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

\[ R = \mathbb{Z}_{2^{2^k+1}} \]

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 $\gg$ coefficients 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


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

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

<table>
<thead>
<tr>
<th>$n$</th>
<th>Time</th>
<th>Year/Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>30h</td>
<td>1993: M. Monagan, Maple, DEC 3100</td>
</tr>
<tr>
<td>300</td>
<td>110h</td>
<td>1994: C. Playoust/A. Steel, Magma, SunMP670</td>
</tr>
<tr>
<td>500</td>
<td>63h</td>
<td>1995: P. Zimmermann, MuPAD, Sun Sparc-10</td>
</tr>
<tr>
<td>1024</td>
<td>51h</td>
<td>1995: V. Shoup, 20-MIPS Sun 4</td>
</tr>
<tr>
<td>200</td>
<td>0.6s</td>
<td>2006: Magma V2.13 on 2.4Ghz Opteran</td>
</tr>
<tr>
<td>300</td>
<td>4.9s</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>7.6s</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>102.8s</td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td>1291.0s</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>20286.7s</td>
<td></td>
</tr>
</tbody>
</table>
Bivariate Factorization

Factorization in $\mathbb{F}_q[x, y]$ (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 $\mathbb{F}_{q^k}[[y]][x]$. 
Bivariate Factorization Example

Factor \( f \in \mathbb{F}_5[x, t] = \)
\[
78125 x^7 + 15625 x^{15625} t^{2750} + 15625 x^{15625} t^{2600} + 4 x^{3125} t^{3750} + 4 x^{3125} t^{3150} + 2 x^{3125} t^{3000} + 4 x^{3125} t^{3500} + 3 x^{625} t^{3350} + 2 x^{625} t^{3200} + 4 x^{625} t^{2750} + 4 x^{625} t^{2600} + 125 x^{125} t^{3300} + 125 x^{125} t^{3270} + 125 x^{125} t^{3120} + 3 x^{125} t^{3000} + 25 x^{25} t^{3350} + 25 x^{25} t^{3230} + 25 x^{25} t^{3200} + 4 x^{5} t^{3270} + 4 x^{5} t^{3120} + 4 x t^{324}.
\]

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 \( \mathbb{F}_5[[t]][x] \) by mapping to integers. Integers each have about \textbf{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 $\geq 10000$ not unusual).

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


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 \( \mathbb{F}_p \), large \( p \): multiply over \( \mathbb{Z} \) by above, then mod by \( p \) at end.

- If \( n \gg k \), 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:


This paper also had the original fast multiplication formulae.
Recursive Echelonization Examples

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

<table>
<thead>
<tr>
<th>n</th>
<th>$A \cdot B$</th>
<th>Rec Det$(A)$</th>
<th>Rec $A^{-1}$</th>
<th>Classical Det$(A)$</th>
<th>Classical $A^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.090</td>
<td>0.070</td>
<td>0.180</td>
<td>0.140</td>
<td>0.670</td>
</tr>
<tr>
<td>1024</td>
<td>0.650</td>
<td>0.450</td>
<td>1.070</td>
<td>1.230</td>
<td>5.540</td>
</tr>
<tr>
<td>2048</td>
<td>4.280</td>
<td>2.900</td>
<td>6.820</td>
<td>9.570</td>
<td>45.840</td>
</tr>
<tr>
<td>4096</td>
<td>29.470</td>
<td>18.860</td>
<td>44.550</td>
<td>75.250</td>
<td>350.680</td>
</tr>
<tr>
<td>8192</td>
<td>215.80</td>
<td>121.75</td>
<td>296.03</td>
<td>621.370</td>
<td>2839.070</td>
</tr>
</tbody>
</table>
Computations in the Finite Field $F_q$

Finite field $F_q \cong F_p[\alpha]/\langle f(\alpha) \rangle$, $q = p^d$, primitive element $\alpha$.

Zech logarithms: $i$ represents $\alpha^i$; $q - 1$ 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 $\mathbb{F}_q$

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

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

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

- Write $e$ in base $\beta$, giving polynomial in $\mathbb{Z}[x]$ 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 $F_{5^2}$

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

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

<table>
<thead>
<tr>
<th>n</th>
<th>$nd(p - 1)^2$</th>
<th>$\beta$</th>
<th>Max coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3200</td>
<td>$2^{12}$</td>
<td>$2^{34.6}$</td>
</tr>
<tr>
<td>1000</td>
<td>32000</td>
<td>$2^{15}$</td>
<td>$2^{44.0}$</td>
</tr>
<tr>
<td>4000</td>
<td>128000</td>
<td>$2^{17}$</td>
<td>$2^{51.0}$</td>
</tr>
</tbody>
</table>
Multiply Matrices over $\mathbf{F}_{23^5}$

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

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

<table>
<thead>
<tr>
<th>n</th>
<th>$nd(p-1)^2$</th>
<th>$\beta$</th>
<th>Max coeff</th>
<th># primes</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>242000</td>
<td>$2^{18}$</td>
<td>$2^{186.4}$</td>
<td>9</td>
</tr>
<tr>
<td>1000</td>
<td>2420000</td>
<td>$2^{22}$</td>
<td>$2^{228.3}$</td>
<td>11</td>
</tr>
<tr>
<td>4000</td>
<td>9680000</td>
<td>$2^{24}$</td>
<td>$2^{250.3}$</td>
<td>12</td>
</tr>
</tbody>
</table>
Brent-Kung Modular Composition (1978)

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

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

Compute $g_j = g^j \text{ mod } h$ 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, $e = f(g) \mod h$.

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 \( g_j = g^j \) mod \( h \) (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 $g = x^q \mod f$, where $q$ is the size of the field.

One then needs $x^{q^i} \mod f$ 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 $\mathbb{F}_{23^5}$.

<table>
<thead>
<tr>
<th>n</th>
<th>Direct</th>
<th>Direct</th>
<th>Mat</th>
<th>Mat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B-K</td>
<td>Total</td>
<td>B-K</td>
<td>Total</td>
</tr>
<tr>
<td>1000</td>
<td>10.4</td>
<td>15.0</td>
<td>3.8</td>
<td>8.4</td>
</tr>
<tr>
<td>2000</td>
<td>57.9</td>
<td>81.9</td>
<td>18.2</td>
<td>43.2</td>
</tr>
<tr>
<td>5000</td>
<td>597.3</td>
<td>734.2</td>
<td>173.5</td>
<td>310.4</td>
</tr>
<tr>
<td>10000</td>
<td>4119.4</td>
<td>4784.5</td>
<td>916.1</td>
<td>1581.5</td>
</tr>
</tbody>
</table>