LiDIA

A library for computational number theory

Ingrid Biehl, Johannes Buchmann, Thomas Papanikolaou
Universität des Saarlandes
Fachbereich 14
66041 Saarbrücken

1 Introduction

In this paper we describe LiDIA, a new library for computational number theory.

Why do we work on a new library for computational number theory when such powerful tools as **Pari** [1], **Kant** [11], **Simath** [10] already exist? In fact, those systems are very useful for solving problems for which there exist efficient system routines. For example, using **Pari** or **Kant** it is possible to compute invariants of algebraic number fields and **Simath** can be used to find the rank of an elliptic curve over $\mathbb{Q}$. However, building complicated and efficient software on top of existing systems has in our experience turned out to be very difficult. Therefore, the software of our research group is developed independently of other computer algebra systems.

Over the past years we have been working on many problems of computational number theory such as factoring integers [5], computing discrete logarithms over finite fields [13], counting the number of points on an elliptic curve over a finite field [8], computing the class number of number fields [2], etc. In those projects software for many basic tasks was needed, for example a multiprecision integer and floating point arithmetic, a polynomial arithmetic, linear algebra routines, etc. However, reusing parts of the software written in one project as modules in other projects was almost impossible. There was not enough documentation, there were no well defined interfaces, in different projects different multiprecision integer arithmetic packages and memory managers were used. Therefore, the same algorithms were implemented many times and not always in the most efficient way. It seems to us that this is a typical situation in scientific computing.

To overcome the problem of reusability we decided in early 1994 to organize our software in the C++-library LiDIA. The first release of LiDIA will be available for the public by the end of February 1995 via anonymous ftp from `crypt1.cs.uni-sb.de`. It is the goal of this paper to describe the design, the contents and the use of LiDIA.

2 Design of LiDIA

We find that *object oriented programming* is appropriate for implementing mathematical algorithms. Using an object oriented language it is possible to create objects which correspond to mathematical objects. Algorithms can be implemented on the right level of abstraction. The implementations look very similar to the mathematical formulation of the algorithm. So far, all software of our group was written in C. Therefore, we decided to use C++ as the *implementation language* for LiDIA.

To guarantee easy *portability* of LiDIA we decided to have a very small machine dependent kernel in LiDIA. That kernel currently only contains the multiprecision integer arithmetic. In the next release it will also contain a memory manager. All LiDIA classes are written in C++ and compiled with different compilers on different architectures. Currently we use the compilers **CC (cfront-3.0.1)** and **g++-2.5.8, g++-2.6.x** on `sparc7`, `sparc8`, `mips R4000`, `intel 386/486`, `hp 9000/712` and `hp 9000/735`.

It is a serious problem to decide which multiprecision integer package and which memory manager should be used in LiDIA. There are competing multiprecision integer packages, for example the **GNU gmp**-package [7], the **libI**-package [6], and the **lip**-package [9]. Some of those packages are more efficient on one architecture and some on others. Also, new architectures will lead to new multiprecision packages. We decided to make LiDIA independent of a particular kernel but to make it easy to replace the LiDIA-kernel. To achieve this, we separate the kernel from the application programs by an *interface* in which the declarations, operators, functions, and procedures dealing with multiprecision integers and the memory management are standardized. All LiDIA-classes use the kernel through that interface. They never use the kernel functions directly. Whenever new kernel packages are used only the interface has to be changed appropriately but none of the LiDIA classes has to be altered. At the moment, it is possible to use all three multiprecision integer packages **libI**, **GNU gmp**, and **lip** with LiDIA.

It is the goal of the LiDIA project to develop extremely *efficient code*. We compare the running times of LiDIA with the running times of other systems, in particular of **Pari**, and we try to be faster. Of course, we learn a lot from the implementations in other systems and we make our improvements available to the groups developing the other systems.

Currently the classes of LiDIA are documented in *manual pages* which are similar to the UNIX manual pages (see [3]). We are working on a tool that makes the documentation interactively available and which in particular supports search in the LiDIA documentation. That documentation tool will be part of a LiDIA development environment which will also be in the public domain such that many groups can participate in the development of LiDIA.

Like other systems, we will make the LiDIA functions available for quick interactive use. There will be the language **lc** which supports the data types and methods of LiDIA and an interpreter for that language. The language **lc** is typeless and very similar to C++. We could have also used another computer algebra language, for example the **Maple**-language [4]. Our idea was, however, that people who develop software using the LiDIA library might want to run experiments and to quickly write and test prototypes using **lc** and then transform the **lc**-code into proper C++.

Concluding this description we show the structure of LiDIA in the following picture.

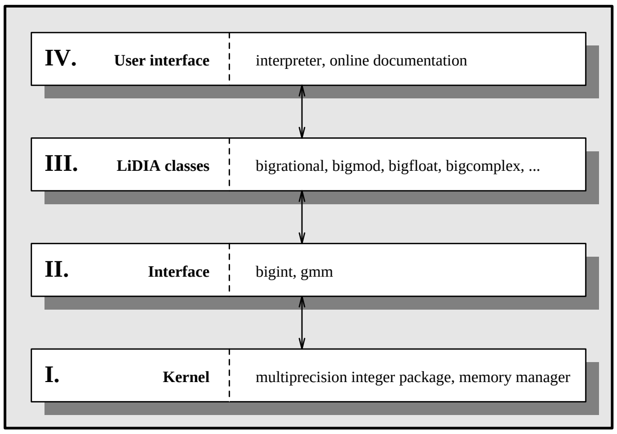

Figure 1: LiDIA's structure

3 Contents of LiDIA

In this section we briefly describe the contents of LiDIA version 1.0 which will be available by the end of February 1995. For a more detailed description we refer to the manual [3].

Kernel The kernel of LiDIA contains the multiprecision integer package which is used by all LiDIA classes. With LiDIA we distribute the multiprecision integer package **libI** [6] which is developed by R. Dentzer [6]. To our knowledge, that package is the most efficient one currently available. It exists for `sparc7`, `sparc8`, `mips R4000`, `intel 368/486`, `hp 9000/712` and `hp 9000/735` architectures.

LiDIA can also be used with the **GNU gmp**-package which is available from `prep.ai.mit.edu` and with the **lip**-package which can be obtained from `flash.bellcore.com:/pub/lenstra` (both by anonymous ftp).

The minimum requirements for a multiprecision integer package used in the kernel are the following. That package must support assignments, the arithmetic operations +, -, $*$, /, the comparisons ==, != , >, <, >=, <=, the comparison with zero and the bit operations (not), & (and), —(or), ^(xor). Also, there must be routines for extracting square roots, for the (extended) Euclidean algorithm and a pseudo random number generator.

Interface The interface is a class **bigint** for doing multiprecision integer arithmetic. That class calls functions from the kernel. All operators that exist in C++ for the machine type **long** are overloaded in **bigint** and behave in the same way. There are also procedural versions of many of the operations.

Classes The class **bigrational** supports arithmetic with rational numbers. The class **bigmod** supports arithmetic in $\mathbb{Z}/m\mathbb{Z}$ for a positive integer m. The class **bigfloat** supports multiprecision floating point arithmetic including the evaluation of elementary functions such as exp, log, sin, cos, etc. The class **bigcomplex** supports arithmetic with complex numbers. Again, in all those classes the standard arithmetic operations and the standard comparison operations are overloaded.

We present timings of multiprecision floating point calculations with the **bigfloat** class of LiDIA, **Pari**-1.38.71, **Maple**-V-R2, and **Mathematica**-2.1. All calculations were done with an accuracy of 1000 decimal places on a SUN Sparc ELC with 21.5 MIPS. The timings are given in milliseconds.

Function	LiDIA	**Pari**	**Maple**	**Mathematica**
π	160	170	0	650
γ	9520	7840	72466	22316
e	170	970	667	183
$\sqrt{3}$	27	50	467	166
exp(log(2))	910	1020	2333	6066
log(exp(2))	1532	1540	1300	4700
$\sin(\pi/3)$	945	1370	6083	6783
$\cos(\pi/3)$	945	1250	6000	6133
$\arcsin(\sqrt{3}/2)$	2850	3460	8767	24200
$\arccos(\sqrt{3}/2)$	2840	3440	8616	23533
sinh(log(2))	940	1040	2683	2633
cosh(log(2))	930	1050	2683	2650
$\mathrm{arsinh}(\pi)$	1590	1620	4584	8950
$\mathrm{arcosh}(\pi)$	1590	1610	4083	4950

There are also classes in which more elaborate algorithms are implemented. The class **bigint_matrix** allows doing linear algebra over the integers. In that class it is possible to compute the kernel, the image, the determinant, the characteristic polynomial, the Hermite normal form, etc. of an integer matrix. In the following table we list the running times needed by a SUN Sparc ELC with 21.5 MIPS for the calculation of the determinant of a $n \times n$ **bigint** matrix. The entries of the matrix were randomly chosen from $[10^8, 10^9 - 1]$. The timings are given in seconds.

n	LiDIA	**Pari**	**Maple**	**Mathematica**
10	0.09	0.05	0.83	0.58
20	0.86	1.03	11.23	2.50
30	3.69	6.49	67.11	7.71
40	11.06	25.17	263.11	19.67
50	26.19	72.23	762.333	40.30
70	96.01	-	3828.33	133.45
90	257.52	-	13103.80	342.35
110	571.61	-	-	729.13
150	1969.19	-	-	2413.13
210	7530.86	-	-	9040.17
230	10790.71	-	-	12925.6
250	15136.49	-	-	-

The "-" means that we were not able to compute a result in a reasonable amount of time and space. For example in the case 70×70 **Pari** required more than 32 megabytes of memory and in the case 110×110 **Maple** would need more the 15 hours. Therefore, those calculations have not been done.

The classes **lattice_gensys** and **lattice_basis** allow to find lattice bases from generating systems and to apply LLL-reduction to lattice bases. In a future release it will also be possible to find shortest non zero lattice vectors, to find Minkowski- and Korkine-Zolotaref-reduced bases etc. In the following table we list the running times needed by a SUN Sparc8 for the calculation of the LLL-reduced matrix of a $n \times n$ Schnorr-Factor-matrix (see [12]).

n	LiDIA	Pari	Maple	Mathematica
5	0.57	0.78	75.43	5.58
10	4.30	6.39	470.25	28.30
15	15.10	22.67	-	69.10
20	37.21	55.70	-	-
25	72.35	107.75	-	-
30	126.50	185.69	-	-
35	199.36	300.55	-	-
40	296.74	435.32	-	-
45	410.87	595.73	-	-
50	580.72	800.73	-	-
55	704.53	-	-	-
60	870.04	-	-	-
70	1258.27	-	-	-

The class **integer_factorization** supports arithmetic with factorizations of integers. In particular, in that class the prime factorization of an integer can be found by means of a strategy which uses trial division and the elliptic curve method (ECM). Our implementation of the quadratic sieve algorithm (QS) is currently being ported to LiDIA.

In the following tables we present two factorizations found by ECM on a Sparc ELC with 21.5 MIPS and the corresponding timings of LiDIA, **Pari**, **Maple** and **Mathematica**. The timings are given in seconds.

$n_1 =$	20968223185509467561
$=$	2335041463 $*$ 8979807647
$n_2 =$	29457531316691559853457337307
$=$	1525079873 $*$ 2888506403 $*$ 6686986153

Number	LiDIA	Pari	Maple	Mathematica
n_1	6.65	65.26	227.21	57.31
n_2	17.52	84.20	375.18	212.31

4 How to use LiDIA

LiDIA is available via anonymous ftp from **crypt1.cs.uni-sb.de**. The LiDIA package comes with an installation routine which should be very easy to use. Once LiDIA is installed it can be used as any other C++-library. Here is a sample program that reads a number from the standard input and prints its factorization on the standard output:

```
#include <LiDIA/integer_factorization.h>

int main()
{
  integer_factorization f;
  bigint n;

  cout << "enter an integer n: ";
  cin >> n;      // 1. input n
  f.factor(n);  // 2. calculate and store the
                //    factorization of n in f
  cout << f;     // 3. output f
}
```

Figure 2: A LiDIA program for factoring integers.

5 The LiDIA Team

At the moment the LiDIA team consists of the following people:

Werner Backes	Oliver Morsch
Franz-Dieter Berger	Markus Maurer
Sascha Demetrio	Stefan Neis
Thomas Denny	Victor Shoup
Kurt Huwig	Oliver van Sprang
Thorsten Lauer	Patrick Theobald
Frank Lehmann	Damian Weber
Andreas Müller	René Weiskircher
Volker Müller	Susanne Wetzel

and the authors.

References

[1] Batut, C., Bernardi, D., Cohen, H., Olivier, M., "User's Guide to PARI - GP", available with PARI-package by anonymous ftp on megrez.math.u-bordeaux.fr, August 94.

[2] Buchmann, J., Düllmann, S., "A probabilistic class group and regular algorithm and its implementation", Computational Number Theory (Pethö, A., Pohst, M., Williams, H.C., Zimmer, H.G.,eds), Walter de Gruiter Verlag Berlin, 1991, p. 35-72.

[3] Buchmann, J., Papanikolaou, T., "LiDIA manual", Lehrstuhl Prof. Buchmann, Universität des Saarlandes, 1994.

[4] Char, B.W., Geddes, K.O., Gonnet, G.H., Leong, B.L., Monagan, M.B., Watt, S.M., "Maple V Library reference Manual", Waterloo Maple Publishing, 1991.

[5] Denny, T. Dodson, B., Lenstra, A. K., Manasse, M. S., "On the factorization of RSA-120", Proceedings of CRYPTO'93, Springer, 1993.

[6] Dentzer, R., "libI: eine lange ganzzahlige Arithmetik", IWR Heidelberg, 1991.

[7] Granlund, T., "The GNU Multiple Precision Arithmetic Library", Free Software Foundation Inc., Cambridge, 1993.

[8] Lehmann, F., Maurer, M., Müller, V., Shoup, V.,"Counting the number of points on elliptic curves over finite fields of characteristic greater than three", Proceedings of the first International Symposium on Algorithmic Number Theory, Lecture Notes in Computer Science 877, p. 60-70, 1994.

[9] Lenstra, A., "lip: long integer package", Bellcore, 1989.

[10] "SIMATH reference manual", available with the SIMATH-package by anonymous ftp on math.uni-sb.de.

[11] von Schmettow, J., Jüntgen, M., "KANT - A Programmer's Guide", available with the KANT package by anonymous ftp on math.tu-berlin.de.

[12] Schnorr, C.P., "Factoring Integers and Computing Discrete Logarithms via Diophantine Approximations", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993.

[13] Weber, D., "An implementation of the general number field sieve to compute discrete logarithms mod p", submitted, 1994.

[14] Wolfram, S., "Mathematica: A System For Doing Mathematics By Computer", Addison-Wesley Publishing Company, 1990.