What Is the Best Way to Add Large Number of Integers: Number-by-Number As Computers Do Or Lowest-Digits-Than-Next-Digits-Etc As We Humans Do?

Olga Kosheleva[1] and Vladik Kreinovich[2]
[1]Department of Teacher Education
[2]Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
olgak@utep.edu, vladik@utep.edu

Abstract

When we need to add several integers, computers add them one by one, while we usually add them digit by digit: first, we add all the lowest digits, then we add all next lowest digits, etc. Which way is faster? Should we learn from computers or should we teach computers to add several integers our way?

In this paper, we show that the computer way is faster. This adds one more example to the list of cases when computer-based arithmetic algorithms are much more efficient than the algorithms that we humans normally use.

1 Formulation of the Problem

When we humans need to add several integers:

- we usually first add their lowest digits,
- then we add their next lowest digits,
- etc.

On the other hand, when computers are given a task of adding several integers, they add these integers number-by-number:

- first, they add the first two numbers,
- then they add the third number to the resulting intermediate sum,

- etc.

Which way is better?

- Should we program computers to add several numbers our way?
- Or should be we learn from the computers and add numbers their way?

This is the question that we answer in this paper.

2 Analysis of the Problem

Notations.

- Let us denote by n the number of integers that we need to add, and
- let us denote by d the number of digits in each of these numbers.

How many extra digits do we need to represent the sum? When we add n d-digit integers, the sum is n times larger than each of the original d-digit integers. So, to represent this sum, we need to use additional digits. How many additional digits do we need?

Every time we add one more digit, the size of the numbers that can be represented increases by a factor of B, where B is the base of the corresponding numerical system:

- $B = 2$ for most computers, and
- $B = 10$ for human computations.

Adding two digits increases the largest number by a factor of B^2. In general, adding k digits increases the largest number by a factor of B^k.

To be able to increase the size by a factor of n, we therefore need to use k additional digits, where $B^k \approx n$. Thus, we need $k = \log_B(n)$ additional digits.

What if we add numbers one by one? When we add two d-digit numbers, we need d digit operations; see, e.g., [1]. When we add numbers one by one, eventually, we will get to numbers with $d + \log_B(n)$ digits, so we will need $d + \log_B(n)$ digits operations for each addition.

Overall, to find the sum of n numbers, we need to perform $n - 1 \approx n$ additions. So, we need

$$n \cdot (d + \log_B(n)) = n \cdot d + n \cdot \log_B(n) \tag{1}$$

digit operations.

What if we first add all lower digits, then all next digits, etc.? When we add all lower digits, we get the value $n \cdot B$. To represent this value, we need $\log_B(n \cdot B) = 1 + \log_B(n)$ digits. So, to perform the addition of the lowest digits of all n numbers, we need $n \cdot (1 + \log_B(n))$ digit operations.

Overall, we need to perform similar summation for all d original digits. Thus, in this case, we need overall

$$d \cdot n \cdot (1 + \log_B(n)) =$$

$$n \cdot d + n \cdot d \cdot \log_B(n) \tag{2}$$

digit operations.

Conclusion: computer way is much faster. By comparing the formulas (1) and (2), we see that number-by-number addition is faster: for the digits-by-digits addition, the term added to $d \cdot n$ is d times larger than for the number-by-number addition.

Discussion. This conclusion is in line with the general trend, that the arithmetic algorithms used by humans are far from being optimal [1]. For example:

- while we have two different algorithms for addition and subtraction, computers use 2's complement implementation of negative numbers that allows both operations to be be performed the same way [1];
- our digit-by-digit multiplication requires $O(d^2)$ digit operations, while there exist faster algorithms based on Fast Fourier Transform that require $O(n \cdot \ln(n)) \ll O(n^2)$ digit operations [1];
- our "long division" algorithm is also not the best: the usual computer way of first computing $1/b$ and then computing $a \cdot (1/b)$ is faster [1].

Another known case when computer-based algorithms are faster is sorting: computer-based mergesort algorithm is much faster than the insertion sort algorithm that we normally use when we need to sort a group of items [1].

In this sense, our paper has added one more example where a usual human algorithm can be improved.

Acknowledgments

This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

References

[1] Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2009.