ANTIC: Algebraic Number Theory In C

W. Hart
(TU Kaiserslautern)

goodwillhart@googlemail.com

Introduction

From about the second year of the development of the Flint (Fast Library for Number Theory) project [3], we had our sights set on extending Flint with functionality in algebraic number theory.

As an algebraic number theorist, there are three obvious computer algebra systems to use: Magma [1], a closed source system developed in Sydney, Pari/GP [5] an open source library and interpreter developed in Bordeaux and the Sage project [7], an international open source project.

As far as algebraic number theory is concerned, much of the functionality in Sage already derives from existing open source packages, such as Flint, NTL and Pari/GP, so it isn't completely independent.

Pari/GP is specialised for algebraic number theory and has reasonable functionality. But the project is less focused on highly optimised code and asymptotically fast algorithms.

The remarkable conclusion is that if one wishes to do computations in algebraic number theory, there isn't a wide choice for the user.

As announced last year at the 4th annual meeting of the DFG Priority Project on computer algebra in Bad Boll, Germany, the ANTIC library [2] aims to provide a modern, highly optimised alternative to the existing options.

Building ANTIC

ANTIC is an extension package for the Flint project, written in C. As a Flint extension it benefits from all of the existing infrastructure in Flint, including its "magic build system" which enables developers to add files, tests and profiles without modifying or reconfiguring the build system, and a vast array of highly optimised, low-level and asymptotically fast routines for integers, rationals, integers modulo n, finite fields and p-adics, and polynomials, power series and matrices over all of the former.

Building ANTIC is trivial. When configuring Flint one supplies:

```
./configure --extensions="/path/to/antic" [other
    options ...]
```

One may also build Flint with the Arb extension [4], a package developed by Fredrik Johansson for arbitrary precision ball arithmetic over the reals and complex numbers. It includes many highly optimised implementations of transcendental functions, including L-series, gamma functions and much more, all with proven error bounds.

ANTIC consists of a number of modules, each of which announces itself to Flint via a header file and similarly named directory at the top level of the ANTIC source tree, which can be unpacked anywhere.

Progress to date

Our task in this report is to update the computer algebra community on the progress to date in the ANTIC library.

There are three modules in ANTIC so far: QFB a module for postive definite binary quadratic forms, NF for precomputing information about number fields to speed up computations, and NF_ELEM a highly optimised module for basic arithmetic in number fields.

The QFB module allows one to compute reduced forms, prime forms, to compose, square and raise forms to a power. There are also functions for computing the exponent of the form class group (the bulk of the cost in computing the structure of the class group of imaginary quadratic number fields).

We have already used this code for finding the class number of imaginary quadratic number fields with smooth group order for discriminants of 110 decimal digits. For example after some hours, our code using ANTIC responded with

```
Discriminant:

          -4000000000000000000000000000000000000
          00000000000000000000000000000000000000
          000000000000000000000000000000000648

Exponent:
```

```
        630482727511402342574628260968489386 82
        076295416810640

Class number:

        2^10 * 3 * 5 * 7 * 12421 * 45841 *
        56149 * 14703947 * 17197469 *
        88721167 * 52322143935097
```

The NF module allows one to specify a number field by its defining polynomial over the rationals. ANTIC distinguishes the case where the defining polynomial is monic and integral, where various optimisations can happen, and the general case.

The polynomial is stored in the Flint FMPQ_POLY format, namely as a polynomial over the integers, with an integer denominator.

We tried various kinds of precomputed inverse to speed up reduction modulo the defining polynomial. In the end, only two methods were actually faster than generic polynomial arithmetic in Flint.

The first method we use is to precompute an inverse of the leading coefficient of the numerator of the defining polynomial.

All of the basic Flint polynomial division and remainder code was modified to optionally accept this precomputed inverse.

The second method was to compute a table of powers of x modulo the defining polynomial. If the degree of the defining polynomial $f(x)$ is n we compute $x^i \pmod{f(x)}$ for $i < 2n + 1$. This optimisation is performed whether the defining polynomial is integral and monic or in the generic rational case.

However, this optimisation is only performed when the degree of the number field is less than 30. Beyond that, the cost of creating the table of powers becomes too large for some applications.

A third kind of precomputation we perform is that of the generalised traces

$$S_k = \sum_i \theta_i^k \text{ for } 0 \le k \le n, \tag{10}$$

where the θ_i are the roots of the minimum polynomial of degree n.

The generalised traces are computed exactly using a recurrence relation, from the coefficients of the defining polynomial.

The NF_ELEM module is used for creating and doing arithmetic with elements inside a number field created with the NF module.

In the general case, a number field element is stored as a rational polynomial. All of the Flint polynomial arithmetic is optimised to deal specially with the case that the denominator of such a polynomial is one.

The NF_ELEM module always ensures that these polynomials have space for $2n - 1$ coefficients. This allows the product of two polynomials of degree $n - 1$ to be computed and stored, without reduction modulo the defining polynomial of the number field. This is a critical optimisation if one is multiplying matrices over a number field, where one wishes to accumulate products of number field elements whilst performing dot products, before doing a single final reduction modulo the defining polynomial at the end.

In the case of linear and quadratic number fields, we have a slightly more efficient representation. In the linear case, we store integers representing a numerator and denominator. In the quadratic case, we have space for three integers for an (unreduced) numerator and one integer for a denominator.

All of the operations in ANTIC are specially optimised for linear and quadratic fields.

Multiplication of number field elements is the most involved code in ANTIC's NF_ELEM module. We allow for multiplication of number field elements without reduction modulo the defining polynomial, and also without canonicalisation of the rational coefficients into lowest terms.

Both of these optimisations seem to be made for number field arithmetic in the Magma computer algebra system.

The norm of number field elements is computed by taking the resultant of the polynomial representing the element and the polynomial defining the number field.

Flint offers asymptotically fast resultant (quasilinear in the degree) using a half-gcd style resultant algorithm. Timings of Magma seem to indicate that this optimisation also exists in Magma, at least for polynomial arithmetic.

The trace of number field elements is computed by taking the appropriate linear combination of the generalized traces that we precomputed when constructing the number field.

If our number field element is

$$\alpha = a_0 + a_1\theta + \ldots + a_{n-1}\theta^{n-1}$$

then we have

$$\mathrm{Tr}(\alpha) = \sum_{k=0}^{n-1} a_k S_k,$$

where the S_k are the generalized traces.

This means that traces can be computed with a number of rational number operations that is linear in the degree of the number field.

Benchmarks

In the graphs in Figure 1, we show the speedup ANTIC achieves over Magma for multiplication of random number field elements in fields of various sizes, with coefficients of 10, 50, 100 and 1000 bits, respectively. We forced Magma to perform reduction and canonicalisation, so that we are not merely timing polynomial arithmetic, but number field arithmetic.

In Figures 2 and 3 we show the speedup over Magma for computing traces and norms, respectively, of random elements with coefficients of 10 bits in number fields of various degrees.

Comparison of trace and norm against Pari/GP is not practical because in order to use Pari's trace and norm functionality, one first needs to create number field objects, which take a very long time to generate.

Future work

The next items we plan to implement in ANTIC are characteristic and minimum polynomials.

Following that, we plan to add matrix and polynomial arithmetic over number fields, followed by ideal arithmetic, computing maximal and equation orders, class groups and unit groups.

Acknowledgement

Many thanks to Fredrik Johansson for assisting with the graphs in this article.

References

[1] Wieb Bosma, John Cannon, and Catherine Playout, *The Magma algebra system*. I. The user language, J. Symbolic Comput., 24 (1997), pp. 235 to 265, `http://magma.maths.usyd.edu.au`

[2] Claus Fieker, William Hart, *ANTIC*, `https://github.com/wbhart/antic/`

[3] William Hart, Fredrik Johansson, Sebastian Pancratz, et. al. *Flint*, `http://www.flintlib.org/`

[4] Fredrik Johansson, *Arb*, `http://fredrikj.net/arb/`

[5] The Pari Development Team, *Pari/GP*, `http://pari.math.u-bordeaux.fr/`

[6] Victor Shoup, *NTL: A Library for doing Number Theory*, `http://www.shoup.net/ntl/`

[7] William A. Stein et al., *Sage Mathematics Software*, The Sage Development Team, `http://www.sagemath.org`

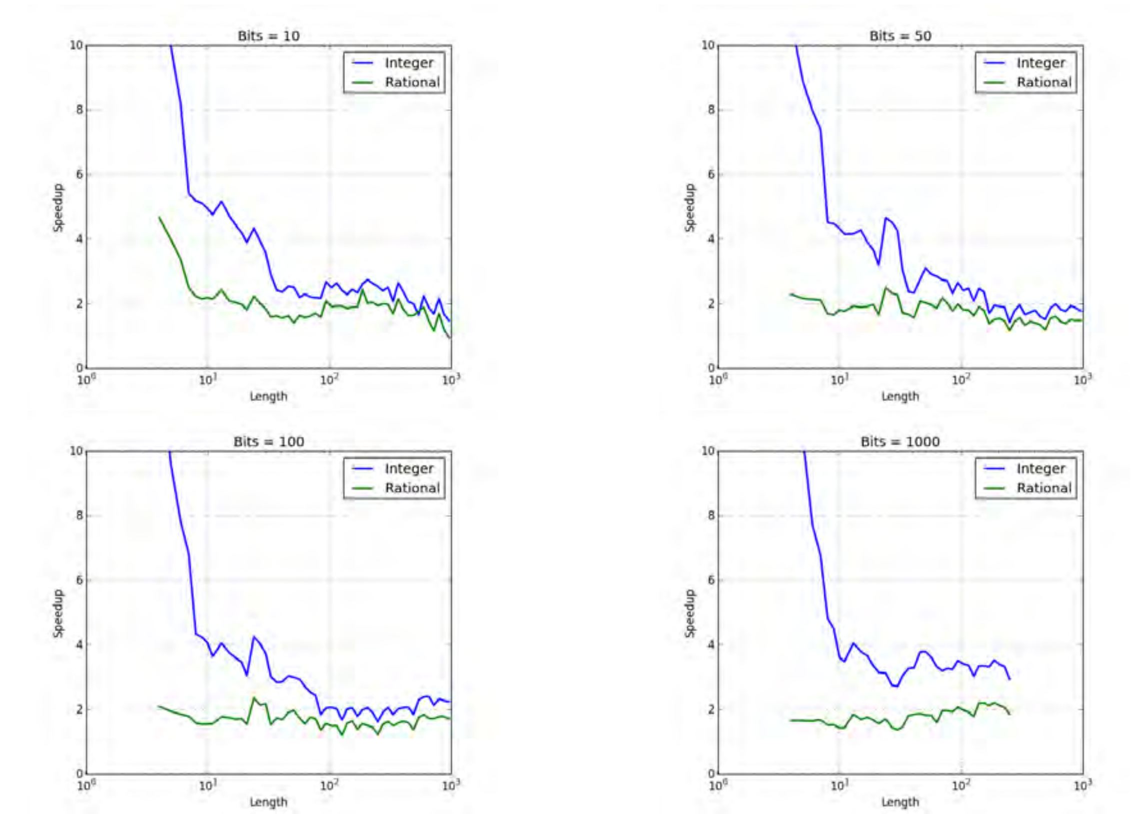

Figure 1: *Number field multiplication speedup vs Magma*

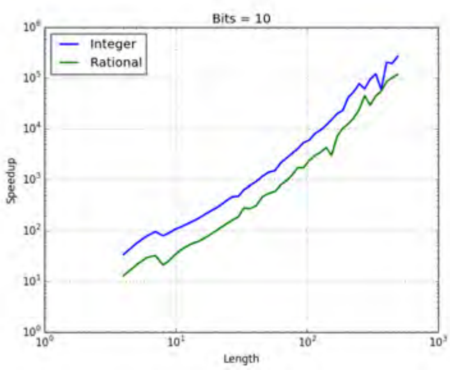

Figure 2: *Number field trace speedup*

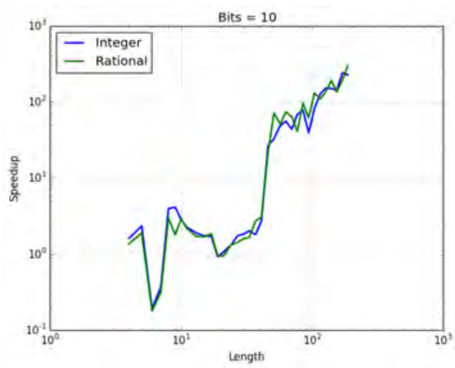

Figure 3: *Number field norm speedup*