Parallel Perspectives for the LinBox library

Clément PERNET

Symbolic Computation Group
University of Waterloo

January 29, 2007
Outline

Introduction

The LinBox library
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Parallelism perspectives
  Design considerations
  Algorithmic perspectives

Conclusion
Exact linear algebra

Building block in exact computation:

- Cryptography : sparse system resolution
- Representation theory : null space
- Topology : Smith form
- Graph theory : characteristic polynomial
- ...

Parallel Perspectives for the LinBox library

Clément PERNET

Introduction

The LinBox library

- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Parallelism perspectives

- Design considerations
- Algorithmic perspectives

Conclusion
## Software solutions for exact computations

<table>
<thead>
<tr>
<th>Libraries</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>finite fields</strong></td>
<td>NTL, GMP, Lidia, Pari, ...</td>
</tr>
<tr>
<td><strong>polynomials</strong></td>
<td>NTL, ...</td>
</tr>
<tr>
<td><strong>integers</strong></td>
<td>GMP, ...</td>
</tr>
</tbody>
</table>
Software solutions for exact computations

<table>
<thead>
<tr>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite fields: NTL, GMP, Lidia, Pari, ...</td>
</tr>
<tr>
<td>polynomials: NTL, ...</td>
</tr>
<tr>
<td>integers: GMP, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Global solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Maple</td>
</tr>
<tr>
<td>▶ Magma</td>
</tr>
</tbody>
</table>

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion
## Software solutions for exact computations

### Libraries

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite fields</td>
<td>NTL, GMP, Lidia, Pari, ...</td>
</tr>
<tr>
<td>polynomials</td>
<td>NTL, ...</td>
</tr>
<tr>
<td>integers</td>
<td>GMP, ...</td>
</tr>
</tbody>
</table>

### Global solutions

- Maple
- Magma
- Sage
Software solutions for exact computations

Libraries

<table>
<thead>
<tr>
<th>finite fields</th>
<th>NTL, GMP, Lidia, Pari, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynomials</td>
<td>NTL, ...</td>
</tr>
<tr>
<td>integers</td>
<td>GMP, ...</td>
</tr>
</tbody>
</table>

Global solutions

- Maple
- Magma
- Sage

Linear Algebra ?
Outline

Introduction

The LinBox library
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Parallelism perspectives
  Design considerations
  Algorithmic perspectives

Conclusion
Outline

Introduction

The LinBox library
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Parallelism perspectives
  Design considerations
  Algorithmic perspectives

Conclusion
LinBox

A generic middleware

Maple → LinBox → GAP → SAGE

LinBox

Finite fields
- NTL
- Givaro
- ...

BLAS
- ATLAS
- GOTO
- ...

GMP
The LinBox project, facts

Joint NFS-NSERC-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 NFS-NSERC-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
Parallel Perspectives for the LinBox library

Clément PERNET

Introduction

The LinBox library

Principles

Organisation of the library
Dense computations
BlackBox computations

Parallelism perspectives
Design considerations
Algorithmic perspectives

Conclusion

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness
Parallel Perspectives for the LinBox library

LinBox-1.0

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion

Solutions

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

Domains of computation

- Finite fields
  - $\mathbb{Z}$, $\mathbb{Q}$
Parallel Perspectives for the LinBox library

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion

Solutions

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

Domains of computation

- Finite fields
  - \( \mathbb{Z}, \mathbb{Q} \)

Matrices

- Sparse, structured
- Dense

LinBox-1.0

Solutions

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

Domains of computation

- Finite fields
  - \( \mathbb{Z}, \mathbb{Q} \)

Matrices

- Sparse, structured
- Dense
A design for genericity

Field/Ring interface

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

Matrix interface

- Sparse, 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
- ...

Parallel Perspectives for the LinBox library

Clément PERNET

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion
Several levels of use

▶ **Web servers:** [http://www.linalg.org](http://www.linalg.org)
Several levels of use

- **Web servers:** [http://www.linalg.org](http://www.linalg.org)
- **Executables:** ```sh $ 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**
**Dense computations**

Building block:

\[ \text{matrix multiplication over word-size finite field} \]

Principle:

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

---

**Parallel Perspectives for the LinBox library**

Clément PERNET

Introduction

The LinBox library

- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Parallelism perspectives

- Design considerations
- Algorithmic perspectives

Conclusion
Dense computations

Building block:

*matrix multiplication over word-size finite field*

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- BLAS cache management
Dense computations

Building block:

*matrix multiplication over word-size finite field*

Principle:

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

![Graph showing performance comparison]

Multiplication classique dans $\mathbb{Z}/65521\mathbb{Z}$ sur un P4, 3.4 GHz

<table>
<thead>
<tr>
<th>Method</th>
<th>Mfops</th>
</tr>
</thead>
<tbody>
<tr>
<td>fgemm Standard</td>
<td>Standard</td>
</tr>
<tr>
<td>Classical</td>
<td>600</td>
</tr>
<tr>
<td>Winograd 1 niveau</td>
<td>800</td>
</tr>
<tr>
<td>Winograd 2 niveaux</td>
<td>1000</td>
</tr>
<tr>
<td>Winograd 3 niveaux</td>
<td>1200</td>
</tr>
<tr>
<td>Winograd 4 niveaux</td>
<td>1400</td>
</tr>
<tr>
<td>Winograd 5 niveaux</td>
<td>1600</td>
</tr>
</tbody>
</table>

Parallel Perspectives for the LinBox library

Clément PERNET

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion
Parallel Perspectives for the LinBox library

Clément PERNET

Introduction

The LinBox library

Principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion

Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- BLAS cache management
Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- BLAS cache management
- Sub-cubic algorithm (Winograd)
Design of other dense routines

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

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

⇒ Block algorithm with multiple cascade

\[ \begin{array}{c}
\begin{array}{c}
X_{1,i-1} \\
X_{1}
\end{array}
\end{array}
\begin{array}{c}
= \\
\end{array}
\begin{array}{c}
V_{1} \\
B_{1,i-1} \\
B_{1}
\end{array}\begin{array}{c}
-1
\end{array} \]
Design of other dense routines

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

⇒ Block algorithm with multiple cascade
Characteristic polynomial

Fact

$O(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.
## 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>magma-2.11</th>
<th>LU-Krylov</th>
<th>New algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.010s</td>
<td>0.005s</td>
<td>0.006s</td>
</tr>
<tr>
<td>300</td>
<td>0.830s</td>
<td>0.294s</td>
<td>0.105s</td>
</tr>
<tr>
<td>500</td>
<td>3.810s</td>
<td>1.316s</td>
<td>0.387s</td>
</tr>
<tr>
<td>1000</td>
<td>29.96s</td>
<td>10.21s</td>
<td>2.755s</td>
</tr>
<tr>
<td>3000</td>
<td>802.0s</td>
<td>258.4s</td>
<td>61.09s</td>
</tr>
<tr>
<td>5000</td>
<td>3793s</td>
<td>1177s</td>
<td>273.4s</td>
</tr>
<tr>
<td>7500</td>
<td>MT</td>
<td>4209s</td>
<td>991.4s</td>
</tr>
<tr>
<td>10 000</td>
<td>MT</td>
<td>8847s</td>
<td>2080s</td>
</tr>
</tbody>
</table>

Computation time for 1 Frobenius block matrices, on a Athlon 2200, 1.8Ghz, 2Gb

MT: Memory thrashing
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

Introduction

The LinBox library
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Parallelism perspectives
  Design considerations
  Algorithmic perspectives

Conclusion
Data Containers/Iterators

Distinction between computation and access to the data:

Example

Iterates \((u^T A^i v)_{i=1..k}\) used for system resolution can be

- precomputed and stored
- computed on the fly
- computed in parallel
Data Containers/Iterators

Distinction between computation and access to the data:

**Example**

Iterates \((u^T A^i v)_{i=1..k}\) used for system resolution can be
- precomputed and stored
- computed on the fly
- computed in parallel

**Solution:** solver defined using generic iterators, independently from the method to compute the data.
Example: A parallel data flow iterator

```cpp
const iterator& iterator::operator++() {
    if (++current > launched) {
        ... 
        for (int i=0; i<n; ++i)
            Fork<launch>(i,...);
        launched += n;
    }
    return *this;
}

const value_type& iterator::operator*() {
    return _d[current].read();
}
```
Existing containers/iterators

- Scalar projections: 
  \[ (v^T A^i u)_{i=1..k} \]
  \[ \Rightarrow \text{Wiedemann’s algorithm} \]
Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  ⇒ Wiedemann’s algorithm

- Block projections: $(A v_i)_{i=1..k}$
  ⇒ Block Wiedemann algorithm
Existing containers/iterators

- Scalar projections: $(v^TA^i u)_{i=1..k}$ ⇒ Wiedemann’s algorithm
- Block projections: $(Av_i)_{i=1..k}$ ⇒ Block Wiedemann algorithm
- Modular homomorphic imaging: $(\text{Algorithm}(A \mod p_i))_{i=1..k}$ ⇒ Chinese Remainder Algorithm
Existing containers/iterators

- Scalar projections:
  ⇒ Wiedemann’s algorithm
  \[(v^T A^i u)_{i=1..k}\]

- Block projections:
  ⇒ Block Wiedemann algorithm
  \[(A v_i)_{i=1..k}\]

- Modular homomorphic imaging:
  ⇒ Chinese Remainder Algorithm
  \[(\text{Algorithm}(A \mod p_i))_{i=1..k}\]

⇒ no modifications to the high level algorithms for the parallelization.
Parallelization tools

Until now, few parallelization:

- attempts with MPI, and POSIX threads
- Higher level systems: Athapascan-1, KAAPI
  - Full design compatibility
  - Provides efficient schedulers; work stealing abilities
Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm:
  
  \[ [u_1, .., u_k]^T (GG^T) u_i, i = 1..k \]

  \[ \Rightarrow \text{Only } \frac{\text{rank}(G)}{k} \text{ iterations} \]

- combined with sigma basis algorithm.
Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm:
  \[ [u_1, \ldots, u_k]^T (GG^T) u_i, \; i = 1..k \]
  \[ \Rightarrow \text{Only } \frac{\text{rank}(G)}{k} \text{ iterations} \]
- combined with sigma basis algorithm.

<table>
<thead>
<tr>
<th>matrix</th>
<th>n</th>
<th>m</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d17</td>
<td>1,548,650</td>
<td>955,128</td>
<td>626,910</td>
</tr>
<tr>
<td>GL7d20</td>
<td>1,437,547</td>
<td>1,911,130</td>
<td>877,562</td>
</tr>
<tr>
<td>GL7d21</td>
<td>822,922</td>
<td>1,437,547</td>
<td>559,985</td>
</tr>
</tbody>
</table>

Timings estimations [in days]

<table>
<thead>
<tr>
<th>matrix</th>
<th>(T_{iter}) [min]</th>
<th>(T_{seq})</th>
<th>(T_{par}(50))</th>
<th>(T_{par}(50, ET))</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d17</td>
<td>0.46875</td>
<td>621.8</td>
<td>12.4</td>
<td>8.16</td>
</tr>
<tr>
<td>GL7d20</td>
<td>0.68182</td>
<td>1361.31</td>
<td>27.2272</td>
<td>16.6214</td>
</tr>
<tr>
<td>GL7d21</td>
<td>0.35714</td>
<td>408.196</td>
<td>8.1644</td>
<td>5.5559</td>
</tr>
</tbody>
</table>
Example: rank computations

[Dumas & urbanska]

- parallel block Wiedemann algorithm:
  \[ \begin{bmatrix} u_1, \ldots, u_k \end{bmatrix}^T (GG^T) \begin{bmatrix} u_i \end{bmatrix}, i = 1..k \]
  \( \Rightarrow \) Only \( \frac{\text{rank}(G)}{k} \) iterations
- combined with sigma basis algorithm.

<table>
<thead>
<tr>
<th>matrix</th>
<th>n</th>
<th>m</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d17</td>
<td>1,548,650</td>
<td>955,128</td>
<td>626,910</td>
</tr>
<tr>
<td>GL7d20</td>
<td>1,437,547</td>
<td>1,911,130</td>
<td>877,562</td>
</tr>
<tr>
<td>GL7d21</td>
<td>822,922</td>
<td>1,437,547</td>
<td>559,985</td>
</tr>
</tbody>
</table>

Timings estimations [in days]

<table>
<thead>
<tr>
<th>matrix</th>
<th>( T_{\text{iter}} ) [min]</th>
<th>( T_{\text{seq}} )</th>
<th>( T_{\text{par}} ) (50)</th>
<th>( T_{\text{par}} ) (50, ET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d17</td>
<td>0.46875</td>
<td>621.8</td>
<td>12.4</td>
<td>8.16</td>
</tr>
<tr>
<td>GL7d20</td>
<td>0.68182</td>
<td>1361.31</td>
<td>27.2272</td>
<td>16.6214</td>
</tr>
<tr>
<td>GL7d21</td>
<td>0.35714</td>
<td>408.196</td>
<td>8.1644</td>
<td>5.5559</td>
</tr>
</tbody>
</table>
Outline

Introduction

The LinBox library

Parallelism perspectives

Conclusion
TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

- divide both rows and columns
  ⇒ Better memory management
  ⇒ Enables to use recursive data structures
TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

- divide both rows and columns
  - Better memory management
  - Enables to use recursive data structures

TURBO vs LQUP for rank computations over $\mathbb{Z}/101\mathbb{Z}$ on a P4−2.4Ghz−512Mo

- (1) TURBO with Givaro−ZpZ
- (2) LQUP with Givaro−ZpZ
The LinBox library

Introduction

The principles

Organisation of the library

Dense computations

BlackBox computations

Parallelism perspectives

Design considerations

Algorithmic perspectives

Conclusion

TURBO triangular elimination

[Roch & Dumas 02]: recursive block algorithm for triangularization

- divide both rows and columns
  - Better memory management
  - Enables to use recursive data structures
- 5 recursive calls (U, V, C, D, Z), including 2 being parallel (C, D)

TURBO vs LQUP for rank computations over \( \mathbb{Z}/101\mathbb{Z} \) on a P4−2.4Ghz−512Mo

(1) TURBO with Givaro−ZpZ
(2) LQUP with Givaro−ZpZ

\[ \begin{aligned}
\text{TURBO vs LQUP for rank computations over } & \mathbb{Z}/101\mathbb{Z} \text{ on a P4−2.4Ghz−512Mo} \\
\text{(1) TURBO with Givaro−ZpZ} & \quad \text{(2) LQUP with Givaro−ZpZ}
\end{aligned} \]
Principle of Workstealing

[Arora, Blumofe, Plaxton01], [Acar, Blelloche, Blumofe02]

- 2 algorithms to complete a task $f$: $f_{\text{seq}}$ and $f_{\text{par}}$
- When a processor becomes idle, ExtractPar steals the work to $f_{\text{seq}}$.

\[ \text{ExtractPar} \]
Application to multiple triangular system solving

\[
\text{TRSM : Compute } \begin{bmatrix} U_1 & U_2 \\ U_3 & \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}
\]

\[
X_2 = \text{TRSM}(U_3, B_2)
\]
\[
B_1 = B_1 - U_2 X_2
\]
\[
X_1 = \text{TRSM}(U_1, B_1)
\]
Application to multiple triangular system solving

\[
\begin{align*}
\text{TRSM} & : \text{Compute } \begin{bmatrix} U_1 & U_2 \\ U_3 & \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \\
X_2 & = \text{TRSM}(U_3, B_2) \\
B_1 & = B_1 - U_2 X_2 \\
X_1 & = \text{TRSM}(U_1, B_1)
\end{align*}
\]

\[f_{\text{seq}}\]

\[
\text{TRSM}(U, B) \Rightarrow T_1 = n^3, T_\infty = O(n)
\]
Application to multiple triangular system solving

TRSM : Compute \[
\begin{bmatrix}
U_1 & U_2 \\
U_3 & U_3
\end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}
\]

- \(X_2 = \text{TRSM}(U_3, B_2)\)
- \(B_1 = B_1 - U_2 X_2\)
- \(X_1 = \text{TRSM}(U_1, B_1)\)

\(\mathbf{f}_{\text{seq}}\) TRSM \((U, B)\)

\[T_1 = n^3, \quad T_\infty = \mathcal{O}(n)\]

\(\mathbf{f}_{\text{par}}\)

Compute \(V = U^{-1}\);

\(\text{TRMM}(V, B)\);

\[T_1 = \frac{4}{3} n^3, \quad T_\infty = \mathcal{O}(\log n)\]
Application to multiple triangular system solving

When sequential \( \text{TRSM} \) and parallel \( \text{Inverse} \) join: Compute \( X_1 = A_1^{-1} B_1 \) in parallel (\( \text{TRMM} \)).
Multi-adic lifting

Solving $Ax = b$ over $\mathbb{Z}$

Standard $p$-adic Lifting [Dixon82]

Compute $A^{-1} \mod p$

$r = b$

for $i = 0..n$ do

$x_i = A^{-1}r \mod p$

$r = (r - Ax_i)/p$

end for

$z = x_0 + px_1 + p^2x_2 + \cdots + x_np^n$

$x = \text{RatReconst}(z)$
Multi-adic lifting

Solving $Ax = b$ over $\mathbb{Z}$

multi-adic lifting:

$$
\text{for all } j=1..k \text{ do }
\begin{align*}
&\text{Compute } A^{-1}\mod p_j \\
&r = b \\
&\text{for } i = 0..n/k \text{ do } \\
&\quad x_i = A^{-1}r \mod p_j \\
&\quad r = (r - Ax_i)/p_j \\
&\text{end for} \\
&z_j = x_0 + p_j x_1 + \cdots + p_j^{\lfloor n/k \rfloor} x_{n/k} \\
\text{end for} \\
z = \text{ChineseRemainderAlg}((z_j, p_j^{\lfloor n/k \rfloor})_{j=1..k}) \\
x = \text{RatReconst}(z)
\end{align*}
$$
Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3
Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3
- Divides a sequential loop into several parallel tasks
Multi-adic lifting

- Used in sequential computation [Chen & Storjohann 05], to balance efficiency between BLAS level 2 and 3
- Divides a sequential loop into several parallel tasks
- Work stealing perspectives...
Outline

Introduction

The LinBox library
  Principles
  Organisation of the library
  Dense computations
  BlackBox computations

Parallelism perspectives
  Design considerations
  Algorithmic perspectives

Conclusion
Conclusion

Large grain parallelism:
- Chinese remaindering
- Multi-adic lifting
- Block Wiedemann
Conclusion

Large grain parallelism:
- Chinese remaindering
- Multi-adic lifting
- Block Wiedemann

Fine grain adaptive parallelism:
⇒ Work stealing
Conclusion

Large grain parallelism:
- Chinese remaindering
- Multi-adic lifting
- Block Wiedemann

Fine grain adaptive parallelism:
⇒ Work stealing

Perspectives
- Development of simple parallel containers
- Parallel distribution of LinBox, based on Kaapi