Karatsuba multiplication with temporary space of size $\leq$ n

Emmanuel Thomé

Karatsuba multiplication with temporary space of size $\leq n$

Emmanuel Thomé

September 2002

Abstract

We describe in this short note how it is possible to perform a Karatsuba multiplication of two polynomials of degree $n-1$ using a buffer holding at most $n-1$ coefficients if n is even, and n coefficients if n is odd. This method is valid for any n (in other words, when $n \geq 2$). Similar results can be obtained for the Karatsuba squaring.

1 Introduction and terminology

The Karatsuba multiplication method (see for instance [**?**]) is the second best-known multiplication algorithm after "schoolbook" multiplication. To multiply two polynomials of degree $n-1$ (having therefore n coefficients), a complexity of $n^{\log_2 3} = n^{1.58}$ is achieved using the following recursive procedure:

Algorithm $\textsc{Mul}(a, b, n)$

Let $p = \lfloor \frac{n}{2} \rfloor$ and $q = \lceil \frac{n}{2} \rceil$ (such that $p + q = n$).
Write $a = a_0 + X^p a_1$ and $b = b_0 + X^p b_1$.
Compute $\alpha = a_0 - a_1$, $\beta = b_1 - b_0$. (see erratum)
return $\textsc{Mul}(a_0, b_0, p) \times (1 + X^p) +$
$\textsc{Mul}(\alpha, \beta, q) \times X^p +$
$\textsc{Mul}(a_1, b_1, q) \times (X^p + X^{2p})$

It is very easy to see that this procedure achieves the announced asymptotic complexity, since a multiplication of size n is performed using three multiplications of size $\frac{n}{2}$.

When concerns lean towards efficiency, the first thing to do is to evaluate the *threshold point* above which this procedure is worthwhile. Below this threshold, the added linear complexity of the Karatsuba algorithm is cumbersome, and the classical "schoolbook" multiplication performs better. The actual threshold value depends of course highly on the ratio between the respective throughputs of addition and multiplication (and therefore depends on the machine and on the nature of the coefficients). We refer to the multiplication below the threshold as the "basecase" multiplication.

The second concern for implementations is the management of the temporary space. How much temporary space is needed for the storage of the quantities α, β, and the three intermediate products constructed (temporary space must also be provided for the recursive steps) ? We assume that the multiplication procedure is not allowed to destroy its input operands, but that any temporary data is allowed to be stored in the destination operand. Requiring $4n$ coefficients of temporary space in addition to this makes the implementation fairly trivial. With some additional care, $2n$ coefficients might be enough. Eventually, we show that n coefficients of temporary space are sufficient, at the expense of some shuffling of the data (more precisely, $n + (n \mod 2) - 1$ coefficients of temporary space are used). Our construction copes even with the limit case where basecase multiplication is only used for $n = 1$.

2 Saving temporary space

We prove the following assertion.

Proposition 2.1 MUL(a, b, n) *can be implemented to use* $n + (n \mod 2) - 1$ *coefficients of temporary space.*

This figure does deliberately not take into account the storage required on the stack for the recursive implementation, which is of the order of $\log_2 n$.

PROOF :

We prove the proposition by induction on n. Since the basecase multiplication uses no temporary space, the result is trivially true for any n below the threshold. The basecase multiplication being used at the very least for $n = 1$, let us assume that $n \geq 2$, and that the assertion is true for any strictly smaller value of n.

Assume that a, b are arrays of coefficients. We denote by $a[\![x \dots y[\![$ the range of $y - x$ coefficients stored in a, starting from coefficient indexed x. The first coefficient stored in a is, by convention, indexed 0.

We choose $p = \lfloor \frac{n}{2} \rfloor$ and $q = \lceil \frac{n}{2} \rceil$, such that $p + q = n$, and $p \leq q \leq p + 1$. Since $n \geq 2$, we have $p \geq 1$ and $q \geq 1$. One can also immediately notice that $2p \geq q$.

We assume that we have at our disposal the result area, denoted $r[\![0 \dots 2n - 1[\![$, and some temporary space, denoted $t[\![0 \dots 2q - 1[\![$. Since $2q = n + (n \mod 2)$, this amount of data corresponds with the announced amount of temporary space.

Our goal is of course to have $a \times b$ stored into $r[\![0 \dots 2n - 1[\![$. More precisely, the product $\alpha \times \beta$ will have to be stored in $r[\![p \dots p + 2q - 1[\![$ (since α and β both have q coefficients, their product has $2q - 1$ coefficients), with the added contribution of $a_0 \times b_0$ to $r[\![0 \dots 2p - 1[\![$ and $r[\![p \dots 3p - 1[\![$, and of $a_1 \times b_1$ to $r[\![p \dots p + 2q - 1[\![$ and $r[\![2p \dots 2n - 1[\![$. We proceed through the following steps.

1. $r[\![0 \dots q[\![\leftarrow a[\![0 \dots p[\![- a[\![p \dots n[\![$ (this is α).
2. $r[\![q \dots 2q[\![\leftarrow b[\![p \dots n[\![- b[\![0 \dots p[\![$ (this is β ; see erratum).

3. $t[\![0 \dots 2q-1[\![\leftarrow r[\![0 \dots q[\![\times r[\![q \dots 2q[\![$, using $r[\![2q \dots 2n-1[\![$ for the temporary space.
4. $r[\![2p \dots 2n-1[\![\leftarrow a[\![p \dots n[\![\times b[\![p \dots n[\![$, using $r[\![0 \dots q[\![$ for the temporary space.
5. $t[\![0 \dots 2q-1[\![\leftarrow t[\![0 \dots 2q-1[\![+ r[\![2p \dots 2n-1[\![$.
6. $r[\![p \dots 2p[\![\leftarrow t[\![0 \dots p[\![$.
7. $r[\![2p \dots p+2q-1[\![\leftarrow r[\![2p \dots p+2q-1[\![+ t[\![p \dots 2q-1[\![$.
8. $t[\![0 \dots 2p-1[\![\leftarrow a[\![0 \dots p[\![\times b[\![0 \dots p[\![$, using $r[\![0 \dots p[\![$ for the temporary space.
9. $r[\![0 \dots p[\![\leftarrow t[\![0 \dots p[\![$.
10. $r[\![p \dots 2p-1[\![\leftarrow r[\![p \dots 2p-1[\![+ t[\![p \dots 2p-1[\![$.
11. $r[\![p \dots 3p-1[\![\leftarrow r[\![p \dots 3p-1[\![+ t[\![0 \dots 2p-1[\![$.

We need to explain why data never overlaps within this procedure. Step 3 is the only delicate step. The amount of temporary step provided is $2n-1-2q$ coefficients, that is, $2p-1$ coefficients. Since step 3 performs a multiplication of size q, we must make sure that $2p-1$ is always greater than or equal to $q+(q \mod 2)-1$. Let us first investigate the cases where $2p-1<q$. This inequality implies $2p \leq q$, hence $2p \leq p+1$, hence $p \leq 1$, which yields immediately $p=1, q=2, n=3$. So $n=3$ is the only possibly embarrassing case. But then, q is an even number strictly smaller than n, which means that $q+(q \mod 2)-1 = q-1 = 1$ coefficient of temporary space is used. Since $2p-1=1$, this fits.

The absence of overlap in the other steps of the procedure is just the matter of a trivial check. ■

3 Code

This piece of code illustrates the proposition above. The objects considered are polynomials over the 2-adic integers with precision equal to the wordsize. We use the syntax of the software multiprecision library `GMP` [**?**]. We assume that the procedure `mul(r,a,b,t,n)` compares `n` with the threshold, and then performs either `kara_mul(r,a,b,t,n)` or otherwise `basecase_mul(r,a,b,t,n)`. Here is the implementation of `kara_mul(r,a,b,t,n)`:

```
void kara_mul(mp_limb_t * r, mp_limb_t * a, mp_limb_t * b,
              mp_limb_t * t, mp_size_t n)
{
        mp_size_t p, q;

        p = n>>1;
        q = n - p;
```

```
        /* step 1 */
        mpn_sub_n(r,a,a+p,p);
        if (q!=p) r[p]=-a[n-1];

        /* step 2 */
        mpn_sub_n(r+q,b+p,b,p);
        if (q!=p) r[n]=b[n-1];

        /* step 3 */
        mul(t,r,r+q,r+2*q,q);

        /* step 4 */
        mul(r+2*p,a+p,b+p,r,q);

        /* step 5 */
        mpn_add_n(t,t,r+2*p,2*q-1);

        /* step 6 */
        memcpy(r+p,t,p*sizeof(mp_limb_t));

        /* step 7 */
        mpn_add_n(r+2*p,r+2*p,t+p,t,2*q-1-p);

        /* step 8 */
        mul(t,a,b,r,p);

        /* step 9 */
        memcpy(r,t,p*sizeof(mp_limb_t));

        /* step 10 */
        mpn_add_n(r+p,r+p,t+p,p-1);

        /* step 11 */
        mpn_add_n(r+p,r+p,t,2*p-1);
}
```

4 Related problems

One can obviously obtain the same result for the Karatsuba squaring. However, be it either for the multiplication or the squaring, it does not seem likely that anything better than this result can be obtained. Step 8 of the algorithm is really tight, and uses all the

storage space available. This proves that less than $n - O(1)$ of temporary storage cannot be sufficient.

If one relaxes the requirement that the input operands be left untouched, then one can do somewhat better. If one operand can be clobbered, then $\lceil \frac{n}{2} \rceil$ is probably not hard to achieve.

The same kind of algorithm should work over the integers too, but the results might not be as good (the fact that two polynomials with n coefficients multiply to form a product with $2n - 1$ coefficients is heavily used here – The size of the product is $2n$ with integers instead of polynomials).

The brave reader might try to figure out how to achieve a similar temporary buffer size for the Toom 3-way multiplication algorithm.

Erratum

20110215: F. Morain pointed out that the pseudo-code on page 1 and on the bottom of page 2 had the sign of β wrong. This has been fixed.