An interface to link the LinBox library to MAPLE

Pascal Giorgi

Symbolic Computation Group,
University of Waterloo, Canada

ORCCA - Joint Lab Meeting
October 14, 2005.

Exact linear algebra is involved
in many mathematical applications.

Applications in computer algebra :

- Gröbner basis [Faugère LIP6],
  rank, triangularization
- cryptography [Thomé 2003],
  large sparse linear system $(1.033.593 \times 766.150)$
- combinatorial, algebraic topology [Dumas 2000],
  Smith normal form (376.320×117.600)
- integer programming [Aardal, Hurkens, Lenstra 1999],
  sparse diophantine linear system (50.000×50.000)
- ...

Real expectations...

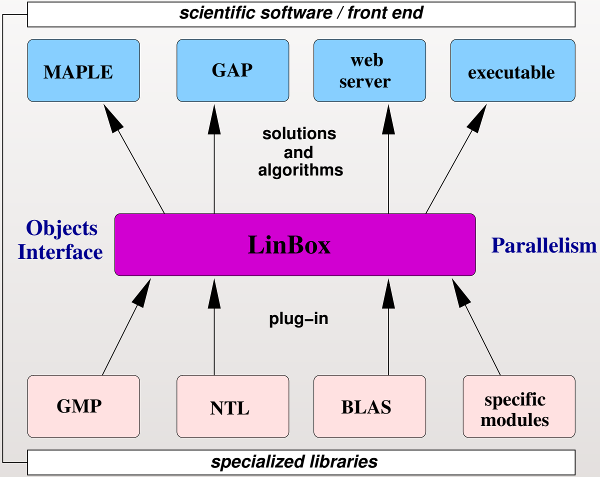
- size of problems becomes larger (matrix dim. > 10.000 is reality),
- recent gains in algorithmic are significant (linear gain, optimality),
- generalist software like MAPLE or MATHEMATICA are no more dominant,
- emergence of high-performance specialized libraries (GMP, NTL, BLAS-LAPACK).

Real expectations...

- size of problems becomes larger (matrix dim. $>$ 10.000 is reality),
- recent gains in algorithmic are significant (linear gain, optimality),
- generalist software like MAPLE or MATHEMATICA are no more dominant,
- emergence of high-performance specialized libraries (GMP, NTL, BLAS-LAPACK).

	recent algorithms
$+$	specialized libraries
$=$	LINBOX library

LINBOX library : a middleware



LINBOX in details

International project

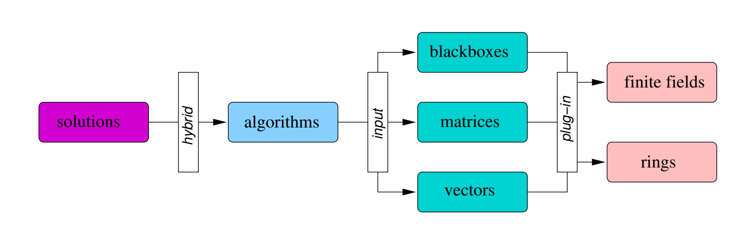
- 32 researchers in Canada, France, USA,
- generic C++ library :
  - LGPL licence,
  - 180.000 lines of code & documentation,
  - LINBOX 1.0 first release [August 2005]
- *www.linalg.org*

Main developments :

- algorithms : linear systems, matrix invariants, ...
- matrices : blackbox, container
- calculation domains : finite fields, integers, rationals
- genericity : template model, plug&play.

LINBOX library : principal of genericity

- 4 levels of implementation (reuse and reconfiguration)



- structures and domains have to match our interfaces
  $\Rightarrow$ genericity achieved with template paradigm
  $\Rightarrow$ integration of external code through *wrappers*

- alternative to static polymorphism :
  archetype $\approx$ Java interface [Kaltofen, Turner]

LINBOX library : solutions

- determinant
- rank
- Smith form
- linear system solving
- minimal polynomial
- characteristic polynomial
- ...

LINBOX library : major methods

- Blackbox algorithms
  - Wiedemann / Block Wiedemann
  - Lanczos / Block Lanczos
- Elimination algorithms
  - matrix product based Gaussian elimination
  - sparse elimination
- Integers via lifting or CRT

LINBOX library : example of use

write C++ code and compile with LINBOX library.

```
#include <linbox/field/modular.h>
#include <linbox/blackbox/sparse.h>
#include <linbox/solution/det.h>
...
// declare the objects
LinBox::Modular<double> Zp(13) ;
LinBox::SparseMatrix<Modular<double> > A(Zp) ;
LinBox::Modular<double>::Element d ;
...
// call the solution
LinBox::det(d,A) ;
```

LinBox library : some performances

- over prime fields :
  - matrix mult. of dim. 5 000 : 36s (30% faster than numeric)
  - dense determinant of dim. 5 000 : 21s (20% slower than numeric)
  - dense inversion of dim. 5 000 : 59s (8% faster than numeric)

- over integers :
  - dense determinant of dim. 2 000 : 184s ($\approx$ time with Maple for dim. 400)
  - dense linear system of dim. 2 500 : 41s ($\approx$ time with Maple for dim. 500)
  - sparse linear system of dim. 10 000 : 2h40mn

How to benefit from LINBOX within MAPLE ?

Our ambitions

Possibility in MAPLE :

- use of LINBOX objects and solutions,
- call LINBOX algorithm with MAPLE objects,

LINBOX library : a driver

objectives :

- make a distribution of the code through a dynamic library,
- avoid explicit type specification,
- provide dynamic objects.

LinBox compiled code

```
#include <linbox/field/modular.h>
#include <linbox/blackbox/sparse.h>
#include <linbox/solution/det.h>
...
// declare the objects
LinBox::Modular<double> Zp(13) ;
LinBox::SparseMatrix<Modular<double> > A(Zp) ;
LinBox::Modular<double>::Element d ;
...
// call the solution
LinBox::det(d,A) ;
```

LinBox compiled code

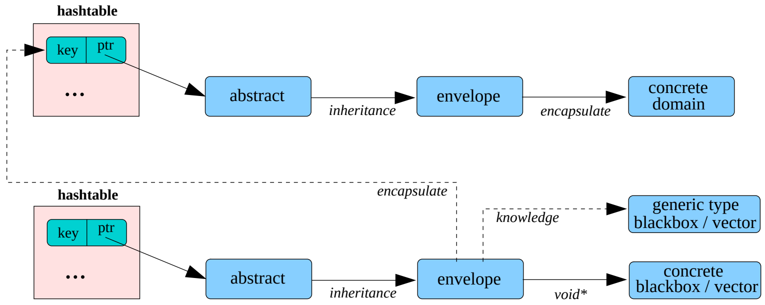
avoid explicit datatype specification ! ! !

```
#include <linbox/field/modular.h>
#include <linbox/blackbox/sparse.h>
#include <linbox/solution/det.h>
...
// declare the objects
LinBox::Modular<double> Zp(13) ;
LinBox::SparseMatrix<Modular<double> > A(Zp) ;
LinBox::Modular<double>::Element d ;
...
// call the solution
LinBox::det(d,A) ;
```

our solution :

- use virtual object and void* to unify type.
- use pointer to funtion to create abstract object :
  each object has a unique pointer to its construction function
- manage virtual object through hashtable :
  each object can be handled simply with a key

Scheme of LINBOX driver object



- blackbox and vector are slightly different

$\Rightarrow$ need to be constructed over a domain

LINBOX driver : example of use

```
#include <linbox/field/modular.h>
#include <linbox/blackbox/sparse.h>
#include <linbox/solution/det.h>
...
// declare the objects
DomainKey Zp = createDomain(13,"linbox_field_dbl");
BlackboxKey A = createBlackbox(Zp, "linbox_sparse");
ElementKey d = createElement(Zp);
...
// call the solution
LinBox::det(d,A);
```

How to call strong type function from a unique key ?

Functionnality over LINBOX driver object

Issue : need to retrieve the concrete type of objects from void*

our solution :

- use call back and visitor [1] to retrieve datatype information

Issue : need to specialize each function on each possible type

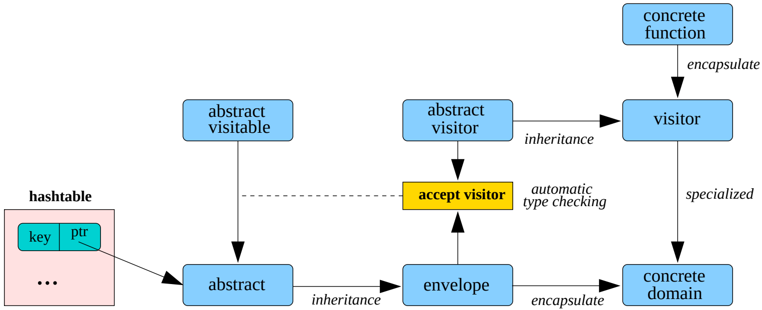
our solution :

- use generic functor and automatic code generation

---

[1] [Alexandrescu. Modern C++ design ; Boost library]

Scheme of LINBOX driver function



- use list of type and automatic code generation to generate all visitors
  - template meta programming
  - multi inheritance

Example of LINBOX driver function

```
// functor to compute the rank
class RankFunctor {
    public :
    template<class Blackbox>
    void operator() (unsigned long &res, Blackbox *B)
       { LinBox::rank(res, *B) ; }
};

// API to compute the rank
void lb_rank(unsigned long &res, const BlackboxKey& key)
{
    RankFunctor fct ;
    BlackboxFunction::call(res, key, fct) ;
}
```

LINBOX driver : example of use

```
#include <lb-driver.h>
...
// declare the objects
DomainKey Zp = createDomain(13,"linbox_field_dbl");
BlackboxKey A = createBlackbox(Zp, "linbox_sparse");
ElementKey d = createElement(Zp);
...
// call the solution
lb_determinant(d,A);
```

linking the LINBOX driver to MAPLE

pretty much straighforward

- need MAPLE object to hanle LINBOX driver object (key),
- provide conversion from MAPLE to LINBOX,
- garbage collect unused LINBOX object.

linking the LINBOX driver to MAPLE

pretty much straighforward

- need MAPLE object to hanle LINBOX driver object (key),
- provide conversion from MAPLE to LINBOX,
- garbage collect unused LINBOX object.

use MAPLE linkage API to extend LINBOX driver through a wrapper

MAPLE interface : a wrapper

- use MaplePointer structure to handle LINBOX driver object.

$\Rightarrow$ use MAPLE garbage collection to deal with unused object

```
ALGEB DomainKeyToMaple (MKernelVector kv, const DomainKey& key)
{
    ALGEB val;
    val = ToMaplePointer(kv, (void*) (&key), (M_INT)&DisposeDomainKey);
    MaplePointerSetMarkFunction (kv, val, MarkDomainKey);
    MaplePointerSetDisposeFunction (kv, val, DisposeDomainKey);
    MaplePointerSetPrintFunction (kv, val, PrintDomainKey);
    return val;
}

const DomainKey& MapleToDomainKey (MKernelVector kv, ALGEB k)
{
    const DomainKey * val = (const DomainKey*) MapleToPointer(kv, k);
    return *val;
}
```

LINBOX/ MAPLE interface

LINBOX driver :

4 000 lines of code ⇒ 20Mo of dynamic library

MAPLE interface :

1 500 lines of code ⇒ 84Ko of dynamic library

Let's do the Demo...

Conclusions

We provide an interface to link the efficient LINBOX library to MAPLE.

- improve the performance of most exact linear algebra solutions.
- allow a direct manipulation of LINBOX (benefit to build efficient implementations)

future works :

- fix the garbage collection
- augment the routines available (Smith form, ...)
- provide choice of algorithm within solutions

How this interface could be incorporated in the LinearAlgebra package ?