LinBox: a Generic Library For Exact Linear Algebra

LinBox Project

Pascal Giorgi (*Arénaire*)

Jean-Guillaume Dumas , Gilles Villard

Laboratoire de l'informatique du parallélisme
ENS Lyon - CNRS - INRIA

Introduction

- LinBox project :

  - Gathering 20 researchers from USA, Canada, France .
    E.Kaltofen [NCSU], D.Saunders [Delaware], M.Giesbrecht [Waterloo], G.Villard [ENSL].
  - Supported by NSF, NSERC, CNRS and ENSL.
  - Online: *http://www.linalg.org*

- C++ library with open source license (40000 lines of code).

  Main Features:

  - Linear Algebra.
  - Most matrix types and black box methods.
  - Genericity w. r. t. the domain of computation.
  - Solutions to various linear algebra problems.

Outline

I) Linbox design.

II) Capabilities provided in Linbox.

III) Computational experiments.

I) LinBox Design

LinBox design

- Three levels of implementation (allowing reuse and reconfiguration)

- Black box and coefficient domain have to respect the specified interface.
- Interfaces= C++ templates and virtual classes named *archetype*:

  - define common object interface.
  - supply one instance as compiled code.
  - control code bloat.
  - optional use at execution.

Field design (coefficient domains)

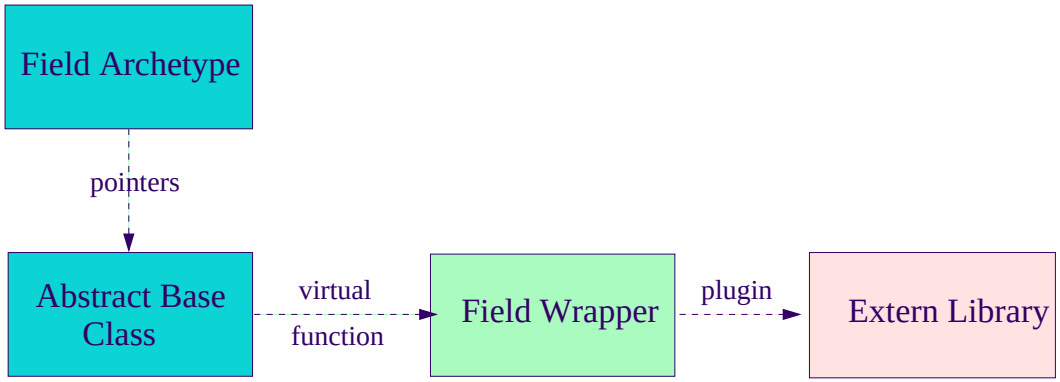
- Parameterized with encapsulated element and random element generator types.
- Elements: contain no information about the field.
- Fields: contain methods for element assignment, equality, arithmetic, IO:

x = y	:	F.assign(x,y)
x == y	:	F.areEqual(x,y)
x = y + z	:	F.add(x,y,z)
cout<< x	:	F.write(cout,x)

- Most fields are extern to the library: Integrated through wrappers.

Fields wrapper and archetype

- An abstract base class has to be compliant with STL (copy constructor).



Timings for different LinBox field levels (arithmetic operations)					
Library	Z/pZ	size of loop	Directly	with Wrapper	with Archetype
NTL::ZZ_p	1978146594666	100.000	0.24 s	0.24 s	0.25 s
//	//	1.000.000	2.45 s	2.45 s	2.54 s
NTL::zz_p	553	300.000	0.16 s	0.17 s	0.21 s
NTL::RR	none	300.000	0.95 s	0.94 s	1.06 s

LinBox Field code example

```
 template<class Domain >
void Fct(Domain& field)
{   typedef typename Domain::element elt;
  elt a,b,r;                  // Declaration of a,b,r as element of Domain

  field.init(a);
  field.init(b);              // Initialisation of a,b,r on the finite field "field"
  field.init(r);

  field.read(cin,a);
  field.read(cin,b);

  field.mul(r,a,b);           // r ← a * b in the finite field "field"

  field.write(cout,r);
}
```

```
void main()
{
  // Declaration of a Z/pZ field over Givaro library with p=53.
  GivaroZpz<Std32> K(53);

  // Declaration of a GF(p^k) field over Lidia library with p=53 and k= 5.
  LidiaGfq Q(53,5);

  cout<<" Z/pZ of Library Givaro \n";
  Fct(K);

  cout<<" GF(p^k) of Library Lidia \n";
  Fct(Q);
}
```

Black box design

- Black box model of a matrix:

- Template-parameterized by a vector class (storage of coefficients).
- Domain of computation given as a parameter or specified as an attribute.
- Only applications to vector allowed:

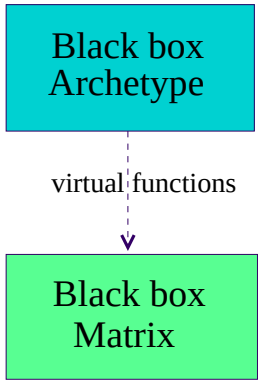
$x = Ay$: A.apply(x,y)
$x = A^T y$: A.applyTranspose(x,y)

- Retrieve matrix dimensions:

A.rowdim()
A.coldim()

Black box archetype and matrix

- No abstract class needed. Directly inherited from archetype.



- Allow to parameterize our algorithms with black box archetype.
- Three kinds of vectors for black box matrices:
  - Dense vector
  - Sparse sequence vector
  - Sparse map vector

Black box archetype code

```
template <class Vector > class BlackboxArchetype {
public:
        virtual  BlackboxArchetype (void) {}
        virtual BlackboxArchetype* clone () const = 0;

        virtual Vector & apply (const Vector &x) const
                        {Vector *y = new Vector ; return apply (*y, x);}
        virtual Vector & apply (Vector &y, const Vector &x) const = 0;
        virtual Vector& applyIn (Vector&x) const
                        {Vectory (x); return apply (x, y);}
        virtual Vector & applyTranspose (const Vector &x) const
                        {Vector *y = new Vector ; return applyTranspose (*y, x);}
        virtual Vector & applyTranspose (Vector & y, const Vector & x) const = 0;
        virtual Vector & applyTransposeIn (Vector &x) const
                        {Vector y (x); return applyTranspose (x, y);}

        virtual size_t rowdim (void) const = 0;
        virtual size_t coldim (void) const = 0;
protected:
        BlackboxArchetype (void)
};
```

II) Capabilities provided in LinBox

Main principles in LinBox

- **Using finite fields** :
  - Heuristic and probabilistic algorithms (Checking, Las-Vegas, Monte-Carlo).
  - Minimal Polynomial (Wiedemann/Lanczos, block methods).
  - Preconditioning (rank , Determinant).
  - Competition with Gaussian elimination.

- **Available methods for rational numbers and integers** :
  - Chinese Remainder Theorem with finite fields.
  - P-adic lifting.
  - Smith normal form (GAP package).
  - Diophantine linear systems.
- **Available Software**:
  - Specialized finite field arithmetic (extensions, tables, polynomials).
  - Solutions :
    - Linear solver.
    - Rank.
    - Determinant.
    - Characteristic polynomial.
    - Minimal polynomial.
  - Matrix conditioners (Toeplitz, butterfly, sparse, diagonal).
  - Matrix normal forms.

III) Computational experiments

Efficiency of the Linbox library

- **Trefethen's One Hundred Digits Challenge:**

  Goal: Compute the $(1,1)$ entry of $\mathbf{A}^{-1}$ with $\mathbf{A}$ a 20000×20000 sparse matrix s. t.

$$\mathbf{A_{ij}} = \begin{cases} \text{the } \mathbf{i}\text{th prime if } \mathbf{i} = \mathbf{j} \\ \mathbf{1} \text{ if } |\mathbf{i} - \mathbf{j}| \text{ is a non-negative power of } \mathbf{2} \\ \mathbf{0} \text{ otherwise} \end{cases}$$

  Computation with Linbox (JG.Dumas, W.turner, Z.Wan):

  - Give the correct result (rational with 100.000 digits).
  - 180 CPUs running for 4 days (about 2 years of CPU time).
  - No memory trashing.

The first digits of the solution (in decimal expansion) :

.72507834626840116746868771925116096886918059447950895787816476920777318999459628357359239278647820204971076163246998474335581698085930387935601894138874208372141550859641621819679121149360170080057048462198581099579540609552948509466411922971401897467433048160703295502693570262002011788628903378989091223659890559067822396534019157226343727468327230077316289848083694790980579565259476908492494879959627551014013005740321928643228161967441484026139665434153

- Since then: improved methods and timings.

Efficiency of LinBox Algorithms

- Rank computation (Gaussian elimination, SuperLU, Wiedemann).
  LinBox: Gauss, Wiedemann - Synaps: SuperLU

Matrix	$\omega, n \times m, r$	Gauss	SuperLU	Wiedem
cyclic8.m11	2462970, 4562x5761, 3903	257.33	448.38	2215.36
bibd.22.8	8953560, 231x319770, 231	100.24	938.91	594.29
n4c6.b12	1721226, 25605x69235, 20165	188.34	1312.27	2158.86
mk13.b5	810810, 135135x270270, 134211	MT		44907.1
ch7-7.b5	211680, 35280x52920, 29448	2179.32	MT	2404.5
ch7-8.b5	846720, 141120x141120, 92959	5375.76	MT	29109.8
ch7-9.b5	2540160, 423360x3175220, 227870	MT		210698
ch8-8.b5	3386880, 564480x376320,279237	MT		363754
TF14	29862, 2644x3160, 2644	50.58	50.34	27.21
TF15	80057, 6334x7742, 6334	734.39	776.68	165.67
TF16	216173, 15437x19321, 15437	18559.40	15625.79	1040.36
TF17	586218, 38132x48630, 38132	MT	MT	7094.97

Rank modulo 65521, Elimination vs. Wiedemann
$n \times m$ is the shape, ω the number of non-zero elements, r the integer rank

Further developements

- Plugin interfaces (Maple, ...)
- Develop a clear and basic documentation (partially achieved).
- Provide the first release of the library.

- Extend the domain of users and developers.
- Implement new algorithms.
- Propose and experiment new arithmetic representations.

www.linalg.org