УДК 004.315.5

Реализация базовых функций задачи горения на основе операции FMA специализированного векторного сопроцессора

С.И. Аряшев, М.Е. Барских, С.Г. Бобков, П.С. Зубковский, Е.В. Ивасюк

Научно-исследовательский институт системных исследований РАН, г. Москва,

ivasyuk@cs.niisi.ras.ru, zubkovsky@cs.niisi.ras.ru

***Аннотация* — В статье приводятся результаты профилирования задачи горения для выделения характерного набора математических функций, описываются результаты работы по оптимизации библиотечных функций и предлагается набор решений для создания специализированного потокового сопроцессора для вычисления трансцендентных функций, реализуемого в базисе самосинхронной логики с использованием модифицированного блока умножения с ошибкой вычисления не более 0.5ulp.**

***Ключевые слова* — задача горения, расширение архитектуры микропроцессора, потоковый сопроцессор, самосинхронная логика.**

I. Введение

В настоящее время развитие ядерной и термоядерной энергетики, электроники, авиа- и ракетостроения, биотехнологий и т.д. стало невозможным без проведения полномасштабных инженерных расчётов с учётом атомно-молекулярного взаимодействия. Одна из ключевых проблем, требующая решения – моделирование процессов сгорания новых видов топлива в двигателях новых конструкций. Учет этих процессов требует предсказательного многомасштабного моделирования, включая течение реакции на уровне отдельных молекул и режимы детонации в камере сгорания. Именно многомасштабность процессов горения не позволяет провести их прямое численное моделирование с использованием существующих супер-ЭВМ. В системе моделирования должны одновременно работать совместно различные вычислительные алгоритмы: квантовая молекулярная динамика, классическая молекулярная динамика, кинетический метод Монте-Карло, имитационное моделирование, моделирование больших вихрей, методы с усреднением числа Рейнольдса и др., что в свою очередь требует использования супер-ЭВМ эксафлопсного класса.

II. Профилирование задачи горения

Современный подход к компьютерному моделированию процессов горения и детонации предусматривает распараллеливание вычислительных программ на многопроцессорных ЭВМ и графических ускорителях [1]. С помощью профилирования можно получить информацию о наиболее часто используемых в таких задачах функциях или о функциях, вычисление которых занимает наибольшее время. В таблице 1 приводятся результаты профилирования программ метода ETDRK4 для системы ОДУ с кинетикой Мааса и Варнаца (горение водорода, 9 компонент, 19 реакций) и газодинамический код с учетом той же кинетики, но решение системы осуществляется 4-х стадийным методом Розенброка. Запуск производился на ЭВМ архитектуры x86 с процессором Intel(R) Core(TM)2 Duo CPU E6550 2.33GHz. В первом столбце указано время выполнения функции в процентах от общего времени выполнения программы, во втором столбце – название функции, а в третьем – примечания. Анализ кода функций, наиболее часто применяемых в рассматриваемой реализации задачи горения, показывает, что в большинстве случаев выполняются операции умножения и сложения над матрицами и векторами, векторные операции деления и извлечения квадратного корня. Вызовы функций вычисления экспоненты и возведения в степень происходят в цикле над отдельными элементами вектора, используя, таким образом, только скалярные функции.

Набор функций, описанный выше на частном примере реализации задачи горения, является характерным для многих вычислительных задач, и именно время выполнения этих операций наряду с возможностями подсистемы памяти характеризует производительность микропроцессора на инженерных и научных расчетах.

III. Оптимизация математических библиотечных функций

В микропроцессоре 1890ВМ9Я, разрабатываемом в НИИСИ РАН по проектным нормам 65 нм, предусмотрен векторный сопроцессор, оптимизированный под задачи линейной алгебры и комплексные вычисления. Вычислительный блок векторного сопроцессора позволяет выполнять операции одинарной и двойной точности над векторами шириной 128 разрядов. Пиковая производительность достигается при использовании команд комплексного умножения с прибавлением и вычитанием третьего операнда («бабочка Фурье») и составляет за такт 10 вещественных операций двойной

точности или 20 вещественных операций одинарной точности.

Отказ от использования локальной специализированной памяти данных (как в сопроцессорах микропроцессоров 1890ВМ6Я и 1890ВМ9Я) позволяет использовать вычислительные возможности сопроцессора на пользовательских приложениях, скомпилированных стандартными средствами с подключением математических библиотек.

Таблица 1

Результаты профилирования задачи горения

Время выполнения	Название функции	Примечание
20.93%	GetTransport	Коэффициенты переноса для газовой смеси на основании молярных масс или плотностей компонентов
11.67%	__ieee754_exp	Библиотечная функция вычисления вещественной экспоненты (библиотека libm-2.17.so)
6.88%	UpdateExrate	Добавка данных от элементарной реакции в интенсивность образования компонент
6.70%	ElemCofs	Элементарный механизм: коэффициенты для прямой и обратной реакции
5.11%	GetThermo	
4.02%	ExplTurbStep	Явная стадия турбулентности
3.53%	GetJacobean	Вычисление якобиана
3.47%	__pow_finite	Библиотечная функция возведения в степень (библиотека libm-2.17.so)
2.59%	GetDifComp	Учет диффузии компонентов
2.53%	Y2X	Вычисление плотности компонентов в каждой ячейке

Важным условием эффективного использования векторного сопроцессора на пользовательских задачах является наличие математической библиотеки, оптимизированной под архитектуру сопроцессора.

Результаты профилирования тестов производительности LinPack показывают, что наиболее используемыми функциями библиотеки BLAS (более 90% от времени исполнения) являются функции DGEMM – перемножение матриц, DTRSM – решение системы линейных дифференциальных уравнений и DGETRF – LU-разложение, причем функция DGEMM является базовой для DTRSM и DGETRF при использовании блочных методов. Эффективность использования векторного сопроцессора оценивалась по формуле $Eff = N_{RTL}/N$, где Eff – искомая эффективность, N_{RTL} число машинных тактов, затраченных на выполнение теста с вызовом функции, N – теоретически вычисленное число арифметических команд сопроцессора, выдаваемых в каждом такте. Максимальное значение эффективности приближается к 64% для случая, когда исходные данные полностью помещаются в кэш память второго уровня, число арифметических команд достаточно велико, чтобы скрыть затраты на передачу данных, а комплексный тип данных позволяет использовать наиболее производительные команды сопроцессора.

Для достижения высокой производительности при проведении инженерных расчетов необходимо не только наличие оптимизированной под конкретную архитектуру математической библиотеки линейной алгебры, но и возможность аппаратного ускорения вычисления трансцендентных функций. Выполнение таких расчетов в целочисленном формате или в формате с плавающей запятой одинарной точности уже не отвечает постоянно возрастающим требованиям к точности инженерных расчетов.

IV. Вычисление элементарных функций

В условиях ограниченной разрядности представления данных в ЭВМ трансцендентные функции не могут быть вычислены точно, а могут быть только аппроксимированы. Одним из распространенных алгоритмов вычисления упомянутых функций является аппроксимация полиномом/рациональной дробью или разложение в степенной ряд. Ключевым параметром в проектировании алгоритма аппроксимации является максимальная допустимая ошибка вычисления. От этой ошибки в первую очередь зависит сложность реализации алгоритма, какой бы он ни был: программный, программно-аппаратный или полностью аппаратный.

Так как диапазон изменения аргумента, от которого требуется найти значение функции, имеет большую протяженность (согласно стандарту IEEE754), то для обеспечения допустимого значения погрешности на всем диапазоне требуется обеспечить очень большую степень аппроксимирующего полинома или большое число членов ряда разложения. Это потребует больших временных затрат на вычисление, и поэтому для упрощения полиномов/рядов применяют сужение диапазона входного аргумента, называемое приведением аргумента (range reduction, argument reduction) [2]. Затем осуществляется вычисление полинома от приведенного аргумента с заданной точностью и восстановление результата до значения, соответствующего изначальному диапазону изменения аргумента. На рис.1 приведены зависимости числа «точных» битов аппроксимации от степени полинома для различных функций для диапазона входного аргумента [0,1][2]. Из рисунка видно, что для того, чтобы обеспечить получение результата в формате double (IEEE754, $2^{-52} \approx 10^{-16}$) с по-

грешностью 1ulp и менее, необходим полином степени более 10.

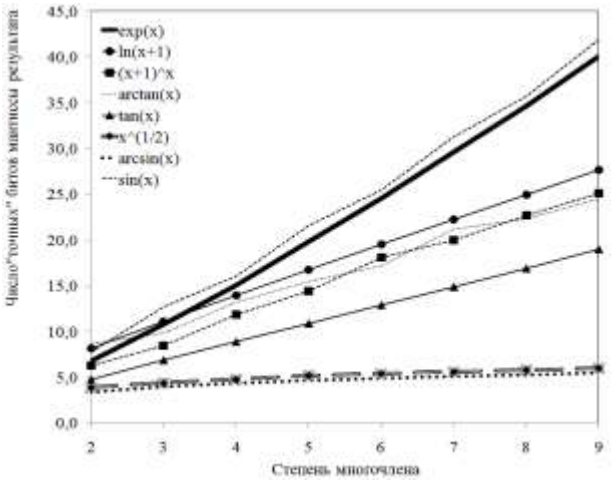

Рис. 1. Зависимость числа «точных» битов аппроксимации от степени полинома для различных функций

Как видно из рисунка, функция e^x является самой простой с точки зрения вычисления на ЭВМ, потому как полином, обеспечивающий ее аппроксимацию, демонстрирует большую скорость сходимости при увеличении степени полинома, а также потому, что диапазон входного аргумента, при котором функция не вызывает исключений предусмотренных стандартом IEEE754, узок относительно полного диапазона стандарта. Так, для формата double, диапазон в котором e^x не вызывает переполнения и потери значимости, составляет [-708, +709], если не обрабатываются денормализованные числа, и [-745, +709], если обрабатываются денормализованные числа.

В настоящее время широко применяется полиномиальная аппроксимация функций по минимаксному принципу, в которой используется алгоритм Ремеза расчета коэффициентов. Минимаксная аппроксимация функции обеспечивает минимизацию максимальной ошибки на заданном диапазоне. Расчеты полиномов сильно упрощаются благодаря использованию таких программ компьютерной алгебры, как, например, Maple.

Ошибка вычисления функции складывается из погрешностей стадии приведения аргумента, погрешности стадии вычисления полинома и погрешности стадии восстановления/округления. Стандарт IEEE754 в последней редакции (2008г) ничего не декларирует об элементарных функциях, но, тем не менее, существуют программные библиотеки, обеспечивающие точное округление, например, CR-LIBM[3], правильность реализации функции экспоненты в которой доказана формально, и итоговая погрешность вычисления составляет 0.5ulp.

V. Приведение аргумента

Для разработки алгоритма e^x на аппаратном или программно-аппаратном уровне необходимо оценить необходимую разрядность внутреннего представления данных для обеспечения погрешности в 0.5ulp и решить, можно ли ее обеспечить на всех стадиях вычисления, используя существующие арифметические модули вещественной арифметики микропроцессора (модуль сложения, модуль умножения, модуль деления).

Наиболее часто для приведения используется алгоритм, основанный на вычитании. Например, в CR-LIBM используется следующее приведение аргумента r = x - k * C, где x – входной операнд, C= ln2/4096, k = int_rnd(x*4096/ln2).

Известно, что основным недостатком алгоритма приведения, основанного на вычитании, является потеря значащих битов при некоторых значениях входного операнда. Близость произведения k*C к значению входного операнда x называется наихудшим случаем. Мера этой близости – количество старших нулей результата вычитания. Например, для числа x = 112.39154770762435 и C = 1.6922538587889290e-4 в формате double и соответствующего k = 664153 количество старших нулей при вычислении операции приведения с помощью независимых команд умножения и сложения будет равно 52, т.е. все значащие биты мантиссы результата будут потеряны. Для минимизации ошибки приведения вычисления требуют значительного (двух/трехкратного) увеличения разрядности, а также обеспечения константы C повышенной точности.

Так, величина внутренней разрядности этапа приведения должна составлять $W_{RED} = N_{ZER\_WORST\_CASE} + W_{MAN\_D} + W_{GUARD}$, где $N_{ZER\_WORST\_CASE}$ – количество старших нулей при вычитании, W_{MAN_D} – разрядность мантиссы формата double (53), W_{GUARD} – число “охранных” битов, необходимых для округления. Таким образом, для того чтобы обеспечить формирование 53 значащих битов мантиссы приведенного аргумента для вышеуказанного наихудшего случая, разрядность внутреннего представления данных на этапе приведения будет составлять, как минимум, W_{RED} = 105 бит.

VI. Аппаратная реализация приведения аргумента в сопроцессоре

Блок FMA (Fused Multiply-Add) [4], используемый в векторном сопроцессоре процессора K64M (микросхема 1890ВМ8), может быть использован для реализации алгоритма приведения аргумента. Внутренняя разрядность данных в этом блоке составляет 161 бит, что позволяет использовать его для создания схемы вычисления полиномов в формате double с погрешностью 0,5ULP. На рис. 2 приведены иллюстрации, поясняющие внутреннюю организацию положения мантисс.

Ряд модификаций существующего блока FMA могут повысить эффективность использования утроенной разрядности внутреннего представления мантиссы.

Для увеличения точности приведенного аргумента необходимо увеличить разрядность представления константы C, а именно, представлять ее двумя операндами формата double: C = C1 + C2. Тогда разрядность

произведения k * (C1 + C2) будет составлять 159 разрядов.

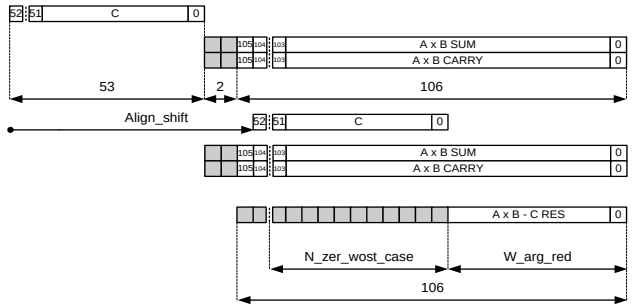

Рис. 2. Внутренняя организация положения мантисс в модулях FMA

Таким образом, модернизация модуля заключается в добавлении входа для второй части константы C (формат double), схемы умножения мантисс k * C2. Структурная схема модернизированного модуля представлена на рис. 3. Модуль может обрабатывать четыре числа в формате double, результат операции равен:

$$r = x - k * (C1 + C2).$$

Необходимо отметить, что режим обработки четырех чисел работает при условии равенства экспонент числа x и виртуального произведения k * (C1 + C2). Для произвольных чисел доступен обычный режим умножения с накоплением.

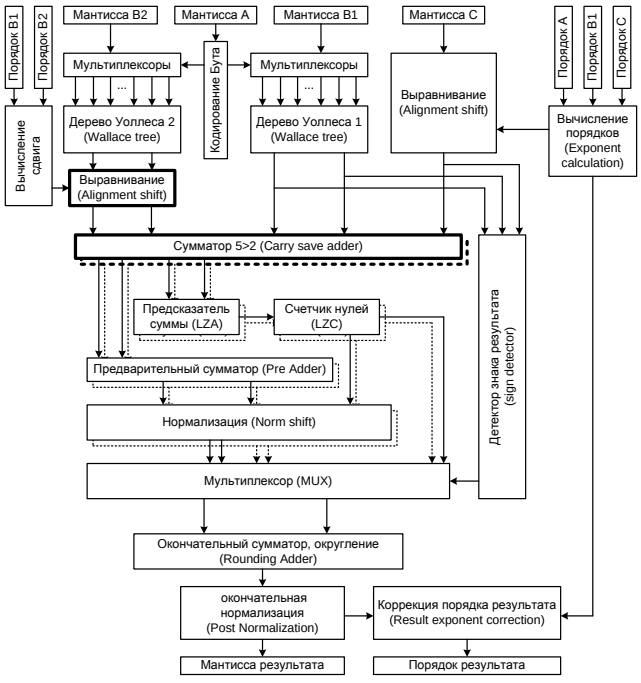

Рис. 3. Структурная схема модернизированного модуля FMA, предназначенного для точного приведения

Данная модернизация позволяет выполнить стадию приведения аргумента за одну вещественную операцию, а не за две (r = x - k * C1 - k * C2). Время выполнения инструкции приведения аргумента увеличится по сравнению с операцией FMA на значение:

$$\Delta_T = T_{выр} + (T_{5>2} - T_{3>2}),$$

где $T_{выр}$ – задержка схемы выравнивания, равная k * C2, $T_{3>2}$ – задержка компрессора трех чисел в два, $T_{5>2}$ – задержка компрессора пяти чисел в два. Задержка $T_{5>2} - T_{3>2}$ эквивалентна четырем элементам XOR. Задержка схемы выравнивания $T_{выр}$ теоретически лимитирована значением сдвиговой схемы с 6-битным управлением. Иначе говоря, задержка обуславливается величиной старших нулей младшей части константы (рис. 4).

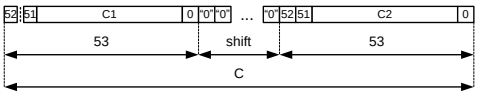

Рис. 4. Деление константы C на два числа

Однако на практике константа C, представленная в двоичной форме, будет иметь небольшое значение shift (рис. 4). Для значений C = n*ln(2) shift = 2, для C = n*π shift = 1. Следовательно, задержка схемы сдвига будет эквивалентна двум мультиплексорам или двум элементам XOR. Полное увеличение задержки Δ_T = 6XOR.

VII. Вычисление полинома

Имея в архитектуре микропроцессора модуль FMA, целесообразно использовать его не только для приведения аргумента, но и для вычисления полинома. При таком подходе полином приводится к виду Горнера, который предполагает трансформировать процесс вычислений в цепь зависимых операций «умножение с накоплением». Структурная схема аппаратного блока, реализующего алгоритм вычисления элементарной функции, представлена на рис. 5a.

Уменьшить вычисляемую степень полинома или глубину цепочки операций «умножение с накоплением» можно распараллеливанием вычислений. Например, можно привести такой полином к виду, который предполагает одновременное вычисление нескольких его частей:

$$P(x)_{2n} = ((\ldots((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2})\ldots) \cdot x + a_0) +$$
$$x^n \cdot ((\ldots((a_{2n} \cdot x + a_{2n-1}) \cdot x + a_{2n-2})\ldots) \cdot x + a_{n+1}).$$

Структурная схема аппаратного блока, обеспечивающего вычисление значения функции на основе такого полинома, представлена на рис. 5c. Для реализации такого алгоритма требуется поддержка вычисления функции x^n, где n-целое число.

Вычисление функции можно осуществить также с помощью рациональной аппроксимации, которая позволяет уменьшить расчетную степень аппроксимирующих полиномов числителя и знаменателя примерно наполовину по сравнению с вариантом, представленным на рис. 5a.

Выражение для вычисления имеет вид:

$$P(x)_n = \frac{((...((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2})...) \cdot x + a_0)}{((...((b_n \cdot x + b_{n-1}) \cdot x + b_{n-2})...) \cdot x + b_0)}.$$

Однако требуется аппаратная поддержка операции деления значений числителя и знаменателя. Структурная схема блока, обеспечивающего вычисление значения функции на основе рациональной аппроксимации, представлена на рис. 5b.

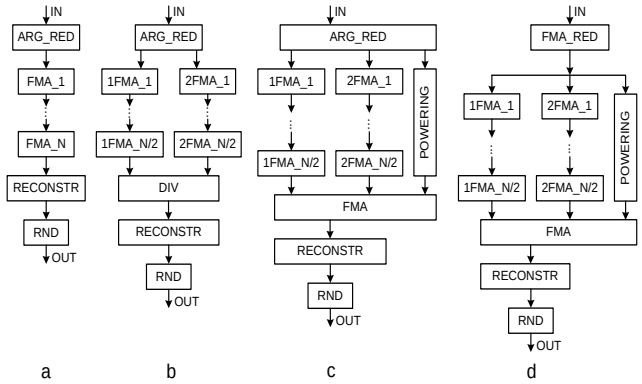

Рис. 5. Структурные схемы аппаратных блоков вычисления элементарных функций

Структурная схема, представленная на рис. 5c, имеет меньшую задержку чем, схема на рис. 5b, так как операция FMA, завершающая вычисление, выполняется существенно быстрее, чем операция DIV, а задержка возведения в степень может быть маскирована задержкой основных конвейеров вычисления полиномов. Структурные схемы на рис. 5 полностью конвейерные, задержки вычисления для данных алгоритмов следующие:

Рис. 5a : $T_{total}=T_{red}+T_{fma}*N+T_{reconstr}+T_{rnd}$;

Рис. 5b : $T_{total}=T_{red}+T_{fma}*N/2+T_{div}+T_{reconstr}+T_{rnd}$;

Рис. 5c : $T_{total}=T_{red}+T_{fma}*N/2+T_{div}+T_{fma}+T_{reconstr}+T_{rnd}$.

Если в качестве базовой операции этапа приведения будет выбрано «умножение с накоплением», реализуемое предложенным модернизированным модулем FMA, то структурная схема модуля, реализующего весь алгоритм, становится более регулярной, рис. 5d.

VIII. Ошибка при вычислении полинома

Вычисление значения полинома высокой степени потребует многократного конвейерного/итерационного использования блока FMA, что может внести не меньшую ошибку в вычисления, чем стадия приведения.

Например, для диапазона приведенного аргумента [-(ln2)/2, (ln2)/2] при точности внутренних расчетов 50 десятичных разрядов для функции e^x Maple дает оценку степени полинома равную 11 для ошибки расчета полинома равной 4.33e-18. Если в качестве «строительного» элемента используется модуль FMA с фиксированной разрядностью входных операндов (double), то коэффициенты придется округлить до 53 бит, и тогда ошибка увеличится до значения 3.4e-16, что уже больше величины 1ulp формата double. Причем в том простом примере не учитывается влияние округления между операциями FMA внутри полинома, и очевидно, что чем больше степень полинома, тем больше цепь операций и больше накапливаемая ошибка. Этого можно избежать, применяя внутри расчетов арифметику типа double-double (triple-double, quad-double, если необходимо), которая заключается в представлении числа в виде суммы двух чисел с непересекающимися полями мантисс. Этот способ позволяет не округлять расчетную разрядность коэффициентов до 53 разрядов, а представлять каждый коэффициент в виде двух чисел формата double. Таким способом можно увеличить виртуальную разрядность внутренних вычислений до значений 100 и более битов.

Альтернативный способ - использовать компенсированные алгоритмы вычисления полинома по схеме Горнера, которые заключаются в вычислении ошибки вещественной операции параллельно основному расчету полинома. Использование такого компенсированного алгоритма обеспечивает уменьшение ошибки расчета до такого уровня, как если бы разрядность повысилась в два раза. Сравнение времени выполнения стандартной схемы Горнера, схемы Горнера, использующей арифметику double-double и компенсированной схемы Горнера показывает, что применение double-double схемы обеспечивает замедление по сравнению со стандартной в 8 раз, в то время как применение компенсированной схемы – замедление в 3 раза [5][6].

В этом случае вычисление значения полинома происходит на раздельных операциях умножения и сложения, а ошибка операций вычисляется с использованием блоков FMA:

function[x,y] = FastTwoSum(a,b)

x = a + b

if(|a| < |b|) y = ((b - x) + a) else y = ((a - x) + a)

function[x,y] = TwoProd(a,b)

x = a * b, y = x – a * b = fma(x, a, b)

Алгоритм компенсированной схемы Горнера будет выглядеть следующим образом:

function_r = CompHorner(P, x)

$r_n = a_i$; $c_n = 0$

for(i = n – 1: -1: 0)

$[p_i, \pi_i]$ = TwoProd(r_{i+1}, x), $t = c_{i+1} * x + \pi_i$

$[r_i, \sigma_i]$ = TwoSum(p_i, a_i), $c_i = t_i + \sigma_i$

Применительно к архитектуре микропроцессора К64М с векторным сопроцессором вычисления по этой схеме могут быть частично распараллелены на блоке вещественной арифметики FPU и векторном сопроцессоре (рис. 6).

IX. Выводы

Для реализации функции e^x в рамках задачи горения достаточно точности результата на уровне 1ulp

мантиссы формата double. Следовательно, на этапе приведения нет необходимости сохранять полную разрядность промежуточного результата и достаточно одной-двух операций FMA для вычисления приведенного аргумента. Даже в случае полной потери значащих битов приведенного значения операнда, точность 1ulp значения e^x обеспечивается разрядностью коэффициентов, так как функция хорошо определена вблизи нуля. В этом случае можно приблизительно оценить быстродействие алгоритмов на рис. 5. Время выполнения операции FMA составляет 4 такта, а операции деления – 8 тактов. Для приведения аргумента достаточно использовать одну операцию FMA, а степень аппроксимирующего полинома составляет 11. Точность результата получается не хуже 1ulp. В этом случае время вычисления экспоненты для схемы на рис. 5a составит примерно 50 тактов, для схемы 5b – около 35 тактов, а параллельное возведение в степень (рис. 5c и 5d) позволяет сократить время выполнения до 30 тактов.

Однако, если применимость проектируемого алгоритма выходит за пределы данной задачи, более того, если предполагается использовать его в качестве элемента универсального сопроцессора для вычисления различных элементарных функций с точностью результата на уровне 0.5ulp, то требования к погрешности промежуточных результатов будут более строгими. В этом случае целесообразно использовать компенсированные алгоритмы вычисления полинома (рис. 6) и приведение аргумента на основе предложенной модифицированной структуры FMA.

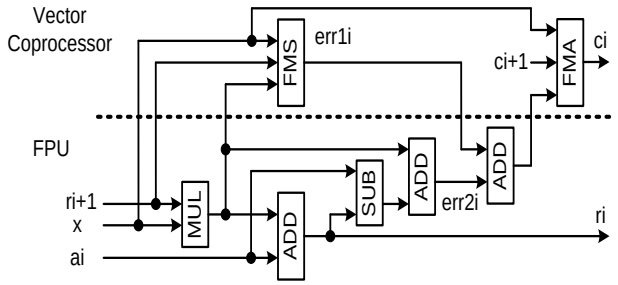

Рис. 6. Схема вычисления компенсированного алгоритма Горнера

Описанные варианты схем вычисления полиномов могут иметь конвейерное исполнение и осуществлять передачу данных от одной стадии к другой синхронно в соответствии с тактирующим сигналом или асинхронно по мере готовности результата на каждой стадии. Второй вариант исполнения придает всей схеме в целом характер потоковой вычислительной схемы. Один из вариантов реализации такой схемы основан на использовании блока FMA, выполненного как заказной блок в базисе самосинхронной логики [13]. Такой вариант реализации не только дает большой выигрыш в энергоэффективности при сохранении быстродействия, сравнимого с синхронной реализацией, но повышает надежность схемы, включая функции самовосстановления после сбоя.

Заключение

Предложенные в статье способы повышения производительности компьютерного моделирования процессов горения и детонации включают в себя выделение базовых функций путем профилирования программного кода задачи горения, оптимизацию функций библиотеки линейной алгебры для ускорения матрично-векторных операций, а также разработку архитектуры специализированного потокового сопроцессора для вычисления аппроксимирующих полиномов трансцендентных функций, в первую очередь экспоненты. Повышение энергоэффективности и надежности потокового сопроцессора при сохранении быстродействия предусматривает использование заказных самосинхронных вычислительных узлов.

Описанные решения позволяют разработать супер-ЭВМ эксафлопного класса, используя порядка 10^6 ядер, что на 2-3 порядка меньше необходимого числа универсальных ядер коммерческого применения без специализированных сопроцессоров. Алгоритмы, реализующие выделенные по результатам профилирования задачи функции, в целом решают поставленную задачу, но требуют дополнительного изучения задачи горения и оценки площади и производительности для окончательного принятия решения. Использование самосинхронной логики позволит решить следующие проблемы, возникающие при разработке эксафлопсной суперЭВМ: снижение потребления и повышение надежности вычислительных узлов.

Работа выполнена в рамках проекта РФФИ № 13-07-12062 «Фундаментальные проблемы создания микропроцессоров и коммуникационных сред супер-ЭВМ эксафлопсного класса, ориентированных на предсказательное моделирование задачи горения».

Литература

[1] Рыбакин Б.П. Решение трехмерных задач газовой динамики на графических ускорителях // Вестник УГАТУ. 2012. Т. 16. №6 (51).

[2] Muller J.-M. Elementary Functions, Algorithms and Implementation. Birkhauser Boston. 2nd edition. 2006.

[3] URL: http://lipforge.ens-lyon.fr/www/crlibm/documentation.html (дата обращения: 28.01.2014).

[4] Зубковский П.С., Ивасюк Е.В., Аряшев С.И. Сопроцессор комплексных вычислений // IV Всероссийская научно-техническая конференция «Проблемы разработки перспективных микро- и наноэлектронных систем – 2010». Сб. трудов. М: ИППМ РАН, 2010. С. 356 – 359.

[5] Kornerup P., Lefèvre V., Louvet N. and Muller J.-M. On the computation of correctly-rounded sums // IEEE Transactions on Computers. 2012. V. 61. Iss. 3. P. 289 - 298.

[6] Langlois P. and Louvet N. More Instruction Level Parallelism Explains the. Actual Efficiency of Compensated Algorithms // Technical report. 2007. DALI research team. University of Perpignan, France.

[7] Плеханов Л.П. Основы самосинхронных электронных схем. М.: БИНОМ. Лаборатория знаний, 2013. 208 с.