FLINT: FAST LIBRARY FOR NUMBER THEORY

WILLIAM B. HART

FLINT is a pure C library for Number Theory which had its beginnings in 2007. The Computer Algebra landscape at the time was not too different than it is today. Many of the finest implementations of "core" algorithms for polynomial arithmetic and linear algebra were in proprietary systems and every implementation, whether open or closed, contained bugs!

In 2007 the Sage project [10] was becoming prominent and decent Open Source packages were being sought for possible inclusion in Sage.

At this time, Victor Shoup's NTL library [11] was used for polynomial arithmetic in Sage. This is a high quality C++ library that had been well tested over time. However, it has some issues. It is not threadsafe, it is no longer actively developed and it is much, much slower than the fastest proprietary code, which in 2007 was in the Magma system [2]. NTL was however, quite a bit faster than Pari [9] (also used extensively by Sage) for univariate polynomial arithmetic. FLINT started life as a pure C, threadsafe, fast version of NTL. Indeed it had originally been called "GMP-NTL".

The initial problem FLINT set out to solve was the one David Harvey (co-maintainer of FLINT at the time) had noticed, namely the enormous gap between the speed of the proprietary Magma implementation of polynomial arithmetic and the fastest open source implementation, which happened to be in NTL.

Magma was up to five times faster than NTL at multiplying univariate polynomials over the integers. However, we could see that NTL already had a sensible implementation of the best algorithm we knew (Schönhage-Strassen) and a variety of other alternatives with tuned cutoffs!

As a result of our experiences, we outlined the following goals for the FLINT project, which remain its goals today.

- Performance ($\geq$ best open or closed alternative)
- Thread safety
- Well tested (test function for each algorithm)
- Regularly profiled
- Written in C for portability and Open Source

The first step in beating Magma's polynomial multiplication was to figure out where it switched algorithms by timing it over a large range of input sizes. It was also possible to see the jumps in speed at power of two lengths due to Magma's FFT.

Our first breakthrough was the realisation that cache locality is absolutely critical to multiplication of large polynomials. Already a recursive implementation of the FLINT FFT gave a 30% speed up.

The next big improvement was a switch internally from GMP's mpz integer format [3] to its mpn format. The reduction in overhead was striking.

Finally, it took a lot of puzzling to figure out that for very small coefficients merely reading and writing polynomials was a bottleneck! We had to develop a data format that used less space per coefficient than an ordinary GMP mpz integer.

Since the release of FLINT 2 we have been using an integer format (fmpz) which takes a single machine word to hold a small integer. On overflow an mpz is automatically allocated and the machine word becomes a pointer not an immediate value. The upshot is that for a polynomial with many small coefficients, only one machine word is used per coefficient.

It took about a year to catch Magma's polynomial multiplication implementation. The support and encouragement we received from the Sage community was really important, most notably the support of William Stein. We began to realise just how much fun it was to implement fast core algorithms in C.

Since that time FLINT has undergone some radical changes. David Harvey wrote a very fast library (zn_poly [8]) for basic polynomial arithmetic over $\mathbb{Z}/n\mathbb{Z}$ for word sized n. This was incorporated into FLINT, and Sage switched from NTL to FLINT (currently version 1.5) for univariate polynomial arithmetic over $\mathbb{Z}$ and $\mathbb{Z}/n\mathbb{Z}$.

Despite the successes, some FLINT design decisions were suboptimal. We had an internal format for polynomials that consisted of a single large allocation of memory which was broken into equal sized blocks, one for each coefficient. The first word of each block specified the actual coefficient size within the block and the sign. The format was fast, but it had a number of significant disadvantages.

Firstly, every time even a single coefficient outgrew its block, the entire polynomial had to be reallocated. Secondly, if the polynomial had many coefficients, making just one of them large meant that the memory usage would skyrocket, possibly sending the machine into orbit. Thirdly, every single algorithm in FLINT needed to deal with not just the number of coefficients of a polynomial but also the maximum size. By the time division and GCD were implemented, this became unwieldy. It was the main source of bugs in FLINT at the development stage.

Perhaps more importantly, if the user wanted to manipulate an individual coefficient of a polynomial they had to use special functions which would handle resizing the whole polynomial. There was no FLINT integer module which could manipulate a coefficient of a polynomial or an entry of a matrix equally.

I then took the decision to rewrite FLINT from scratch.... all 80,000 lines of it!

At that time Andy Novocin began implementing the new algorithm Mark van Hoeij and he had developed [4] for factorization of polynomials over the integers. Of course this meant that FLINT 1 swelled another 40,000 lines!

Apart from solving the issues with the polynomial format, the FLINT 2 series also made some other changes. We wanted it to be really simple to contribute to. We developed a "magic build system". To add a new file to FLINT 2 no modification of the build system is required. The contributor simply drops the file into the relevant source directory and it automatically gets built as part of FLINT.

FLINT 2 also aims for code clarity above insane premature optimization. Anyone should be able to read, understand and modify FLINT 2 code.

FLINT 2 has gained two new regular contributors, Sebastian Pancratz and Fredrik Johansson. Sebastian notably added a p-adic module and polynomials over $\mathbb{Q}$. He also developed an amazing parser for turning the FLINT 2 text file documentation into a single beautiful pdf document. Fredrik has notably added arithmetic functions, power series functions and matrix modules over numerous base rings.

Surprisingly, many of the functions in FLINT 2 are actually much faster than the ones in FLINT 1 because of the improved design and the focus on simplicity!

At the time of writing, FLINT 2 has almost caught up with FLINT 1.5 functionality and has many totally new modules in addition. Here is a list of FLINT 2 modules:

Mathematica 7	Magma 2.17-5	Sage 4.7	Pari 2.4.3	FLINT 2.3
$f(f(x)) \pmod{x^{200}}$ for $f = x + 2x^2 + 3x^3 + \cdots$ over $\mathbb{Z}$				
3340 ms	30 ms	176 ms	100 ms	7.8 ms
reversion $f^{-1}(x) \pmod{x^{200}}$ for $f = x + 2x^2 + 3x^3 + \cdots$ over $\mathbb{Z}$				
12 s	100 ms	131 ms	—-	61 ms
$f(g(x)) \pmod{x^{200}}$ for $f(x) = \exp(x)$, $g(x) = x\log(1+x)$ over $\mathbb{Q}$				
57 s	120 ms	410 ms	2720 ms	140 ms
$f(f(x)) \pmod{x^{1000}}$ for $f(x) = x + 2x^2 + 3x^3 + \cdots$ over $\mathbb{Z}/(2^{63}+29)\mathbb{Z}$				
—-	770 ms	2960 ms	24.7 s	49 ms
$f^{-1}(x) \pmod{x^{400}}$ for $f = x + 2x^2 + 3x^3 + \cdots$ over $\mathbb{Z}/(2^{63}+29)\mathbb{Z}$				
—-	1530 ms	1680 ms	—-	33 ms

Table 1. Comparison of power series operations

Magma 2.17-5	Sage 4.6	Pari 2.4.3	FLINT 2.2
multiply 100× 100 matrices with 10 bit num/den over $\mathbb{Q}$			
320 ms	1480 ms	1164 ms	380 ms
solve $Ax = b$ for A (100× 100), b(100× 1) over Z/mZ, m 50 bits			
80 ms	770 ms	248 ms	1.65 ms
det of 20× 20 matrix over $Z[x]$, length 3 polys, 3 bit coeffs			
130 ms	256 ms	112 ms	16 ms

Table 2. Comparison of matrix operations

- ulong_extras - word sized unsigned integers (including modular algorithms, primality testing, factorization, etc.)
- fmpz - the new FLINT 2 multiprecision integer format
- fmpq - multiprecision rational numbers
- fmpz_poly - polynomials and power series over $\mathbb{Z}$
- fmpq_poly - polynomials and power series over $\mathbb{Q}$
- fmpz_poly_q - rational functions over $\mathbb{Q}$
- nmod_poly - polynomials and power series over $\mathbb{Z}/n\mathbb{Z}$ for word sized n
- padic - p-adic numbers
- fmpz_mat - matrices over $\mathbb{Z}$
- nmod_mat - matrices over $\mathbb{Z}/n\mathbb{Z}$ for word sized n
- fmpq_mat - matrices over $\mathbb{Q}$
- fmpz_poly_mat - matrices over $\mathbb{Z}[x]$
- arith - implementations of various arithmetic functions

There are also numerous experimental modules in the works including multivariate polynomials, LLL, polynomials over the reals and a generics module.

FLINT 2 contains some novel algorithms, e.g. when factoring word sized integers in a certain range we use a variant of Lehmer's algorithm called One Line Factor [5]. Over a certain range it beats a fast implementation of SQUFOF.

David Harvey invented some variants of Kronecker Segmentation for fast multiplication of polynomials over $\mathbb{Z}/n\mathbb{Z}$ for small n. He implemented these in his zn_poly library, which he generously allowed to be included in FLINT.

Andy Novocin and I also repurposed an algorithm of Brent for polynomial composition and we have a divide-and-conquer algorithm for division of polynomials [6] based on Mulders' algorithm.

FLINT was recently used for the calculation of all the congruent numbers up to 10^{12} [7] (subject to the BSD conjecture). This involved multiplication of power series having up to $10^{12}/8$ terms, on a small cluster. Gonzalo Tornaria contributed code for that project and we also reported the result in a press release [1].

To give an indication of performance, we include some timing samples above. Most of these benchmarks are courtesy of Fredrik Johansson.

Here is a small snippet of C code to demonstrate how one may use FLINT. It computes the determinant of the matrix $(ix + j)_{i,j=1..20} \in \mathrm{Mat}(\mathbb{Z}[x])$.

```
#include "fmpz_poly_mat.h"
...

unsigned long i, j;
fmpz_poly_t p;
fmpz_poly_init(p);
fmpz_poly_mat_t A;
fmpz_poly_mat_init(A, 20, 20);
for (i = 0; i < 20; i++) {
   for (j = 0; j < 20; j++) {
      fmpz_poly_set_coeff_ui(p, 1, i + 1);
      fmpz_poly_set_coeff_ui(p, 0, j + 1);
      fmpz_poly_set(fmpz_poly_mat_entry(A, i, j), p);
   }
}
fmpz_poly_mat_det(p, A);
fmpz_poly_print_pretty(p, "x");
```

Our immediate plans for the future of FLINT are to implement the remaining functionality from NTL that we don't have yet and to begin working on algebraic number fields.

The FLINT story is by no means complete – there are many episodes to come, and we are still very much enjoying its development. Of course new contributors are most welcome. See our website `http://www.flintlib.org/` or email one of the current developers for further details.

References

[1] American Institute of Mathematics, *A Trillion Triangles*, `http://www.aimath.org/news/congruentnumbers/`

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

[3] Torbjorn Granlund et al., *The GNU Multiple Precision Arithmetic Library*, `http://www.gmplib.org/`

[4] William Hart, Mark van Hoeij, Andrew Novocin, *Practical polynomial factoring in polynomial time*, Proceedings, 36th International Symposium on Symbolic and Algebraic Computation, San Jose, June 2011, pp 163–170

[5] William Hart, *A one line factoring algorithm*, (preprint)

[6] William Hart, Andrew Novocin, *Practical divide-and-conquer algorithms for polynomial arithmetic*, (preprint)

[7] William Hart, Gonzalo Tornaria, Mark Watkins, *Congruent number theta coefficients to* 10^{12}, proceedings Algorithmic Number Theory IX, Nancy, July 2010, LNCS 6197, pp. 186–200

[8] David Harvey, *zn_poly: A library for polynomial arithmetic*, `http://web.maths.unsw.edu.au/~davidharvey/code/zn_poly/`

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

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

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

E-mail address: `W.B.Hart@warwick.ac.uk`