Faster polynomial multiplication via multipoint Kronecker substitution

David Harvey, New York University

5th February 2009
Kronecker substitution

KS = an algorithm for multiplying polynomials in \( \mathbb{Z}[x] \).

Example:

\[ f = 41x^3 + 49x^2 + 38x + 29, \quad g = 19x^3 + 23x^2 + 46x + 21. \]

To find \( h = fg \), evaluate

\[ f(10^4) = 41004900380029, \quad g(10^4) = 19002300460021 \]

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 $\mathbb{Z}[x]$ 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 $\mathbb{Z}[x]$ and $(\mathbb{Z}/n\mathbb{Z})[x]$ when coefficients are small.
- NTL uses KS to reduce arithmetic in $\text{GF}(p^n)[x]$ to arithmetic in $\text{GF}(p)[x]$. 
Kronecker substitution

What is the running time?

Suppose $f, g$ have (non-negative) coefficients with $c$ bits.
Suppose $\text{len } f = \text{len } g = n$ (i.e. they have degree $n - 1$).

Coefficients of $h = fg$ are bounded by $2^{2^c} n$, 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):

\[ f = 41x^3 + 49x^2 + 38x + 29, \quad g = 19x^3 + 23x^2 + 46x + 21. \]

Then

\[ f(10^2) = 41493829, \quad g(10^2) = 19234621, \]
\[ f(-10^2) = -40513771, \quad g(-10^2) = -18774579. \]

Packed with alternating signs — still linear time.

Two half-sized integer multiplications:

\[ h(10^2) = f(10^2)g(10^2) = 798118074653809, \]
\[ h(-10^2) = f(-10^2)g(-10^2) = 760628994227409. \]
Problem: coefficients of $h$ overlap, in both $h(10^2)$ and $h(-10^2)$:

\[
\frac{779373534440609}{187445402132} + \frac{779373534440609}{187445402132} - \frac{779373534440609}{760628994227409} = h(10^2)
\]

\[
\frac{187445402132}{798118074653809} = h(-10^2)
\]

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

\[
h^0(10^4) = \frac{1}{2}(h(10^2) + h(-10^2)) = 779373534440609
\]

\[
10^2 h^1(10^4) = \frac{1}{2}(h(10^2) - h(-10^2)) = 18744540213200
\]

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)

David Harvey, New York University

Multipoint Kronecker substitution
KS3 algorithm

Another set of points to try...

\[ f = 41x^3 + 49x^2 + 38x + 29, \quad g = 19x^3 + 23x^2 + 46x + 21. \]

Then

\[ f(10^2) = 41493829, \quad g(10^2) = 19234621, \]
\[ 10^6 f(10^{-2}) = 29384941, \quad 10^6 g(10^{-2}) = 21462319. \]

Packed in \textit{reversed order} — still linear time.

Two half-sized integer multiplications:

\[ h(10^2) = f(10^2)g(10^2) = 798118074653809, \]
\[ 10^{12} h(10^{-2}) = 10^6 f(10^{-2})10^6 g(10^{-2}) = 630668977538179. \]
**KS3 algorithm**

<table>
<thead>
<tr>
<th>0779</th>
<th>0609</th>
</tr>
</thead>
<tbody>
<tr>
<td>1874</td>
<td>2132</td>
</tr>
<tr>
<td>3735</td>
<td>3444</td>
</tr>
<tr>
<td>3735</td>
<td>4540</td>
</tr>
<tr>
<td>4540</td>
<td>3735</td>
</tr>
<tr>
<td>3444</td>
<td>0609</td>
</tr>
<tr>
<td>0609</td>
<td>0779</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>798118074653809</th>
<th>630668977538179</th>
</tr>
</thead>
<tbody>
<tr>
<td>$= h(10^2)$</td>
<td>$= 10^{12}h(10^{-2})$</td>
</tr>
</tbody>
</table>

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

Let $h = h_6x^6 + h_5x^5 + h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$. 

David Harvey, New York University

Multipoint Kronecker substitution
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

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 $79 + 18$?
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.)
Therefore we completely recover $h_6 = 779$, and we can subtract it from the appropriate location in both sums.
KS3 algorithm

\[
\begin{align*}
0609 & \quad 2132 \\
3735 & \quad 3444 \\
4540 & \quad 4540 \\
3444 & \quad 3735 \\
2132 & \quad 1874 \\
0609 & + \\
1911 & + \\
1874074653809 & + \\
6306689775374 & + \\
\end{align*}
\]

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!
An example where the ‘bogus carry propagation’ problem occurs: both

\[ h(x) = 9901x^2 + 9901x \quad \text{and} \quad h(x) = 100x^3 + 100 \]

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.
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 vs Polynomial length for different polynomial lengths:
- 5-bit coefficients
- 10-bit coefficients
- 20-bit coefficients

David Harvey, New York University
Multipoint Kronecker substitution
KS4 algorithm

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

\[ f = 41x^3 + 49x^2 + 38x + 29, \quad g = 19x^3 + 23x^2 + 46x + 21. \]

\[
\begin{align*}
  f(10) &= 46309, \quad g(10) = 21781, \\
  f(-10) &= -36451, \quad g(-10) = -17139, \\
  10^3 f(10^{-1}) &= 33331, \quad 10^3 g(10^{-1}) = 25849, \\
  10^3 f(-10^{-1}) &= 25649, \quad 10^3 g(-10^{-1}) = 16611.
\end{align*}
\]

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

\[
\begin{array}{c}
  4929 \\
  4138 - \\
  \hline \\
  -36451
\end{array}
\]
KS4 algorithm

Leads to four multiplications of one fourth the size:

\[
\begin{align*}
    h(10) &= f(10)g(10) = 1008656329 \\
    h(-10) &= f(-10)g(-10) = 624733689 \\
    10^6 h(10^{-1}) &= 10^3 f(10^{-1})10^3 g(10^{-1}) = 861573019 \\
    10^6 h(-10^{-1}) &= 10^3 f(-10^{-1})10^3 g(-10^{-1}) = 426055539
\end{align*}
\]

Then:

\[
\begin{align*}
    h(10) \quad \text{KS2} \quad &\rightarrow \quad \begin{cases} 
    h^0(10^2) \\
    h^1(10^2) 
\end{cases} \\
    h(-10) \quad \text{KS2} \quad &\rightarrow \quad \begin{cases} 
    h^0(10^{-2}) \\
    h^1(10^{-2}) 
\end{cases} \\
    h(10^{-1}) \quad \text{KS2} \quad &\rightarrow \quad \begin{cases} 
    h^0(10^{-2}) \\
    h^1(10^{-2}) 
\end{cases} \\
    h(-10^{-1}) \quad \text{KS2} \quad &\rightarrow \quad \begin{cases} 
    h^0(10^{-2}) \\
    h^1(10^{-2}) 
\end{cases}
\end{align*}
\]

and

\[
\begin{align*}
    h^0(10^2) \quad \text{KS3} \quad &\rightarrow h^0(x) \\
    h^0(10^{-2}) \quad \text{KS3} \quad &\rightarrow h^1(x)
\end{align*}
\]
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

Polynomial length
- 5-bit coefficients
- 10-bit coefficients
- 20-bit coefficients

David Harvey, New York University

Multipoint Kronecker substitution
KS2, KS3, KS4 vs KS1 for 20-bit coefficients

Polynomial length

running time as proportion of KS1

KS2
KS3
KS4

David Harvey, New York University
Multipoint Kronecker substitution
Could we evaluate at other points?

One possibility:

\[ f(10i) = -4871 + 40620i, \quad g(10i) = -2279 - 18540i \]

\[ h(10i) = f(10i)g(10i) = -741993791 + 182881320i. \]

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.
These algorithms are implemented in zn\_poly, a new library for polynomial arithmetic in \((\mathbb{Z}/m\mathbb{Z})[x]\), 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.
Algorithms for multiplication in \texttt{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)

Multipoint Kronecker substitution
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 $\mathbb{F}_p$ 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).
Other folks who have used zn\_poly:


- *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 $(\mathbb{Z}/m\mathbb{Z})[x]$. 

English
Thank you!