Elements of Design for Containers and Solutions in the LinBox library
Extended abstract

Brice Boyer[1], Jean-Guillaume Dumas[2], Pascal Giorgi[3], Clément Pernet[4], and B. David Saunders[5]

[1] Department of Mathematics, North Carolina State University, USA[†]
bbboyer@ncsu.edu.
[2] Laboratoire J. Kuntzmann, Université de Grenoble. France[‡]
Jean-Guillaume.Dumas@imag.fr.
[3] LIRMM, CNRS, Université Montpellier 2, France[‡]
pascal.giorgi@lirmm.fr.
[4] Laboratoire LIG, Université de Grenoble et INRIA, France[‡]
clement.pernet@imag.fr.
[5] University of Delaware, Computer and Information Science Department, USA
saunders@udel.edu.

Abstract. We describe in this paper new design techniques used in the C++ exact linear algebra library LinBox, intended to make the library safer and easier to use, while keeping it generic and efficient. First, we review the new simplified structure for containers, based on our *founding scope allocation* model. We explain design choices and their impact on coding: unification of our matrix classes, clearer model for matrices and submatrices, *etc.* Then we present a variation of the *strategy* design pattern that is comprised of a controller–plugin system: the controller (solution) chooses among plug-ins (algorithms) that always call back the controllers for subtasks. We give examples using the solution mul. Finally we present a benchmark architecture that serves two purposes: Providing the user with easier ways to produce graphs; Creating a framework for automatically tuning the library and supporting regression testing.

Keywords: LinBox; design pattern; algorithms and containers; benchmarking; matrix multiplication algorithms; exact linear algebra.

1 Introduction

This article follows several papers and memoirs concerning LinBox[6] (*cf.* [2, 7, 8, 13, 19]) and builds upon them. LinBox is a C++ template library for fast and

[†] This material is based on work supported in part by the National Science Foundation under Grant CCF-1115772 (Kaltofen)
[‡] This material is based on work supported in part by the Agence Nationale pour la Recherche under Grant ANR-11-BS02-013 HPAC (Dumas, Giorgi, Pernet).
[6] See http://www.linalg.org.

exact linear algebra, designed with generality and efficiency in mind. The LinBox library is under constant evolution, driven by new problems and algorithms, by new computing paradigms, new compilers and architectures. This poses many challenges: we are incrementally updating the *design* of the library towards a 2.0 release. The evolution is also motivated by developing a high-performance mathematical library available for researchers and engineers that is easy to use and help produce quality reliable results and quality research papers.

Let us start from a basic consideration: we show in the Table 1 the increase in the "lines of code" size[7] of LinBox and its coevolved dependencies Givaro and Fflas–Ffpack[8]. This increase affects the library in several ways. First, it demands

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 ($\times 1000$)	77.3	85.8	93.5	103	108	109	112	135
Fflas–Ffpack	n/a	n/a	n/a	1.3.3	1.4.0	1.4.3	1.5.0	1.8.0
loc	—	—	—	11.6	23.9	25.2	25.5	32.1
Givaro	n/a	n/a	3.2.16	3.3.3	3.4.3	3.5.0	3.6.0	3.8.0
loc	—	—	30.8	33.6	39.4	41.1	41.4	42.8
total	77.3	85.8	124	137	171	175	179	210

Table 1. Evolution of the number of lines of code in LinBox.

a stricter development model, and we are going to list some techniques we used. For instance, we have transformed Fflas–Ffpack (*cf.* [10]) into a new standalone header library, resulting in more visibility for the Fflas–Ffpack project and also in better structure and maintainability of the library. A larger template library is harder to manage. There is more difficulty to trace, debug, and write new code. Techniques employed for easier development include reducing compile times, enforcing stricter warnings and checks, supporting more compilers and architectures, simplifying and automating version number changes, automating memory leak checks, and setting up buildbots to check the code frequently.

This size increase also requires more efforts to make the library user friendly. For instance, we have: Developed scripts that install automatically the latest stable/development versions of the trio, resolving version dependencies; Eased the discovery of Blas/Lapack libraries; Simplified and sped up the checking process, covering more of the library; Updated the documentation and distinguished user and developer oriented docs; Added comprehensive benchmarking tools.

Developing generic high performance libraries is difficult. We can find a large literature on coding standards and software design references in (*cf.* [1,11,15,17, 18]), and draw from many internet sources and experience acquired by/from free software projects. We describe advances in the design of LinBox in the next three sections. We will first describe the new *container* framework in Section 2, then, in Section 3, the improved *matrix multiplication* algorithms made by contributing special purpose matrix multiplication plugins, and, finally, we present the new *benchmark/optimization* architecture (Section 4).

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

[8] symbol †when Givaro is included and ‡when contains Fflas–Ffpack