К. С. Исупов, В. С. Князьков

Арифметика многократной точности на основе систем остаточных классов

Аннотация. В работе рассматриваются алгоритмы высокоточной арифметики, основанные на использовании многомодульных систем остаточных классов (СОК) для представления мантисс чисел с плавающей точкой произвольной разрядности; порядок хранится в двоичной системе счисления. Такое представление обеспечивает большой динамический диапазон и допускает эффективное распараллеливание арифметических операций над цифрами многоразрядных мантисс по модулям СОК, что хорошо согласуется с особенностями архитектуры современных параллельных вычислительных систем. Дополнительно, в числовой формат включена атрибутивная информация, которая обеспечивает быструю оценку величины мантиссы, масштабированной относительно произведения модулей, и способствует ускоренному выполнению целого ряда проблемных для СОК немодульных процедур, таких как сравнение, контроль переполнения диапазона, округление и пр. Представлены результаты экспериментальной оценки точности и быстродействия алгоритмов, а также эффективности использования SIMD.

Ключевые слова и фразы: компьютерная арифметика, высокоточные вычисления, параллельные алгоритмы, система остаточных классов, SIMD.

Введение

С развитием параллельных аппаратных платформ и ростом масштабов вычислений одним из важнейших вопросов, требующих решения, становится обеспечение приемлемой точности арифметики с плавающей точкой. В ближайшее время суперкомпьютеры, вероятно, достигнут эксафлопсной производительности, и стоит ожидать, что получение результатов необходимой точности на таких компьютерах в машинной арифметике IEEE 754 будет непростой задачей [1]. В связи с этим в настоящее время активно развиваются методы и программные средства, обеспечивающие достоверность и воспроизводимость результатов вычислений на параллельных системах [2, 3].

Исследование выполнено при финансовой поддержке РФФИ в рамках научных проектов № 14-07-31075 мол_а и № 16-37-60003 мол_а_дк.

© К. С. Исупов, В. С. Князьков, 2016

© Вятский государственный университет, 2016

© ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ, 2016

Они основываются на алгоритмах обработки машинных чисел, позволяющих компенсировать погрешность округления. Однако возникает все больше различных задач, особенно в компьютерном моделировании, корректное решение которых может быть получено только посредством использования арифметики многократной точности, позволяющей оперировать числами произвольной разрядности [4].

Уже сейчас арифметика многократной точности востребована во многих областях, которые включают решение плохо обусловленных систем, продолжительное и / или крупномасштабное моделирование, исследование мелкомасштабных явлений, вычисление рядов, численное интегрирование и пр. [5, 6]. Конкретно, арифметика многократной точности используется при изучении динамических систем (например, исследование фрактальных свойств аттрактора Лоренца [7]), для вычисления моментов произвольных порядков в хроматографии [8], при моделировании климата [9], для вычисления амплитуд рассеяния фундаментальных частиц [10], в задачах прямой кинематики [11] и строительной механики [12], в вычислительной геометрии и во многих других. С ростом производительности суперкомпьютеров и масштабов вычислений число приложений высокоточной арифметики существенно возрастает, и имеются основания полагать, что в будущем эта тенденция будет лишь усиливаться.

Большой размер задач, необходимость их быстрого решения и особенности современных архитектур (такие, как вложенный параллелизм) определяют новые требования к высокоточному программному обеспечению. К ним относятся, в первую очередь, высокая скорость, потоковая безопасность и, что важно, возможность распараллеливания многоразрядных операций. Однако классические методы длинной арифметики [13], которые лежат в основе большинства известных пакетов (GMP, MPFR, ARPREC, QD, NTL и пр.), приводят к довольно медленным и неэффективным реализациям. Фундаментальная причина этого — возникновение цепочек переносов, из-за которых алгоритмы обработки многоразрядных мантисс становятся вычислительно сложными и не распараллеливаются. В результате высокоточные расчеты сопровождаются большими затратами времени и неэффективным использованием вычислительных ресурсов.

В качестве перспективного подхода к решению обозначенной проблемы стоит отметить разработанные Н.Н. Непейвойдой позиционные системы счисления с перекрытием [14]. Представление чисел большой

разрядности в одной из таких систем счисления позволяет исключить необходимость распространения цепочек переносов и допускает эффективное распараллеливание вычислений.

В основе нашего подхода, рассматриваемого далее, лежит идея эффективной параллельной обработки многоразрядных мантисс за счет их представления в виде разложений по модулям в системах остаточных классов (СОК) [15–18], обладающих высоким уровнем естественного параллелизма арифметических операций.

Оставшаяся часть статьи построена следующим образом. В разделе 1 описано используемое представление чисел с плавающей точкой произвольной длины (MF-формат). В разделе 2 рассмотрен метод интервально-позиционных характеристик, позволяющий решить основные проблемы СОК, связанные с высокой сложностью немодульных вычислений. Раздел 3 содержит кодировки числовых и нечисловых величин, представимых в MF-формате. В разделе 4 определены границы числового диапазона, рассматриваются вопросы однозначного определения его переполнения. Раздел 5 содержит описание базовых алгоритмов арифметики многократной точности. В разделе 6 содержатся некоторые результаты оценки эффективности предложенного подхода. В разделе 7 определены направления дальнейших исследований, и, наконец, раздел 8 содержит краткие выводы.

1. Представление данных

Используется следующее представление чисел произвольной длины. Число с плавающей точкой представляется знаком $s \in \{0, 1\}$, мантиссой в системе остаточных классов X , несмещенным порядком (экспонентой) e и интервально-позиционной характеристикой (ИПХ) мантиссы $I(X/M)$. Порядок e представлен целым машинным числом со знаком. Мантисса X представлена кортежем остатков $\langle x_1, x_2, \dots, x_n \rangle$ по модулям СОК $m_1, m_2, \dots, m_n$. Каждый i -й остаток, в свою очередь, представляется неотрицательным целым машинным числом, удовлетворяющим сравнению $x_i \equiv X \pmod{m_i}$. Мантисса X интерпретируется как целое число без знака, изменяющееся в интервале от 0 до $M - 1$, причем $M = \prod_{i=1}^n m_i$ — динамический диапазон СОК. ИПХ мантиссы представляется двумя направленно округленными числами машинной точности — нижней границей $\underline{X/M}$ и верхней границей $\overline{X/M}$ — и локализует значение мантиссы X , масштабированное относительно произведения модулей M , т.е. $\underline{X/M} \leq X/M \leq \overline{X/M}$. Описанное

числовое представление называется модулярно-позиционным форматом с плавающей точкой (сокращ. MF-формат от Modular-positional Floating-point, или МП-формат) [19, 20]. Для компактной записи чисел в таком формате используется нотация $x \rightarrow \{s, X, e, I(X/M)\}$, которая выражает следующее числовое значение:

$$(1) \quad x = (-1)^s \times \left| \sum_{i=1}^n x_i |M_i^{-1}|_{m_i} M_i \right|_M \times 2^e,$$

где $M_i = M/m_i = m_1 m_2 \cdots m_{i-1} m_{i+1} \cdots m_n$ и $|M_i^{-1}|_{m_i}$ — мультипликативная инверсия M_i по модулю m_i , представляющая собой такое целое неотрицательное число, что $|M_i^{-1}|_{m_i} M_i \equiv 1 \pmod{m_i}$. Можно доказать существование мультипликативной инверсии для каждого M_i , при условии, что все модули m_i , $i = 1, 2, \dots, n$, попарно взаимно простые, т.е. $\gcd(m_i, m_j) = 1$ для всех $i \neq j$. Сумма по модулю M в соотношении (1) выражает десятичную величину мантиссы X и выводится из китайской теоремы об остатках [17].

MF-формат обладает информационной избыточностью: ИПХ является полем формата, но, как следует из (1), не участвует в образовании значения числа. Ее предназначение здесь — дать быструю и достоверную оценку величины мантиссы при выполнении немодульных операций, являющихся крайне проблемными в СОК (см. раздел 2). Включение ИПХ в числовое представление целесообразно, по крайней мере, по двум причинам:

- во-первых, это позволяет организовать эффективную схему вычислений с предварительной оценкой операндов (см. раздел 5.1);
- во-вторых, это дает возможность использовать интервальную арифметику при выполнении модульных операций (умножение, сложение, вычитание и пр.), позволяя рассчитать ИПХ резуль-
татной мантиссы за несколько тактов.

2. Интервальная оценка в СОК

Несомненным преимуществом систем остаточных классов, по сравнению с классическими системами счисления, является их непозиционная (невзвешенная) природа, благодаря которой для таких операций, как сложение, вычитание и умножение, большой динамический диапазон разбивается на поддиапазоны пониженной разрядности путем разделения потока вычислений на несколько независимых параллельных каналов (рис. 1).

Рис. 1. Параллелизм арифметических операций в СОК

Пусть даны числа $X = \langle x_1, x_2, \dots, x_n \rangle$ и $Y = \langle y_1, y_2, \dots, y_n \rangle$ (беззнаковые) в СОК с модулями $m_1, m_2, \dots, m_n$. Тогда их сложение и умножение будут определяться в соответствии с алгоритмами 1 и 2.

АЛГОРИТМ 1. Сложение чисел в СОК

```

1: procedure RNSSUM( $X, Y$ )
2:   for  $i \leftarrow 1$  to  $n$  do
3:      $z_i = |x_i + y_i|_{m_i}$  ▷ Все  $z_i$  вычисляются параллельно
4:   end for
5:   return  $Z = \langle z_1, z_2, \dots, z_n \rangle$ 
6: end procedure

```

АЛГОРИТМ 2. Умножение чисел в СОК

```

1: procedure RNSPROD( $X, Y$ )
2:   for  $i \leftarrow 1$  to  $n$  do
3:      $z_i = |x_i \times y_i|_{m_i}$  ▷ Все  $z_i$  вычисляются параллельно
4:   end for
5:   return  $Z = \langle z_1, z_2, \dots, z_n \rangle$ 
6: end procedure

```

Представленные алгоритмы полностью параллельны и эффективны при реализации с использованием SIMD-расширений современных процессоров, легковесных GPU-потоков или аппаратных ускорителей, таких как Intel Xeon Phi. По аналогии выполняются и другие *модульные* операции — вычитание, деление без остатка и пр. Очевидно, что когда мы имеем дело с классической длиной арифметикой, добиться такой простоты и естественного параллелизма не представляется возможным (для сравнения можно рассмотреть алгоритмы

INTEGERADDITION и KARATSUBAMULTIPLY из [13]). Это обстоятельство делает весьма привлекательными методы многоразрядной арифметики на основе СОК. Однако они имеют свои недостатки, особенно в отношении таких вопросов, как выполнение *немодульных* операций (сравнение, определение знака, контроль переполнения и пр.). Эти операции требуют в той или иной форме оценки величины числа, представленного кортежем вычетов. Но такая оценка затруднена тем, что отдельные вычеты не позволяют судить о целочисленном значении, выражаемом ими. Пусть, например, в СОК с модулями 3, 5, 7 записаны числа $X = \langle 1, 0, 6 \rangle$ и $Y = \langle 2, 4, 0 \rangle$, которые нужно сравнить. Классический способ сделать это базируется на китайской теореме об остатках и состоит в вычислении двоичных представлений чисел с последующим их анализом. Так, после преобразования, $X = 55$, $Y = 14$, откуда $X > Y$. Такой способ, однако, является трудоемким, поскольку предполагает выполнение сложных операций умножения и сложения больших чисел и редукции по модулю M .

К числу других методов оценки величины в СОК относятся преобразование к смешанной системе (Mixed-Radix Conversion) [15, 17, 21], использование монотонных функций [22, 23], ранг [16], контроль четности [24], ядро (Core Function) [18] и другие. Однако часто они оказываются непрактичными, так как требуют большого объема вычислений и / или хранения внушительных подстановочных таблиц.

Для решения обозначенной проблемы мы используем подход, который базируется на интервальной оценке относительных величин в СОК. В его основе лежит замечание, что для определения результата той или иной немодульной операции в большинстве случаев не требуется знать точные значения операндов, достаточно лишь иметь информацию о диапазонах их изменения. Причем, чтобы исключить необходимость работы с большими числами в диапазоне $[0, M)$, где M — произведение всех модулей, имеет смысл оперировать масштабированными относительно M значениями [25], изменяющимися в диапазоне $[0, 1)$. В частности, возвращаясь к рассмотренному выше примеру, для сравнения чисел X и Y не обязательно преобразовывать их в двоичную систему. Достаточно знать, к примеру, что X/M находится в интервале от 0.5 до 0.7, а Y/M — в интервале от 0.1 до 0.3. Тогда можно с уверенностью сказать, что $X > Y$, так как $X/M > Y/M$. Подобную информацию дает ИПХ $I(X/M) = [\underline{X/M}, \overline{X/M}]$, которая для заданного X представляет собой интервальную функцию от остатков $x_1, x_2, \dots, x_n$, в общем случае удовлетворяющую соотношениям

$\underline{X/M} \leq X/M \leq \overline{X/M}$. Границы ИПХ хранятся в виде чисел с плавающей точкой машинной точности и вычисляются с использованием направленных округлений: $\underline{X/M}$ — с округлением вниз (toward $-\infty$), а $\overline{X/M}$ — с округлением вверх (toward $+\infty$).

В дальнейшем будем обозначать символами $\nabla, \nabla, \nabla, \nabla$ операции сложения, вычитания, умножения и деления соответственно, выполняемые с округлением вниз. Аналогично, символы $\triangle, \triangle, \triangle, \triangle$ будут использоваться для обозначения операций, выполняемых с округлением вверх. Кроме этого будем считать, что если перед некоторым выражением стоит символ ∇ или $\triangle$, то все операции в рамках этого выражения выполняются с соответствующим округлением. Вопросы выполнения направленных округлений рассмотрены в работах [26, 27].

Пусть в СОК с модулями $m_1, m_2, \dots, m_n$ дано $X = \langle x_1, x_2, \dots, x_n \rangle$. Тогда простейший метод вычислить границы его ИПХ, непосредственно следующий из китайской теоремы об остатках, определяется следующей парой соотношений, в которых $|\cdot|_1$ — дробная часть:

$$(2) \quad \underline{X/M} = \left| \nabla \sum_{i=1}^n \left(|x_i| M_i^{-1} |_{m_i} \nabla m_i \right) \right|_1,$$

$$(3) \quad \overline{X/M} = \left| \triangle \sum_{i=1}^n \left(|x_i| M_i^{-1} |_{m_i} \triangle m_i \right) \right|_1.$$

Классическое последовательное вычисление выражений (2) и (3) выполняется за время $O(n)$, при условии, что каждая операция производится в двоичной арифметике машинной точности за постоянное время. При распараллеливании эта оценка сокращается до $O(\log n)$. Кроме этого, в [28] предложен алгоритм, позволяющий вычислить ИПХ практически с любой априорно заданной точностью на всем диапазоне СОК¹ без использования многоразрядных типов данных.

Если динамический диапазон СОК сравнительно небольшой (порядка 1000 бит), то для представления границ ИПХ достаточно стандартного формата IEEE двойной точности. Если же требуется существенно больший диапазон ($M \gg 2^{1000}$), то следует использовать представление с расширенной экспонентой (extended-range floating-point). Такое представление может быть достаточно просто построено

¹В том числе и для малых чисел, для которых относительная ошибка ИПХ при вычислении с использованием выражений (2) и (3) значительно возрастает.

посредством объединения машинного целого i с обычным машинным числом с плавающей точкой f , и рассмотрения этой пары как числа

$$f \times B^i,$$

где B — константа, обозначающая основание системы счисления [29]. Алгоритмы обработки таких представлений легки в реализации и не приводят к значительным вычислительным издержкам, поскольку не предполагают увеличения точности (разрядности мантииссы f).

Важным достоинством ИПХ является возможность использования следующих интервальных соотношений:

$$(4) \quad I(X/M) + I(Y/M) = [\underline{X/M} \nabla \underline{Y/M}, \overline{X/M} \triangle \overline{Y/M}],$$

$$(5) \quad I(X/M) - I(Y/M) = [\underline{X/M} \nabla \overline{Y/M}, \overline{X/M} \triangle \underline{Y/M}],$$

$$(6) \quad I(X/M) \times I(Y/M) = [\underline{X/M} \nabla \underline{Y/M} \nabla \underline{1/M}, \overline{X/M} \triangle \overline{Y/M} \triangle \underline{1/M}],$$

$$(7) \quad I(X/M)/I(Y/M) = [\underline{X/M} \nabla \underline{1/M} \nabla \overline{Y/M}, \overline{X/M} \triangle \overline{1/M} \triangle \underline{Y/M}],$$

где $I(1/M) = [\underline{1/M}, \overline{1/M}]$ — интервальная аппроксимация константы $1/M$. Предполагается, что $\underline{Y/M}, \overline{Y/M} \neq 0$ в формуле (7).

Благодаря свойствам направленных округлений [26, раздел 2.2], если $X/M \in I(X/M), Y/M \in I(Y/M), 1/M \in I(1/M)$, результатный интервал гарантированно будет содержать математически точное значение, даже если операции над границами неточны. Иными словами, при использовании выражений (4)–(7) гарантируется, что

$$(8) \quad (X \diamond Y)/M \in I(X/M) \diamond I(Y/M)$$

где $\diamond \in \{+, -, \times, /\}$. Свойство (8) известно также как свойство интервального включения (inclusion property) [27], с поправкой на относительность выражаемых величин. Помимо этого, выражения (4)–(7) примечательны тем, что не ограничивают возможные значения результатной ИПХ диапазоном $[0, 1)$. Это позволяет, в частности, легко и непосредственно отслеживать переполнение или определять знак результата арифметических операций, несмотря на цикличность модулярного диапазона. Другие функции для оценки величины в СОК не позволяют сделать этого (по крайней мере, в такой простой форме), поскольку они вычисляются непосредственно по значениям вычетов.

ИПХ играет важную роль в MF-формате, позволяя эффективно выполнить целый ряд немодульных процедур. Например, для того чтобы определить, превышает ли мантисса X некоторый предопределенный предел L , достаточно сравнить $I(X/M)$ с заранее вычисленным интервалом $I(L/M)$: если $\underline{X/M} > \overline{L/M}$, то $X > L$, и наоборот, если

ТАБЛИЦА 1. Допустимые кодировки в MF-формате

Класс	Знак, s	Порядок, e	Знак и мантисса, $\pm X$
Числа	$\{0, 1\}$	$e_{\min} \leq e \leq e_{\max}$	$\pm \langle x_1, x_2, \dots, x_n \rangle$
$\pm\infty$	$\{0, 1\}$	$e_{\max} + 1$	$\pm \langle 0, 0, \dots, 0 \rangle$
Not-a-Number	$\{0, 1\}$	$e_{\max} + 1$	$\pm \langle x_1, x_2, \dots, x_n \rangle, \exists x_i \neq 0$

$\overline{X/M} < \underline{L/M}$, то $X < L$. Если $I(X/M)$ и $I(L/M)$ пересекаются, что однозначно определяется сравнением противоположных границ интервалов, то имеется возможность использовать другие методы оценки величины X . Более детальные правила применения ИПХ даны в [19].

3. Числовые кодировки

В MF-формате могут быть представлены конечные числа (включая знаковые нули), бесконечности со знаком и специальные нечисловые данные (Not-a-Number). Их кодировки приведены в таблице 1.

Как и в стандарте IEEE 754 [30], бесконечности интерпретируются в аффинном смысле, т.е. $-\infty < x < +\infty$, где x — любое конечное число. Основное предназначение бесконечностей состоит в том, что они позволяют получить хотя бы близкий к правильному результат вычисления в случае переполнения. Мантиссы нулей и бесконечностей представлены одинаковым кодом $\langle 0, 0, \dots, 0 \rangle$, поэтому для однозначной интерпретации необходим анализ порядка e : нулю соответствует значение $e = 0$, а бесконечности — значение $e = e_{\max} + 1$. На уровне алгоритмов числовой обработки в MF-формате реализована стандартная арифметика бесконечностей. Возникновение NaNs² является свидетельством некорректной операции или математической неопределенности результата, например, при попытке вычисления $\infty \times 0$.

4. Диапазон, переполнение и потеря значимости

Множество чисел в MF-формате однозначно определяется следующими целочисленными параметрами:

- $m_1, m_2, \dots, m_n$ — попарно взаимно простые модули СОК;
- $e_{\min}$ — минимальное значение порядка;
- $e_{\max}$ — максимальное значение порядка.

²В отличие от стандарта IEEE 754 явного разделения на signaling NaNs и quiet NaNs здесь не предусмотрено.

Обозначим символами ω и Ω наименьшее и наибольшее по абсолютному значению конечные ненулевые числа. Поскольку мантисса X может принимать целочисленные значения из интервала $[0, M - 1]$, где M — произведение всех m_i , то из (1) непосредственно следует, что $\omega = 1 \times 2^{e_{\min}}$, а $\Omega = (M - 1) \times 2^{e_{\max}}$. Полный диапазон представимых чисел определяется интервалом $[-\Omega, \Omega]$.

Ситуация, когда ненулевой результат x операции по абсолютному значению получился строго больше Ω , интерпретируется как арифметическое переполнение (overflow). В этом случае, в зависимости от режима округления, x заменяется бесконечностью или числом Ω . Ситуация, когда x по абсолютному значению строго меньше ω , интерпретируется как потеря значимости (underflow). В этом случае x заменяется нулем или числом ω , в зависимости от режима округления.

Следует отметить, что MF-формат ориентирован, прежде всего, на программную реализацию, вследствие чего предполагается, что для хранения порядка e используется отдельная переменная — машинное целое число со знаком. Это обеспечит очень большое расстояние между ω и Ω , достаточное для того, чтобы свести к минимуму вероятность переполнения или потери значимости. Например, если порядок представлен 32-битным числом и $M \approx 2^{1024}$, то $\omega \approx 10^{-646456993}$, $\Omega \approx 10^{646457300}$. Соотношение между этими предельными величинами составляет приблизительно $10^{1292914293}$. Сложно себе представить обстоятельства, при которых результат вычислений, пусть даже промежуточный, выйдет за эти пределы. Для сравнения, отношение расстояния от Земли до наиболее удаленного из известных квазара ULAS J1120+0641 к радиусу протона составляет лишь 1.43×10^{41} .

Вместе с тем, учитывая, что в общем случае для порядка e может быть отведено менее 32 бит, полностью исключить возможность переполнения или потери значимости было бы опрометчиво. Поэтому необходимы алгоритмы, позволяющие однозначно определить факт возникновения одной из этих ситуаций. Они должны учитывать, что рассматриваемое числовое представление, строго говоря, не является нормализованным (по крайней мере, в терминах IEEE 754), поскольку мантисса почти всегда будет больше 2 (базы экспоненты), а контроль единицы в старшем бите ее двоичного представления является излишне затратным. Это допускает существование для одного и того же конечного значения более чем одной кодировки³. Например, если

³Исключением является нулевое значение, для которого предусмотрен один уникальный способ записи.

СОК задана модулями 3, 5, 7, то число 0.25 может быть записано в одной из следующих форм: $\langle 1, 1, 1 \rangle \times 2^{-2}$, $\langle 2, 2, 2 \rangle \times 2^{-3}$, $\langle 1, 4, 4 \rangle \times 2^{-4}$, $\langle 2, 3, 1 \rangle \times 2^{-5}$ и т.д. Впрочем, указанная особенность не приводит к каким-либо затруднениям или ограничениям при реализации алгоритмов арифметических вычислений. Однако при выборе $e_{\min}$ и $e_{\max}$ необходимо иметь ввиду, что e может принимать значения в диапазоне от $e_{\min} - W$ до $e_{\max} + W$, где $W = \lfloor \log_2(M-1) \rfloor$. При этом число, которому соответствует этот порядок, может лежать в пределах конечного диапазона, то есть по абсолютному значению быть не меньше ω (по крайней мере, при $X = M-1$) и не превышать Ω (при $X = 1$). Как следствие, анализ на предмет переполнения или потери значимости должен выполняться на основании совместного анализа порядка e и мантиссы X . В частности, в отношении переполнения справедливы следующие положения, инвариантные к форме представления числа (кодировке) и выводимые из простых арифметических соображений:

- если $e \leq e_{\max}$, то переполнения нет при любом значении X ;
- если $e > e_{\max} + W$, то переполнение имеет место при любом X ;
- если $e = e_{\max} + i$, где i — целое число в интервале от 1 до W включительно, то факт переполнения имеет место только тогда, когда $X > \lfloor (M-1)/2^i \rfloor$.

Алгоритм проверки переполнения представлен ниже. Он возвращает 1, если переполнение возникло, и 0 — в противном случае. Алгоритм определения потери значимости CHECKUNDERFLOW задается очень схожим образом.

АЛГОРИТМ 3. Определение переполнения

```

1: procedure CHECKOVERFLOW( $X, e$ )
2:   if  $e \leq e_{\max}$  then
3:     return 0                                ▷ Переполнения нет
4:   end if
5:   if  $e > e_{\max} + W$  then
6:     return 1                                ▷ Переполнение
7:   end if
8:    $A \leftarrow \text{MIXEDRADIX}(X)$                 ▷ Преобразование из СОК в MRS
9:    $i \leftarrow e - e_{\max}$ 
10:   $B \leftarrow \mathbf{T}[i]$                         ▷ Выбор MR-кода  $\lfloor (M-1)/2^i \rfloor$ 
11:  return COMPARE( $A, B$ ) > 0 ? 1 : 0
12: end procedure

```

В строках 8–11 мантисса X сравнивается с числом $\lfloor (M-1)/2^i \rfloor$ посредством системы со смешанными основаниями (Mixed-Radix System, MRS). Мантисса преобразуется из СОК в MRS алгоритмически, а MR-представление числа $\lfloor (M-1)/2^i \rfloor$ выбирается из подстановочной таблицы **Т**. Далее выполняется обычное попарное сравнение MR-цифр, начиная со старшей. Конечно здесь можно было использовать ИПХ-метод, что во многих случаях позволило бы ускорить процедуру сравнения. Однако быстроедействие этой процедуры не имеет принципиального значения, поскольку в подавляющем большинстве случаев алгоритм CHECKOVERFLOW завершится раньше.

5. Основные алгоритмы

Некоторые из рассматриваемых далее алгоритмов ранее были представлены в работах [19, 20]. Здесь они приведены с оптимизациями и усовершенствованиями, не меняющими сути.

Чтобы не перегружать псевдокод малозначимыми операциями копирования всюду далее предполагается, что операнды передаются по значению, а не по указателю, поэтому их локальные модификации не распространяются за пределы рассматриваемой подпрограммы.

5.1. Округление

Рассматриваемое числовое представление позволяет реализовать ряд эффективных решений при обработке многоразрядных чисел, не свойственных позиционной длинной арифметике. Одно из таких решений — новая схема округления, представленная на рис. 2.

Согласно этой схеме решение о необходимости округления принимается на основании анализа мантисс операндов непосредственно *перед* арифметической операцией, а не после ее выполнения. Такая схема предпочтительна с точки зрения точности и снижения количества итераций масштабирования мантиссы, так как позволяет избежать ненужных округлений, если известно, что мантисса результата последующей операции будет находиться в допустимом диапазоне $[0, M - 1]$. Наибольший эффект от использования такой схемы достигается в аддитивных операциях, которые не приводят к существенному росту мантиссы результата. В качестве недостатка стоит отметить, что использование такой схемы в некоторой степени затрудняет априорный анализ ошибок округления. Эффективная реализация представленной схемы возможна благодаря наличию малоразрядной ИПХ в представлении числа, которая позволяет дать быструю оценку величины мантиссы непосредственно перед выполнением операции.

Алгоритм 4 выполняет округление. Он принимает на вход число $x \rightarrow \{s, X, e, I(X/M)\}$ и количество разрядов округления r , которое определяется в вызывающей подпрограмме. Для ускорения масштабирования мантиссы степенью двойки (строка 7) разработан ускоренный итерационный метод [20], рассмотрение которого выходит за рамки данной работы. Отметим лишь, что он не требует преобразования из СОК в двоичную систему счисления и допускает эффективное распараллеливание вычислений.

АЛГОРИТМ 4. Округление

```

1: procedure ROUND( $x, r$ )
2:   if  $r \leq 0$  then
3:     return  $x$ 
4:   end if
5:    $s_r \leftarrow s$ 
6:    $e_r \leftarrow e + r$ 
7:    $R \leftarrow X/2^r$  ▷ С округлением до целого
8:   if CHECKOVERFLOW( $R, e_r$ ) = 1 then ▷ Переполнение
9:     return TRAPOWERFLOW( $x$ )
10:  end if
11:  Вычисление  $I(R/M)$  ▷ Алгоритм из [28]
12:  return  $\{s_r, R, e_r, I(R/M)\}$ 
13: end procedure

```

ТАБЛИЦА 2. Спецификация операции умножения модулей чисел $|x|$ и $|y|$

$\times$	+0	number	$+\infty$	NaN
+0	+0	+0	NaN	NaN
number	+0	$ x \times y $	$+\infty$	NaN
$+\infty$	NaN	$+\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN

Метод масштабирования из [20], вычисляет число $\lfloor X/2^r \rfloor$. Поэтому при его использовании представленный Алгоритм 4 работает в режиме «round toward zero», т.е. позволяет округлять до меньшего по модулю числа. Вместе с тем, реализация режима «round to nearest» (округление до ближайшего) не вызывает затруднений. Для этого достаточно модифицировать алгоритм следующим образом: после масштабирования мантиссы следует вычислить остаток в СОК $T = \langle t_1, t_2, \dots, t_n \rangle = X - 2^r \times R$ и сравнить его с числом $C = 2^{r-1}$ (модулярный код для C может быть вычислен заранее и записан в подстановочную таблицу). Если окажется, что $T > C$, то записать $R \leftarrow R + 1$. Более того, при использовании ИПХ можно вообще не вычислять непосредственно остаток T . Достаточно рассчитать $I(T/M) = I(X/M) - 2^r \times I(R/M)$, используя интервальное соотношение (5), и сравнить полученный интервал с заранее вычисленным $I(C/M) = [\underline{C/M}, \overline{C/M}]$. При этом если на аппаратном уровне поддерживается операция Fused Multiply-Add (FMA), то для вычисления $I(T/M)$ потребуется выполнить лишь две инструкции.

Если мантисса на шаге 7 округляется до ближайшего или большего по модулю числа, то возможно переполнение. В этом случае управление передается обработчику TRAP_OVERFLOW. При округлении до меньшего по модулю переполнения, очевидно, не возникнет, так как $\lfloor X/2^r \rfloor \times 2^{e+r} \leq X \times 2^e \leq \Omega$. Также не возникнет и потери значимости, вне зависимости от используемого режима округления.

5.2. Умножение

Таблица 2 содержит спецификацию операции $|x| \times |y|$. В ней «number» обозначает ненулевое конечное число. Эта таблица соответствует стандарту IEEE 754 [26, раздел 8.4]. Знак результата вычисляется сложением по модулю два знаков сомножителей.

Алгоритм 5 выполняет умножение. Он принимает сомножители $x \rightarrow \{s_x, X, e_x, I(X/M)\}$, $y \rightarrow \{s_y, Y, e_y, I(Y/M)\}$ и возвращает произведение $z \rightarrow \{s_z, Z, e_z, I(Z/M)\}$. При умножении экстремально больших чисел может возникнуть переполнение. И напротив, если сомножители очень малы, то возможна потеря значимости. В любой из этих ситуаций управление передается соответствующему обработчику.

Благодаря использованию эффективной процедуры перемножения мантисс (RNSPROD), умножение чисел многократной точности в MF-формате, как показывают тесты, выполняется в среднем на порядок быстрее, по сравнению с традиционными алгоритмами.

АЛГОРИТМ 5. Умножение

```

1: procedure MULTIPLY( $x, y$ )
2:   Фильтрация вырожденных случаев
3:    $I(Z/M) \leftarrow I(X/M) \times I(Y/M)$ 
4:   if  $\overline{Z/M} > \frac{M-1}{M}$  then
5:      $x \leftarrow \text{CHECKANDROUND}(x)$  ▷ Округление
6:      $y \leftarrow \text{CHECKANDROUND}(y)$ 
7:      $I(Z/M) \leftarrow I(X/M) \times I(Y/M)$ 
8:   end if
9:    $s_z \leftarrow s_x \oplus s_y$ 
10:   $e_z \leftarrow e_x + e_y$ 
11:   $Z \leftarrow \text{RNSPROD}(X, Y)$  ▷ Параллельное умножение
12:  if  $\text{CHECKOVERFLOW}(Z, e_z) = 1$  then ▷ Переполнение
13:    return TRAPOWERFLOW( $z$ )
14:  end if
15:  if  $\text{CHECKUNDERFLOW}(Z, e_z) = 1$  then ▷ Потеря значимости
16:    return TRAPUNDERFLOW( $z$ )
17:  end if
18:  return  $z \rightarrow \{s_z, Z, e_z, I(Z/M)\}$ 
19: end procedure

```

В строке 2 производится анализ сомножителей на предмет соответствия одной из кодировок таблицы 1. Если хотя бы один из операндов равен нулю, бесконечности или NaN, то, в соответствии с указанной таблицей, формируется вырожденный результат операции и алгоритм завершается. При этом может быть установлен статусный флаг, свидетельствующий о возникшем исключении.

Перед умножением производится проверка необходимости округления и округление операндов (строки 3–8). Поясним суть этой процедуры. Вначале определяется возможность выхода мантиссы произведения Z за диапазон представления посредством вычисления ИПХ в соответствии с формулой (6) и сравнения ее верхней границы с нижней границей заранее вычисленного интервала, локализирующего точку $(M - 1)/M$. Если $\overline{Z/M} \leq \frac{M-1}{M}$, то Z гарантированно будет лежать в пределах диапазона (действительно, выполнение этого неравенства означает, с учетом свойства интервального включения, что точное значение Z/M не превысит $(M - 1)/M$, а следовательно, Z не превысит $M - 1$). В противном случае Z *может выйти* за пределы диапазона. Для исключения такой возможности мантиссы сомножителей округляются до $M' = \lfloor \sqrt{M - 1} \rfloor$. Количество разрядов округления r для каждого сомножителя определяется соотношением (на примере x)

$$(9) \quad X/2^r \leq M',$$

откуда получается точное выражение для r :

$$(10) \quad r = \lceil \log_2(X/M') \rceil = \lceil \log_2 X - \log_2 M' \rceil.$$

Если $r < 0$, то округление не требуется. Вычислить (10) можно несколькими способами. Простейший способ сводится к преобразованию мантиссы X из СОК в двоичную систему счисления. Однако его высокая вычислительная сложность сводит на нет смысл его использования. Более эффективный способ получается, если обе части соотношения (9) поделить на M :

$$r = \lceil \log_2(X/M) - \log_2(M'/M) \rceil.$$

Значение $\log_2(X/M)$ аппроксимируется посредством логарифмирования ИПХ $I(X/M)$; при вычислении логарифмов ИПХ используются соответствующие направленные округления:

$$\log_2 \underline{X/M} \leq \log_2(X/M) \leq \log_2 \overline{X/M}.$$

Алгоритм 6 выполняет округление числа $x \rightarrow \{s, X, e, I(X/M)\}$ перед умножением. Заранее вычисляется $I(\log_2 \alpha) = [\log_2 \underline{\alpha}, \log_2 \overline{\alpha}]$, где $\alpha = M'/M$. Процедура СНЕСК возвращает пару чисел (r, d) , где r — количество разрядов округления, а d — максимально возможное количество избыточных разрядов округления, показывающее на сколько в худшем случае вычисленное r может отличаться от точного значения, определяемого в соответствии с соотношением (10). Число d может быть использовано для контроля точности вычислений. Если

ИПХ мантииссы вычислена с достаточной точностью, то $d = 0$. Если же d превышает некоторый предел, то ИПХ требует уточнения с последующим повторным выполнением процедуры ЧЕЧЕК.

АЛГОРИТМ 6. Проверка и округление при умножении

```

1: procedure CHECKANDROUND( $x$ )
2:    $(r, d) \leftarrow \text{CHECK}(x)$   $\triangleright r$  — число разрядов округления
3:   Анализ  $d$  и оценка точности
4:   return ROUND( $x, r$ )
5: end procedure

6: procedure CHECK( $x$ )
7:    $b \leftarrow \log_2 \overline{X/M}$ 
8:    $c \leftarrow \log_2 \underline{X/M}$ 
9:    $r \leftarrow \lceil c \triangle \log_2 \underline{\alpha} \rceil$ 
10:   $r_l \leftarrow \lceil b \nabla \log_2 \overline{\alpha} \rceil$ 
11:  if  $r > 0$  then
12:     $d \leftarrow r - r_l$ 
13:    return  $(r, d)$ 
14:  else
15:    return  $(0, 0)$ 
16:  end if
17: end procedure

```

5.3. Выравнивание порядков

Если порядки операндов различны, то перед сложением или вычитанием должно производиться их выравнивание, которое целесообразно осуществлять до меньшего из порядков, что подразумевает умножение мантииссы операнда с большим порядком на $2^{|\Delta e|}$, где Δe — разность порядков. Подобный способ выравнивания используется в десятичной арифметике с плавающей точкой [31]. Однако если разность Δe по абсолютному значению велика, то при умножении может произойти переполнение диапазона. Поэтому здесь используется компенсационная схема выравнивания, при которой происходит одновременное умножение мантииссы операнда с большим порядком и масштабирование мантииссы другого операнда коэффициентом, определяемым остаточной разностью порядков, которая не может быть учтена при умножении. Данная схема представлена на рис. 3.

Рис. 3. Схема выравнивания порядков при $\Delta e > 0$

Пусть x и y – числа с мантиссами X и Y , порядками e_x и e_y , интервально-позиционными характеристиками $I(X/M)$ и $I(Y/M)$ соответственно, и пусть $\Delta e = e_x - e_y > 0$. Определим наименьшее целое r , такое, что при умножении X на $2^{\Delta e - r}$ не произойдет переполнения. Тогда 2^r будет определять коэффициент масштабирования Y . Очевидно, что r находится по аналогии с числом разрядов округления в Алгоритме 6 (поэтому используется одно и то же обозначение). Так, должно выполняться соотношение

$$\frac{X}{2^r} \leq \frac{M-1}{2^{\Delta e}},$$

где M – произведение модулей СОК. После преобразований, принимая во внимание, что r должно быть целым числом, получим

$$r \geq \lceil -\log_2(M-1) + \Delta e + \log_2 M + \log_2(X/M) \rceil.$$

Приняв допущение, что M достаточно большое для того, чтобы разницей между $\log_2(M-1)$ и $\log_2 M$ можно было пренебречь, получаем следующее точное выражение для r :

$$(11) \quad r = \lceil \log_2(X/M) + \Delta e \rceil.$$

Наконец, умножив X на $2^{\Delta e - r}$ и разделив Y на 2^r , получим операнды с одинаковым порядком $e_x - (\Delta e - r) = e_y + r$. Вместо вычисления $\log_2(X/M)$ будем вычислять $\log_2 \bar{X}/M$, где $\bar{X}/M$ – верхняя граница интервально-позиционной характеристики $I(X/M)$.

Алгоритм 7 выполняет выравнивание порядков. Предполагается, что на вход поступают конечные числа, а фильтрация нулей и специальных величин выполняется в вызывающей программе.

АЛГОРИТМ 7. Выравнивание порядков

```

1: procedure ALIGNMENT( $x, y$ )
2:    $\Delta e \leftarrow e_x - e_y$ 
3:   if  $\Delta e = 0$  then
4:     return  $x, y$ 
5:   end if
6:   if  $\Delta e < 0$  then
7:      $t \leftarrow x, x \leftarrow y, y \leftarrow t$  ▷  $x$  и  $y$  меняются местами
8:      $\Delta e \leftarrow -\Delta e$ 
9:   end if
10:  if  $\overline{X/M} < 2^{-\Delta e}$  then
11:     $r \leftarrow 0$  ▷ Округление числа  $y$  не требуется
12:  else
13:     $r \leftarrow \lceil \log_2 \overline{X/M} \rceil \triangle \Delta e$ 
14:     $c \leftarrow \log_2 \underline{Y/M}$ 
15:    if  $c < r \triangle \log_2 M$  then ▷ Сравниваем  $Y$  и  $2^r$ 
16:       $y \leftarrow (-1)^{s_y} \times 0$ 
17:      return  $x, y$ 
18:    end if
19:  end if
20:   $X \leftarrow \text{RNSPROD}(X, 2^{\Delta e - r})$ 
21:   $I(X/M) \leftarrow I(X/M) \times 2^{\Delta e - r}$ 
22:   $e_x \leftarrow e_x - (\Delta e - r)$ 
23:   $y \leftarrow \text{ROUND}(y, r)$  ▷ Округление  $y$  на  $r$  бит
24:  return  $x, y$ 
25: end procedure

```

После выполнения алгоритма будут получены ненулевые числа с одинаковыми порядками, либо одно из них будет нулем (то, чей порядок был меньшим), а другое не изменится. При округлении y (если оно необходимо) переполнения не возникнет, так как увеличенный порядок e_y , будет, по крайней мере, не больше исходного e_x . Для ускорения умножения мантииссы (строка 20) нужно записать в подстановочную таблицу модулярные коды всех натуральных степеней двойки до $2^{\lceil \log_2 M \rceil}$. Размер таблицы равен $\lceil \log_2 M \rceil \times n$.

Так же как и в Алгоритме 6, при выравнивании достаточно просто может быть реализован контроль точности вычислений. Для этого нужно вычислить $r_l = \lceil \log_2 \underline{X/M} \rceil \nabla \Delta e$ и сравнить это значение с

Таблица 3. Выбор алгоритма сложения / вычитания чисел

Операция	Знаки операндов	Выполняемый алгоритм
ADD	Совпадают	ADDITION
ADD	Различны	SUBTRATION
SUB	Совпадают	SUBTRATION
SUB	Различны	ADDITION

Таблица 4. Спецификация операции сложения модулей чисел $|x|$ (по строкам) и $|y|$ (по столбцам)

+	+0	number	$+\infty$	NaN
$+\infty$	NaN	$+\infty$	$+\infty$	NaN
+0	+0	$ y $	$+\infty$	NaN
number	$ x $	$ x + y $	$+\infty$	NaN
$+\infty$	$+\infty$	$+\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN

числом r , вычисляемым на шаге 13. Если разность $r - r_l$ оказывается неприемлемо большой, следует уточнить $I(X/M)$. Аналогичная проверка может быть реализована на шаге 14 при принятии решения об обнулении операнда с меньшим порядком.

5.4. Сложение

Знак числа представляется отдельно от мантиссы, поэтому достаточно реализовать алгоритмы сложения (ADDITION) и вычитания (SUBTRATION) чисел одинаковых знаков. Выбор необходимого алгоритма осуществляется исходя из типа выполняемой операции (ADD или SUB) и знаков операндов в соответствии с таблицей 3 [32, с. 419].

Таблица 4 содержит спецификацию операции $|x| + |y|$. Она соответствует стандарту IEEE 754 [26, раздел 8.3]. В этой таблице «number» обозначает ненулевое конечное число. Знак результата сложения определяется общим знаком слагаемых.

Алгоритм 8 выполняет сложение чисел одинаковых знаков. На вход подаются $x \rightarrow \{s_x, X, e_x, I(X/M)\}$ и $y \rightarrow \{s_y, Y, e_y, I(Y/M)\}$, в

результате возвращается сумма $z \rightarrow \{s_z, Z, e_z, I(Z/M)\}$. Алгоритм вычитания SUBTRATION очень схож с алгоритмом сложения. При этом знак разности определяется на основании ИПХ, вычисляемой по формуле (5). Заметим, что при вычитании переполнения не происходит, однако возможна потеря значимости.

АЛГОРИТМ 8. Сложение

```

1: procedure ADDITION( $x, y$ )
2:   Фильтрация вырожденных случаев
3:   ALIGNMENT( $x, y$ )
4:   if  $x = 0$  then return  $y$ 
5:   else if  $y = 0$  then return  $x$ 
6:   end if
7:    $I(Z/M) \leftarrow I(X/M) + I(Y/M)$ 
8:   if  $\overline{Z/M} > \frac{M-1}{M}$  then
9:      $x \leftarrow \text{ROUND}(x, 1)$  ▷ Округление на 1 разряд
10:     $y \leftarrow \text{ROUND}(y, 1)$ 
11:     $I(Z/M) \leftarrow I(X/M) + I(Y/M)$ 
12:   end if
13:    $s_z \leftarrow s_x$  ▷  $s_x = s_y$ 
14:    $e_z \leftarrow e_x$  ▷  $e_x = e_y$ 
15:    $Z \leftarrow \text{RNSSUM}(X, Y)$  ▷ Параллельное сложение
16:   if CHECKOVERFLOW( $Z, e_z$ ) = 1 then
17:     return TRAPOVERFLOW( $z$ )
18:   end if
19:   return  $z \rightarrow \{s_z, Z, e_z, I(Z/M)\}$ 
20: end procedure

```

6. Оценка эффективности

Рассмотренные алгоритмы реализованы в высокоточной программной библиотеке языка С MF-Library [19]. Для их оценки были выполнены следующие эксперименты:

- (1) Исследование точности вычислительных операций.
- (2) Вычисление полинома Румпа.
- (3) Вычисление рекуррентного соотношения Мюллера.
- (4) Исследование эффективности SIMD-распараллеливания.
- (5) Исследование производительности операций GEMV и GEMM.

Задачи 2 и 3 являются классическими примерами расчетов, которые даже при незначительном объеме вычислений сопровождаются возникновением катастрофических погрешностей. Такие задачи часто используются для верификации высокоточных программных средств.

Во всех экспериментах система остаточных классов для представления мантисс была задана 32 15-битными модулями, обеспечивающими диапазон $M \approx 2^{479}$. Вычисления выполнялись на машине следующей конфигурации: Intel Core i7-4702MQ / 4 Cores / 6 MB Cache / 4 GB RAM / Intel C++ Compiler XE 14.0.

6.1. Точность вычислительных операций

Мантиссы чисел в MF-формате могут принимать значения в диапазоне $[0, M - 1]$. В общем случае для предотвращения переполнения производится их округление до $\lfloor \sqrt{M - 1} \rfloor$. Таким образом, можно дать грубую априорную оценку точности p вычислений (в битах):

$$(12) \quad p = \lfloor \log_2 \lfloor \sqrt{M - 1} \rfloor \rfloor.$$

Например, если $M = 2^{479}$, то $p = 239$. Однако используемая схема округления (см. пункт 5.1) позволяет производить округление лишь *при необходимости*, для предотвращения переполнения, что обеспечивает в ряде случаев получение результата удвоенной точности. Например, если анализ интервально-позиционных характеристик показал, что мантиссы исходных операндов не превышают $\lfloor \sqrt{M - 1} \rfloor$, то при их умножении округления не будет, следовательно, будет получен точный результат (относительная ошибка равна нулю, если исходные операнды являются точными). Вероятность получения результата удвоенной точности возрастает при выполнении аддитивных операций (сложение, вычитание), когда порядки операндов примерно равны.

Вычислявшиеся в ходе эксперимента выражения и результаты эксперимента представлены в таблице 5. Исходными данными (x_i, y_i, z_i) для выражений были псевдослучайные 239-битные числа, равномерно распределенные в диапазоне $(0, 1)$, полученные с помощью алгоритма «Вихрь Мерсенна». Число слагаемых в каждой вычисляемой сумме и сомножителей в каждом вычисляемом произведении равнялось 10^6 . Каждое выражение вычислялось 100 раз. Для оценки результатов использовалась библиотека MPFR [33], реализующая корректное с точки зрения стандарта IEEE 754 округление. Использовался режим округления «round toward zero». Оценивалась максимальная апостериорная относительная ошибка: $\epsilon = \max |(x - x')/x|$, где x — точный результат, x' — вычисленный результат.

Таблица 5. Результаты экспериментальной оценки точности арифметики с представлением чисел в MF-формате

№	Выражение	Относительная ошибка ϵ	
		MF-Library (239 бит)	MPFR (239 бит)
1	$\sum x_i$	0	3.866×10^{-67}
2	$\sum 1/x_i$	2.203×10^{-74}	4.342×10^{-67}
3	$\sum x_i y_i$	2.803×10^{-139}	3.866×10^{-67}
4	$\sum z_i - x_i y_i$	0	1.133×10^{-72}
5	$\sum (1/x_i - 1/y_i)^2$	1.400×10^{-72}	5.919×10^{-67}
6	$\sum 1/(i + x_i)^2$	1.827×10^{-72}	1.022×10^{-66}
7	$\prod x_i$	6.483×10^{-67}	8.172×10^{-67}
8	$\prod x_i + y_i$	6.688×10^{-67}	1.036×10^{-66}
9	$\prod (x_i - y_i)^2$	1.487×10^{-66}	1.633×10^{-66}

Таким образом, в общем случае полученные результаты согласуются с априорной оценкой (12). Более точные результаты в задачах 1, 3 и 4 получены благодаря отмеченной выше особенности округления.

6.2. Полином Румпа

Вычислялся следующий полином (S.M. Rump, 1988 [34]):

$$(13) \quad f(a, b) = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/2b,$$

при $a = 77617.0$ и $b = 33096.0$. В качестве эталонного использовалось решение, полученное с использованием 4096-битной арифметики MPFR. Результаты представлены в таблице 6. Здесь, как и в предыдущем эксперименте, более высокая точность MF-Library по сравнению с MPFR (256 бит) объясняется используемой схемой округления.

6.3. Рекуррентное соотношение Мюллера

Другой пример расчетов, сопровождающихся потерей точности, который приводит профессор W. Кахан в работе [35] — это вычисление следующего, на первый взгляд тривиального, соотношения:

$$(14) \quad \begin{aligned} f(y, z) &= 108 - \frac{815 - 1500/z}{y}, \\ x_0 &= 4.00, \quad x_1 = 4.25, \\ x_i &= f(x_{i-1}, x_{i-2}). \end{aligned}$$

ТАБЛИЦА 6. Результаты вычисления полинома (13)

Тип данных (точность)	Результат	Верных цифр
Float (24 бит)	-6.33825×10^{29}	0
Double (53 бит)	-1.18059×10^{21}	0
MPFR (64 бит)	-5.76461×10^{17}	0
MPFR (128 бит)	0.8273960599...	39
MPFR (256 бит)	0.8273960599...	77
MF-Library (239 бит)	0.8273960599...	140
MPFR (512 бит)	0.8273960599...	155

Параметры соотношения (14) подобраны так, что при точных вычислениях $\lim_{n \rightarrow \infty} x_n = 5.0$. Однако из-за влияния ошибок округления (а конкретно, из-за невозможности точного представления $x_2 = 76/17$ в виде числа с плавающей точкой) последовательность $\{x_n\}$ отходит от верного ответа к точке $x^* = 100.0$. Номер итерации, с которой начинается такой переход, прямо пропорционален точности вычислений. Результаты эксперимента представлены на рис. 4. Графики показывают, что точность MF-Library (239 бит) сравнима с точностью MPFR (256 бит).

6.4. Эффективность SIMD-распараллеливания

SIMD-параллелизм — важная составляющая современных вычислений. В разной степени большинство из современных процессорных архитектур поддерживают SIMD-расширения или SIMD-инструкции. Производители процессоров возлагают большие надежды на этот принцип вычислений, систематически наращивая разрядность векторных регистров. В свою очередь, появляются все более совершенные команды SIMD-обработки, которые позволяют одновременно обрабатывать все большее количество пар операндов, а современные компиляторы позволяют производить автоматическую векторизацию при условии подходящих для этого структур данных. Эффективность алгоритмов с точки зрения SIMD на центральных процессорах может косвенно свидетельствовать о целесообразности реализации этих алгоритмов на других аппаратных платформах, таких как графические ускорители. Эти обстоятельства нельзя не принимать во внимание при разработке программного обеспечения и, в частности, программных библиотек

Рис. 4. Результаты вычисления соотношения (14)

многократной точности. Однако известные библиотеки, ориентированные на центральные процессоры, как правило, не задействуют SIMD-параллелизм, либо задействуют его лишь в незначительной степени, что обусловлено вполне очевидными причинами, связанными, прежде всего, с образованием множественных зависимостей по данным при обработке мантисс большой разрядности. При вычислениях в MF-формате отдельные цифры мантисс, представленных в СОК, взаимно независимы, что позволяет организовать их эффективную векторную обработку.

Рис. 5. Производительность высокоточных арифметических операций с представлением чисел в MF-формате

Для оценки эффективности SIMD-распараллеливания запуск MF-Library выполнялся в двух конфигурациях: при установленном запрете на векторизацию (прописыванием директив `#pragma novector`) и при использовании средств автоматической векторизации компилятора Intel C++. В последнем случае векторизовались циклы вычисления модулярных мантисс и ИПХ. Исследованы операции сложения (add), вычитания (sub), умножения (mult), деления (div) и умножения с накоплением (mac, $x \leftarrow x + yz$). Результаты представлены на рис. 5.

При векторизации всех исследованных операций, кроме деления, получено двукратное ускорение, что является довольно высоким показателем для данной тестовой конфигурации. При векторизации деления ускорение отсутствует. Это связано с особенностями используемого алгоритма (см. [20]), приводящими к необходимости затратного преобразования модулярных мантисс в двоичную систему счисления с применением многоразрядной позиционной арифметики. Лучшие показатели могут быть достигнуты при использовании более эффективного алгоритма деления. Можно ожидать большего ускорения, если вместо автоматической векторизации, выполняемой средствами компилятора, вручную использовать SIMD-расширения команд, а также при выполнении на системе, содержащей более широкие векторные регистры и более совершенные SIMD-команды.

Рис. 6. Время последовательного и параллельного (4 ядра) выполнения операции GEMM с точностью 239 бит

Рис. 7. Время последовательного и параллельного (4 ядра) выполнения операции GEMV с точностью 239 бит

6.5. Быстродействие BLAS

Быстродействие MF-Library исследовано на операциях обобщенного матричного умножения (GEMM, BLAS Level-3) и общего векторного умножения (GEMV, BLAS Level-2) с точностью 239 бит. Рассматривались классические последовательные, а также параллельные алгоритмы, что позволило дополнительно оценить потоковую безопасность. При выполнении GEMM ($C \leftarrow \alpha AB + \beta C$) матрицы A, B, C были плотными, их порядок n изменялся с шагом 50 в интервале от 100 до 1000. Для верификации результата вычислялась норма $\|C\|_1$. Данные эксперимента представлены на рис. 6. Среднее ускорение MF-Library по сравнению с MPFR v. 3.0.1 на операции GEMM составило 1.9 раза и 2.3 раза соответственно при последовательных и параллельных вычислениях. При распараллеливании скорость MF-Library возросла в 5.3 раза (сверхлинейное ускорение объясняется поддержкой Hyper-Threading). При выполнении GEMV ($y \leftarrow \alpha Ax + \beta y$) векторы x, y и матрица A также были плотно заполнены. Размер векторов варьировался в диапазоне от 500 до 1500 с шагом 100. Корректность операции оценивалась по норме $\|y\|_1$. Результаты экспериментов представлены на рис. 7. В данном случае по сравнению с MPFR получено ускорение 2.4 раза и 2.6 раза соответственно при последовательных и параллельных вычислениях. При распараллеливании скорость библиотеки MF-Library увеличилась в среднем в 4 раза.

Другие результаты экспериментального исследования эффективности рассматриваемого в данной работе подхода в сравнении с аналогами представлены в работе [19].

7. Направления дальнейших исследований

Рассмотренная в настоящей работе техника вычислений многократной точности появилась сравнительно недавно, что обуславливает весьма широкий перечень требующих проработки вопросов, как исключительно алгоритмических, так и по части разработки программного обеспечения. К числу наиболее приоритетных мы относим следующие.

- (1) Развитие методов вычисления и анализа интервально-позиционных характеристик для быстрого выполнения немодульных операций в СОК. Эта задача включает в себя:
 - определение отличительных, инвариантных к виду модулей и их количеству, признаков чисел в СОК, лежащих вблизи границ диапазона представления, и разработка на их основе универсального способа классификации чисел по вычетам;

- реализацию алгоритмов обработки чисел с расширенным порядком (extended-range floating-point), необходимых для корректной работы с интервально-позиционными характеристиками в больших динамических диапазонах СОК ($P > 2^{1000}$);
 - разработку параллельных алгоритмов уточненного вычисления интервально-позиционной характеристики;
 - разработку и обоснование новых ускоренных методов модулярного масштабирования, не требующих хранения больших подстановочных таблиц и не накладывающих специфических требований на масштабирующий коэффициент.
- (2) Получение строгих оценок для верхних границ ошибок округления, возникающих при выполнении основных арифметических операций. Эта задача играет важную роль в вопросах воспроизводимости и достоверности результатов вычислений.
 - (3) Оптимизация алгоритмов с целью более эффективного использования SIMD-параллелизма.
 - (4) Адаптация форматов, структур данных и алгоритмов для эффективной реализации на графических процессорах и ускорителях типа Intel Xeon Phi. Разработка соответствующих подпрограмм.
 - (5) Разработка эффективного алгоритма деления. Текущая реализация приводит к необходимости преобразования мантисс из СОК в позиционную систему счисления, вследствие чего является крайне медленной. Имеет смысл, вероятно, отказаться от прямых методов деления в пользу быстрых численных методов, таких как метод Ньютона-Рафсона или метод Гольдшмидта [36]. Их основное преимущество, помимо достаточно быстрой сходимости, состоит в том, что они не требуют трудоемкого деления мантисс в СОК, а сводятся к выполнению последовательности сложений и умножений, которые эффективно распараллеливаются.
 - (6) Расширение функциональности посредством разработки и программной реализации методов высокоточного вычисления алгебраических и трансцендентных (логарифмических и показательных, тригонометрических, гиперболических и соответствующих им обратных) функций, наиболее употребительных плотных и разреженных BLAS-операций, а также ряда численных методов высокой точности, в частности, для решения систем уравнений.
 - (7) Комплексная отладка и оптимизация кода, тестирование.

8. Заключение

Камнем преткновения на пути создания эффективной высоко-точной арифметики является обработка многоразрядных мантисс, алгоритмически сложная и приводящая к большим временным затратам. Для решения этой проблемы мы использовали системы остаточных классов (СОК), которые позволяют представить мантиссу произвольной длины в виде набора взаимно-независимых малоразрядных вычетов с возможностью их эффективной параллельной обработки.

Традиционно СОК применялись при решении целочисленных задач в таких областях, как цифровая обработка сигналов, криптография, обнаружение и исправление ошибок в кодах, обработка изображений. Использование СОК в арифметике с плавающей точкой потребовало решения ряда специфических проблем, не свойственных вычислениям с целыми числами, таких как округление, выравнивание порядков, контроль над переполнением числового диапазона. Для этих целей в числовое представление введена атрибутивная информация — интервально-позиционная характеристика, позволяющая быстро оценить величину мантиссы по без полного ее преобразования в двоичную систему. В результате получен рассмотренный в этой работе модулярно-позиционный формат с плавающей точкой — MF-формат.

Алгоритмы высокоточной арифметики в MF-формате сочетаются с особенностями современных параллельных вычислительных архитектур, позволяя эффективно использовать SIMD, многопоточный, и (при достаточно большом числе модулей СОК) многоядерный параллелизм. Кроме этого, они поддерживают арифметику бесконечностей и обработку основных исключительных ситуаций, определенных стандартом IEEE 754 применительно к двоичной арифметике с плавающей точкой. Также в этой работе показана *возможность контроля за точностью вычислений*, имеющая важное значение при выполнении априорного анализа ошибок округления вычислительных алгоритмов.

Определены приоритетные задачи дальнейших исследований. Ожидается, что результатом решения этих задач будет новое быстродействующее кроссплатформенное программное обеспечение параллельной арифметики многократной точности для гибридных вычислительных систем, эффективно задействующее доступные ресурсы параллелизма как центральных, так и графических процессоров. Кроме этого, в перспективе мы планируем рассмотреть возможности аппаратной реализации разработанных алгоритмов для FPGA.

Список литературы

- [1] S. Collange, D. Defour, S. Graillat, R. Iakymchuk. “Reproducible and Accurate Matrix Multiplication for High-Performance Computing”, *16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, SCAN’2014 (Wurzburg, Germany, 2014), pp. 42–43. ⁶¹
- [2] S. Collange, D. Defour, S. Graillat, R. Iakymchuk. “Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures”, *Parallel Computing*, **49** (2015), pp. 83–97. ⁶¹
- [3] J. Demmel, H.D. Nguyen. “Parallel Reproducible Summation”, *IEEE Transactions on Computers*, **64**:7 (2015), pp. 2060–2070. ⁶¹
- [4] A. Pescaru, E. Oanta, T. Axinte, A.D. Dascalescu. “Extended Precision Data Types for the Development of the Original Computer Aided Engineering Applications”, *IOP Conference Series: Materials Science and Engineering*, **95**:1 (2015), URL: <http://stacks.iop.org/1757-899X/95/i=1/a=012125> ⁶²
- [5] D.H. Bailey, R. Barrio, J.M. Borwein. “High-Precision Computation: Mathematical Physics and Dynamics”, *Applied Mathematics and Computation*, **218**:20 (2012), pp. 10106–10121. ⁶²
- [6] D.H. Bailey, J.M. Borwein. “High-Precision Arithmetic in Mathematical Physics”, *Mathematics*, **3** (2015), pp. 337–367. ⁶²
- [7] D. Viswanath, S. Sahutöglu. “Complex Singularities and the Lorenz Attractor”, *SIAM Review*, **52** (2010), pp. 294–314. ⁶²
- [8] S. Leweke, E. von Lieres. “Fast Arbitrary Order Moments and Arbitrary Precision Solution of the General Rate Model of Column Liquid Chromatography With Linear Isotherm”, *Computers & Chemical Engineering*, **84** (2016), pp. 350–362. ⁶²
- [9] Y. He, C. Ding. “Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications”, *The Journal of Supercomputing*, **18** (2001), pp. 259–277. ⁶²
- [10] C.F. Berger, Z. Bern, L.J. Dixon, C.F. Febres, D. Forde, H. Ita, D.A. Kosower, D. Maitre. “Automated Implementation of On-shell Methods for One-loop Amplitudes”, *Physical Review D*, **78** (2008). ⁶²
- [11] J.-P. Merlet. “On the Real-Time Calculation of the Forward Kinematics of Suspended Cable-Driven Parallel Robots”, 14th World Congress in Mechanism and Machine Science (Taipei, Taiwan, 2015). ⁶²
- [12] В.Л. Якушев, В.Н. Симбиркин, А.В. Филимонов, П.А. Новиков, И.Н. Коньшин, Г.Б. Сушко, С.А. Харченко. «Решение плохообусловленных симметричных СЛАУ для задач строительной механики параллельными итерационными методами», *Вестник ННГУ*, 2012, №4(1), с. 238–246. ⁶²

- [13] R. Brent, P. Zimmermann. *Modern Computer Arithmetic*, Cambridge University Press, New York, NY, USA, 2010, 236 p. ↑^{62,66}
- [14] Н. Н. Нелейвода, И. Н. Григорьевский, Е. П. Лилитко. «О представлении действительных чисел», *Программные системы: теория и приложения*, 5:4(22) (2014), с. 105–121, URL: http://psta.psiras.ru/read/psta2014_4_105-121.pdf ↑⁶²
- [15] N.S. Szabo, R.I. Tanaka. *Residue Arithmetic and its Application to Computer Technology*, McGraw-Hill, New York, NY, USA, 1967, 236 p. ↑^{63,66}
- [16] И. Я. Акушский, Д. И. Юдицкий. *Машинная арифметика в остаточных классах*, Сов. Радио, М., 1968, 440 с. ↑^{63,66}
- [17] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, Oxford, 2000, 490 p. ↑^{63,64,66}
- [18] A. Omondi, B. Premkumar. *Residue Number Systems: Theory and Implementation*, Imperial College Press, London, UK, 2007, 312 p. ↑^{63,66}
- [19] K. Isupov, V. Knyazkov. “A Modular-Positional Computation Technique for Multiple-Precision Floating-Point Arithmetic”, *Parallel Computing Technologies*, Lecture Notes in Computer Science, vol. **9251**, Springer International Publishing, 2015, pp. 47–61. ↑^{64,69,72,81,88}
- [20] К. С. Исупов, А. Н. Мальцев. «Способ представления чисел с плавающей точкой большой разрядности, ориентированный на параллельную обработку», *Вычислительные методы и программирование*, 15:4 (2014), с. 631–643. ↑^{64,72,73,74,86}
- [21] M. Akkal, P. Siy. “A New Mixed Radix Conversion Algorithm MRC-II”, *Journal of Systems Architecture*, **53**:9 (2007), pp. 577–586. ↑⁶⁶
- [22] G. Dimauro, S. Impedovo, G. Pirlo. “A New Technique for Fast Number Comparison in the Residue Number System”, *IEEE Transactions on Computers*, **42**:5 (1993), pp. 608–612. ↑⁶⁶
- [23] G. Pirlo, S. Impedovo. “A New Class of Monotone Functions of the Residue Number System”, *International Journal of Mathematical Models and Methods in Applied Sciences*, **7**:9 (2013), pp. 802–809. ↑⁶⁶
- [24] J.-H. Yang, C.-C. Chang, C.-Y. Chen. “A High-Speed Division Algorithm in Residue Number System Using Parity-Checking Technique”, *International Journal of Computer Mathematics*, **81**:6 (2004), pp. 775–780. ↑⁶⁶
- [25] C. Y. Hung, B. Parhami. “An Approximate Sign Detection Method for Residue Numbers and Its Application to RNS Division”, *Computers & Mathematics with Applications*, **27**:4 (1994), pp. 23–35. ↑⁶⁶
- [26] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres. *Handbook of Floating-Point Arithmetic*, Birkhauser, Boston, 2010, 572 p. ↑^{67,68,74,80}
- [27] U. Kulisch. *Computer Arithmetic and Validity — Theory, Implementation and Applications*, de Gruyter, Berlin, 2008, 409 p. ↑^{67,68}

- [28] К. С. Исупов. «Об одном алгоритме сравнения чисел в системе остаточных классов», *Вестник АГТУ. Серия «Управление, вычислительная техника и информатика»*, 2014, №3, с. 40–49. ↑^{67,73}
- [29] J. R. Hauser. “Handling Floating-Point Exceptions in Numeric Programs”, *ACM Transactions on Programming Languages and Systems*, **18**:2 (1996), pp. 139–174. ↑⁶⁸
- [30] IEEE Standard for Floating-Point Arithmetic. Introduced 2008-08-29, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2008, 70 p. ↑⁶⁹
- [31] M. F. Cowlishaw. “Decimal Floating-Point: Algorithm for Computers”, *16th IEEE Symposium on Computer Arithmetic*, ARITH-16’03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 140–147. ↑⁷⁷
- [32] M. D. Ercegovac, T. Lang, *Digital Arithmetic*, The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann, San Francisco, 2004, 709 p. ↑⁸⁰
- [33] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélicier, P. Zimmermann. “MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding”, *ACM Transactions on Mathematical Software*, **33**:2 (2007). ↑⁸²
- [34] S. M. Rump. “Algorithms for Verified Inclusions — Theory and Practice”, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, ed. R. E. Moore, Academic Press Professional, San Diego, CA, USA, 1988, pp. 109–126. ↑⁸³
- [35] W. Kahan. *How Futile are Mindless Assessments of Roundoff in Floating-Point Computation*, <https://www.cs.berkeley.edu/~wkahan/Mindless.pdf>, 2006. ↑⁸³
- [36] I. Kong, E. E. Swartzlander. “A Goldschmidt Division Method With Faster Than Quadratic Convergence”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **19**:4 (2011), pp. 696–700. ↑⁸⁹

Рекомендовал к публикации

Программный комитет

Третьего национального суперкомпьютерного форума **НСКФ-2015**

Об авторах:

Константин Сергеевич Исупов

Ведущий научный сотрудник кафедры электронных вычислительных машин Вятского государственного университета, кандидат технических наук. Область научных интересов: программное обеспечение высокопроизводительных систем, параллельные алгоритмы, высокоскоростная компьютерная арифметика, вычисления многократной точности.

e-mail:

ks_isupov@vyatsu.ru

Владимир Сергеевич Князьков

Директор научно-образовательного центра супервычислительных технологий и систем, профессор кафедры электронных вычислительных машин Вятского государственного университета, доктор технических наук. Область научных интересов: параллельные вычислительные архитектуры, высокопроизводительные вычисления, реконфигурируемые вычислительные системы.

e-mail:

knyazkov@vyatsu.ru

Пример ссылки на эту публикацию:

К. С. Исупов, В. С. Князьков. «Арифметика многократной точности на основе систем остаточных классов», *Программные системы: теория и приложения*, 2016, **7:1(28)**, с. 61–97.

URL:

http://psta.psisiras.ru/read/psta2016_1_61-97.pdf

Konstantin Isupov, Vladimir Knyazkov. *Parallel multiple-precision arithmetic based on residue number system.*

ABSTRACT. This paper deals with algorithms of multiple-precision arithmetic, based on the use of multi module residue number systems for representing of arbitrary length significands of floating-point numbers; the exponent is represented in the binary number system. Such number representation provides a large dynamic range and allows for effective parallelization of arithmetic operations on the digits of multiple-precision significands across RNS modules. This agrees well with the architectural features of modern parallel computing systems. Additionally, the attributive information which provides a fast estimation for the relative value of significand and allows you to increase the speed of executing complex non-modular operations in RNS, such as comparison, overflow control, rounding, etc., is included into the number format. Results of an experimental study on precision, performance and SIMD efficiency of multiple-precision algorithms are presented. (*In Russian*).

Key words and phrases: computer arithmetic, high-precision computations, parallel algorithms, residue number system, SIMD.

References

- [1] S. Collange, D. Defour, S. Graillat, R. Iakymchuk. “Reproducible and Accurate Matrix Multiplication for High-Performance Computing”, *16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, SCAN’2014 (Wurzburg, Germany, 2014), pp. 42–43.
- [2] S. Collange, D. Defour, S. Graillat, R. Iakymchuk. “Numerical Reproducibility for the Parallel Reduction on Multi- and Many-Core Architectures”, *Parallel Computing*, **49** (2015), pp. 83–97.
- [3] J. Demmel, H. D. Nguyen. “Parallel Reproducible Summation”, *IEEE Transactions on Computers*, **64**:7 (2015), pp. 2060–2070.
- [4] A. Pescaru, E. Oanta, T. Axinte, A. D. Dascalescu. “Extended Precision Data Types for the Development of the Original Computer Aided Engineering Applications”, *IOP Conference Series: Materials Science and Engineering*, **95**:1 (2015), URL: <http://stacks.iop.org/1757-899X/95/i=1/a=012125>
- [5] D. H. Bailey, R. Barrio, J. M. Borwein. “High-Precision Computation: Mathematical Physics and Dynamics”, *Applied Mathematics and Computation*, **218**:20 (2012), pp. 10106–10121.
- [6] D. H. Bailey, J. M. Borwein. “High-Precision Arithmetic in Mathematical Physics”, *Mathematics*, **3** (2015), pp. 337–367.
- [7] D. Viswanath, S. Sahutöglu. “Complex Singularities and the Lorenz Attractor”, *SIAM Review*, **52** (2010), pp. 294–314.
- [8] S. Leweke, E. von Lieres. “Fast Arbitrary Order Moments and Arbitrary Precision Solution of the General Rate Model of Column Liquid Chromatography With Linear Isotherm”, *Computers & Chemical Engineering*, **84** (2016), pp. 350–362.
- [9] Y. He, C. Ding. “Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications”, *The Journal of Supercomputing*, **18** (2001), pp. 259–277.

- [10] C. F. Berger, Z. Bern, L. J. Dixon, C. F. Febres, D. Forde, H. Ita, D. A. Kosower, D. Maitre. “Automated Implementation of On-shell Methods for One-loop Amplitudes”, *Physical Review D*, **78** (2008).
- [11] J.-P. Merlet. “On the Real-Time Calculation of the Forward Kinematics of Suspended Cable-Driven Parallel Robots”, 14th World Congress in Mechanism and Machine Science (Taipei, Taiwan, 2015).
- [12] V. L. Yakushev, V. N. Simbirkin, A. V. Filimonov, P. A. Novikov, I. N. Kon’shin, G. B. Sushko, S. A. Kharchenko. “Solution of ill-conditioned symmetric slae for structural mechanics problems by parallel iterative methods”, *Vestnik NNGU*, 2012, no.4(1), pp. 238–246 (in Russian).
- [13] R. Brent, P. Zimmermann. *Modern Computer Arithmetic*, Cambridge University Press, New York, NY, USA, 2010, 236 p.
- [14] N. N. Nepeyvoda, I. N. Grigorevskiy, Ye. P. Lilitko. “New representation of real numbers”, *Programmye Sistemy: Teoriya i Prilozheniya*, **5**:4 (2014), pp. 105–121 (in Russian), URL: http://psta.psir.ru/read/psta2014_4_105-121.pdf
- [15] N. S. Szabo, R. I. Tanaka. *Residue Arithmetic and its Application to Computer Technology*, McGraw-Hill, New York, NY, USA, 1967, 236 p.
- [16] I. Ya. Akushskiy, D. I. Yuditskiy. *Machine Arithmetic in Residue Classes*, Sov. Radio, M., 1968 (in Russian), 440 p.
- [17] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, Oxford, 2000, 490 p.
- [18] A. Omondi, B. Premkumar. *Residue Number Systems: Theory and Implementation*, Imperial College Press, London, UK, 2007, 312 p.
- [19] K. Isupov, V. Knyazkov. “A Modular-Positional Computation Technique for Multiple-Precision Floating-Point Arithmetic”, *Parallel Computing Technologies*, Lecture Notes in Computer Science, vol. **9251**, Springer International Publishing, 2015, pp. 47–61.
- [20] 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).
- [21] M. Akkal, P. Siy. “A New Mixed Radix Conversion Algorithm MRC-II”, *Journal of Systems Architecture*, **53**:9 (2007), pp. 577–586.
- [22] G. Dimauro, S. Impedovo, G. Pirlo. “A New Technique for Fast Number Comparison in the Residue Number System”, *IEEE Transactions on Computers*, **42**:5 (1993), pp. 608–612.
- [23] G. Pirlo, S. Impedovo. “A New Class of Monotone Functions of the Residue Number System”, *International Journal of Mathematical Models and Methods in Applied Sciences*, **7**:9 (2013), pp. 802–809.
- [24] J.-H. Yang, C.-C. Chang, C.-Y. Chen. “A High-Speed Division Algorithm in Residue Number System Using Parity-Checking Technique”, *International Journal of Computer Mathematics*, **81**:6 (2004), pp. 775–780.
- [25] C. Y. Hung, B. Parhami. “An Approximate Sign Detection Method for Residue Numbers and Its Application to RNS Division”, *Computers & Mathematics with Applications*, **27**:4 (1994), pp. 23–35.

- [26] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres. *Handbook of Floating-Point Arithmetic*, Birkhauser, Boston, 2010, 572 p.
- [27] U. Kulisch. *Computer Arithmetic and Validity — Theory, Implementation and Applications*, de Gruyter, Berlin, 2008, 409 p.
- [28] K. S. Isupov. “On an algorithm for number comparison in the residue number system”, *Vestnik AGTU. Seriya “Upravleniye, vychislitel’naya tekhnika i informatika”*, 2014, no.3, pp. 40–49 (in Russian).
- [29] J. R. Hauser. “Handling Floating-Point Exceptions in Numeric Programs”, *ACM Transactions on Programming Languages and Systems*, **18**:2 (1996), pp. 139–174.
- [30] IEEE Standard for Floating-Point Arithmetic. Introduced 2008-08-29, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2008, 70 p.
- [31] M. F. Cowlshaw. “Decimal Floating-Point: Algorism for Computers”, *16th IEEE Symposium on Computer Arithmetic*, ARITH-16’03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 140–147.
- [32] M. D. Ercegovac, T. Lang, *Digital Arithmetic*, The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann, San Francisco, 2004, 709 p.
- [33] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélicier, P. Zimmermann. “MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding”, *ACM Transactions on Mathematical Software*, **33**:2 (2007).
- [34] S. M. Rump. “Algorithms for Verified Inclusions — Theory and Practice”, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, ed. R. E. Moore, Academic Press Professional, San Diego, CA, USA, 1988, pp. 109–126.
- [35] W. Kahan. *How Futile are Mindless Assessments of Roundoff in Floating-Point Computation*, <https://www.cs.berkeley.edu/~wkahan/Mindless.pdf>, 2006.
- [36] I. Kong, E. E. Swartzlander. “A Goldschmidt Division Method With Faster Than Quadratic Convergence”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **19**:4 (2011), pp. 696–700.

Sample citation of this publication:

Konstantin Isupov, Vladimir Knyazkov. “Parallel multiple-precision arithmetic based on residue number system”, *Program systems: theory and applications*, 2016, **7**:1(28), pp. 61–97. (In Russian).

URL: http://psta.psir.ru/read/psta2016_1_61-97.pdf