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.

```
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)
{
  if (la <= 4) { // use naive method
          long_multiplication(a, b, c, la, lb);
          return;
  }
  m = (la+1)/2;                      // $\lceil la/2 \rceil$
  copyto(w + 0, a + 0, m);           // $a_0, \ldots, a_{m-1}$ into $w_0, \ldots, w_{m-1}$
  w[m] = 0;                          // clear carry digit
  addto(w + 0, a + m, la - m);       // form $a_l + a_h$ into $w_0, \ldots, w_m$
  copyto(w + (m+1), b + 0, m);       // $b_0, \ldots, b_{m-1}$ into $w_{m+1}, \ldots, w_{2m}$
  w[m+1+m] = 0;                      // clear carry digit
  addto(w + (m+1), b + m, lb - m); // form $b_l + b_h$ into $w_{m+1}, \ldots, w_{2m+1}$
  // compute $(a_l + a_h)(b_l + b_h)$ into $c_m, \ldots, c_{3m+1}$
  karatsuba(w + 0, w + (m+1), c + m, w + 2*(m+1), m+1, m+1);
  lt = (la - m) + (lb - m) + 1;      // space needed for $a_h b_h$
  clear(w + 0, lt);                  // clear result array
  // compute $a_h b_h$ into $w_0, \ldots, w_{la+lb-2m-1}$
  karatsuba(a + m, b + m, w + 0, w + lt, la - m, lb - m);
```

```
18   addto(c + 2*m, w, (la - m) + (lb - m)); // add a_h b_h B^{2m}
19   subfrom(c + m, w, (la - m) + (lb - m)); // subtract a_h b_h B^m
20   lt = m + m + 1;                         // space needed for a_l b_l
21   clear(w + 0, lt);                       // clear result array
     // compute a_l b_l into w_0, ..., w_{2m-1}
22   karatsuba(a, b, w + 0, w + lt, m, m);
23   addto(c + 0, w, m + m);                 // add a_l b_l
24   subfrom(c + m, w, m + m);               // subtract a_l b_l B^m
25   return;
26 }
```

3 Allocation for the result array

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.

$$cspace(n) = \begin{cases} 2n & \text{if } n \le lim \\ \lfloor \frac{n+1}{2} \rfloor + cspace(\lfloor \frac{n+1}{2} \rfloor + 1) & \text{otherwise} \end{cases} \tag{1}$$

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

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

In the following subsection we prove a sharp upper bound for $cspace(n)$. If $la \le lim$, 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:*

$$cspace(n) \le 3(\lfloor \frac{n+1}{2} \rfloor + 1), \text{ for } lim \ge 4,\, n > lim \tag{3}$$

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

Assuming

$$\begin{aligned} cspace(\lfloor \frac{n+1}{2} \rfloor + 1) &\le 3(\lfloor \frac{\lfloor \frac{n+1}{2} \rfloor + 1 + 1}{2} \rfloor + 1) \qquad (hyp1) \\ &= 6 + 3(\lfloor \frac{n+1}{4} \rfloor), \end{aligned}$$

we need to show:

$$\begin{aligned} cspace(n) &= \lfloor \frac{n+1}{2} \rfloor + cspace(\lfloor \frac{n+1}{2} \rfloor + 1) && (4) \\ &\le \lfloor \frac{n+1}{2} \rfloor + 6 + 3(\lfloor \frac{n+1}{4} \rfloor) && (5) \\ &\le 3(\lfloor \frac{n+1}{2} \rfloor + 1) && (6) \end{aligned}$$

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

$$cspace(n) = \lfloor \frac{n+1}{2} \rfloor + 2(\lfloor \frac{n+1}{2} \rfloor + 1) \quad (7)$$

$$\leq 3(\lfloor \frac{n+1}{2} \rfloor + 1) \quad (8)$$

This is clearly true.

□

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

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

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

References

[1] M. Kaufmann and J. S. Moore. ACL2 Theorem Prover. `http://www.cs.utexas.edu/~moore/acl2/`.

[2] R. E. Maeder. *Storage allocation for the Karatsuba integer multiplication algorithm*, pages 59–65. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1993. http://www.springerlink.com/content/w15058mj6v59t565/.

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-guards cspace)


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

;; Rewriting floor in terms of mod frequently helps.
(defthmd 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 (* 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.
(defthmd upper-bound-n<=8
  (implies (and (<= 4 lim) (< lim n) (<= n 8)
                (natp lim) (natp n))
           (<= (cspace n lim)
               (* 3 (+ 1 (floor (+ n 1) 2)))))
  :hints (("Subgoal *1/5’’" :cases ((= n 5) (= n 6) (= n 7) (= n 8)))))


;; The upper bound for cspace.
(defthmd upper-bound
  (implies (and (<= 4 lim) (< lim n)
                (natp lim) (natp n))
           (<= (cspace n lim)
               (* 3 (+ 1 (floor (+ n 1) 2)))))
  :hints (("Goal" :induct t)
          ("Subgoal *1/2.1’" :use (upper-bound-n<=8))))
```