

Overwriting inputs:

<table>
<thead>
<tr>
<th>#</th>
<th>operation</th>
<th>loc.</th>
<th>#</th>
<th>operation</th>
<th>loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( S_3 = A_{11} - A_{21} )</td>
<td>( C_{11} )</td>
<td>12</td>
<td>( S_4 = A_{12} - S_2 )</td>
<td>( C_{22} )</td>
</tr>
<tr>
<td>2</td>
<td>( S_1 = A_{21} + A_{22} )</td>
<td>( A_{21} )</td>
<td>13</td>
<td>( P_6 = S_2 T_2 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>3</td>
<td>( T_1 = B_{12} - B_{11} )</td>
<td>( C_{22} )</td>
<td>14</td>
<td>( U_2 = P_1 + P_6 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>4</td>
<td>( T_3 = B_{22} - B_{12} )</td>
<td>( B_{12} )</td>
<td>15</td>
<td>( U_3 = U_2 + P_7 )</td>
<td>( C_{21} )</td>
</tr>
<tr>
<td>5</td>
<td>( P_7 = S_3 T_3 )</td>
<td>( C_{21} )</td>
<td>16</td>
<td>( P_3 = S_4 B_{22} )</td>
<td>( B_{11} )</td>
</tr>
<tr>
<td>6</td>
<td>( S_2 = S_1 - A_{11} )</td>
<td>( B_{12} )</td>
<td>17</td>
<td>( U_7 = U_3 + P_5 )</td>
<td>( C_{22} )</td>
</tr>
<tr>
<td>7</td>
<td>( P_1 = A_{11} B_{11} )</td>
<td>( C_{11} )</td>
<td>18</td>
<td>( U_6 = U_3 - P_4 )</td>
<td>( C_{21} )</td>
</tr>
<tr>
<td>8</td>
<td>( T_2 = B_{22} - T_1 )</td>
<td>( B_{11} )</td>
<td>19</td>
<td>( U_4 = U_2 + P_5 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>9</td>
<td>( P_5 = S_1 T_1 )</td>
<td>( A_{11} )</td>
<td>20</td>
<td>( U_5 = U_4 + P_3 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>10</td>
<td>( T_4 = T_2 - B_{21} )</td>
<td>( C_{22} )</td>
<td>21</td>
<td>( P_2 = A_{12} B_{21} )</td>
<td>( B_{11} )</td>
</tr>
<tr>
<td>11</td>
<td>( P_4 = A_{22} T_4 )</td>
<td>( A_{21} )</td>
<td>22</td>
<td>( U_1 = P_1 + P_2 )</td>
<td>( C_{11} )</td>
</tr>
</tbody>
</table>

\[ C \leftarrow A \times B \Rightarrow \text{fully in-place} \]
Results

Overwriting inputs:

<table>
<thead>
<tr>
<th>#</th>
<th>operation</th>
<th>loc.</th>
<th># operation</th>
<th>loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S_3 = A_{11} - A_{21}$</td>
<td>$C_{11}$</td>
<td>12</td>
<td>$S_4 = A_{12} - S_2$</td>
</tr>
<tr>
<td>2</td>
<td>$S_1 = A_{21} + A_{22}$</td>
<td>$A_{21}$</td>
<td>13</td>
<td>$P_6 = S_2 T_2$</td>
</tr>
<tr>
<td>3</td>
<td>$T_1 = B_{12} - B_{11}$</td>
<td>$C_{22}$</td>
<td>14</td>
<td>$U_2 = P_1 + P_6$</td>
</tr>
<tr>
<td>4</td>
<td>$T_3 = B_{22} - B_{12}$</td>
<td>$B_{12}$</td>
<td>15</td>
<td>$U_3 = U_2 + P_7$</td>
</tr>
<tr>
<td>5</td>
<td>$P_7 = S_3 T_3$</td>
<td>$C_{21}$</td>
<td>16</td>
<td>$P_3 = S_4 B_{22}$</td>
</tr>
<tr>
<td>6</td>
<td>$S_2 = S_1 - A_{11}$</td>
<td>$B_{12}$</td>
<td>17</td>
<td>$U_7 = U_3 + P_5$</td>
</tr>
<tr>
<td>7</td>
<td>$P_1 = A_{11} B_{11}$</td>
<td>$C_{11}$</td>
<td>18</td>
<td>$U_6 = U_3 - P_4$</td>
</tr>
<tr>
<td>8</td>
<td>$T_2 = B_{22} - T_1$</td>
<td>$B_{11}$</td>
<td>19</td>
<td>$U_4 = U_2 + P_5$</td>
</tr>
<tr>
<td>9</td>
<td>$P_5 = S_1 T_1$</td>
<td>$A_{11}$</td>
<td>20</td>
<td>$U_5 = U_4 + P_3$</td>
</tr>
<tr>
<td>10</td>
<td>$T_4 = T_2 - B_{21}$</td>
<td>$C_{22}$</td>
<td>21</td>
<td>$P_2 = A_{12} B_{21}$</td>
</tr>
<tr>
<td>11</td>
<td>$P_4 = A_{22} T_4$</td>
<td>$A_{21}$</td>
<td>22</td>
<td>$U_1 = P_1 + P_2$</td>
</tr>
</tbody>
</table>

$\Rightarrow$ fully in-place

Question:

Is there an in-place $O(n^{2.807})$ algorithm with constant inputs?
Results

Overwriting inputs:

<table>
<thead>
<tr>
<th>#</th>
<th>operation</th>
<th>loc.</th>
<th>#</th>
<th>operation</th>
<th>loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( S_3 = A_{11} - A_{21} )</td>
<td>( C_{11} )</td>
<td>12</td>
<td>( S_4 = A_{12} - S_2 )</td>
<td>( C_{22} )</td>
</tr>
<tr>
<td>2</td>
<td>( S_1 = A_{21} + A_{22} )</td>
<td>( A_{21} )</td>
<td>13</td>
<td>( P_6 = S_2 T_2 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>3</td>
<td>( T_1 = B_{12} - B_{11} )</td>
<td>( C_{22} )</td>
<td>14</td>
<td>( U_2 = P_1 + P_6 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>4</td>
<td>( T_3 = B_{22} - B_{12} )</td>
<td>( B_{12} )</td>
<td>15</td>
<td>( U_3 = U_2 + P_7 )</td>
<td>( C_{21} )</td>
</tr>
<tr>
<td>5</td>
<td>( P_7 = S_3 T_3 )</td>
<td>( C_{21} )</td>
<td>16</td>
<td>( P_3 = S_4 B_{22} )</td>
<td>( B_{11} )</td>
</tr>
<tr>
<td>6</td>
<td>( S_2 = S_1 - A_{11} )</td>
<td>( B_{12} )</td>
<td>17</td>
<td>( U_7 = U_3 + P_5 )</td>
<td>( C_{22} )</td>
</tr>
<tr>
<td>7</td>
<td>( P_1 = A_{11} B_{11} )</td>
<td>( C_{11} )</td>
<td>18</td>
<td>( U_6 = U_3 - P_4 )</td>
<td>( C_{21} )</td>
</tr>
<tr>
<td>8</td>
<td>( T_2 = B_{22} - T_1 )</td>
<td>( B_{11} )</td>
<td>19</td>
<td>( U_4 = U_2 + P_5 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>9</td>
<td>( P_5 = S_1 T_1 )</td>
<td>( A_{11} )</td>
<td>20</td>
<td>( U_5 = U_4 + P_3 )</td>
<td>( C_{12} )</td>
</tr>
<tr>
<td>10</td>
<td>( T_4 = T_2 - B_{21} )</td>
<td>( C_{22} )</td>
<td>21</td>
<td>( P_2 = A_{12} B_{21} )</td>
<td>( B_{11} )</td>
</tr>
<tr>
<td>11</td>
<td>( P_4 = A_{22} T_4 )</td>
<td>( A_{21} )</td>
<td>22</td>
<td>( U_1 = P_1 + P_2 )</td>
<td>( C_{11} )</td>
</tr>
</tbody>
</table>

\[ \Rightarrow C \leftarrow A \times B \Rightarrow \text{fully in-place} \]

Question:

Is there an in-place \( O(n^{2.807}) \) algorithm with constant inputs?

\[ \Rightarrow \text{yes} \]
Principle of the fully in-place algorithm

\[ \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \]

Instead of \[ n^2 \cdot 807 \] instead of \[ n^2 \cdot 807 \]
Principle of the fully in-place algorithm
Principle of the fully in-place algorithm

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
C_{11} \\
Temp
\end{bmatrix}
\]
## Principle of the fully in-place algorithm

<table>
<thead>
<tr>
<th></th>
<th>A11</th>
<th>A12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B11</th>
<th>B12</th>
</tr>
</thead>
<tbody>
<tr>
<td>B21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C11</th>
<th>C12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temp
Principle of the fully in-place algorithm

A₁₁ A₁₂
A₂₁

B₁₁ B₁₂
B₂₁ B₂₂

C₁₁ C₁₂
Temp
Principle of the fully in-place algorithm

\[
\begin{array}{cc}
A11 & A12 \\
A21 & A22 \\
\end{array}
\quad\begin{array}{cc}
B11 & B12 \\
B21 & B22 \\
\end{array}
\quad\begin{array}{cc}
C11 & C12 \\
C21 & \text{Temp} \\
\end{array}
\]
Principle of the fully in-place algorithm

\[\begin{array}{ccc}
A11 & A12 & A21 & A22 \\
B11 & B12 & B21 & B22 \\
C11 & C12 & C21 & C22 \\
\end{array}\]
Principle of the fully in-place algorithm

\[ 7.2n^{2.807} \text{ instead of } 6n^{2.807} \]
Outline

LinBox: an overview
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Memory efficient implementations
  In-place eliminations
  Fast matrix multiplication

Linear algebra over big integers
The problem

- Reasonably small dimension \( n = 2..100 \)
- Unreasonably large entries
  \( (\log_2 \|A\|_\infty = 1,000,000..1,000,000,000) \)
The problem

- Reasonably small dimension ($n = 2..100$)
- Unreasonably large entries
  \[ \log_2 \|A\|_{\infty} = 1,000,000..1,000,000,000 \]

\[ \text{mul} \gg \text{add} \]

despite FFT
The problem

- Reasonably small dimension ($n = 2..100$)
- Unreasonably large entries
  \[ \log_2 \| A \|_\infty = 1,000,000..1,000,000,000 \]

\[ \text{mul} \gg \text{add} \]

despite FFT

- Fast Matrix Multiplication is always better than classic,
- Can do better than Strassen-Winograd
Dealing with odd dimensions

Padding: add 0 columns and rows to the nearest power of 2 (more operations)

Peeling: slice down to the nearest power of 2, and use classical block algorithm (less “sub-cubic”).
Dealing with odd dimensions

**Padding:** add 0 columns and rows to the nearest power of 2 (more operations)

**Peeling:** slice down to the nearest power of 2, and use classical block algorithm (less “sub-cubic”).

**Static:** Before actual computation.

**Dynamic:** At each recursive level, dimension 1 modifications.
Dealing with odd dimensions

**Padding:** add 0 columns and rows to the nearest power of 2 (more operations)

**Peeling:** slice down to the nearest power of 2, and use classical block algorithm (less “sub-cubic”).

- **Static:** Before actual computation.
- **Dynamic:** At each recursive level, dimension 1 modifications.

**Virtual dynamic padding:**
- Recursive splitting with odd dimensions
- No extra operations (virtual 0)
- Better operation count than peeling
Winograd 68

Formula for dot-product:

\[ a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2 \]
Winograd 68

Formula for dot-product:

\[ a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2 \]

1: \textbf{for} \ k=1..n/2 \ \textbf{do}
2: \hspace{1em} \textbf{for} \ i=1..n \ \textbf{do}
3: \hspace{2em} \alpha_{i,k} = (a_{i,2k} a_{i,2k+1})
4: \hspace{1em} \textbf{end for}
5: \textbf{for} \ j=1..n \ \textbf{do}
6: \hspace{1em} \beta_{k,j} = (b_{2k,j} b_{2k+1,j})
7: \hspace{1em} \textbf{end for}
8: \textbf{for} \ i=1..n \ \textbf{do}
9: \hspace{1em} \textbf{for} \ j=1..n \ \textbf{do}
10: \hspace{2em} C_{i,j}++ = (a_{i,2k} + b_{2k+1,j})(a_{i,2k+1} + b_{2k,j}) - \alpha_{i,k} - \beta_{k,j}
11: \hspace{1em} \textbf{end for}
12: \textbf{end for}
13: \textbf{end for}
Winograd 68

Formula for dot-product:

\[ a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2 \]

1: for k=1..n/2 do
2: for i=1..n do
3: \( \alpha_{i,k} = (a_{i,2k} a_{i,2k+1}) \)
4: end for
5: for j=1..n do
6: \( \beta_{k,j} = (b_{2k,j} b_{2k+1,j}) \)
7: end for
8: for i=1..n do
9: for j=1..n do
10: \( C_{i,j} = (a_{i,2k} + b_{2k+1,j})(a_{i,2k+1} + b_{2k,j}) - \alpha_{i,k} - \beta_{k,j} \)
11: end for
12: end for
13: end for

- Requires commutativity (no recursive algorithm)
- Still \( O(n^3) \)
- But better constant: \( T_2(n) = 1/2 n^3 + n^2 \) instead of \( 1n^3 \)
From 2 to 3

\[ a_1 b_1 + a_2 b_2 + a_3 b_3 = (a_1 + a_2 + b_3)(a_3 + b_1 + b_2) \]

\[ T_3(n) = 1/3n^3 \]
From 2 to 3

\[ a_1 b_1 + a_2 b_2 + a_3 b_3 = (a_1 + a_2 + b_3)(a_3 + b_1 + b_2) - (a_1 + a_2)a_3 - b_3(b_1 + b_2) \]

\[ T_3(n) = \frac{1}{3}n^3 + \frac{2}{3}n^2 \]
From 2 to 3

\[ a_1 b_1 + a_2 b_2 + a_3 b_3 = (a_1 + a_2 + b_3)(a_3 + b_1 + b_2) - (a_1 + a_2)a_3 - b_3(b_1 + b_2) - a_1 b_2 - a_2 b_1 \]

\[ T_3(n) = \frac{1}{3}n^3 + T(n, \frac{2}{3}n) + \frac{2}{3}n^2 \]
Comparison

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>216</td>
<td>512</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>7</td>
<td>26</td>
<td>49</td>
<td>182</td>
<td>343</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>729</td>
<td>1000</td>
<td>1728</td>
<td>3375</td>
<td>5832</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>560</td>
<td>770</td>
<td>1274</td>
<td>2794</td>
<td>3920</td>
</tr>
</tbody>
</table>
### Comparison

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>216</td>
<td>512</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>7</td>
<td>26</td>
<td>49</td>
<td>182</td>
<td>343</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>7</td>
<td>25</td>
<td>49</td>
<td>175</td>
<td>343</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>729</td>
<td>1000</td>
<td>1728</td>
<td>3375</td>
<td>5832</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>560</td>
<td>770</td>
<td>1274</td>
<td>2794</td>
<td>3920</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>763</td>
<td>1225</td>
<td>2161</td>
<td>3401</td>
<td></td>
</tr>
</tbody>
</table>
Comparison

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>216</td>
<td>512</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>7</td>
<td>26</td>
<td>49</td>
<td>182</td>
<td>343</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>7</td>
<td>25</td>
<td>49</td>
<td>175</td>
<td>343</td>
</tr>
<tr>
<td>Winograd 68</td>
<td>8</td>
<td></td>
<td>48</td>
<td>144</td>
<td>320</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>729</td>
<td>1000</td>
<td>1728</td>
<td>3375</td>
<td>5832</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>560</td>
<td>770</td>
<td>1274</td>
<td>2794</td>
<td>3920</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>559</td>
<td>763</td>
<td>1225</td>
<td>2161</td>
<td>3401</td>
</tr>
<tr>
<td>Winograd 68</td>
<td></td>
<td>600</td>
<td>1008</td>
<td></td>
<td>3240</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>216</td>
<td>512</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>7</td>
<td>26</td>
<td>49</td>
<td>182</td>
<td>343</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>7</td>
<td>25</td>
<td>49</td>
<td>175</td>
<td>343</td>
</tr>
<tr>
<td>Winograd 68</td>
<td>8</td>
<td>48</td>
<td>144</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>New algorithm</td>
<td></td>
<td>24</td>
<td></td>
<td>158</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>9</th>
<th>10</th>
<th>12</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical algorithm</td>
<td>729</td>
<td>1000</td>
<td>1728</td>
<td>3375</td>
<td>5832</td>
</tr>
<tr>
<td>Strassen+Peeling</td>
<td>560</td>
<td>770</td>
<td>1274</td>
<td>2794</td>
<td>3920</td>
</tr>
<tr>
<td>Strassen+virt. padding</td>
<td>559</td>
<td>763</td>
<td>1225</td>
<td>2161</td>
<td>3401</td>
</tr>
<tr>
<td>Winograd 68</td>
<td></td>
<td>600</td>
<td>1008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New algorithm</td>
<td>489</td>
<td></td>
<td>1088</td>
<td>2093</td>
<td>3456</td>
</tr>
</tbody>
</table>
Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $\mathcal{O}(n^{2.779})$
Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $O(n^{2.779})$
- build a database for small dimensions,
Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $O(n^{2.779})$
- build a database for small dimensions,
- automatically generate a combination of base case algorithms for a given dimension