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 × 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**).

\[
\text{recent algorithms} + \text{specialized libraries} = \text{LINBOX library}
\]
LINBOX library: a middleware

LinBox

Objects Interface

scientific software / front end

MAPLE
GAP
web server
executable

solutions and algorithms

plug-in

GMP
NTL
BLAS
specific modules

Parallelism

specialized libraries
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
  \[ \Rightarrow \text{genericity achieved with template paradigm} \]
  \[ \Rightarrow \text{integration of external code through } \textit{wrappers} \]

- alternative to static polymorphism:
  \textbf{archetype} \approx \textit{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.

```cpp
#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. 5000: 36s (30% faster than numeric)
  - dense determinant of dim. 5000: 21s (20% slower than numeric)
  - dense inversion of dim. 5000: 59s (8% faster than numeric)

- **over integers:**
  - dense determinant of dim. 2000: 184s (∼ time with MAPLE for dim. 400)
  - dense linear system of dim. 2500: 41s (∼ time with MAPLE for dim. 500)
  - sparse linear system of dim. 10000: 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,
objectives:

- make a distribution of the code through a dynamic library,
- avoid explicit type specification,
- provide dynamic objects.
#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);
avoid explicit datatype specification!!!

```cpp
#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 function 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.
blackbox and vector are slightly different
⇒ need to be constructed over a domain
LINBOX driver : example of use

```cpp
#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]
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

```c
#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 straightforward

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

pretty much straightforward

- need MAPLE object to handle 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.
  ⇒ use MAPLE garbage collection to deal with unused object

```c
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?