УДК 004.222.3+519.6

В. С. Князьков, К. С. Исупов

Алгоритмы высокоточной арифметики с плавающей точкой 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 длины. Порядок и знак чисел хранятся и обрабатываются в двоичной системе счисления. Такое представление с одной стороны обеспечивает большой динамический диапазон, а с другой – допускает эффективное распараллеливание обработки многоразрядных мантисс по модулям СОК и хорошо согласуется с архитектурой современных параллельных аппаратных платформ. Для ускоренного выполнения немодульных операций над мантиссами (сравнение, определение знака, контроль переполнения диапазона, округление и пр.) в представление числа включена интервально-позиционная характеристика, которая обеспечивает быструю оценку величины мантиссы, масштабированной относительно произведения модулей. Рассмотрены алгоритмы округления, выравнивания порядков, сложения и умножения.

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 ближайшие десять лет вычислительные системы, вероятно, достигнут экзафлопсной производительности, и получение точных результатов на таких компьютерах в машинной арифметике IEEE-754 будет непростой задачей [1]. В связи с этим в настоящее время активно разрабатываются методы и программные средства,

Работа выполнена при поддержке РФФИ (проект № 14–07–31075–мол_а).

Вятский государственный университет, 610000, Россия, г. Киров, 2015

обеспечивающие достоверность и воспроизводимость результатов вычислений на параллельных аппаратных платформах [2, 3]. Они основываются на алгоритмах обработки машинных чисел с плавающей точкой, позволяющих компенсировать погрешность округления.

Однако существует широкий класс задач, корректное решение которых может быть получено лишь с использованием арифметики многократной точности, которая позволяет оперировать числами неограниченной разрядности (формально разрядность ограничена лишь объемом оперативной памяти вычислительной машины).

Сегодня арифметика многократной точности востребована во многих областях научных и промышленных вычислений. Такие области включают решение плохо обусловленных систем, крупные суммирования, продолжительное и / или крупномасштабное моделирование, исследование мелкомасштабных явлений, экспериментальные математические расчеты и т.д. [4, 5]. В частности, к приложениям арифметики многократной точности относятся задачи моделирования климата [6], изучение динамических систем (например, исследование фрактальных свойств аттрактора Лоренца [7]), вычисление амплитуд рассеяния фундаментальных частиц [8], вычисление интегралов Изинга [9], задачи строительной механики [10], задачи химической кинетики с одновременным присутствием медленно и быстро протекающих реакций и многие другие. С ростом масштабов вычислений и производительности суперкомпьютеров число приложений высокоточной арифметики существенно возрастает. Следует полагать, что в будущем эта тенденция будет не только сохраняться, но и 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 арифметических операций. Однако методы длинной позиционной арифметики, которые лежат в основе большинства высокоточных пакетов (GMP, MPFR, ARPREC, QD, NTL и пр.), приводят к довольно медленным и неэффективным реализациям. Причина этого – возникновение цепочек переносов, в результате которых алгоритмы обработки многоразрядных мантисс становятся вычислительно сложными и не распараллеливаются. В результате этого высокоточные расчеты приводят к большим затратам времени и неэффективному использованию вычислительных ресурсов.

В этой области многообещающими выглядят подходы, основанные на использовании непозиционных систем счисления, обладающих высоким уровнем естественного параллелизма при выполнении арифметических операций. В основе нашего подхода лежит идея эффективной параллельной обработки многоразрядных мантисс за счет их представления в виде разложений (наборов малоразрядных неотрицательных вычетов) в системе остаточных классов (СОК) [11, 12].

1. Представление данных

Используется следующее представление чисел произвольной длины [13, 14]. Число с плавающей точкой представляется знаком s, мантиссой в системе остаточных классов M, несмещенным порядком e и интервально-позиционной характеристикой мантиссы $I(M/P)$. Порядок e представлен целым машинным числом со знаком. Мантисса M представлена набором остатков $\langle m_1, m_2, \ldots, m_n \rangle$ по модулям СОК $p_1, p_2, \ldots, p_n$. Каждый остаток представляется неотрицательным целым машинным числом и определяется как $m_i \equiv M \bmod p_i$, где p_i – i-й модуль СОК. Мантисса M интерпретируется как целое число в интервале от 0 до $P-1$, причем $P = \prod_{i=1}^{n} p_i$. Интервально-позиционная характеристика (ИПХ) мантиссы представляется двумя направленно округленными числами машинной точности – нижней границей $\underline{M/P}$ и верхней границей $\overline{M/P}$. ИПХ локализует относительное (масштабированное) значение мантиссы, т.е. отношение между M и P так, что $\underline{M/P} \leq M/P \leq \overline{M/P}$. Представленное числовое представление называется модулярно-позиционным форматом с плавающей точкой (modular-positional floating-point, МП-формат). Кратко, число в таком формате записывается следующим образом: $x \to \{s, M, e, I(M/P)\}$. Десятичное значение числа определяется выражением:

$$x = (-1)^s \times \left| \sum_{i=1}^{n} m_i |P_i^{-1}|_{p_i} P_i \right|_P \times 2^e, \tag{1}$$

где $P_i = P/p_i$, $|P_i^{-1}|_{p_i}$ – мультипликативная инверсия P_i по p_i.

Включение ИПХ в числовое представление позволяет организовать эффективную схему вычислений с предварительной оценкой операндов (см. раздел 4), а также дает возможность использовать интервальную арифметику при выполнении модульных операций (умножение, сложение, вычитание и пр.), позволяя рассчитать ИПХ результатной мантиссы за несколько тактов (см. раздел 2).

2. Интервально-позиционная оценка в СОК

Как следует из (1), ИПХ $I(M/P) = \left[\underline{M/P}, \overline{M/P}\right]$ не участвует в образовании значения числа. Ее предназначение – дать быструю но достоверную оценку величины мантиссы при выполнении немодульных операций, являющихся крайне проблемными в СОК (сравнение, определение знака, контроль переполнения диапазона, округление и пр.). Основная идея состоит в том, что для выполнения этих операций в большинстве случаев не требуется знать точное двоичное значение мантиссы, а вполне достаточно лишь иметь некоторое представление (возможно грубое) о ее величине.

Например, для того, чтобы определить, превышает ли мантисса M некоторый предопределенный предел L, достаточно сравнить $I(M/P)$ с заранее вычисленным интервалом $I(L/P)$: если $\underline{M/P} > \overline{L/P}$, то $M > L$, и наоборот, если $\overline{M/P} < \underline{L/P}$, то $M < L$. Некоторые другие правила применения ИПХ в немодульных вычислениях даны в [13].

В дальнейшем будем обозначать символами $\overline{\nabla}, \nabla, \nabla, \underline{\nabla}$ операции сложения, вычитания, умножения и деления соответственно, выполняемые с округлением вниз (toward $-\infty$). Аналогично, символы $\overline{\Delta}, \Delta, \Delta, \underline{\Delta}$ будут использоваться для обозначения соответствующих операций, выполняемых с округлением вверх (toward $+\infty$). Будем также считать, что если перед групповым оператором стоят символы ∇ или Δ, то все операции выполняются с соответствующим округлением. Вопросы округления в машинной арифметике с плавающей точкой рассмотрены, например, в работах [15, 16].

Простейший способ вычислить ИПХ для мантиссы в СОК $M = \langle m_1, m_2, \ldots, m_n \rangle$ – использовать следующие соотношения:

$$\underline{M/P} = \left| \nabla \sum_{i=1}^{n} \left(\left| m_i |P_i^{-1}|_{p_i} \right|_{p_i} \underline{\nabla} \, p_i \right) \right|_1, \tag{2}$$

$$\overline{M/P} = \left| \Delta \sum_{i=1}^{n} \left(\left| m_i |P_i^{-1}|_{p_i} \right|_{p_i} \overline{\Delta} \, p_i \right) \right|_1, \tag{3}$$

Вычисление выражений (2) и (3) выполняется в двоичной арифметике с плавающей точкой машинной точности за $O(n)$ операций. При распараллеливании эта оценка сокращается до $O(\log n)$. Кроме этого, в [17] предложен алгоритм, позволяющий вычислить ИПХ с априорно заданной точностью на всем диапазоне СОК. Этот алгоритм позволяет в полной мере использовать SIMD и многопоточный

параллелизм центральных процессоров и графических ускорителей, обеспечивая тем самым быстрое получение высокоточной информации о величине мантиссы в СОК по ее остаткам без необходимости трудоемкого преобразования в двоичную систему.

Если P имеет величину порядка 2^{1000}, то границы ИПХ могут быть представлены в стандартном формате IEEE двойной точности (binary64). Когда требуется бóльшая точность, следует использовать представление с расширенной экспонентой (extended-range floating-point). Такое представление может быть построено 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 – предопределенная константа, обозначающая основание системы счисления [18]. Такое представление границ ИПХ не приведет к значительному снижению производительности, поскольку оно не предполагает увеличения точности (разрядности мантиссы f).

Важным достоинством ИПХ является возможность использования следующих интервальных соотношений:

$$I(X/P) + I(Y/P) = \left[\, \underline{X/P} \bigtriangledown \underline{Y/P},\ \overline{X/P} \bigtriangleup \overline{Y/P} \,\right], \tag{4}$$

$$I(X/P) - I(Y/P) = \left[\, \underline{X/P} \bigtriangledown \overline{Y/P},\ \overline{X/P} \bigtriangleup \underline{Y/P} \,\right], \tag{5}$$

$$I(X/P) \times I(Y/P) = \left[\underline{X/P} \bigtriangledown \underline{Y/P} \bigtriangledown \overline{1/P},\ \overline{X/P} \bigtriangleup \overline{Y/P} \bigtriangleup \underline{1/P}\right], \tag{6}$$

$$I(X/P)/I(Y/P) = \left[\underline{X/P} \bigtriangledown \underline{1/P} \bigtriangledown \overline{Y/P},\ \overline{X/P} \bigtriangleup \overline{1/P} \bigtriangleup \underline{Y/P}\right], \tag{7}$$

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

Соотношения (4)–(7) естественным образом учитывают возникающие ошибки округления, расширяя границы результатной ИПХ согласно свойству интервального включения (inclusion property) [16], т.е. гарантируется, что $X/P \diamond Y/P \in I(X/P) \diamond I(Y/P)$, где $\diamond \in \{+, -, \times, /\}$.

3. Числовые кодировки

В МП-формате можно представить конечные числа (включая знаковые нули), бесконечности со знаком и нечисловые величины (Таблица 1). Бесконечности позволяют получить хотя бы близкий к правильному результат вычисления в случае переполнения. Мантиссы нуля и бесконечностей представлены кодом $\langle 0, 0, \ldots, 0 \rangle$, поэтому для однозначной интерпретации необходим анализ порядка e: для нуля

$e = 0$, а для бесконечности $e = e_{\max} + 1$. Нечисловые величины не могут являться исходными данными арифметических операций, но могут быть получены в ходе их выполнения в случае неопределенности (например, при попытке вычисления $0/0$).

Таблица 1. Допустимые кодировки в МП-формате

Класс	Знак, s	Порядок, e	Мантисса, M
Числа	$\{0, 1\}$	$e_{\min} \le e \le e_{\max}$	$\pm \langle m_1, m_2, \ldots, m_n \rangle$
$\pm\infty$	$\{0, 1\}$	$e_{\max} + 1$	$\langle 0, 0, \ldots, 0 \rangle$
Not-a-Number	$\{0, 1\}$	$e_{\max} + 1$	$\langle m_1, m_2, \ldots, m_n \rangle$, $\exists m_i \neq 0$

4. Алгоритмы высокоточной арифметики

Рассматриваемые далее алгоритмы описаны в работах [13, 14]. Здесь они представлены с незначительными оптимизациями и исправлениями, не меняющими их сути. Также в работе [14] можно ознакомиться с алгоритмами деления и модулярного масштабирования, которые не рассматриваются здесь. Далее предполагается, что все операнды передаются по значению, а не по указателю, поэтому их локальные модификации не распространяются за пределы алгоритма.

4.1. Округление

Использование ИПХ для быстрой оценки мантиссы позволило реализовать новую схему округления, представленную на Рис. 1.

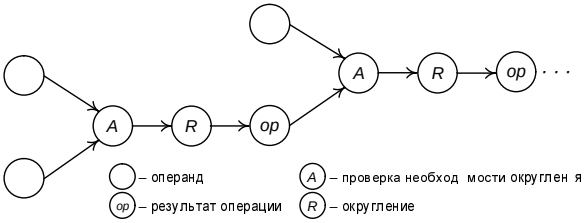

Рис. 1. Используемая схема округления

Согласно этой схеме решение о необходимости округления принимается на основании анализа ИПХ 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 она позволяет избежать

ненужных округлений, если мантисса результата последующей операции будет находиться в допустимом диапазоне $[0, P-1]$. Наибольший эффект от использования такой схемы достигается в аддитивных операциях, которые не приводят к существенному росту мантиссы результата. Эффективная реализация схемы возможна за счет наличия малоразрядной ИПХ в представлении числа, которая позволяет дать быструю оценку величины мантиссы перед выполнением операции.

Алгоритм 1 реализует операцию округления. Он принимает на вход число $x \to \{s, M, e, I(M/P)\}$ и количество разрядов округления $\mathrm{rsh}(M)$, которое определяется в вызывающем алгоритме.

Algorithm 1. Округление

```
1:  procedure ROUND(x, rsh(M))
2:      if rsh(M) ≤ 0 then                          ▷ Округление не требуется
3:          return x
4:      end if
5:      s_r ← s
6:      e_r ← e + rsh(M)
7:      if e_r > e_max then                         ▷ Возникло переполнение
8:          return OWERFLOWHANDLE(x)                ▷ Обработка исключения
9:      end if
10:     M_r ← ⌊M/2^rsh(M)⌋                          ▷ Алгоритм из [14]
11:     I(M_r/P) ← ISAC(M_r)                        ▷ Алгоритм из [17]
12:     return {s_r, M_r, e_r, I(M_r/P)}
13: end procedure
```

Все шаги Алгоритма 1, кроме модулярного масштабирования мантиссы степенью двойки (строка 10) и вычисления ИПХ (строка 11), являются элементарными. Для ускорения модулярного масштабирования разработан итерационный метод, описание которого приведено в [14]. Этот метод значительно быстрее метода дихотомии и, кроме этого, эффективно распараллеливается.

Процедура OwerflowHandle(x) выполняет обработку ошибки арифметического переполнения. По умолчанию устанавливается статусный флаг, а в качестве результата операции округления выдается кодировка бесконечности со знаком исходного операнда. В случае бинарных операций OwerflowHandle принимает на вход три аргумента – операнды и идентификатор операции.

Алгоритм 1 реализует округление в режиме “round toward zero”, т.е. до меньшего по модулю числа. Для реализации режима “round to nearest” (округление до ближайшего) достаточно модифицировать алгоритм следующим образом: после масштабированния мантиссы следует вычислить остаток в СОК $R = \langle r_1, r_2, \ldots, r_n \rangle = M - 2^{\mathrm{rsh}(M)} \times M_r$ и сравнить его с числом $C = 2^{\mathrm{rsh}(M)-1}$ (модулярный код для C может быть вычислен заранее и записан в подстановочную таблицу). Если окажется, что $R > C$, то принять $M_r = M_r + 1$. Более того, при использовании ИПХ можно вообще не вычислять остаток R. Достаточно рассчитать $I(R/P) = I(M/P) - 2^{\mathrm{rsh}(M)} \times I(M_r/P)$, используя интервальную формулу (5), и сравнить полученный интервал с заранее вычисленным $I(C/P) = \left[\underline{C/P}, \overline{C/P}\right]$. При этом если на аппаратном уровне поддерживается операция fused multiply-add (FMA), то для вычисления $I(R/P)$ потребуется выполнить лишь две машинные инструкции (и два переключения режимов округления).

4.2. Умножение

Результат умножения чисел в МП-формате определяется следующей таблицей (где x и y – конечные ненулевые числа):

	0	x	$\pm\infty$
0	0	0	NaN
y	0	$xy \vee \pm\infty$	$\pm\infty$
$\pm\infty$	NaN	$\pm\infty$	$\pm\infty$

Операция умножения реализуется Алгоритмом 2. На вход подаются числа $x \to \{s_x, M_x, e_x, I(M_x/P)\}$ и $y \to \{s_y, M_y, e_y, I(M_y/P)\}$, в результате возвращается число $z \to \{s_z, M_z, e_z, I(M_z/P)\}$.

Процедура Filtration$(x, y, \times)$ анализирует операнды на предмет соответствия одной из кодировок из Таблицы 1. Если хотя бы один из операндов равен нулю, бесконечности или NaN, то, в соответствии с указанной выше таблицей, формируется вырожденный результат операции и алгоритм завершается. При этом может быть установлен статусный флаг, свидетельствующий о возникшем исключении.

RndCheck(x) выполняет проверку необходимости округления путем оценки положения M относительно $\lfloor\sqrt{P-1}\rfloor$. Для работы процедуры необходимо заранее вычислить $I(\log_2 \alpha) = [\log_2 \underline{\alpha}, \log_2 \overline{\alpha}]$, где $\alpha = \lfloor\sqrt{P-1}\rfloor/P$ (здесь и всюду далее предполагается, что при вычислении логарифмов ИПХ используются направленные округления).

Algorithm 2. Умножение

procedure MULTIPLY(x, y)
  FILTRATION($x, y, \times$) ▷ Проверка вырожденных случаев
  $I(M_z/P) = I(M_x/P) \times I(M_y/P)$ ▷ Формула (6)
  if $\overline{M_z/P} > \overline{\frac{P-1}{P}}$ **then** ▷ Требуется округление
    $rx \leftarrow$ RNDCHECK(x)
    $ry \leftarrow$ RNDCHECK(y)
    $x \leftarrow$ ROUND(x, rx)
    $y \leftarrow$ ROUND(y, ry)
    $I(M_z/P) = I(M_x/P) \times I(M_y/P)$
  end if
  $s_z \leftarrow s_x \oplus s_y$
  $e_z \leftarrow e_x + e_y$
  if $e_z > e_{\max}$ **then** ▷ Возникло переполнение
    return OWERFLOWHANDLE($x, y, \times$)
  end if
  $M_z \leftarrow$ MODULARMULTIPLY(M_x, M_y)
  return $z \to \{s_z, M_z, e_z, I(M_z/P)\}$
end procedure

procedure RNDCHECK(x) ▷ Проверка необходимости округления
  $l \leftarrow \log_2 \underline{M/P}$
  $u \leftarrow \log_2 \overline{M/P}$
  $\mathrm{rsh}(M) \leftarrow \max\left\{\lceil l \bigtriangledown \log_2 \overline{\alpha}\rceil, \lceil u \bigtriangleup \log_2 \underline{\alpha}\rceil, 0\right\}$
  return $\mathrm{rsh}(M)$ ▷ Количество разрядов округления
end procedure

procedure MODULARMULTIPLY(X, Y) ▷ Умножение в СОК
  for $i \leftarrow 1, n$ **do**
    $z_i = |x_i \cdot y_i|_{p_i}$ ▷ Все z_i могут вычисляться параллельно
  end for
  return $Z = \langle z_1, z_2, \ldots, z_n \rangle$
end procedure

Процедура MODULARMULTIPLY(X, Y) производит умножение мантисс в СОК. Она полностью параллельна и может быть эффективно реализована с использованием SIMD-расширений центральных процессоров и Intel Xeon Phi, либо легковесными потоками GPU. В резуль-

тате этого, как показано в [13], высокоточное умножение выполняется значительно быстрее, по сравнению с традиционными методами.

При умножении может возникнуть переполнение на этапе округления (см. Алгоритм 1) и при суммировании порядков. В этом случае управление передается процедуре OwerflowHandle$(x, y, \times)$, которая по умолчанию возвращает в качестве результата бесконечность со знаком, равными сумме по модулю два (XOR) знаков сомножителей.

4.3. Выравнивание порядков

При сложении и вычитании разномасштабных величин существенную роль играет операция выравнивания порядков. В двоичной системе выравнивание осуществляется до большего порядка. Так, если x и y – двоичные числа с мантиссами m_x и m_y и порядками e_x и e_y, причем $\Delta e = e_x - e_y > 0$, то порядки выравниваются до e_x. Для этого мантисса m_y сдвигается вправо (в сторону уменьшения) на Δe разрядов, что эквивалентно ее масштабированию коэффициентом $2^{\Delta e}$.

В МП-формате выравнивать порядки до большего нецелессообразно по причине более высокой трудоемкости масштабирования мантиссы в СОК степенью двойки (если только не известно, что мантисса делится нацело на эту степень двойки). Целесообразно выравнивать порядки до меньшего, умножатя мантиссу операнда с бóльшим порядком на Δe-ю степень двойки. Это позволит избежать потери точности в процессе выравнивания и ускорит его, так как умножение в СОК выполняется быстро и распараллеливается по модулям. Однако если разность порядков велика, то при умножении может произойти переполнение диапазона. Поэтому в качестве компромиссного варианта предлагается компенсационная схема выравнивания, при которой происходит одновременное увеличение мантиссы операнда с бóльшим порядком и уменьшение мантиссы другого операнда (Рис. 2).

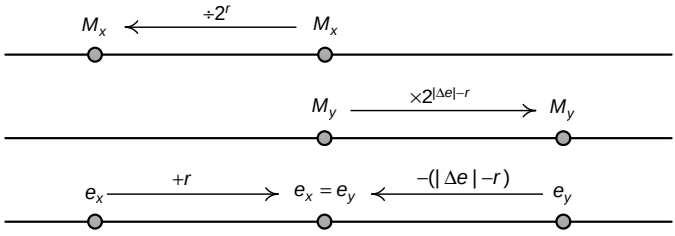

Рис. 2. Схема выравнивания порядков

Пусть x и y – числа в МП-формате с мантиссами M_x и M_y, порядками e_x и e_y, интервально-позиционными характеристиками $I(M_x/P)$ и $I(M_y/P)$ соответственно. Предположим $\Delta e = e_x - e_y < 0$ и определим наименьшее целое r, такое, что при умножении M_y на $2^{|\Delta e|-r}$ не произойдет переполнения. Тогда 2^r будет определять коэффициент масштабирования M_x. Значение r находится из условия:

$$\frac{M_y}{2^r} \le \frac{P-1}{2^{|\Delta e|}},$$

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

$$r \ge \lceil -\log_2(P-1) + |\Delta e| + \log_2 P + \log_2(M_y/P) \rceil.$$

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

$$r = \lceil \log_2(M_y/P) + |\Delta e| \rceil.$$

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

Алгоритм 3 реализует операцию выравнивания порядков. Предполагается, что на вход поступают конечные числа, а фильтрация нулей и специальных величин выполняется в вызывающей программе. 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 числа x (если оно необходимо) переполнения не возникнет, поскольку увеличенный порядок e_x, будет, по крайней мере, не больше чем e_y.

В процессе выравнивания происходит умножение мантиссы операнда с бóльшим порядком на некоторую степень двойки (строка 19), равную либо меньшую модулю разности порядков. Для ускорения этой операции следует заранее вычислить и записать в подстановочную таблицу модулярные коды всех натуральных степеней двойки вплоть до $2^{\lfloor \log_2 P \rfloor}$. Размер этой таблицы составит $\lfloor \log_2 P \rfloor \times n$ слов. Например, если $P \approx 2^{1024}$, $n = 128$, и каждый модуль представлен 32-битным числом, то размер таблицы будет равен 512 Кбайт.

Algorithm 3. Выравнивание порядков

```
1: procedure ExpAlignment(x, y)
2:     Δe = e_x − e_y
3:     if Δe = 0 then
4:         return x, y
5:     end if
6:     if Δe > 0 then
7:         Interchange(x, y)                    ▷ x и y меняются местами
8:     end if
9:     if I(M_y/P) < 2^Δe then
10:        r ← 0                    ▷ Округление числа x не требуется
11:    else
12:        I(M_y/P) ← ISAC(M_y)                 ▷ Алгоритм из [17]
13:        r ← ⌈log_2 overline(M_y/P) + |Δe|⌉
14:        if log_2 overline(M_x/P) < r − log_2 P then   ▷ Сравниваем M_x и 2^r
15:            x ← ±0
16:            return x, y
17:        end if
18:    end if
19:    M_y ← ModularMultiply(M_y, 2^|Δe|−r)
20:    I(M_y/P) ← I(M_y/P) × 2^|Δe|−r
21:    e_y ← e_y − |Δe| + r
22:    x ← Round(x, r)                  ▷ Округление x на r бит
23:    return x, y
24: end procedure
```

4.4. Сложение

Результат сложения двух чисел одинаковых знаков в МП-формате определяется следующей таблицей:

	0	x	$\pm\infty$
0	0	x	$\pm\infty$
y	y	$(x+y) \vee \pm\infty$	$\pm\infty$
$\pm\infty$	$\pm\infty$	$\pm\infty$	$\pm\infty$

Операция сложения реализуется Алгоритмом 4. На вход подаются числа $x \to \{s_x, M_x, e_x, I(M_x/P)\}$ и $y \to \{s_y, M_y, e_y, I(M_y/P)\}$, в результате возвращается число $z \to \{s_z, M_z, e_z, I(M_z/P)\}$.

Algorithm 4. Сложение

procedure Addition(x, y)
  Filtration$(x, y, +)$ ▷ Проверка вырожденных случаев
  ExpAlignment(x, y) ▷ Выравнивание порядков
  if $x = 0$ **then**
    return y
  else if $y = 0$ **then**
    return x
  end if
  $e_z \leftarrow e_x$ ▷ После выравнивания $e_x = e_y$
  $I(M_z/P) = I(M_x/P) + I(M_y/P)$ ▷ Формула (4)
  if $\overline{M_z/P} > \frac{P-1}{P}$ **then** ▷ Необходимо округление
    if $e_z = \overline{e_{\max}}$ **then** ▷ Возникнет переполнение
      return OwerflowHandle$(x, y, +)$
    else
      $x \leftarrow$ Round$(x, 1)$ ▷ Округляем на 1 разряд
      $y \leftarrow$ Round$(y, 1)$ ▷ Округляем на 1 разряд
      $I(M_z/P) \leftarrow I(M_z/P)/2$
      $e_z \leftarrow e_z + 1$
    end if
  end if
  $s_z \leftarrow s_x$ ▷ Предполагается, что $s_x = s_y$
  $M_z \leftarrow$ ModularAdd(M_x, M_y)
  return $z \rightarrow \{s_z, M_z, e_z, I(M_z/P)\}$
end procedure

procedure ModularAdd(X, Y) ▷ Сложение в СОК
  for $i \leftarrow 1, n$ **do**
    $z_i = |x_i + y_i|_{p_i}$ ▷ Все z_i могут вычисляться параллельно
  end for
  return $Z = \langle z_1, z_2, \ldots, z_n \rangle$
end procedure

Процедура ModularAdd(X, Y) производит сложение мантисс в СОК. Она, точно также как и ModularMultiply(X, Y), полностью параллельна и не требует какого-либо распространения переносов, поэтому может быть эффективно реализована на современных вычислительных архитектурах.

5. Заключение

Мы рассмотрели формат с плавающей точкой, базирующийся на использовании многомодульных систем остаточных классов для представления многоразрядных мантисс чисел, а также алгоритмы сложения, умножения, округления и выравнивания порядков в этом формате. Данные алгоритмы хорошо сочетаются с особенностями современных вычислительных архитектур. Они позволяют эффективно использовать SIMD, многопоточный, и (при использовании очень большого числа модулей) многоядерный параллелизм.

Рассмотренные алгоритмы, а также ряд других, реализованы в библиотеке MF-Library. В настоящее время производится отладка и расширение функциональности ее версии для центральных процессоров. При экспериментальной оценке эффективности библиотеки получены достаточно оптимистичные результаты [**13**, **19**]. Также начата работа по созданию версии библиотеки для систем с графическими ускорителями. В перспективе мы планируем адаптировать нашу библиотеку под Intel Xeon Phi и рассмотреть возможности реализации разработанных алгоритмов на FPGA.

Список литературы

[1] Collange S., Defour D., Graillat S., Iakymchuk R. Reproducible and Accurate Matrix Multiplication for High-Performance Computing // 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2014), September 21–26, 2014, Würzburg, Germany, Book of Abstracts. – 2014. – P. 42–43. ↑ 1.

[2] Collange S., Defour D., Graillat S., Iakymchuk R. Numerical reproducibility for the parallel reduction on multi- and many-core architectures // Parallel Computing. – 2015. – Vol. 49. – P. 83–97. ↑ 2.

[3] Demmel J., Hong Diep Nguyen. Parallel reproducible summation // IEEE Transactions on Computers. – 2015. – Vol. 64, Issue 7. – P. 2060–2070. ↑ 2.

[4] Bailey D.H., Barrio R., Borwein J.M. High-precision computation: Mathematical physics and dynamics // Appl. Math. Comput. – 2012. – Vol. 218, Issue 20. – P. 10106–10121. ↑ 2.

[5] Bailey D.H., Borwein J.M. High-precision arithmetic in mathematical physics // Mathematics. – 2015. – Vol. 3. – P. 337–367. ↑ 2.

[6] He Y., Ding C. Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications // J. Supercomput. – 2001. – Vol. 18. – P. 259—277. ↑ 2.

[7] Viswanath D., Sahutŏglu S. Complex singularities and the Lorenz attractor // SIAM Rev. – 2010. – Vol. 52. – P. 294–314. ↑ 2.

[8] Berger C.F., Bern Z., Dixon L.J., Febres C.F., Forde D., Ita H., Kosower D.A., Maitre D. Automated implementation of on-shell methods for one-loop amplitudes // Phys. Rev. D. – 2008. – Vol. 78, 036003. ↑ 2.

[9] Bailey D.H., Borwein J.M., Crandall R.E. Integrals of the Ising class // J. Physics A: Math. Gen. – 2006. – Vol. 39. – P. 12271–12302. ↑ 2.

[10] Якушев В.Л. Решение плохообусловленных симметричных СЛАУ для задач строительной механики параллельными итерационными методами / В.Л. Якушев, [и др.] // Вестник ННГУ. – 2012. – № 4(1). – С. 238–246. ↑ 2.

[11] Szabo N.S., Tanaka R.I. Residue Arithmetic and its Application to Computer Technology. – New York, NY, USA : McGraw-Hill, 1967. ↑ 3.

[12] Акушский И. Я., Юдицкий Д.И. Машинная арифметика в остаточных классах. – М. : Сов. Радио, 1968. – 440 с. ↑ 3.

[13] Isupov K., Knyazkov V. A Modular-Positional Computation Technique for Multiple-Precision Floating-Point Arithmetic // Parallel Computing Technologies, ser. LNCS. – Cham, Switzerland : Springer International Publishing. – 2015. – Vol. 9251. – P. 47—61. ↑ 3, 4, 6, 10, 14.

[14] Исупов К.С., Мальцев А.Н. Способ представления чисел с плавающей точкой большой разрядности, ориентированный на параллельную обработку // Выч. методы и программирование. – 2014. – Т. 15, № 4. – С. 631–643. ↑ 3, 6, 7.

[15] Muller J.-M. Handbook of Floating-Point Arithmetic // J.-M. Muller [et al.] — Birkhäuser Boston, 2010. ↑ 4.

[16] Kulisch U. Computer Arithmetic and Validity – Theory, Implementation and Applications. – de Gruyter, Berlin, 2008. ↑ 4, 5.

[17] Исупов К.С. Об одном алгоритме сравнения чисел в системе остаточных классов // Вестник АГТУ. Серия «Управление, вычислительная техника и информатика». – 2014. – № 3. – С. 40–49. ↑ 4, 7, 12.

[18] Hauser J.R. Handling Floating-Point Exceptions in Numeric Programs // ACM Transactions on Programming Languages and Systems. – 1996. – Vol. 18, No. 2. – P. 139–174. ↑ 5.

[19] Исупов К.С., Князьков В.С. Библиотека параллельной арифметики многократной точности для высокопроизводительных систем // Суперкомпьютерные дни в России: Тр. Международной конференции (28–29 сентября 2015 г., г. Москва). – CEUR Workshop Proceedings : CEUR-WS.org, 2015. – Т. 1482. – С. 110–121. ↑ 14.