Parallel Perspectives for the LinBox library

Clément PERNET

Symbolic Computation Group
University of Waterloo

January 29, 2007

Outline

Exact linear algebra

Building block in exact computation:

Cryptography : sparse system resolution

Representation theory : null space

Topology : Smith form

Graph theory : characteristic polynomial

...

Software solutions for exact computations

Libraries

finite fields: NTL, GMP, Lidia, Pari, ...

polynomials: NTL, ...

integers: GMP, ...

Software solutions for exact computations

Libraries

finite fields: NTL, GMP, Lidia, Pari, ...

polynomials: NTL, ...

integers: GMP, ...

Global solutions

- Maple
- Magma

Software solutions for exact computations

Libraries

finite fields: NTL, GMP, Lidia, Pari, ...

polynomials: NTL, ...

integers: GMP, ...

Global solutions

- Maple
- Magma
- Sage

Software solutions for exact computations

Libraries

finite fields: NTL, GMP, Lidia, Pari, ...

polynomials: NTL, ...

integers: GMP, ...

Global solutions

- Maple
- Magma
- Sage

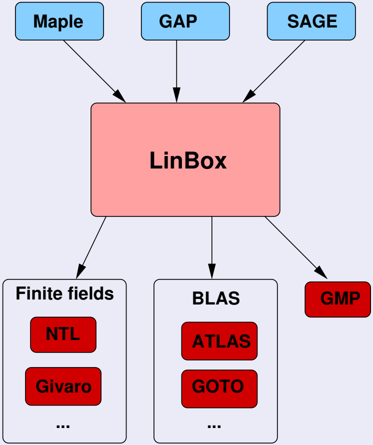
Linear Algebra ?

Outline

Outline

LinBox

A generic middleware



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 developpers

LinBox-1.0

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
- $\mathbb{Z}, \mathbb{Q}$

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

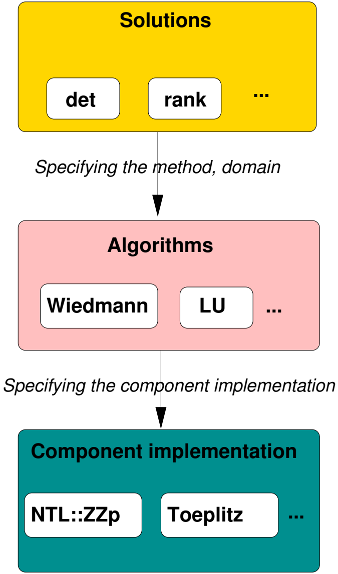
Field/Ring interface

- 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



Several levels of use

- Web servers: `http://www.linalg.org`

Several levels of use

- Web servers: `http://www.linalg.org`
- Executables: `$ charpoly MyMatrix 65521`

Several levels of use

- Web servers: `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`
- Executables: `$ charpoly MyMatrix 65521`
- Call to a solution:

```
NTL::ZZp F(65521);
Toeplitz<NTL::ZZp> A(F);
Polynomial<NTL::ZZp> P;
charpoly (P, A);
```

- Calls to specific algorithms

Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

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

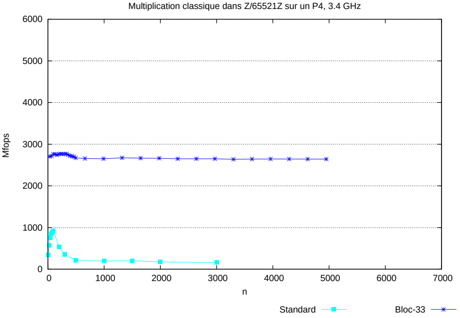
Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

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



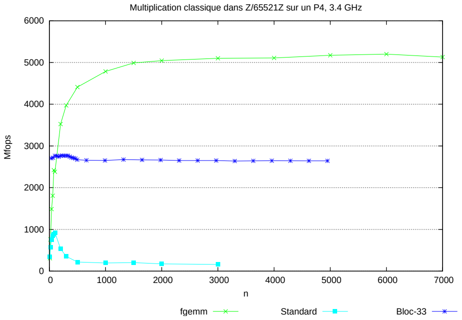
Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

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



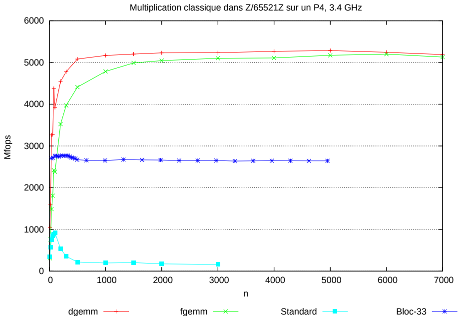
Dense computations

Building block:

matrix multiplication over word-size finite field

Principle:

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



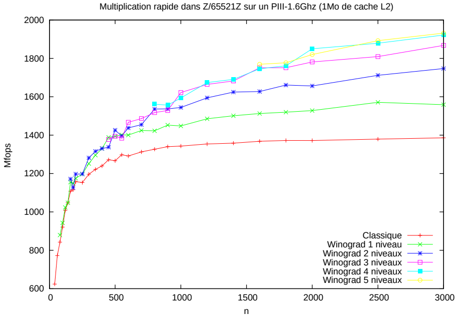
Dense computations

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)



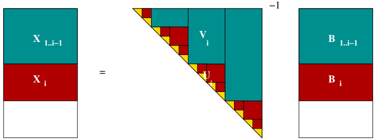
Design of other dense routines

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

Design of other dense routines

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

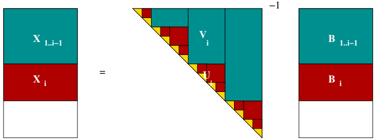
⇒Block algorithm with multiple cascade



Design of other dense routines

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

⇒Block algorithm with multiple cascade



	n	*1000*	*2000*	*3000*	*5000*	*10 000*
TRSM	$\frac{ftrsm}{dtrsm}$	1,66	1,33	1,24	1,12	1,01
LQUP	$\frac{lqup}{dgetrf}$	2,00	1,56	1,43	1,18	1,07
INVERSE	$\frac{inverse}{dgetrf+dgetri}$	1.62	1.32	1.15	0.86	0.76

Characteristic polynomial

Fact

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

Characteristic polynomial

Fact

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

Practical algorithm :

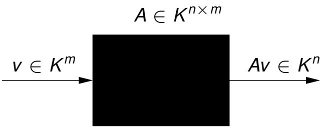
n	magma-2.11	LU-Krylov	New algorithm
100	0.010s	0.005s	0.006s
300	0.830s	0.294s	0.105s
500	3.810s	1.316s	0.387s
1000	29.96s	10.21s	2.755s
3000	802.0s	258.4s	61.09s
5000	3793s	1177s	273.4s
7500	MT	4209s	991.4s
10 000	MT	8847s	2080s

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

MT: Memory thrashing

BlackBox computations

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 preconditionners: 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 $i = 1..2d/k$

$\Rightarrow$Balance efficiency between BlackBox and dense compations

Outline

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

```
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: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$ Wiedemann's algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  $\Rightarrow$Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  $\Rightarrow$Block Wiedemann algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  ⇒Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  ⇒Block Wiedemann algorithm
- Modular homomorphic imaging:
  $$(\texttt{Algorithm}(A \mod p_i))_{i=1..k}$$
  ⇒Chinese Remainder Algorithm

Existing containers/iterators

- Scalar projections: $(v^T A^i u)_{i=1..k}$
  ⇒Wiedemann's algorithm
- Block projections: $(Av_i)_{i=1..k}$
  ⇒Block Wiedemann algorithm
- Modular homomorphic imaging:
  $(\texttt{Algorithm}(A \mod p_i))_{i=1..k}$
  ⇒Chinese Remainder Algorithm

⇒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
  $\Rightarrow$Full design compatibility
  $\Rightarrow$Provides efficient schedulers; work stealing abilities

Example: rank computations

[Dumas & urbanska]

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

Example: rank computations

[Dumas & urbanska]

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

matrix	n	m	rank
GL7d17	1,548,650	955,128	626,910
GL7d20	1,437,547	1,911,130	877,562
GL7d21	822,922	1,437,547	559,985

Example: rank computations

[Dumas & urbanska]

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

matrix	n	m	rank
GL7d17	1,548,650	955,128	626,910
GL7d20	1,437,547	1,911,130	877,562
GL7d21	822,922	1,437,547	559,985

Timings estimations [in days]

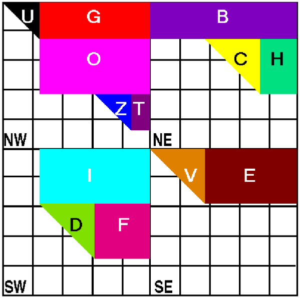
matrix	T_{iter} [min]	T_{seq}	T_{par} (50)	T_{par} (50, ET)
GL7d17	0.46875	621.8	12.4	8.16
GL7d20	0.68182	1361.31	27.2272	16.6214
GL7d21	0.35714	408.196	8.1644	5.5559

Outline

TURBO triangular elimination

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

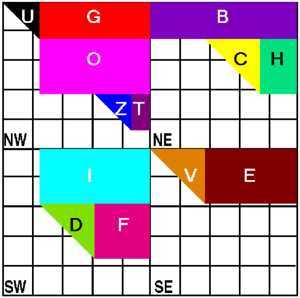
- divide both rows and columns
  - ⇒Better memory management
  - ⇒Enables to use recursive data structures

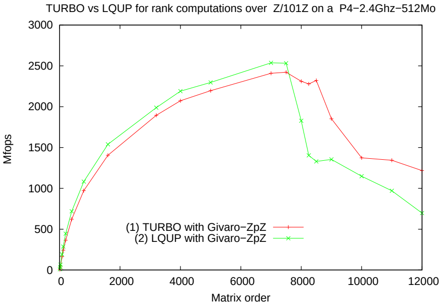


TURBO triangular elimination

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

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

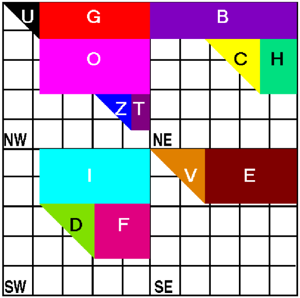


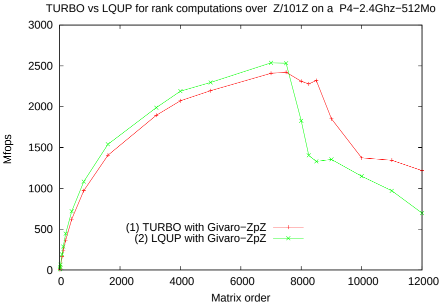


TURBO triangular elimination

[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)

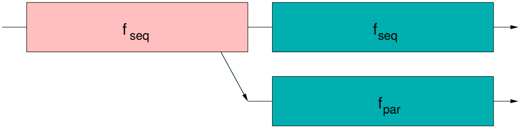




Principle of Workstealing

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

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



Application to mutiple triangular system solving

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

$$\begin{aligned} X_2 &= \texttt{TRSM}(U_3, B_2) \\ B_1 &= B_1 - U_2 X_2 \\ X_1 &= \texttt{TRSM}(U_1, B_1) \end{aligned}$$

Application to mutiple triangular system solving

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

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

f_{seq}

$\texttt{TRSM}(U, B)$
$\Rightarrow T_1 = n^3, \ T_\infty = \mathcal{O}(n)$

Application to mutiple triangular system solving

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

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

f_{seq}

$\texttt{TRSM}(U, B)$

$\Rightarrow T_1 = n^3,\ T_\infty = \mathcal{O}(n)$

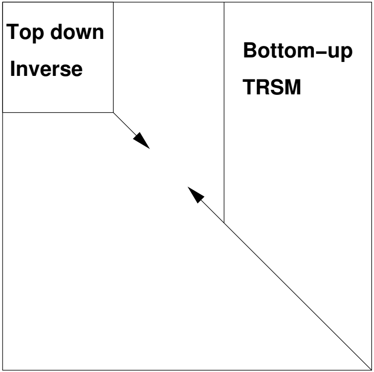
f_{par}

Compute $V = U^{-1}$;

$\texttt{TRMM}(V, B)$;

$\Rightarrow T_1 = \frac{4}{3} n^3,\ T_\infty = \mathcal{O}(\log n)$

Application to multiple triangular system solving



When sequential `TRSM` and parallel `Inverse` join:
Compute $X_1 = A_1^{-1} B_1$ in parallel (`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
$z = x_0 + px_1 + p^2 x_2 + \cdots + x_n p^n$
$x = \texttt{RatReconst}(z)$

Multi-adic lifting

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

multi-adic lifting:

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

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

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 parallelsim:

$\Rightarrow$Work stealing

Conclusion

Large grain parallelism:

- Chinese remaindering
- Multi-adic lifting
- Block Wiedemann

Fine grain adaptive parallelsim:

$\Rightarrow$Work stealing

Perspectives

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