УДК 004.315

С. И. Аряшев, канд. техн. наук, зав. отд., **С. Г. Бобков,** д-р техн. наук, зам. дир-ра,
П. С. Зубковский, зав. сектором, **Е. В. Ивасюк,** науч. сотр.,
Научно-исследовательский институт системных исследований РАН, e-mail: zubkovsky@cs.niisi.ras.ru

Реализация блока компенсированного суммирования для повышения точности аппаратных вычислений

Приведено обоснование необходимости разработки арифметических блоков, реализующих операции компенсированного сложения и умножения для ускорения трансцендентных функций в инженерных расчетах газовой и термодинамики, а также описана предложенная авторами модернизация блока вещественного суммирования для выполнения макрофункций компенсированного сложения в библиотеке CRLIBM.

Ключевые слова: *компенсированное сложение, трансцендентные функции, аппаратное ускорение*

Введение

При выполнении сложных инженерных расчетов, таких как, например, моделирование химической и газодинамической фаз процессов горения и детонации, требования к арифметической производительности микропроцессора не ограничиваются только основными арифметическими операциями — сложением и умножением. Эффективная аппаратная поддержка векторных, матричных, комплексных и трансцендентных операций над вещественными числами одинарной и двойной точности оказывает большое влияние на производительность микропроцессора.

Результаты профилирования [1] программного кода одного из вариантов реализации задачи горения были получены для выделения функций, вычисление которых с применением стандартного блока вещественной арифметики занимает наибольшее время. От реализации аппаратной поддержки этих функций можно получить наибольший выигрыш в производительности всей задачи в целом. Помимо матрично-векторных операций, в первых строчках результатов профилирования оказались функции вычисления экспоненты и логарифма для вещественных чисел двойной точности.

Полная аппаратная поддержка трансцендентных функций в микропроцессоре возможна в том случае, если допускается большая погрешность вычислений и формат вещественной арифметики одинарной точности. Если требуется получение результата в формате *double* с погрешностью не более 0.5ulp, то целесообразно прибегнуть к программно-аппаратному подходу, так как затраты для полностью аппаратной реализации оказываются чрезвычайно велики.

Несмотря на то что стандарт IEEE754 ничего не декларирует об элементарных (трансцендентных) функциях, существуют программные библиотеки, обеспечивающие вычисление таких функций с ошибкой не более 0.5ulp мантиссы формата *double*. Это гарантирует стабильность результатов программного обеспечения, позволяя переносить выполняемый код с платформы на платформу. Следовательно, в качестве альтернативы традиционному аппаратному ускорению вычисления элементарных функций существует способ обеспечения аппаратного ускорения части алгоритмических "примитивов", задействованных в программной библиотеке. Речь идет о компенсированных функциях сложения и умножения, которые являются основным строительным элементом библиотек двойной/тройной внутренней точности (*double-double*, *triple-double*).

Например, библиотека CRLIBM [2] реализует вычисление элементарных функций с корректным округлением. Библиотека написана на языке C, поддерживает точность промежуточных вычислений на уровне *double-double/triple-double*, а также обеспечивает поддержку аппаратных операций *FMA* (*Fused Multiply-Add*) [3]. Необходимость оперирования с переменными, представленными двумя/тремя числами типа *double*, делает необходимым использование компенсированных операций умножения и сложения. Простейшими являются операции сложения и умножения (*Add12*, *Mul12*, *Add12Cond*, *Mul12Cond*), получающие на вход по два числа формата *double*, результатом операций является также пара чисел, сумма которых дает

точный результат. Более сложные компенсированные операции, складывающие и умножающие числа в формате *double-double/triple-double*, основаны на операциях *Add12*, *Mul12*, *Add12Cond*, *Mul12Cond*, следовательно, последние являются основным строительным элементом библиотеки, от быстродействия которых зависит скорость вычисления.

Разработка аппаратных блоков вещественной арифметики, реализующих "нижний" уровень компенсированных операций, позволит ускорить выполнение функций библиотеки компенсированных вычислений путем переноса вычислений со стандартного блока *FPU* на специализированный вычислитель. Ниже приведено описание предложенной авторами модернизации блока вещественного сумматора, позволяющей реализовать компенсированное сложение *Add12*, *Add12Cond* посредством одной операции.

Вычисление ошибки округления

Так как алгоритм вещественного сложения/вычитания включает этап округления, результат операции содержит ошибку, которая равна разности округленного числа и точного. Согласно стандарту IEEE754 значение ошибки не превышает 0,5 *ulp* мантиссы в режиме округления *RNE* (к ближайшему/четному) и 1 *ulp* в остальных режимах (к плюс/минус бесконечности, к нулю). Известно, что в режиме округления *RNE* ошибка представима вещественным числом в формате, в котором проводится основная операция [4].

Данная ошибка может быть определена как программным путем, так и полностью аппаратным. Программный подход применим в случае, если ничего неизвестно про внутреннее устройство модуля сложения/вычитания. Существуют два компенсированных алгоритма суммирования пары вещественных чисел. Первый алгоритм связан со сравнением чисел и требует сортировки [5]:

$$\begin{gathered} function[x,\ y] = FastTwoSum(a,\ b), \\ x = fl(a + b),\ y = err \\ \begin{cases} |a| > |b|\colon x = fl(a + b),\ y = fl(fl(a - x) + b) \\ |a| < |b|\colon x = fl(a + b),\ y = fl(fl(b - x) + a). \end{cases} \end{gathered} \tag{1}$$

Операций в цепочке всего три, однако требуется сравнение, которое может привести к значительной потере производительности вследствие ошибок предсказания условных переходов.

Второй алгоритм [5] не требует сравнения исходных операндов, однако включает большее число операций (5 операций в цепочке, 6 всего):

$$\begin{gathered} function[x,\ y] = TwoSum\{a,\ b), \\ x = fl\{a + b),\ y = err \\ z = fl(x - a), \\ y = fl(fl(a - fl(x - z)) + fl(b - z)). \end{gathered} \tag{2}$$

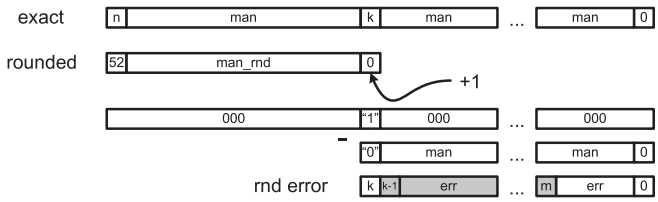

Рис. 1. Иллюстрация процесса округления и вычисления ошибки округления

Таким образом, обеспечив микропроцессор аппаратными блоками, вычисляющими погрешность округления простых операций типа сложения и умножения, можно получить существенный выигрыш по быстродействию на задачах, содержащих компенсированные операции.

На практике чаще применяется второй алгоритм, потому что он не содержит в своем составе операций сравнения, которые могут быть преобразованы в команды условного перехода, несмотря на то что его задержка составляет пять вещественных операций против трех у первого алгоритма.

Этап округления результата вещественной операции сложения обеспечивает выделение необходимой разрядности точной мантиссы результата, ее инкрементирование в зависимости от действующего режима округления (*rounding mode*) и от присутствия значащих битов в отбрасываемой части (*sticky bit*). На рис. 1 проиллюстрирован процесс округления точного результата и последующего вычисления ошибки округления. Часть [*n*:*k*] точной мантиссы результата определяется действующим рабочим режимом точности: [52:0] для *double*, [23:0] для *single*. Для формирования ошибки округления отбрасываемая часть мантиссы должна быть корректно сформирована, особенно в части $[k-1,\ m]$, разрядность которой равна разрядности основной части мантиссы.

Модернизация аппаратного модуля сумматора

Быстродействующие алгоритмы сложения/вычитания в настоящее время строятся по принципу разделения вычислительного процесса на две параллельные ветви, работающие при различных значениях разности экспонент операндов. Так, алгоритм, описанный в работе [5], делится на две части, называемые *n_path* и *far_path*.

Ниже приведены характеристики ветви *near_path*.

1. Разница экспонент равна одному из значений $\delta \in \{-1, 0, 1\}$ и вычисляется вычитанием двух младших битов экспонент исходных чисел. Сдвиг выравнивания определяется значением в одну позицию.

2. Происходит эффективное вычитание (SEFF), поэтому меньший операнд инвертируется. Используется представление *one's complement* отрицательного числа.

3. Разница мантисс (после перестановки и предварительного сдвига) лежит в диапазоне $fsum \in (-2,\ 2)$

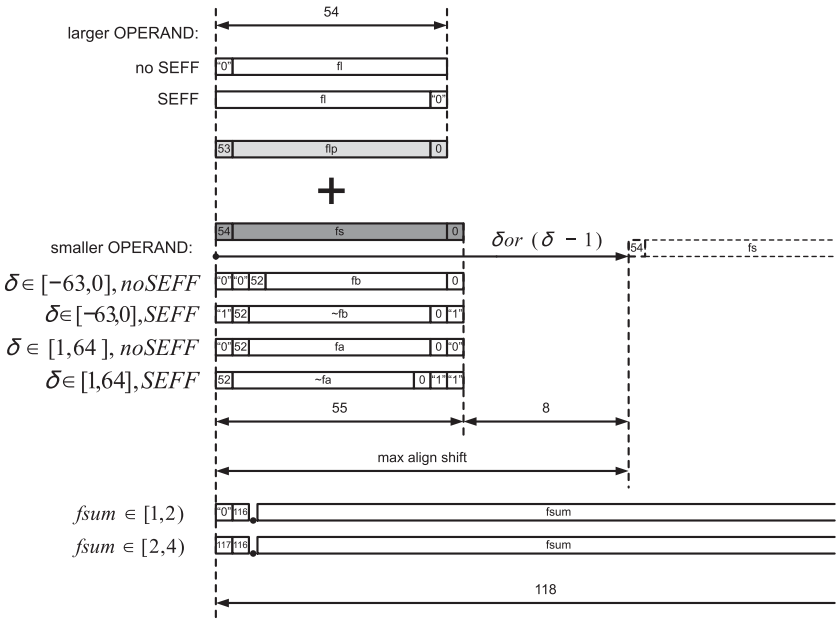

Рис. 2. Сложение выровненных мантисс в ветви *far_path*

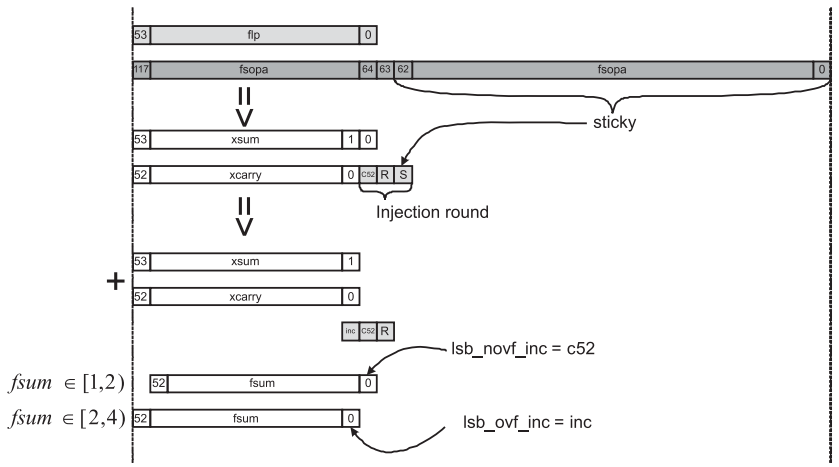

Рис. 3. Иллюстрация процесса округления в ветви *far_path*

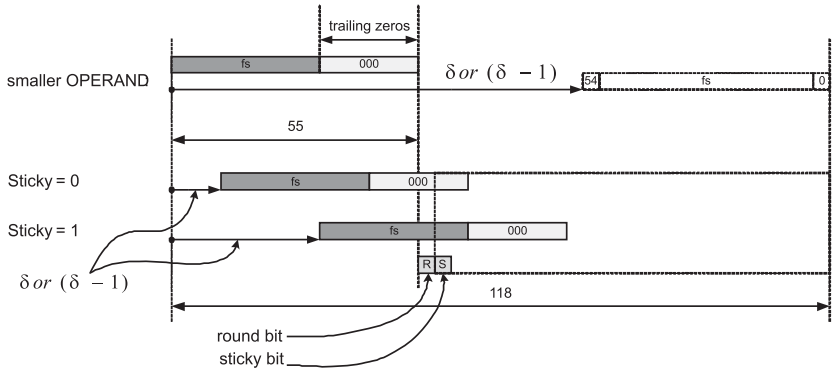

Рис. 4. Предсказание *sticky bit*

и, следовательно, может быть точно представлена 52 битами правее десятичной точки. Поэтому нет необходимости проводить округление.

Характеристики ветви *far_path* следующие.

1. Происходит эффективное сложение.

2. Происходит эффективное вычитание с разностью мантисс $fsum \in [2, 4)$ после предварительного сдвига.

3. Абсолютное значение разности экспонент $|\delta| \geqslant 2$.

Методика разделения на параллельные ветви обеспечивает выигрыш по быстродействию, позволяя упростить или убрать некоторые стадии простейшего алгоритма в каждой из ветвей. Более того, полноценное округление выполняется в ветви *far_path*, так как происходит существенное увеличение внутренней разрядности мантиссы.

Процесс выравнивания мантисс ветви *far_path* проиллюстрирован на рис. 2. Схема вычисления сдвига выравнивания алгоритма имеет следующие особенности:

- расчет сдвига выравнивания выполняется в коде *one's complement*:
$$\delta = \begin{cases} \delta - 1, if\ \delta > 0; \\ \delta, \quad if\ \delta \leqslant 0, \end{cases}$$
- сдвиг ограничен значением 64 (6 бит): 63 + предварительный сдвиг, при фактическом значении $\delta > 64$ сдвиг равен 64.

На рис. 3 показан процесс округления точной мантиссы ветви *far_path*.

Как видно из иллюстраций на рис. 2 и 3, формирование отбрасываемой части мантиссы осуществляется способом, который не подходит для вычисления ошибки округления ввиду ограничения максимального сдвига выравнивания меньшего слагаемого значением, равным 64. Отбрасываемая часть при таком подходе сигнализирует только о наличии потери точности при округлении. Это сделано для унификации поля выровненного значения меньшего операнда для сложения и вычисления признака потери точности (*sticky-bit*). Поэтому для вычисления ошибки округления необходимо корректное формирование отбрасываемой части

мантиссы, по крайней мере, следующих $[k-1, m]$ бит (см. рис. 1). Для этого необходимо увеличить порог ограничения максимального сдвига выравнивания до 127 бит.

Чтобы не усложнять внутреннее устройство оригинального алгоритма и сохранить ширину промежуточного значения мантиссы в ветви *far_path* на уровне 118 бит, не теряя при этом возможности корректно вычислять *sticky-bit*, необходимо заменить его прямое вычисление (межбитовое *OR fsopa*[62:0]) на предсказание. Для этого необходимо сравнить значение сдвига со значением младших нулей мантиссы меньшего числа *is* (рис. 4):

$$Sticky = \begin{cases} 1, fs_{tzer} < ashift, \\ 0, fs_{izer} \geqslant ashift. \end{cases}$$

В табл. 1 приведены выражения для предсказания *sticky-bit*. *Fa_tznum*[5:0], *fb_tznum*[5:0] — значения младших нулей мантисс операндов *a* и *b*, *delta*[11:0] — значение разности экспонент. *Sign_med* — знак разности экспонент для случая $|\delta| < 64$, **SEFF** — признак эффективного вычитания.

Таблица 1

Предсказание *Sticky-bit*

SIGN_MED	***SEFF***	*Sticky-bit condition* *(Sticky-bit = 1 if condition is negative)*
0	0	{6'*b*0, *fb_tznum*[5:0]} + ~*delta*[11:0] + 12'*b*01
0	1	{6'*b*0, *fb_tznum*[5:0]} + ~*delta*[11:0] + 12'*b*10
1	0	{6'*b*0, *fa_tznum*[5:0]} + *delta*[11:0] + 12'*b*10
1	1	{6'*b*0, *fa_tznum*[5:0]} + *delta*[11:0] + 12'*b*11

Таблица 2

Выражения для вычисления ошибки округления с учетом знака

RND_INC	***SEFF***	***RND_ERR***
0	0	−(*fsopa*[63:0]) = ~*fsopa*[63:0] + 64'*b*1
0	1	−(*fsopa*[63:0] + 64'*b*1) = *fspa*[63:0]
1	0	{1, 64'*b*0} − {1'*b*0, *fsopa*[63:0]} = = {1, 64'*b*0} + {1'*b*1, ~*fsopa*[63:0]} + 65'*b*1
1	1	{1, 64'*b*0} − {1'*b*0, (*fsopa*[63:0] + 64'*b*1)} = = {1, 64'*b*0} + {1'*b*1, *fspa*[63:0]}

Таблица 3

Выражения для вычисления модуля ошибки округления

RND_INC	***SEFF***	***|RND_ERR|***
0	0	*fsopa*[63:0]
0	1	*fsopa*[63:0] + 64'*b*1
1	0	~*fsopa*[63:0] + 64'*b*1
1	1	~*fsopa*[63:0]

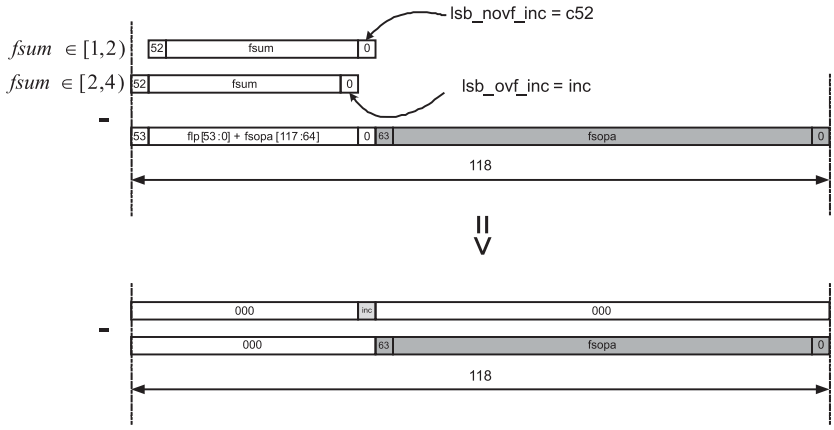

Рис. 5. Упрощение процесса вычитания при вычислении ошибки

Как было отмечено выше, ошибку округления можно вычислить как разность округленного числа и точного. Округление, в общем смысле, представляет собой инкрементирование определенной части мантиссы точного результата (определяемой разрядностью рабочего формата). В зависимости от битов, находящихся в этой части, инкрементирование может привести к распространению переноса до старших битов и привести к переполнению, а затем к корректировке экспоненты. Для упрощения аппаратного алгоритма вычисления ошибки целесообразно выполнять вычитание точного результата не из округленного числа, а вычитание отбрасываемой части мантиссы из константы округления — бита инкремента, что позволит не проводить вычитание старших полей мантисс, представляющих результат (рис. 5).

Значение выровненной мантиссы *fsopa* может быть отрицательным (*one's complement*), когда действует эффективное вычитание, поэтому действия, необходимые для вычисления ошибки округления, будут различны для разных сочетаний значений **SEFF** и **RND_INC**, табл. 2 и 3.

Модернизация упомянутого модуля будет заключаться в модификации одной из ветвей работы алгоритма — *far_path*. Список изменений следующий.

1. Увеличение порога ограничения максимального сдвига выравнивания до 127, для корректного формирования следующих 53 бит ошибки после *lsb* мантиссы основного результата.

2. Замена прямого вычисления *sticky-bit* (межбитовое *OR fsopa*[62:0]) на предсказание для повышения быстродействия: дополнение аппаратуры модуля счетчиками младших нулей операндов *a* и *b*, добавление компараторов определения потери точности.

3. Дополнение аппаратуры модуля инкрементором отбрасываемой части мантиссы для завершения двоичного дополнения до 2. Вычисления внутри оригинального алгоритма проводятся в форме *one's complement*.

4. Дополнение аппаратуры модуля подблоком предсказания отбрасываемой инкрементированной части мантиссы и счетчиком старших нулей предсказанной суммы для ускорения процесса нормализации мантиссы ошибки.

5. Дополнение аппаратуры модуля подблоками основной и постнормализации значения ошибки.

Модернизированная структурная схема аппаратной части *far_path* алгоритма приведена на рис. 6. Жирными линиями отмечены модификации, отличающиеся от оригинального алгоритма. Предложенные модификации оригинального блока нацелены на достижение новой функциональности и сохранение быстродействия компенсированного сложения на уровне обычной операции.

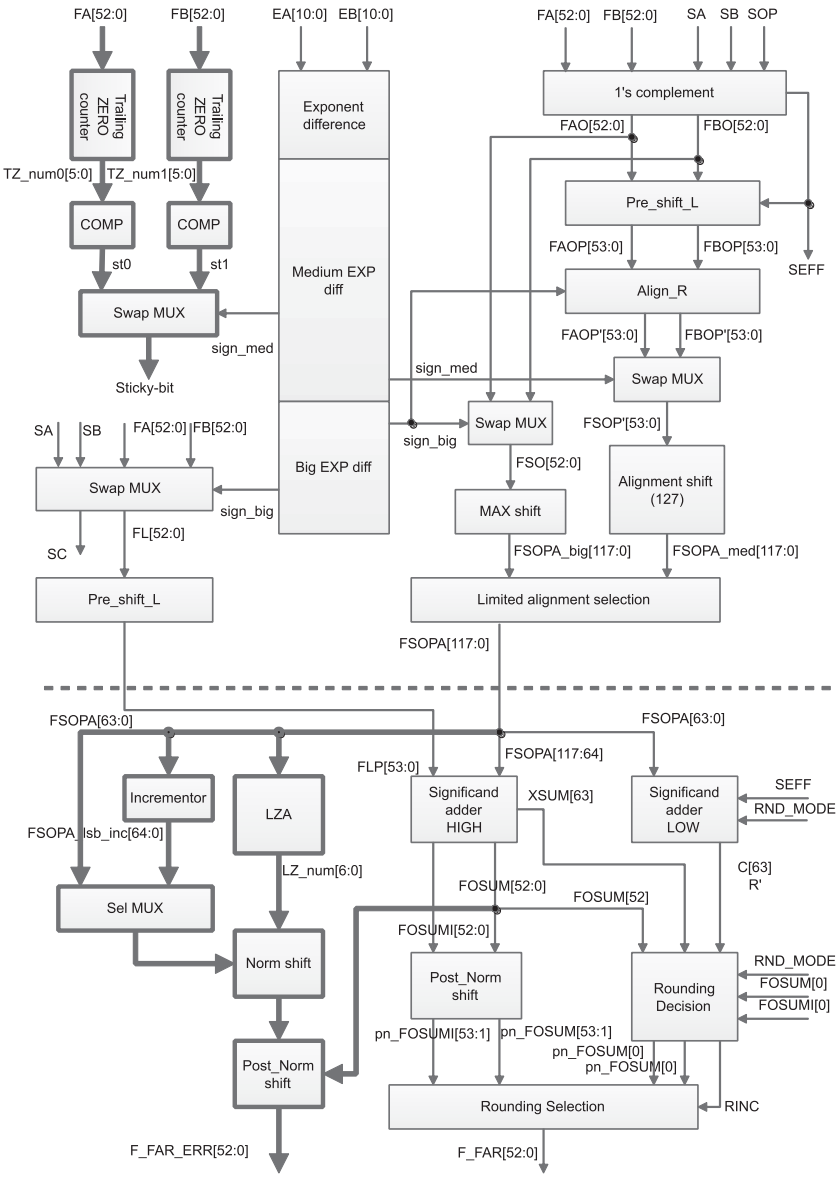

Рис. 6. Структурная схема модернизированной аппаратной *far_path* части алгоритма сложения вещественных чисел

Оценка влияния на быстродействие библиотеки

Оценить ускорение вычислений можно на примере вычисления функции библиотеки *exp*. Корректность вычисления экспоненты доказана авторами с помощью программ автоматической верификации. Стадия приведения аргумента осуществляется следующей последовательностью действий:

$$Mul12(\&s1, \&s2, msLog2DivLh, kd);$$

$$s3 = kd*msLog2Div2Lm;$$

$$s4 = s2 + s3;$$

$$s5 = x + s1;$$

$$Add12Cond = (rh, rm, s5, s4);$$

Mul12 — компенсированное умножение; *Add12Cond* — компенсированное сложение. Эти операции осуществляют приведение аргумента по формуле $r = x - k \cdot C$, где x — входной операнд; $C = \ln 2/4096$; $k = (x \cdot 4096/\ln 2)$. В данном случае константа C представлена двумя числами в формате *double: msLog2Div2Lh* и *msLog2Div2Lm*.

Компенсированное сложение было описано выше и состоит из шести вещественных операций сложения. Компенсированное умножение является существенно более сложной операцией и содержит в своем составе 17 вещественных операций:

$$function[x, y] = TwoMul(a, b),$$

$$x = fl(a*b),\ y = err,$$

$$ap = fl(a*c),\ bp = fl(b*c),$$

$$a1 = fl(fl(a - ap) + ap),$$

$$b1 = fl(fl(b - bp) + bp),$$

$$a2 = fl(a - a1),\ b2 = fl(b - b1),$$

$$y = fl(fl(fl(fl(fl(a1*b1) - x) + fl(a1*b2)) + fl(a2*b1)) + fl(a2*b2)).$$

Замена этих двух макрофункций на модифицированные операции сложения и умножения, длительность которых равна обычным операциям, повысит производительность вычислений. Принимая во внимание, что макрокоманды компенсированного сложения и умножения заменяются компилятором на 6 и 17 операций типа сложения и умножения, можно сделать вывод, что приведенная часть кода будет преобразована в 26 операций. При реализации аппаратного компенсированного сложения данная часть будет занимать 21 операцию, при реализации еще и компенсированного умножения — 5 операций.

Заключение

Выигрыш от аппаратной поддержки компенсированного сложения при вычислении экспоненты достигает 20 %, а реализация аппаратного компенсированного умножения увеличивает этот выигрыш до 5 раз. Таким образом, создание специализированного сопроцессора, базирующегося на блоках компенсированных арифметических операций и выполняющего набор основных трансцендентных функций для инженерных расчетов в сфере газовой и термодинамики, является перспективным направлением.

Список литературы

1. **Бобков С. Г., Аряшев С. И., Барских М. Е., Зубковский П. С., Ивасюк Е. В.** Высокопроизводительные расширения архитектуры универсальных микропроцессоров для ускорения инженерных расчетов // Информационные технологии. 2014. № 6. С. 27—37.
2. **CRLIBM** Documentation. URL: http://lipforge.ens-lyon.fr/www/crlibm/documentation.html. (Дата обращения: 30.06.2014).
3. **Аряшев С. И., Зубковский П. С., Ивасюк Е. В.** Реализация модуля операции "Умножение с накоплением" повышенной точности и быстродействия // 9-я Российская научно-техническая конференция "Электроника, микро- и наноэлектроника": сб. науч. трудов. — М.: МИФИ, 2007. С. 174—178.
4. **Dekkir T. J.** A floating-point technique for extending the available precision // Numer. Math. 1971. N. 18. P. 224—242.
5. **Knuth D. E.** The Art of Computer Programming, Vol. 2, Seminumerical Algorithms. Reading, MA; USA: Addison-Wesley, Third edition, 1998.
6. **Seidel P. M., Even G.** Delay-optimized implementation of IEEE floating-point addition // Computers, IEEE Transactions. 2004. N. 53 (2). P. 97—113.

S. I. Aryashev, Head of Department, **S. G. Bobkov,** Deputy Director,
P. S. Zubkovskiy, Head of Sector, **E. V. Ivasyuk,** Researcher
SRISA RAS, Moscow, e-mail: zubkovsky@cs.niisi.ras.ru

Development of Compensated Addition Hardware Module to Improve Calculation Accuracy

The article presents a development reasoning for arithmetic units that implement the compensated operations of addition and multiplication to accelerate elementary functions in scientific calculations of gas dynamics and thermodynamics.

Full hardware support for transcendental functions in the microprocessor is allowed in the case of a large calculation errors and the single-precision real arithmetic format. If double-precision result with an error of no more than 0,5 ulp is required, it is advisable to use a hardware-software approach, since the cost for a fully hardware implementation are extremely high.

Despite the fact that the standard 1EEE754 did not declare anything about elementary (transcendental) functions, there are software libraries that provide the calculation of these functions with an error of not more than 0,5 ulp mantissa format double. This ensures the stability of the software results, enabling cross-platform compatibility. Therefore, there is a method for providing hardware accelerated some algorithmic "primitives", involved in a software library, as an alternative to conventional hardware acceleration calculating elementary functions. This is the compensated functions of addition and multiplication, which are the basic building block of libraries with double/triple internal precision (double-double, triple-double).

For example, the library CRLIBM implements the calculation of elementary functions with correct rounding. The library is written in C and it is support the double-double/triple-double accuracy of intermediate calculations, and also provides support for hardware operations FMA (Fused Multiply-Add).

Development of hardware floating-point units that implement the compensated lower-level operations will accelerate the implementation of compensated library functions allowing the use of specialized arithmetic unit instead of standard FPU.

Authors described the proposed improvement of the floating-point addition module to perform the addition of macro compensated CRLIBM.

The benefits of adding the compensated hardware-unit support in the exponent function calculation reached 20 percent, and implementation of compensated hardware multiplier provides the acceleration up to 5 times. Thus, the creation of a specialized coprocessor based on compensated arithmetic units and supported a set of basic transcendental functions operations for scientific calculations in the field of thermodynamics and gas is a promising direction.

Keywords: *compensated addition, elementary functions, hardware acceleration*

References

1. **Bobkov S. G., Aryashev S. I., Barskyh M. E., Zubkovskiy P. S., Ivasyuk E. V.** Informacionnye tehnologii, 2014, no. 6, pp. 27—37 (in Russian).
2. http://lipforge.ens-lyon.fr/www/crlibm/documentation.html (accessed 30.06.2014).
3. **Jlektronika,** mikro- i nanojelektronika. 9-ja Rossijskaja nauchno-tehnicheskaja konferencija, 2007, pp. 174—178 (in Russian).
4. **Dekker T. J.** A floating-point technique for extending the available precision. Numer. Math., 1971, no. 18, pp. 224—242.
5. **Knuth D. E.** The Art of Computer Programming, Vol. 2, Seminumerical Algorithms. Addison-Wesley, Reading, MA, USA, 1998, 784 p.
6. **Seidel P. M., Even G.** Delay-optimized implementation of IEEE floating-point addition, Computers, IEEE Transactions on, 2004, no. 53 (2), pp. 97—113.