Using of Redundant Signed-Digit Numeral System for Accelerating and Improving the Accuracy of Computer Floating-Point Calculations

Otsokov Sh.A[1]
Dept. of Computing Machines, Systems and Networks
National Research University "Moscow Power Engineering Institute", Moscow, Russian Federation

Magomedov Sh.G[2]
Dept. of Intelligent Information Security Systems
MIREA Russian Technological University
Moscow, Russian Federation

***Abstract*—The article proposes a method for software implementation of floating-point computations on a graphics processing unit (GPU) with an increased accuracy, which eliminates sharp increase in rounding errors when performing arithmetic operations of addition, subtraction or multiplication with numbers that are significantly different from each other in magnitude. The method is based on the representation of floating-point numbers in the form of decimal fractions that have uniform distribution within a range and the use of redundant signed-digit numeral system to speed up calculations. The results of computational experiments for evaluating the effectiveness of the proposed approach are presented. The effect of accelerating computations is obtained for the problems of calculating the sum of an array of numbers and determining the dot product of vectors. The proposed approach is also applicable to the discrete Fourier transform.**

***Keywords*—High-precision computation; redundant signed-digit numeral system; signed-digit floating-point format; redundant signed-digit arithmetic; decimal fractions**

I. INTRODUCTION

Most computer calculations are carried out in floating-point format and double precision computer calculations are sufficient for solving many computational problems.

However, there are a number of problems, for example, in computational geometry and other areas where double precision floating-point arithmetic is not sufficient [1]. To solve such problems, the well-known libraries of high-precision computations are used, such as ZREAL (Russia), MPARITH (Germany), GMP (USA), which implement floating-point calculations at the software level with a mantissa length set by the user [2, 3, 6, 7, 8, 9, 10].

But these libraries have the property of sharply increasing the calculation time with the increasing in the length of the mantissa and the number of arithmetic operations. In addition, they have the inherent disadvantages of the floating-point format itself, which does not always guarantee an accurate result of computer calculations.

One of such disadvantages is the uneven distribution of floating-point numbers. Fig. 1 below shows the uneven distribution of normalized floating-point numbers with the mantissa length of 3 binary digits and the order from 0 to 4 [4].

As an example of the loss of accuracy in computer calculations consider the problem of determining the dot product of two vectors with following coordinates:

$$\vec{\mathbf{x}} = (10^{\alpha}, 1223, 10^{\alpha-1}, 10^{\alpha-2}, 3, -10^{\alpha-5}),$$

$$\vec{\mathbf{y}} = (10^{\beta}, 2, -10^{\beta+1}, 10^{\beta}, 2111, 10^{\beta+3}),$$

where α, β are given parameters.

The true result is 8779. The dot product was calculated in the single precision format, the relative error was calculated with the constant value of $\alpha = 1$ and β ranging from 1 to 21.

The relative error of the dot product was calculated using the formula:

$$\delta = \frac{|(x, y) - 8779|}{8779} \cdot 100 \quad (1)$$

The dependence of the relative error of the dot product of vectors on the parameter β in single precision floating-point format is presented in the graph shown in Fig. 2.

Fig. 2 shows that for significantly different values of α and β, starting from the value 18 for β, there is a sharp loss in the accuracy of the dot product results, which is due to the fact that calculations are performed with numbers that differ greatly from each other in magnitude.

Using double precision floating-point format, the increase in the relative error occurs at larger values of β compared to the single precision format.

Fig. 1. Distribution of Floating-Point Numbers.

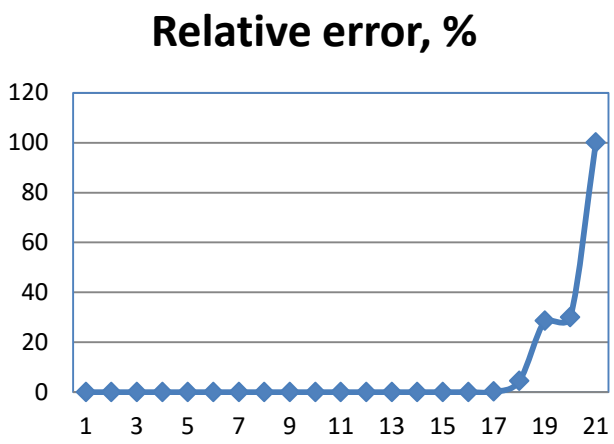

Fig. 2. Dependence of the Relative Error on the Parameter β.

Consider the following example that demonstrates the loss of precision in floating-point calculations associated with the discrete Fourier transform.

Let a vector $\vec{\mathbf{x}} = (x_1, x_2, x_3, \ldots, x_n)$

be given with coordinates defined as follows:

$$x_i = \begin{cases} \frac{i}{10}, \text{ if } i \text{ is odd} \\ \frac{i}{10} + \alpha, \text{ if } i \text{ is even} \end{cases} \quad (2)$$

$$i = 1, \ldots, n, \ \alpha > 0$$

The discrete Fourier transform of a vector $\vec{\mathbf{x}}$ into a vector $\vec{\mathbf{y}}$ is performed using the following formula:

$$y_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \left[\cos \frac{2 \cdot \pi \cdot j \cdot k}{n} - i \cdot \sin \frac{2 \cdot \pi \cdot j \cdot k}{n} \right],$$

$$0 \le k \le n-1 \quad (3)$$

The inverse Fourier transform is performed using the following formula:

$$x_k = \sum_{j=0}^{n-1} y_j \cdot \left[\cos \frac{2 \cdot \pi \cdot j \cdot k}{n} + i \cdot \sin \frac{2 \cdot \pi \cdot j \cdot k}{n} \right],$$

$$0 \le k \le n-1 \quad (4)$$

Obviously, if we carry out the direct discrete Fourier transform, then the inverse Fourier transform of the vector.

$$\vec{\mathbf{x}} = (x_1, x_2, x_3, \ldots, x_n)$$

we get the vector

$$\vec{\mathbf{x}}^* = (x_1^*, x_2^*, x_3^*, \ldots, x_n^*)$$

which should approximately coincide with $\vec{\mathbf{x}}$, and the maximum relative error of the transformations can be estimated using the formula:

$$\delta = \max_i \left(\frac{|x_i^* - x_i|}{x_i} \cdot 100 \right) \quad (5)$$

Fig. 3 also demonstrates the loss of accuracy of the floating-point calculations with increasing α.

The first goal of this work is to eliminate the sharp loss of accuracy in calculations with numbers that differ greatly from each other in magnitude. The second goal is to speed up computations by parallelizing them.

The first goal is achieved by moving from floating-point representation to decimal representation that is evenly distributed within the range, as shown in Fig. 4, for example for decimal fractions of the first degree.

The second goal is achieved through the use of a redundant signed-digit numeral system, in which redundant negative digits are introduced into the system of bases in such a way that the propagation of the carry when adding is not allowed further than one digit [4,5]. Due to this, the arithmetic operations of addition, subtraction and multiplication are parallelized, which leads to their acceleration, especially when the number of digits increases. The time required for addition or subtraction of numbers does not depend on the digit capacity of the numbers.

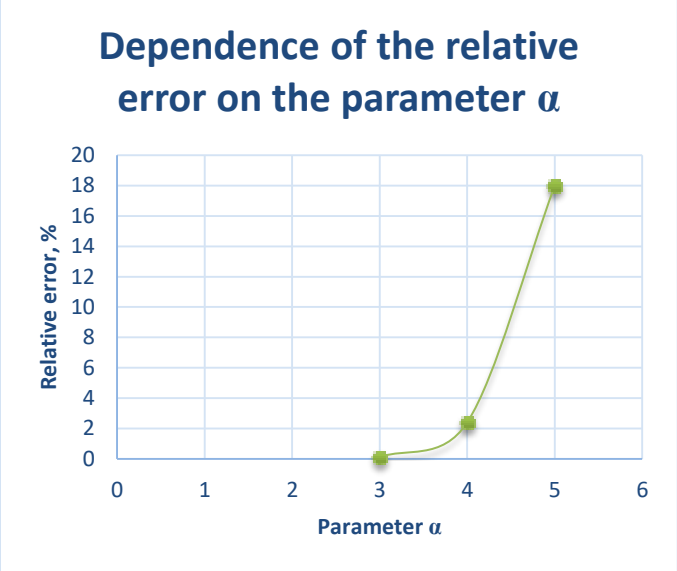

Fig. 3. Dependence of the Relative Error on the Parameter α using Single Precision Format.

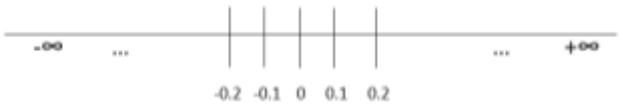

Fig. 4. Even Distribution of Decimal Fractions.

In [11,12,14,15,16] methods of representation and algorithms for performing arithmetic operations in a modular numeral system and a method for their acceleration due to parallelization in several modules are presented.

This article proposes a different approach based on the transition to a redundant signed-digit numeral system.

Such numeral system has an advantage over modular systems in that it is simpler to convert directly and inversely to the traditional numeral system and there is no need for overflow control.

A method for summing a group of numbers and calculating the dot product, oriented towards parallel implementation on a GPU, is considered. The results of experimental studies of the effectiveness of this method of high-precision calculations are obtained.

The next section considers a possible way to represent numbers in a redundant signed-digit numeral system.

II. Representing Numbers in a Redundant Signed-Digit Numeral System

Consider the representation of floating-point numbers of the following form [4]:

$$A = K \cdot q^t \tag{6}$$

where

A is a floating-point number,

K is the mantissa of the number A, an integer such that satisfies the inequality

$$|K| \le q^{n_f} - 1,$$

q is the base (radix) of the numeral system,

t is the order, an integer such that satisfies the inequality $|t| \le k_f$

n_f is a natural number characterizing the length of the mantissa of the floating-point number,

k_f is a natural number characterizing the maximum order of representable numbers.

Table I includes positive and negative minimum and maximum numbers representable in the form (6) [11,13,17,18].

The range of representable numbers (6) is as follows:

$$\left(-q^{n_f+k_f}, q^{n_f+k_f}\right) \tag{7}$$

Consider the sum of numbers of the form:

$$S = K_1 \cdot q^{t_1} + K_2 \cdot q^{t_2} + ... + K_n \cdot q^{t_n} \tag{8}$$

Suppose $t_S = \min(t_1, t_2, ..., t_n)$ then:

$$S = q^{t_S}\left(K_1 \cdot q^{t_1 - t_S} + K_2 \cdot q^{t_2 - t_S} + ... + K_n \cdot q^{t_n - t_S}\right) =$$

$$= K_S \cdot q^{t_S} \tag{9}$$

where

$$K_S = K_1 \cdot q^{t1-ts} + ... + K_n \cdot q^{tn-ts}$$

Let us estimate the maximum number of digits required to describe the result of the sum (9). Using Table I, we have:

$$K_S = K_1 \cdot q^{t1-ts} + ... + K_n \cdot q^{tn-ts}$$

$$|K_S| \le q^{nf+kf} + ... + q^{nf+kf} = n \cdot q^{nf+kf} \tag{10}$$

If $q = 10$, the maximum number of digits required to describe the sum will be equal to:

$$\lfloor \lg n + nf + kf \rfloor$$

From the last expression it can be seen that to implement the addition of groups of numbers in floating-point format calculations with large numbers are required.

Consider the format for representing floating-point numbers (6) in the signed-digit numeral system as follows:

$$A = [\,(\alpha_1, \alpha_2, ..., \alpha_i, ..., \alpha_n), t\,] \tag{11}$$

where

α_i are the digits of the signed-digit representation.

This format will be referred to as the floating-point signed-digit format.

Consider the rule for adding two numbers and a group of numbers in this format.

TABLE I. Maximum and Minimum Positive and Negative Numbers

Maximum positive	$q^{n_f+k_f}$
Minimum positive	$q^{-k_f-n_f}$
Minimum negative	$-q^{n_f+k_f}$
Maximum negative	$-q^{-k_f-n_f}$

III. Rules for Performing Arithmetic Operations in Signed-Digit Floating-Point Format

Let there be given two floating-point numbers of the form (11).

$$A_1 = [\,(\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_n), t_1\,],$$

$$A_2 = [\,(\beta_1, \beta_2, \ldots, \beta_i, \ldots, \beta_n),\ s_1\,]$$

Calculating the sum $A_1 + A_2$

$$A_4 = A_1 + A_2$$

Suppose $s_1 > t_1$. This does not change the generality of reasoning. Then:

$$A_1 \pm A_2 = K_1 \cdot q^{t_1} \pm K_2 \cdot q^{s_1} = q^{t_1} \cdot \left(K_1 \pm q^{s_1 - t_1} \cdot K_2\right) = \\ = [(\delta_1, \delta_2, \ldots, \delta_n), t_1] \quad (12)$$

Let $-6, -5, \ldots, 0, \ldots 6$ be the digits of the signed-digit numeral system.

The time required for adding numbers does not depend on the number of digits. The addition is performed using the following steps:

1. Calculation of the sum $ui = xi + yi - 10ci$, where
$$c = \begin{cases} 1, if\ (xi + yi) > 6 \\ -1, if\ (xi + yi) < -6 \\ 0, otherwise \end{cases}$$
2. Calculation of the final result $\delta_i = u_i + c_{i-1}$

Product of numbers $A_1 \cdot A_2$ is calculated by the formula:

$$A_3 = A_1 \cdot A_2 = [\,(\chi_1, \chi_2, \ldots, \chi_i, \ldots, \chi_n), t_1 + s_1\,] \quad (13)$$

where coefficients $\chi_1, \chi_2, \ldots, \chi_i, \ldots, \chi_n$ are determined according to the rules for multiplying numbers in the signed-digit numeral system.

In the next section, the first and second methods of summing a group of numbers are considered.

IV. Methods of Summing Groups of Numbers

The first method of summing groups of numbers is carried out according to the formula (12) using the addition rule presented in Section III in redundant signed-digit arithmetic in parallel over the digits and sequentially for each number of the group. This method, when implemented on GPU, requires a large number of synchronizations between cores. In Section V the results of experimental study of the effectiveness of this method are presented.

Consider the second method of summation with fewer synchronizations.

Let the number of digits of the summed numbers equal *d*, and the maximum possible number of digits required to describe the sum equal $max\,d$.

Let a set of numbers for summation be given:

1st number: $a_1^1 a_2^1 \ldots . a_d^1$

2nd number: $a_1^2 a_2^2 \ldots . a_d^2$,

kth number: $a_1^k a_2^k \ldots . a_d^k$

Let us denote their sum by *S*.

Let us add these numbers to the left with zeros to the maximum possible number of digits $max\,d$, getting the following:

$$\begin{array}{cccccc} 0 & \ldots & 0 & a_1^1 & \ldots & a_d^1 \\ 0 & \ldots & 0 & a_1^2 & \ldots & a_d^2 \\ 0 & \ldots & 0 & a_1^3 & \ldots & a_d^3 \\ 0 & \ldots & 0 & a_1^4 & \ldots & a_d^4 \\ 0 & \ldots & 0 & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & a_1^k & \ldots & a_d^k \end{array}$$

Summation is carried out as follows:

1. Each one of $max\,d$ threads in parallel and independently calculates the sums of the numbers of the corresponding column, for example, thread number $max\,d$ calculates the sum of the numbers:
$$S\max d = \sum_{i=1}^{k} a_d^i$$
thread number $max\,d - 1$ calculates the sum of the numbers:
$$S\max d - 1 = \sum_{i=1}^{k} a_{d-1}^i$$
2. At the end of the first step, the calculations are synchronized.
3. By definition the value *S* equals:

$$S = S_{\max d - d} \cdot 10^{d-1} + \ldots + S_{\max d - 3} \cdot 10^2 + S_{\max d - 2} \cdot 10 + S_{\max d - 1} \quad (14)$$

Each thread extracts digits from its result, for example, thread number $max\,d-1$ finds digits: i_1, i_2, i_3, thread number $max\,d-2$ finds digits: j_1, j_2, j_3.

4. Each thread forms numbers of the form:

$$\begin{matrix} 0 & \dots & 0 & i_1 & i_2 & i_3 \\ 0 & \dots & j_1 & j_2 & j_3 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots \end{matrix}$$

Thread number $max\,d-1$ forms the first line, thread number $max\,d-2$ forms the second line etc.

5. These numbers are summed sequentially one after another bit-parallel in the signed-digit numeral system.
6. The final result is transferred from the GPU to the CPU.

Such summation method requires d - 1 synchronizations in the process of summing this array.

Consider an example.

Suppose d = 3, $max\,d$ = 5, k = 3.

Numbers for the summation:

5	5	6
1	7	9
9	7	9

Calculated sums:

$S4 = 24$

$S3 = 19$

$S2 = 15$

$S1 = 0$

$S0 = 0$

Each thread forms one of the following numbers:

0	0	0	2	4
0	0	1	9	0
0	1	5	0	0

Then these numbers are summed sequentially one after another bit-parallel in the signed-digit numeral system according to the first method.

Next section considers the results of experimental studies of the efficiency of summation of groups of numbers.

V. Experimental Study of the Efficiency of High-Precision Summation of Groups of Numbers in the Signed-Digit Numeral System

Numerical experiments were carried out on the addition of groups of integers of different magnitudes, with the number of integers k = 10000, 100000, 1000000. The addition was carried out according to the rules of traditional arithmetic on the CPU bitwise each number of the group sequentially and using the first method on the Nvidia GPU (1.78 GHz, 1280 cores) bit-parallel and sequentially for each number of the group.

GPU calculations were performed as follows:

1) Initial data were generated randomly, integers of fixed length were generated and stored in arrays.

2) Arrays were transferred to the GPU.

3) A number of threads were created matching the number of digits. Each thread carried out sequential summation of the array numbers in its corresponding digit in parallel and independently.

4) The result of the summation was transferred from the GPU to the CPU.

The time required for summation of numbers on the CPU and the GPU was calculated, considering the transfer of data to the GPU and in the opposite direction to the CPU. On the basis of these calculations, the absolute acceleration coefficients were determined for different numbers of digits and values of k by the formulas:

$$Kabs = \frac{Tcpu}{Tgpu} \quad (15)$$

Fig. 5 shows the dependence of this coefficient on the number of digits.

Fig. 5 shows that the acceleration effect provided by the first method is insignificant and is achieved for numbers exceeding 400 decimal digits.

Experiments showed that data transfer from CPU to GPU and from GPU to CPU was very fast and did not lead to delays in the computation process. One of the main reasons for the low efficiency of the first method of summation on the GPU in the signed-digit numeral system is associated with the need for synchronization after addition of each pair of numbers in the group, which slows down the computation. If the array contains k numbers, then this summation method requires k-1 synchronizations in the process of summing this array.

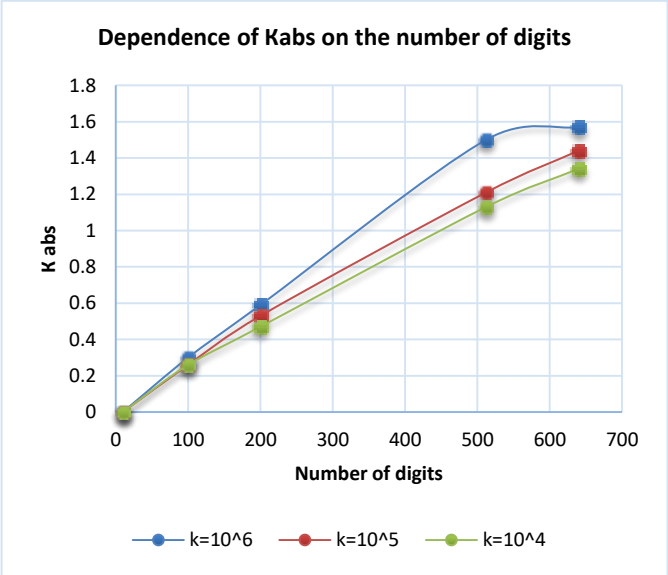

Fig. 5. Dependence of the Absolute Acceleration Coefficient on the Number of Digits for the First Summation Method.

Experiments on summing the same groups of numbers using the second method have shown that it is more efficient than the first method. For the second method the computation times on the CPU and the GPU were calculated, on the basis of which the absolute acceleration coefficient was determined by the formula (15) for different values of *k*.

Fig. 6 shows the dependence of the absolute acceleration coefficient on the number of digits for this summation method.

Fig. 6 shows that using the second summation method on the GPU with numbers comprising of 800 to 900 digits speeds up the computation 3 to 4 times in comparison with summation on the CPU. With further increase in value of *k* and the number of digits, the acceleration is supposed to be even greater.

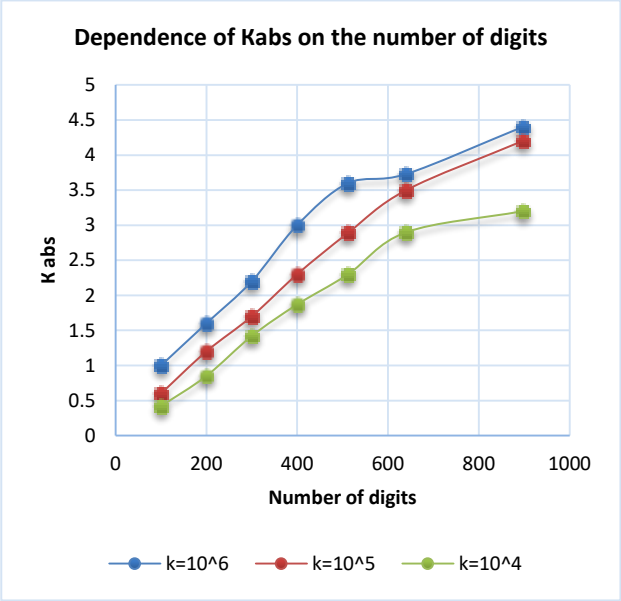

Fig. 6. Dependence of the Absolute Acceleration Coefficient on the Number of Digits for the Second Summation Method.

Next section considers a method for multiplying numbers based on redundant signed-digit arithmetic.

VI. Method for Multiplying Numbers

Let the number of digits required to represent initial data equal *d*, and the maximum possible number of digits required to represent the result of multiplication equal $max\,d$.

Let two numbers be given:

1st number: $a_1^1 a_2^1 \ldots a_d^1$

2nd number: $a_1^2 a_2^2 \ldots a_d^2$,

Let us add these numbers to the left with zeros to the maximum possible number of digits $max\,d$, getting the following:

$$\begin{array}{cccccc} 0 & \ldots & 0 & a_1^1 & \ldots & a_d^1 \\ 0 & \ldots & 0 & a_1^2 & \ldots & a_d^2 \end{array}$$

Multiplication process on the GPU involves d threads.

Thread number 1 forms the results (partial products):

$$0 \quad \ldots \quad 0 \quad a_1^1 \cdot a_d^2, a_2^1 \cdot a_d^2, \quad \ldots \quad a_d^1 \cdot a_d^2$$

Thread number 2 forms the results:

$$0 \quad \ldots \quad a_1^1 \cdot a_{d-1}^2, a_2^1 \cdot a_{d-1}^2, \quad a_d^1 \cdot a_{d-1}^2 \quad 0 \qquad (16)$$

Thread number d forms the results:

$$0 \quad \ldots \quad a_1^1 \cdot a_1^2 \quad ,\ldots, a_2^1 \cdot a_1^2, a_d^1 \cdot a_1^2 \quad \ldots \quad 0$$

The results are formed in parallel and independently by each thread for each product of two numbers.

Then they are summed up using the second method described in Section IV.

Next section considers the results of experimental studies of the efficiency of calculating the dot product of vectors.

VII. Experimental Study of the Efficiency of High-Precision Calculation of the Dot Product of Vectors in Signed-Digit Numeral System

The dot product of vectors (x,y), where x = (x1,x2,…,xk), y = (y1,y2,…,yk), is calculated as follows:

1) The values of arrays x,y are transferred to the GPU.

2) For each pair xi and yi partial products (24-26) are calculated in parallel and independently on the GPU.

3) Next, the summation of the obtained partial products is carried out using the second method.

4) The result of the dot product is transferred to the CPU.

Numerical experiments were carried out to calculate the dot product of vectors with different numbers of coordinates *k* = 10000, 100000, 1000000. The coordinates were integers with the length of 50 decimal digits. The dot product was calculated according to the rules of traditional integer arithmetic on the CPU and in redundant signed-digit arithmetic bit-parallel on the GPU.

The results of the experiments are presented on the graph in Fig. 7.

The graph shows that with the increase of the number of digits required to represent initial data *d* and the maximum possible number of digits required to represent the result of the dot product $max\,d$ the proposed method provides greater acceleration of the computation process.

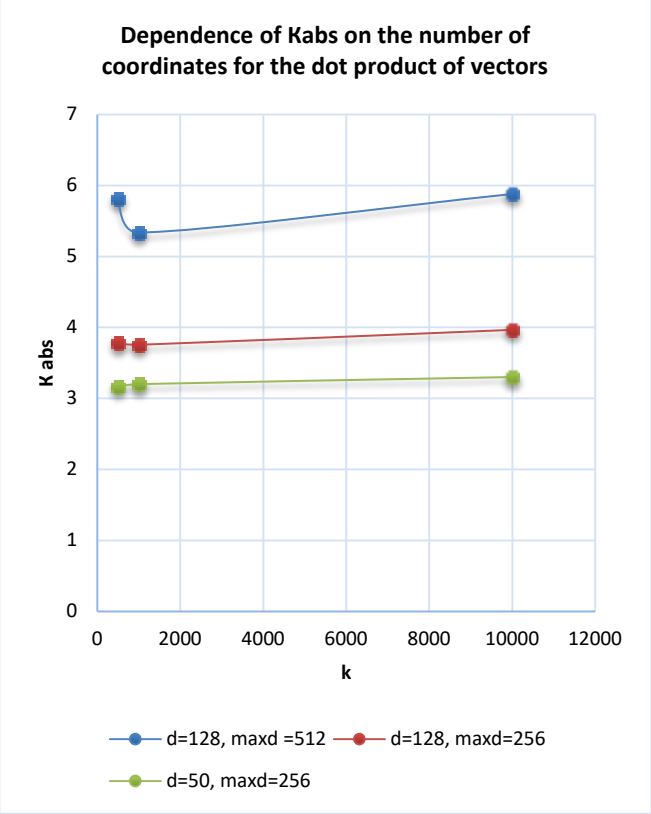

Fig. 7. Dependence of the Absolute Acceleration Coefficient on k with different Values of d, $max\ d$.

VIII. Conclusion

This article proposes an approach to software implementation of computations on a GPU, which prevents sharp loss of precision in calculations with numbers that differ greatly from each other in magnitude. The approach is based on representation of floating-point numbers in the form of decimal fractions and the use of a redundant signed-digit numeral system to speed up computations with them on the GPU.

The effect of accelerating computations was obtained and proven experimentally for the operations of summation of an array of numbers on the GPU and calculating the dot product of vectors.

The proposed approach is also applicable for the discrete Fourier transform, for the case presented in the article as well as in other cases.

References

[1] Bailey D.H., Barrio R., Borwein J.M. High-precision computation: Mathematical physics and dynamics // Applied Mathematics and Computation. 2012. Vol. 218, No. 20. P. 10106-10121.

[2] D. H. Bailey, J. M. Borwein. “High-Precision Arithmetic in Mathematical Physics”, Mathematics, 3 (2015), pp. 337–367.

[3] David H. Bailey. High-Precision Computation and Mathematical Physics Lawrence Berkeley National Laboratory, 2009

[4] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lef`evre, G. Melquiond, N. Revol, D. Stehl´e, S. Torres. Handbook of Floating-Point Arithmetic, Birkhauser, Boston, 2010, 572 p.

[5] Koren Israel. Computer arithmetic algorithms // University of Massachusetts, 2nd ed. 2002.

[6] Fousse L., Hanrot G., Lefèvre V., Pélissier P., Zimmermann P. MPFR: a multiple-precision binary floating-point library with correct rounding // ACM Transactions on Mathematical Software. 2007. Vol. 33, No. 2. Article No. 13.

[7] MParithm - package for high precision computation, 2015, www.wolfgang-ehrhardt.de/mp_intro.html

[8] GNU Scientific Library 2.5 released — 2018, https://savannah.gnu.org/forum/forum.php?forum_id=9175.

[9] Operations with multi-digit real numbers of the ZReal type. http://ishodniki.ru/list/index.php?action=name&show=pascal-math&cat=11.

[10] D. H. Bailey, X. S. Li and B. Thompson, “ARPREC: An arbitrary precision computation package,” Sep 2002, http://crd.lbl.gov/~dhbailey/dhbpapers/arprec.pdf.

[11] Otsokov Sh. A and Magomedov Sh.G, “On the Possibility of Implementing High-Precision Calculations in Residue Numeral System” International Journal of Advanced Computer Science and Applications(IJACSA), 10(11), 2019. DOI: 10.14569/IJACSA .2019.0101102.

[12] R. Amos Omondi, Benjamin , “Residue Number Systems: Theory and Implementation,” Imperial College Press, 2007.

[13] Solovyev R.A., Balaka E.S., Telpukhov D.V. Device for calculation of vector dot product with error correction based on residue number system // Problems of the development of prospective micro- and nanoelectronic systems - 2014. Proceedings / edited by A.L. Stempkovskiy. M.: IPPM RAS, 2014. Part IV. pp. 173-178.

[14] Jen-Shiun Chiang, Mi Lu, «Floating-point numbers in residue number systems» Computers & Mathematics with Applications, vol. 22, issue 10, pp. 127–140, 1991.

[15] K. S. Isupov, A. N. Mal’tsev. “A parallel-processing-oriented method for the representation of multi-digit floating-point numbers”, Vychislitel’nyye metody i programmirovaniye, 15:4 (2014), pp. 631–643 (in Russian).

[16] Magomedov S.G. Increasing the efficiency of microprocessors in an access control systems. International Journal of Engineering and Technology (UAE). 2018. T. 7. № 4.36. C. 80-83.

[17] Mukunoki D., Ogita T. Performance and energy consumption of accurate and mixed-precision linear algebra kernels on GPUs //Journal of Computational and Applied Mathematics. – 2020. – T. 372. – C. 112701.

[18] Isupov K., Knyazkov V., Kuvaev A. Design and implementation of multiple-precision BLAS Level 1 functions for graphics processing units //Journal of Parallel and Distributed Computing. – 2020. – T. 140. – C. 25-36.