Just-in-Accuracy: Mobile Approach to Uncertainty

Martine Ceberio, Christoph Lauter and Vladik Kreinovich*

Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA
E-mail: mceberio@utep.edu; cqlauter@utep.edu; vladik@utep.edu
**Corresponding Author*

Received 29 October 2023; Accepted 26 February 2024

Abstract

To make a mobile device last longer, we need to limit computations to a bare minimum. One way to do that, in complex control and decision making problems, is to limit precision with which we do computations, i.e., limit the number of bits in the numbers' representation. A problem is that often, we do not know with what precision should we do computations to get the desired accuracy of the result. What we propose is to first do computations with very low precision, then, based on these computations, estimate what precision is needed to achieve the given accuracy, and then perform computations with this precision.

Keywords: Uncertainty, precision, accuracy, mobile computing.

1 Formulation of the Problem

For mobile devices, an important restriction is energy. Mobile devices are very convenient, but they need to be recharged ever so often – e.g., after a certain number of hours. This need comes from the fact a mobile device can

Journal of Mobile Multimedia, Vol. 20_3, 665–678.
doi: 10.13052/jmm1550-4646.2036
© 2024 River Publishers

store only a limited amount of energy; see, e.g., [1, 2, 4, 10]. Every time we perform a computation, we use some energy.

How to make mobile devices last longer. Since every bit operation requires energy, the only way to make a mobile device last longer is to reduce the number of bit operations needed to perform the corresponding computations.

The number B of bit operations can be estimated as the number A of arithmetic operations times the number b of bit operations needed for each arithmetic operation: $B = A \cdot b$. How can we minimize this product?

Existing methods for limiting computations in mobile devices to increase their autonomy. Most current computational devices use a fixed number of bits to represent the corresponding values. In this arrangement, a natural idea is to minimize the number of arithmetic operations A, i.e., to come up with faster (fewer-operations) algorithms for performing user-required computations.

This has been the main focus of designers of mobile devices. As a result, most algorithms used in mobile devices have already been optimized from this viewpoint – the corresponding number of arithmetic operations is as small as possible. And still, users would like to have devices that last even longer.

How can we make mobile devices last even longer? Since the number A of arithmetic operations is, in many cases, already as small as we can make it, the only way to further decrease the number of bit operations is to decrease the number of bit operation needed to perform one arithmetic operation.

How can we decrease the number of bit operations. For each arithmetic operation, the number of bit operations depends on the number n of bits used to represent a number. The more bits we need to process to perform each operation, the more bit operations we need. So, the only way to make a mobile device last longer seems to be using fewer bits to represent the corresponding numbers.

But can we do it? There is a reason why modern computers use a large number of bits (usually, 64, sometimes 32) to perform arithmetic operations: many computations require high precision, and computations with lower precision result in lower accuracy than we want. For some computations, even 64 bits are not enough, we need double precision (i.e., 128 bits) or even higher.

Because of such computations, we cannot simply reduce the number of bits used to represent each number – that will make many computations impossible.

Some computational tasks do not require high accuracy but some do: examples. Some computations require high accuracy, but many computations do not need such an accuracy. Let us take medical applications as an example. When an app installed on a mobile device estimates the dosage of a medicine needed for a patient – based on the patients age, weight, and disease severity – two (or even one) decimal digits accuracy is usually sufficient, since the weight is only known with this accuracy.

This does not mean, of course, that high accuracy computations are not needed. For example, in apps for EEG processing, when we want to detect a possible incoming heart attack based on relatively weak signals, we want to extract as much information from the measurements as possible, even if this would mean spending more energy on the corresponding computations.

So what can we do? So how can we decrease the overall amount of bit operations and still be able to perform computations requiring high accuracy?

For some algorithms, we know what precision to use to achieve the desired accuracy. For example, in most applications of deep learning, 8-bit computations are sufficient; see, e.g., [9]. In such cases, this is exactly the precision that we need to use for such computations – provided, of course, that the CPU allows computations with different precision.

This knowledge is available for many existing algorithms. However, new algorithms appear all the times, algorithms for which such a study of needed precision has not yet been done. What can we do in this case – other than compute all these algorithms with the highest precision?

What we do in this paper. In this paper, we provide a possible solution to this problem: namely, we show how to decrease the overall number of bit operations without sacrificing the desired accuracy.

2 What We Propose

Our main idea. If we have an algorithm for which we do not know what precision we need to achieve the desired accuracy, then:

- first, we perform computations with low precision;

- based on results of these computations, we determine what precision is absolutely needed to achieve the desired accuracy; and then
- we perform computations with thus determined precision.

Terminological comment. In analogy with just-in-time delivery, when we save on storage expenses by scheduling delivery for exactly the time at which the delivered objects are needed, we call the proposed approach – in which we exactly as many digits as needed to achieve the desired accuracy, no more, no less – *just-in-accuracy* approach.

How we can implement this idea. In order to implement this idea, we need to do the following three tasks:

- first, we need to find out how to estimate the accuracy of the result of computations with low precision;
- second, we need to find out how to use this estimate to determine the desired precision;
- finally, we need to make sure that we indeed decrease overall number of bit operations.

Let us consider these tasks one by one.

Why do we need the third task? The first two tasks are clearly needed, but why is the third task important? Because:

- on the one hand, when low-precision computations are already sufficient, the proposed approach definitely decreases the number of bit operations;
- on the other hand, for problems that really need high-precision computations, we have to perform these computations anyway; so in our approach, in addition to these high-precision computations, we also perform additional low-precision computation – and thus, increase the number of bit operations.

We need to make sure that decrease is larger than the increase – then the overall number of bit operations will decrease.

How practical is it? At present, most computational devices use fixed number of 8-bit bytes to store numbers. From this viewpoint, it may look like we do not have much a choice: either we use 8 bits, or 16 bits, or 24 bits, etc. However, from the viewpoint of computer design, there is nothing magic about 8 buts: in the early days of computing, some computing – PDP-10 the most well-known example – used variable numbers of from 1 to 36 to store numbers [3], and a recent book [7] by John L. Gustafson, one of the world

leaders in computer engineering – shows that a similar feature is possible with the modern computer technology as well.

Historical comment. Dr. Gustafon is former Director at Intel Labs and former Chief Product Architect at AMD. He introduced cluster computing in 1985 and first demonstrated scalable massively parallel performance on real applications in 1988. He won the inaugural ACM Gordon Bell Prize for achievements in high performance computing. He is also a recipient of the IEEE Computer Society's Golden Core Award.

How to estimate the accuracy of the result of low-precision computations. The relative inaccuracy caused by limited precision is relatively small: if we use n-bit precision, then the relative round-off error is of order 2^{-n}. Even if we use a very low 8-bit precision, this error is about $2^{-8} \approx 1.5\%$.

Thus, we can apply the idea typically used in physics: we expand the dependence of the result of the round-off errors in Taylor series and keep only the first few terms in this expansion; see, e.g., [6, 11]. For the relative error of 1.5%, its square is about 0.02% – which is much smaller than the error itself. Thus, to estimate the effect of these errors, we can safely ignore terms which are quadratic (and of higher order) in terms of these errors, and only keep linear terms. So, the accuracy of the resulting computations is a linear function of these errors – i.e., it is proportional to 2^{-n}.

How can we estimate this error? We can repeat the low-precision computations with two different low precisions: n_1 and $n_2 > n_1$; for example, we can take $n_1 = 8$ (enough to get at least a crude approximation in most problems [9]) and $n_2 = n_1 + 1$. For $n_2 = n_1 + 1$, the result r_1 of the n_1-precision computations differs from the unknown actual value a by some value $r_1 - a \approx c \cdot 2^{-n_1}$, while the result r_2 is the n_2-precision computations differs by the value

$$r_2 - a \approx c \cdot 2^{-(n_1+1)} = 0.5 \cdot c \cdot 2^{-n_1}.$$

Thus, we have

$$r_1 - r_2 = (r_1 - a) - (r_2 - a) \approx c \cdot 2^{-n_1} - 0.5 \cdot c \cdot 2^{-n_1} = 0.5 \cdot c \cdot 2^{-n_1} \approx r_2 - a.$$

Hence, the difference $r_1 - r_2$ between the results of these two computations can be used as an approximate estimate for the accuracy $r_2 - a$ of the somewhat-more-precise computation result r_2. The corresponding relative accuracy is thus approximately equal to the ratio

$$\frac{|r_1 - r_2|}{|r_2|}.$$

It is reasonable to gauge this relative accuracy by the number of correct digits in the binary expansion of the computation result r_2. Having d digits means relative accuracy 2^{-d}. Thus, this value d can be determined from the approximate equality

$$2^{-d} \approx \frac{|r_1 - r_2|}{|r_2|},$$

hence

$$d \approx \log_2(|r_2|) - \log_2(|r_1 - r_2|).$$

How to determine the desired precision. Let us denote the desired number of correct bits in the computation result by k. This means that we want to have the relative accuracy 2^{-k} of the computation result. How many bits do we need to use in our computations to reach this accuracy?

Due to our linearity assumption, in general, the relative accuracy resulting from computations with n digits – for which the relative round-off error is about 2^{-n} – is approximately equal to $c \cdot 2^{-n}$ for some constant c. To find the value of this constant c, we need to take into account that, based on our low-precision computations, we know that the relative round-off error 2^{-n_2} leads to the accuracy 2^{-d} in the computation result. Thus, $2^{-d} = c \cdot 2^{-n_2}$, so $c = 2^{n_2 - d}$.

We want to find out the value n for which the resulting accuracy is $c \cdot 2^{-n} = 2^{-k}$. Substituting the above formula for c into this expression, we conclude that $2^{n_2-d} \cdot 2^{-n} = 2^{n_2-d-n} = 2^{-k}$. Taking binary logarithm of both sides of this equality, we get $n_2 - d - n = -k$, so $n = k + n_2 - d$.

Let us summarize what we have found.

Resulting algorithm. Suppose that we want to get the result with k significant digits. Let us select some small number n_1.

- First, we perform the computations with n_1 bits and with $n_2 = n_1 + 1$ bits, and get approximate results r_1 and r_2.
- Then, we find $d \approx \log_2(|r_2|) - \log_2(|r_1 - r_2|)$.
- Finally, we perform computations again, this time with $n = k + n_2 - d$ bits.

3 The Proposed Algorithm Does Decrease the Overall Number of Bit Operations

How can we decide whether this algorithm decreases the overall number of bit operations? To answer this question, we need to compare the overall

number of bit operations in two settings:

- the traditional setting, when all the computations are performed with the same high precision N, and
- the proposed setting, when we first perform two computations with low-precision values n_1 and n_2, and then perform computations with the needed precision n.

To compare these two setting, we need to know:

- how frequent are situations with different values n of needed precision, and
- how the number of bit operations grows with n.

Let us try to answer these two questions.

How frequent are situations with different values n of needed precision? As we have mentioned, for different problems, we need different precision, i.e., different numbers n of bits-per-number, from $n = 1$ to $n = N$ for some large N (e.g., $N = 64$ or $N = 128$). We do not know the frequency with which different values n appear, and we have no reason to believe that some of these values are more frequent than other. Thus, it is reasonable to use Laplace Indeterminacy Principle (see, e.g., [8]) and conclude that all these N values are equally probable, i.e., that each of them occurs with the same probability $1/N$.

How does the number of bit operations needed for each arithmetic operation change with n?

- For addition and subtraction, this number is proportional to n: we need a constant number of operations per bit.
- For multiplication, we need order of n^2 operations: indeed, the usual multiplication algorithm means that for each digit in the second number, we use n bit operations to multiply the first number by this digit, and then we add n n-digit results.

Since $n^2 \gg n$, in the first approximation, we can simply take into account n^2-operations – e.g., multiplication – and ignore time needed for addition and subtraction. Thus, we can conclude that the number of bit operations grows with n as $c \cdot n^2$ for some constant c.

Now, we are ready to compare the two settings. Since we answered both questions, we can now provide the desired comparison.

- In the traditional setting, for each problem, we need $c \cdot N^2$ bit operations.

- In the proposed setting, for needed precision n, we need $c \cdot n_1^2 + c \cdot n_2^2 + c \cdot n^2$ bit operations. All values n from 1 to N are equally probable, so the average number of bit operations is equal to

$$c \cdot n_1^2 + c \cdot n_2^2 + c \cdot \frac{1}{N} \cdot \sum_{n=1}^{N} n^2. \tag{1}$$

It is known that the last sum in the expression (1) is equal to

$$\sum_{n=1}^{N} n^2 = \frac{N \cdot (N+1) \cdot (2N+1)}{6}. \tag{2}$$

For large N, this expression is approximately equal to $N^3/3$, so the expression (1) takes the form

$$c \cdot \left(n_1^2 + n_2^2 + \frac{N^2}{3} \right). \tag{2}$$

When $n_1 \ll N$ and $n_2 \ll N$, the number of bit operations in the proposed setting is approximately equal to $c \cdot (N^2/3)$ and is, thus, three times smaller than in the traditional setting.

First conclusion. *So indeed, the proposed setting decreases the number of bit operations: it decreases this number by a factor of three.*

Comment. From the viewpoint of algorithmic complexity, making computations three times faster is not a big deal: most algorithms provide must more drastic speed-up; see, e.g., [5]. But let us recall that our goal is to extend the between-charges time for mobile computational devices. From this viewpoint, increasing the time-to-next-charging by a factor of three – e.g., from 8 hours to 24 hours – is a significant increase.

A natural question: can we do even better? We decreased the number of bit operations by a factor of three. A natural question is: is this the best we can do? Or can we decrease the number of bit operations even more?

To answer this question, let us consider the ideal situation when we know the exact precision n needed for each computational problem. In this ideal case, for each value n, we need exactly $c \cdot n^2$ bit operations. Thus, the average number of bit operations is equal to

$$c \cdot \frac{1}{N} \cdot \sum_{n=1}^{N} n^2.$$

We already know that for large N, this expression is approximately equal to $c \cdot (N^2/3)$ – which is exactly what our setting provides. Thus, we can make the following conclusion.

Second conclusion. *The proposed method is asymptotically optimal: it provides (asymptotically) the smallest possible number of bit operations – and thus, the smallest possible energy consumption that we can achieve without improving the algorithms.*

Let us summarize our results.

4 Conclusions

While, in general, mobile devices are convenient to use, this convenience comes at the expense of the need to recharge these devices every few hours. To make their use more convenient, it is desirable to make sure that they last as long as possible. Every bit operation requires some energy. So, to make the mobile devices last longer, a natural idea is to minimize the number of bit operations. In a computational device, the only hardware-supported operations are elementary arithmetic operations. Thus, whatever we compute, the computational device performs a sequence of arithmetic operations. Because of this, designers of mobile devices have been focused mostly on minimizing the number of arithmetic operations needed to perform the user-required tasks. As a result of these efforts, at present, this number is practically as small as it can be.

How can we further decrease the devices' energy consumption? A natural idea is to take into account that the overall number of bit operations can be obtained by multiplying the number of elementary arithmetic operations by the number of bit operations needed for each arithmetic operation. The number of these bit operations depends on the number of bits used to represent each number. Since the number of arithmetic operations is already close to its minimum, it is therefore desirable to decrease the number of bits per number to a bare minimum needed for the desired uncertainty of the computational result.

The problem with this idea is that for many complex algorithms used in modern mobile devices, it is difficult to estimate a priori how many bits are needed to get the result with the desired accuracy. To solve this challenging problem, we propose: (1) first, to perform the computations with two small numbers of bits, and (2) then, to use extrapolation to estimate how many bits are needed to reach the desired accuracy. While we spend extra time

performing these preliminary low-bit computations, we, in general, save time overall by avoiding the energy-consuming use of too many bits.

Specifically, we show: (1) that the proposed setting decreases the number of bit operations by the factor of three, and (2) that the proposed method is asymptotically optimal: it provides (asymptotically) the smallest possible number of bit operations – and thus, the smallest possible energy consumption that we can achieve without improving the algorithms.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

The authors are thankful to Yuriy Kondratenko for his encouragement, and to the anonymous referees for valuable suggestions.

References

[1] A. Abunaser and S. Alshattnawi, "Mobile cloud computing and other mobile technologies: survey", *Journal of Mobile Multimedia*, 2013, Vol. 8, No. 4, pp. 241–252.

[2] M. Aleksy, R. Gitzel, G. Vollmar, N. Fantana, C. Stich, and M. Takizawa, "Techniques for the efficient resource management of contextsenstivie mobile applications and their utilization in industrial field service", *Journal of Mobile Multimedia*, 2013, Vol. 4, No. 3–4, pp. 200–209.

[3] Digital Equipment Corporation (DEC), *PDP-l0 System Reference Manual*, Maynard, Massachusetts, 1968, https://web.archive.org/web/20170925091321/http://bitsavers.org/pdf/dec/pdp10/KA10/DEC-10-HGAA-D_PDP-10_System_Reference_Manual_May1968.pdf.

[4] B. Debaillie, F. Brunier, D. Morche, E. Isa, and J. Craninckx (eds.), *Technologies Enabling Future Mobile Connectivity and Sensing*, River Publishers, Aalborg, Nordjylland, Denmark, 2024.
[5] Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2022.
[6] R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics*, Addison Wesley, Boston, Massachusetts, 2005.
[7] J. L. Gustafson, *The End of Error: Unum Computing*, Chapman & Hall/CRC, Boca Raton, Floria, 2015.
[8] E. T. Jaynes and G. L. Bretthorst, *Probability Theory: The Logic of Science*, Cambridge University Press, Cambridge, UK, 2003.
[9] C. Lauter and A. Volkova, "A framework for semi-automatic precision and accuracy analysis for fast and rigorous deep learning", *Proceedings of the 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, Portland, Oregon, USA, June 7–10, 2020, pp. 103–110.
[10] K. Sha, A, Striegel, and M. Song (eds.), *Advances in Computer Communications and Networks: From Green, Mobile, Pervasive Networking to Big Data Computing*, River Publishers, Aalborg, Nordjylland, Denmark, 2016.
[11] K. S. Thorne and R. D. Blandford, *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*, Princeton University Press, Princeton, New Jersey, 2021.

Biographies

Martin Ceberio is a professor of Computer Science at the University of Texas at El Paso. She joined UTEP in 2003 after obtaining her PhD in Computer Science from the University of Nantes, France (2003). Her research revolves around reliable decision-making under uncertainty, optimization, constraint solving, solving dynamical systems, and studying neural networks

with uncertainty. She is passionate about teaching and mentoring students. She is a fierce advocate of her students' success and seizes every opportunity to provide advice and mentoring. Her other passion is in broadening participation in Computer Science and fostering cultures of inclusion. She hosts young women high-school students as research interns in her research lab to introduce them to computing in summers.

Christoph Lauter received his PhD in Computer Science from Ecole Normale Superieure de Lyon, France, in 2008. He worked as a Software Engineer in the Numerics Team at Intel Corporation in 2008-10 and in 2022. Since 2010, he has been Maitre de Conferences (Associate Professor) at Sorbonne Universite, Paris, France. In 2018, he joined University of Alaska Anchorage (UAA), where he worked as an Assistant Professor until 2022. In 2022, he joined University of Texas at El Paso (UTEP), where he holds an Associate Professor position. His research interests are Floating-Point Arithmetic, Uncertainty Quantification and Validated Numerical Computing.

Vladik Kreinovich is Professor of Computer Science at the University of Texas at El Paso. His main interests are representation and processing of uncertainty, especially interval computations and intelligent control. He has published 13 books, 44 edited books, and more than 2,000 papers.

Vladik is President-Elect of the International Fuzzy Systems Association (IFSA), Secretary of the IEEE Systems, Man, and Cybernetics Society, Fellow of International Fuzzy Systems Association (IFSA), Fellow of Mexican Society for Artificial Intelligence (SMIA), Fellow of the Russian Association for Fuzzy Systems and Soft Computing.