Implementation of the DKSS Algorithm for Multiplication of Large Numbers

Christoph Lüders

Universität Bonn

The International Symposium on Symbolic and Algebraic Computation, July 6–9, 2015, Bath, United Kingdom
In 2008, De, Kurur, Saha & Saptharishi (DKSS) published a paper on how to multiply large numbers based on ideas of Fürer’s algorithm.

Their procedure was implemented and compared to Schönhage-Strassen multiplication to see how it performs in practice.

*But first, some context…*
On 64-bit machines a word can hold non-negative values $< W = 2^{64}$.

A large number $0 \leq a < W^n$ is represented as array of $n$ words: $(a_0, a_1, \ldots, a_{n-1})$.

Each word $a_i$ is a “digit” of $a$ in base $W$.

Ordinary (grade-school) multiplication of $a \cdot b$: multiply each $a_i$ with each $b_j$. Run-time is $O(n^2)$. Function name OMUL.

Can we do better?
**Multiplication: Karatsuba**

- (Karatsuba 1960): cut numbers $a$ and $b$ in half. With the help of some linear time operations, only 3 half-sized multiplications are needed:

  \[
  a = a_0 + a_1 W^n, \quad b = b_0 + b_1 W^n
  \]

  \[
  P_0 = a_0 b_0, \quad P_1 = (a_0 - a_1)(b_0 - b_1), \quad P_2 = a_1 b_1
  \]

  \[
  ab = P_0 (1 + W^n) - P_1 W^n + P_2 (W^n + W^{2n})
  \]

- When done recursively run-time is $O(n^{\log_2 3}) \approx O(n^{1.58})$. Function name **KMUL**.
Multiplication: Toom-Cook

- (Toom 1963, Cook 1966): cut numbers in \( k \geq 2 \) pieces and perform only \( 2k - 1 \) “small” multiplications plus some linear time operations.

- Run-time is \( O(n^{\log_k(2k-1)}) \). For \( k = 3, 4, 5 \) this is \( \approx O(n^{1.46}), O(n^{1.40}), O(n^{1.37}) \). Function name for \( k = 3 \) is \text{T3MUL}.  

- Problem: the number of linear time operations grows quickly with \( k \). 

(Strassen 1968): Cut numbers $a$ and $b$ in $n/2$ pieces each and interpret pieces as coefficients of polynomials over $R[x]$, $R$ ring.

Evaluate polynomials at $n$ points, multiply the sample values and interpolate to obtain product. Propagate carries.

If $\omega$ is primitive $n$-th root of unity in $R$, evaluation and interpolation can be done on $\omega^k$, $0 \leq k < n$. We can use the fast Fourier transform (FFT) with $O(n \cdot \log n)$ steps. Function name QMUL.

Problem: the larger $n$ becomes, the more precision is needed in coefficient ring $R$. This limits the length of input numbers.
Multiplication: Schönhage-Strassen

- (Schönhage & Strassen 1971): Use $R = \mathbb{Z}/(2^K + 1)\mathbb{Z}$ and $\omega = 2$ as primitive $2K$-th root of unity for the FFT.
- Multiplications by $\omega^k$ are just cyclic shifts, can be done in linear time.
- Run-time is $O(N \cdot \log N \cdot \log \log N)$, coefficient length is $O(\sqrt{N})$. Function name **SMUL**.

- Problem: the order of $\omega$ is not very high. Except for $\sqrt{2}$, there are generally no higher order roots of unity, thus FFT length is quite limited.
- Nevertheless, Schönhage-Strassen is the standard for multiplication of large numbers with over $\approx 150,000$ bits.
Crossover Points Between Algorithms

\[
\begin{align*}
\text{OMUL} & \geq 28 \\
\text{KMUL} & \geq 28 \\
\text{T3MUL} & \geq 152 \\
\text{SMUL} & > 2464
\end{align*}
\]

Input in 64-bit words
(De, Kurur, Saha & Saptharishi 2008): Use polynomial quotient ring
\[ R = \mathcal{P}[\alpha]/(\alpha^m + 1) \] with \( \mathcal{P} = \mathbb{Z}/p^c\mathbb{Z} \), \( p = h \cdot 2^M + 1 \) prime.

Select \( M = N/\log^2 N \) and \( m = \log N \) as powers of 2, \( M > m \).
Let \( \mu = M/m \).

From a generator of \( \mathbb{F}_p^* \) calculate a primitive \( 2^M \)-th root of unity \( \rho \in \mathcal{P}[\alpha] \) with \( \rho^\mu = \alpha \).

With \( \alpha \) as primitive \( 2^m \)-th root of unity and modulus \( (\alpha^m + 1) \) multiplications by \( \alpha^k \) are cyclic shifts: fast!

\( \rho \) is high order root of unity: large FFT length.
A length-$2M$ FFT can be calculated like this:

- $2M = \mu \cdot 2m$.
- Interpret the coefficients as a matrix with $2m$ rows and $\mu$ columns.
- Do $\mu$ many length-$2m$ FFTs (on the columns) with $\alpha$ as root of unity.
- Perform *bad multiplications* on the coefficients, i.e. multiply them by some $\rho^k$.
- Do $2m$ many length-$\mu$ FFTs (on the rows) by calling the FFT routine recursively.

Multiplication in $R$ is reduced to integer multiplication by use of Kronecker-Schönhage substitution.

Run-time is $O(N \cdot \log N \cdot K^{\text{log}^* N})$ with $K = 16$, coefficient length is $O(\log^2 N)$. Function name **DKSS_MUL**.
In genuine DKSS, prime $p$ is searched at run-time. To keep that time low, $p$ must be kept small. So, input numbers are encoded as $k$-variate polynomials, $k$ constant.

Since input length is limited by available memory, we can precompute all of the required primes $p$ and generators of $\mathbb{F}_p^*$. This allows to use univariate polynomials and simplifies calculation of the root of unity $\rho$. We can use $c = 1$ and hence $\mathcal{P} = \mathbb{Z}/p\mathbb{Z}$.

For 64-bit architecture, only 6 primes need to be precomputed.
Comparison of Execution Time

![Comparison of Execution Time Graph](image)

- SMUL
- DKSS_MUL

Input in 64-bit words

Execution cycles
Quotient of Run-times

\[ \frac{T_{DKSS\_MUL}}{T_{SMUL}} \]

- Input in 64-bit words

Christoph Lüders (Universität Bonn)
Implementation of DKSS Multiplication
ISSAC 2015, Bath, U.K.
Results

For the numbers tested (up to 1.27 GB input size, total temporary memory required 26 GB):

- **DKSS_MUL** is between 27 and 36 times slower than **SMUL**.
- **DKSS_MUL** requires $\approx 2.3$ times the temporary memory than **SMUL**.
- About 80 % of run-time is spent with *bad multiplications*, i.e. multiplications by $\rho^k$ that are not powers of $\alpha$.
- Another 9 % are spent for pointwise products.
- Recursion did not take place. Even with the largest inputs, inner multiplications were just 195 words long.
- Cache effects did not slow it down, either.
When Will DKSS Beat Schönhage-Strassen?

- Model **SMUL** run-time:
  \[ T_\sigma \leq \sigma \cdot N \cdot \log N \cdot \log \log N. \]

- Model **DKSS_MUL** run-time:
  \[ T_\eta \leq \eta \cdot N \cdot \log N \cdot K^{\log^* N}, \quad K = 16. \]

- Find fitting constants \( \sigma \) and \( \eta \) from measured run-times.
- Solve \( T_\sigma \geq T_\eta \) numerically:
  \[ N \geq 10^{10^{4796}} \]

Christoph Lüders (Universität Bonn)
Implementation of DKSS Multiplication
ISSAC 2015, Bath, U.K.
Future work

Some ideas:

- Exploit the sparseness of the factors in the underlying multiplication. Estimated speed-up: factor 2.
- Use variant of Kronecker-Schönhage substitution (Harvey).
- Parameters $p$, $M$ and $m$ should be selected with more care. Estimated speed-up: maybe 30%.
- Modular reduction should be sped up (Montgomery’s trick or other). Estimated speed-up: about 22%.
- Total estimated possible speed-up: factor 3.2, but even then $\text{DKSS\_MUL}$ is at best 8.5 times slower than $\text{SMUL}$. 
Implementation was done in C++ and assembly language under Windows as part of BIGNUM, my large integer library.

Multiplication compares favorably with MPIR (GMP for Windows) and is only 1.3 times slower on average.

Source code is available from http://www.wrogn.com/bignum and licensed under LGPL.

Many thanks to Andreas Weber and Michael Clausen.