LinBox XXX

aut1 aut2 aut3 autn

Abstract

⚠In this paper…

1 Introduction

The LinBox library is under constant evolution, driven by new problems and algorithms, new compilers, new computing paradigms. This poses new challenges. We are incrementally updating the design of the library towards a 2.0 release.

We show in the next table 1 the increase in the size[1] of LinBox.

Table 1: Evolution of the number of lines of code (loc, in thousands) in LinBox, FFLAS and Givaro (†contains Givaro, ‡contains FFLAS).

LinBox	1.0.0†,‡	1.1.0†,‡	1.1.6‡	1.1.7‡	1.2.0	1.2.2	1.3.0	1.4.0
loc	77.3	85.8	93.5	103.1	107.7	109	111.8	xxx
FFLAS	N/A	N/A	N/A	1.3.3	1.4.0	1.4.3	1.5.0	1.x.x
loc	—	—	—	11.6	23.9	25.2	25.5	xxx
Givaro	N/A	N/A	3.2.16	3.3.3	3.4.3	3.5.0	3.6.0	3.x.x
loc	—	—	30.8	33.5	39.4	48.3	48.6	xxx
total	77.3	85.8	124	137	171	182	186	xxx

The increase in the library size demands a stricter developpement model. For instance, we have put FFLAS files into a new stand-alone FFLAS header library, reduced compile times, enforced stricter warnings and checks, support for more compilers and architectures, simplified and automatised version number changes, automatised memory leak checks,…This demand on the developper side is also driven by the fact that code introduced by various one-timers needs to be maintained.

But this increase also forces the library to be more user friendly. For instance, we have: Developped an `auto-install.sh` script that installs automatically the lastest stable or developpement versions of the trio; Facilitated the discovery of the BLAS/LAPACK libraries; Simplified and sped up the checking process; Added comprehensive benchmarking tools,…

[1] Using sloccount, available at http://sourceforge.net/projects/sloccount/.

This article follows several papers and memoirs on LinBox ([10, 14, 3, 7, 8]) and builds upon them.

Developping generic and high-performance libraries is difficult. We can find a large litterature on coding standards and software desin references in ([1, 9, 13, 12, 11]),and many internet sources and a lot of experience acquired by free software projects.

We are going to describe the advancement in the design of LinBox in the next three sections. We will first describe the new *container* framework in section 2, then improve the *matrix multiplication* algorithms in section 3 by contributing special purpose matrix multiplication plugings, and finally present the new *benchmark/optimisation* architecture (section 4).

2 Containers

LinBox is mainly conceived around the RAII concept with re-entrant function (Resource Acquisition Is Initialisation), introduced by [12], or the *mother model* ([8]) in the sense that the memory used by objects is allocated in the constructor and freed only at its destruction. The gestion of the memory allocated by an object is exlusively reserved to it.

LinBox essentially uses matrix and vectors over fields as data objects. The fragmentation of the containers into various matrices and blackboxes needed to be addressed and simplified. The many different matrix and vector types with different interfaces needed to be simplified into only two (possibly essentially one in the future) containers: `Matrix` and `Vector`.

2.1 Dense Vectors and Matrix

First, all matrices and vectors now depend on the field and not its element, so that operation can always be performed on a matrix or vector. The storage is given by another template parameter that can be defauted to *e.g.* dense BLAS type matrices, *cf.* listing 1.

```
template< class _Field, class _Storage = denseDefault >
class Matrix ;

template< class _Field, class _Storage = denseDefault >
class Vector ;
```

Listing 1: Matrix classes in LinBox.

2.1.1 Interface

The empty constructors are forbidden (no field). In the mother model ([8]), we need types that Own/Share memory. The `SubMatrix` and `SubVector` types

share the memory while `Matrix` and `Vector` own it. The common interface to all matrices is the BlackBox interface listing 2.

```
template< … >
class Matrix {
public:
        /* constructors */
        // with a field, rows, columns, nbnz, other matrix\dots
        …
        /* types */
        // submatrix types
        …

        /* y <- A.x ou y <- A^T.x */
        template<class _in, class _out>
        _out& apply(_out &y, const _in &x) const;
        template<class _in, class _out>
        _out& applyTranspose(_out &y, const _in &x) const;

        /* operator rebind */
        template<typename _Tp1>
        struct rebind ;

        /* dimensions */
        size_t rowdim() const;
        size_t coldim() const;

        /* field */
        const Field& field() const;

        /* conversions ?*/
        // resize
        // export/import to default types
        // read from MM/stdout/...
        // write to MM/stdout/...
        // setEntry/init()/finalise()/optimize()
protected:
        /* internals */
        …
};
```

Listing 2: Interface of an BlackBox

Notes on xxxEntry: Some handy functions are missing from the BlackBox interface. For instance, `refEntry(...)` may be difficult to impelement (compressed fields, sparse matrices). The function `getEntry(...)` may be specialised because potentially costly (it is however a solution). `setEntry(...)` can be used to populate/grow the matrix (from some `init()` until a `finish()` is emitted). setEntry can be (very) costly (ELL format) so set entry should always

build some COO or CSR and then convert to the format. `clearEntry(...)` can be used to zero out an entry if this is allowed (possibly not for structured matrices). ⚠needs trait system for algos to adapt. How to check at compile time (static_assert) ?

2.1.2 Sparse Matrix

Sparse Matrix are tricky to handle and must be well implemented to offer best BlackBox performance (*cf.* [4]). There is a huge litterature on sparse matrix formats, some of which are becoming standard. Numerical analysis brings various shapes for sparse matrices and numerous algorithms and routines. Just like the BLAS numerical routines, we would like to take advantage of existing high performance libraries.

⚠sparse blas, metis, sparse suite, zero-one matrices

Matrix market matrices must be supported (we have a convertor in ffspmvgpu, I should commit it)

SparseMatrix are Matrix with specified storage : COO, CSR, CSC, ELL, ELL_R,..., and more to come. (including 0-1 based)

```
namespace matrixStorage {
        struct sparse {} ;
        struct COO : public sparse {} ;
        struct CSR : public sparse {} ;
        struct ELL : public sparse {} ;
        struct ELLR: public sparse {} ;
        struct HYB : public sparse {} ;
        …
};

template<class _Field >
class Matrix<_Field, matrixStorage::COO> {
        //specialisation  for COO
};
```

Listing 3: Standard sparse matrix formats.

HYB format is a specialisation for ELL(_R)+COO/CSR. no easy setEntry (in the COO/CSR) part ? a function `optimize()`. Could use Metis too. litterature for other HYB.

We use MM reader.

Added to the interface is a convert method to/from CSR (default).

We can adapt the hearder to suit our needs. In particular write matrix in CSR fashion (saving roughly 1/3 space over COO)

XXX timing new matrices, example rank.

2.1.3 Other Matrices

All other matrices (including the special case of Permutations) : the same Structured matrices, add, sub, stacked,...

2.2 The `apply` method

The `apply` method (left or right) is arguably the most important feature in the matrix interface and the LinBox library. It performs what a linear application is defined for: apply to a vector (and by extension a block of vectors, *i.e.* a matrix).

Proposed interface :

```
// y = A.x
template< class _In, class _Out  >
_Out& apply(_Out &y, const _In& x, enum Side) ;

// y = alpha y + beta A.x
template< class _In, class _Out  >
_Out& applyAcc(_Out &y, const Element& alpha, const _In& x, const
    Element& beta, enum Side) ;
```

Listing 4: Proposed apply.

The apply method should be using a `mul` solution as in the listing 5.

```
        template<class _outVector, class _inVector>
        OutVector &apply (_outVector &y, const _inVector &x) const
        {
                /* return _MD.vectorMul (y, *this, x); */
                return blas3::mul(y, *this, x); // selects the
                    best mul algorithm
        }
```

Listing 5: Implantation d'`apply` sur un vecteur par solution `mul`.

3 Improving LinBox matrix multiplication

Efficient matrix multiplication is key to LinBox library.

3.1 Plugin

We propose the following design pattern (the closest design pattner to our knowledge is the *strategy* one, see also [6, Fig 2.]. The main advantage of this design pattern is that the modules always call back the controller so that the best choice is always chosen. Besides modules can be easily added as *plug-ins*. An analogy

can be drawn with dynamic systems—once the controller sends a correction to the system, it receives back a new measure that allows for a new correction.

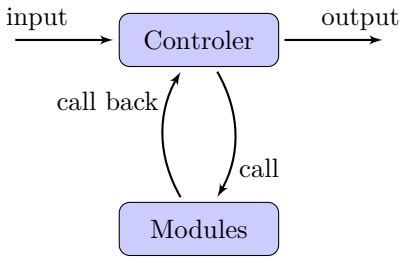

Figure 1: Controller–Modules design pattern

For instance, we can write the standard cascade algorithms in that model:

Algorithm 1: `AlgoThresh`: controler

Input: A and B, denses, with resp. dimensions $n \times k$ and $k \times n$.
Output: $C = A \times B$
if $\min(m, k, n) < t$ **then**
  `BaseCase`(C,A,B) `/* fast BLAS */`
else
  `RecursiveCase`(C,A,B,t)
end
return C ;

Algorithm 2: `RecursiveCase`: recursive module

Input: A, B, C and t as in controller.
Output: $C = A \times B$
Cuts A,B,C in $S_i, T_i \cdots$
...
$P_i =$ `AlgoThresh`(S_i, T_i, t)
...
return C

Figure 2: Conception of a recursive controlled algorithm

This method allows for the reuse of modules and ensures efficiency. It is then possible to adapt to the architecture, the available modules, the resources. The only limitation is that the choice of the module should be done fast. ⚠timing old fgemm/plugin fgemm with no noticeable change ?

3.2 New algorithms/infrastructure

We introduce now several new algorithms that improve on matrix multiplication in various ways: reducing memory consumption, introducing new efficient algorithms, using graphics capabilities, generalizing the BLAS to integer routines.

3.2.1 New algorithms: low memory

`ffgem` in FFLAS uses the classic schedules for the multiplication and the product with accumulation (*cf.* [5]), but we also implement the lower memory routines therein.

The difficutlty consists in using the part of the memory contained in a sub-matrix of the original matrix. It is two-fold. – First we use some part of a memory that has already been allocated to the input matrices, therefore we cannot free and reallocate part of it. – Second, several of these algorithms are meant for square matrices and rectangular sub-matrices will just not be enough. For instance,

⚠table comparing speeds

3.2.2 New algorithms: Bini

We recall in table 2[2] Bini's algorithm.

The formula is recovered as this:

$$C = \sum_{r=1}^{\mu} \left\langle A, X^{(r)} \right\rangle \left\langle B, Y^{(r)} \right\rangle Z^{(r)} \tag{1}$$

where $\langle \cdot, \cdot \rangle$ is the usual Frobenius product.

It is well known ([2]) that approximate formulas for matrix multiplication yield exact matrix matrix multiplication formula. This gives an approximate multiplication formula of type $(3, 2, 2)$ in 10 multiplications (hence $\omega \approx 2.78$, in particular lower than Strassen–Winograd).

However, we can use this algorithm in several settings that seem to be new for getting an exact algorithm. We can for instance set $\epsilon = 2^{-26}$ (so that $\epsilon^2 = 0$ for a `double`) and then round off; or use $\epsilon = p$ for computation in $\mathbf{Z}/p\mathbf{Z}$

Size of p XXX

Implementation and timings XXX use something else than 101 (`float`...)

3.2.3 integer blas

⚠pascal

3.2.4 OpenCL

⚠dave

[2]This table corrects some typo in the original W of the reference.

$$
\begin{array}{lll}
X^{(1)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} & Y^{(1)} = \begin{pmatrix} \epsilon & 0 \\ 0 & 1 \end{pmatrix} & Z^{(1)} = \begin{pmatrix} \epsilon^{-1} & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \\
X^{(2)} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} & Y^{(2)} = \begin{pmatrix} 0 & 0 \\ -1 & -1 \end{pmatrix} & Z^{(2)} = \begin{pmatrix} \epsilon^{-1} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \\
X^{(3)} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} & Y^{(3)} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} & Z^{(3)} = \begin{pmatrix} -\epsilon^{-1} & -\epsilon^{-1} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \\
X^{(4)} = \begin{pmatrix} 0 & \epsilon \\ 0 & 1 \\ 0 & 0 \end{pmatrix} & Y^{(4)} = \begin{pmatrix} -\epsilon & 0 \\ 1 & 0 \end{pmatrix} & Z^{(4)} = \begin{pmatrix} \epsilon^{-1} & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \\
X^{(5)} = \begin{pmatrix} 1 & \epsilon \\ 0 & 0 \\ 0 & 0 \end{pmatrix} & Y^{(5)} = \begin{pmatrix} 0 & \epsilon \\ 0 & 1 \end{pmatrix} & Z^{(5)} = \begin{pmatrix} 0 & \epsilon^{-1} \\ 0 & -1 \\ 0 & 0 \end{pmatrix} \\
X^{(6)} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} & Y^{(6)} = \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} & Z^{(6)} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & \epsilon^{-1} \end{pmatrix} \\
X^{(7)} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} & Y^{(7)} = \begin{pmatrix} -1 & -1 \\ 0 & 0 \end{pmatrix} & Z^{(7)} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \epsilon^{-1} \end{pmatrix} \\
X^{(8)} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} & Y^{(8)} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} & Z^{(8)} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -\epsilon^{-1} & -\epsilon^{-1} \end{pmatrix} \\
X^{(9)} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ \epsilon & 0 \end{pmatrix} & Y^{(9)} = \begin{pmatrix} 0 & 1 \\ 0 & -\epsilon \end{pmatrix} & Z^{(9)} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & \epsilon^{-1} \end{pmatrix} \\
X^{(10} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ \epsilon & 1 \end{pmatrix} & Y^{(10)} = \begin{pmatrix} 1 & 0 \\ \epsilon & 0 \end{pmatrix} & Z^{(10)} = \begin{pmatrix} 0 & 0 \\ -1 & 0 \\ \epsilon^{-1} & 0 \end{pmatrix}
\end{array}
$$

Table 2: Approximate $(3, 2, 2)$ Bini Formula.

Table 3: XXX Algorithme approché de Bini *vs.* Winograd (`fgemm`) sur $\mathbf{Z}/101\mathbf{Z}$ avec seuil de 1,000.

dimensions	temps (s)	
	`fgemm-bini`	`fgemm-wino`
(1,080, 1,080, 1,080)	0.33	**0.32**
(1,800, 1,800, 1,800)	1.37	**1.35**
(2,700, 2,700, 2,700)	**4.38**	4.56
(2,700, 1,800, 1,800)	**3.18**	3.24
(4,500, 1,800, 1,800)	**1.95**	1.98
(4,050, 2,700, 2,700)	**6.35**	6.75

3.2.5 Using conversions

- using flint for integer matmul is faster, even with conversion. Need better CRA implementation.
- implementation of Toom-Cook for GF(q)
- when does spmv choose to optimise ?

4 Benchmarking

Benchmarking was introduced in LinBox for several reasons. It would give the user a user-friendly way for producing nice graph with no necessary knowledge of the graphing library gnuplot[3] or provide LinBox website with automatic new tables/graphs. It would be used for regression testing. It can be used for selecting default method, threshold. (cite atlas and?)

XXX needs introductory bench references
XXX BTL[4]/eigen ⚠dave benchmark formats

4.1 Graph/Table creation

XXX data/plot structure of abstraction (update code 6.1 with new plot structure)

XXX Time spent on each data is limited (will not start execution if fit (linear least squares) forecasts 'too long'.
XXX Adapts to the environment.

4.2 Regression Testing

XXX Tests are supposed to be comparable: regression testing.
XXX howto.

4.3 Method Selecting

XXX Default are provided, method can be selected via a benchmark (cf wino_threshold)
XXX howto

References

[1] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. C++ in-depth series. Addison-Wesley, 2001.

[2] D. Bini. Relations between exact and approximate bilinear algorithms. applications. *Calcolo*, 17:87–97, 1980. 10.1007/BF02575865.

[3] http://www.gnuplot.info/
[4] http://projects.opencascade.org/btl/

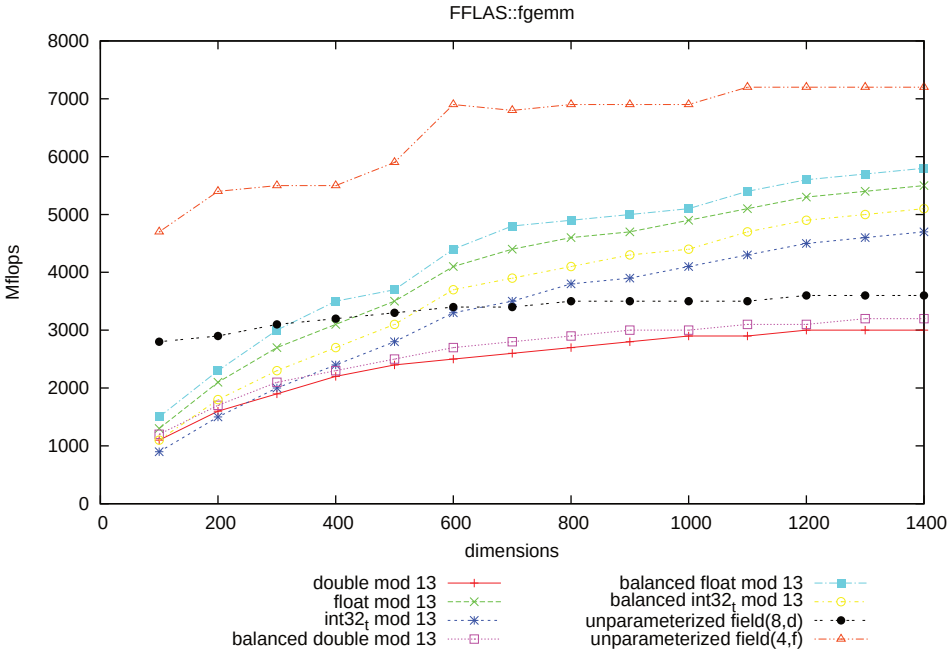

Figure 3: Example of `benchmark`: `fgemm`.

[3] B. Boyer. *Multiplication matricielle efficace et conception logicielle pour la bibliothèque de calcul exact LinBox*. PhD thesis, Université de Grenoble, June 2012.

[4] B. Boyer, J.-G. Dumas, and P. Giorgi. Exact sparse matrix-vector multiplication on GPU's and multicore architectures. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pages 80–88, New York, NY, USA, 2010. ACM.

[5] B. Boyer, J.-G. Dumas, C. Pernet, and W. Zhou. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, ISSAC '09, pages 55–62, New York, NY, USA, 2009. ACM.

[6] V.-D. Cung, V. Danjean, J.-G. Dumas, T. Gautier, G. Huard, B. Raffin, C. Rapine, J.-L. Roch, and D. Trystram. Adaptive and hybrid algorithms: classification and illustration on triangular system solving. In J.-G. Dumas, editor, *Proceedings of Transgressive Computing 2006, Granada, España*, Apr. 2006.

[7] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. LinBox: A

generic library for exact linear algebra. In *Proceedings of the 2002 International Congress of Mathematical Software, Beijing, China*. World Scientific Pub, Aug. 2002.

[8] J.-G. Dumas, T. Gautier, C. Pernet, and B. D. Saunders. LinBox founding scope allocation, parallel building blocks, and separate compilation. In K. Fukuda, J. Van Der Hoeven, and M. Joswig, editors, *Proceedings of the Third International Congress Conference on Mathematical Software*, volume 6327 of *ICMS'10*, pages 77–83, Berlin, Heidelberg, Sept. 2010. Springer-Verlag.

[9] E. Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

[10] P. Giorgi. *Arithmétique et algorithmique en algèbre linéaire exacte pour la bibliothèque* LinBox. PhD thesis, École normale supérieure de Lyon, Dec. 2004.

[11] D. Gregor, J. Järvi, M. Kulkarni, A. Lumsdaine, D. Musser, and S. Schupp. Generic programming and high-performance libraries. *International Journal of Parallel Programming*, 33:145–164, 2005. 10.1007/s10766-005-3580-8.

[12] B. Stroustrup. *The design and evolution of C++*. Programming languages/C++. Addison-Wesley, 1994.

[13] H. Sutter and A. Alexandrescu. *C++ Coding Standards: 101 Rules, Guidelines, And Best Practices*. The C++ In-Depth Series. Addison-Wesley, 2005.

[14] W. J. Turner. *Blackbox linear algebra with the* LinBox *library*. PhD thesis, North Carolina State University, May 2002.