A Modified Algorithm for Taking Reciprocal of n-bit Integers

By Hiroshi TANIGUCHI, Itsuo TAKANAMI and Katsushi INOUE

(Received July 16, 1979)

Abstract

For an integer P whose bit-length is just n, its reciprocal is defined by $[2^{2n-1}/P]$, where $[x]$ denotes the greatest integer equal to or less than x. It is well known[1] that the time for taking reciprocal is, to within a constant factor, the same as the time to do integer multiplication in bit operation. So we first explain Cook's Algorithm which requires the same order of time as multiplication. Next, we propose a modified algorithm and prove its correctness and analyse the complexity of computation. This shows that the algorithm is expected to improve constant factor in complexity.

1. Introduction

An algorithm shown by Schönhage-Strassen[2] is asymptotically the fastest known way to multiply two integers together. The algorithm requires time $O_B(n \log n \log\log n)$[+], where n is a bit length of the integers. On the other hand, it is well known that certain integer operations, such as division, squaring and taking reciprocals, require the same order of time as multiplication.

Specially for taking reciprocal operation, there is the Cook's algorithm using divide-and-conquer approach which requires the same order of time as multiplication. We first explain the algorithm which is defined only for integers whose bit-length is a power of 2. If the bit-length is not a power of 2, the generalization should be obvious — add 0's and change scale. For both cases, we also discuss the upper bound of executing time. Finally we propose a new algorithm, prove its correctness and compute the upper bound of its executing time. The new algorithm is considered to improve constant factor asymptotically.

2. Preliminaries and Cook's algorithm

In this section, we first give a notation of integers and a definition of reciprocals of integers. Next we present Cook's algorithm whose executing time is $O(M(n))$, where $M(n)$ is the time to multiply two integers of size n. We also discuss the upper bound of its executing time more rigorously.

Department of Electronics Faculty of Engineering Yamaguchi University Ube, 755 Japan

+ A function $g(n)$ is said to be $O(f(n))$ (read order $f(n)$) if there exists a constant c such that $g(n) \leqq cf(n)$ for all but some finite (possibly empty) set of nonnegative values for n. We use O_B to indicate order of magnitude under the bitwise computation. In what follows, we omit the suffix B, if no confusion occurs.

DEFINITION 1: Let $P=[p_1p_2\cdots p_n]$ be an n-bit integer with $p_1=1$, where $[x]$ is the integer denoted by the bit string x (e.g., $[110]=6$). Its reciprocal integer Q is an integer such that

$$Q=\lfloor 2^{2n-1}/P\rfloor . \tag{2.1}$$

That is,

$$2^{2n-1}=QP+S \qquad (0\leqq S<P), \tag{2.2}$$

where S is a unique integer depending on 2^{2n-1} and P.

Now, we present Cook's algolithm. For correctness of this algorithm, see [1].

ALGORITHM C: A recursive algorithm for taking integer reciprocals.
Input. An n-bit integer $P=[p_1p_2\cdots p_n]$, with $p_1=1$. For convenience, we assume that n is a power of 2.
Output. The integer $A=[a_0a_1\cdots a_n]$ such that $A=\lfloor 2^{2n-1}/P\rfloor$.
Method. We call RECIPROCAL($[p_1p_2\cdots p_n]$), where RECIPROCAL is the recursive procedure in Fig. 1. For any k which is a power of 2, it computes an approximation to $\lfloor 2^{2k-1}/[p_1p_2\cdots p_k]\rfloor$. Note that the result is normally a k-bit integer except when P is a power of 2, in which case the result is a $(k+1)$-bit integer.

```
                    procedure RECIPROCAL([p1p2⋯pk]):
    begin
1   if k=1 then return [10]
    else
        begin
2           [c0c1⋯ck/2]←RECIPROCAL([p1p2⋯pk/2]);
3           [d1d2⋯d2k]←[c0c1⋯ck/2]*2^(3k/2)−[c0c1⋯ck/2]^2
                        [p1p2⋯pk];
            comment Although the right-hand side of line 3 appears to
              produce a (2k+1)-bit number, the leading [(2k+1)st] bit is
              always zero;
4           [a0a1⋯ak]←[d1d2⋯dk+1];
            comment [a0a1⋯ak] is a good approximation to 2^(2k−1)/[p1p2⋯
              pk]. The following loop improves the approximation by adding
              to the last three places if necessary;
5           for i←2 step −1 until 0 do
6               if ([a0a1⋯ak]+2^i)*[p1p2⋯pk]≦2^(2k−1) then
7                   [a0a1⋯ak]←[a0a1⋯ak]+2^i
8           return [a0a1⋯ak]
        end
    end
```

Fig. 1. Procedure RECIPROCAL.

We give some assumptions and notations before we discuss the executing time.

(i) $M(n)$ denotes the time to multiply two integers of size n. $M(n)$ satisfies the following condition: for $a \geqq 1$

$$a^2 M(n) \geqq M(an) \geqq aM(n).$$

(ii) $A(n)$ ($S(n)$) denotes the time to add (substract) two integers of size n. $A(n)$ satisfies the following condition: for some constant c, $A(n) \leqq cn$.

(iii) A shift operation, a multiplication or a division by some powers of 2, does not require any time on bit operations.

On the upper bound of its executing time, we get the following theorem.

THEOREM 1: Let $T_R(n)$ denotes the time for executing procedure RECIPROCAL, where n is a bit length of a given integer. Then we have the following inequality.

$$T_R(n) \leqq 5M(n) + c_0 n - 5M(n)/n + c_1 \tag{2.3}$$

where c_0, c_1 are some constants not depending on n.

Proof. Line 2 requires $T_R(k/2)$ bit operations. Line 3 consists of squaring, multiplication and subtraction, requiring $M(k/2+1)$, $M(k+2)$ and $O(k)$ times, respectively. (Recall that a multiplication by some powers of 2 does not any time.)

By our assumption on M, $M(k/2+1) \leqq M(k+2)/2$. Furthermore, $M(k+2) - M(k)$ is $O(k)$ and thus line 3 is bounded by $3M(k)/2 + c'k$ for some constant c' not depending on n. Line 4 is clearly $O(k)$.

It appears that the loop of lines 5–7 requires three multiplication, but the calculation can be done by one multiplication, $[a_0 a_1 \cdots a_k] * [p_1 p_2 \cdots p_k]$, and some additions and subtractions of most $2k$-bit integers. Thus lines 5–7 are bounded by $M(k) + c''k$ for some constant c''.

Putting all costs together, we have

$$T_R(n) \leqq T_R(n/2) + \frac{5}{2} M(n) + c'''n \tag{2.4}$$

for some constant c'''.

To solve the above recurrence, put $n = 2^l$, $T_R(2^i) = r_i$, and $M(2^i) = m_i$. we have

$$r_i \leqq r_{i-1} + \frac{5}{2} m_i + c'''2^i \qquad \text{for} \quad 1 \leqq i \leqq l \tag{2.5}$$

By summing up (2.5) from $i=1$ to l, we have

$$r_l \leqq r_0 + \frac{5}{2} \sum_{i=1}^{l} m_i + c''' \sum_{i=1}^{l} 2^i \tag{2.6}$$

By our assumption on M, $m_i \leqq \frac{2^i}{2^l} m_l$ for $1 \leqq i \leqq l$.

Then we have

$$r_l \leqq r_0 + \left(\frac{5m_l}{2^{l+1}} + c'''\right)\sum_{i=1}^{l} 2^i$$

$$= 5m_l + 2c'''2^l - \frac{5m_l}{2^l} + r_0 - 2c'''$$

Therefore, we have

$$T_R(n) \leqq 5M(n) + c_0 n - \frac{5M(n)}{n} + c_1$$

where c_0 and c_1 are some constants such that $c_0 \geq 2c'''$, $c_1 \geq T_R(1) - 2c'''$. Q. E. D.

3. Generalizalization of Cook's algorithm

In the previous algorithm, it was assumed that a bit-length n was a power of 2. If not, the generalization should be obvious by adding extra 0's and changing scale. That is, assume that P is an n-bit integer and $n = l + m$ (l is a power of 2 and $0 < m < l$).

Now consider P' whose length is just $2l$ by adding $l-m$ 0's to P. It is clear that $P' = P2^{l-m}$. Then the previous algorithm can be applied to P' since the size of P' is a power of 2.

Let Q' be the reciprocal of P'. We have

$$Q = \lfloor Q'/2^{l-m} \rfloor . \tag{3.1}$$

Below, we modify the definition on M to discuss the difference between the executing times of the previous case and this case.

DEFINITION 2. Let $M'(n, m)$ be the time to multiply an n-bit integer and an m-bit integers. $M'(n, m)$ satisfies the following condition for each n, m.

(i) $M'(n, m) = M'(m, n)$. (3.2)

(ii) Let $n_L = \mathrm{Max}(n, m)$ and $n_S = \mathrm{Min}(n, m)$. Then,

$$M(n_S) \leqq M'(n, m) \leqq M(n_L) . \tag{3.3}$$

Let f be a function such that

$$f: \mathbf{N}^2 \longrightarrow \mathbf{R}$$

and $f(n, m) = M'(n, m+1) - M'(n, m)$ for each n, m in $\mathbf{N}$, where $\mathbf{N}$ is the set of non-negative integers and $\mathbf{R}$ the set of real numbers. We can assume that f satisfies the following conditions.

(i) If $n \geqq l$, then for each m

$$f(n, m) \geqq f(l, m) \tag{3.4}$$

(ii) If $m \geqq l$, then for each n

$$f(n, m) \geqq f(n, l) \tag{3.5}$$

Let $T'_R(n)$ denote the time for executing procefure RECIPROCAL, where n is not a power of 2. It is clear that the following inequality holds.

$$T'_R(n) \leqq T_R(2l). \tag{3.6}$$

On the other hand, it is natural that the upper bound of $T'_R(n)$ should be really less than $T_R(2l)$, since the shift operations do not require any time. This version is shown by the following corollary whose proof is similar to that of Theorem 1.

COROLLARY 1: Let $n=l+m$, l be a power of 2 and $0<m<l$. We have

$$T'_R(n) \leqq T_R(l)+M(l)+2M'(2l, n)+c_2 l \tag{3.7}$$

where c_2 is some constant not depending on n.

4. A modified algorithm

In section 2, we assert that Q and S are the quotient and the remainder, respectively, when 2^{2n-1} is divided by P. In Cook's algorithm, only the quotient Q was used recurrently. Now if we construct a modified algorithm which use not only the quotient Q but also the remainder S, we can expect that the algorithm should be faster than Cook's one. We also require the modified algorithm compute the reciprocals of n-bit integer without adding extra 0's, even if n is not a power of 2. So, propose the following algorithm.

ALGORITHM **M**: A modified algorithm for taking reciprocals.
Input. An n-bit integer $P=[p_1 p_2 \cdots p_n]$, with $p_1=1$.
Output. The integer $Q=[q_0 q_1 \cdots q_n]$ and $S=[s_1 s_2 \cdots s_n]$ which are the quotient and the remainder respectively, when 2^{2n-1} is divided by P. Note that S is an at most n-bit integer.
Method. We call the procedure MODIFIED$([p_1 p_2 \cdots p_n])$ which is the recursive procedure in Fig. 2.

---

procedure MODIFIED$([p_1 p_2 \cdots p_k])$:

```
    begin
1   if k=1 then return ([10], [0])
    comment The first and second component is the quotient and the remainder,
      respectively.
    else
        begin
2           l←⌋k⌊;  m←k−l;
            comment ⌋1⌊=1 and for i>1, ⌋i⌊ is the greatest power of 2 less
              than i.  Note ⌋2^m⌊=2^(m−1) for m≧1.
```

```
3        ([c0c1⋯cl], [t1t2⋯tl])←MODIFIED([p1p2⋯pl]);
4        [r1r2⋯rm]←[pl+1pl+2⋯pk];
5        [bsb1⋯bk]←[t1t2⋯tl]*2^m−[r1r2⋯rm]*[c0c1⋯cl]
         comment bs is a sign bit: bs=0 if nonnegative.
6        [dsd1⋯dk+l]←[bsb1⋯bk]*[c0c1⋯cl];
7        [e0e1⋯ek+2l−1]←[c0c1⋯cl]*2^(m+2l−1)+[dsd1⋯dk+l];
8        [q0q1⋯qk]←[e0e1⋯ek];
9        for i←2 step −1 until 0 do
           begin
10           [g0g1⋯g2k]←([q0q1⋯qk]+2^i)*[p1p2⋯pk];
11           if [g0g1⋯g2k]≦2^(2k−1) then
               begin
12                 [q0q1⋯qk]←[q0q1⋯qk]+2^i;
13                 [s1s2⋯sk]←2^(2k−1)−[g0g1⋯g2k];
               end
           end
14       return ([q0q1⋯qk], [s1s2⋯sk])
       end
   end
```

Fig. 2. Procedure MODIFIED ($[p_1p_2\cdots p_k]$).

THEOREM 2: The algorithm **M** finds $[q_1q_2\cdots q_k]$ and $[s_1s_2\cdots s_k]$ such that

$$[q_0q_1\cdots q_k]*[p_1p_2\cdots p_k]=2^{2k-1}-[s_1s_2\cdots s_k]$$

and $0\leqq[s_1s_2\cdots s_k]<[p_1p_2\cdots p_k]$, for a given integer $[p_1p_2\cdots p_k]$.

Proof. The proof is by induction on k. The basis, $k=1$, is trivial by line 1. For the inductive step, let C be $[c_0c_1\cdots c_l]$ obtained at line 3, $P'=[p_1p_2\cdots p_l]$, $R=[p_{l+1}p_{l+2}\cdots p_k]$, and T be $[t_1t_2\cdots t_l]$ obtained at line 3.

Then $P=[p_1p_2\cdots p_k]=p'2^m+R$. By the induction hypothesis we have

$$CP'+T=2^{2l-1} \tag{3.8}$$

where $0\leqq T<P'$. Let $B=[b_sb_1\cdots b_k]$, $D=[d_sd_1\cdots d_{k+l}]$, and $E=[e_0e_1\cdots e_{k+2l-1}]$. Furthermore, let Q be $[q_0q_1\cdots q_k]$ obtained at line 8.

Since $p_1=1$, $2^{l-1}\leqq p'<2^l$, $0\leqq R<2^m$, and thus $2^{l-1}\leqq C\leqq 2^l$, $0\leqq T<P'<2^l$. It follows that $-2^k<B<2^k$ and hence the $(k+1)$-bits including a sign bit are sufficient to represent B. Similarly $(k+l+1)$-bits including a sign bit and $(k+2l)$-bits are sufficient to represent D and E, respectively, since $-2^{k+l}<D<2^{k+l}$ and $E<2^{k+2l-1}+2^{k+l}$.

Consider the product $$PQ=P\left(C2^m+\left\lfloor\frac{BC}{2^{2l-1}}\right\rfloor\right). \tag{3.9}$$

Now let $Q'=C2^m+BC/2^{2l-1}$ (i.e., $Q=\lfloor Q'\rfloor$).

From (3.8) we have

$$CP'2^{2m}+T2^m=2^{2k-1}.$$

Thus,

$$PC2^m=2^{2k-1}-2^m(T2^m-RC)=2^{2k-1}-2^mB. \tag{3.10}$$

Therefore,

$$\begin{aligned} PQ' &= 2^{2k-1}-B\left(2^m-\frac{PC}{2^{2l-1}}\right) \\ &= 2^{2k-1}-B\left\{2^m-\frac{(P'2^m+R)C}{2^{2l-1}}\right\}. \end{aligned} \tag{3.11}$$

From (3.8) $P'C=2^{2l-1}-T$, and by substituting $2^{2l-1}-T$ for $P'C$ and performing some simplifications, we have

$$PQ'=2^{2k-1}-B^2/2^{2l-1}=2^{2k-1}-U \tag{3.12}$$

where $0\leqq U=B^2/2^{2l-1}<2^{2m+1}$, since $-2^k<B<2^k$ and thus $0\leq B^2<2^{2k}=2^{2(l+m)}$. Now

$$PQ=P\lfloor Q'\rfloor>P(Q'-1),$$

so from (3.12) we have,

$$2^{2k-1}\geqq PQ'\geqq PQ>PQ'-P=2^{2k-1}-U-P>2^{2k-1}-2^{m+1}-2^k.$$

Thus

$$PQ=2^{2k-1}-S'$$

where $0\leqq S'<2^{2m+1}+2^k$. Since $P\geqq 2^{k-1}$, it follows that by adding at most 5* to Q, we obtain the quotient which satisfies the statement of theorem for k. This job is done by lines 9–13. Once the quotient was determined, it follows that the remainder is also correctly computed at line 13. So the induction step follows. Q. E. D.

Finally, we discuss the time of executing this algorithm. We first need the following lemma.

LEMMA 1: For each nonnegative integer j, we have

$$2M'(2j, j)\leqq M(2j)+M(j).$$

Proof. Put $a(j)=M(2j)-M'(2j, j)$ and $b(j)=M'(2j, j)-M(j)$. Then, we have

$$\begin{aligned} a(j) &= \sum_{i=0}^{j-1}\{M'(2j, j+i+1)-M'(2j, j+i)\} \\ &= \sum_{i=0}^{j-1} f(2j, j+i) \qquad \text{(See definition 2)}. \end{aligned}$$

* If k is a power of 2, then at most 5, otherwise at most 3.

Similarly, we have $b(j)=\sum_{i=0}^{j-1} f(j+i, j)$.
By using (3.4) and (3.5), we have

$$\begin{aligned} a(j)-b(j) &= \sum_{i=0}^{j-1}\{f(2j, j+i)-f(j+i, j)\} \\ &\geqq \sum_{i=0}^{j-1}\{f(j+i, j+i)-f(j+i, j)\} \\ &\geqq \sum_{i=0}^{j-1}\{f(j+i, j)-f(j+i, j)\} \\ &=0. \end{aligned}$$

Therefore, $2M'(2j, j)\leqq M(2j)+M(j)$. Q.E.D.

THOREM 3: Let $T_M(n)$ denote the time of executing the modified algorithm, where n is a bit length of integer P. We have the following inequality.

$$T_M(n)\leqq T_M(l)+M'(l, m)+M'(n, l)+M(n)+c_3 n \tag{3.13}$$

where $l=\rfloor n\lfloor$, $m=n-l=n-\rfloor n\lfloor$ and c_3 is some constant.
If n is a power of 2, the solution of (3.13) is

$$T_M(n)\leqq \frac{9}{2} M(n)+c_4 n-\frac{9}{2}\frac{M(n)}{n}+c_5 \tag{3.14}$$

where c_4 and c_5 are some constants.

Proof. The proof is similar to that of theorem 1. First we show (3.13). The times required at lines 3, 5, 6, 7, 9–13, and 14 are $T_M(l)$, $M'(l, m)+O(n)$, $M(n, l)$, $O(n)$, $M'(n)+O(n)$, and $O(n)$, respectively. So we have (3.13).

Let n be a power of 2, $n=2^k$ for some integer k, then $l=\rfloor n\lfloor=n/2$ and $m=n/2$. Furthermore put $T_M(2^i)=t_i$, $M(2^i)=m_i$, and $M'(2^i, 2^{i-1})=u_i$, then we have

$$t_i\leqq t_{i-1}+m_{i-1}+u_i+m_1+c_3 2^i, \tag{3.15}$$

for i such that $1\leqq i\leqq k$.
By summing up (3.15) from $i=1$ to k, we have

$$t_k\leqq t_0+\sum_{i=1}^{k}(u_i+m_i+m_{i-1}+c_3 2^i).$$

From Lemma 1, we have $u_i\leqq\frac{1}{2}(m_i+m_{i-1})$ for each i such that $1\leqq i\leqq k$. By our assumption on M and performing some algebraic simplifications, we can readily have the inequality (3.14). Q.E.D.

5. Conclusion

We have investigated some algorithms for taking reciprocals and have proposed a modified algorithm expected faster than the previous one asymptotically.

Acknowledgement

H. Taniguchi, one of the authors, would like to thank Professors H. Anzai of Kyushu Institute of Technology and S. Yoshida of Kyushu University for their hearty encouragement.

References

[1] Cook. S. A. and S. O. Aanderaa, "On the minimum complexcity of functions", Trans. Amer. Math. Soc., **142**, 291–314 (1969).

[2] Schönhage. A. and V. Strassen, "Schnell Multiplikation grosser Zahlen", Computing, **7**, 281–292 (1971)

[3] A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The design and analysis of computer algorithms", (Addison-Wesley Pub. Co., 1974)

[4] H. Taniguchi, I. Takanami, and K. Inoue, "Some New Algorithms for Taking Reciprocal of n bit Integer", Technical Report No. AL79–14, IECE of Japan (1979) (in Japanese).