Reduce Everything to Multiplication

Allan Steel
University of Sydney
(Magma Project)

Asymptotically-Fast Algorithms

One of my ongoing research aims in Magma has been to develop algorithms for fundamental problems in Computer Algebra which:

(1) Have the best theoretical complexity;

(2) Work very well in practice (i.e., beat classical algorithms within practical ranges).

This seems to be achieved now in Magma for a wide range of algorithms for exact algebraic computations with:

(1) Integers

(2) Polynomials

(3) Matrices

Integer Multiplication

Fast Fourier Transform (FFT)-based integer multiplication is the **critical basis** of all asymptotically-fast polynomial algebra.

Schönhage-Strassen integer multiplication: compute in

$$R = \mathbf{Z}_{2^{2^k}+1}$$

so 2 is a $2^{2^{k+1}}$-th root of unity in R.

Complexity for multiplying n-bit integers: $n \log(n) \log(\log(n))$.

Multiply two million-decimal-digit integers on 2.4GHz Opteron: 0.06s (17 times a sec).

FFT Polynomial Multiplication

(1) Kronecker-Schönhage substitution/Segmentation: map to integer mult (evaluate at suitable power of 2).

(2) Direct Schönhage-Strassen (polynomial) FFT multiplication.

Segmentation is a common approach and always better when degree $>>$ coefficents bit-length.

Direct S-S better when coefficient bit-length roughly $\geq$ $1/2$ degree.

Mult 2 polys, each degree 1000 and 1000-bit coefficients:
Segmentation: 0.0307s (2^{21}-bit integers), Direct S-S: 0.0125s.

Reduce arithmetic to Multiplication

Reduce (univariate) operations to multiplication:

- Division
- GCD
- Resultant
- Rational reconstruction

Thus FFT-complexity possible for all these algorithms (possibly with extra log factor), and this works well in practice, so is not just theoretical.

Univariate Factorization Over Finite Fields

Von zur Gathen/Kaltofen/Shoup algorithm currently best algorithm. Shoup's critical components to make it fast:

- Perform divisions by **multiplying** by inverse of modulus.
  - Pre-compute inverse of modulus and store FFT **transform**.
  - Use two short **products** and a wrapped convolution.
- Brent-Kung modular evaluation algorithm (1978) to compute x^{q^i} mod f quickly.

Factorization Challenge over Finite Fields

J. von zur Gathen, SIGSAM Bulletin (April 1992).

Let p_n be the first prime $> \pi \cdot 2^n$ (thus has $n+1$ bits).

Factor $x^n + x + 1$ over $\mathbf{F}_{p_n}$.

n		
200	30h	1993: M. Monagan, Maple, DEC 3100
300	110h	1994: C. Playoust/A. Steel, Magma, SunMP670
500	63h	1995: P. Zimmermann, MuPAD, Sun Sparc-10
1024	51h	1995: V. Shoup, 20-MIPS Sun 4
200	0.6s	2006: Magma V2.13 on 2.4Ghz Opteron
300	4.9s	
500	7.6s	
1024	102.8s	
2048	1291.0s	
4096	20286.7s	

Bivariate Factorization

Factorization in $\mathbf{F}_q[x, y]$ (with M. van Hoeij et al.):

- Use van Hoeij idea to collect relations based on traces.

- Direct linear algebra (no LLL or approximation needed).

- Time dominated by Hensel lifting over power series. Involves multiplying in $\mathbf{F}_{q^k}[[y]][x]$.

Bivariate Factorization Example

Factor $f \in \mathbf{F}_5[x,t] =$

$$x^{78125} + x^{15625}t^{2750} + x^{15625}t^{2600} + 4x^{3125}t^{3750} + 4x^{3125}t^{3150} + 2x^{3125}t^{3000} + 4x^{3125} + 4x^{625}t^{3500} + 3x^{625}t^{3350} + 2x^{625}t^{3200} + 4x^{625}t^{2750} + 4x^{625}t^{2600} + x^{125}t^{3300} + x^{125}t^{3270} + 2x^{125}t^{3150} + x^{125}t^{3120} + 3x^{125}t^{3000} + x^{25}t^{3350} + x^{25}t^{3230} + x^{25}t^{3224} + 3x^{25}t^{3200} + 4x^{5}t^{3270} + 4x^{5}t^{3120} + 4xt^{3224}.$$

Due to G. Malle (Dickson Groups). Extremely sparse: $(78125, 3750)$, 24 terms. Factors in 25 minutes (2GHz Opteron, 590MB). Factors x-degrees: 1, 15624, 15750, 15750, 15500, 15500 (most about 6000 terms).

One Hensel step: multiply polys in $\mathbf{F}_{5^3}[[t]][x]$ by mapping to integers. Integers each have about **45 million decimal digits** and multiplied in 6.7 seconds. So FFT-based even over small finite field!

Matrix Multiplication

Consider multiplication of a pair of 2 by 2 matrices:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Classical Method: 8 multiplications, 4 additions.

$$c_{11} = a_{11}b_{11} + a_{12}b_{21},$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22},$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21},$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22}.$$

General Complexity: $O(n^3)$.

Strassen's Method (1969): 7 multiplications, 18 additions or subtractions (18 improved to 15 by Winograd).

$$x_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22}),$$

$$x_2 = (a_{21} + a_{22}) \cdot b_{11},$$

$$x_3 = a_{11} \cdot (b_{12} - b_{22}),$$

$$x_4 = a_{22} \cdot (b_{21} - b_{11}),$$

$$x_5 = (a_{11} + a_{12}) \cdot b_{22},$$

$$x_6 = (a_{21} - a_{11}) \cdot (b_{11} + b_{12}),$$

$$x_7 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22}),$$

$$c_{11} = x_1 + x_4 - x_5 + x_7,$$

$$c_{12} = x_3 + x_5,$$

$$c_{21} = x_2 + x_4,$$

$$c_{22} = x_1 + x_3 - x_2 + x_6.$$

- Strassen's method leads to a **recursive algorithm** for matrix multiplication – commutativity is NOT used!

- Implementation is very complicated for non-square matrices, and for dimensions which are not powers of 2.

- **Much** more difficult to implement than Karatsuba (the simplest asymptotically-fast methods for their respective problems).

Complexity: $O(n^{\log_2 7}) \approx O(n^{2.807})$.

Strassen IS Applicable In Practice

- For rings for which no fast modular algorithm is available: dim 2.

- Bit length of entries very much larger than dimension: dim 2.

- Mod p, where residues are represented via double floating-point numbers so ATLAS (Automatically Tuned Linear Algebra Software) can be used: dim 500.

- Small prime finite field: dim 1000 (matrices of dim ≥ 10000 not unusual).

- Non-prime finite field mapping technique (see below): dim 125.

Also sprach der Meister:

Donald Knuth, *AOCP*, Vol. 2, **3rd Ed., 1997** (my emphasis):

> These **theoretical** results [Strassen's method] are quite striking, but from a practical standpoint they are of **little use** because n must be very large. . . [p. 501].

> Richard Brent (1970) estimated that Strassen's scheme would not begin to excel over Winograd's [cubic complexity] scheme until $n \approx$ **250**; and such **enormous** matrices **rarely** occur in practice unless they are very sparse, when other techniques apply [p. 501].

> **Of course** such asymptotically "fast" multiplication is **strictly of theoretical interest** [p. 718; added in 1997 edition!!!].

In response:

Erich Kaltofen, *Challenges of Symbolic Algebra* (EECAD 1998):

> Open Problem 7: Convince Donald Knuth that these asymptotically fast methods are of practical value. If he pays you \$2.56 for this technical error, you have solved this problem.

Allan Steel (2000):

> For quite practical sizes, Strassen's method is **streets ahead** of the classical method. *[German etymological pun]*

Modular Matrix Multiplication

Multiply a pair of n by n matrices, with all entries being integers of up to k bits each.

$M(n)$: complexity of the matrix multiplication algorithm (arithmetic operations).

Assume k is small enough that only classical integer multiplication is applicable (true for k up to several hundreds).

Classical method: $M(n)O(k^2)$ bit operations.

Modular method: $M(n)O(k) + O(n^2)O(k^2)$ bit operations.

Modular method: $M(n)O(k) + O(n^2)O(k^2)$ bit operations.

- Reduce the input modulo several primes, multiply each such pair modularly and use Chinese remaindering for the result. Use ATLAS for the modular computations.

- To multiply matrices over $\mathbf{F}_p$, large p: multiply over $\mathbf{Z}$ by above, then mod by p at end.

- If $n >> k$, modular method is practically linear in k.

Recursive Echelonization

Reduces to matrix multiplication so that the complexity is that of multiplication.

V. Strassen sketched such an algorithm for computing the **inverse** of a square matrix, assuming some conditions:

> *Gaussian Elimination is not Optimal* (Numer. Math., 1969).

This paper also had the original fast multiplication formulae.

Recursive Echelonization Examples

Dense random matrices over $\mathbf{F}_p$, where $1024 \cdot p^2 \leq 2^{53}$ (2.4GHz Opteron), so ATLAS applicable.

n	$A \cdot B$	Rec Det(A)	Rec A^{-1}	Classical Det(A)	Classical A^{-1}
512	0.090	0.070	0.180	0.140	0.670
1024	0.650	0.450	1.070	1.230	5.540
2048	4.280	2.900	6.820	9.570	45.840
4096	29.470	18.860	44.550	75.250	350.680
8192	215.80	121.75	296.03	621.370	2839.070

Computations in the Finite Field $\mathbf{F}_q$

Finite field $\mathbf{F}_q \cong \mathbf{F}_p[\alpha]/\langle f(\alpha)\rangle$, $q = p^d$, primitive element α.

Zech logarithms: i represents α^i; $q-1$ represents 0.

Multiplication/inversion/division easy.

Addition via $a + b = a(1 + b/a)$; store successor table for:

$$\alpha^{s(i)} = \alpha^i + 1,$$

which takes $O(q)$ bytes.

Fast Matrix Multiplication over $\mathbf{F}_q$

Multiply $n \times n$ matrix over $\mathbf{F}_q \cong \mathbf{F}_p[\alpha]/\langle f(\alpha)\rangle$, $q = p^d$.

Find smallest $\beta = 2^k$ with $nd(p-1)^2 < \beta$.

Interpret entries as polynomials in $\mathbf{F}_p[\alpha]$ and map element via $\alpha \mapsto \beta$ to yield integer. Multiply the integral matrices and map entry e back thus:

- Write e in base β, giving polynomial in $\mathbf{Z}[x]$ and reduce coefficients mod p.

- Form low l and high h elements from the blocks of d coefficients, mapping back to Zech form.

- Result is $l + \alpha^d h$.

Multiply Matrices over $\mathbf{F}_{5^2}$

$q = 5^2$. Can use C doubles (ATLAS) for mapped integral product.

Size	Old Mult	New Mult	Speed-up	Old Inverse	New Inverse	Speed-up
100	0.006	0.001	6.1	0.008	0.005	1.8
200	0.041	0.006	7.0	0.061	0.020	3.0
500	0.612	0.068	9.0	0.916	0.160	5.7
1000	4.820	0.510	9.4	7.290	0.870	8.3
2000	38.770	3.530	10.9	58.590	5.540	10.5
4000	304.150	25.000	12.1	472.500	36.760	12.8

n	$nd(p-1)^2$	β	Max coeff
100	3200	2^{12}	$2^{34.6}$
1000	32000	2^{15}	$2^{44.0}$
4000	128000	2^{17}	$2^{51.0}$

Multiply Matrices over $\mathbf{F}_{23^5}$

$q = 23^5 = 6436343$. Mapped integral product computed using modular CRT algorithm for large integers.

Size	Old Mult	New Mult	Speed-up	Old Inverse	New Inverse	Speed-up
100	0.84	0.02	33.6	0.96	0.14	6.8
200	6.69	0.12	53.5	7.23	0.66	10.9
500	103.80	1.22	85.1	108.55	4.99	21.7
1000	828.68	7.13	116.2	850.19	24.32	34.9
2000	6615.68	49.07	134.8	6764.02	131.86	51.3
4000	52799.74	366.84	143.9	53807.77	966.75	55.65

n	$nd(p-1)^2$	β	Max coeff	# primes
100	242000	2^{18}	$2^{186.4}$	9
1000	2420000	2^{22}	$2^{228.3}$	11
4000	9680000	2^{24}	$2^{250.3}$	12

Brent-Kung Modular Composition (1978)

Given polynomials $f, g, h \in K[x]$, K field, degrees $\leq n$: compute $f(g)$ mod h.

Baby-step/giant-step technique. Set $s = \lfloor \sqrt{n} \rfloor, t = \lceil n/s \rceil$.

Compute $g_j = g^j$ mod h for $j = 1, \ldots, s$, by successively multiplying by g and reducing mod h.

Divide f into t blocks of s consecutive coefficients.

Set e to zero.

For each block $C_i = (c_{i,1}, \ldots, c_{i,s})$ for $i = t, t-1, \ldots 1$, compute the linear combination r of the g_j given by C_i, and set e to $e \cdot f + r$.

At the end, $e = f(g) \bmod h$.

Thus the cost is approximately $2 \cdot \sqrt{n}$ modular products, instead of n modular products (using standard Horner's rule).

Matrix version

Write each $g_j = g^j \bmod h$ (for $j = 1, \ldots, s$) as a vector and form matrix B from these vectors: $s \times n$.

When applying, make each block $C_i = (c_{i,1}, \ldots, c_{i,s})$ (for $i = t, t-1, \ldots 1$) a vector and form matrix A from these vectors: $t \times s$.

Multiply A by B: $(t \times s) \times (s \times n)$.

Fast matrix multiplication applicable.

Application: Return to Factoring

Factor polynomial f over a finite field (Cantor/Zassenhaus, von zur Gathen/Kaltofen/Shoup):

First compute $g = x^q \bmod f$, where q is the size of the field.

One then needs $x^{q^i} \bmod f$ for $i = 2, \ldots$.

Instead of successively raising g to the power of q mod f and repeating, one can instead compute $g_2 = g(g)$, $g_3 = g_2(g)$, etc., all done mod f.

Each of these compositions are efficiently done via the Brent-Kung algorithm.

Factoring comparison

Factor $x^n + x + 1$ over $\mathbf{F}_{23^5}$.

n	Direct B-K	Direct Total	Mat B-K	Mat Total
1000	10.4	15.0	3.8	8.4
2000	57.9	81.9	18.2	43.2
5000	597.3	734.2	173.5	310.4
10000	4119.4	4784.5	916.1	1581.5