Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Efficient multiple-precision integer division algorithm

Debapriyay Mukhopadhyay [a], Subhas C. Nandy [b,*]

[a] IXIA Technologies Pvt. Ltd., Kolkata 700091, India
[b] Indian Statistical Institute, Kolkata 700108, India

ARTICLE INFO

Article history:
Received 16 February 2013
Received in revised form 3 October 2013
Accepted 19 October 2013
Available online 25 October 2013
Communicated by Jinhui Xu

Keywords:
Division algorithm
Normalization
Computational arithmetic
Cryptography

ABSTRACT

Design and implementation of division algorithm is one of the most complicated problems in multi-precision arithmetic. Huang et al. [1] proposed an efficient multi-precision integer division algorithm, and experimentally showed that it is about three times faster than the most popular algorithms proposed by Knuth [2] and Smith [3]. This paper reports a bug in the algorithm of Huang et al. [1], and suggests the necessary corrections. The theoretical correctness proof of the proposed algorithm is also given. The resulting algorithm remains as fast as that of [1].

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Arithmetic operations on large integers are often used in cryptographic algorithms. The usual arithmetic operations are performed in the machine using the built-in functions. Each machine has a base B in its number system, and can store unsigned integers of values $\{0, 1, 2, \ldots, B-1\}$ in built-in integer locations for that machine. The time complexity of a single usual arithmetic operation is assumed to be $O(1)$.

A large integer cannot be stored in machine dependent built-in size for integers, and arithmetic operations on such integer(s) cannot be performed using the built-in routines for those arithmetic operations. The operations on large integers are called multi-precision arithmetic operations. The multi-precision division is the hardest among all the four multi-precision arithmetic operations. Multi-precision division plays a crucial role in cryptographic research [4], and primality testing [5]. The commonly used multi-precision division algorithm is proposed by Knuth [2].

Normalization is one of the key steps of multi-precision division, and it is defined as the act of restoring the individual digits or words in the range $[0, B-1]$. Since each word or digit of the quotient is guessed in each step of the division, so it is difficult to skip normalization. The division algorithm proposed by Smith [3] reduces the intermediate normalization steps. Huang et al. [1] proposed an efficient algorithm for multi-precision integer division that reduces the number of normalizations to a single normalization step. The uniqueness of the algorithm is that, if it is applied for long integer division, then both the quotient and remainder simultaneously gets calculated at the end. There is no need for any correction step or any extra multiplication or subtraction for computing the remainder. It is experimentally shown that the algorithm in [1] is three times faster than the algorithm by Knuth [2].

We have identified a bug in the algorithm by Huang et al. [1], and propose the necessary corrections. We theoretically justify the correctness of our algorithm. The detailed experiment justifies that our corrected version of the algorithm runs with the same efficiency as suggested in [1].

The paper is organized as follows. Section 2 briefly describes the algorithm of Huang et al. [1]. In Section 3, we report the bug by providing an example. We describe the corrected algorithm in Section 4, and also provide the correctness proof. Finally, the concluding remarks appear in Section 5.

* Corresponding author.
E-mail addresses: debapriyaym@gmail.com (D. Mukhopadhyay), nandysc@isical.ac.in (S.C. Nandy).

0020-0190/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ipl.2013.10.005

2. Overview of the algorithm of [1]

Let B be the base of the multi-precision integers under consideration. For two multi-precision positive integers a and b ($a > b > 0$), we need to find out multi-precision integers q and r such that $a = bq + r$. Let m and n denote the number of words required to store a and b respectively. Thus $a = a_1 + a_2.B + \cdots + a_m.B^{m-1}$ and $b = b_1 + b_2.B + \cdots + b_n.B^{n-1}$. Algorithm starts by copying a into a work array W of size $m+1$, whose each element can hold integers that require no more than 4 bytes, i.e., it ranges from $-2\,147\,483\,648$ to $2\,147\,483\,647$ in decimal number system. We use a work array W for the division, and use a variable MAX that contains the integer 2 147 483 648. A zero digit is put at $W[0]$; then the digits of a are stored in successive locations starting from the most significant digit a_m at $W[1]$, followed by a_{m-1} at $W[2]$ and so on. In this way, a_1 will be stored at $W[m]$. Similarly, the digits $b_1, b_2, \ldots, b_n$ are stored in $b[n-1], b[n-2], \ldots, b[0]$ of the array b. The work array W is updated during the iteration process and there are chances of overflow in some elements of W during the execution. So, normalization is required to restore the digits in the range $[0, B-1]$. At the end of the algorithm the least significant n elements of W gives the remainder r and the most significant $m-n+1$ elements correspond to the quotient q. Algorithm 1 gives an intuitive description of the algorithm of [1] assuming that the working register and the arithmetic circuit for division of the computer can handle the three digited numbers with base B. For detailed description of this algorithm, see [1].

Algorithm 1. Multi-precision_Division.

```
1.  (* Form the denominator of division using first two consecutive
        elements of $b$ *)
    Compute $D = b[0] * B + b[1]$
2.  for $i = 0, 1, 2, \ldots, m-n$ do (* $i$ indicates the iteration number. *)
3.      (* Form the numerator using three consecutive elements
            $W[i]$, $W[i+1]$ and $W[i+2]$ *)
        Compute $N = W[i] * B^2 + W[i+1] * B + W[i+2]$
            (* Assume that $N$ can store a number less than $B^4$. *)
4.      Compute the quotient $Q = \lfloor \frac{N}{D} \rfloor$
5.      (* Update the array $W$ *)
        for $j = 1$ to $n$ do
            $W[i+j] = W[i+j] - Q.b[j-1]$
        endfor
6.      $W[i+1] = W[i] * B + W[i+1]$
7.      $W[i] = Q$ (* Put $Q$ in $W[i]$ *)
8.  endfor
9.  Normalize W
10. Report the quotient and remainder from the array $W$.
```

Now, we describe the normalization procedure as proposed in [1]. Here the objective is to consider each word of the quotient and remainder, and restore its value in the range $[0, B-1]$. The main idea of the normalization procedure centers around finding a multiplicative factor c such that when $c * B$ is added with the content of an element in the array W, it will lie in the range $[0, B-1]$. For a negative word, the algorithm finds a positive c, and for a word greater than B, the algorithm finds a negative c, to appropriately adjust the word. If a word $W[i]$ gets adjusted, then its immediate next higher word $W[i-1]$ needs to be adjusted so that the value remains unchanged. Next, $W[i-1]$ is considered for normalization. The normalization procedure described in [1] is stated below.

Algorithm 2. Normalize (X).

```
1. for $i = m, m-1, m-2, \ldots, 1$ do (* $i$ indicates the iteration number *)
2.     $c = 0$
3.     if $(X[i] < 0)$ then $c = \lfloor(-X[i]-1)/B\rfloor + 1$
4.     else if $(X[i] \geqslant B)$ then $c = -\lfloor X[i]/B \rfloor$
5.     end if
6.     $X[i] = X[i] + c * B$
7.     $X[i-1] = X[i-1] - c$
8. endfor
```

We now describe two interesting properties of the normalization procedure of [1]. Let X be an array containing a multi-precision number obtained by the **Multi-precision_Division** algorithm prior to the normalization (i.e., the array W after Step 8). Thus, $X = (X[l-1] + X[l-2].B + \cdots + X[0].B^{l-1})$, where l is the number of words in X. We now define

$$Val_{l-i} = \sum_{j=1}^{i} X[l-j].B^{j-1}, \quad \text{for } i = 1, 2, \ldots, l. \tag{1}$$

Observe that, $Val_0 = X$. It needs to be mentioned that Val_i's, for $i = 0, 1, 2, \ldots, l-1$, need not be computed/stored in the algorithm. This only helps in characterizing which indices of the array X requires normalization.

Property 1. *For each $i = l-1, l-2, \ldots, 1$, if either $Val_i < 0$ or $Val_i \geqslant B^i$, then the normalization is required for $X[i]$. The normalization factor c is positive if $Val_i < 0$ and negative if $Val_i \geqslant B^i$.*

Property 2. *For any value of $i \in \{1, 2, \ldots, l-1\}$, if the normalization is not required for $X[i]$, then Val_i remains unaltered after the normalization. Since normalization algorithm described above doesn't normalize the location $X[0]$, $Val_0 (= X)$ remains same before and after the normalization.*

Let us demonstrate the algorithm with a small example in Table 1. Here $B = 10$, $a = 60\,541$, $b = 432$; thus $m = 5$ and $n = 3$. We show the iterations of the *for* loop of Step 2 of the algorithm **Multi-precision_Division** in Table 1 and also show the outcome of the normalization step of the algorithm of Huang et al. [1]. Note that, D remains equal to 43 throughout the execution.

After the normalization step, the higher order $m-n+1$ words form the quotient and the remaining n words form the remainder. Therefore, for the above example quotient $q = 140$ and remainder $r = 61$. But the above normalization procedure may lead to an incorrect result as described in the next section.

3. Description of the bug

In [1], the correctness of the algorithm (i.e., whether $a = b.q + r$ holds) is not established. We could identify a pathological instance to show that the algorithm of

Download English Version:

https://daneshyari.com/en/article/427424

Download Persian Version:

https://daneshyari.com/article/427424

Daneshyari.com