УДК 004.414.2
DOI: 10.24160/1993-6982-2018-3-109-115

Сравнение универсального числового формата со стандартом IEEE 754 по критерию достоверности

С.И. Ермилов

Проблема повышения точности вычислений — актуальное направление в области теоретической информатики, вследствие большого объема научных и инженерных задач. Стандарт IEEE 754, применяемый для представления чисел с плавающей точкой, обладает определенными недостатками и имеет ограничения для прямого применения в высокоточных вычислениях. Для гарантии достоверности результата вычислений следует проводить численный анализ, но в многих случаях его не проводят. Из-за отсутствия анализа при выполнении расчетов могут возникать некорректные результаты, приводящие к неизвестному поведению программы вычислений и вызывающие серьезные последствия в программном обеспечении, критичном к точности вычислений.
В 2014 г. Дж. Густафсоном предложен формат «universal number», в котором гарантировалась достоверность полученных результатов за счет добавления дополнительных полей в битовой строке для представления числа и внедрения преимуществ интервальной арифметики. Интервальная арифметика обеспечивает достоверность результатов, а дополнительные поля в «universal number» позволяют сократить требования к пропускной способности шины памяти, а также уменьшить энергопотребление вычислительной системы. Формат автоматически подстраивает длину битовой строки под требования вычислений.
Описаны форматы IEEE 754 и «universal number», приведено их сравнение при арифметических вычислениях, решении систем линейных уравнений и вычислении выпуклой оболочки. Представлены результаты, что использование «universal number» для решения вычислительных задач позволяет повысить достоверность вычислений по сравнению с IEEE 754.

Ключевые слова: арифметика с плавающими числами, система линейных алгебраических уравнений, вычислительная геометрия.

Для цитирования: Ермилов С.И. Сравнение универсального числового формата со стандартом IEEE 754 по критерию достоверности // Вестник МЭИ. 2018. № 3. С. 109—115. DOI: 10.24160/1993-6982-2018-3-109-115.

Comparison of Universal Number with the IEEE 754 Standard with Respect to Validation Criterion

S.I. Ermilov

Improving the accuracy of computations is a topical line of research in the field of theoretical informatics due to a large volume of scientific and engineering problems. The IEEE 754 standard used for representing floating-point numbers has certain drawbacks and involves limitations for being directly used in high-precision computations. To ensure the reliability of the computation results, a numerical analysis is necessary, but in many cases such analysis is not carried out. Due to lack of analysis, incorrect results may occur in performing calculations, leading to unknown behavior of the calculation program. Incorrect results can cause serious consequences in software that is critical to the accuracy of calculations.
In 2014, John Gustafson proposed a so-called universal number format, in which reliability of the obtained results was guaranteed by adding additional fields in the bit string to represent the number and introduce the advantages of interval arithmetic. The use of interval arithmetic ensures reliability of the results, and the additional fields in the universal number can reduce the memory bus bandwidth requirements, as well as reduce the computer system power consumption requirements. The universal number automatically adjusts the length of the bit string to the computation requirements.
The article describes the IEEE 754 and the universal number formats, and compares them in carrying out arithmetic calculations, solving systems of linear equations, and calculating a convex hull. Results demonstrating that the use of universal number for solving computational problems makes it possible to improve the reliability of calculations as compared with those performed using the IEEE 754 format are presented.

Key words: floating-point arithmetic, system of linear algebraic equations, computational geometry.

For citation: Ermilov S.I. Comparison of Universal Number with the IEEE 754 Standard with Respect to Validation Criterion. MPEI Vestnik. 2018;3:109—115. (in Russian). DOI: 10.24160/1993-6982-2018-3-109-115.

Введение

Современные супер-ЭВМ развиваются за счет использования новых архитектурных решений и усовершенствования элементной базы. Тенденцией является создание процессоров с многоядерной архитектурой. Увеличение количества ядер в процессоре повышает производительность, снижает стоимость и энергопотребление. С ростом производительности расширяется спектр решаемых задач и их размерность, повышаются требования к точности компьютерных вычислений [1]. В связи с этим проблему повышения производительности ЭВМ следует решать в тесной взаимосвязи с задачей повышения точности вычислений. Большинство компьютерных вычислений проводится в арифметике с плавающей точкой, где неизбежны ошибки округле-

ния, из-за них не выполняются законы алгебры (коммутативности, дистрибутивности), например $x \neq (x + x) - x$. Нарушение алгебраических свойств приводит к появлению вычислительных аномалий. Программисты должны учитывать все особенности арифметики с плавающей точкой, составляя корректные с вычислительной точки зрения программы. Существуют правила для программистов по составлению вычислительно-корректных программ. Так, при суммировании массива чисел с плавающей точкой для уменьшения роста ошибок округления необходимо предварительно сортировать массив по возрастанию и начать суммирование с меньших чисел. Однако, они не всегда гарантируют обнаружение в расчетной программе скрытых вычислительных аномалий. Для их обнаружения необходим анализ ошибок округления. Существует множество работ, посвященных анализу ошибок округления для типовых задач линейной алгебры, таких как решение систем линейных уравнений, нахождение обратной матрицы и др. Для остальных задач провести такой анализ очень сложно и требуются специальные знания. Простейшим способом, обнаруживающим некоторые вычислительные аномалии, является решение задачи с другим режимом округления в арифметике с плавающей точкой или с большей точностью вычислений. Однако, при написании программ анализ ошибок округления обычно не проводится, а создание даже простейших функций $(x + y)/2$ требует анализа для четкой работы самой функции [2]. Поэтому существует необходимость в новом механизме, обеспечивающем достоверность вычислений и обнаружение вычислительных аномалий.

Цель данной работы — сравнение нового формата для представления чисел с форматом universal number, предложенным в 2014 г., для решения вышеуказанных проблем с числами с плавающей точкой IEEE 754.

Стандарт IEEE 754—2008

В 1985 г. комитетом IEEE был принят стандарт IEEE 754 для представления и операций над числами с плавающей точкой [3]. Его основная цель заключалась в обеспечении кроссплатформенности вычислений: одно и то же арифметическое выражение должно вычисляться с одинаковым результатом на разных архитектурах. Стандарт определял формат представления чисел, арифметические операции, преобразования и исключения.

В IEEE 754 существуют 4 класса чисел с плавающей точкой: нормальные, денормализованные, нули и бесконечности, также стандарт представляет NaN (Not a Number — «не число»), использующиеся при неопределенных и непредставимых результатах. На рис. 1 представлены поля в формате IEEE 754, а в табл. 1 — интерпретация битовой строки IEEE.

Unum (universal number)

Unum также основывается на числах с плавающей точкой, но в отличии от IEEE 754 использует не только конечное множество точно представимых чисел, но и интервалы между ними. Он определяется как формат переменной длины, динамически адаптируемой к представляемому числу. Множество Unum является надмножеством чисел с плавающей точкой, $U \supset F$.

Unum — это битовая строка переменной длины, имеющая 6 полей: знаковый бит, экспоненту, мантиссу, бит неопределенности (ubit), размеры экспоненты и мантиссы [4]. Знаковый бит: определяет знак числа; экспонента — поле переменной длины, содержащее смещенную экспоненту; мантисса — поле переменной длины, содержащее мантиссу; размер экспоненты — поле, содержащее битовый размер экспоненты, а размер мантиссы — поле, содержащее битовый размер мантиссы. Если ubit = 0, то число представлено точно, т.е. является рациональным, если ubit = 1, то Unum выглядит как множество действительных чисел, представленных открытым интервалом (a, b), где a — число с плавающей точкой, записанное в первых

Рис. 1. Битовая строка IEEE 754—2008

Таблица 1

Интерпретация битовой строки IEEE 754—2008

Экспонента		**Мантисса** $f = \sum_{i=1}^{p} d_i 2^{-i}$ $(0 \le f < 1 - 2^{-p})$	
		0	**≠0**
$e = \sum_{i=0}^{q-1} e_i d^i$; $e \in (0;\ 2^q - 1)$	0	Нули со знаком ±0	Денормализованные числа $\pm f \cdot 2^{-(\text{bias} - 1)}$
	$2^q - 1$	Бесконечность ±∞	NAN $q\text{NaN}(d_1 = 0)$; $s\text{NaN}$ $(d_1 = 1$
	Другие	Нормальные числа $\pm(1 + f) \cdot 2^{e - \text{bias}}$	

трех полях Unum; b — следующее представимое число с плавающей точкой при заданных размерах экспоненты и мантиссы.

На рис. 2 изображена битовая строка в формате Unum. Размеры экспоненты и мантиссы используются для автоматического масштабирования числа бит, применяющихся для представления мантиссы и экспоненты в зависимости от результата.

Ubound — это либо одиночный Unum либо пара Unum, представляющих интервал. Закрытые границы представляются точными Unum (ubit = 0), а открытые — неточными (ubit = 1). Обозначим множество всех Ubound как JU.

Как показано в статье [5], арифметика для закрытых, открытых, полуоткрытых интервалов JU свободна от исключений, поскольку операции в JU всегда ведут в JU, в отличие от классической интервальной арифметики. Подробное описание Unum приведено в [4], а математический аппарат Unum досконально разобран в [5].

Запись вида Unum(x,y) означает, что поле «размер экспоненты» имеет x бит, а «размер мантиссы» — y бит.

В табл. 2 — 4 приведены условные обозначения и интерпретации битовой строки Unum при ubit = 0 и ubit = 1;

Основная разница между форматами IEEE 754 и Unum состоит в том, что вместо округления Unum использует открытые интервалы. Переменный формат битовой строки позволяет сократить количество бит на представление числа (в [6, 7] показана эффективность для решения задач гидродинамики). В Unum имеется специальный бит ubit, хранящий информацию о достоверности вычислений. Кроме того, отсутствуют переполнения и потери значимости. Арифметические операции в Unum определяются похожим образом, как и в IEEE 754. В Unum 1/0 = NaN, но 1/(0, mindenor) = = (1/mindenor, ∞), тогда как в IEEE 754 1/+0 = +∞.

Проведем несколько тестов для сравнения Unum и Float, для библиотеки Unum используется язык C [8].

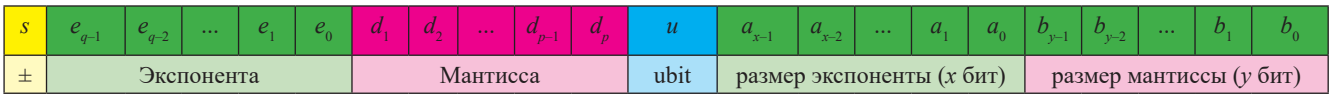

Рис. 2 Битовая строка в формате Unum

Таблица 2

Условные обозначения для Unum

$q = 1 + \sum_{i=0}^{x-1} a_i 2^i$	$e = \sum_{i=0}^{q-1} e_i 2^i$	$\max\_e = 2^{2^x - 1}$	$\text{bias} = 2^{q-1} - 1$
$p = 1 + \sum_{i=0}^{y-1} b_i 2^i$	$f = \sum_{i=0}^{p-1} d_i 2^{-i}$	$\max\_f = 1 - 2^{-2^y}$	$\text{ulp} = 2^{e - \text{bias} - p}$
$N = (1 + f) \cdot 2^{e - \text{bias}}$	$D = f \cdot 2^{-(\text{bias} - 1)}$	$\text{mindenor} = 2^{e - \text{bias} - p}$	–

Таблица 3

Интерпретация битовой строки Unum при ubit = 0

s = 0		**ubit = 0 Мантисса (f)**		
		0	**(0, max_f)**	**max_f**
Экспонента (e)	0	Нули 0	Денормализованные числа D	
	max_e	Нормальные числа N		Бесконечности ∞
	Другие	Нормальные числа N		

Таблица 4

Интерпретация битовой строки Unum при ubit = 1

s = 0		**ubit = 1 Мантисса (f)**		
		0	**(0, max_f)**	**max_f**
Экспонента (e)	0	(0, mindenor)	(D, D + ulp)	
	max_e	(N, ∞)		–
	Другие	(N, N + ulp)		

Тест 1. Потеря значимости

Тест заключается в вычислении скалярного произведения двух векторов

$$x = \left(10^a,\ 1223,\ 10^{a-1},\ 10^{a-2},\ 3,\ -10^{a-5}\right);$$
$$y = \left(10^b,\ 2,\ -10^{b+1},\ 10^b,\ 2111,\ 10^{b+3}\right)$$

с использованием чисел с плавающей запятой и Unum при разных значениях a и b. Компоненты векторов подобраны таким образом, чтобы их скалярное произведение было равно 8779, независимо от a и b [9].

Данный тест показывает проблемы с достоверностью результата скалярного произведения при разномасштабных коэффициентах в компонентах векторов. Результаты эксперимента представлены в табл. 5, 6.

Суммируя результаты эксперимента, можно утверждать, что в отличие от чисел с плавающей точкой, страдающих потерей значимости, Unum не скрывает проблем, возникающих при вычислении (не обращается в ноль при достаточно больших значениях a и b). Длина достоверного интервала при Unum(3, 4) при больших b велика, но она, по крайней мере, известна. Если полученная точность вычисления не соответствует заранее заданным требованиям, то выражение может быть вычислено заново с использованием более высокоточной арифметики.

Тест 2. Закон ассоциативности

Тест состоит в аппроксимации значения $\pi^2/6$ суммированием достаточного числа начальных значений бесконечного ряда $\sum 1/i^2$ (сумма первых 4800 слагаемых равна 1,644725755214774951145). Он выбран для демонстрации неассоциативности арифметики с плавающими числами и последствий в вычислениях.

Прямой подход ведет к неточному результату. Сумматор выравнивает порядки операндов перед выполнением сложения, при этом проводится сдвиг мантиссы, вызывающий потерю младших битов в ней. Крайний случай — когда количество значащих бит в операнде меньше, чем разница порядков операндов. В такой ситуации теряются все значащие биты мантиссы. В настоящем тесте эта ситуация возникает, поскольку все члены данного ряда положительные и квадратично убывающие, поэтому для получения более точных результатов лучше суммировать, начиная с самых малых величин [7]. Результаты представлены в табл. 7. Вне зависимости от порядка суммирования членов ряда арифметика Unum дает достоверные результаты 1.644725755214774951145.

Операция сложения неассоциативна как в IEEE 754, так и в Unum, но арифметика Unum вычисляет интервал, содержащий точный результат независимо от порядка суммирования. Как и в IEEE 754, вычисления с Unum можно улучшить, используя грамотно разработанные алгоритмы, однако, применение неоптимального алгоритма для вычисления не приведет к некорректному и недостоверному результату.

Тест 3. Система линейных алгебраических уравнений

Тест состоит в вычислении плохо обусловленных систем линейных алгебраических уравнений (СЛАУ) методом Гаусса. Этот метод вычислительно неустойчив из-за недостатков чисел с плавающей точкой, что делает его слабо пригодным для плохо обусловленных СЛАУ из-за ошибок округлений [10], но все же метод часто используется для решения СЛАУ, поэтому его

Таблица 5

Результаты эксперимента при A = 5

b	float32	float64	Unim(3, 4)	Unim(4, 6)
5	8384	8779	(–125952.270336)	8779
8	8192	8779	(–269484032,405798912)	8779
12	0,000	8781	(–1108101562368,4423816314880)	8779
15	0,000	6272	(–2260595906707456,3404087999594496)	(8777,8781)
20	0,000	0,000	(–4450277007782429327 36,297453748188566519808)	(4096; 532480)

Таблица 6

Результаты эксперимента при A = 10

b	float32	float64	Unim(3, 4)	Unim(4, 6)
5	0	8779	(–26038239232,17582522368)	8779
8	0	8764	(–35459249995776,26800595927040)	8779
12	0	0	{–43459736404125286443684916385493811 2}	(8380; 8896)
15	0	0	(–4450277007782429327 36,594907496377133039616)	(4096; 532480)
20	0	0	(–3913897341002361953111244 8,19796160296189552735813632)	(–34896609280,69793218560)

Таблица 7

Результаты эксперимента при вычислении ряда

Арифметика	От i = 1 до 4800	От i = 4800 до 1
Float32	1,644725322723	1,644725799561…
Float64	1,644725711812	1,644725755215…
Unim(3; 4)	(1,63928;1,71237))	(1,64465;1,64476)
Unim(3; 5)	(1,64472516230307; 1,64472627663053)	(1,64472575462423; 1,64472575578838586)
Unim(4; 6)	(1,6447257118119970435991; 1,644725711811997303)	(1,64472575521477495100636433126217; 1,64472575521477495133189487188923 3)

прямое применение без анализа исходных данных может приводить к неточным результатам.

Эксперимент заключался в следующем: генерируем матрицу коэффициентов таким образом, чтобы число обусловленности было велико и решаем методом Гаусса с разными типами данных.

Результаты представлены в табл. 8, 9 и показывают, что простое повышение разрядности чисел с плавающей точкой не приводит к существенному приближению к результату. При вычислениях с Unum результаты содержат вектор точного решения, но это необходимо доказать строго математически. Использование Unum позволяет отследить случай «плохих» исходных данных и дать знание о необходимости применения других методов для решения СЛАУ.

Тест 4. Вычислительная геометрия

Тест заключается в применении Unum для решения задач вычислительной геометрии, в частности, для построения выпуклой оболочки множества. Как показано в [11], наивное применение чисел с плавающей точкой в алгоритме Грэхема [12] ведет к некорректному построению выпуклой оболочки.

Для Unum алгоритм Грэхема был немного модифицирован. Изменена функция проверки ориентации векторов. Если результат функции равен интервалу $(-a, b)$, то ориентация точек не может быть определена, следовательно, выпуклая оболочка не будет построена. Если результат равен нулю при ubit = 0, то 3 точки лежат на одной прямой.

Исходный набор точек A:

$$P = \{\{4,0,\ 4,0\},\{27,6435643564,\ldots,-21,88118811881\ldots\},$$
$$\{73,4158415841,\ldots,8,86138613861\ldots\},$$
$$\{83,3663366336,\ldots,15.54455445544\ldots\}\}.$$

$$A = \begin{pmatrix} -30,4812 & 46,2324 \\ 47,7336 & -72,3824 \end{pmatrix};\ B = \begin{pmatrix} 87,792 \\ -67,331 \end{pmatrix}\ \text{Cond} = 19729,1;$$

$$\text{точный вектор решения } X = \begin{pmatrix} 6047,54 \\ 3989,07 \end{pmatrix}.$$

Таблица 8

Вычисление СЛАУ с разными типами данных

X	float32	float64	Unum(3,4)	Unum(3,5)	Unum(4,6)
X_1	6042,105	6042,602	(5444,44;6807,44)	(6047,53;6047,56)	6047,54
X_2	3985,483	3985,810	(3591,53;4490,00)	(3989,06;3989,08)	3989,07

$$A = \begin{pmatrix} 54,3154 & 8,70059 \\ -53.8930 & -8,63633 \end{pmatrix};\ B = \begin{pmatrix} -99,4942 \\ 17,7535 \end{pmatrix}\text{Cond} = 19019,2;$$

$$\text{точный вектор решения } X = \begin{pmatrix} 1963,05 \\ -12266,2 \end{pmatrix}.$$

Таблица 9

Вычисление СЛАУ с разными типами данных

X	float32	float64	Unum(3,4)	Unum(3,5)	Unum(4,6)
X_1	1964,019	1964,109	(1862,08; 2041,58)	(1963,05)	(1963,05)
X_2	–12272,27	–12272,829	(–12756,00;–11636,25)	(–12266,21;–12266,19)	–12266,2

В таблице 10 приведены результаты вычисления выпуклой оболочки, на рис. 3 показано графическое представление этого набора точек (точка p_3 немного смещена влево для лучшей визуализации оболочек).

Исходный набор точек B:

$$P = \{6,6\}\{24,0,\ 6,0\},$$
$$\{24,00000000000005,\ 24,000000000000053\}$$
$$\{24,00000000000068,\ 24,000000000000071\}$$
$$\{54,85,\ 6,0\}\{54,85000000000357,\ 61,000000000000121\}.$$

В таблице 11 представлены результаты вычисления выпуклой оболочки, на рис. 4 — графическое представление данного набора точек.

Из результатов эксперимента следует, что использование чисел с плавающей точкой приводит к некорректному построению выпуклой оболочки. Применение Unum с небольшой модификацией алгоритма позволяет отследить подобные случаи и сообщить пользователю об их появлении. Кроме того, возможно модифицировать алгоритм Грэхема с Unum, чтобы в случае невозможности построения выпуклой оболочки автоматически повышалась разрядность Unum с целью вычисления оболочки с повышенной точностью.

Заключение

Формат Unum был разработан для представления вещественных чисел. Проведенные исследования показывают, что он позволяет писать более надежные программы, чем IEEE 754. Поскольку Unum дает точную информацию о своих результатах, пользователь может легко проверить, соответствуют ли результаты вычислений требованиям к точности. Unum — интересная альтернатива IEEE 754, его надежность позволит повысить производительность программиста, а адаптивность к размеру битов — реализовать энергоэффективные арифметические устройства.

Литература

1. **Bailey D.H.** High-precision Floating-point Arithmetic in Scientific Computation // Computing in sci.&eng. 2005. V. 7. No. 3. Pp. 54—61.

2. **Barr E.T. e. a.** Automatic Detection of Floating-point Exceptions // ACM Sigplan Notices. 2013. V. 48. No. 1. Pp. 549—560.

3. **IEEE 754—2008.** Standard for Floating-point Arithmetic.

4. **Gustafson J.L.** The End of Error: Unum Computing. CRC Press, 2015.

5. **Kulisch U.** Up-to-date Interval Arithmetic: from Closed Intervals to Connected Sets of Real Numbers // Proc. 11th Intern. Conf. Parallel Proc. and Appl. Math. Krakow, 2016. Pp. 413—434.

6. **Lloyd S.** LULESH Execution with Unums in C/C++ [Электрон. ресурс] https://unum.soe.ucsc.edu/

Таблица 10

Выпуклая оболочка с разными типами данных для набора *A*

Корректный результат	Результат double	Результат Unum(3;4)	Результат Unum(4;6)
$\{p_1, p_2, p_4\}$	$\{p_1, p_2, p_3, p_4\}$	не построена	$\{p_1, p_2, p_4\}$

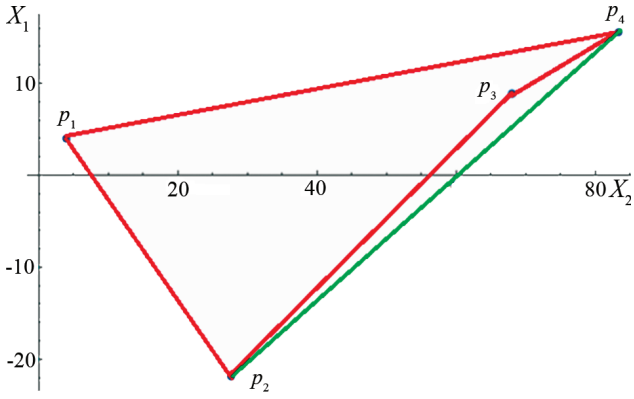

Рис. 3. Правильная выпуклая оболочка (зеленая линия) и оболочка, полученная с типом данных double (красная линия) для набора данных *A*

Таблица 11

Выпуклая оболочка с разными типами данных для набора *B*

Корректный результат	Результат double	Результат Unum(3;4)	Результат Unum(4;6)
$\{p_1, p_2, p_4\}$	$\{p_1, p_2, p_3, p_4, p_5, p_6\}$	не построена	$\{p_1, p_2, p_4\}$

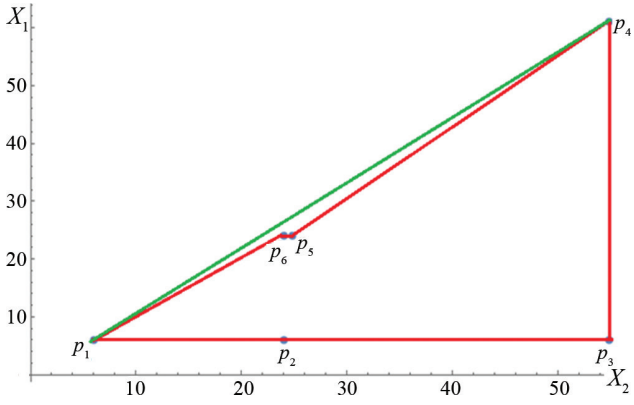

Рис. 4. Правильная выпуклая оболочка (зеленая линия) и оболочка, полученная с типом данных double (красная линия) для набора данных *B*

sites/default/files/Scott_Lloyd_UnumLULESH.pdf. (дата обращения 22.05.2017).

7. **Morancho E.** Unum: Adaptive Floating-point Arithmetic // IEEE Euromicro Conf. Digital System Design. Limassol, 2016. Pp. 651—656.

8. **Universal** Number Library [Офиц. сайт] https://github.com/LLNL/unum (дата обращения 22.05.2017).

9. **Оцоков Ш.А.** Структурно-алгоритмические методы организации высокоточных вычислений на основе теоретических обобщений в модулярной системе счисления: дис. … докт. техн. наук. М.: Изд-во МЭИ, 2010.

10. **Амосов А.А., Дубинский Ю.А., Копченова Н.В.** Вычислительные методы для инженеров. М.: Изд-во МЭИ, 2003.

11. **Kettner L. e. a.** Classroom Examples of Robustness Problems in Geometric Computations // Computational Geometry. 2008. V. 40. No. 1. Pp. 61—78.

12. **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.** Алгоритмы. Построение и анализ. Киев: Вильямс, 2005.

References

1. **Bailey D.H.** High-precision Floating-point Arithmetic in Scientific Computation. Computing in sci.&eng. 2005;7;3:54—61.

2. **Barr E.T. e. a.** Automatic Detection of Floating-point Exceptions. ACM Sigplan Notices. 2013;48;1:549—560.

3. **IEEE 754—2008.** Standard for Floating-point Arithmetic.

4. **Gustafson J.L.** The End of Error: Unum Computing. CRC Press, 2015.

5. **Kulisch U.** Up-to-date Interval Arithmetic: from Closed Intervals to Connected Sets of Real Numbers. Proc. 11th Intern. Conf. Parallel Proc. and Appl. Math. Krakow, 2016:413—434.

6. **Lloyd S.** LULESH Execution with Unums in C/C++ [Elektron. Resurs] https://unum.soe.ucsc.edu/sites/default/files/Scott_Lloyd_UnumLULESH.pdf. (Data Obrashcheniya 22.05.2017).

7. **Morancho E.** Unum: Adaptive Floating-point Arithmetic. IEEE Euromicro Conf. Digital System Design. Limassol, 2016:651—656.

8. **Universal** Number Library [Ofits. Sayt] https://github.com/LLNL/unum (Data Obrashcheniya 22.05.2017).

9. **Otsokov Sh.A.** Strukturno-algoritmicheskie Metody Organizatsii Vysokotochnykh Vychisleniy na Osnove Teoreticheskikh Obobshcheniy v Modulyarnoy Sisteme Schisleniya: Dis. … Doktora Tekhn. Nauk. M.: Izd-vo MPEI, 2010. (in Russian).

10. **Amosov A. A., Dubinskiy Yu.A., Kopchenova N.V.** Vychislitel'nye Metody Dlya Inzhenerov. M.: Izd-vo MPEI, 2003. (in Russian).

11. **Kettner L. e. a.** Classroom Examples of Robustness Problems in Geometric Computations. Computational Geometry. 2008;40; 1:61—78.

12. **Kormen T., Leyzerson Ch., Rivest R., Shtayn K.** Algoritmy. Postroenie i Analiz. Kiev: Vil'yams, 2005. (in Russian).

Сведения об авторе

Ермилов Сергей Игоревич – аспирант кафедры вычислительных машин, систем и сетей НИУ «МЭИ», e-mail: Ermilov_s@mail.ru

Information about author

Ermilov Sergey I. – Ph.D.-student of Computing Machines, Systems and Networks Dept., NRU MPEI, e-mail: Ermilov_s@mail.ru

Статья поступила в редакцию 20.06.2017