Computing Clipped Products

Arthur C. Norman[1] and Stephen M. Watt[2]

[1] Trinity College, Cambridge CB2 1TQ, UK
acn1@cam.ed.uk

[2] Cheriton School of Computer Science, University of Waterloo, N2L 3G1 Canada
https://cs.uwaterloo.ca/~smwatt
smwatt@uwaterloo.ca

Abstract. Sometimes only some digits of a numerical product or some terms of a polynomial or series product are required. Frequently these constitute the most significant or least significant part of the value, for example when computing initial values or refinement steps in iterative approximation schemes. Other situations require the middle portion. In this paper we provide algorithms for the general problem of computing a given span of coefficients within a product, that is the terms within a range of degrees for univariate polynomials or range digits of an integer. This generalizes the "middle product" concept of Hanrot, Quercia and Zimmerman. We are primarily interested in problems of modest size where constant speed up factors can improve overall system performance, and therefore focus the discussion on classical and Karatsuba multiplication and how methods may be combined.

Keywords: integer product, polynomial product, convolution subrange, product approximation

1 Introduction

The classical discussion of integer multiplication concentrates on forming the product of a pair of N-digit numbers to form a $2N$ digit result. The techniques used apply also to dense univariate polynomials or to truncated power series. We start by pointing out what a limited viewpoint this represents.

There are three use-cases where at first it seems that balanced multiplication (*i.e.* ones where the two inputs are the same size) will be central in large computations. The first is in fixed but extended precision floating point arithmetic where N-digit floating values are handled as N-digit integers alongside an exponent value. But here it is not really appropriate to use full general multiplication because only the top N of the product digits are retained, so computing the low N digits will represent a waste. The same issue arises (but minus the complication of carries) in work with truncated power series where it is the low half of the full polynomial product that has to be kept. The second is in cryptography where there is some large modulus p and all values worked with are in the range

0 to $p-1$. But hereafter any multiplication that generates a double-length product there needs to be a reduction mod p. This has the feel of (even if it is not exactly the same as) just wanting the low N digits of the full $2N$ digit product. Often the cost of the remaindering operation will be at least as important as the multiplication, and we will refer later to our thoughts on that. The third may be in computation of elementary constants such as e and π to extreme precision where the precision of all the computational steps is very rigidly choreographed, but again at least at the end it is liable to have much in common with the extended precision floating point case.

Other real-world cases are more liable to be forming $M \times N$ products where M and N may happen to be close in magnitude but can also be wildly different, and where the values of M and N are liable to differ for each multiplication encountered. Such a situation arises in almost any computation involving polynomials with exact integer coefficients — GCD calculations, Gröbner bases, power series with rational coefficients, quantifier elimination. Many of these tasks have both broad practical applicability and can be seriously computationally demanding so that absolute performance matters. We expand on these arguments in [10].

Part of our motivation for this work was that, when considering division (and hence remainder) using a Newton-style iteration to compute a scaled inverse of the divisor, we found that in a fairly natural way we were wanting to multiply an N digit value by one of length $2N$ but then only use the middle N digits of the full $3N$-length product. Failing to exploit both the imbalance and the clipping there would hurt the overall performance of division.

This specific problem of computing the middle third of a $2N \times N$ product has been considered previously [3], where it has been used to speed up the division and square root of power series. In this situation, the middle third is exactly the part of the product where the convolution for each term uses all terms of the shorter factor. This allows a Karatsuba-like recursive scheme to compute the middle third of the product, or to use FFTs of size $2N$ instead of $4N$, saving a factor of 2 multiplying large values. This reduces the Newton iteration time for division from $3M(N) + O(N)$ to $2M(N) + O(N)$, where $M(N)$ is the time for multiplication of size N.

Another important case is that of computing the initial terms of a product. Mulders' algorithm [9,4] produces the first N terms of a power series product, called a "short product". This is done by selecting a cut off point k and computing one $k \times k$ product and two $(N-k) \times (N-k)$ short products recursively.

These "middle third of a $2N \times N$ product" and "prefix" situations are important cases but proper code needs to allow for additional generality. So we view the proper general problem to address is the formation of an $M \times N$ product where only digits (or terms) from positions a to b are required. We call this a *clipped product*.

A clipped product can obviously be implemented by forming the full $M \times N$ product, or indeed by padding the smaller of the two inputs to get the $N \times N$ case (and sometimes going further and padding the size there to be a power of 2!) and then at the end just ignoring unwanted parts of the result. From an asymptotic big-O perspective that may suffice, but we are interested in implementations where even modest improvements in absolute performance on "medium sized" problems matters. So can we do better? Achieving certified lower bounds on cost becomes infeasible both with the large number of input size and size-related parameters and with the fact that for medium sized problems overheads matter, and they can be platform- and implementation-sensitive. But despite that we will show we can suggest better approaches than the naïve one.

The main results of this paper are algorithms for clipped multiplication of integers and polynomials over a general ring, adapting both the classical $O(N^2)$ and Karatsuba $O(N^{log_2 3})$ methods. An analysis is given to show how much look-back is required in order to have the correct carry in for the lowest digit in the classical integer case.

The remainder of the article is organized as follows: Section 2 provides some notation and gives a definition of clipping. Section 3 shows some straightforward methods to compute clipped products. The main idea is that one can compute a lower part by multiplying only the lower terms and clipping afterwards. For polynomials, one can use reversed polynomials to get a higher part. Section 4 shows how to adapt polynomial multiplication algorithms so that only the required part is calculated. Section 5 shows how to adapt integer multiplication so that only the required part is calculated. The difference from polynomial multiplication is in dealing with carries. Section 6 discusses some issues that arise in combining sub-multiplications. Finally, Section 7 presents some further thoughts and conclusions. The algorithms are presented in high-level Maple code, though in practice a lower-level language like C or Rust may be used.

2 Preliminaries

For many algorithms, integers and univariate polynomials behave similarly, however especially when losing low order digits is called for integer calculations are complicated by the need to allow for carries up from those discardable parts of the results. We use the notation of [12,13] to allow generic discussion of concepts relating to both:

$[a..b], [a..b)$, *etc*	integer intervals, *i.e.* real intervals intersected with $\mathbb{Z}$
$\mathrm{prec}_B\, n$	number of base-B digits of an integer n, $\lfloor \log_B \lvert n \rvert \rfloor + 1$
$\mathrm{prec}_x\, p$	number of coefficients of a polynomial p, $\deg_x p + 1$
$M(n, m)$	the time to multiply two values with prec n and m.

The integer interval notation, "$[a..b]$" *etc*, is used by Knuth, *e.g.* [8]. In discussions that apply to both integers and polynomials we may use t as a generic base, which may stand for a integer radix or a polynomial variable, in which case we may write, *e.g.*, $\mathrm{prec}_t\, u$. We write the coefficients of u as $u_{t,i}$, where

$$u = \sum_{i \in [0..\mathrm{prec}_t\, u)} u_{t,i} t^i.$$

In the integer case, it is required to have $0 \le u_{B,i} < B$ for uniqueness. If $i < 0$ or $i \ge \mathrm{prec}_t\, u$, then $u_{t,i} = 0$. When the base for integers or variable for polynomials is understood, then we may simply write $\operatorname{prec} u$ and u_i. To refer to part of an integer or polynomial, we use the notation below and call this the *clipped* value.

$\operatorname{clip}_{t,I} u$	$\sum_{i\in I} u_{t,i} t^i$, I an integer interval,

As before, when the base for integers or the variable for polynomials is understood, we may simply write $\operatorname{clip}_I u$.

Examples
Letting $p = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$,

$$\operatorname{prec}_x p = 6$$
$$\operatorname{clip}_{x,[2..4)} p = a_3x^3 + a_2x^2$$

Letting $n = 504132231405$,

$$\operatorname{prec}_{100} n = 6$$
$$\operatorname{clip}_{100,[2..4]} n = 4132230000.$$

3 Straightforward Methods

We begin with the obvious methods to compute the clipped value of a product.

Direct Clipped Product

The simplest approach is simply to compute the whole product and extract the desired part. The clipped product $\operatorname{clip}_{t,[a..b]} (f \times g)$ may be computed as shown in Figure 1. Here, `clip(p, t, a, b)` computes $\operatorname{clip}_{t,[a..b]} p$. The computation time is $M(\operatorname{prec} f, \operatorname{prec} g) + O(|r|)$.

Example With

$$f = 4x^3 + 83x^2 + 10x - 62, \quad g = 82x^5 - 80x^4 + 44x^3 - 71x^2 + 17x + 75$$

we have

$$f \times g = 328x^8 + 6486x^7 - 5644x^6 - 2516x^5 - 425x^4 - 1727x^3 + 10797x^2 - 304x - 4650$$

so

$$\operatorname{clip}_{x,[2..3]} (f \times g) = -1727x^3 + 10797x^2.$$

Bottom Clipped Product

It is possible to improve on the direct clipped product when only the lower index coefficients are required. In this case integers and polynomials can be treated identically since carries out from the calculated range have no impact on the final result. If the clipped product range is $[0..b]$ then no coefficient of index greater than b of the operands can affect the clipped product. Therefore those coefficients can be discarded prior to computing the product. Then, as shown in Figure 1, one proceeds as before. The computation time is

$$M_{\text{bot}}(\text{prec}\, f, \text{prec}\, g, b) = M(\min(\text{prec}\, f, b), \min(\text{prec}\, g, b)) + O(b).$$

Example With f and g as before, we may compute $\text{clip}_{x,[0..3]}\,(f \times g)$ by multiplying $\text{clip}_{x,[0..3]}\, f = 4x^3+83x^2+10x-62$ and $\text{clip}_{x,[0..3]}\, g = 44x^3-71x^2+17x+75$ to get $176x^6+3368x^5-5385x^4-1727x^3+10797x^2-304x-4650$ which is then clipped to retain the terms of degrees in $[0..3]$, namely $-1727x^3+10797x^2-304x-4650$.

General Clipped Product from Bottom

If it is desired to compute a clipped product with a range with lower bound other than zero, it is possible to do so with one call to `BottomClippedProduct`, as shown in Figure 1. The computation time is

$$M_{\text{gbot}}(\text{prec}\, f, \text{prec}\, g, a, b) = M(\min(\text{prec}\, f, b), \min(\text{prec}\, g, b)) + O(b - a).$$

Example To compute $\text{clip}_{x,[2..3]}\,(f \times g)$ we obtain $\text{clip}_{x,[0..3]}\,(f \times g)$ using the bottom clipped product, and clip it as $\text{clip}_{x,[2..3]}\,\cdot$

Top Clipped Product From Reverse

As the upper limit of the clipping range approaches the precision of the multiplicands, `BottomClippedProduct` loses its computational advantage. In particular, if the top part of the product is desired, then there is no advantage at all. While the previous methods apply equally to integers and polynomials, here carries complicate the integer case. It is possible to compute polynomial clipped products with higher ranges only using polynomial reversal. Let

$$\text{rev}_x\, p = x^{\deg_x p} p(1/x).$$

Then the product clipped to $[a..\deg f + \deg g]$, *i.e.* the top, may be computed as shown in Figure 2. The time is determined by the `BottomClippedProduct` computation,

$$M_{\text{bot}}(\text{prec}\, f, \text{prec}\, g, \text{prec}\, f + \text{prec}\, g + a - 2).$$

```
DirectClippedProduct := proc(f, g, t, a, b)
    local p, r;
    p := f * g;
    r := clip(p, t, a, b);
    return r
end;

BottomClippedProduct := proc(f, g, t, b)
    local clipf, clipg, p, r;
    clipf := clip(f, t, 0, min(b, prec(t,f)-1));
    clipg := clip(g, t, 0, min(b, prec(t,g)-1));
    p := clipf * clipg;
    r := clip(p, t, 0, b);
    return r
end;

ClippedProductFromBottom := proc(f, g, t, a, b)
    local rb, r;
    rb := BottomClippedProduct(f, g, t, b);
    r  := clip(rb,  t, a, b);
    return r
end;
```

Fig. 1. Straightforward clipped products.

```
TopClippedPolynomialProduct := proc(f, g, x, a)
    local degf, degg, revf, revg, p, r;
    degf := degree(f, x); degg := degree(g, x);
    revf := rev(f, x);    revg := rev(g, x);
    p := BottomClippedProduct(revf, revg, x, degf+degg-a);
    r := x^a * rev(p, x);
    return r
end
```

Fig. 2. Top clipped product using polynomial reversal

Example To compute $\text{clip}_{x,[6..8]}(f \times g)$, we first compute

$$\text{rev}_x\, f = -62x^3 + 10x^2 + 83x + 4, \quad \text{rev}_x\, g = 75x^5 + 17x^4 - 71x^3 + 44x^2 - 80x + 82.$$

Then $\deg_x f + \deg_x g - a = 3 + 5 - 6 = 2$. So we compute

$$p = \text{clip}_{x,[0..2]}(\text{rev}_x\, f \times \text{rev}_x\, g) = -5644 * x^2 + 6486 * x + 328$$

and the result is

$$x^6 \text{rev}_x\, p = 328x^8 + 6486x^7 - 5644x^6.$$

Disadvantage of Straightforward Methods

These straightforward methods provide clipped products that generally cost less than clipping a full product, but they still perform significant extra work. Forming the $[a..b]$ clipped product via `BottomClippedProduct` on $0..b$ computes $2b+1$ coefficients where only $b-a+1$ are required. This is a significant difference for $O(N^p)$ multiplication methods such as the classical or Karatsuba algorithms. We therefore consider how to improve on this.

```
ClippedClassicalPolynomialProduct := proc(f0, g0, x, a, b)
    local f, g, df, dg, s, k, t, i, i0;

    # Ensure dg <= df
    f := f0; df := degree(f, x);
    g := g0; dg := degree(g, x);
    if dg > df then f,g := g,f;  df,dg := dg,df fi;

    # Form column sums.
    s := 0;                      # Note A
    for k from a to b do
        if   k <= dg then i0 := 0
        elif k <= df then i0 := k-dg
        else              i0 := k-df
        fi;
        t := 0;
        for i from i0 to k do t := t + coeff(f,x,i) * coeff(g,x,k-i) od;
        s := setcoeff(s,x,k,t) # Note A. s + t*x^k
    od;
    return s
end
```

Fig. 3. Clipped classical polynomial multiplication

Mulders [9] provides a more sophisticated method to compute products clipped to $[0..b]$ and hence the others. Mulders' method is discussed further in Section 6. If we do not require the entire prefix, that is for general $[a..b]$, then we can compute only what is needed, as shown in the next sections.

4 Clipped Polynomial Products

For the moment we concentrate on polynomial products, since this avoids the technicalities of dealing with carries. In practice, different multiplication algorithms give best performance for ranges of problem size. We therefore consider modified classical, Karatsuba and FFT multiplication.

Clipped Classical Polynomial Multiplication

Classical multiplication of univariate polynomials f and g requires $\operatorname{prec} f \times \operatorname{prec} g$ coefficient multiplications and a similar number of coefficient additions. It is easy to compute only the desired coefficients, as shown in Figure 3. This is Maple code, so `coeff(f,x,i)` computes f_i, the coefficient of x^i. The statement `s := setcoeff(s,x,k,t)` uses an auxilliary function to set the coefficient of x^k in s to be t. Depending on the implementation language, the lines annotated `Note A` would typically be setting values in a coefficient array indexed from zero. A suitable representation would be as a pair of the integer value `a` and a coefficient array `coeffs`, where `coeffs[i]` was the coefficient of x^{a+i}.

The greatest number of coefficient operations occurs when a and b both are in $[\deg g..\deg f]$, in which case $(b-a+1)*(\deg g+1)$ multiplications and $(b-a+1)*(\deg g)$ additions are required. We therefore have the bound

$$M_{\text{cclip}}(\operatorname{prec} f, \operatorname{prec} g, a, b) \leq (b-a+1) \times (\min(\deg f, \deg g)+1).$$

Example Let $\deg f = 7$, $\deg g = 4$, $a = 5$ and $b = 7$. Then the multiplication table looks like the following with the bold face entries calculated:

				$\mathbf{f_7g_0}$	$\mathbf{f_6g_0}$	$\mathbf{f_5g_0}$	f_4g_0	f_3g_0	f_2g_0	f_1g_0	f_0g_0
			f_7g_1	$\mathbf{f_6g_1}$	$\mathbf{f_5g_1}$	$\mathbf{f_4g_1}$	f_3g_1	f_2g_1	f_1g_1	f_0g_1	
		f_7g_2	f_6g_2	$\mathbf{f_5g_2}$	$\mathbf{f_4g_2}$	$\mathbf{f_3g_2}$	f_2g_2	f_1g_2	f_0g_2		
	f_7g_3	f_6g_3	f_5g_3	$\mathbf{f_4g_3}$	$\mathbf{f_3g_3}$	$\mathbf{f_2g_3}$	f_1g_3	f_0g_3			
f_7g_4	f_6g_4	f_5g_4	f_4g_4	$\mathbf{f_3g_4}$	$\mathbf{f_2g_4}$	$\mathbf{f_1g_4}$	f_0g_4				

So here 15 multiplications and 12 additions are required rather than the 40 and 28 that would be required by `ClippedProductFromBottom`.

Clipped Karatsuba Polynomial Multiplication

The Karatsuba multiplication scheme [7] splits the multiplicands in two and forms the required product using three recursive multiplications and four additions. For simplicity, assume $\operatorname{prec} f = \operatorname{prec} g = p = 2^n$. Let $f = f_h x^{p/2} + f_l$, $g = g_h x^{p/2} + g_l$, $f_m = f_h + f_l$ and $g_m = g_h + g_l$. For Karatsuba multiplication the three products $f_h \cdot g_h$, $f_m \cdot g_m$ and $f_l \cdot g_l$ are required. Then the product $f \cdot g = z_h x^p + z_m x^{p/2} + z_l$ where

$$z_h = f_h g_h, \qquad z_m = f_m g_m - f_h g_h - f_l g_l, \qquad z_l = f_l g_l.$$

We show that using clipping while computing a product by Karatsuba multiplication can significantly reduce the cost compared to clipping after the product is computed. Let $K(p)$ denote the number of required coefficient multiplications for Karatsuba multiplication of polynomials of prec $= p$. We have $K(p) = 3K(p/2)$ for $p > 1$ so $K(p) = p^{\log_2 3}$. If only a top (or bottom) part of the product is needed, then less work is required. Suppose only a top portion of the coefficients are required. Instead of computing all the required smaller products in full, some may be ignored and some may require only their upper parts. This reasoning leads to the following result.

Theorem 1. *When multiplying polynomials of degree $p-1$ by Karatsuba's method, if only the top (or bottom) $1/2^\ell$ fraction of the coefficients are required, with $2^\ell < p$, then the number of required coefficient multiplications is at most $K(p)/3^{\ell-1}$.*

Proof. Let f and g be the two precision p polynomials to be multiplied and let $T(p, \ell)$ denote the number of coefficient multiplications required for the top $1/2^\ell$ fraction of the product. If $\ell = 1$, then the full product z_h is needed, as is the top half of z_m. This requires all of $f_h g_h$ and the top half of $f_m g_m$ and $f_l g_l$, so

$$\begin{aligned} T(p, 1) &= K(p/2) + 2T(p/2, 1) \\ &\leq K(p). \end{aligned}$$

If $\ell > 1$, then only the top $1/2^{\ell-1}$ fraction of the coefficients of z_h are required and neither z_m nor z_l are needed. So

$$\begin{aligned} T(p, \ell) &= T(p/2, \ell - 1) \\ &\leq K(p/2^{\ell-1}) = K(p)/3^{\ell-1}. \end{aligned}$$

```
ClippedKaratsubaPolynomialProduct := proc(f, g, x, a, b, gmul)
    local df, dg, p, fh,gh, fl,gl,fm, gm, restrict,
          zha, zhb, zma, zmb, zla, zlb, zh, zm, zl;

    df := gdegree(f, x);  dg := gdegree(g, x);                   # Note A

    if a > df+dg then return 0 fi;
    if a > b     then return 0 fi;
    if b = 0     then return gmul(coeff(f,x,0),coeff(g,x,0)) fi; # Note A

    # Size of z parts.
    p := max(df, dg)+1;
    p := p + irem(p, 2);     # Make even

    # Product factors.
    fh := clip(f,x,p/2,p-1)*x^(-p/2); fl := clip(f,x,0,p/2-1);    # Note A
    gh := clip(g,x,p/2,p-1)*x^(-p/2); gl := clip(g,x,0,p/2-1);    # Note A
    fm := fh + fl; gm := gh + gl;

    # Cases where zm is not needed.
    if b < p/2 then
        return ClippedKaratsubaPolynomialProduct(fl,gl,x,a,  b,  gmul)
    elif a > 3*p/2-2 then
        return ClippedKaratsubaPolynomialProduct(fh,gh,x,a-p,b-p,gmul)
               * x^p
    fi;

    # Need all products.
    restrict :=   # return i or the closest interval endpoint.
        proc(i) if i < 0 then 0 elif i > p-2 then p-2 else i fi end:
    zha := restrict(a - p);   zhb := restrict(b - p);
    zma := restrict(a - p/2); zmb := restrict(b - p/2);
    zla := restrict(a);       zlb := restrict(b);

    # Expand high and low product ranges as needed for zm.
    zha := min(zha, zma); zhb := max(zhb, zmb);
    zla := min(zla, zma); zlb := max(zlb, zmb);
    zh  := ClippedKaratsubaPolynomialProduct(fh,gh,x,zha,zhb,gmul);
    zl  := ClippedKaratsubaPolynomialProduct(fl,gl,x,zla,zlb,gmul);
    zm  := ClippedKaratsubaPolynomialProduct(fm,gm,x,zma,zmb,gmul)
         - zh - zl;

    # Combine and clip
    return clip(zh*x^p + zm*x^(p/2) + zl, x, a, b)               # Note A
end
```

Fig. 4. Clipped Karatsuba polynomial product

Together, these cases give the result for the top $1/2^\ell$ fraction. A similar argument gives the result for the bottom. □

Note that for $p = 2^N$, we have $K(p)/3^{\ell-1} = 3^{N-\ell+1} + 1$, an integer.

The idea of pushing the clipping range down onto the Karatsuba sub-products is straightforward to implement, as shown in Figure 4. As for the classical case, if polynomials are represented using arrays, the lines annotated with `Note A` are performed with array operations. The parameter `gmul` is the coefficient multiplication function and `gdegree` gives the degree of the polynomial in whatever representation is used. In practice, one would have a cut over to classical

Table 1. Coefficient multiplications count for $f \times g$ by clipped Karatsuba multiplication

$a\backslash b$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	1	3	5	9	11	15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
1		3	5	9	11	15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
2			5	9	11	15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
3				9	11	15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
4					11	15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
5						15	19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
6							19	27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
7								27	29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
8									29	33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
9										33	37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
10											37	45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
11												45	49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
12													49	56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
13														56	64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
14															64	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
15																80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
16																	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
17																		56	56	56	56	56	56	56	56	56	56	56	56	56	56
18																			48	48	48	48	48	48	48	48	48	48	48	48	48
19																				44	44	44	44	44	44	44	44	44	44	44	44
20																					36	36	36	36	36	36	36	36	36	36	36
21																						32	32	32	32	32	32	32	32	32	32
22																							28	28	28	28	28	28	28	28	28
23																								26	26	26	26	26	26	26	26
24																									18	18	18	18	18	18	18
25																										14	14	14	14	14	14
26																											10	10	10	10	10
27																												8	8	8	8
28																													4	4	4
29																														3	3
30																															1

multiplication for small sizes rather than recursing all the way down to single coefficients.

Table 1 shows how many coefficient multiplications are required by the above implementation to multiply the two polynomials

$$
\begin{aligned}
f = \quad & \begin{bmatrix} -89 & 56 \\ -96 & 72 \end{bmatrix} x^{15} + \begin{bmatrix} -8 & 64 \\ -32 & 61 \end{bmatrix} x^{14} + \begin{bmatrix} 45 & 66 \\ 69 & 76 \end{bmatrix} x^{12} \\
& + \begin{bmatrix} -96 & 47 \\ 15 & -85 \end{bmatrix} x^{11} + \begin{bmatrix} -96 & 62 \\ -74 & -65 \end{bmatrix} x^{10} + \begin{bmatrix} -92 & 54 \\ -18 & -64 \end{bmatrix} x^{9} + \begin{bmatrix} -56 & -28 \\ 56 & -8 \end{bmatrix} x^{8} \\
& + \begin{bmatrix} 23 & -31 \\ -85 & 94 \end{bmatrix} x^{7} + \begin{bmatrix} -45 & -58 \\ 73 & -70 \end{bmatrix} x^{6} + \begin{bmatrix} -6 & -7 \\ 72 & 4 \end{bmatrix} x^{5} + \begin{bmatrix} -64 & 61 \\ 10 & 45 \end{bmatrix} x^{4} \\
& + \begin{bmatrix} -29 & -43 \\ -95 & 16 \end{bmatrix} x^{3} + \begin{bmatrix} 31 & -9 \\ -42 & 28 \end{bmatrix} x^{2} + \begin{bmatrix} -52 & -87 \\ -51 & -27 \end{bmatrix} x + \begin{bmatrix} -48 & -33 \\ -55 & -22 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
g = \quad & \begin{bmatrix} 1 & 75 \\ 7 & -15 \end{bmatrix} x^{15} + \begin{bmatrix} -22 & 43 \\ 85 & 25 \end{bmatrix} x^{14} + \begin{bmatrix} -29 & -90 \\ -38 & 3 \end{bmatrix} x^{13} + \begin{bmatrix} 39 & -92 \\ 0 & 18 \end{bmatrix} x^{12} \\
& + \begin{bmatrix} 56 & 41 \\ -53 & 6 \end{bmatrix} x^{11} + \begin{bmatrix} -10 & 53 \\ -8 & 83 \end{bmatrix} x^{10} + \begin{bmatrix} 58 & -98 \\ 61 & 1 \end{bmatrix} x^{9} + \begin{bmatrix} -28 & 7 \\ 17 & 36 \end{bmatrix} x^{8} \\
& + \begin{bmatrix} -64 & 16 \\ -58 & 64 \end{bmatrix} x^{7} + \begin{bmatrix} -76 & -66 \\ 83 & 76 \end{bmatrix} x^{6} + \begin{bmatrix} 6 & 3 \\ 34 & 8 \end{bmatrix} x^{5} + \begin{bmatrix} -80 & -71 \\ -15 & 88 \end{bmatrix} x^{4} \\
& + \begin{bmatrix} -9 & -83 \\ 77 & 28 \end{bmatrix} x^{3} + \begin{bmatrix} 59 & -28 \\ 48 & 94 \end{bmatrix} x^{2} + \begin{bmatrix} 40 & -91 \\ -34 & 32 \end{bmatrix} x + \begin{bmatrix} 8 & 39 \\ 9 & -28 \end{bmatrix}
\end{aligned}
$$

We see the cost of the clipped Karatsuba depends on how close the clipping interval is to the center of the product. If $a \le b \le p$ then, it depends on b, and if $p \le a \le b$ it depends on a. If the interval spans the center, then no savings are achieved.

Clipped FFT Polynomial Multiplication

The basic scheme for FFT-based multiplication of polynomials f and g starts with zero padded coefficient vectors for f and g of dimension $N = 2^{\lceil \log_2(\deg f + \deg g + 1) \rceil}$,

$$
\begin{aligned}
v_f &= [f_0, \ldots, f_{\deg f}, 0, \ldots 0] \\
v_g &= [g_0, \ldots, g_{\deg g}, 0, \ldots 0].
\end{aligned}
$$

Then FFTs in an appropriate field are computed

$$
\begin{aligned}
\tilde{v}_f &= \mathrm{FFT}(v_f) \\
\tilde{v}_g &= \mathrm{FFT}(v_g).
\end{aligned}
$$

The FFT of the product is computed as

$$
\tilde{v}_{fg} = [(\tilde{v}_f)_0 \times (\tilde{v}_g)_0, \ldots, (\tilde{v}_f)_{N-1} \times (\tilde{v}_g)_{N-1}],
$$

where $\times$ denotes multiplication in the chosen field. The coefficient vector for the product is then the inverse FFT of $\tilde{v}_{fg}$. Asymptotically, an FFT requires fewer than $\frac{3}{2} N \log_2 N$ coefficient multiplications. This yields a polynomial multiplication cost triple that, from the two forward and one inverse FFT.

Whereas with classical and Karatsuba multiplication, computing unneeded terms has an $O(N^p)$ cost, computing extra FFT terms has only a quasi-linear cost. So in many situations the straightforward methods of Section 3 may be used. To compute a clipped multiplication it will sometimes be possible to avoid computing some of the coefficients of $\tilde{v}_{fg}$. The butterfly transformation of the Cooley-Tukey algorithm [2] does not have every coefficient of $\mathrm{FFT}^{-1}(\tilde{v})$ depend on every coefficient of $\tilde{v}$. For example, for $N = 16$ the dependencies are shown in Table 2. Thus, depending on the clipping range it is possible to avoid computing some coefficients of $\widetilde{fg}$.

We do not pursue this in detail here since our focus in this paper is on products from general purpose applications — but our observation shows that

Table 2. Coefficient dependencies for inverse FFT on 16 elements

i	$\mathbf{FFT^{-1}}(\tilde{v}_i)$ **dependencies**
0	$\tilde{v}_0$
1	$\tilde{v}_0$, $\tilde{v}_1$
2	$\tilde{v}_0$, $\tilde{v}_2$
3	$\tilde{v}_1$, $\tilde{v}_2$, $\tilde{v}_3$
4	$\tilde{v}_0$, $\tilde{v}_4$
5	$\tilde{v}_1$, $\tilde{v}_4$, $\tilde{v}_5$
6	$\tilde{v}_2$, $\tilde{v}_4$, $\tilde{v}_6$
7	$\tilde{v}_3$, $\tilde{v}_5$, $\tilde{v}_6$, $\tilde{v}_7$
8	$\tilde{v}_0$, $\tilde{v}_8$
9	$\tilde{v}_1$, $\tilde{v}_8$, $\tilde{v}_9$
10	$\tilde{v}_2$, $\tilde{v}_8$, $\tilde{v}_{10}$
11	$\tilde{v}_3$, $\tilde{v}_9$, $\tilde{v}_{10}$, $\tilde{v}_{11}$
12	$\tilde{v}_4$, $\tilde{v}_8$, $\tilde{v}_{12}$
13	$\tilde{v}_5$, $\tilde{v}_9$, $\tilde{v}_{12}$, $\tilde{v}_{13}$
14	$\tilde{v}_6$, $\tilde{v}_{10}$, $\tilde{v}_{12}$, $\tilde{v}_{14}$
15	$\tilde{v}_7$, $\tilde{v}_{11}$, $\tilde{v}_{13}$, $\tilde{v}_{14}$, $\tilde{v}_{15}$

even for extreme precision there is scope for special treatment where clipped products are required.

In the situation where $f \times g$ is of size $2N \times N$, if only the middle third of the product is required, then the method of Hanrot, Quercia and Zimmerman [3] may be applied, saving a factor of 2. It remains an open question how far this technique can be generalized. Nonetheless, there will be clipping ranges and argument lengths where it is worthwhile to pad the factors to use this method.

Additionally, the concept of a truncated FFT [5] can be useful. It computes all of the terms, but avoids unnecessary multiplications. Other work [6] shows how multiplication may be performed when the arguments may be decomposed into blocks.

5 Clipped Integer Multiplication

The methods for clipped polynomial products can be adapted for the computation of clipped integer products by an analysis of the required number of lower guard digits for carries. We show how this can be done for classical multiplication.

Clipped Classical Integer Multiplication

The idea here is really simple. To clip the product to digits from a to b you form a naïve clipped product from $a - G$ to b and then only keep digits from a up. So G is a number of guard digits. The sole issue is selecting a value for G. And indeed understanding if there is any value of G that guarantees correct results.

This amounts to a need to understand how far carries can propagate through the process of the addition that combines partial products. While in general addition there is no limit to how far carries can propagate, here there are limits, and they flow from the fact that if B is the base the largest a coefficient product can be is $(B-1)^2$ and the high half of that is quite a lot smaller than B.

Theorem 2. *To compute the $[a..b]$ clipped product of two base-B integers f and g to within 1 unit in position a using classical multiplication with partial products $f \cdot g_i B^i$ requires at most $G = \min(a, \lceil \log_B \operatorname{prec} g \rceil + 1)$ guard digits.*

By this we mean that the $[a..b]$ clipped product requires summing only the columns for $(fg)_i$, for $a - G \le i \le b$, and the columns $[a-G..a-1]$ are used solely to determine a carry.

Proof. Let $\operatorname{prec} f = s+1$ and $\operatorname{prec} g = t+1$, and let c_k be the carry into the k-th column in the full multiplication tableau,

$$\begin{array}{cccccccccc}
c_{s+t+1} & c_{s+t} & c_{s+t-1} & & \cdots & c_s & c_{s-1} & & \cdots & c_0 = 0 \\
 & & & & & f_s \cdot g_0 & f_{s-1} \cdot g_0 & & \cdots & f_0 \cdot g_0 \\
 & & & & \iddots & & & & & \iddots \\
 & & & \iddots & & & & & \iddots & \\
 & & f_s \cdot g_{t-1} & f_{s-1} \cdot g_{t-1} & \cdots & \cdots & & f_0 \cdot g_{t-1} & & \\
 & f_s \cdot g_t & f_{s-1} \cdot g_t & \cdots & \cdots & f_0 \cdot g_t & & & &
\end{array}$$

The sum for the k-the column is

$$s_k = c_k + \sum_{i=0}^{k} f_i g_{k-i} \le c_k + \operatorname{prec} g \cdot (B-1)^2.$$

The equation uses the fact that $u_i = 0$ for $i \notin [0..\operatorname{prec} u)$, and the inequality takes into account both that $f_i g_j \le (B-1)^2$ and that $g_{k-i} = 0$ when $i > k$ (*i.e.* the empty lower right hand triangle in the tableau). The carry into the next column is

$$c_{k+1} = \lfloor s_k / B \rfloor \le s_k / B$$

so

$$B\ c_{k+1} \le c_k + \operatorname{prec} g \cdot (B-1)^2.$$

If there is a maximum carry, $c_{\max}$, then it must satisfy

$$B\, c_{\max} \le c_{\max} + \operatorname{prec} g \cdot (B-1)^2$$

so we have

$$c_{\max} \le \operatorname{prec} g \cdot (B-1).$$

It follows that at most $\lceil \log_B \operatorname{prec} g \rceil + 1$ guard digits are required to guarantee the correct value carried into the clipping range. No guard digits can be taken before position 0, therefore clipped classical multiplication for integers needs to sum columns $[a-G..b]$, for $G = \min(a, \lceil \log_B \operatorname{prec} g \rceil + 1)$, and the columns $[a-G..a-1]$ are used only to compute c_a. □

```
ClippedClassicalIntegerProduct := proc(f0, g0, B, a, b)
    local f, g, pf, pg, nGuard, carry, s, k, t, i, i0;

    # Ensure pg <= pf
    f := f0; pf := iprec(f, B);
    g := g0; pg := iprec(g, B);
    if pg > pf then f,g := g,f;  pf,pg := pg,pf fi;

    # Compute number of guard digits
    nGuard := min(a, ceil(log[B](iprec(g, B))) + 1);

    # Form column sums
    carry := 0;
    s     := 0;
    for k from a - nGuard to b do
        if   k < pg then i0 := 0
        elif k < pf then i0 := k - pg
        else             i0 := k - pf
        fi;

        t := carry;
        for i from i0 to k do t := t + icoeff(f,B,i) * icoeff(g,B,k-i) od;

        carry, t := iquo(t, B), irem(t, B);
        if i >= a then s := isetcoeff(s, B, k, t) fi;
    od;
    return clip(s, B, a, b)
end:

# These would be O(1) or O(b-a) array operations in C.
iprec     := (n, B)       -> ceil(log[B](n+1)):
icoeff    := (n, B, i)    -> irem(iquo(n, B^i), B):
isetcoeff := (n, B, i, v) -> n + v*B^i:
clip      := (n, B, a, b) -> irem(iquo(n, B^a), B^(b-a+1)):
```

Fig. 5. Clipped classical integer product

Note that in general a carry can have multiple digits. With modern computers, the base can be chosen to be large (*e.g.* 2^{64}) so $\lceil \log_B \operatorname{prec} g \rceil$ will be 1 for all practical problems and 2 guard digits will suffice. The details are shown in Figure 5.

6 Combining Methods

Some of the methods we have described require recursive multiplications of parts. The recursive calls need not use the same multiplication method. So there will be platform-dependent thresholds to determine when to use which method. We note some additional considerations below.

Issues in Mulders Multiplication

Mulders [9] considered short multiplication of power series, *i.e.* keeping just the top N terms of the product of two series each of length N. His scheme could save perhaps 20% to 30% of the time that would have been spent had the result been generated by using Karatsuba to form a full product and then just discarding the unwanted low terms.

Full multiplication of the two inputs computes unwanted parts of the result that correspond to half the partial products that classical multiplication would use. Mulders performs a smaller full multiply that still computes many unwanted terms, and which also leaves some necessary parts of the result incomplete. Henriot and Zimmermann [4] investigated just what proportion of the full multiplication would be optimal, but for our purposes it will suffice to approximate that as $0.7N$ by $0.7N$. The parts of the result not computed by this will have the form of a couple of additional instanced of the short product problem, so get handled by recursion.

When this scheme is used for integer multiplication the issue of carries from a discarded low part of a product arise. As with the same issue when using simple classical multiplication this can be handled by using some guard digits, and because Mulders works with and then discards low partial products beyond those used in a classical scheme a bound that is good enough for classical can be applied here.

The idea of using a fast complete multiplication that computes somewhat more of a product than will be needed and discarding the excess, but then needing to fill in some gaps, can be generalized beyond $N \times N$ cases and beyond the case where exactly the top (or bottom) half of a product is needed, and the original Mulders overlap fraction can in general guide usage – however all additional cases could need fresh analysis to find the exact optimal value for that parameter.

Different Methods for Different Ranges

We now return to a discussion on our motivating problem, computing clipped integer products in the approximation scheme described in [12]. There, at various points, it is desired to compute clipped products for the most significant, least significant and middle parts of an asymmetric multiplication. Depending on the size of the prefix or suffix, we can use clipped classical, Karatsuba or FFT for these. For the middle third, a more complex scheme may be used, as shown in Figure 6, illustrating a product $f \times g$. (Diagonal lines from upper left to lower right are multiples of g by a term of f. Diagonal lines from upper right to lower left are multiples of f by a term of g.) This is motivated by Mulders but needs to clip at both high and low ends. Use of a $3N$ by $2N$ variant of Toom-Cook [1,11] gives rather close to the levels of overlap at each and that would be good for Mulders.

A different and somewhat extreme case would be where the number of output digits (say h) required is much smaller than the size of either input. In that case a classical multiplication can clearly deliver a result in something like $h \times N$ but a decomposition of the strip as in Figure 7 turns out on analysis to start to win once h is significantly larger that the threshold at which Karatsuba breaks even for all the tiled sub-products.

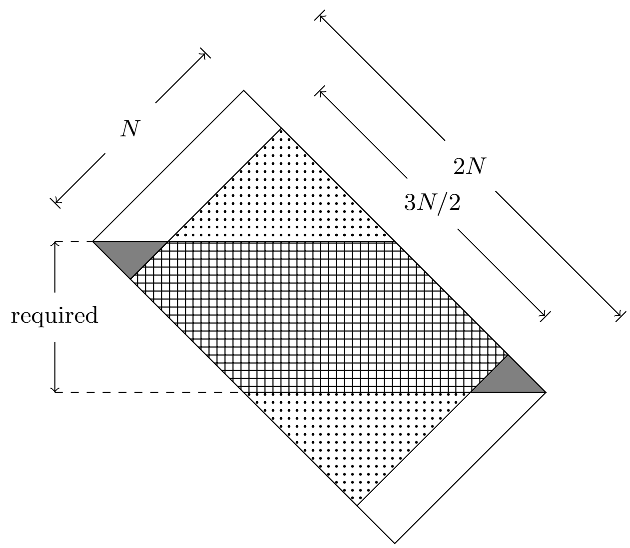

Fig. 6. The desired region of the asymmetric $N \times 2N$ product. Horizontal lines are terms of equal degree, and the dashed lines show the range of terms desired. The $N \times 3N/2$ rectangle is calculated and the cross hatched area is used and the dotted area is not used. The solid gray areas are calculated separately.

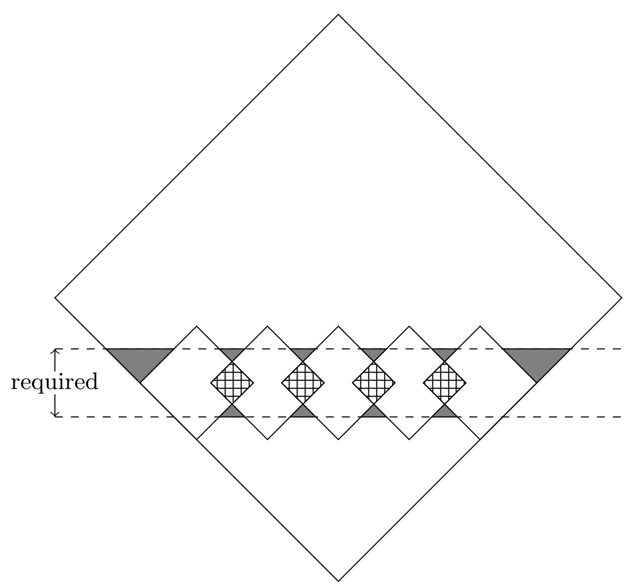

Fig. 7. A scheme to compute the middle digits. The squares are calculated by a size-appropriate method (classical, Karatsuba, FFT). The hatched areas of overlap are double counted, so must be subtracted. The solid areas are calculated sparately.

7 Further Thoughts and Conclusions

We have studied the problem of computing a specified portion of integer and polynomial products, giving some algorithms, cost analysis, bounds on carry propagation and examples.

We have shown a number of methods, and which is least costly depends on the size of the values to be multiplied and on the interval of the product desired. For example, if a very few digits of the middle of an integer product are desired, then clipped classical multiplication will give the result in time linear in the input with a good constant factor. On the other hand, if a substantial fraction of the leading terms of a huge polynomial product are desired, then a straightforward bottom clipped FFT product of the reverse polynomials may be best.

In general, one needs a polyalgorithm that reduces to specific schemes in particular regimes. We believe the boundaries between these regimes, and the remaining gaps, have not been particularly well explored, and the present work is a step toward filling them.

References

1. Cook, S.A.: On the Minimum Computation Time of Functions. Ph.D. thesis, Harvard University (1966)
2. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965)
3. Hanrot, G., Quercia, M., Zimmermann, P.: The middle product algorithm I: Speeding up the division and square root of power series. Applicable Algebra in Engineering, Communication and Computing **14**, 415–438 (2004)
4. Hanrot, G., Zimmermann, P.: A long note on Mulders' short product. J. Symbolic Computation **37**, 391–401 (2004)
5. van der Hoeven, J.: The truncated Fourier transform and applications. In: Proc. ISSAC 2004. pp. 290—296. ACM, New York (2004)
6. van der Hoeven, J., Lecerf, G.: On the complexity of multivariate blockwise polynomial multiplication. In: Proc. ISSAC 2012. pp. 211—218. ACM, New York (2012)
7. Karatsuba, A., Yu., O.: Multiplication of many-digital numbers by automatic computers. Proceedings of the USSR Academy of Sciences **145**, 293–294 (1962), translation in the academic journal Physics-Doklady, 7 (1963), pp. 595–596
8. Knuth, D.E.: The Art of Computer Programming, Volume 4b: Combinatorial Algorithms, Part 2. Addison-Wesley, Boston (2022)
9. Mulders, T.: On short multiplication and division. In: Proceedings AAECC 11, 1. pp. 69–88 (2000)
10. Norman, A.C., Watt, S.M.: A symbolic computing perspective on software systems **arxiv:2406.09085**, 1–18 (2024)
11. Toom, A.L.: The complexity of a scheme of functional elements realizing the multiplication of integers. Soviet Mathematics Doklady **3**, 714–716 (1963)
12. Watt, S.M.: Efficient generic quotients using exact arithmetic. In: Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC 2023). pp. 535–544. ACM, New York (2023)
13. Watt, S.M.: Efficient quotients of non-commutative polynomials. In: 25^{th} International Workshop on Computer Algebra in Scientific Computing (CASC 2023). pp. 370–392. Springer Cham LNCS 14139, New York (2023)