A lower bound for the size of the result array in a Karatsuba algorithm by R. E. Maeder

Stefan Krah
skrah@acm.org
May 31, 2009

Abstract

This paper attempts to correct the lower bound for the size of the result array in a Karatsuba multiplication algorithm by R. E. Maeder.

Keywords: Karatsuba multiplication, storage allocation

1 Introduction

In [2] R. E. Maeder presents a Karatsuba multiplication algorithm with low storage requirements and a single allocation strategy. For the temporary space he proves a sharp bound for the minimum storage requirements.

This paper focuses on the bound given for the size of the result array, which is too low in some cases.

2 The algorithm

The karatsuba function is a direct quote from [2]. All pointers point to arrays of base $B$ digits. $a$ and $b$ are the factors, $c$ is the result array and $w$ is the temporary work space.

```c
1 void karatsuba(digit *a, digit *b, digit *c, digit *w, int la, int lb)
   // add the product of a and b to c.
   // we assume $la \geq lb > \lceil la/2 \rceil$. c must be $la+lb+1$ in size
   // the array w is used as a work array (temporary storage)
2 {
3 if (la <= 4) { // use naive method
4     long_multiplication(a, b, c, la, lb);
5     return;
6 }
7 m = (la+1)/2; // $\lceil la/2 \rceil$
8 copyto(w + 0, a + 0, m); // $a_0,..,a_{m-1}$ into $w_0,..,w_{m-1}$
9 w[m] = 0; // clear carry digit
10 addto(w + 0, a + m, la - m); // form $a_l+a_h$ into $w_0,..,w_m$
11 copyto(w + (m+1), b + 0, m); // $b_0,..,b_{m-1}$ into $w_{m+1},..,w_{2m}$
12 w[m+1+m] = 0; // clear carry digit
13 addto(w + (m+1), b + m, lb - m); // form $b_l+b_h$ into $w_{m+1},..,w_{2m+1}$
14 // compute $(a_l+a_h)(b_l+b_h)$ into $c_{m,..,c_{3m+1}}$
15 karatsuba(w + 0, w + (m+1), c + m, w + 2*(m+1), m+1, m+1);
16 lt = (la - m) + (lb - m) + 1; // space needed for $a_hb_h$
17 clear(w + 0, lt); // clear result array
18 // compute $a_hb_h$ into $w_0,..,w_{la+lb+2m-1}$
19 karatsuba(a + m, b + m, w + 0, w + lt, la - m, lb - m);
```
The algorithm allocates $la + lb + 1$ units of storage for the result array $c$. However, this is not sufficient in a number of cases. We focus on repeated calls of karatsuba() in line 14 until the base case is reached. Each call consumes $m = \lceil la/2 \rceil$ units of $c$, and the base case call of long_multiplication() needs $2la$ units. If we substitute $n$ for $la$, the total amount of storage required for $c$ by this particular path in the call tree is given by this recursive equation, where $lim$ is the limit for the base case.

$$cspace(n) = \begin{cases} 
2n & \text{if } n \leq lim \\
\left\lfloor \frac{n+1}{2} \right\rfloor + cspace(\left\lfloor \frac{n+1}{4} \right\rfloor + 1) & \text{otherwise}
\end{cases}$$  \hspace{1cm} (1)

Since $la + lb + 1$ might be greater than $cspace(la)$, the minimum allocation for $c$ is given by:

$$\max(cspace(la), la + lb + 1)$$  \hspace{1cm} (2)

In the following subsection we prove a sharp upper bound for $cspace(n)$. If $la \leq lim$, clearly the bound $la + lb + 1$ is valid, so we only consider the case $la > lim$.

**Theorem 3.1.** An upper bound for $cspace(n)$ is given by:

$$cspace(n) \leq 3(\left\lfloor \frac{n+1}{2} \right\rfloor + 1), \text{ for } lim \geq 4, n > lim$$  \hspace{1cm} (3)

**Proof.** A formal proof written for the ACL2 theorem prover [1] is provided in appendix A. This is an outline of the main steps:

For $lim \geq 4$, $n > lim$, $cspace(n)$ terminates with $n$ decreasing in each call. This paves the way for a well founded induction.

Assuming

$$cspace(\left\lfloor \frac{n+1}{2} \right\rfloor + 1) \leq 3(\left\lfloor \frac{n+1}{2} \right\rfloor + 1 + 1) \text{ (hyp1)}$$

we need to show:

$$cspace(n) = \left\lfloor \frac{n+1}{2} \right\rfloor + cspace(\left\lfloor \frac{n+1}{2} \right\rfloor + 1)$$  \hspace{1cm} (4)

$$\leq \left\lfloor \frac{n+1}{2} \right\rfloor + 6 + 3(\left\lfloor \frac{n+1}{4} \right\rfloor)$$  \hspace{1cm} (5)

$$\leq 3(\left\lfloor \frac{n+1}{2} \right\rfloor + 1)$$  \hspace{1cm} (6)
Omitting the details of the formal proof, we state that the term in (5) is indeed less or equal than the term in (6) for $n > 8$. For $n \leq 8$, the formal proof resorts to case analysis to show directly that the theorem holds.

In the case that $n > \text{lim}$, but $\lfloor \frac{n+1}{2} \rfloor + 1 \leq \text{lim}$, we must show:

\[
\text{cspace}(n) = \lfloor \frac{n+1}{2} \rfloor + 2(\lfloor \frac{n+1}{2} \rfloor + 1) \leq 3(\lfloor \frac{n+1}{2} \rfloor + 1)
\]

This is clearly true.

It remains to analyse the other two recursive calls, where $\text{lt}$ units of storage are reserved for the result array $w$.

In line 22, $\text{karatsuba}()$ is called with $\text{la}' = \text{lb}' = \text{m}$, with $\text{lt} = 2\text{la}' + 1$. Since $2\text{la}' + 1$ is a valid bound for $\text{la}' \leq \text{lim}$, and $\text{cspace}(\text{la}') < 2\text{la}' + 1$, this call is safe.

In line 17, $\text{la}' = \text{la} - \text{m}$, $\text{lb}' = \text{lb} - \text{m}$, $\text{la}' \geq \text{lb}'$, with $\text{lt} = \text{la}' + \text{lb}' + 1$. Here $\text{lt}$ might be too low. To amend this, line 15 could be replaced by $\text{lt} = (\text{la} - \text{m}) + (\text{la} - \text{m}) + 1$, or by using (2). The $w$ array has enough space for both options.

References


A The formal proof

As promised, here is the complete formal proof of theorem 3.1 written for the ACL2 theorem prover. This file should also be available as cspace.lisp in the same directory as this paper:

;; Books containing helper theorems.
(include-book "arithmetic/top-with-meta" :dir :system)
(include-book "arithmetic-2/floor-mod/floor-mod" :dir :system)

;; Amount of memory needed for c when following a certain call path.
(defun cspace (n lim)
  (declare (xargs :guard (and (natp n)
                               (natp lim)
                               (<= 4 lim))
               :verify-guards nil))
  (and (<= 4 lim)
       (if (<= (nfix n) lim)
           (* 2 n)
           (let ((m (floor (+ n 1) 2)))
             (+ m (cspace (+ m 1) lim))))))
(defthm natp-cspace
  (implies (and (natp n) (natp lim) (<= 4 lim))
            (natp (cspace n lim)))
  :rule-classes :type-prescription)

(verify-guard cspace)

;; ==============================================================
;; Tight upper bound: 3 * (ceil(n/2) + 1)
;; ==============================================================

;; Rewriting floor in terms of mod frequently helps.
(defthm floor-to-mod
  (implies (and (< 0 m) (rationalp m)
                (rationalp x))
           (equal (floor x m)
                  (- (/ x m) (/ (mod x m) m))))))

(defthm lemma-1a
  (implies (and (< 0 m) (natp m)
                (natp n))
           (<= (+ (* 4 (mod n m))
                (- (* 3 (mod n (* 2 m)))) m))
  :rule-classes :linear)

(defthm lemma-1b
  (implies (and (< 8 n) (natp n))
           (<= (+ (* 3 (floor (+ 1 n) 4))
                (* 2 (floor (+ 1 n) 2))))
  :hints ("Goal" :use ((:instance floor-to-mod
                        (x (+ 1 n))
                        (m 2))
                     (:instance floor-to-mod
                        (x (+ 1 n))
                        (m 4))))
  :rule-classes :linear)

;; For n <= 8 case analysis is required.
(defthm upper-bound-n<=8
  (implies (and (<= 4 lim) (< lim n) (<= n 8)
                (natp lim) (natp n))
           (<= (cspace n lim)
                (* 3 (+ 1 (floor (+ 1 n) 2))))
  :hints ("Subgoal *1/5" :cases ((= n 5) (= n 6) (= n 7) (= n 8)))
  :rule-classes :linear)

;; The upper bound for cspace.
(defthm upper-bound
  (implies (and (<= 4 lim) (< lim n)
                (natp lim) (natp n))
           (<= (cspace n lim)
                (* 3 (+ 1 (floor (+ 1 n) 2))))
  :hints ("Goal" :induct t)
         ("Subgoal *1/2.1" :use (upper-bound-n<=8)))

4