Faster polynomial multiplication via multipoint Kronecker substitution

David Harvey, New York University

5th February 2009

Kronecker substitution

KS = an algorithm for multiplying polynomials in $\mathbf{Z}[x]$.

Example:

$$f = 41x^3 + 49x^2 + 38x + 29, \qquad g = 19x^3 + 23x^2 + 46x + 21.$$

To find $h = fg$, evaluate

$$f(10^4) = 41004900380029, \qquad g(10^4) = 19002300460021$$

by 'packing' coefficients together. Then

$$h(10^4) = f(10^4)g(10^4) = 779187437354540344421320609.$$

Coefficients of f and g are < 50, so coefficients of h are $< 4 \cdot 50^2 = 10^4$. Can 'unpack' $h(10^4)$ to obtain

$$h = 779x^6 + 1874x^5 + 3735x^4 + 4540x^3 + 3444x^2 + 2132x + 609.$$

Kronecker substitution

Notes:

- Same idea reduces multiplication in $R[x, y]$ to multiplication in $R[x]$ for any ring R, via $y \mapsto x^n$ for large enough n (Kronecker 1882).
- Application to arithmetic in $\mathbf{Z}[x]$ suggested by Schönhage (1982).
- On real hardware, use a power of 2, not 10. We assume packing and unpacking is linear time.

Kronecker substitution

Advantages of KS over ‘direct’ multiplication algorithms:

- If coefficients are small relative to machine word size, makes more efficient use of hardware multiply instruction.
- Places burden of optimisation on existing libraries like GMP (GNU Multiple Precision Arithmetic Library) — already obscenely optimised for huge variety of platforms.

Examples of real implementations:

- The Magma computer algebra package (popular in number theory and arithmetic geometry) uses KS for arithmetic in $\mathbf{Z}[x]$ and $(\mathbf{Z}/n\mathbf{Z})[x]$ when coefficients are small.
- NTL uses KS to reduce arithmetic in $\mathrm{GF}(p^n)[x]$ to arithmetic in $\mathrm{GF}(p)[x]$.

Kronecker substitution

What is the running time?

Suppose f, g have (non-negative) coefficients with c bits.

Suppose $\operatorname{len} f = \operatorname{len} g = n$ (i.e. they have degree $n - 1$).

Coefficients of $h = fg$ are bounded by $2^{2c} n$, so suffices to evaluate at 2^b where $b = 2c + \lceil \log_2 n \rceil$.

Running time is therefore $M(nb) + O(nb)$, where

- $M(k)$ = time to multiply k-bit integers,
- $O(nb)$ is the linear-time packing/unpacking cost.

KS2 algorithm

Idea: evaluate at *several* (carefully selected!) points, thereby reducing to several smaller integer multiplications.

Example (in base 10):

$$f = 41x^3 + 49x^2 + 38x + 29, \qquad g = 19x^3 + 23x^2 + 46x + 21.$$

Then

$$\begin{aligned} f(10^2) &= 41493829, & g(10^2) &= 19234621, \\ f(-10^2) &= -40513771, & g(-10^2) &= -18774579. \end{aligned}$$

Packed with *alternating signs* — still linear time.

Two half-sized integer multiplications:

$$\begin{aligned} h(10^2) &= f(10^2)g(10^2) = 798118074653809, \\ h(-10^2) &= f(-10^2)g(-10^2) = 760628994227409. \end{aligned}$$

KS2 algorithm

Problem: coefficients of h *overlap*, in both $h(10^2)$ and $h(-10^2)$:

$$\begin{array}{rr} 779373534440609 & \\ 187445402132 & + \\ \hline 798118074653809 & \\ = h(10^2) & \end{array} \qquad \begin{array}{rr} 779373534440609 & \\ 187445402132 & - \\ \hline 760628994227409 & \\ = h(-10^2) & \end{array}$$

Solution: if $h(x) = h^0(x^2) + xh^1(x^2)$, then

$$h^0(10^4) = \tfrac{1}{2}(h(10^2) + h(-10^2)) = 779373534440609$$
$$10^2 h^1(10^4) = \tfrac{1}{2}(h(10^2) - h(-10^2)) = 18744540213200$$

Unpacking is still linear time.

KS2 algorithm

What's the point?

Running time is about $2M(nb/2) + O(nb)$, compared to $M(nb) + O(nb)$ for standard KS.

Assume that $M(k) = O(k^\alpha)$. Then

$$\frac{M(nb)}{2M(nb/2)} = \frac{(nb)^\alpha}{2(nb/2)^\alpha} = 2^{\alpha-1}.$$

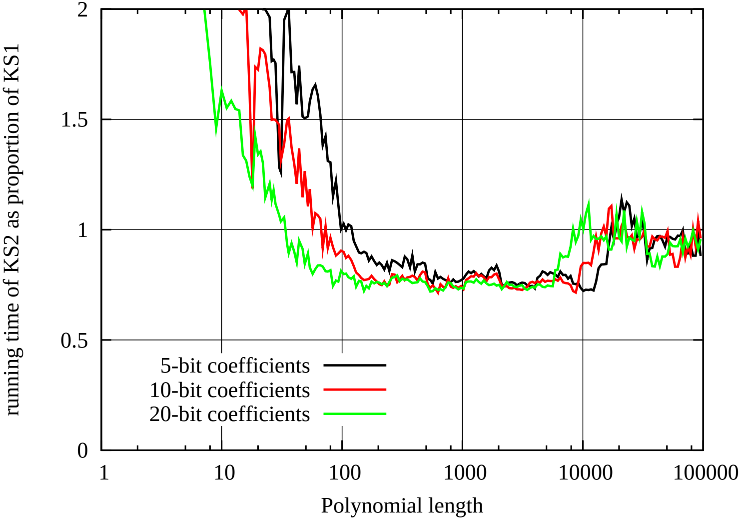
For classical multiplication, $\alpha = 2$. Expect 2x speedup.

For Karatsuba multiplication, $\alpha \approx 1.58$. Expect 1.5x speedup.

In practice, the linear terms get in the way!!

For FFT multiplication, $M(k) \sim k \log k$. No constant speedup expected; but perhaps some savings from better memory locality.

KS2 vs KS1 (64-bit, Core 2 Duo)



KS3 algorithm

Another set of points to try...

$$f = 41x^3 + 49x^2 + 38x + 29, \qquad g = 19x^3 + 23x^2 + 46x + 21.$$

Then

$$\begin{aligned} f(10^2) &= 41493829, & g(10^2) &= 19234621, \\ 10^6 f(10^{-2}) &= 29384941, & 10^6 g(10^{-2}) &= 21462319. \end{aligned}$$

Packed in *reversed order* — still linear time.

Two half-sized integer multiplications:

$$\begin{aligned} h(10^2) &= & f(10^2)g(10^2) &= 798118074653809, \\ 10^{12}h(10^{-2}) &= & 10^6 f(10^{-2})10^6 g(10^{-2}) &= 630668977538179. \end{aligned}$$

KS3 algorithm

$$\begin{array}{r}
0779 \\
1874 \\
3735 \\
4540 \\
3444 \\
2132 \\
0609 \quad + \\
\hline
798118074653809 \\
= h(10^2)
\end{array}
\qquad
\begin{array}{r}
0609 \\
2132 \\
3444 \\
4540 \\
3735 \\
1874 \\
0779 \quad + \\
\hline
630668977538179 \\
= 10^{12} h(10^{-2})
\end{array}$$

Problem: coefficients of h overlap. How to reconstruct h?

Let $h = h_6 x^6 + h_5 x^5 + h_4 x^4 + h_3 x^3 + h_2 x^2 + h_1 x + h_0$.

KS3 algorithm

$$
\begin{array}{r}
0779 \\
1874 \\
3735 \\
4540 \\
3444 \\
2132 \\
0609 \quad + \\
\hline
798118074653809
\end{array}
\qquad
\begin{array}{r}
0609 \\
2132 \\
3444 \\
4540 \\
3735 \\
1874 \\
07\color{red}{79} \quad + \\
\hline
6306689775381\color{red}{79}
\end{array}
$$

We can recover the *bottom half* of h_6 from the *lowest base-100 digit* of $10^{12}h(10^{-2})$, since there is no overlap there.

Where is the *top half* of h_6?

KS3 algorithm

$$\begin{array}{rr}
\color{red}{07}79 & \\
1874 & \\
3735 & \\
4540 & \\
3444 & \\
2132 & \\
0609 & + \\
\hline
\color{red}{7}98118074653809 &
\end{array}
\qquad
\begin{array}{rr}
0609 & \\
2132 & \\
3444 & \\
4540 & \\
3735 & \\
1874 & \\
07\color{red}{79} & + \\
\hline
630668977538\color{red}{179} &
\end{array}$$

The top half of h_6 is located in the *highest base-100 digit* of $h(10^2)$.

But hang on... couldn't there be a carry from $79 + 18$?

KS3 algorithm

$$\begin{array}{r} 0779 \\ 1874 \\ 3735 \\ 4540 \\ 3444 \\ 2132 \\ 0609 \quad + \\ \hline 798118074653809 \end{array} \qquad \begin{array}{r} 0609 \\ 2132 \\ 3444 \\ 4540 \\ 3735 \\ 1874 \\ 0779 \quad + \\ \hline 630668977538179 \end{array}$$

NO: because $79 < 98$.

(Strictly speaking, we also need to know that $18 < 99$, otherwise we could get burned by a carry propagating from further down, e.g. from $74 + 37$. I'll return to this later.)

KS3 algorithm

$$
\begin{array}{r}
1874 \\
3735 \\
4540 \\
3444 \\
2132 \\
0609 \quad + \\
\hline
19118074653809
\end{array}
\qquad
\begin{array}{l}
0609 \\
2132 \\
3444 \\
4540 \\
3735 \\
1874 \\
+ \\
\hline
6306689775374
\end{array}
$$

Therefore we completely recover $h_6 = 779$, and we can subtract it from the appropriate location in both sums.

KS3 algorithm

$$
\begin{array}{r}
1874 \\
3735 \\
4540 \\
3444 \\
2132 \\
0609 \quad + \\
\hline
19118074653809
\end{array}
\qquad
\begin{array}{l}
0609 \\
2132 \\
3444 \\
4540 \\
3735 \\
1874 \\
 + \\
\hline
6306689775374
\end{array}
$$

Do the same thing again:

Bottom half of h_5 is 74.

This time there *was* a carry, because $11 < 74$.

Therefore top half of h_5 is $19 - 1 = 18$. Subtract h_5 and repeat!

KS3 algorithm

An example where the 'bogus carry propagation' problem occurs: both

$$h(x) = 9901x^2 + 9901x \qquad \text{and} \qquad h(x) = 100x^3 + 100$$

satisfy

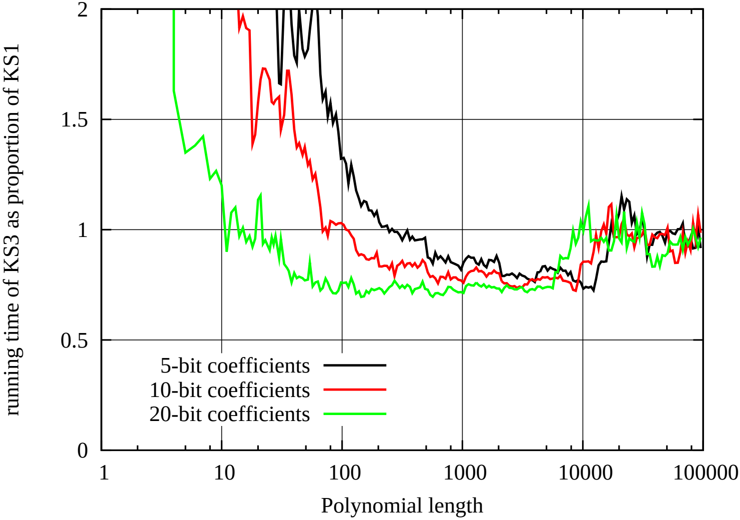
$$h(10^2) = 10^{12} h(10^{-2}) = 100000100.$$

We can protect against this by insisting that the coefficients of h are bounded by 9899 instead of 9999. This doesn't materially affect the applicability of the algorithm.

KS3 algorithm

This ‘unpacking’ algorithm runs in linear time, so we obtain the same estimate $2M(nb/2) + O(nb)$ for the running time of KS3.

KS3 vs KS1



running time of KS3 as proportion of KS1
2
1.5
1
0.5
0
1
10
100
1000
10000
100000
Polynomial length
5-bit coefficients
10-bit coefficients
20-bit coefficients

KS4 algorithm

Key ideas of KS2 and KS3 are orthogonal. Let's try *four* points:

$$f = 41x^3 + 49x^2 + 38x + 29, \qquad g = 19x^3 + 23x^2 + 46x + 21.$$

$$\begin{aligned} f(10) &= 46309, & g(10) &= 21781, \\ f(-10) &= -36451, & g(-10) &= -17139, \\ 10^3 f(10^{-1}) &= 33331, & 10^3 g(10^{-1}) &= 25849, \\ 10^3 f(-10^{-1}) &= 25649, & 10^3 g(-10^{-1}) &= 16611. \end{aligned}$$

Notice there is now even overlap in the evaluation phase, e.g. for $f(-10)$ we have

$$\begin{array}{rr} 4929 & \\ 4138 & - \\ \hline -36451 & \end{array}$$

KS4 algorithm

Leads to four multiplications of one fourth the size:

$$\begin{aligned}
h(10) &= & f(10)g(10) &= 1008656329 \\
h(-10) &= & f(-10)g(-10) &= 624733689 \\
10^6 h(10^{-1}) &= & 10^3 f(10^{-1}) 10^3 g(10^{-1}) &= 861573019 \\
10^6 h(-10^{-1}) &= & 10^3 f(-10^{-1}) 10^3 g(-10^{-1}) &= 426055539
\end{aligned}$$

Then:

$$\begin{array}{c}
\left.\begin{array}{r} h(10) \\ h(-10) \end{array}\right\} \xrightarrow{\text{KS2}} \left\{\begin{array}{l} h^0(10^2) \\ h^1(10^2) \end{array}\right. \\
\left.\begin{array}{r} h(10^{-1}) \\ h(-10^{-1}) \end{array}\right\} \xrightarrow{\text{KS2}} \left\{\begin{array}{l} h^0(10^{-2}) \\ h^1(10^{-2}) \end{array}\right.
\end{array}
\quad \text{and} \quad
\begin{array}{c}
\left.\begin{array}{r} h^0(10^2) \\ h^0(10^{-2}) \end{array}\right\} \xrightarrow{\text{KS3}} h^0(x) \\
\left.\begin{array}{r} h^1(10^2) \\ h^1(10^{-2}) \end{array}\right\} \xrightarrow{\text{KS3}} h^1(x)
\end{array}$$

KS4 algorithm

Running time is now $4M(nb/4) + O(nb)$.

If $M(k) = O(k^\alpha)$, then

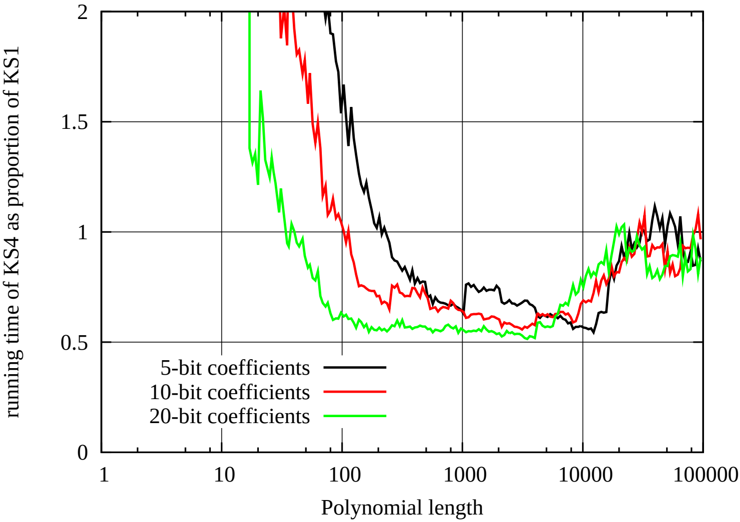
$$\frac{M(nb)}{4M(nb/4)} = 4^{\alpha-1}.$$

Classical multiplication $\implies$ 4x speedup over ordinary KS.

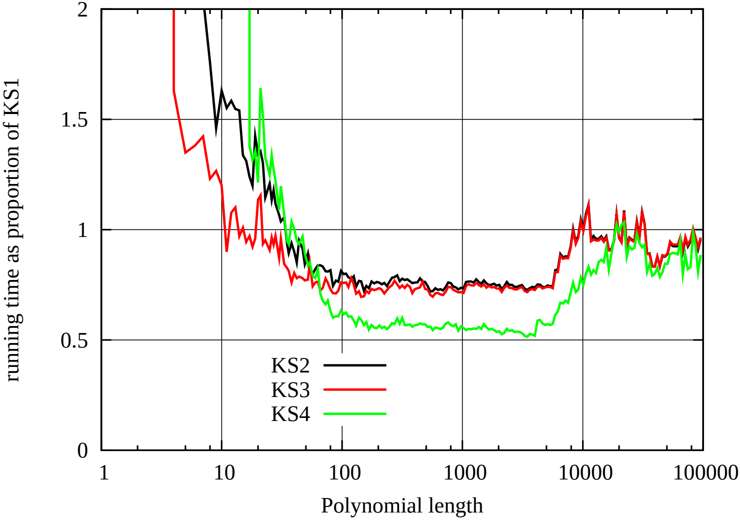
Karatsuba multiplication $\implies$ 2.25x speedup.

FFT multiplication $\implies$ no constant speedup.

KS4 vs KS1


running time of KS4 as proportion of KS1
2
1.5
1
0.5
0
1
10
100
1000
10000
100000
Polynomial length
5-bit coefficients
10-bit coefficients
20-bit coefficients

KS2, KS3, KS4 vs KS1 for 20-bit coefficients



KS $\geq$ 5?

Could we evaluate at other points?

One possibility:

$$f(10i) = -4871 + 40620i, \qquad g(10i) = -2279 - 18540i$$

$$h(10i) = f(10i)g(10i) = -741993791 + 182881320i.$$

This yields $h^0(-10^2)$, $h^1(-10^2)$.

Then use $h(\pm 10) = f(\pm 10)g(\pm 10)$ to recover $h^0(10^2)$, $h^1(10^2)$, and then $h(x)$.

Problem: complex multiplication requires *three* real multiplications.

So this strategy reduces to *five* multiplications of $1/4$ the size.

Other roots of unity lead to similar problems.

Applications

These algorithms are implemented in zn_poly, a new library for polynomial arithmetic in $(\mathbf{Z}/m\mathbf{Z})[x]$, where n fits into a machine word ('long' in C).

Currently zn_poly is pretty good at multiplication and middle product, not very good at division yet.

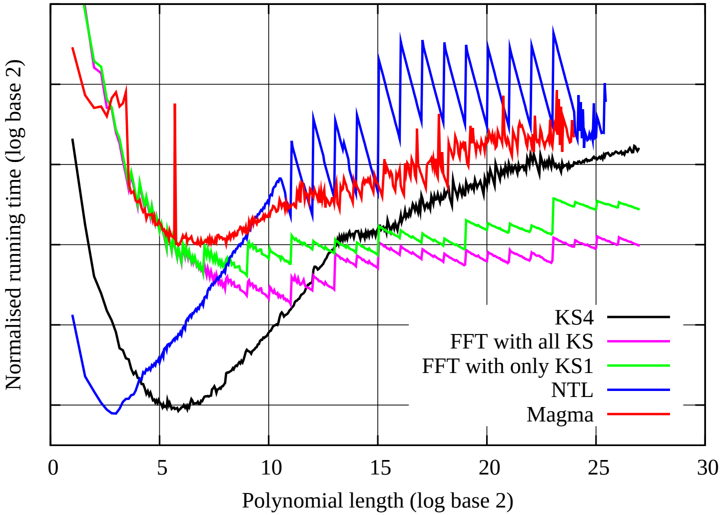
Applications

Algorithms for multiplication in zn_poly:

- Direct classical/Karatsuba for small degree
- KS1/KS2/KS3/KS4 for medium degree
- Schönhage–Nussbaumer FFT for large degree (odd modulus only)

Note that the FFT reduces a length-n multiplication to about $\sqrt{n}$ length-$\sqrt{n}$ multiplications, so the improved KS affects huge multiplications (even though it isn't used directly).

Comparison of packages (48-bit coeffs, 2.6GHz Opteron)



Applications

'Real-life' uses of zn_poly:

- My Ph.D. thesis: a new algorithm for computing zeta functions of hyperelliptic curves over finite fields (important problem in cryptography). Record example: genus 3 over $\mathbf{F}_p$ for $p = 2^{55} - 55$, Jacobian has $\approx 2^{165}$ points. Used 30 hours and 90 GB RAM on single Opteron core.
- Joint work with Joe Buhler: verification of Vandiver's conjecture and computation of cyclotomic invariants for $p < 163{,}000{,}000$. Used about 21 CPU years on a supercomputer at TACC (Texas Advanced Computing Center).

Applications

Other folks who have used zn_poly:

- *Computing L-series of hyperelliptic curves*, Andrew Sutherland and Kiran Kedlaya (MIT), ANTS-VIII, 2008.
- *Computing Hilbert class polynomials with the Chinese Remainder Theorem* (in preparation), Andrew Sutherland.
- The FLINT library ("Fast Library for Number Theory", William Hart, Warwick) uses zn_poly for arithmetic in $(\mathbf{Z}/m\mathbf{Z})[x]$.

Thank you!