Theory of Computing Systems (2020) 64:1331–1337
https://doi.org/10.1007/s00224-020-09986-5

Multiplication Algorithm Based on Collatz Function

David Barina[1]

Published online: 15 May 2020

Abstract

This article presents a new multiplication algorithm based on the Collatz function. Assuming the validity of the Collatz conjecture, the time complexity of multiplying two n-digit numbers is $O(kn)$, where the k is the number of odd steps in the Collatz trajectory of the first multiplicand. Most likely, the algorithm is only of theoretical interest.

Keywords Multiplication algorithm · Division algorithm · Computer arithmetic · Collatz conjecture

1 Introduction

One of the most famous problems in number theory that remains unsolved is the Collatz conjecture, which asserts that, for arbitrary positive integer x, a sequence defined by repeatedly applying the function

$$C(x) = \begin{cases} 3x + 1 & \text{if } x \text{is odd, or} \\ x/2 & \text{if } x \text{is even} \end{cases} \tag{1}$$

will always converge to the cycle passing through the number 1. The terms of such sequence typically rise and fall repeatedly, oscillate wildly, and grow at a dizzying pace. The conjecture has never been proven. There are however experimental evidence [1] and heuristic arguments [2–4] that support it. There is also an extensive literature, [5, 6], on this question.

Quoting Chamberland [7], some authors work with the Collatz function, $\mathbf{Z}^+_{\text{odd}} \to \mathbf{Z}^+_{\text{odd}}$, defined by

$$F(x) = (3x + 1) \,/\, 2^{\text{ctz}(3x+1)}, \tag{2}$$

✉ David Barina
ibarina@fit.vutbr.cz

[1] Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations, Brno, Czech Republic

where $\mathrm{ctz}(x)$ equals the number of factors of 2 contained in x. In computer science, the $\mathrm{ctz}(x)$ is also known as the count trailing zeros operation, and, considering binary number systems, can be computed in $O(n)$. Please note that the n denotes the number of digits (here specifically bits) of the x. Considering this definition, the trajectory $x, F(x), F^2(x), F^3(x), \dots, 1$ can be computed using Algorithm 1.

Algorithm 1 Trajectory of the Collatz function.

Require: x is positive integer
Ensure: returns 1

```
function F-TRAJECTORY($x$)
    OUTPUT(x)
    while $x \neq 1$ do
        $x \leftarrow 3x + 1$
        $x \leftarrow x \,/\, 2^{\mathrm{ctz}(x)}$
        OUTPUT(x)
    end while
    return $x$
end function
```

The algorithm starts with any positive odd integer. Every iteration of the while-loop replaces the x with $F(x)$, and therefore effectivelly computes odd integer as defined by F^i (ith function iterate). Let us denote the total number of iterations k. Provided that the Collatz conjecture holds, the above algorithm will always halt (converge to and return the number 1), for any initial x, after exactly k steps. Note that although the F is defined only on positive odd integers, Algorithm 1 and all subsequent algorithms in this paper also work on positive even integers. In this case, however, the very first iteration is different from that one defined by the Collatz function.

2 Towards Multiplication Algorithm

Note that multiplying the x by positive odd integer a does not affect the result of the ctz operation. By multiplying the Collatz function by an odd integer a, and tracking $m = ax$ rather than x, we get

$$G_a(m) = (3m + a) \,/\, 2^{\mathrm{ctz}(3m+a)}, \tag{3}$$

where each iterate $G_a^i(m) = a\, F^i(x)$. Analogously, we get Algorithm 2.

Algorithm 2 Trajectory of the $G_a(m)$.

```
Require: $m$ is positive and $a$ is positive odd integer such that $a \mid m$
Ensure: returns $a$

function G-TRAJECTORY($m, a$)
    OUTPUT(m)
    while $m \neq a$ do
        $m \leftarrow 3m + a$
        $m \leftarrow m \,/\, 2^{\mathrm{ctz}(m)}$
        OUTPUT(m)
    end while
    return $m$
end function
```

Still, the algorithm will always halt for any initial m and a such that $a \mid m$, provided that the Collatz conjecture holds. In other words, the G-trajectory of any integer $m = ax$ will pass through the a.

The same idea can be used to construct a division Algorithm 3. The idea behind this is to exploit the above algorithm to pull out the a from $m = ax$, and then reassemble the x. Or in other words, we will divide the dividend m by the divisor a, and get the quotient x. The only question is how to reassemble the x.

Algorithm 3 Division algorithm.

```
Require: $m$ is positive and $a$ is positive odd integer such that $a \mid m$
Ensure: returns $x$ such that $m = ax$

function DIV($m, a$)
    $i \leftarrow 0$
    while $m \neq a$ do
        $m \leftarrow 3m + a$
        $\beta_i \leftarrow \mathrm{ctz}(m)$
        $m \leftarrow m/2^{\beta_i}$
        $i \leftarrow i + 1$
    end while
    $x \leftarrow 1$
    while $i \neq 0$ do
        $i \leftarrow i - 1$
        $x \leftarrow x \times 2^{\beta_i}$
        $x \leftarrow (x - 1)/3$
    end while
    return $x$
end function
```

To do that, we need to store all decisions β_i from all iterations of the algorithm, and then use them in reverse order (run the algorithm backward). The only difference is that we start with 1 (multiplicative identity) rather than a, and therefore get x rather than ax at the end.

The algorithm basically computes the sequence $m, G_a(m), G_a^2(m), \ldots, a$ which is equal to $a\,x, a\,F(x), a\,F^2(x), \ldots, a$. When it does, the algorithm remembers all decisions $\text{ctz}(3m + a) = \text{ctz}(3x + 1)$. In the next step, the a is replaced with 1, and the algorithm uses an inverse of (2), namely

$$x = \frac{F(x)\,2^{\text{ctz}(3x+1)} - 1}{3}, \tag{4}$$

to recover the sequence $1, \ldots, F^2(x), F^1(x), x$ (backward) and therefore gets the x at the end.

Finally, we can use the above algorithm in the opposite manner. Instead of dividing m by x, and then multiplying by 1, we divide x by 1, and multiply by a to get the $m = ax$. Note that in this way, there is no restriction on a parity of the a. The pseudocode is given in Algorithm 4.

Algorithm 4 Multiplication algorithm.

Require: x and a are positive integers
Ensure: returns m such that $m = ax$

```
function MUL($x, a$)
    $i \leftarrow 0$
    while $x \neq 1$ do
        $x \leftarrow 3x + 1$
        $\beta_i \leftarrow \text{ctz}(x)$
        $x \leftarrow x/2^{\beta_i}$
        $i \leftarrow i + 1$
    end while
    $m \leftarrow a$
    while $i \neq 0$ do
        $i \leftarrow i - 1$
        $m \leftarrow m \times 2^{\beta_i}$
        $m \leftarrow (m - a)/3$
    end while
    return $m$
end function
```

Addition, subtraction, multiplication and division by two (suppose representation in a binary system) are obviously in $O(n)$. It may not be so obvious, but multiplication and division by three are also in $O(n)$. The algorithm essentially converts multiplication to a series of k subtractions (see $m - a$ instead of $x \times a$). The subtractions are performed on numbers that occur in the trajectory of x, and these numbers may be larger than x. The time complexity bound relies on the empirical observation

[8] and stochastic model [9] which assume that the maximum value attained by the iterates starting at x grows like x^2. In this case, the number of required bits should be at most about twice as large as the number of bits required for the starting value. As the run time is proportional to the k, the algorithm is fast if the number of steps k is small. Putting it all together, the entire algorithm takes $k \times O(n)$, or simply $O(kn)$.

The worst case that could happen is that, in each iteration of the algorithm, the β_i will be equal to 1 (except the very first step for even inputs). This would however contradict the validity of the Collatz conjecture. Thus we have $F(x) < 2x$, and after k steps, we have $F^k(x) < 2^k x$. Thus the number of required bits is $k + n$, and the hypothetical worst-case time complexity is $O(k(k + n))$, still assuming the validity of the Collatz conjecture.

Similarly to the previous case, the algorithm initially computes the sequence $x, F(x), F^2(x), \ldots, 1$, while remembering all decisions $\operatorname{ctz}(3x + 1)$. Because the number of decisions is equal to the k, the amount of memory space required to solve this algorithm is bounded by $O(k)$, or $O(k + n)$ when inputs are included. Then the algorithm runs backward, starting from a, and computes the sequence $a, \ldots, a\,F^2(x), a\,F(x), a\,x$, and therefore gets the $a \times x$ at the end. The key is that, if the decision $\operatorname{ctz}(3x + 1) = \operatorname{ctz}(3m + a)$ is known, one can quickly recover the $m = a\,x$ from $G_a(m) = a\,F(x)$ using an inverse of (3), namely

$$m = \frac{G_a(m)\,2^{\operatorname{ctz}(3m+a)} - a}{3}. \tag{5}$$

3 Comparison with other Algorithms

Consider the multiplication of two n-digit numbers. The time complexity of the long grade-school multiplication is obviously in $O(n^2)$. Faster algorithms like Karatsuba [10] or Toom–Cook [11] further decrease the exponent (e.g., up to $O(n^{1.404})$ for 4-Way Toom–Cook algorithm). FFT-based multiplication algorithms can get the runtime down to almost $O(n \log n)$. For example, the Schönhage–Strassen algorithm [12] exhibits the time complexity $O(n \log n \log \log n)$.

From a practical perspective, the new $O(kn)$ multiplication algorithm will be useful if the number of steps k is known in advance (and can be assured that it is small enough). This is, for example, the case of multiplication with numbers pre-calculated in a lookup-table. If no prior knowledge of the multiplicand is known in advance, the average speed of the algorithm will most likely be much slower than any other known algorithm.

To illustrate the $O(kn)$ complexity of the algorithm, see the experiment in Fig. 1, which compares the $O(kn)$ algorithm against the multiplication algorithm implemented in the GNU MP library (libgmp). The libgmp switches different underlying algorithms (Karatsuba, Toom-Cook, FFT), depending on the length of multiplicands. The horizontal axis indicates the length of both multiplicands ranging from 1024 bits to about 1 million bits). The first multiplicand has been chosen in such a way that the number of steps k is 35. This is a very special case used just to illustrate the $O(kn)$ complexity. The second multiplicand has been generated as a uniformly distributed

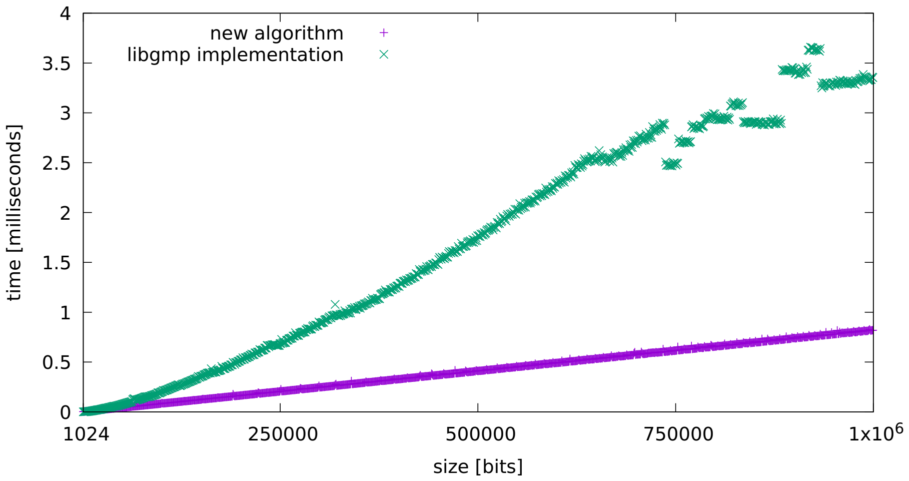

Fig. 1 Comparison of computing time for multiplying two n-bit numbers

random integer (a Mersenne Twister algorithm). The vertical axis indicates the time (in milliseconds) needed for multiplication by both algorithms (average of 1000 measurements). The results were obtained on 3.0 GHz CPU (AMD Ryzen Threadripper 2990WX).

To recommend to the reader a number where the number of steps is not too large, consider, for example, this 64-bit integer $x = 10760600709663905109$ with $k = 6$ steps. The integer contains 32 non-zero bits in its binary representation, so the long multiplication would have to perform 32 additions, whereas our algorithm has to perform 6 subtractions.

4 Conclusion

It is possible to perform multiplication of large n-digit numbers in fewer operations than the state-of-the-art algorithms. However, this possibility is conditioned by the prior knowledge of the number of steps k in the Collatz function for the multiplicand. If this number is small enough, the new $O(kn)$ algorithm achieves a higher speed compared to the other algorithms. Since there is no known method for fast calculation of the k (other than computing the iterates of the Collatz function), the algorithm is most likely only of theoretical interest.

Acknowledgments Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602 and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center – LM2015070.

References

1. Hercher, C.: Über die Länge nicht-trivialer Collatz-Zyklen. Die Wurzel, 6 and 7 (2018)
2. Tao, T.: The Collatz conjecture, Littlewood-Offord theory, and powers of 2 and 3 (2011)
3. Lagarias, J.C.: The $3x + 1$ problem and its generalizations. The American Mathematical Monthly **92**(1), 3–23 (1985)
4. Crandall, R.E.: On the "$3x + 1$" problem. Mathematics of Computation **32**(144), 1281–1292 (1978)
5. Lagarias, J.C.: The $3x + 1$ problem: An annotated bibliography (1963–1999) (sorted by author). arXiv:math/0309224 (2003)
6. Lagarias, J.C.: The $3x + 1$ problem: An annotated bibliography, II (2000-2009). arXiv:math/0608208 (2006)
7. Chamberland, M.: Una actualizacio del problema 3x+1. Butlleti de la Societat Catalana de Matematiques **22**(2), 1–27 (2003). An English version "An Update on the 3x+1 Problem"
8. Oliveira e Silva, T.: Empirical verification of the 3x+1 and related conjectures. In: Lagarias, J.C. (ed.) The Ultimate Challenge: The 3x+1 Problem, pp. 189–207. American Mathematical Society (2010)
9. Lagarias, J.C., Weiss, A.: The $3x + 1$ problem: Two stochastic models. Annals of Applied Probability **2**(1), 229–261 (1992)
10. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Physics – Doklady **7**, 595–596 (1963). Originally published in 1962
11. Toom, A.: The complexity of a scheme of functional elements realizing the multiplication of integers. Soviet Mathematics – Doklady **3**, 714–716 (1963). Originally published in Russian
12. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing **7**(3), 281–292 (1971)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.