Efficient Multiprecision Floating Point Multiplication with Exact Rounding

Werner Krandick*

Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
krandick@risc.uni-linz.ac.at

Jeremy R. Johnson†

Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104
jjohnson@king.mcs.drexel.edu

Abstract

An algorithm is described for multiplying multiprecision floating point numbers. The returned result is equal to the floating point number obtained by rounding the exact product. Software implementations of multiprecision floating point multiplication can reduce the computing time by a factor of two if they do not compute the low order digits of the product of the two mantissas. However, these algorithms do not necessarily provide exactly rounded results. The algorithm described in this paper is guaranteed to produce exactly rounded results and typically obtains the same savings.

1 Introduction

We present an algorithm for multiplying multiprecision floating point numbers. The returned result is equal to the floating point number obtained by rounding the exact product. A rounding operation which satisfies this requirement is called *exact rounding*. Exact rounding provides a well defined, implementation independent semantics for floating point arithmetic. For this reason, floating point arithmetic with exact rounding has been advocated even if it is more costly than algorithms which do not produce exactly rounded results [6]. The IEEE standard 754 for hardware binary floating point arithmetic supports exact rounding [4]. In software implementations of multiprecision floating point arithmetic exact rounding can be accomplished by calculating the full product of the mantissas. Our algorithm produces exact roundings yet typically requires only half as much computing time as the full product computation.

We use the "short product", outlined in Exercise 15 in Section 4.3.1 of Knuth [6], which uses classical multiplication but suppresses the computation of low order digits when the two mantissas are multiplied. This method will asymptotically save about half of the computing time if both operands and the result have the same precision. We will show that the short product can also be used in conjunction with Karatsuba's algorithm for fast integer multiplication. Other multiprecision packages either use the full product (Maple V [2], Reduce [9]), or they use the short product (MPFUN [1]) but do not provide exact rounding.

The short product computes $M + 2$ digits, where M is the precision of the result. Our algorithm is based on an efficient test to determine if the result obtained by rounding the short product is the same as that obtained by rounding the full product. The test uses two guard digits and an estimate for the error obtained by using the short product instead of the full product. In certain cases we also need to know if the error was exactly equal to zero or not. This can be done by counting the number of trailing zeros in the input operands. This part of the test is similar to an algorithm proposed by Santoro, Berwick, and Horowitz in [8] for computing the sticky bit.

If the test succeeds then we can return the exactly rounded result using the short product instead of the full product. We show that if the radix used to represent the mantissas is sufficiently large, the test will almost always succeed. The radix will be sufficiently large in a software implementation of multiprecision floating point arithmetic where the radix is typically chosen to be as close to the largest single precision integer as possible. If the test fails additional guard digits can be computed and a similar test can be ap-

*Supported by the Austrian Science Foundation (Grant M022-PHY).

†Supported by NSF-grant CCR-9211016.

plied using the additional guard digits. However, in some very rare situations the full product is needed to compute the exact rounding.

An earlier version of this paper appeared in the proceedings of the 11th IEEE Symposium on Computer Arithmetic [7]. The short product was used to provide *optimal directional rounding*, i.e. to return the smallest (greatest) floating point number greater (smaller) than or equal to the true product. In the present paper we show that the short product can also be used to provide *nearest/even rounding*, i.e. to return the floating point number which is closest to the true product; in case the true product is halfway in between two floating point numbers the one with an even mantissa is returned. In addition we provide an improved condition for detecting whether the short product can be used to compute the exactly rounded result. Along with the improved test we present a tighter analysis of the effectiveness of the test, and discuss the underlying assumptions on the distribution of digit products upon which this analysis is based.

The final improvement deals with the implementation of the short product. The discussion of the implementation of the short product was enhanced by considering array implementations and by providing additional empirical computing times. For the first time we present a recursive formulation of the short product which can be easily combined with Karatsuba's algorithm for fast integer multiplication. We analyze this version of the short product and again provide empirical computing times.

Section 2 reviews basic notation used in the paper. Section 3 gives a concise presentation of the main idea. Section 4 supplies rigorous definitions and proofs for the claims made in Section 3. Section 5 describes an algorithm for floating point multiplication with exact rounding which uses the short product; the probability that the short product indeed suffices is also analyzed. The analysis involves the probability that the short product of two normalized numbers is normalized; an estimate is calculated in Section 6. Section 7 shows how the short product can be implemented efficiently; both, classical multiplication and the recursive use of Karatsuba's algorithm are discussed. Finally, Section 8 presents a theoretical and empirical analysis of the algorithms discussed in Section 7.

2 Notation

Let a be a non-zero integer; let β be an integer > 1. The representation of a with respect to base β has *β-length*

$$L_\beta(a) = \lfloor \log_\beta |a| \rfloor + 1.$$

In the sequel β will be assumed to be a power of 2, namely

$$\beta = 2^\zeta \text{ with } \zeta \geq 1. \tag{1}$$

Typically ζ depends on the wordsize of the computer. For all non-zero integers a

$$L_\beta(a) = \lceil L_2(a)/\zeta \rceil.$$

Definition 1 *An integer $a \neq 0$ is called* normalized *if*

$$L_\beta(a) = L_2(a)/\zeta.$$

Lemma 1 *The following conditions are equivalent for non-zero integers a:*

1. *a is normalized.*
2. *The high-order bit in the high-order β-digit of a is 1.*
3. *With $n = L_\beta(a)$,*
$$\frac{\beta}{2}\beta^{n-1} \leq |a| < \beta^n.$$
4. *With $n = L_\beta(a)$,*
$$2^{\zeta n-1} \leq |a| < 2^{\zeta n}.$$

Definition 2 *The* β-order $o_\beta(a)$ *of an integer a is defined as*

$$o_\beta(a) = \max\{k \mid \beta^k | a\}$$

The β-order of an integer is the number of trailing zeros in its representation with respect to base β.

For convenience all integers occurring in this paper will assumed to be positive. Of course, the results carry over to arbitrarily signed integers. Furthermore, A and B will always stand for positive integers consisting of n_1 and n_2 β-digits, respectively:

Definition 3

$$A = \sum_{h=1}^{n_1} a_h \beta^{h-1} \quad \textit{and} \quad B = \sum_{j=1}^{n_2} b_j \beta^{j-1},$$

with $n_1, n_2 \geq 1$, $0 \leq a_h, b_j < \beta$ for all h, j, and $0 < a_{n_1}, b_{n_2}$.

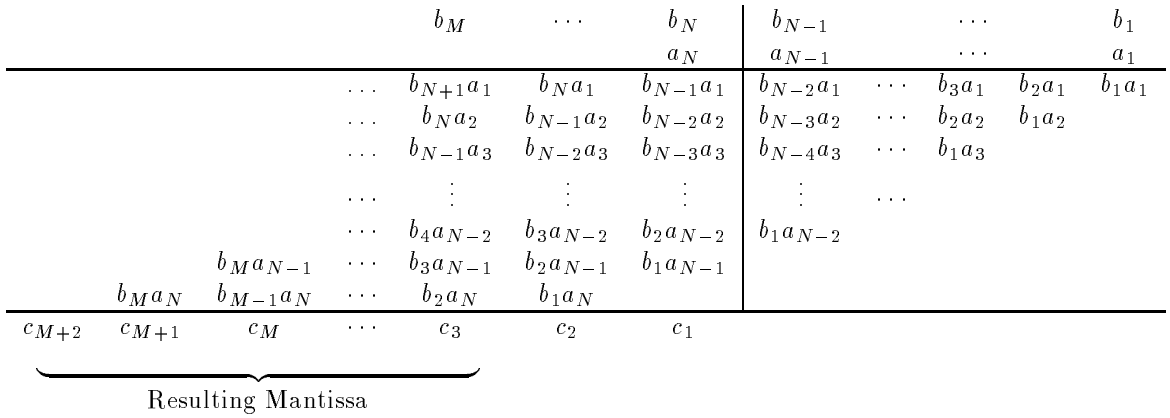

Figure 1: Illustration of the short product for the case $n_1 \leq n_2$ with $N = \min(n_1, n_2)$ and $M = \max(n_1, n_2)$.

3 The Idea

This section outlines the main idea. For precise definitions and proofs see Section 4. Let two floating point numbers be given with normalized mantissas A and B and with precisions $L_\beta(A) = n_1$ and $L_\beta(B) = n_2$. We want to compute the floating point product to precision $M = \max(n_1, n_2)$. Let $N = \min(n_1, n_2)$. The "short product"

$$C = \sum_{h+j \geq N} a_h b_j \beta^{h+j-N}.$$

can be computed using just

$$n_1 n_2 - (\frac{N^2 - 3N}{2} + 1)$$

digit multiplications (Lemma 9. Non-classical multiplication is discussed in Section 7). C will have $M+2$ β-digits, where $M = \max(n_1, n_2)$ (Proposition 1).

$$C = \sum_{k=1}^{M+2} c_k \beta^{k-1},$$

where the c_k's are β-digits and $c_{M+2} \neq 0$.

The short product can be visualized by looking at the usual pencil and paper method for multiplying integers. Figure 1 illustrates the case $n_1 \leq n_2$ with $N = \min(n_1, n_2)$ and $M = \max(n_1, n_2)$. The indices of the digit products in each column sum to j, for $j = 2, \ldots, M+N$. The error obtained by suppressing the low order digit products (those to the right of the column with $j = N$) is $\leq (N-2)\beta^{N-1}$ (Lemma 2). Therefore the M leading β-digits of the product are the same as the M leading β-digits of the short product if the error term added to c_2 does not produce a carry, corrupting c_3. This clearly implies that the leading $M\zeta$ binary digits of the product are the same as the leading $M\zeta$ binary digits of the short product if C is normalized and the error term added to c_2 does not produce a carry into the low order bit of c_3. Moreover, it will be shown (Proposition 2) that the $M\zeta$ leading bits are also the same if C is not normalized and the error term added to c_2 does not produce a carry into the high order bit of c_2.

For exact directional rounding it suffices to know the $M\zeta$ leading bits of C and whether the suppressed digit products were all zero. The latter is the case if and only if

$$o_\beta(A) + o_\beta(B) \geq N - 2$$

(Proposition 3).

The probability that C suffices for exact directional rounding thus depends on the probability P_n that C is normalized and on the distribution of values of c_2. The probability P_n is approximately 61% (cf. Section 6), and the values of c_2 seem to be uniformly distributed (cf. Section 5). If they are indeed, then our method will fail with probability

$$p = P_n \frac{N-2}{\beta} + (1 - P_n)\frac{2(N-2)}{\beta}$$

(Proposition 5). In practice this number will be very small. For example, if $N = 100$ and $\beta = 2^{32}$ then $p < 10^{-7}$.

When nearest rounding is desired the considerations are slightly different (Section 5), but the probability that it suffices to compute the short product is the same.

4 Proof of Correctness

We define a generalization of the integer product.

Definition 4 *Let integers A, B be given as in Definition 3. The* short product *of A and B with respect to an integer N is defined as*

$$C = A \times_N B = \sum_{h+j \geq N} a_h b_j \beta^{h+j-N}. \qquad (2)$$

Remark 1 *Some elementary properties of the short product are listed for later use. Properties 1,2 and 3 are special cases of the definition; they will allow an elegant statement of the recursion formula in Proposition 7.*

1. *For $N < 2$, $A \times_N B = AB\beta^{2-N}$.*
2. *$A \times_2 B = AB$.*
3. *For $N > n_1 + n_2$, $A \times_N B$ is the empty sum and thus equal to 0 .*
4. *$A \times_{(n_1+n_2)} B = a_{n_1} b_{n_2}$.*
5. *For $M \leq N$, $\beta^{N-2}(A \times_N B) \leq \beta^{M-2}(A \times_M B)$.*
6. *For $A \leq \tilde{A}$, $A \times_N B \leq \tilde{A} \times_N B$.*
7. *$A \times_N B = B \times_N A$.*

Proposition 1 *Let $2 \leq N \leq n_1 + n_2$. The length of $A \times_N B$ is*

1. *in general*

$$n_1 + n_2 - N + 1 \leq L_\beta(A \times_N B) \leq n_1 + n_2 - N + 2,$$

2. *for normalized A and B*

$$L_\beta(A \times_N B) = n_1 + n_2 - N + 2,$$

and — more strongly —

$$\zeta(n_1 + n_2 - N + 1) + (\zeta - 1) \leq L_2(A \times_N B) \leq \zeta(n_1 + n_2 - N + 2).$$

Proof. Using the statements of Remark 1 the first inequality in statement 1 is obtained as follows:

$$\begin{aligned} \beta^{N-2}(A \times_N B) &\geq \beta^{(n_1+n_2)-2}(A \times_{(n_1+n_2)} B) \\ &\geq \beta^{n_1+n_2-2}(\beta^{n_1-1} \times_{(n_1+n_2)} \beta^{n_2-1}) \\ &= \beta^{n_1+n_2-2}, \end{aligned} \qquad (3)$$

and so, $L_\beta(A \times_N B) \geq (n_1 + n_2 - 1) - (N - 2) = n_1 + n_2 - N + 1$.

The second inequality in statement 1 is proven similarly:

$$\begin{aligned} \beta^{N-2}(A \times_N B) &\leq \beta^{2-2}(A \times_2 B) \\ &= AB \\ &\leq (\beta^{n_1} - 1)(\beta^{n_2} - 1) \\ &< \beta^{n_1+n_2}, \end{aligned}$$

and so, $L_\beta(A \times_N B) \leq (n_1 + n_2) - (N - 2) = n_1 + n_2 - N + 2$.

If A and B are normalized, Inequality (3) can be strengthened to

$$\beta^{N-2}(A \times_N B) \geq$$

$$\beta^{n_1+n_2-2}(\frac{\beta}{2}\beta^{n_1-1} \times_{(n_1+n_2)} \frac{\beta}{2}\beta^{n_2-1}) = \frac{\beta}{4}\beta^{n_1+n_2-1},$$

so that $L_2(A \times_N B) \geq (\zeta - 1) + \zeta(n_1 + n_2 - 1) - \zeta(N - 2) = \zeta(n_1 + n_2 - N + 1) + (\zeta - 1)$. This proves the first inequality in statement 2. The second inequality follows from the second inequality in statement 1. □

Definition 5 *The* error ϵ_N *of the short product $A \times_N B$ is defined as*

$$\epsilon_N = AB - \beta^{N-2}(A \times_N B).$$

Remark 2 *For $2 \leq N \leq n_1 + n_2$ the error ϵ_N is the accumulated contribution to AB of the digit products suppressed in $A \times_N B$, namely*

$$\epsilon_N = \sum_{h+j<N} a_h b_j \beta^{h+j-2}. \qquad (4)$$

Lemma 2 *Let $2 \leq N \leq n_1 + n_2$. Then*

$$\epsilon_N \leq (N - 2)\beta^{N-1}.$$

Proof. The bound is derived from Equation (4):

$$\begin{aligned} \epsilon_N &= \sum_{h+j<N} a_h b_j \beta^{h+j-2} \\ &\leq \sum_{l=2}^{N-1} \sum_{h+j=l} (\beta - 1)^2 \beta^{l-2} \\ &= \sum_{l=2}^{N-1} (l - 1)(\beta - 1)^2 \beta^{l-2} \\ &= \sum_{l=0}^{N-3} (l + 1)(\beta - 1)^2 \beta^{l} \\ &= (N - 2)\beta^{N-1} + (1 - N)\beta^{N-2} + 1 \\ &\leq (N - 2)\beta^{N-1} \square \end{aligned}$$

Lemma 3 *Let $2 \le N \le n_1 + n_2$. Then*

$$\beta^{N-2}(A \times_N B) \le AB \le \beta^{N-2}(A \times_N B) + (N-2)\beta^{N-1}.$$

Proof. The first inequality follows from combining statement 2 of Remark 1 with statement 5 (letting $M = 2$). The second inequality follows from Lemma 2 and Definition 5. □

Proposition 2 *Let A and B be normalized, and let $n_1, n_2 \ge 2$. Let $N = \min(n_1, n_2)$, let $M = \max(n_1, n_2)$, and let*

$$C = A \times_N B = \sum_{k=1}^{M+2} c_k \beta^{k-1} \tag{5}$$

with β-digits c_k. Note that according to Proposition 1 $L_2(C)$ is either equal to $(M+2)\zeta$ or to $(M+2)\zeta - 1$.

1. *If $L_2(C) = (M+2)\zeta$ and*

$$c_2 + (N-2) < \beta \tag{6}$$

then the $M\zeta$ leading bits of C and the $M\zeta$ leading bits of AB agree.

2. *If $L_2(C) = (M+2)\zeta - 1$ let γ be the number formed by the $(\zeta - 1)$ low order bits of c_2. Now*

$$\gamma + (N-2) < \beta/2 \tag{7}$$

implies that the $M\zeta$ leading bits of C and the $M\zeta$ leading bits of AB agree.

Proof. In case 1 the $M\zeta$ leading bits of C are in the β-digits $c_{M+2}, \ldots, c_3$. Inequality (6) thus implies that the $M\zeta$ leading bits of C and $C + (N-2)\beta$ are the same. Therefore the $M\zeta$ leading bits of $\beta^{N-2}C$ and $\beta^{N-2}C + (N-2)\beta^{N-1}$ are the same, and Lemma 3 implies the assertion.

In case 2 the $M\zeta$ leading bits of C are in the β-digits $c_{M+2}, \ldots, c_3$ and in the high order bit of c_2. The rest of the proof is analogous to case 1.

Remark 3 *If — in the situation of Proposition 2 — $n_1 = 1$ or $n_2 = 1$ then $N = 1$, and $C = A \times_N B = AB\beta$. In this case the $M\zeta$ leading binary digits of C trivially agree with the $M\zeta$ leading binary digits of AB.*

Remark 4 *A variant of Proposition 2 is needed when the short product is to be used for floating point multiplication with nearest rounding. The variant provides a condition for the $M\zeta + 1$ leading bits of C to agree with the $M\zeta + 1$ leading bits of AB. The corresponding statements are obtained by replacing Inequality (6) with $c_2 + (N-2) < \beta/2$ and Inequality (7) with $\gamma + (N-2) < \beta/4$.*

Proposition 3 *Let $2 \le N \le n_1 + n_2$. Then*

$$\epsilon_N = 0 \quad \textit{if and only if} \quad o_\beta(A) + o_\beta(B) \ge N - 2.$$

Proof. Due to Equation (4) the error ϵ_N vanishes if and only if

$$(a_k b_l \ne 0 \Rightarrow k + l \ge N). \tag{8}$$

It thus suffices to show that (8) is equivalent to $o_\beta(A) + o_\beta(B) \ge N - 2$.

1. Assume (8). Let $\kappa = o_\beta(A)$, $\lambda = o_\beta(B)$. Then $a_{\kappa+1} \ne 0$ and $b_{\lambda+1} \ne 0$, hence $a_{\kappa+1} b_{\lambda+1} \ne 0$, and so, because of (8), $\kappa + \lambda + 2 \ge N$.

2. Assume $o_\beta(A) + o_\beta(B) \ge N - 2$. Assume further that $a_k b_l \ne 0$. Then $k \ge o_\beta(A) + 1$ and $l \ge o_\beta(B) + 1$, and hence $k + l \ge o_\beta(A) + o_\beta(B) + 2 \ge N$. □

5 Floating Point Multiplication Using the Short Product

In this section we show how the short product can be used to implement multiprecision floating point multiplication with exact rounding. We show that with a very high probability the short product can be used to provide the same result as would have been obtained by rounding the exact result computed with the full product. Moreover, Proposition 2 and Proposition 3 provide a test to show when the two results must agree. In the few cases where the test fails, the full product can be computed to return the exact rounding.

We begin by reviewing some elementary properties of floating point arithmetic using the notation from this paper. Let $F = A2^{e-\zeta n_1}$ and $G = B2^{f-\zeta n_2}$, where A and B are as in Definition 3. In addition, we will assume that A and B are normalized, i.e. $\beta/2\beta^{n_1-1} \le A < \beta^{n_1}$ and $\beta/2\beta^{n_2-1} \le B < \beta^{n_2}$. The floating point product of F and G, $F \otimes G$, is defined to be round(FG), where $FG = AB \cdot 2^{e+f-\zeta(n_1+n_2)}$ is the exact product and "round" produces a floating point number of precision $M = \max(n_1, n_2)$ (occasionally the precision will be denoted by a subscript). Any of

the IEEE rounding modes can be specified, i.e. round to nearest and the directional rounding modes round to plus infinity, round to minus infinity, and round to zero.

Let x be a real number. If round to nearest is used, then $\mathrm{round}_p(x)$ will denote the floating point number, of precision p, nearest to x. If x is equally near two floating point numbers the floating point number with even mantissa will be selected. If round to infinity is used, $\mathrm{round}_p(x)$ denotes the smallest floating point number, of precision p, greater than or equal to x, and for round to minus infinity, $\mathrm{round}_p(x)$ denotes the largest floating point number, of precision p, less than or equal to x.

Suppose $2^{e-1} \leq x < 2^e$. Then, if nearest rounding is used the absolute error is bounded by

$$|x - \mathrm{round}_p(x)| \leq 1/2\beta^{-p}2^e = 2^{e-\zeta p-1},$$

which implies that the magnitude of the relative error is bounded by

$$\left|\frac{x - \mathrm{round}_p(x)}{x}\right| \leq 2^{e-\zeta p-1}/2^{e-1} = 2^{-\zeta p} = \beta^{-p}.$$

Rather than computing the product of two floating point number exactly an estimate for $F \otimes G$ can be obtained by rounding the short product. More specifically, letting $N = \min(n_1, n_2)$ and $M = \max(n_1, n_2)$, compute

$$\mathrm{round}_M(F \times_N G) = \mathrm{round}_M(A \times_N B) \cdot 2^{e+f-\zeta(M+2)}.$$

The following proposition, which is an easy consequence of Proposition 2, summarizes the well known observation that the relative error obtained from rounding the short product is not much worse than the relative error obtained from rounding the full product.

Proposition 4 *Let F and G be normalized floating point numbers of precision p. Then,*

$$\left|\frac{FG - round(F \times_p G)}{FG}\right| \leq \left(\frac{4(p-2)}{\beta} + 1\right)\beta^{-p}.$$

Proof.

$$\begin{aligned} |FG - F \times_p G| &= AB2^{e+f-2\zeta p} - A \times_p B2^{e+f-(p+2)\zeta} \\ &= \left(AB - A \times_p B2^{\zeta(p-2)}\right)2^{e+f-2\zeta p} \\ &= \left(AB - A \times_p B\beta^{p-2}\right)2^{e+f-2\zeta p} \\ &\leq (p-2)\beta^{p-1}2^{e+f-2\zeta p} \end{aligned}$$

Since F and G are normalized, $FG \geq \frac{1}{4}\beta^{2p}2^{e+f-2\zeta p}$, and

$$\left|\frac{FG - F \times_p G}{FG}\right| \leq 4(p-2)\beta^{-p-1}.$$

Furthermore, by the definition of the short product and nearest rounding

$$\begin{aligned} &\left|\frac{F \times_p G - \mathrm{round}(F \times_p G)}{FG}\right| \\ \leq &\left|\frac{F \times_p G - \mathrm{round}(F \times_p G)}{F \times_p G}\right| \leq \beta^{-p}. \end{aligned}$$

Combining these inequalities proves the proposition.

Rather than being satisfied with the fact that the relative error obtained by rounding the short product is nearly the same as the relative error obtained by rounding the full product, with negligible extra cost, Propositions 2 and 3 can be used in almost all cases to guarantee that the short product produces the same rounding as the full product. In the cases where one is not certain that the short product produces the exact rounding, one can resort to the full product. We will show that the probability that the full product will be required is very nearly zero.

Examples where the full product is indeed needed to compute the directionally rounded result can be constructed as follows. Let $A = B = \beta^N - 1 = \sum_{i=1}^{N}(\beta - 1)\beta^{i-1}$. Then $AB = \beta^{2N} - 2\beta^N + 1 = \beta^N(\beta^N - 2) + 1$, $\mathrm{round}_N^+(AB) = \beta^N - 1$, and $\mathrm{round}_N^-(AB) = \beta^N - 2$. The full product is needed, because one needs to know the low order bit of AB in order to tell that $AB > \beta^N(\beta^N - 2)$. In decimal notation the product $9999 \cdot 9999 = 99980001$ illustrates this point.

Examples where the full product is needed to compute the nearest rounded result can be constructed as follows. Assume that β is even. Let $A = \beta/2\beta^{N-1} + (\beta^{N-1} - 1)$ and $B = (\beta - 2)\beta^{N-1} + (\beta^{N-1} - 1) = (\beta - 1)\beta^{N-1} - 1 = (\beta - 2)\beta^{N-1} + \sum_{i=1}^{N-1}(\beta - 1)\beta^{i-1}$. Then $AB = (\beta/2 + 1)(\beta - 1)\beta^{2N-2} - 3/2\beta^N + 1 = \beta^{N-1}(\beta/2\beta^{N+1} + \beta/2\beta^N - \beta^{N-1} - 3/2\beta) + 1 = \beta^{N-1} \cdot D + 1$, where $D = \beta/2\beta^{N+1} + (\beta/2 - 1)\beta^N + (\beta - 2)\beta^{N-1} + \sum_{i=3}^{N-1}(\beta - 1)\beta^{i-1} + (\beta - 2)\beta + \beta/2$, hence $\mathrm{round}_N(AB) = \lfloor D/\beta \rfloor + 1$, while $\mathrm{round}_N(A \times_p B) = \lfloor D/\beta \rfloor$ for all $2 < p \leq N$. In decimal notation the product $5999 \cdot 8999 = 53985001$ illustrates this point.

Tables 1 and 2 show how the short product can be used for exact directional and nearest rounding. The tables use C, c_1, c_2 and γ as defined in Proposition 2. According to Proposition 1 C is either normalized ($L_2(C) = (M+2)\zeta$) or it must be shifted left by one bit position. The predicate $\epsilon_N = 0$ can be evaluated at low cost using Proposition 3. The condition $N - 2 < \beta/2$ is implicitely assumed for two cases ($c_2 > \beta/2, \gamma > \beta/4$) of Table 2. In most cases the returned result is $H = \mathrm{round}(C2^{e+f-\zeta(M+2)})$, where

Table 1: Directional Rounding.

C normalized		result
$c_2 + (N-2) < \beta$	$c_2 \neq 0 \vee c_1 \neq 0$	H
	$c_2 = 0 \wedge c_1 = 0 \wedge \epsilon_N = 0$	H
	$c_2 = 0 \wedge c_1 = 0 \wedge \epsilon_N \neq 0$	H^+
$c_2 + (N-2) \geq \beta$		fail
C not normalized		
$\gamma + (N-2) < \beta/2$	$\gamma \neq 0 \vee c_1 \neq 0$	H
	$\gamma = 0 \wedge c_1 = 0 \wedge \epsilon_N = 0$	H
	$\gamma = 0 \wedge c_1 = 0 \wedge \epsilon_N \neq 0$	H^+
$\gamma + (N-2) \geq \beta/2$		fail

Table 2: Nearest Rounding.

C normalized	result
$c_2 > \beta/2$	H
$c_2 = \beta/2 \wedge c_1 \neq 0$	H
$c_2 = \beta/2 \wedge c_1 = 0 \wedge \epsilon_N = 0$	H
$c_2 = \beta/2 \wedge c_1 = 0 \wedge \epsilon_N \neq 0$	H^+
$c_2 + (N-2) < \beta/2$	H
$c_2 + (N-2) \geq \beta/2$	fail
C not normalized	
$\gamma > \beta/4$	H
$\gamma = \beta/4 \wedge c_1 \neq 0$	H
$\gamma = \beta/4 \wedge c_1 = 0 \wedge \epsilon_N = 0$	H
$\gamma = \beta/4 \wedge c_1 = 0 \wedge \epsilon_N \neq 0$	H^+
$\gamma + (N-2) < \beta/4$	H
$\gamma + (N-2) \geq \beta/4$	fail

"round" is the respective rounding function. Sometimes H^+, the right floating point neighbor of H is returned. When the algorithm can not determine the exactly rounded result it returns "fail".

The only cases where we can not determine the exact directionally rounded result from the short product are

$$\beta - (N-2) \leq c_2 < \beta \tag{9}$$

when C is normalized, and

$$\beta/2 - (N-2) \leq \gamma < \beta/2 \tag{10}$$

when C is not normalized.

The only cases where we cannot determine the nearest rounding from the short product are

$$\beta/2 - (N-2) \leq c_2 < \beta/2 \tag{11}$$

when C is normalized, and

$$\beta/4 - (N-2) \leq \gamma < \beta/4 \tag{12}$$

when C is not normalized.

Inequalities (9), (10), (11), and (12), can be used to estimate the probability that one can not determine the exact rounding from the short product.

Proposition 5 *Let $F = A2^{e-\zeta n_1}$ and $G = B2^{f-\zeta n_2}$ be two normalized floating point numbers of precision n_1 and n_2 respectively. Let $M = \max(n_1, n_2)$, $N = \min(n_1, n_2)$, $H = F \times_N G$ and let $C = A \times_N B$. Let P_n be the probability that C is normalized, and let P_s be the probability that C is not normalized (and hence must be shifted left by one bit position). Then, if c_2 is uniformly distributed, the probability that the short product can not be used to determine the exact rounding of FG is*

$$P_n \frac{N-2}{\beta} + P_s \frac{2(N-2)}{\beta}.$$

Section 6 estimates the probabilities P_n and P_s used in Proposition 5.

The validity of Proposition 5 depends on the assumption that c_2 is uniformly distributed. While this assumption seems reasonable, we have not been able to prove it. Nonetheless exhaustive computer experiments were performed to calculate the distribution of c_2 for small values of N and β. The short product was calculated for for all normalized inputs of precision N and radix β, and the occurrences of each possible value of c_2 was counted. This information is summarized in Table 3, Table 4, and Table 5. The data shows that the distributions are nearly uniform. However, the distribution is slightly weighted towards zero, and the even digits occur more frequently than the odd digits.

Table 3: Distribution of Digits i in c_2: $N = 3$, $\beta = 4$

i	count	probability
0	272	26.56 %
1	254	24.80 %
2	256	25.00 %
3	242	23.63 %

Table 4: Distribution of Digits i in c_2: $N = 3$, $\beta = 8$

i	count	probability
0	8414	12.84 %
1	8214	12.53 %
2	8258	12.60 %
3	8246	12.58 %
4	8226	12.55 %
5	8042	12.27 %
6	8126	12.39 %
7	8010	12.22 %

Table 5: Distribution of Digits i in c_2: $N = 3$, $\beta = 16$

i	count	probability
0	263180	6.27 %
1	262156	6.25 %
2	262766	6.26 %
3	262178	6.25 %
4	263344	6.28 %
5	262240	6.25 %
6	262454	6.26 %
7	261558	6.24 %
8	264180	6.30 %
9	261108	6.23 %
10	261522	6.24 %
11	261086	6.22 %
12	261968	6.25 %
13	261024	6.22 %
14	261834	6.24 %
15	261706	6.24 %

6 Probability of a Normalized Short Product

We want to estimate the probability that the short product of two normalized integers is normalized.

If A and B are normalized positive integers consisting of n_1 and n_2 β-digits, respectively, we have

$$\frac{\beta}{2}\beta^{n_1-1} \le A < \beta^{n_1}$$

and

$$\frac{\beta}{2}\beta^{n_2-1} \le B < \beta^{n_2}.$$

We will assume that A and B are uniformly distributed in their respective intervals. Let $N = \min(n_1, n_2)$ and $M = \max(n_1, n_2)$.

According to Proposition 1 the short product $A \times_N B$ consists of $M + 2$ β-digits; hence it is normalized if and only if

$$\frac{\beta}{2}\beta^{M+1} \le A \times_N B$$

or – equivalently – if and only if

$$\frac{\beta}{2}\beta^{M+N-1} \le \beta^{N-2}(A \times_N B).$$

From Lemma 3 we have

$$AB - (N-2)\beta^{N-1} \le \beta^{N-2}(A \times_N B),$$

hence the probability that $A \times_N B$ is normalized is at least as large as the probability that

$$\frac{\beta}{2}\beta^{M+N-1} \le AB - (N-2)\beta^{N-1}$$

or – equivalently – as the probability that

$$\frac{\beta}{2}\beta^{n_1+n_2-1} + (N-2)\beta^{N-1} \le AB.$$

We will derive an estimate for the latter probability by considering the continuous analogue of the situation.

Proposition 6 *Let x be uniformly distributed in the interval $I_x = [\frac{\beta}{2}\beta^{n_1-1}, \beta^{n_1})$ and let y be uniformly distributed in the interval $I_y = [\frac{\beta}{2}\beta^{n_2-1}, \beta^{n_2})$. Then the probability that $\frac{\beta}{2}\beta^{n_1+n_2-1} + (N-2)\beta^{N-1} \le xy$ is equal to*

$$2 - \left(2 + \frac{4(N-2)\beta^{N-1}}{\beta^{n_1+n_2}}\right)\ln 2.$$

Table 6: Probability of Normalized Result.

ζ	N	$N \cdot \zeta$	short product	full product
2	3	6	.580078	.583984
2	4	8	.605103	.605347
3	3	9	.609299	.609665
2	5	10	.611660	.611721
4	3	12	.613188	.613215
3	4	12	.613206	.613215
2	6	12	.613210	.613215
3	5	15	.613644	.613644

Proof. We will compute the ratio of the area of the part of $I_x \times I_y$ which lies above the curve

$$H(\xi) = (\frac{\beta}{2}\beta^{n_1+n_2-1} + (N-2)\beta^{N-1})\frac{1}{\xi}$$

to the total area of $I_x \times I_y$. Since

$$\int_{I_x} H(\xi)d\xi = (\frac{\beta}{2}\beta^{n_1+n_2-1} + (N-2)\beta^{N-1})\ln 2,$$

the area of the part of $I_x \times I_y$ which lies above the curve is

$$\frac{1}{2}\beta^{n_1+n_2} - (\frac{1}{2}\beta^{n_1+n_2} + (N-2)\beta^{N-1})\ln 2$$

The total area of $I_x \times I_y$ is

$$\frac{1}{4}\beta^{n_1+n_2},$$

and the desired ratio is obtained as stated.

Remark 5 *The probability that the full product of A and B is normalized can be estimated in the same way; one obtains $2 - 2\ln 2 \approx .613705638$. The same value is obtained if one lets $n_1, n_2 \to \infty$ in Proposition 6.*

Table 6 lists the true probabilities for the short (full) product of two normalized numbers to be normalized. The entries were obtained by multiplying all pairs of normalized numbers with N β-digits, where $\beta = 2^\zeta$ is a small power of 2, and by counting the number of normalized products. One can prove that both probabilities are monotone increasing in the number of bits $N \cdot \zeta$, and that for any N the probability of a normalized short product is less than the probability of a normalized full product.

7 Efficient Implementation

In this section we describe two methods for implementing the short product and several techniques for improving each implementation. The first implementation is similar to the classical algorithm for multiplying integers. The second implementation is based on a recursive formulation of the short product, which can be easily combined with Karatsuba's fast integer multiplication algorithm (see [6] for a discussion of Karatsuba's algorithm).

The following lemma is useful for implementations of the short product calculation.

Lemma 4 *Assume*

$$2 \le N \le n_1 + n_2 \ \wedge \ 1 \le n_1 \ \wedge \ 1 \le n_2.$$

Then the lowest index k such that a_k contributes to $A \times_N B$ is

$$k = \max(N - n_2, 1),$$

and the lowest index l such that b_l contributes to $A \times_N B$ is

$$l = \max(N - n_1, 1).$$

Proof. Due to statement 7 of Remark 1 only the first statement needs to be proved. Let k be such that a_k contributes to the short product $A \times_N B$. Since a_k contributes to the short product if and only if there exists an l between 1 and n_2 such that $k + l \ge N$, $k \ge N - n_2$, and since $1 \le k \le n_1$, $k \ge \max(N - n_2, 1)$. Furthermore, since $N \le n_1 + n_2$, $N - n_2 \le n_1$, and $\max(N - n_2, 1)$ is in the range of A indices □

This lemma can be used to rewrite the short product as a nested sum, which leads to a program implementing the short product in a nested loop.

$$C = \sum_{h=\max(N-n_2,1)}^{n_1} \sum_{j=\max(N-h,1)}^{n_2} a_h b_j \beta^{h+j-N} \qquad (13)$$

This nested summation corresponds to computing the partial products in Figure 1 row by row, starting with row $\max(N - n_2, 1)$, where each row is computed from right to left starting with either the partial product whose indices sum to N or the partial product whose b-index is 1.

The efficiency of this program can be improved if the loop is unrolled so that it is no longer necessary to compute $\max(N - h, 1)$ in each iteration of the outer loop.

No unrolling is needed if $N > n_1$, because then Equation (13) can be written as

$$C = \sum_{h=\max(N-n_2,1)}^{n_1} \sum_{j=N-h}^{n_2} a_h b_j \beta^{h+j-N}. \qquad (14)$$

However, if $N \leq n_1$ the nested sum in Equation (13) must be unrolled into two nested sums.

$$\begin{aligned} C &= \sum_{h=\max(N-n_2,1)}^{N-1} \sum_{j=N-h}^{n_2} a_h b_j \beta^{h+j-N} + \\ &\quad \sum_{h=N}^{n_1} \sum_{j=1}^{n_2} a_h b_j \beta^{h+j-N}. \end{aligned} \qquad (15)$$

In the special case where $n_1 \leq n_2$ and $N = \min(n_1, n_2) = n_1$ this simplifies further to

$$\begin{aligned} C &= \sum_{h=1}^{N-1} \sum_{j=N-h}^{n_2} a_h b_j \beta^{h+j-N} + \\ &\quad \sum_{j=1}^{n_2} a_N b_j \beta^{j}. \end{aligned} \qquad (16)$$

Alternatively the short product can be implemented recursively. To see how to do this, we describe a recursive formulation of the short product. The benefit of this approach is that it provides a simple way of incorporating Karatsuba's fast integer multiplication algorithm.

Lemma 5

$$A\beta^n \times_N B = A \times_{N-n} B$$

Proof.

$$\begin{aligned} A\beta^n \times_N B &= \sum_{h+n+j \geq N} a_h b_j \beta^{h+n+j-N} \\ &= \sum_{h+j \geq N-n} a_h b_j \beta^{h+j-(N-n)} \\ &= A \times_{N-n} B \qquad \square \end{aligned}$$

Proposition 7 (Recursion Formula) *Let* $A = A_1\beta^k + A_0$ *with* $A_0 < \beta^k$, *and let* $B = B_1\beta^l + B_0$ *with* $B_0 < \beta^l$. *Then*

$$\begin{aligned} A \times_N B &= A_1 \times_{N-k-l} B_1 + A_1 \times_{N-k} B_0 + \\ &\quad A_0 \times_{N-l} B_1 + A_0 \times_N B_0. \end{aligned} \qquad (17)$$

Proof. While the short product is not distributive in general, it follows from Definition 4 that

$$(A_1\beta^k + A_0) \times_N B = A_1\beta^k \times_N B + A_0 \times_N B.$$

The short products on the right hand side can be decomposed in the same way. Finally, Lemma 5 produces the desired equation. $\square$

The base cases for this recursive formula are provided by statements 1 and 3 of Remark 1. Proposition 7 can be applied recursively to represent the short product as a sum of certain full products. The full products can be computed using Karatsuba's algorithm for fast integer multiplication. It is beneficial to apply the recursion formula in the situation where k and l are chosen such that in each recursion step $A_1 \times_{N-k-l} B_1$ is a full product with $L_\beta(A_1)L_\beta(B_1)$ being maximal and $A_0 \times_N B_0 = 0$. The following lemma can be used to compute k and l satisfying these conditions.

Lemma 6 *Let* $F(x,y) = xy$. *Then the maximum value of* $F(x,y)$ *over the domain* $D = \{(x,y) \in \mathbf{R}^2 |\ 1 \leq x \leq n_1$ *and* $1 \leq y \leq n_2\}$ *subject to the constraint that* $x + y \leq N$ *where* $2 \leq N \leq n_1 + n_2$, *is obtained at the point*

$$\begin{array}{ll} (N - n_2, n_2) & \text{if } N/2 < N - n_2 \\ (N/2, N/2) & \text{if } N - n_2 \leq N/2 \leq n_1 \\ (n_1, N - n_1) & \text{if } n_1 < N/2 \end{array}$$

If (x,y) *are restricted to be integers, then the maximal value is obtained at the point*

$$\begin{array}{ll} (N - n_2, n_2) & \text{if } N/2 < N - n_2 \\ (\lfloor N/2 \rfloor, \lceil N/2 \rceil) \text{ or} & \\ (\lceil N/2 \rceil, \lfloor N/2 \rfloor) & \text{if } N - n_2 \leq N/2 \leq n_1 \\ (n_1, N - n_1) & \text{if } n_1 < N/2 \end{array}$$

Proof. It is clear that $F(x,y)$, subject to the constraint that $x + y \leq N$ must obtain its maximum value on the boundary $x + y = N$. Maximizing $F(x,y)$, on the domain D, subject to $x + y = N$ is equivalent to maximizing $f(x) = F(x, N-x) = x(N-x)$ on the domain $\max(1, N - n_2) \leq x \leq \min(n_1, N-1)$. If the maximum occurs in the interior of this interval, then x must be a root of the derivative of $f(x)$, namely $x = N/2$. Since $N/2$ is a global maximum, the only times we must check the boundary is when $N/2$ is not in the domain of x. This proves the first part of the lemma. Since $f(x)$ is increasing as x approaches $N/2$ (from either side), $\lfloor N/2 \rfloor + \lceil N/2 \rceil = N$, and $F(x,y) = F(y,x)$ one easily obtains the second part of the lemma $\square$

Letting $x = (n_1 - k + 1)$ and $y = (n_2 - l + 1)$ in the previous lemma we obtain the following proposition, which shows how to select k and l in the application of Proposition 7 to computing the short product.

Proposition 8 *Let $n_1, n_2 \geq 1$ and $2 \leq N \leq n_1 + n_2$. Define*

$$S = \{(k,l) | 1 \leq k \leq n_1, 1 \leq l \leq n_2, k + l \geq N\}$$

and

$$F(k,l) = (n_1 - k + 1)(n_2 - l + 1).$$

A global maximum of F on S is attained at

$$(k_0, l_0) \in S,$$

where

$$k_0 = \max(m, \min(k', M)),$$
$$l_0 = N - k_0$$

with

$$m = \max(1, N - n_2),$$
$$M = \min(n_1, N - 1),$$
$$k' \in \left\{ \left\lfloor \frac{n_1 - n_2 + N}{2} \right\rfloor, \left\lceil \frac{n_1 - n_2 + N}{2} \right\rceil \right\}.$$

To see how this proposition is applied, let k_0 and l_0 be chosen as in Proposition 8, so that $k_0 + l_0 = N$ and $(n_1 - k_0 + 1)(n_2 - l_0 + 1)$ is maximal. Let $A = A_1\beta^k + A_0$, $A_0 < \beta^k$, and $B = B_1\beta^l + B_0$, $B_0 < \beta^l$, where $k = k_0 - 1$ and $l = l_0 - 1$. Then $L_\beta(A_1) = (n_1 - k_0 + 1)$, $L_\beta(A_0) = k_0 - 1$, $L_\beta(B_1) = (n_2 - l_0 + 1)$, and $L_\beta(B_0) = l_0 - 1$. Since $(k_0 - 1) + (l_0 - 1) = N - 2 < N$, Remark 1 implies that $A_0 \times_N B_0 = 0$. Since $k_0 + l_0 = N$, $A_1 \times_{N-(k_0+l_0)+2} B_1 = A_1 \times_2 B_1 = A_1 B_1$. Therefore,

$$\begin{aligned} A \times_N B &= (A_1\beta^{k_0 - 1} + A_0)(B_1\beta^{l_0 - 1} + B_0) \\ &= A_1B_1 + A_1 \times X_{N-k_0+1} B_0 + \\ &\quad A_0 \times X_{N-l_0+1} B_1 \end{aligned} \quad (18)$$

Moreover, $L(A_1)L(B_1) = (n_1 - k_0 + 1)(n_2 - l_0 + 1)$, and by the choice of k_0 and l_0 this is maximal. In general, the number of digit products in $A_0 \times_N B_0$, which is equal to $(k_0 - 1)(l_0 - 1)$, may not be maximal. However, in the special case where, $n_1 = n_2$ is an even number, $(k_0 - 1)(l_0 - 1)$ is also maximal. If $n_1 = n_2$ is an odd number, a small change in the recursion formula also allows both the number of digits in $A_0 \times_N B_0$ and A_1B_1 to be simultaneously maximized. In general it is more complicated to accomplish this, but it can be done at the cost of an extra recursive short product computation.

In the application of Karatsuba's algorithm to the short product, Karatsuba's algorithm is used to compute the full product A_1B_1 in Equation (18). This idea is then applied recursively to the two remaining short products in Equation (18), until it no longer pays to apply Karatsuba, at which time the classical short product is used. Even though the choice of k_0 and l_0 in Proposition 8 forces the lengths of the inputs to Karatsuba's algorithm to nearly be the same, there are inputs where the lengths are not well suited to our implementation of Karatsuba's algorithm. For example, if $n_1 = n_2 = N = 2^m$, then $L(A_1) = L(B_1) = (2^m - 2^{m-1} + 1) = 2^{m-1} + 1$. This forces Karatsuba's algorithm to perform an extra recursive call where one of the inputs is of length 1. This situation can be remedied by using a slightly different application of the recursion formula. Suppose $n_1 = n_2 = N$ and $A = A_1\beta^k + A_0$ and $B = B_1\beta^k + B_0$, where $k = \lfloor n/2 \rfloor$. Then

$$\begin{aligned} A \times_N B &= A_1 \times_{N-2k} B_1 + A_1 \times_{N-k} B_0 \\ &\quad + A_0 \times_{N-k} B_1 + A_0 \times_N B_0 \end{aligned} \quad (19)$$

If N is even, then $A_1 \times_{N-2k} B_1 = A_1B_1\beta^2$ and $A_0 \times_N B_0 = a_kb_k$. If N is odd, then $A_1 \times_{N-2k} B_1 = A_1B_1\beta$ and $A_0 \times_N B_0 = 0$.

8 Computing Time

In this section we compare the percentage improvement of the short product over the full product for various implementations. Empirical data is compared to theoretical estimates obtained by counting the number of digit products. All algorithms were implemented in Portable C using the `SACLIB`-library [3] of computer algebra programs. The radix used, $\beta = 2^{29}$, is determined by the requirements of `SACLIB`. All experiments were performed running optimized code under the ULTRIX V4.2A operating system on a Dec Workstation 5000/240, with a 40 MHz CPU.

We begin with two propositions which provide a theoretical estimate of the computing time for the short product. The first proposition counts the number of digit products in the short product. The second proposition estimates the computing time for the recursive formulation of the short product using Karatsuba's algorithm for computing full products.

Proposition 9 *Let $2 \leq N \leq n_1 + n_2$. Computing the*

short product $A \times_N B$ using Equation (2) requires

$$\mu_N = n_1 n_2 - (\frac{N^2 - 3N}{2} + 1) \qquad (20)$$

digit multiplications.

Proof. It suffices to show that the sum on the right hand side of Equation (4) consists of $\frac{N^2-3N}{2}+1$ terms. For each $l \in \{2, \ldots, N-1\}$ there are $l-1$ pairs (h, j) such that $h + j = l$, hence the sum in (4) consists of

$$\sum_{l=2}^{N-1} (l-1) = \frac{N-2}{2}(N-1) = \frac{N^2 - 3N + 2}{2}$$

terms. □

We will now show that computing the short product with k applications of the recursion formula (17) will take about

$$\tau_k = 1 - \frac{1}{3} \cdot \left(\frac{2}{3}\right)^{k-1}$$

of the time required for computing the full product using Karatsuba's algorithm; we assume that the products $A_1 B_1$ in the recursion formula are computed using Karatsuba's algorithm.

Let n be a power of 2. Let $K(n)$ be the number of digit products computed by Karatsuba's algorithm when multiplying two n-digit numbers A and B. Let $S(n)$ be the number of digit products computed in a short product computation using the formula $A \times_n B = A_1 B_1 + A_1 \times_{n/2} B_1 + A_0 \times_{n/2} B_1$, and Karatsuba's algorithm to compute $A_1 B_1$. Then we are in the situation of Proposition 10.

Proposition 10 *Let $K(n)$ and $S(n)$ be defined for non-negative powers of 2 such that for all arguments n*

$$K(2n) = 3K(n)$$

and

$$S(2n) = K(n) + 2S(n).$$

If $S(1) = K(1)/2$ then, for $k > 0$,

$$\frac{S(2^k)}{K(2^k)} = 1 - \frac{1}{3} \cdot \left(\frac{2}{3}\right)^{k-1}.$$

Proof. Clearly, $K(2^k) = 3^k K(1)$. By induction on k

$$S(2^k) = 2^k S(1) + (3^k - 2^k) K(1).$$

Because of $S(1) = K(1)/2$ this reduces to

$$S(2^k) = (3^k - 2^{k-1}) K(1).$$

Now

$$\frac{S(2^k)}{K(2^k)} = \frac{(3^k - 2^{k-1})K(1)}{3^k K(1)} = 1 - \frac{1}{3} \cdot \left(\frac{2}{3}\right)^{k-1}$$

□

First we compare the short product, implemented with Equation (13), to an implementation of the full product using the classical algorithm for integer multiplication. This comparison is made using both lists and arrays to represent integers. Second we use the unrolling given in Equation (15) to improve the implementation of the array version of the short product. Finally, we present timings of an implementation of the short product using Karatsuba's fast integer multiplication algorithm and the recursion formula in Proposition 7, and compare this to an implementation of the full product using Karatsuba's algorithm.

Since we want to use the short product for floating point multiplication we time computations of $A \times_N B$ for $N = min(n_1, n_2)$. For simplicity we restrict the experiments to operands A and B of equal length, so we have $n_1 = n_2 = N$. In addition to computing the short product, our programs determine the sticky bit using Proposition 3.

Algorithm 1 is the `SACLIB` algorithm, *IPROD*, for classical computation of the full product; it uses a list representation of integers. Algorithm 2 computes the short product according to Equation (13) using lists. Algorithm 3 is a classical full product computation using arrays. Algorithm 4 computes the short product according to Equation (13) using arrays. Algorithm 5 computes the short product according to Equation (16) which holds in the special case where $N = n_1 \leq n_2$; it uses arrays. For details of the implementation see [3] and [5].

Table 7 lists the computing time ratios t_i/t_j, in percent, where t_i is the computing time for Algorithm i and t_j is the computing time for Algorithm j. Table 7 also lists the ratio

$$\sigma = \frac{\mu_N}{N^2}$$

of the respective numbers of digit products required by the short product and the full product. According to Lemma 9 the number μ_N of digit products required by Algorithm 2 was obtained from Equation (20) by letting $n_1 = n_2 = N$.

The ratio t_2/t_1 is the ratio of the list implementation of the short product compared to the list implementation of the full product. Initially, the full product is faster than the short product. This is due to the extra overhead required by the shortproduct. For $N \geq 4$, the list version of the short product becomes

Table 7: Speed-up Due to Short Product.

N	computing time ratios in %			
	σ	t_2/t_1	t_4/t_3	t_5/t_3
2	100	126	112	95
3	89	105	97	84
4	81	91	85	75
5	76	83	79	71
10	64	61	64	59
20	57	52	56	53
30	55	48	53	51
40	54	46	52	49
50	53	45	51	49

faster than the full product, and for $N > 10$ the ratio becomes smaller than the ratio, σ, of digit products of the short product compared to the full product.

The ratio t_4/t_3 compares an array implementation of the short product to the full product. Except for $N = 2$, the short product is always faster than the full product, and when $N \geq 10$ the ratio becomes smaller than the ratio of digit products. The ratio t_5/t_3 compares an unrolled array implementation of the short product to the array version of the full product. We see that this version is always faster than the full product and is consistently 5% smaller than the prediction given by the ratio of digit products.

Table 8 compares shows the improvement we obtained using Karatsuba's algorithm. Algorithm 6 uses the implementation of Karatsuba's algorithm, *IPRODK*, provided in `SACLIB`, for computing the full product. Algorithm 7, *ISPRODK*, uses a list implementation of the recursion formula in Equation (19) for computing the short product. This algorithm uses Algorithm 6, *IPRODK*, to compute the full product of the high order parts, A_1B_1.

IPRODK switches to the classical product, *IPROD*, when it no longer pays to use Karatsuba's algorithm. We experimentally computed the optimal cutoff point, on the Dec 5000/240, for which we should switch to *IPROD*. This was determined, by computing the smallest power of two for which one application of Karatsuba's formula provided a reduction of computing time as compared to the classical product. This value was 64. A similar determination was made for the short product version of Karatsuba's algorithm, *ISPRODK*. Ideally we would expect this cutoff to be twice the size of the cutoff for Karatsuba's algorithm, since *IPRODK* is applied to integers which are half the size of the inputs to *ISPRODK*. However, experimentally, the cutoff point for *ISPRODK* turned out to be 256, which is 4 times the cutoff point for

Table 8: Speed-up Due to Karatsuba.

N	computing time ratios in %					
	t_6/t_1	t_7/t_2	t_7/t_6	τ	t_2/t_1	t_2/t_6
128	82	101	59	–	48	58
256	67	97	69	67	47	71
512	50	85	77	78	45	91
1024	39	72	84	85	46	116
2048	30	58	87	90	46	151
4096	20	39	91	93	48	232

IPRODK. The extra factor of two probably comes from the additional overhead of the recursive formulation of the short product.

As in Table 7, the ratio t_i/t_j in Table 8 is the ratio of the computing time for algorithm i to the computing time for algorithm j.

The ratio t_6/t_1 shows the improvement that Karatsuba's algorithm provides over the classical product. This ratio will approach 0. The ratio t_7/t_2 shows the improvement that the short product using Karatsuba's algorithm provides over the classical short product. This ratio will also approach 0; however, at a slower rate than t_6/t_1. The ratio t_7/t_6 shows the ratio of the short product using Karatsuba's algorithm to the full product using Karatsuba's algorithm. Proposition 10 shows that this ratio will eventually approach 1. In fact, the ratios t_7/t_6 are fairly close to the ratios τ predicted by the proposition. The ratio t_2/t_1 is the ratio of the classical short product to the classical full product. Proposition 9 predicts that this ratio should approach 1/2. Finally, the ratio t_2/t_6 compares the classical short product to the full product using Karatsuba's algorithm. The table indicates that, it is preferable to use the full product, computed with Karatsuba's algorithm, rather than the short classical product when $N \approx 1024$. By combining the short product with Karatsuba's algorithm, the range for which the short product provides an improvement over the full product is extended. For our implementation, we see that the short product using Karatsuba's algorithm is approximately 10% faster than the full product computed with Karatsuba's algorithm for inputs that are 4096 β-digits long.

References

[1] David H. Bailey. A portable high performance multiprecision package. Technical report, NASA Ames Research Center, 1992. RNR Technical Report RNR-90-022.

[2] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, New York, 1991.

[3] George E. Collins et al. `SACLIB` user's guide. Technical Report RISC-Linz Report Series Number 93-19, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[4] IEEE. *ANSI/IEEE Standard 754-1985 for binary floating-point arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, 1985. Reprinted in SIGPLAN 22,2,9-25.

[5] J. R. Johnson and W. Krandick. A multiprecision floating point and interval arithmetic package for symbolic computation. Technical Report RISC-Linz Report Series Number 93-20, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[6] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 1981. Seminumerical Algorithms.

[7] W. Krandick and J. R. Johnson. Efficient multiprecision floating point multiplication with optimal directional rounding. In *11th IEEE Symposium on Computer Arithmetic, Windsor, Ontario, 1993*, IEEE Computer Society Press, P.O.Box 3041, Los Alamitos, CA 90720-1264, Phone: 714 821-8380, 1993.

[8] Mark R. Santoro, Gary Berwick, and Mark A. Horowitz. Rounding algorithms for IEEE multipliers. In Milos D. Ercegovac and Earl Swartzlander, editors, *Proceedings 9th Symposium on Computer Arithmetic*, pages 176–183. IEEE Computer Society Press, 1989.

[9] T. Sasaki. An arbitrary precision real arithmetic package in REDUCE. In E. W. Ng, editor, *EUROSAM 79, Colloque International sur les Méthodes de Calcul Symbolique et Algébrique, Marseille, France*, number 72 in Lecture Notes in Computer Science, pages 358–368, Berlin-Heidelberg-New York, 1979. Springer-Verlag.