LinBox : évolutions et interactions d'une bibliothèque d'algèbre linéaire exacte

Pascal Giorgi

équipe DALI - Université de Perpignan

Journées Nationales de Calcul Formel,
Luminy, 1 février 2007

La bibliothèque LINBOX en quelques mots

- projet international, 32 chercheurs depuis 1997 (Canada, France, USA),
- bibliothèque générique en C++ :
  - licence LGPL ,
  - 180.000 lignes de code & documentation,
  - LINBOX 1.1 - 2nd release [31 janvier 2007]
- *www.linalg.org*

Principaux développements :

- algorithmes : systèmes linéaires, invariants matriciel, ...
- matrices : boîtes noires, denses, structurées
- domaines de calcul : corps finis, entiers, rationnels, polynômes
- généricité : patron "template", plug&play.

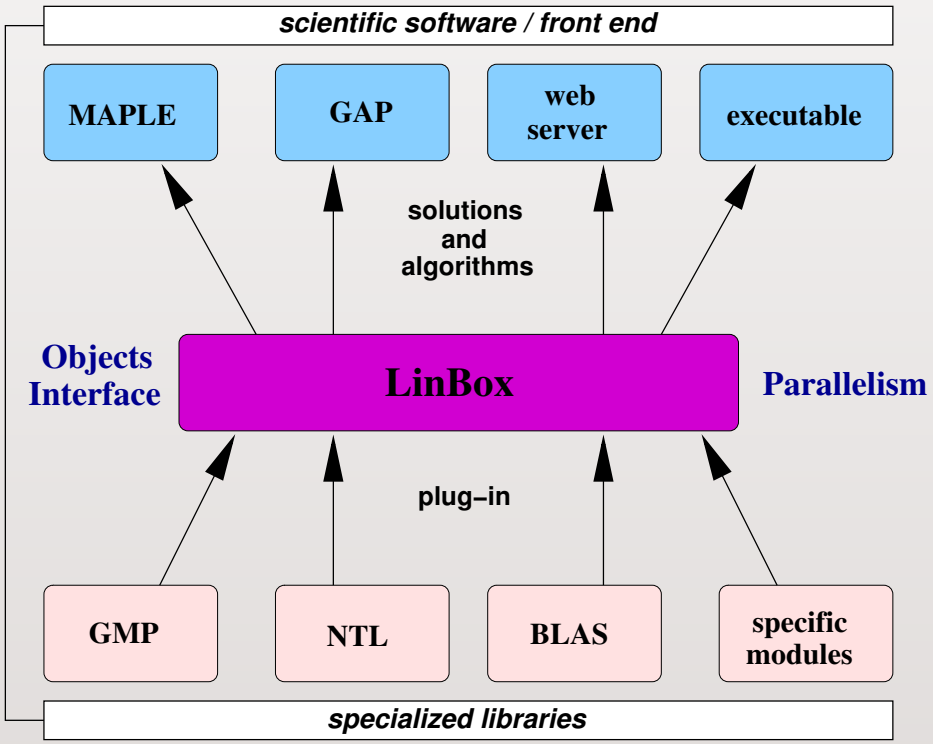
Des besoins réels...

- taille des problèmes de plus en plus grand (matrix dim. $>$ 10.000 courant),
- gains algorithmiques importants (gain linéaire, algorithmes optimaux),
- logiciels généralistes comme MAPLE ou MATHEMATICA ne sont pas adaptés,

Des besoins réels...

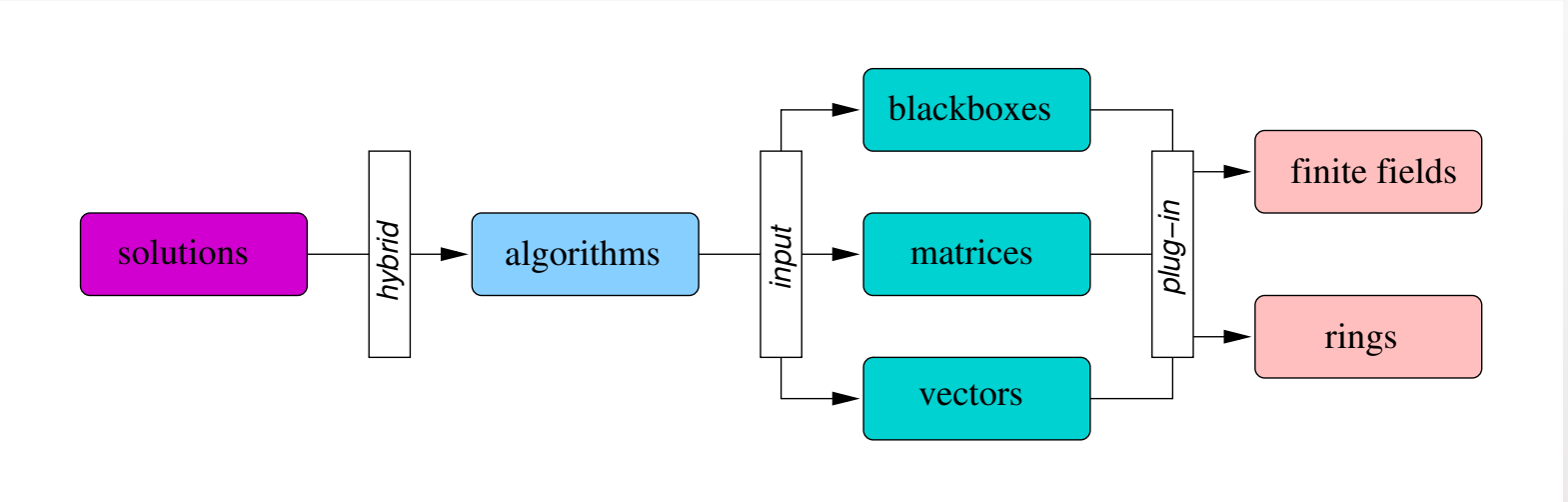
- taille des problèmes de plus en plus grand (matrix dim. $>$ 10.000 courant),
- gains algorithmiques importants (gain linéaire, algorithmes optimaux),
- logiciels généralistes comme MAPLE ou MATHEMATICA ne sont pas adaptés,

Philosophie de LINBOX (*fournir un intergiciel*)



LINBOX : principe de généricité

- 4 niveaux d'implantations (réutilisables et reconfigurables)



- structures et domaines respectent des interfaces
  - $\Rightarrow$ généricité au travers de la notion de patron « *template* »
  - $\Rightarrow$ intégration de codes externes par « *wrappers* »

- alternative au polymorphisme statique :
  archétype $\approx$ Java interface [Kaltofen, Turner]

LINBOX : principaux concepts algorithmiques

- Algorithmes de type boîte noire « *blackbox* »
  - Wiedemann / Block Wiedemann,
  - Lanczos / Block Lanczos.
- Algorithmes à base d'élimination
  - élimination de Gauß par blocs,
  - élimination creuse.
- Solutions entières
  - développement p-adique,
  - théorème des restes chinois,
  - approches adaptatives (terminaison anticipée).

LINBOX : principaux concepts algorithmiques

- Algorithmes de type boîte noire « *blackbox* »
  - Wiedemann / Block Wiedemann,
  - Lanczos / Block Lanczos.
- Algorithmes à base d'élimination
  - élimination de Gauß par blocs,
  - élimination creuse.
- Solutions entières
  - développement p-adique,
  - théorème des restes chinois,
  - approches adaptatives (terminaison anticipée).

algorithmes principalement probabilistes (Monte Carlo / Las Vegas)

LINBOX : concrètement ...

déjà disponible (cf. LINBOX 1.1)

matrices denses ou creuse sur un corps finis ou sur les entiers

- déterminant,
- rang,
- forme de Smith,
- systèmes linéaires (*solutions rationnelles et diophantiennes*),
- polynôme minimal et caractéristique,
- valence,
- matrice définie positive / semi-définie positive.

LINBOX : exemple d'utilisation

écriture d'un code C++ & compilation avec LINBOX

calcul sur un corps finis avec une matrice creuse

```
#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 : exemple d'utilisation

écriture d'un code C++ & compilation avec LINBOX

calcul sur les entiers avec une matrice dense

```
#include <linbox/field/PID-integer.h>
#include <linbox/blackbox/dense.h>
#include <linbox/solution/det.h>
...
// declare the objects
LinBox::PID_integer Z ;
LinBox::DenseMatrix<PID_integer> A(Z) ;
LinBox::PID_integer::Element d ;
...
// call the solution
LinBox::det(d,A) ;
```

LINBOX : quelles sont les performances ?

- Sur un corps finis (*matrice dense d'ordre 5 000*) :
  - Multiplication de matrices : 36s (22% plus rapide que BLAS)
  - Calcul du déterminant : 21s (20% plus lent que LAPACK)
  - Calcul de l'inverse : 59s (8% plus rapide que LAPACK)

- Sur les entiers :
  - Déterminant d'ordre 2 000 : 184s ($\approx$ ordre 400 avec MAPLE)
  - Systèmes linéaires denses d'ordre 2 500 : 41s ($\approx$ ordre 500 avec MAPLE)
  - Systèmes linéaires creux d'ordre 10 000 : 2h40mn ($\approx$ ordre 1 500 avec MAPLE)

LINBOX : encore des performances

calcul du polynôme caractéristique sur GF(547 909)
(ATHLON 2200, 1.8 Ghz, 2Go RAM)[a]

n	500	1 000	2 000	3 000	5 000	10 000
magma 2.11	3.8s	29.9s	238s	802s	3793s	MT
LINBOX	0.4s	2.7s	17.9s	61s	273s	2080s
speed-up	9.5	11	13	13	14	-

[a] implantations et tests réalisés par C. Pernet (2007)

Comment bénéficier de LINBOX dans

son logiciel de calcul formel favori : MAPLE ?

Les problèmes a surmonter

- Fournir une version compilée de LINBOX (pb template).
- Éviter la spécification explicite des types de données.
- Fournir des objets dynamiquement polymorphes.

```
#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);
```

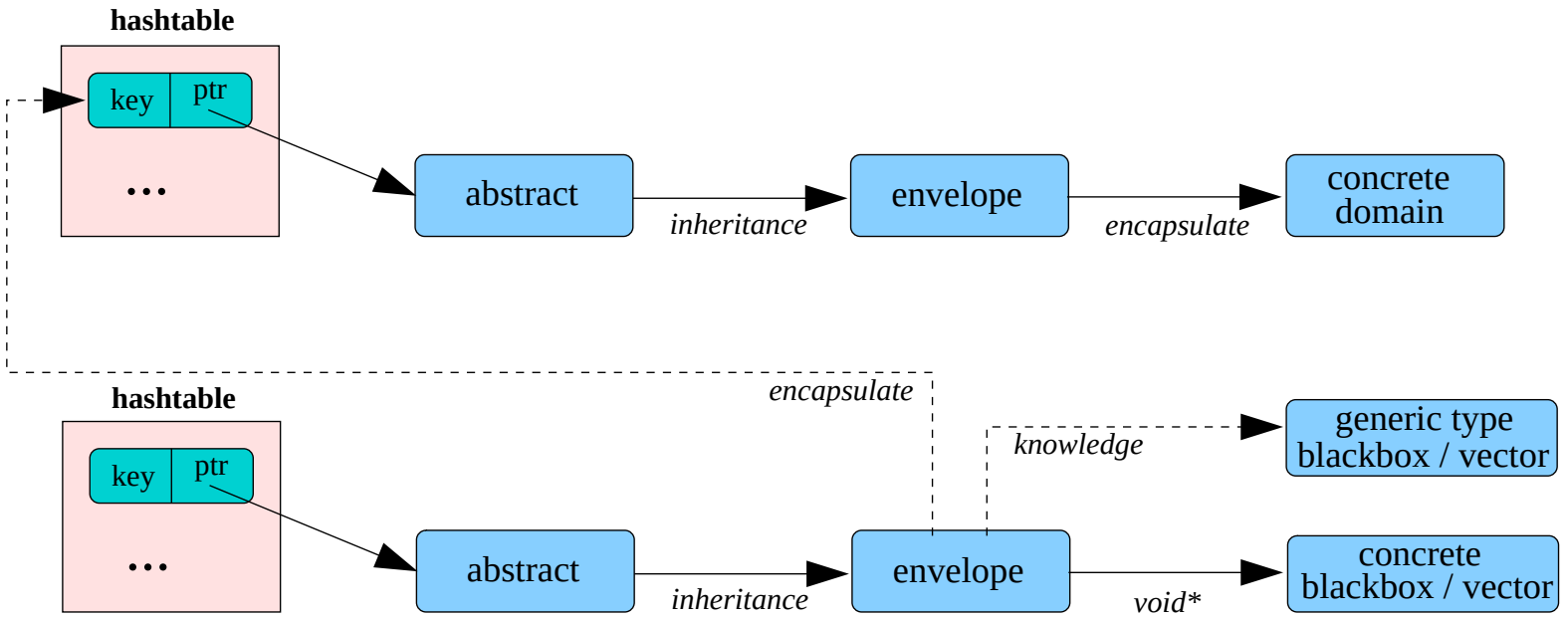
Les problèmes a surmonter

- Fournir une version compilée de LINBOX (pb template).
- Éviter la spécification explicite des types de données.
- Fournir des objets dynamiquement polymorphes.

```
#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);
```

Développement d'un driver pour LINBOX

- Unification des types par objets virtuels et `void*`,
- Création d'objets abstraits par pointeur de fonctions : « factory »,
- Gestion des objets par table de hachage.



Driver LINBOX : suppression des types explicites

```
#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);
```

Comment appeler les bonnes fonctions

à partir d'objets non typés ?

Driver LINBOX : gestion des appels de fonctions

Difficultés majeures :

- besoin de retrouver le type concret des objets vus comme void*,
- besoin d'instancier l'ensemble des fonctions sur tous les types possibles.

Driver LINBOX : gestion des appels de fonctions

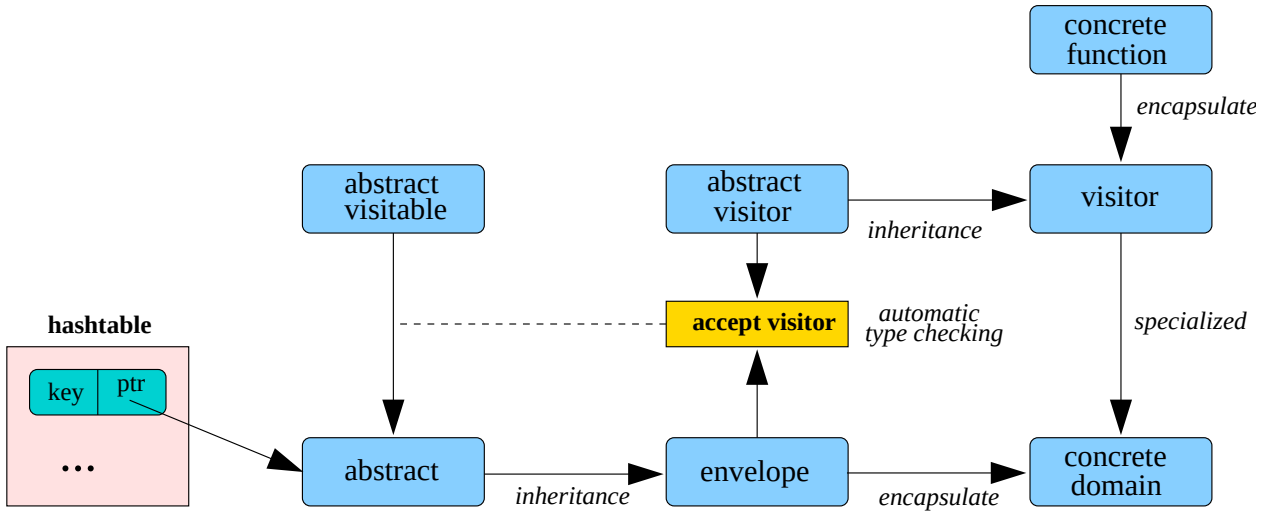
Difficultés majeures :

- besoin de retrouver le type concret des objets vus comme void*,

↪ utilisation de « *call back* » et de « *visitor* »

- besoin d'instancier l'ensemble des fonctions sur tous les types possibles.

↪ utilisation de « *functor* » *générique et de génération automatique de code*



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

Exemple de fonctions du driver LINBOX

Avantage : ajout simple de fonctionnalités

```
// 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 : exemple d'utilisation

```
#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);
```

Incorporation du driver LINBOX dans MAPLE

pratiquement immédiat en
utilisant l'API C fournit par MAPLE

- incorporation des objets LINBOX (clés) dans des objets MAPLE,
- conversion des données (MAPLE $\leftrightarrow$ LINBOX),
- garbage collection des objets LINBOX inutilisés,
- fournir les fonctions MAPLE correspondantes aux fonctions LINBOX.

L'interface en pratique

une petite démo...

Quelles améliorations pour MAPLE ?

Comparaison pour des matrices entières denses à coefficient dans [1..100]

déterminant

n	50	100	200	400
Maple	0.2s	1.6s	16s	536s
LinBox	0.01s	0.09s	0.5s	1.7s
speed-up	15	17	32	302

rang

50	100	200	400
0.46s	1.43s	17.1s	542s
0.006s	0.01s	0.08s	0.29s
77	89	206	1821

systèmes linéaires

n	50	100	200	400
Maple	0.33s	0.8s	4.1s	52s
LinBox	0.01s	0.05s	0.3s	2.5s
speed-up	25	16	10	20

polynôme caractéristique

20	40	80	160
0.12s	2.22s	40.4s	906s
0.007s	0.018s	0.14s	1.3s
17	123	282	678

Conclusions

Nous avons présenté une interface permettant de bénéficier de la bibliothèque LINBOX au sein de MAPLE.

- amélioration considérable des performances de la plupart des calculs en algèbre linéaire exacte,
- préserve le concept de généricité de LINBOX et facilite l'intégration future de nouvelles fonctionnalités,
- le driver facilite l'interfaçage avec d'autre logiciels.

driver LINBOX : 4 000 lignes de code ⇒ 20Mo de bibliothèque dynamique
interface MAPLE : 1 500 lignes de code ⇒ 84Ko de bibliothèque dynamique
disponible dans LinBox 1.1 sur `www.linalg.org`

perspectives :

- augmenter les fonctionnalités disponibles,
- proposer un choix sur les algorithmes utilisés,
- améliorer le *back end layer* pour une programmation effective,
- incorporer LINBOX dans le paquetage LinearAlgebra de MAPLE.