Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 5 N 1

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2015

Тематика номера:
Моделирование в микроэлектронике, моделирование динамических природных и рукотворных процессов, высокопроизводительные вычисления, математические исследования, методы вычислений и вопросы программирования

Журнал публикует оригинальные статьи по следующим областям исследований: математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника

The topic of the issue:
Modeling in microelrctronics, modeling of dynamic natural and man-made processes, high-performance computing, mathematical investigations, methods of calculation and programming problems

The Journal publishes novel articles on the following research arias: mathematical and computer modeling, system analysis, image processing, visualization, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics

Заведующий редакцией: Ю.Н. Штейников

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

V. ОБУЧЕНИЕ АЛГОРИТМИКЕ И ПРОГРАММИРОВАНИЮ

Моделирование интервально стохастических нестационарных и нелинейных тепловых процессов в электронных системах

А.Г. Мадера

доктор технических наук, профессор

Аннотация. Тепловые процессы в реальных электронных системах в общем случае носят неопределенный, нестационарный и нелинейный характер. Однако в настоящее время их моделирование проводится в предположении как о детерминированности всех определяющих факторов, так и линейности тепловых процессов. В статье анализируется состояние исследований и постановка проблемы создания адекватных методов математического и компьютерного моделирования тепловых процессов в электронных системах, максимально полно учитывающих реальные особенности электронных систем.

Ключевые слова: тепловые процессы, электронная система, интервальный, стохастический, нестационарный, нелинейный

Введение

Существующие в настоящее время методы математического и компьютерного моделирования тепловых процессов в электронных системах (ЭС) основаны на допущении, что факторы, определяющие тепловые процессы, полностью известны и однозначно определены, то есть являются детерминированными. Между тем практика показывает, что факторы, определяющие тепловые процессы в ЭС (далее – определяющие факторы) отнюдь не являются точно и однозначно известными. Так, сравнение одних и тех же определяющих факторов у различных экземпляров «идентичных» ЭС, как правило, существенно различаются между собой. Измерения этих факторов для партии «одинаковых» ЭС, например, теплового сопротивления корпусов микросхем одного и того же типа, или их мощностей потребления, показывают, что значения определяющих факторов изменяются внутри некоторых интервалов, границы которых определяются уровнем и характером применяемой технологии. Конкретный экземпляр ЭС содержит множество различных и взаимозаменяемых элементов (микросхем (МС), электро- и радиоэлементов (ЭРЭ)), причем каждый их них имеет вполне определенные определяющие факторы, однако, какие именно значения принимают эти факторы у данного конкретного элемента, используемого в данной ЭС, априори не известно. Поэтому при тепловом проектировании необходимо учитывать, что для любой ЭС, построенной из множества различных элементов (МС и ЭРЭ), могут быть известны лишь границы интервалов, внутри которых заключены реальные значения определяющих факторов. Поэтому детерминированный подход к моделированию тепловых процессов, принятый в настоящее время при тепловом проектировании ЭС, не позволяет получать адекватные реальности результаты, а пренебрежение интервально неопределенным характером факторов, определяющих тепловые процессы, может приводить к ошибкам проектирования ЭС и, как следствие – нежелательным последствиям (ложному срабатыванию, выходу из строя, снижению надежности, быстродействия и пр. [3]).

Таким образом, факторы, определяющие тепловые процессы в ЭС, не являются детерминированными, а носят неопределенный интервальный характер. Каждый определяющий фактор представляет собой случайную величину, принимающую значения внутри своего интервала изменения и подчиняющуюся некоторому усеченному (ограниченному границами интервала) закону распределения вероятностей и равному нулю вне его. Иначе говоря, определяющие факторы в ЭС носят интервальный стохастический характер. Неопределенный характер определяющих факторов обусловливается, во-первых, статистическим технологическим разбросом параметров ЭС и ее элементов при изготовлении и монтаже, во-вторых, случайными факторами, возникающими при функционировании ЭС и, в-третьих, случайными параметрами окружающей среды и условий эксплуатации ЭС.

Первая группа факторов, вызывающая неопределенность тепловых процессов, обусловлена статистическим технологическим разбросом при изготовлении ЭС и ее элементов: разбросом расположения элементов в ЭС относительно друг друга, разбросом размеров, состояний поверхностей (волнистостей, неровностей, неплоскостностей), разбросом сопряжений элементов между собой, зазоров между ними, площадей контакта между элементами и т.д. Статистический технологический разброс проявляется также при монтаже, установке и сборке элементов в ЭС. Ко второй группе факторов неопределенности тепловых процессов относятся все факторы, возникающие в процессе функционирования ЭС (эксплуатации, измерениях, испытаниях), которые также носят интервально стохастический характер: энергопотребление элементов в ЭС, напряжения и токи электропитания, мощности тепловыделений, параметры среды внутри ЭС (температура, влажность, давление, скорости потоков среды),

теплообмен между элементами и теплоносителями в системе охлаждения (температуры на входах в каналы, скорости потоков, распределения тепловых потоков в каналах и пр.). Третью группу факторов неопределенности составляют интервально стохастические параметры окружающей среды (температура, влажность, давление, скорости среды), а также условий эксплуатации и установки ЭС включая неопределенность параметров воздействий различной физической природы, оказываемых на ЭС со стороны других технических систем.

Таким образом определяющие факторы в ЭС носят неопределенный интервально стохастический характер, что, в свою очередь, обусловливает интервально стохастический характер тепловых процессов в ЭС. Иначе говоря реальные температуры в различных точках и элементах ЭС, значения которых реализуются на практике, являются не точно и однозначно определенными, как это принято сейчас в моделировании тепловых процессов, а будут заключены внутри интервалов своего изменения, принимая в них любое значение в соответствии с их усеченным законом распределения вероятностей. Поэтому основной задачей моделирования тепловых процессов в реальных ЭС является математическое и компьютерное моделирование с целью определения границ интервалов изменения значений температуры в различных точках и элементах ЭС.

Отметим, что принятие специальных мер, направленных на снижение интервалов неопределенности определяющих факторов, путем стабилизации энергопитания, параметров окружающих условий (например, кондиционирование), ужесточение входного и выходного контроля, уменьшение допусков и др., – позволяет в ряде случаев сократить ширину интервалов изменения определяющих факторов, однако, следует сознавать, что неопределенность носит принципиальный и неустранимый характер.

Ввиду актуальности рассматриваемой проблемы и ее важности для практических приложений и проектирования ЭС, в существующей литературе имеется достаточно много работ, посвященных методам моделирования и анализа стохастических тепловых процессов и температурных распределений. Во многих работах рассматриваются методы определения стохастических распределений температуры в различных твердых телах со стохастическими граничными условиями (температура окружающей среды, коэффициент теплоотдачи, тепловой поток на границе) и случайной начальной температурой [11, 17, 19, 23, 26, 27]. В других работах рассматриваются методы определения стохастических температурных полей в потоке жидкости при различных стохастических факторах – скорости жидкости, температуры [12, 14, 20, 24]. Более общие случаи анализа стохастических температурных распределений, описываемые стохастическими уравнениями в обыкновенных или частных производных со случайными параметрами или процессами, численные методы их решения рассматриваются, например, в работах [18, 21, 25]. Как правило авторы стремятся получить замкнутые конечные дифференциальные уравнения (обыкновенные или в частных производных) для непосредственного определения статистических мер (математических ожиданий, дисперсий, ковариаций), которых обычно достаточно для практических целей, где не требуется знания законов распределения вероятностей. Причем уравнения для статистических мер могут быть как стационарными, так и нестационарными. Для достижения указанной цели в большинстве работ стохастические факторы (граничные условия, начальные температуры, теплофизические характеристики среды) задаются в виде таких стохастических процессов, как гауссовы белые шумы или винеровские процессы (броуновское движение). Между тем, указанные стохастические процессы не являются адекватными, поскольку в реальности не встречаются. Достаточно упомянуть тот факт, что белый шум имеет бесконечную дисперсию, а производная винеровского процесса по времени представляет собой белый шум. Но поскольку для стохастических процессов Винера имеется развитый математический аппарат (уравнения Ито, Колмогорова, Фокера-Планка, Ланжевена и др. [13, 15]) то его использование и позволяет авторам получать конечные дифференциальные уравнения для непосредственного определения статистических мер стохастических температурных полей. Следует, однако, сознавать, что эти уравнения, равно как и получаемые из них результаты не представляют интереса для практики моделирования тепловых процессов в ЭС, поскольку основаны на неадекватных реальности стохастических процессах, не встречающихся в природе.

Один из подходов к моделированию температурных полей с интервальной неопределенностью входных данных, связан с применением методов интервального анализа [1, 10], позволяющим находить интервальные температурные поля [2, 7]. Между тем получаемые при этом результаты не могут быть признаны удовлетворительными и адекватными реальности, что объясняется рядом принципиальных недостатков, не позволяющих рекомендовать его для моделирования температурных полей в условиях интервальной неопределенности. Во-первых, результаты вычислений, выполняемых по правилам интервальной арифметики, существенно зависят от последовательности преобразования алгебраических выражений, что вообще недопустимо. Во-вторых, найденные интервалы имеют неоправданно большой размах, который не имеет физического обоснования, а возникает лишь вследствие применения интервальной арифметики. По существу метод интервального анализа является расчетом на худший (лучший) случай, когда все определяющие факторы одновременно принимают свои наихудшие (наилучшие) значения с точки зрения получения наиболее высоких (низких) значений температур. Между тем, вероятность наступления события, при котором параметры одновременно принимают свои крайние значения ничтожно мала и потому является невозможным событием, не реализуемым на практике. Поэтому выводы и рекомендации, получаемые методами интервального анализа, не могут служить ориентиром при проектировании конкурентоспособных ЭС.

Адекватное моделирование интервально стохастических тепловых процессов и температурных полей в ЭС, должно осуществляться при условии, что определяющие факторы ЭС представляют собой случайные величины, принимающие свои значения внутри соответствующих интервалов и подчиняющиеся опреде-

ленным законам распределения вероятностей. Данный подход был реализован при тепловом проектировании компьютерных систем [16], с использованием метода Монте-Карло, электронных модулей [8, 9], с использованием композиции матрично-топологического метода [4, 6] и метода Монте-Карло для статистических реализаций нелинейностей.

Рассмотренные в [8, 9, 16] методы относятся к моделированию интервально стохастических стационарных температурных полей, в то время как тепловые процессы в ЭС носят нестационарный характер. Кроме того, в силу значительной зависимости электрических и теплообменных факторов в ЭС от температуры, тепловые процессы имеют существенно нелинейный характер. Поэтому адекватные практике математические и компьютерные методы и модели должны иметь возможность эффективно и с достаточной для практики точностью (не хуже 5 – 7 %) и на всех иерархических уровнях ЭС [5] осуществлять моделирование тепловых процессов, являющихся нестационарными, нелинейными, интервально стохастическими, при условии, что интервальные определяющие факторы в общем случае могут подчиняться произвольным усеченным законам распределения вероятностей и быть свободными от неоправданных нереальных допущений о стохастических процессах типа белого шума, или винеровских. В существующей литературе на сегодняшний день отсутствуют работы, в которых бы содержались методы и модели адекватного моделирования на всех иерархических уровнях ЭС нестационарных, нелинейных тепловых процессов в условиях интервально стохастической неопределенности определяющих факторов, подчиняющихся произвольным усеченным законам распределения вероятностей.

Математическая модель

Приведем математическую модель, описывающую нестационарные, нелинейные интервально стохастические тепловые процессы в ЭС.

Тепловые процессы в ЭС протекают на всех пяти иерархических уровнях ЭС и на каждом уровне иерархии конструкция ЭС может быть представлена как множество разнородных трехмерных элементов, находящихся в состоянии теплового взаимодействия между собой и окружающей средой [4, 5]. Сложная трехмерная область ЭС $G = \bigcup_{i=1}^{N} G_i$ имеет границу ∂G, состоящую из N разнородных подобластей G_i с границами ∂G_i, отображающими элементы ЭС. В общем случае все определяющие факторы в ЭС носят интервальный стохастический и нелинейный характер. Причем определяющие факторы могут иметь любое усеченной распределение вероятностей.

Интервально стохастический нестационарный тепловой процесс $T = T(x,t,\omega)$ в каждой точке $x = (x_1, x_2, x_3) \in G + \partial G$ области тепловой модели ЭС для каждого элементарного события $\omega \in \Omega$ описывается следующей стохастической математической моделью:

$$\rho c \frac{\partial T}{\partial t} + A(T,x,t,\omega)T = P(T,x,t,\omega),$$
$$(x,t,\omega) \in G \times [0,\tau] \times \Omega,$$

с условиями на границах области ЭС как вне, так и внутри $\partial G = \bigcup_{i=1}^{N} \partial G_i$ (n – нормаль к поверхности), включающими процессы конвективного и лучистого теплообмена

$$\lambda(x,\omega)\frac{\partial T}{\partial n} + \alpha(T,x,\omega)(T - T_a(x,t,\omega)) + \varepsilon\sigma T^4 = q(T,x,t,\omega),$$
$$(x,t,\omega) \in \partial G \times [0,\tau] \times \Omega,$$

условиями на границах ∂G_i и ∂G_j контакта между любыми контактирующими элементами i и j, выражающими равенства температур и тепловых потоков в месте контакта,

$$T\big|_{\partial G_i} = T\big|_{\partial G_j},$$
$$\lambda_i(x,\omega)\frac{\partial T}{\partial n}\bigg|_{\partial G_i} = \lambda_j(x,\omega)\frac{\partial T}{\partial n}\bigg|_{\partial G_j},$$
$$(x,t,\omega) \in \partial G_i \times \partial G_j \times [0,\tau] \times \Omega,$$

где $A(T,x,t,\omega)$ – стохастический, нестационарный, нелинейный оператор

$$A(T,x,t,\omega) = -\sum_{i=1}^{3} \frac{\partial}{\partial x_i}\left(\lambda(x_i,\omega)\frac{\partial}{\partial x_i}\right);$$

$\lambda(x,\omega)$ – случайные величины коэффициентов теплопроводности материалов элементов тепловой модели ЭС; $\alpha(T,x,\omega)$ – случайные величины коэффициентов конвективной теплоотдачи с поверхностей ЭС и ее элементов; ε и σ – степени черноты и постоянная Стефана-Больцмана; $q(T,x,t,\omega)$ – заданные случайные тепловые потоки на поверхностях ЭС и ее элементов; $T_a(x,t,\omega)$ – случайное распределение температуры среды внутри и вне ЭС; $P(T,x,t,\omega)$ – заданные случайные распределения объемных плотностей мощности внутренних источников тепла в ЭС. Случайные величины в векторе $\theta(\omega)$

$$\theta(\omega) = (\lambda(\omega), \alpha(\omega), q(\omega), T_a(\omega), P(\omega))$$

математической модели являются интервальными $\theta(\omega) \in [\underline{\theta}, \overline{\theta}]$ и подчиняются в общем случае усеченным законам распределения F произвольного вида в интервалах $[\underline{\theta}, \overline{\theta}]$ и равными нулю вне их, где

$$\underline{\theta}(\omega) = (\underline{\lambda}(\omega), \underline{\alpha}(\omega), \underline{q}(\omega), \underline{T}_a(\omega), \underline{P}(\omega)),$$

$\overline{\theta}(\omega) = (\overline{\lambda}(\omega), \overline{\alpha}(\omega), \overline{q}(\omega), \overline{T}_a(\omega), \overline{P}(\omega))$ – нижняя и верхняя границы интервалов для каждой из случайных величин в векторе $\theta(\omega)$.

Уравнения приведенной выше математической модели являются, вообще говоря, избыточными, и чрезвычайно сложными. Между тем для целей инженерного проектирования ЭС достаточно знания статистических мер стохастических тепловых процессов, а именно полей математических ожиданий, дисперсий и ковариаций. Поэтому разработка математических и компьютерных методов, позволяющих определять нестационарные и нелинейные статистические меры тепловых процессов ЭС, представляет собой чрезвычайно важную актуальную задачу. Создание на основе разработанных методов многофункционального программного комплекса для моделирования интервально стохастических, нестационарных, нелинейных статистических мер тепловых процессов на всех иерархических уровнях ЭС, предоставит разработчикам мощный и адекватный вычислительный инструмент, максимально приближенный к реальности, что позволит осуществлять проектирование и разработку сложных конкурентоспособных ЭС и высокопроизводительных вычислительных комплексов.

Modeling of interval stochastic unsteady and nonlinear thermal processes in electronic systems

A.G.Madera

Abstract. Thermal processes in a real electronic systems in general are uncertain, unsteady and non-linear. Now, however, their modeling is based on the assumption as of about determinates of determinants factors, and of linearity thermal processes. In the article are analyzed the state of researches and formulation of the problem of about creating adequate methods for mathematical and computer modeling of thermal processes in electronic systems that fully take into account the actual characteristics of electronic systems.

Keywords: thermal processes, electronic system, interval, stochastic, unsteady, nonlinear

Литература

1. Г. Алефельд, Ю. Херцбергер. Введение в интервальные вычисления. – М.: Мир, 1987.

2. Б.С. Добронец, В.С. Злобин. Численное моделирование температурных полей с интервлаьными неопределенностями // Сибирский журнал индустриальной математики. 2004. Том. VII. № 3(19). С. 95 – 101.

3. Конструкторско технологическое проектирование электронной аппаратуры / Под общ. редакцией В.А. Шахнова. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005, 568 с.

4. А.Г. Мадера. Моделирование теплообмена в технических системах. – М.: Науч. фонд «Первая исслед. лаб. им. акад. В.А. Мельникова», 2005

5. А.Г. Мадера. Иерархический подход при тепловом проектировании электронных изделий // Программные продукты и системы. 2008. № 4 (84). С. 10.

6. А.Г. Мадера, П.И. Кандалов. Матрично-топологический метод математического и компьютерного моделирования температурных полей в электронных модулях: программный комплекс STF-ELECTRONMOD // Программные продукты и системы. 2012. №4. С. 34.

7. А.Г. Мадера, П.И. Кандалов. Моделирование температурных полей технических систем в условиях интервальной неопределенности // Тепловые процессы в технике. 2014. № 5. С. 225 – 229

8. А.Г. Мадера, П.И. Кандалов. Анализ интервально стохастических температурных полей технических систем // Программные продукты и системы. 2014. №4. С. 41 – 45

9. А.Г. Мадера, П.И. Кандалов. Компьютерное моделирование температурных полей технических систем при интервально стохастической неопределенности параметров // Прикладная информатика. 2015. № 1 (55). С. 106 – 113

10. С.П. Шарый. Конечномерный интервальный анализ. – Новосибирск: СО РАН, 2012.

11. A. Campo, T. Yishimura. Random heat transfer in flat channels with timewise variation of ambient temperature // Int. J. Heat Mass Transfer. 1979. Vol. 22. P. 5 – 12.

12. R.F. Fox, R. Baracat. Heat conduction in a random medium // J. Stat. Phys. 1978. Vol. 18. No. 2. P. 171 – 178.

13. C. Gardiner. Stochastic methods. A Handbook for the Natural and Social Sciences – N.Y.: Springer, 2009

14. J.C. Georgiadis. On the approximate solution on non-deterministic heat and mass transport problems // Int. J. Heat Mass Transfer. 1991. V. 33. n. 8. P.2099 – 2105

15. N.G. Kampen van. Stochastic Processes in Physics and Chemistry. – North Holland: Elsevier. 2007

16. C.J. Keller, V.W. Antonetti. Statistical thermal design for computer electronics // Electronic Packaging and Production. 1979. V.19.n.3. P. 55 – 62.

17. A.G. Madera. Modelling of stochastic heat transfer in a solid // Applied Mathematical Modelling. 1993, T. 17, № 12. P. 664 – 668

18. A.G. Madera. Simulation of stochastic heat conduction processes // International Journal of Heat and Mass Transfer. 1994. T. 37. № 16. C. 2571 – 2577.

19. A.G. Madera. Heat transfer from an extended surface at a stochastic heat-transfer coefficient and stochastic environmental temperature // International Journal of Engineering Science. 1996. T. 34. № 9. C. 1093 – 1099.

20. A.G. Madera, A.N. Sotnikov. Method for analyzing stochastic heat transfer in a fluid flow // Applied Mathematical Modelling. 1996. T. 20. № 8. P. 588 – 592.

21. J. Padovan, Y.H. Guo. Solution of non-deterministic finite element and finite difference heat conduction simulations // Numer. Heat Transfer A. 1989. Vol. 15, P. 383 – 398.

22. M. Perlmutter. Heat transfer in a channel with random variations in fluid velocity // Proceedings of the

23. J.C. Samuels. Heat conduction in solids with random external temperatures and / or random internal heat generation // Int. J. Heat Mass Transfer. 1966. Vol. 9. P. 301 – 314.

24. S.E. Serrano, T.E. Unny, W.C. Lennox. Analysis of stochastic ground water flow problems // J. Hydrol. 1985. Vol. 82. P. 285 – 306.

25. S.E. Serrano, T.E. Unny. Random evolution equations in hydrology // Appl. Math. Comput. 1990. Vol. 38. P. 201 – 226.

26. Tzow Da Yu. Stochastic analysis of temperature distribution in a silid with random heat conductivity // Trans. ASME, J. Heat Transfer. 1988. Vol. 110, P. 23 – 29.

27. T. Yoshimura, A. Campo. Extended surface heat rejection accounting for stochastic sink temperatures // AIAA J. 1981. Vol. 19. P. 221 – 225.

2D температурная модель распределения потенциала в двух затворных полностью обедненных нанотранзисторах со структурой «германий на изоляторе»

Н.В. Масальский

кандидат физико-математических наук

Аннотация: Разработана 2D аналитическая температурная модель распределения потенциала в рабочей области двух затворного полностью обедненного полевого нанотранзистора со структурой «германий на изоляторе». В параболическом приближении функции распределения потенциала в рабочей области получено решение 2D уравнения Пуассона. Численно исследована температурная зависимость распределения поверхностного потенциала в диапазоне температур от 200 К до 500 К от ряда технологических параметров. Показано, что для повышения верхней границы температурного диапазона на 100 К необходимо увеличивать уровень легирования рабочей области на порядок.

Ключевые слова: потенциал, нанотранзистор, уравнение Пуассона

Введение

На современном этапе развития микроэлектроники формируется новое направление – создание на наноразмерных структурах «германий на изоляторе» высокоэффективных микросхем для сверхвысокопроизводительных вычислительных систем [1-3]. В настоящее время во всех ведущих мировых исследовательских центрах ведутся поисковые исследования теоретических принципов моделирования, включая такой важный этап, как разработка аналитических моделей, применимых для схемотехнического моделирования, и проектирования сверхбольших интегральных схем на структурах «германий на изоляторе».

Двух затворные полевые транзисторы - одна из самых многообещающих архитектур для реализации рубежей, заявленных в новом направлении [1, 2, 4-6]. Двух затворная архитектура обладает уникальными возможностями для масштабирования микросхем в наноразмерной области [4, 5, 7]. В настоящее время получены практические результаты, отражающие достижения аналитического моделирования данного типа транзистора. Однако ни один из современных подходов не рассматривает температурную зависимость транзисторных параметров. Например, изменение рабочей температуры транзистора приводит, в частности, к изменению подвижности носителей, их скорости насыщения, эффекту модуляции длины канала и пороговому напряжению, что приводит к снижению тока транзистора [8-10].

В данной работе обсуждается подход аналитического моделирования температурной зависимости 2D распределения потенциала в рабочей области двух затворного нанотранзистора со структурой «германий на изоляторе», поскольку, как известно, распределение потенциала определяет ключевые характеристики транзистора.

1. Распределение потенциала: квазидвумерная модель

В рамках концепции зарядового разделения [11] рассмотрим квазиклассическую задачу определения 2D распределения потенциала в тонкопленочной двух затворной транзисторной структуре «германий на изоляторе». В мало-сигнальном приближении и приближении «плавного канала»: однородного распределения легирующей примеси в его рабочей области - рассмотрим случай n-канального транзистора. В общем случае необходимо решать самосогласованную задачу, связанную с нахождением распределения потенциала 2D потенциала рабочей области $\varphi(x,y,T)$, где оси x и y выбраны так: x – ортогонально поверхности раздела рабочая область - фронтальный подзатворный окисел в направлении погруженного окисла, y – вдоль поверхности раздела рабочая область – подзатворный фронтальный окисел. 2D уравнение Пуассона в рабочей области рассматриваемой структуры имеет вид [10]:

$$\frac{\partial^2\varphi(x,y,T)}{\partial x^2}+\frac{\partial\varphi(x,y,T)}{\partial y^2}=-\frac{q}{\varepsilon_S}N_A, \qquad (1)$$

где q – заряд электрона, T – температура, ε_S - диэлектрическая проницаемость рабочей области, N_A – концентрация легирования рабочей области. Следует отметить, что в данном подходе не учитывается возможный двухмерный характер распределения потенциала в подзатворном диэлектрике (погруженном окисле) обратного затвора.

Решение уравнения Пуассона аппроксимируется степенным полиномом с коэффициентами, зависящими от y:

$$\varphi(x,y,T)=\sum_{m=0}^{N}a_m(y,T)x^m. \quad (2)$$

При таком выборе решения уравнения (1) необходимо определить функции $a_m(y,T)$ (m=0,1,2), которые связаны тремя условиями по координате x. Электрические поля на границах связаны с напряжениями на фронтальном и обратном затворах следующими соотношениями [10]:

$$\left.\frac{\partial\varphi(x,y,T)}{\partial x}\right|_{x=0}=\frac{C_f}{\varepsilon_S}[\varphi_f(y,T)-U_{Gf}+U_{FB_f}(T)],$$
$$\left.\frac{\partial\varphi(x,y,T)}{\partial x}\right|_{x=t_S}=\frac{C_b}{\varepsilon_S}[U_{Gb}-U_{FB_b}(T)-\varphi_b(y,T)]. \quad (3)$$

где t_S – толщина рабочей области, U_{Gf} , U_{Gb} – напряжения на фронтальном и обратном затворах, C_f , C_b – емкости фронтального затвора и обратного затвора соответственно, U_{FB_f} , U_{FB_b} – напряжения плоских зон для фронтального и обратного затворов.

Следуя [12] напряжение плоских зон $U_{FB_f}=\Phi_{MS_f}-(\chi_f+\frac{E_g(T)}{2q}+U_F(T))$, где Φ_{MS_f} - работа выхода фронтального затвора, χ_f - сродство электрона, $E_g(T)$ - ширина запрещенной зоны, $U_F(T)$ - уровень Ферми. Температурная зависимость ширина запрещенной зоны $E_g(T)=E_g(0)-g_{E_g}(\frac{T^2}{T+T_0})$, где $E_g(0)$ - ширина запрещенной зоны при 0 К, g_{E_g} - температурный градиент ширины запрещенной зоны, T_0 - начальная температура (оба последних параметра являются по сути подгоночными и определяются экспериментально). Температурная зависимость уровня Ферми $U_F(T)=U_t\ln\frac{N_A}{n_i(T)}$, где $U_t=\frac{kT}{q}$ - тепловой потенциал, k – константа Больцмана,

$n_i(T)=\sqrt{N_c(300)N_v(300)}(\frac{T}{300})^{3/2}\exp(-\frac{E_g(T)}{2kT})$ - собственная концентрация носителей, $N_c(300)$ и $N_v(300)$ - плотность поверхностных состояний в зоне проводимости и валентной зоне при температуре 300 К, соответственно.

Граничные условия для данного уравнения имеют следующий вид:

$$\varphi_f(0,T)=u_{bi}(T)$$
$$\varphi_f(L_g,T)=u_{bi}(T)+U_{ds}, \quad (4)$$

где $u_{bi}(T)=\frac{kT}{q}\ln(\frac{N_AN_{ds})}{n_i^2(T)})$ – контактная разность потенциалов, U_{ds} – напряжение сток-исток, L_g - длина затвора, N_{ds} - максимальная концентрация легирования областей стока и истока..

Из условий (2)-(3) можно получить уравнения для определения коэффициентов $a_m(y)$:

$$a_o(y,T)=\varphi_f(y,T)$$
$$a_1(y,T)=\frac{C_f}{\varepsilon_S}[\varphi_f(y,T)-U_{Gf}+U_{FB_f}(T)]$$
$$a_o(y,T)+a_1(y,T)t_S+a_2(y,T)t_S^2=\varphi_b(y,T) \quad (5)$$

Дифференцируя решение (2) по x с учетом (5), получаем выражение для $a_2(y,T)$:

$$a_2(y,T)=\frac{-(C_f+C_b+\frac{C_fC_b}{C_S})\varphi_f(y,T)}{(C_b+2C_S)t_S^2}+$$
$$+\frac{C_f(1+\frac{C_b}{C_S})(U_{Gf}-U_{FB_f}(T))+C_b(U_{Gb}-U_{FB_b}(T))}{(C_b+2C_S)t_S^2} \quad (6)$$

где C_S – емкость рабочей области.

Подставляя решение (2), (5), (6) в уравнение (1), получим дифференциальное уравнение, положив в котором $x = 0$, перейдем к уравнению для потенциала на фронтальной поверхности:

$$\frac{d^2\varphi_f(y,T)}{dy^2}-\frac{2(C_f+C_b+\frac{C_fC_b}{C_S})}{(C_b+2C_S)t_S^2}\varphi_f(y,T)=A_f(T) \quad (7)$$

где

$$A_f(T)=\frac{q}{\varepsilon_S}N_A-$$
$$-2\frac{C_f(1+\frac{C_b}{C_S})(U_{Gf}-U_{FB_f}(T))+C_b(U_{Gb}-U_{FB_b}(T))}{(C_b+2C_S)t_S^2}$$

Учитывая тот факт, что граничные условия для уравнения (7) такие же, что и для уравнения Пуассона общего вида, выражение для поверхностного потенциала имеет вид:

$$\varphi_f(y,T) = \frac{(u_{bi}(T) + l^2 A_f(T))(\exp(\frac{L_g}{l}) - 1) - U_{ds}}{2sh(\frac{L_g}{l})} \times$$

$$\times \exp(-\frac{y}{l}) +$$

$$\frac{U_{ds} - (u_{bi}(T) + l^2 A_f(T))(\exp(-\frac{L_g}{l}) - 1)}{2sh(\frac{L_g}{l})} \exp(\frac{y}{l}) -$$

$$- l^2 A_f(T)$$

(8)

где l - характеристическая длина и $l = t_S \sqrt{\frac{(1 + 2\frac{C_S}{C_b})}{2(1 + \frac{C_f}{C_S} + \frac{C_f}{C_b})}}$.

Тогда в соответствии с общим видом решения и соотношениями для функций $a_m(y,T)$, распределение потенциала в рабочей области транзистора подчиняется выражению:

$$\varphi(x,y,T) = (\frac{(u_{bi}(T) + l^2 A_f(T))(\exp(\frac{L_g}{l}) - 1) - U_{ds}}{2sh(\frac{L_g}{l})} \times$$

$$\times \exp(-\frac{y}{l}) +$$

$$+ \frac{U_{ds} - (u_{bi}(T) + l^2 A_f(T))(\exp(-\frac{L_g}{l}) - 1)}{2sh(\frac{L_g}{l})} \exp(\frac{y}{l}) - \quad (9)$$

$$- l^2 A_f(T))(1 + \frac{C_f}{\varepsilon_f} x - \frac{x^2}{l^2}) + x \frac{C_f}{\varepsilon_S} (U_{FB_f}(T) - U_{Gf})$$

$$+ \frac{A_f(T)}{2} x^2$$

2. Результаты моделирования

2.1 Температурные зависимости основных физических величин

Температурная зависимость распределения потенциала в основном определяется температурными зависимостями таких физических величин, как ширина запрещенной зоны, собственная концентрация, встроенная разность потенциалов и напряжение плоских зон [8].

Температурная зависимость ширина запрещенной зоны для структуры «германий на изоляторе» приведена на рис. 1 , где для сравнения представлена аналогичная зависимость для структуры КНИ.

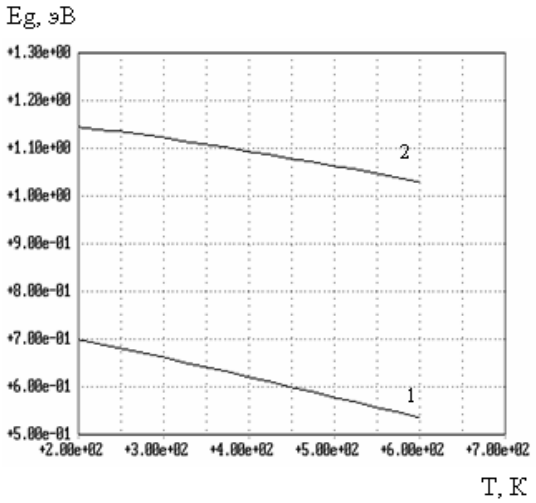


Рис. 1. Температурная зависимость запрещенной зоны, где 1 – германий, 2 – кремний

Такой вид зависимости близкой к линейной в исследуемом диапазоне температур обусловливается температурным изменением взаимодействия носителей с решеткой и температурной зависимостью распределения фононов вследствие линейных колебаний решетки [9].

Температурная зависимость собственной концентрации обоих структур имеет ярко выраженный нелинейный характер (см. ниже рис. 2).

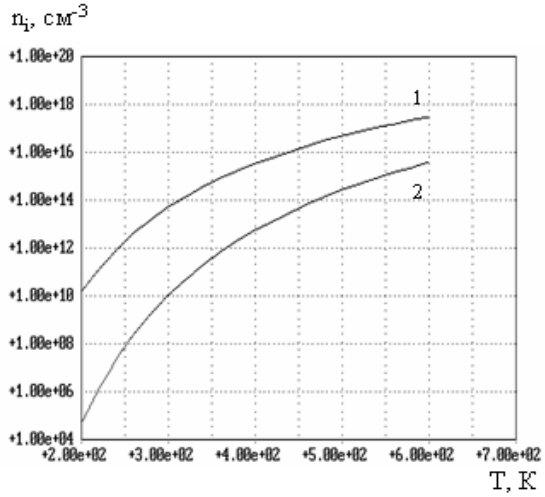


Рис. 2. Температурная зависимость собственной концентрации, где 1 – германий, 2 – кремний

Следует отметить, что с ростом температуры величина собственной концентрации резко возрастает. Она каждый раз удваивается при повышении температуры примерно на 10 К. Поэтому при высоких температурах процесс термогенерации становится определяющим для концентрации носителей.

Положение уровня Ферми определяется уровнем собственной концентрации носителей. Поскольку, при

повышении температуры уровень собственной концентрации растет быстрее, чем тепловой потенциал, следовательно, уровень Ферми будет приближаться к нижней границе запрещенной зоны, что иллюстрируется результатами расчетов, приведенными на рис 3. Аналогичным поведением характеризуется и контактная разность потенциалов, представленная на рис 4.

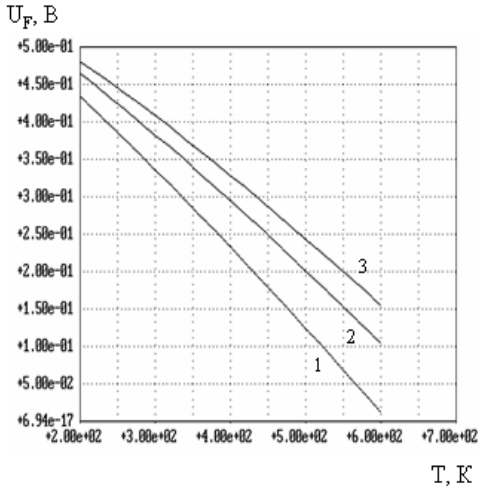


Рис. 3а. Температурная зависимость положения уровня Ферми в кремнии при разной концентрации легирования рабочей области, где
1 - N_A=5.5x10^{15} см$^{-3}$, 2 - N_A=3.0x10^{16} см$^{-3}$, 3 - N_A=8.0x10^{16} см$^{-3}$

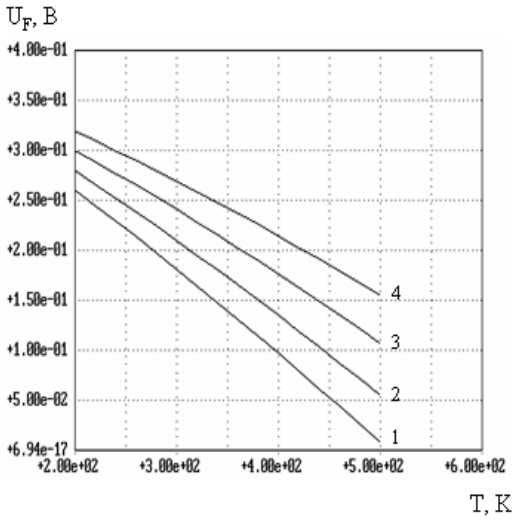


Рис. 3б. Температурная зависимость положения уровня Ферми в структуре германий на изоляторе, где
1- N_A = 2.5x10 ^{16} см$^{-3}$, 2- N_A = 7.5x10 ^{16} см$^{-3}$, 3- N_A = 2.5x10 ^{17} см$^{-3}$, 4- N_A = 7.5x10 ^{17} см$^{-3}$,
2- N_{ds} = 5x10 ^{19} см$^{-3}$

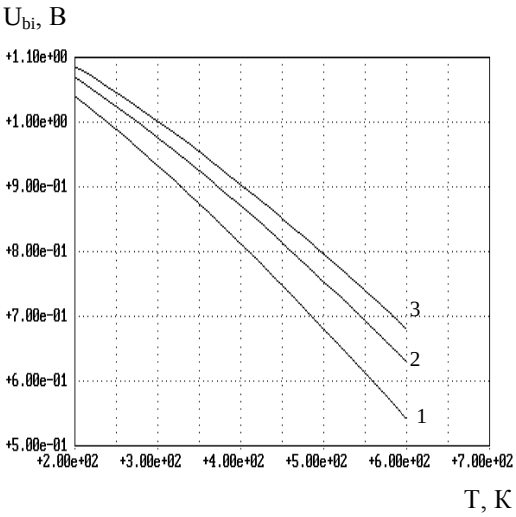


Рис. 4а. Температурная зависимость контактной разности потенциалов в кремнии при тех же концентрациях, что и на рис 3а.

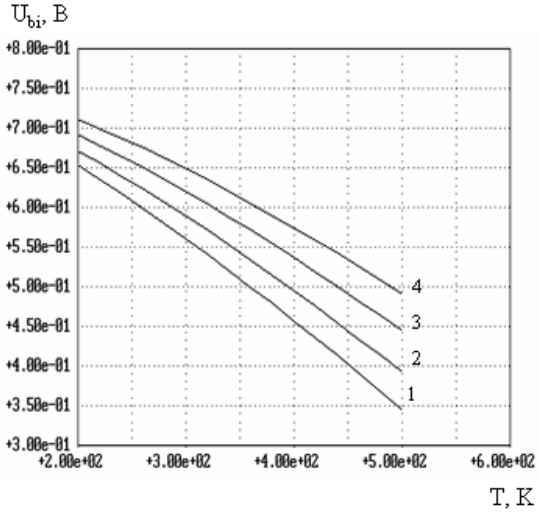


Рис. 4б. Температурная зависимость контактной разности потенциала в структуре германий на изоляторе при тех же концентрациях, что и на рис. 3б

Важно отметить, что характер поведения температурных зависимостей основных физических величин анализируемых структур одинаков.

2.2. Температурные зависимости распределения поверхностного потенциала

Для численных экспериментов был выбран прототип двух затворного полностью обедненного транзистора n-типа со структурой «германий на изоляторе». Параметры транзисторной структуры: длина затвора L_g = 45 нм, t_s = 8 нм, t_f = 2.5 нм, t_b = 100 нм, N_A = 2.5x10^{16} см$^{-3}$, N_{ds} = 5x10^{19} см$^{-3}$, где t_f и t_b – толщины подзатворных окислов фронтального и обратного затворов, соответственно. При помощи соотношений (3, 4, 7, 8) для малосигнального случая при U_{ds}=U_{Gf}=0.1 В U_{Gb} = 0 В вычислены значения фронтального поверхностного потенциала для разных значений температур. Рис. 5 иллюстрирует

распределение данного потенциала вдоль длины канала в диапазоне температур от 200 до 500 К.

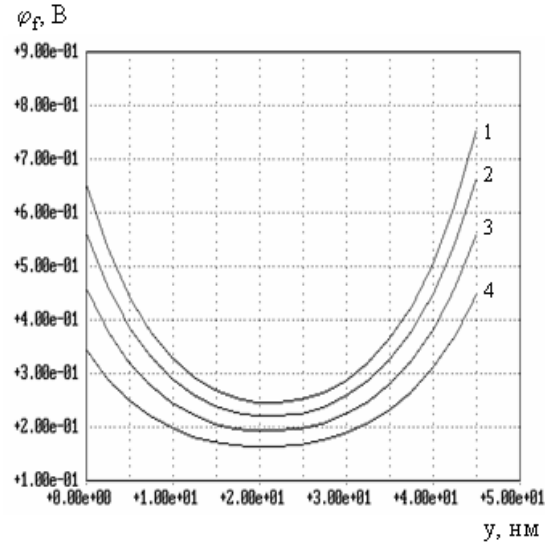


Рис. 5. Температурная зависимость распределения фронтального поверхностного потенциала, где 1-200 К, 2-300 К, 3-400 К, 4-500 К

Такой вид распределения поверхностного потенциала обусловлен температурной зависимостью $U_{bi}(T)$, в результате которой абсолютное значение фронтального поверхностного потенциала снижается с повышением температуры. Однако, его нормированная величина возрастает с ростом температуры во всех точках канала, что иллюстрируется рис. 6.

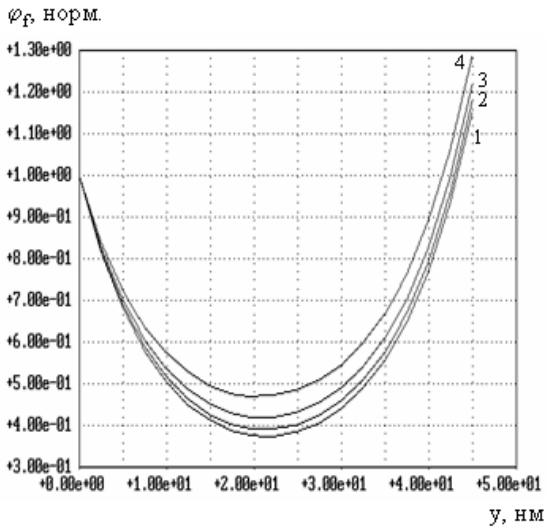


Рис. 6. Нормированная температурная зависимость распределения фронтального поверхностного потенциала, где 1-200 К, 2-300 К, 3-400 К, 4-500 К

Это связано с тем, что с повышением температуры увеличивается собственная концентрация носителей, а уровень Ферми понижается. Из результатов моделирования следует, что в исследуемом диапазоне температур зависимость величины изменения потенциала от температуры в любой точке канала носит нелинейный характер.

Этот вывод необходимо отнести также и к поведению температурной зависимости минимума поверхностного потенциала. Функция минимума поверхностного потенциала, как известно, используется в модели порогового напряжения, причем с учетом коротко-канальных эффектов. Так же полезным результатом для моделирования ключевых характеристик нанотранзисторов является температурная зависимость минимума поверхностного потенциала для разных концентраций легирования рабочей области, представленная на рис. 7

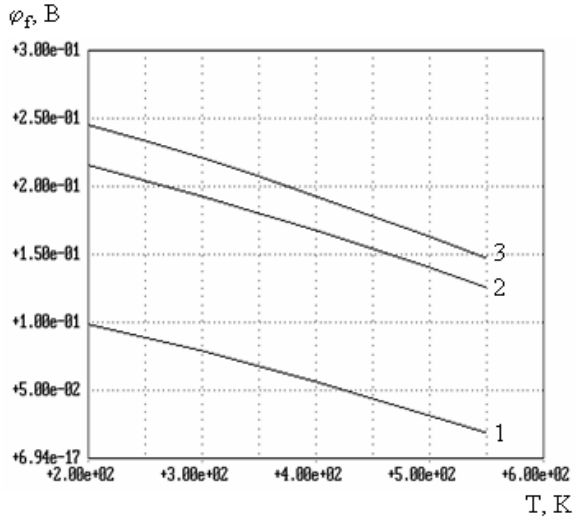


Рис. 7. Температурная зависимость минимума поверхностного потенциала для разных концентраций легирования канала, где 1- $N_A = 2.5x10^{16}$ см$^{-3}$, 2- $N_A = 7.5x10^{16}$ см$^{-3}$, 3 - $N_A = 2.5x10^{17}$ см$^{-3}$

Из представленных результатов расчетов следует простое правило. Для повышения верхней границы температурного диапазона на 100 К необходимо увеличивать уровень легирования рабочей области на порядок.

Следует отметить, что результаты расчетов по представленной модели, но для случая КНИ устройств в силу вышеупомянутой схожести материалов рабочей области сопоставлялись с результатами моделирования выполненными другими авторами для КНИ структур с аналогичной архитектурой [13, 14]. Сразу отметим, что рассогласование результатов во всех случаях было менее 5%. В [13] анализировались длинно-канальные КНИ структуры, однако следует подчеркнуть, что авторы сопоставляли свои результаты с моделированием выполненными при помощи программного ракета SILVACO TCAD [15] и всегда получали хорошее согласование менее 10%. В [14] исследовались КНИ нанотранзисторы, но по непонятным причинам авторы считали напряжение плоских зон величиной не зависящей от температуры. При соответствующей модернизации нашей методики расчетов и было получено хорошее согласование результатов моделирования.

Заключение

Представлена 2D аналитическая температурная модель распределения потенциала в рабочей области двух затворного полностью обедненного полевого нанотранзистора со структурой «германий на

изоляторе». В диапазоне температур от 200 К до 500 К численно исследованы температурные зависимости распределения основных физических характеристик транзисторной структуры и поверхностного потенциала от ряда технологических параметров. Показано, что для повышения верхней границы температурного диапазона на 100 К необходимо увеличивать уровень легирования рабочей области на порядок.

Получено хорошее согласование результатов численных расчетов для КНИ нанотранзисторов с результатами моделирования выполненными другими авторами, в частности, при помощи программного ракета SILVACO TCAD.

2D temperature model of potential distribution for double gate fully depleted nanoransistors on structure «germanium on insulator»

N.V. Masalsky

Abstract: The 2D analytical temperature potential distribution model for operation area of double gate fully depleted field nanotransistor on structure of "germanium on insulator" is developed. In parabolic approximation of a potential distribution function in operation area the solution 2D Poisson equation is received. Numerically temperature dependence of distribution of surface potential at the temperature range from 200 K to 500 K from a row of technological parameters is probed. It is shown that for increase of upper bound of the temperature range on 100 K it is necessary to increase the level of an alloying of operation area much.

Keywords: potential, nanotransistor, Poisson equation

Литература

1. URL: http://public.itrs.net/International technology roadmap for semiconductor 2014 edition.
2. H. Shang, M. M Frank, E. P. Gusev, J. O. Chu, S. W. Bedell, K. W. Guarini, M. Ieong. Germanium channel MOSFETs: opportunities and challenges // IBM J. Res. Develop.- № 4(50), 2006. P. 377–386
3. A. Dimoulas, E. Gusev, P.C. Mclntyre, M. Heyns. Advanced gate stacks for high mobility semiconductors.- Hardcover :Springer, 2007
4. A. Kranti, G. A. Armstrong. Engineering source/drain extension regions in nanoscale double gate (DG) SOI MOSFETs: Analytical model and design considerations // Solid-State Electron. - № 2(50), 2006. P. 437 - 447
5. P. Batude, X. Garros, L. Clavelier, C. Le Royer, J. M. Hartmann, V. Loup, P. Besson, L. Vandroux, Y. Campidelli, S. Deleonibus, F. Boulanger. Insights on fundamental mechanisms impacting Ge metal oxide semiconductor capacitors with high-*k*/metal gate stacks // J. Appl. Phys. -№3(102), 2007. P. 514-523
6. Н.В. Масальский. Моделирование распределения потенциала в рабочей области полевого транзистора со структурой «германий на изоляторе»: Аналитическая модель и ее приложения // Программные продукты и системы. - № 4, 2014. С. 17-24
7. M. Chan, P. Su, H. Wan. Modeling the floating-body effects of fully depleted, partially depleted and body-grounded SOl MOSFETs // Solid-State Electron.- №6(48), 2004. P. 969-978
8. R. Gharabagi. A model for fully depleted double gate SOI CMOS transistors including temperature effects // Modeling and Simulation of Microsystems.- №2(5), 2001. P. 490-498
9. T. Rudenko, V. Kikhytska, J. P. Colinge. On the high-temperature subthreshold slope of thin-film SOI MOSFETs // IEEE Electron Device Lett.- №3(23), 2002. P. 148-150
10. A. K. Goel, T. H. Tan. High temperature and self-heating effects in fully depleted SOI MOSFETs // Microelectronics Journal. - №4(37), 2006. P. 963-971
11. J.-P. Colinge. Silicon Insulator Technology: Materials to VLSI. - Boston, Dordrecht, London: Kluwer Acad. Publ., 1997
12. С. Зи. Физика полупроводниковых приборов. – М.: Мир, 1984
13. W. Jin W, P. Chan, J. Lau. A physical thermal noise model for SOI MOSFET // IEEE Trans Electron Devices.- №6(47), 2000. P. 768-772
14. R. Sharma. Analytical modeling of volume inversion and channel length modulation in fully depleted double gate nanoscale SOI MOSFETs // Journal of Electron Devices. - №9(18), 2013. P. 1553-1563
15. ATLAS user's manual. SILVACO International.2006

Оценка годности кристаллов при помощи искусственных нейронных сетей

Е.Н. Ефимов

Аннотация: В статье рассмотрена задача оценки годности кристаллов при помощи нейронных сетей на основе анализа результатов измерений электрических параметров и проведения функционального контроля кристаллов при нормальных климатических условиях и результатов измерения микросхем той же партии при повышенной рабочей температуре среды. Показано, что нейронная сеть позволяет с высокой точностью установить связь между данными измерениями, за счет чего выполнить оценку количества брака. Экспериментальные данные, свидетельствуют об успешной идентификации более 90% кристаллов, забракованных при контроле в составе микросхемы при повышенной рабочей температуре среды, на основе результатов измерений кристалла при нормальных климатических условиях.

Ключевые слова: нейронные сети, кристалл, температура

Введение

В данной работе рассматривается задача оценки годности кристаллов СБИС на основе анализа результатов измерений их электрических параметров и функционального контроля на общей пластине при нормальных климатических условиях и результатов соответствующих измерений этих же кристаллов, собранных в корпуса, проведенных при повышенной рабочей температуре окружающей среды. На рисунке 1 показана упрощенная схема программ измерений пластин и закорпусированных кристаллов (микросхем). В целях минимизации затрат возможны две стратегии измерения:

1) измерение пластины при нормальной (+25°C) температуре, затем при повышенной (+125°C) температуре, затем корпусирование;

2) измерение пластины только при нормальной температуре, затем корпусирование.

Рис. 1. Упрощенная блок-схема программ измерений пластин и микросхем.

Вне зависимости от выбора стратегии конечной целью является получение микросхем, работающих во всем диапазоне рабочих температур. Первая стратегия позволяет определить заведомо сбойные при повышенной рабочей температуре кристаллы и не выполнять их корпусирование, что позволяет сэкономить время на корпусирование и, естественно, корпуса. Однако первая стратегия требует, по сравнению со второй, дополнительной операции измерения. Такая операция так же имеет эквивалентную цену, включающую стоимость работы оборудования, временные и трудозатраты.

Вторая стратегия предполагает измерение только при нормальных условиях и корпусирование всех годных кристаллов, некоторые кристаллы, неработающие при повышенной рабочей температуре так же будут закорпусированы и отбракованы на этапе измерения микросхем. Преимуществом данной стратегии является отсутствие дополнительной измерительной операции, за счет чего достигается экономия ресурсов. Однако выполнение корпусирования заведомо сбойных кристаллов так же сопряжено с трудозатратами и расходами на корпуса.

На основании экспертных оценок приемлем следующий подход: если количество забракованных микросхем при повышенной рабочей температуре окружающей среды велико, то стоимость корпусов может превышать стоимость трудозатрат на дополнительную измерительную операцию и целесообразно применение стратегии номер один; если же количество брака мало, то для минимизации расходов целесообразно применение стратегии номер два. Схематично данная ситуация показана на рисунке 2: для зоны A целесообразна стратегий №1, для зоны B – стратегия №2.

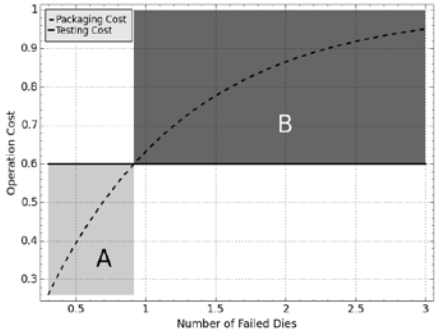


Рис. 2. Стоимость выполнения операций в зависимости от количества брака на пластине.

В связи с вышесказанным для производства является актуальной задача оценки работоспособности кристаллов при повышенной рабочей температуре. В данной работе показано, что эту оценку можно провести, обладая историей результатов измерений кристал-

лов при нормальной температуре и историей результатов измерений микросхем при повышенной рабочей температуре при помощи искусственных нейронных сетей.

1. Нейросетевая обработка данных

Нейронные сети представляют собой математические модели, построенные по аналогии с биологическими нейронными сетями [1]. В настоящее время искусственные нейронные сети широко применяются для решения ряда практических задач, в том числе задач кластеризации и категоризации [2].

Существует несколько типов нейронных сетей, одним из них является сеть типа многослойного персептрона, применяется в данной работе. Многослойный персептрон (Multilayer Perceptron – MLP) как и прочие нейронные сети обладает возможностью работать с зашумленными данными, обобщая из (generalization property).

На практике алгоритм применения нейронной сети включает следующие шаги.

1. Выборка входных данных.
2. Нормализация данных.
3. Разделение выборки на «Контрольную выборку» и «Обучающую выборку».
4. Обучение нейронной сети.
5. Применение обученной нейронной сети.

Применяемая в данной работе нейронная сеть прямого распространения класса MLP имеет структуру, представленную на рис 3. И состоит трех слоев:

- входной слой – элементы L_{1*};
- скрытый слой – элементы L_{2*};
- выходной слой – элемент L_3.

Каждый слой состоит из адаптивных элементов – нейронов, выполняющих определенные функциональные преобразования над входными сигналами. В свою очередь каждый из нейронов обладает определенной функцией активации и набором весовых коэффициентов. Особенности строения и функционирования нейронных сетей подробно описаны в [1]. В данной работе внимание сконцентрировано на их практическом применении.

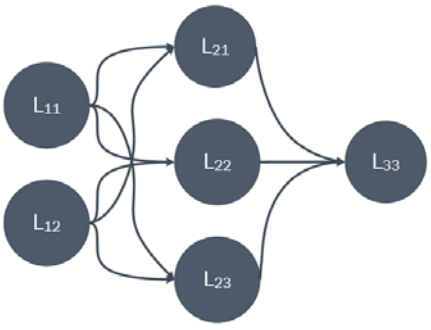


Рис. 3 Структура искусственной нейронной сети класса «Многослойный персептрон».

1.1. Применение искусственных нейронных сетей

В общем случае нейронная сеть рассматривается как система, выполняющая преобразование над входным вектором данных, и формирующая выходной вектор данных:

$$\mathbf{y}(\mathbf{x}) = T(\mathbf{w}, \mathbf{x}), \quad (1)$$

где $\mathbf{y}$ – выходной вектор данных, $\mathbf{x}$ – входной вектор данных, T – функциональное преобразование (передаточная функция системы), $\mathbf{w}$ – синоптические веса (параметры нейронов).

В качестве вектора входных данных $\mathbf{x}$ выступает вектор результатов измерений кристалла при нормальной температуре. В качестве вектора выходных данных выступает степень годности соответствующей микросхемы при повышенной рабочей температуры по правилу:

$$\mathbf{y}_i = \begin{cases} \{1\}, i \in P \\ \{0\}, i \notin P \end{cases}, \quad (2)$$

где $\mathbf{y}_i$ –выходной вектор для i-ого кристалла, P – подмножество кристаллов, соответствующие которым микросхемы не функционируют при повышенной температуре. Таким образом, решается задача категоризации. Определено две категории кристаллов: кристаллы, соответствующие которым микросхемы функционируют при повышенной температуре, и которые не функционируют при повышенной температуре.

Задача сводится к формированию системы, принимающей на вход результаты измерения кристалла при нормальной температуре и выдающей на выходе оценку работоспособности соответствующей ему микросхемы при повышенной температуре. Обладая определенной историей измерений - экспериментальными данными - можно синтезировать искусственную нейронную сеть, обучить ее при помощи этих данных и использовать для оценки. При этом нейронная сеть выявит сокрытую в данных зависимость и обобщит имеющиеся экспериментальные данные.

Как показано в [3, 4], для решения задачи категоризации подходящей функцией активации для нейронов скрытого и выходного слоев является сигмоида, показанная на рисунке 4.

$$y(x) = \frac{1}{1+e^{-x}} \quad (3)$$

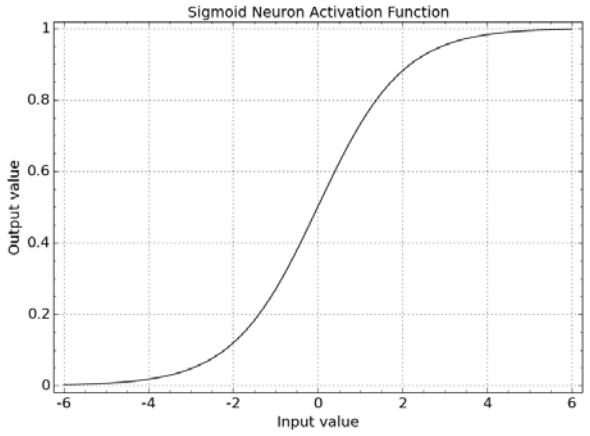


Рис. 4 Функция активации нейронов скрытого и выходного слоев.

2. Экспериментальные данные

В данной работе предложенный подход использован для анализа экспериментальных данных, полученных при измерении кристаллов СБИС на 5 пластинах и закорпусированных кристаллов. Общее количество отдельных результатов измерений – 46412. Общее количество годных кристаллов – 283. При этом для трех пластин известны результаты измерения соответствующих микросхем при повышенной рабочей температуре среды, для оставшихся двух такие результаты полагаются неизвестными. Результаты измерения микросхем, собранных из кристаллов с первых трех пластин приведены в таблице 1.

Таблица 1.

Пластина	Количество забракованных микросхем
Пластина №1	27
Пластина №2	14
Пластина №3	8

В соответствии с рекомендуемыми в [3] методиками обучения все кристаллы распределены изначально по трем группам:

- *Обучающая выборка (Training Set)* – группа из 117 кристаллов, выбранных случайно из кристаллов первых трех пластин; для этих кристаллов известны как результаты измерений при нормальных условиях, так и результаты функционального контроля соответствующих микросхем (т.е. принадлежность к одной из определенных ранее категорий). Пары входных и выходных данных из этой выборки принято называть *шаблонами обучения* [5];
- *Контрольная выборка (Testing Set)* – группа из 59 кристаллов, выбранных случайно из кристаллов первых трех пластин и не входящих в обучающую выборку; для кристаллов данной группы так же известны все результаты, однако данные этой выборки не учувствуют в обучении и используются для оценки обобщающих свойств сети;
- *Рабочая выборка (Target Set)* – кристаллы четвертой и пятой пластин, для которых известны только результаты измерений при нормальных климатических условиях. Ставится задача оценки количества брака при повышенной рабочей температуре в этой группе. В зависимости от этой оценки предполагается сделать выбор в пользу первой, либо второй стратегии измерения пластин.

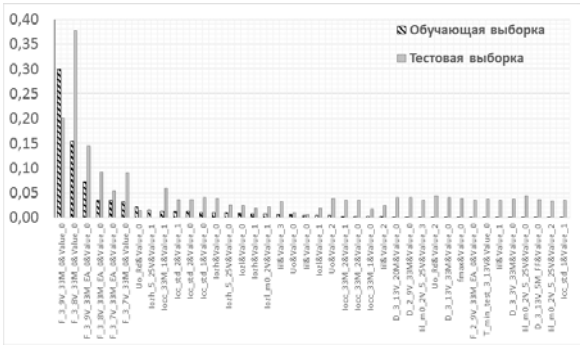


Рис. 5. Оценка парциального вклада параметров кристаллов в результат категоризации.

Каждый кристалл при измерении при нормальной температуре характеризуется 54 числовыми параметрами, что делает на практике невозможным решение поставленной задачи вручную экспертом.

Синтез нейронной сети для вектора входных данных из 54 элементов является достаточно ресурсоемкой задачей, ввиду этого выполнена оценка парциального вклада параметров в результат категоризации при помощи выражения:

$$p_i = -\ln\left(\sum_{j=0}^{N}\left(z_{j,d} - z_{j,a}\right)^2\right), \qquad (4)$$

где p_i – оценка производительности нейронной сети при использовании только i-ого параметра кристалла, N – количество шаблонов обучающей выборки, $z_{j,d}$ – требуемое выходное значение нейронной сети для j-ого шаблона обучения, $z_{j,a}$ – фактическое выходное значение нейронной сети для j-ого шаблона обучения.

После формирования шаблонов обучения для каждого кристалла обучающей выборки на основе единственного параметра было поведено обучение нейронной сети по методу градиентного спуска. После 512 итераций обучения проведена оценка производительности сети по формуле (4). Результаты приведены на рисунке 5.

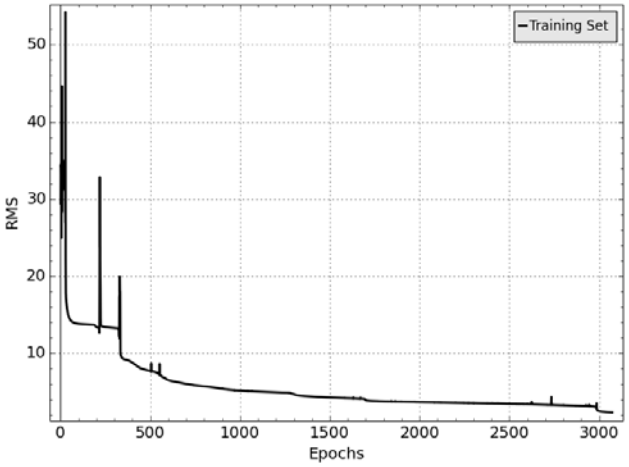


Рис. 6. Изменение значения СКО в процессе обучения для обучающей выборки.

Из рисунка 5 явно видно, что не все параметры одинаково сильно влияют на результат категоризации. Было выбрано подмножество величиной $K=8$ параметров, вносящих наибольший парциальный вклад в результат категоризации.

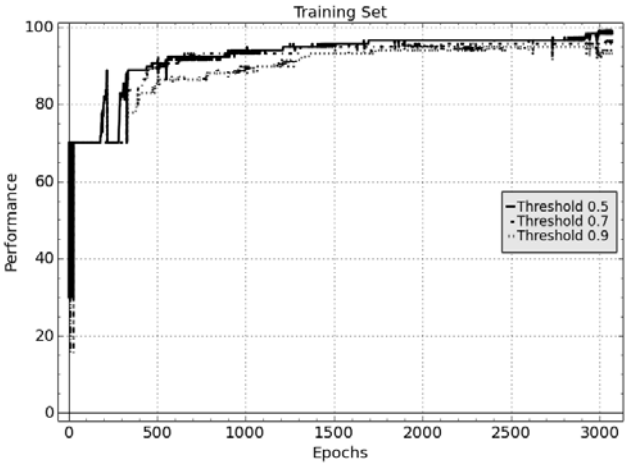


Рис.7. Изменение СКО в процессе обучения для обучающей выборки.

По определенному подмножеству из *8* параметров сформированы шаблоны обучения и синтезирована нейронная сеть класса MLP с размером скрытого слоя *8×3=24* нейрона с сигмоидными функциями активации. Проведено обучение нейронной сети по методу градиентного спуска в режиме batch, в процессе обучения производительность нейронной сети оценивалась как среднеквадратическое отклонение (СКО) требуемого выходного значения от фактического, как показано в следующем выражении:

$$RMS = \sum_{j=0}^{N} \left(z_{j,d} - z_{j,a} \right)^2. \qquad (5)$$

Изменение СКО в процессе обучения показано на рисунке. 6. Так же в процессе обучения производилась оценка качества категоризации сети отдельно для обучающей выборки и контрольной выборки. Качество категоризации определено следующим выражением:

$$Q = {N_p}/{N_f} \times 100\%, \qquad (6)$$

где Q – качество категоризации, N_p – количество кристаллов, категория которых определена правильно, N_f – количество кристаллов, категория которых определена неправильно.

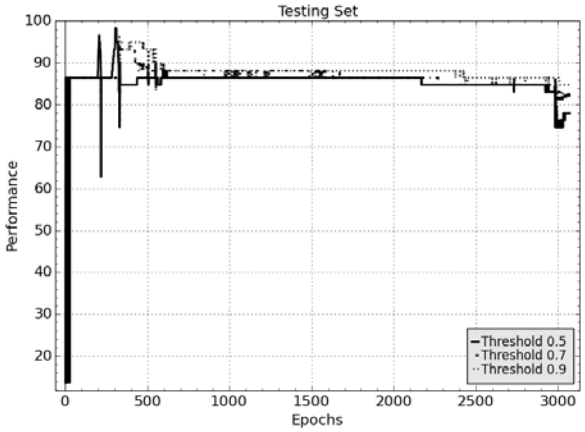


Рис.8. Изменение качества категоризации в процессе обучения для контрольной выборки.

Как видно из рисунка 6 в процессе обучения СКО падает, что соответствует результатам на рисунке 7 и 8. После примерно 2800 итераций качество категоризации на обучающей выборке возрастает до примерно 90 %. Важно отметить, что выходное значение нейронной сети – непрерывная величина, в то время как количество категорий – конечная, задаваемая в шаблонах обучения дискретно как *0* и *1*. На практике для решения задачи категоризации необходимо так же задаваться пороговым значением для сравнения. На рисунках 7 и 8 показаны кривые для различных пороговых значений.

После завершения процесса обучения произведена более подробная оценка производительности сети. Ошибки, допускаемые при категоризации делятся на два вида:

- *промахи* – кристалл, соответствующая которому микросхема не функционирует при измерении при повышенной температуре отнесен к категории «годный»;
- *ложные срабатывания* – кристалл, соответствующая которому микросхема функционирует при измерении при повышенной температуре, признан к категории «брак».

С точки зрения производственного процесса промах приводит к корпусированию бракованного кристалла, т.е. эквивалентен затратам на корпус. Ложное срабатывание приводит к отказу от корпусирования годного кристалла, что наносит серьезный ущерб, т.к. кристалл, как правило, дороже корпуса.

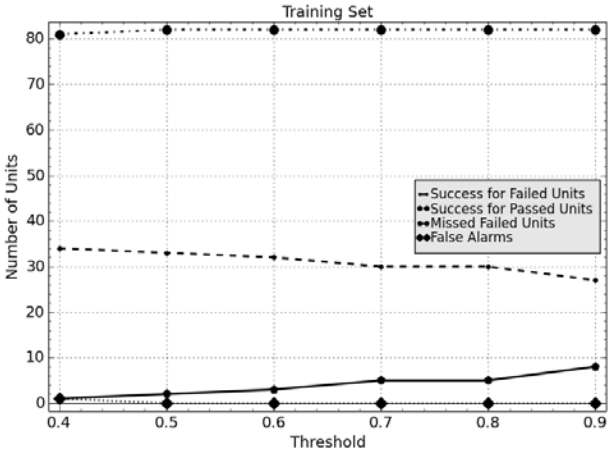


Рис. 9. Результаты анализа качества категоризации для обучающей выборки.

Результаты анализа качества категоризации, выполняемой нейронной сетью, для обучающей выборки приведена на рисунке 9. Из рисунка видно, что сеть качественно справляется с поставленной задачей, что соответствует снижению СКО.

Результаты для контрольной выборки, данные которой являются для сети новыми, т.к. они не использовались в процессе обучения, представлены на рисунке 10. Из рисунка видно несколько худшее качество категоризации.

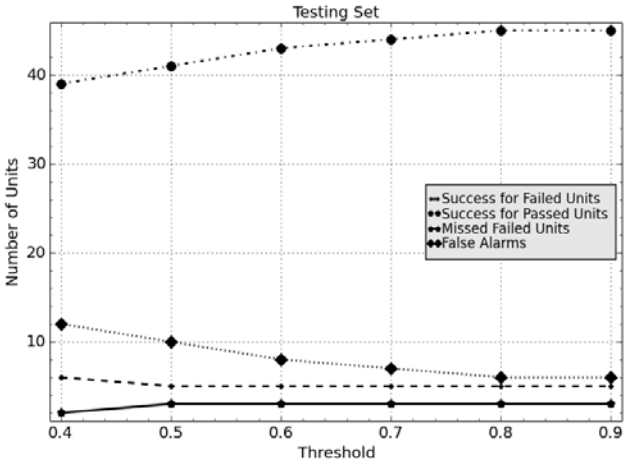


Рис. 10. Результаты анализа качества категоризации для контрольной выборки.

В описанном в данной работе эксперименте обученная нейронная сеть была применена для оценки количества брака на пластинах №4 и №5 – система идентифицировала порядка 30 сбойных кристаллов на двух пластинах. В результате было принято решение провести дополнительное измерение пластин №4 и №5, что позволило оценить апостериорно достигнутое нейронной сетью качество оценки. Результаты приведены на рис 11 и таблице 2.

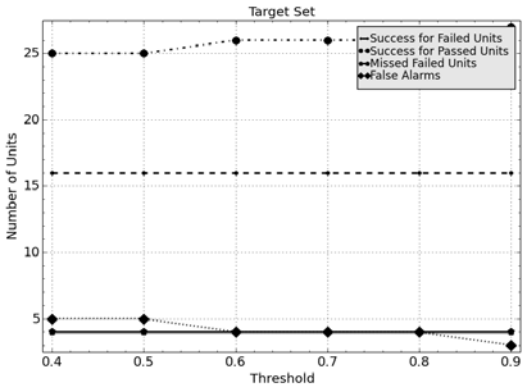


Рис. 11. Результаты анализа качества категоризации для рабочей выборки.

Таблица 2

Пороговое значение	Успешно идентифицированные годные кристаллы	Успешно идентифицированные бракованные кристаллы	Ложные срабатывания	Промахи
0.4	16	25	4	5
0.5	16	25	4	5
0.6	16	26	4	4
0.7	16	26	4	4
0.8	16	26	4	4
0.9	16	27	4	3

Заключение

В работе показано применение перспективного метода обработки данных – нейростевого анализа. Представлены экспериментальные результаты, свидетельствующие о возможности успешного применения данного метода для установления связи между результатами измерений кристаллов и микросхем при различных температурах и оценки количества брака. Экспериментальные данные, приведенные в таблице 2 показывают успешную идентификацию 90% кристаллов, забракованных при контроле в составе микросхемы.

Estimation of the suitability of the dies using Artificial Neural Networks

E. N. Efimov

Abstract. An approach to estimate suitability of the dies using Artificial Neural Networks based on the results of electrical measurements and functional control of these dies under the normal environmental conditions and the measurements of the chips from the same lot under the high environmental temperature is presented in this paper. It is shown that the Artificial Neural Network detects the hidden dependency between these measurements and uses that dependency to predict the outcome of the functional testing of the chips. Experimental data indicates that over 90% of the failed under the high environmental temperature dies are successfully detected based on the measurements under the normal environmental conditions without actual testing under the high environmental temperature.

Keywords: Neural networks crystal temperature

Литература

1. S. Haykin. Neural Networks – A Comprehensive Foundation / S. Haykin; Ed. by 2. — Prentice Hall, 1998.

2. M. H. Hassoun. Fundamentals of Artificial Neural Networks. — The MIT Press, 1995.

3. S. Samarasinghe. Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition. — 1 edition. — Auerbach Publications, 2006.

4. Е.Н. Ефимов, Т.Я. Шевгунов. Построение нейронных сетей прямого распространения с использованием адаптивных элементов // Журнал радиоэлектроники, 2012, № 8.

5. R. Battiti. First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method // Neural Computation, 1992, vol. 4, no. 2.

Верификация RTL-моделей при помощи случайных тестовых воздействий

Р. Р. Стамболян

Аннотация: В статье рассмотрены основные проблемы, которые возникают при верификации современных СБИС. Целью статьи является оптимизация процесса верификации за счёт правильной архитектуры тестовой системы. Проанализированы различные подходы к построению тестовых систем, и, предложена оптимальная архитектура тестовой системы, имеющая модульное построение. Также, предложен способ организации самих тестов, который заключается в использовании генераторов тестов.

Ключевые слова: СБИС, верификация, тест.

1. Введение

Прогресс в полупроводниковой промышленности, обусловленный непрерывным уменьшением проектных норм, привёл к возникновению принципиально новых проблем в конструировании *ИС*. Помимо трудностей технологического характера возникли проблемы с обеспечением эффективного процесса проектирования. При переходе на технологические нормы 0.12 мкм и ниже, у разработчиков достаточно транзисторов на кристалле для проектирования практически любой СБИС. А возрастание числа реализуемых на СБИС функций привело к развитию современных систем на кристалле, содержащих множество различных элементов, таких как процессоры, контроллеры, модули памяти и др. В результате, сложность современных проектов ИС многократно возросла. И на сегодняшний день, верификация крупных СБИС занимает более 60% всего времени проектирования[5], что ставит на первый план проблему верификации таких проектов.

Проблема верификации связана с двумя аспектами:

- Высокая трудоёмкость верификации, вследствие увеличения сложности проектируемых устройств
- Большие временные затраты на верификацию

С другой стороны, стоимость комплекта фотошаблонов для субмикронных *СБИС* настолько высока, что их повторное изготовление, в случае пропущенной ошибки, является недопустимой издержкой, связанной как с задержкой выхода устройства на рынок и потерей репутации, так и с ростом конечной стоимости. Данный аспект приводит к другому требованию – повышению качества верификации путём повышения степени покрытия тестируемого устройства. Все вышеперечисленные факторы послужили стимулов для разработки принципиально новых методов верификации.

2. Тестовое окружение

Традиционно, верификация устройства проводилась по следующей схеме:

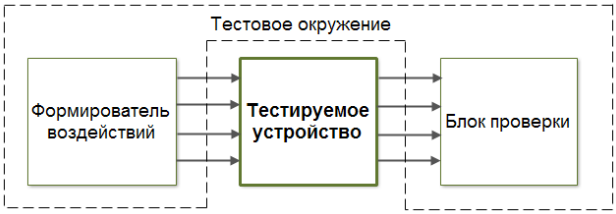


Рис. 1. Прямые тестовые воздействия.

Данная схема иллюстрирует метод прямых тестовых воздействий. Он обладает следующими преимуществами: его достаточно просто использовать, и, благодаря заранее известным входным воздействиям, несложно построить блок проверки, который заведомо «знает» верный отклик устройства. Однако, данный метод обладает и рядом недостатков. Во-первых, тестируемые устройства имеют огромное количество состояний, и чтобы проверить всевозможные их комбинации прямыми тестами, зачастую требуется создать довольно громоздкие тестовые наборы. Во-вторых, в большинстве случаев, разрабатываемые компоненты являются специфическими для конкретного устройства, и для каждого последующего устройства, потребуется разрабатывать данные компоненты заново. Перечисленные недостатки стали очередным толчком к дальнейшему развитию средств и методик верификации. Появились такие принципиально новые методы верификации как:

- моделирование на уровне транзакций (TLM) – введение наборов стандартных методов для работы с транзакциями, что позволит ускорить и упростить моделирование и отладку на системном уровне
- введение средств для количественной оценки переходов между состояниями RTL-моделей, а также для сбора функционального покрытия
- внесение языковых элементов для формирования случайных тестовых воздействий
- использование методов объектно-ориентированного программирования (ООП) для максимального заимствования кода при верификации

В 2005 году на основе языка Verilog-2001 (IEEE Std. 1364-2001), появился язык SystemVerilog (IEEE Std 1800™-2005) который стал первым стандартом на язык проектирования и тестирования аппаратуры, включающий в себя вышеперечисленные аспекты, а также, несколько новых:

- возможность генерации случайных тестовых воздействий, управляемых внешними ограничениями
- возможность описания точек анализа функционального покрытия, а также их мониторинг для оценки эффективности тестовой системы

Хоть SystemVerilog и представлял множество новых средств для построения эффективных тестовых воздействий, разработка с «нуля» тестового окружения для каждого устройства всё ещё остаётся достаточно трудоёмким процессом. Для уменьшения трудоёмкости требуется возможность повторного использования компонентов тестовой системы для различных устройств. Для этого нужна некая методика, которая опишет то, как должны выглядеть все эти компоненты, что сделает возможным повторное их использование не зависимо от проекта и друг от друга.

Существует несколько различных методологий верификации, среди которых можно назвать RVM, URM, AVM, VMM, OVM. Однако, с появлением новой методологии UVM в 2011году, которая стала первым международным стандартом и объединила усилия всех своих предшественниц за последнее десятилетие, развитие всех остальных методологий остановилось.

3. Архитектура тестовой системы

Тестовое окружение – это многоуровневая структура, состоящая из трёх уровней. Первый уровень – это тестируемое устройство (DUT). Второй уровень – это уровень транзакторов, которые представляют собой конвертирующие компоненты между уровнем транзакций и уровнем внешних входов-выходов DUT. На третьем уровне располагаются все компоненты уровня транзакций.

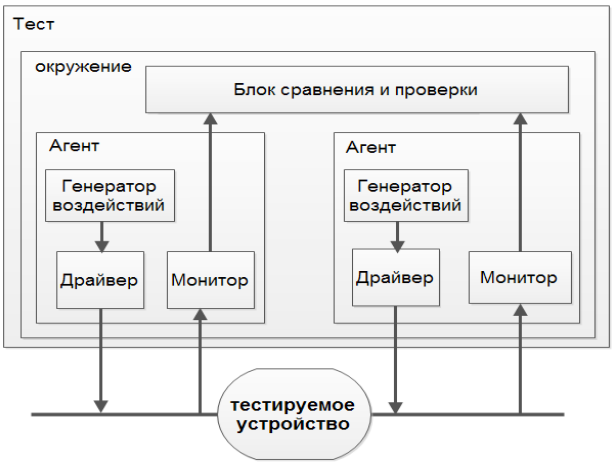


Рис. 2. Модульное тестовое окружение.

- Генератор воздействий создаёт набор транзакций для тестирования устройства, а затем передаёт их драйверу на исполнение. Наборы транзакций могут быть случайными, направленными или случайными с внешними ограничениями
- Драйвер конвертирует последовательности транзакций, полученные от генератора, в сигналы выводов тестируемого устройства
- Монитор является пассивным элементов, который ведёт наблюдение за шиной устройства и конвертирует сигналы на шине в пачки транзакций для дальнейшей обработки, а также, отвечает за сбор покрытия теста. Ещё одной важной его функцией является общий контроль ошибок
- мониторы, драйверы и генераторы могут быть повторно использованы независимо друг от друга, однако для этого требуется знание некоторых особенностей реализации данных компонентов для их настройки и правильного подключения. Чтобы уменьшить количество информации, которую нужно запомнить для использования данных компонентов используют агенты, которые инкапсулируют драйвер, монитор и генератор, тем самым снижая трудоёмкость использования
- Блок сравнения и проверки определяет, корректно ли работает тестируемое устройство. Он осуществляет проверку целостности потоков данных

Основным компонентом в иерархии тестовой системы является окружение теста, которое также называется универсальным верификационным компонентом. Окружение теста должно представлять собой логически завершённый, полностью самодостаточный компонент тестовой системы, который служит для верификации одного конкретного интерфейса (Ethernet, Jtag, Pcie, или др.). Таким образом, данный компонент тестовой системы абсолютно не зависит от специфики тестируемого устройства, т.к. он привязан к стандартизованному интерфейсу. Это позволяет свободно использовать различные комбинации такого рода тестовых окружений, что сильно уменьшает трудоёмкость за счёт высокой степени заимствования кода.

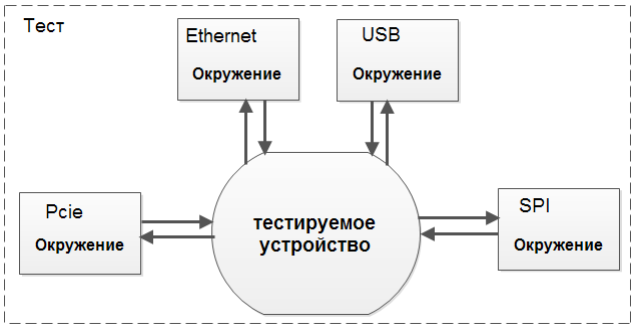


Рис. 3. Универсальные верификационные компоненты.

Немаловажным является вопрос построения самих тестов. Соотношение сложности и количества тестов должно давать наибольшую степень покрытия.

Оптимальным решением является построение тестов на основе генераторов тестов. Генератор теста представляет собой некий шаблон, содержащий множество параметров, которые при генерации самого теста, принимают случайные значения в определённом диапазоне значений, который прописан в генераторе теста. Полученный таким образом набор тестов непосредственно используется в тестовом окружении.
Для того, чтобы максимально снизить трудоёмкость написания генератора тестов, нужно, чтобы каждый генератор, тестировал лишь одну функциональную возможность устройства. Разработав по одному генератору тестов на каждую функциональную возможность, можно будет сгенерировать тестовый набор, который будет покрывать все устройство целиком, а в то же время трудоёмкость написания каждого из генераторов будет максимально низкой, т.к. каждый генератор будет сосредоточен на своей функциональной возможности. Но, прежде чем перейти к написанию генераторов тестов, требуется осуществить разбиение тестируемого устройства на функциональные возможности. Основным критерием, по которому нужно проводить разбиение — это независимость функциональных возможностей устройства друг от друга. Благодаря этому станет возможным запуск сгенерированных тестов независимо друг от друга.

4. Заключение

Традиционные методы верификации уже не годятся для современных проектов СБИС. Разработка тестовых окружений согласно оптимальной архитектуре описанной в данной статье, которая базируется на методологии построения тестовых окружений UVM, позволяет решить ряд проблем связанных с возросшей сложностью проектов СБИС. Модульное структура окружений теста, даёт гибкую тестовую систему, которая не зависит от тестируемого устройства, что в свою очередь даёт высокую степень заимствования кода для других проектов. Концепция

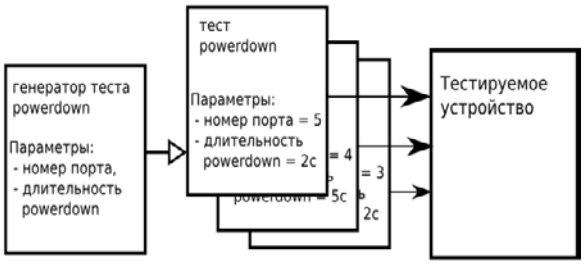


Рис. 4. Пример генератора тестов

разработки генераторов тестов функциональных возможностей устройства, вместо написания самих тестов для устройства, даёт тестовую систему, состоящую из простых тестовых наборов, но, в то же время, с высокой перебирательной способностью, т.к. тесты покрывают все функциональные возможности устройства. Тестовые окружения, построенные по данной архитектуре, используются для верификации различных коммутационных узлов среды RapidIO в НИИСИ РАН.

Random RTL-model verification

R. R. Stambolyan

Abstact: This paper discusses the main issues emerging from verification of modern VLSI. The main idea of the paper is optimization of verification process by means of correct architecture. Different approaches for testing system construction is considered and presented optimal module structured architecture of testing system. In addition the way of test organization is provided connected with the use of test generators.

Keywords: VLSI verification, test

Литература

1. А.Пеженко, Д.Радченко. Язык systemVerilog: Проектирование СБИС и систем. –М.: журнал Электроника, N4,2006.
2. IEEE 1800-2005.- IEEE Standard for SystemVerilog - unified hardware design, specification and verification language. 2005.
3. И.Г. Каршенбойм. SystemVerilog и улучшение отладки проектов // Компоненты и технологии. - 2010. - №1.
4. А.А. Рабаволюк. Функциональная верификация цифровых HDL-проектов: методология на основе ассертов // Электроника: Наука, Технология, Бизнес. - 2010. - №1.
5. M.Glasser. Open Verification Methodology Cookbook. – N.Y., Springer, 2009. – 236 c.
6. J. Bergeron. Writing testbenches using SystemVerilog. New York : Springer Science+Business Media Inc., 2005.

7. L. A.Thomas. Open Verification Methodology: Fulfilling the Promise of SystemVerilog // information quarterly. - 2008. - №1.

8. J.Bergeron, E.Cerny, A.Hunter. Verification Methodology Manual for SystemVerilog // Ch 1, 3, 7. - N.Y., Springer, 2005.

9. Проектирование СБИС типа «Система на кристалле». Маршрут проектирования // "Время электроники": Интернет-журнал URL: http://www.russianelectronics.ru

10.Accellera, Universial verification methodology (UVM) 1.1 user's guide, www.accellera.org, 2011

11. N. Kitchen and A. Kuehlmann. Stimulus generation for constrainted random simulation. - // International Conference on Computer-Aided Design, 2007, pages 258–265

12. S. Rosenberg, K. Meade. A Practical Guide to Adopting the Universal Verification Methodology (UVM), Cadence Design Systems, 2010

Моделирование цепей питания при проектировании печатных плат

А.С. Мухин

Аннотация. Приведены общие сведения о цепях питания в печатных платах. Определен частотный диапазон, в котором печатные платы вносят наибольший вклад в импеданс цепей питания. Рассмотрены свойства конденсаторов и особенности их подключения в цепи питания. Представлена простейшая модель конденсатора. Составлен маршрут моделирования цепей питания при проектировании печатной платы.

Ключевые слова: моделирование, печатные платы, конденсатор.

1. Введение

В настоящее время при создании проектов современной электроники одним из актуальных условий является максимальное ускорение процедуры проектирования. Непрекращающееся усложнение создаваемых систем приводит к неравномерному распределению появления и удаления ошибок по различным этапам проектирования. Чем раньше будет обнаружена ошибка, тем меньше времени будет затрачено на проектирование. Поэтому процесс моделирования играет очень важную роль в ходе проектирования. Заблаговременное выявление ошибок в ходе моделирования позволяет сэкономить не только временные, но и денежные ресурсы. Все это в полной мере относится и к проектированию печатных плат. И здесь можно выделить несколько сфер, на которые требуется обратить особое внимание: целостность питания и целостность сигналов. Данная статья посвящена вопросу целостности питания и моделированию цепей питания при проектировании печатных плат.

2. Общие положения

Схема разводки питания состоит из всех соединительных элементов между модулем стабилизатора напряжения и контактами микросхем, по которым распространяются прямые и возвратные токи потребления. Она включает в себя такие элементы, как: источник питания, развязывающие конденсаторы, переходные отверстия, дорожки и слои металлизации, выводы микросхем [1]. Простейшее представление схемы разводки питания представлено на рис. 1.

Главная задача цепей питания – обеспечение постоянного напряжения питания непосредственно на микросхемах в пределах установленной погрешности (обычно около ±5%). Это напряжение должно поддерживаться в допустимых рамках во всем рабочем диапазоне частот.

При протекании тока через полигон питания с неким импедансом по закону Ома возникнет падение напряжения на этом межсоединении, т.е. шум. Это падение напряжения приводит к тому, что фактическое значение напряжения питания непосредственно на ножках микросхем отличается от заявленной величины. Поэтому обычно предъявляется максимально допустимое значение просадки (на постоянном токе) и пульсации (на переменном токе) (обычно ±5% от номинала). В результате некорректного проектирования схемы разводки питания может образоваться значительный шум в шинах питания микросхем. Это может привести к потере работоспособности, как отдельной микросхемы, так и всего модуля в целом.

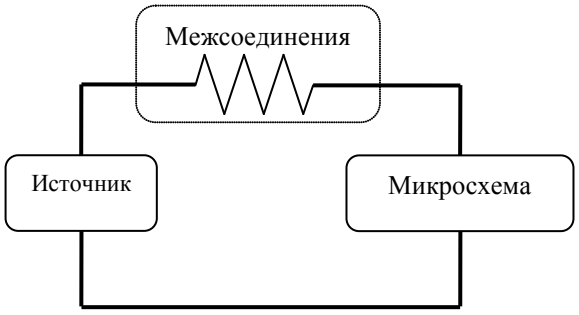


Рис. 1. Простейшее представление схемы разводки питания.

Важным шагом при проектировании схемы разводки питания является определение целевого импеданса. Это необходимо делать независимо для всех компонентов, подключенных к разным цепям питания. Максимальное значение импеданса пути от источника к нагрузке, т.е. целевой импеданс, определяется исходя из следующих оснований: это наибольший импеданс, на котором пульсации напряжения будут ниже допустимых.

Компоненты схемы разводки питания в частотной области могут быть разделены на 5 частей в зависимости от того, в каком диапазоне частот они доминируют при определении импеданса (рис. 2) [1]. На самых низких частотах (до 10 КГц) основной вклад в импеданс вносит стабилизатор напряжения (СН), если последовательное сопротивление межсоединений

не превышает его. В идеале при работе СН напряжение на выходе поддерживается постоянным независимо от тока нагрузки. Большое изменение тока вызывает малое изменение напряжения – это поведение цепи с низким импедансом.

Следующий частотный диапазон 10 - 100 КГц. Тут определяющую роль играют электролитические и танталовые конденсаторы с большим значением емкости, которые обеспечивают низкий импеданс в данном диапазоне.

Импеданс на самых высоких частотах (в ГГц режиме) устанавливается внутрикристальными разделительными емкостями. Они обладают наименьшей индуктивностью и позволяют обеспечить довольно низкий импеданс в области наивысших частот.

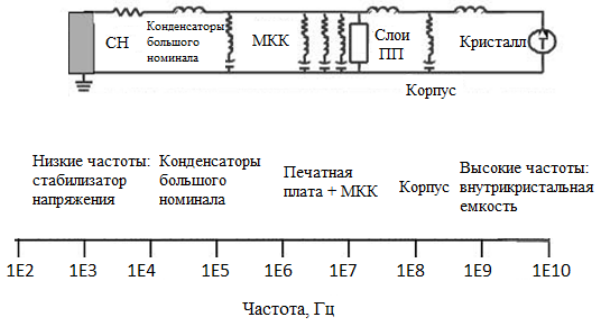


Рис. 2. 5 частей схемы разводки питания и диапазоны их частотного влияния на импеданс пути

Исходя из всего вышесказанного, область, в которой можно влиять на импеданс на уровне печатной платы, находится в диапазоне 100 КГц - 100 МГц. Снижения импеданса в этом диапазоне можно добиться путем изменения структуры слоев печатной платы или при помощи добавления многослойных керамических конденсаторов (МКК). Причины такого ограничения влияния печатной платы на импеданс на частотах выше 100 МГц объясняются в следующей главе.

3. Область влияния печатной платы.

Между контактными площадками кристалла и печатной платы располагается корпус, а точнее его выводы. То есть с точки зрения платы мы не можем достичь кристалла, минуя корпус. И даже если импеданс платы будет спроектирован идеально, начиная с некоторой частоты (эффективной частоты корпуса), индуктивность выводов корпуса, стоящая на пути к контактным площадкам кристалла, будет ограничивать импеданс, наблюдаемый со стороны кристалла.

Индуктивность выводов корпуса создает некий барьер между ними [1]. Импеданс индуктивности определяется следующим образом:

$$Z = 2\pi f L, \qquad (1)$$

где f – это частота, а L – индуктивность.

В простых корпусах с внешними выводами индуктивность контура пары одиночных ножек земли и питания может составлять 5 нГн. В многослойных BGA-корпусах с числом слоев не меньше четырех эта индуктивность выводов может быть снижена до величин ниже, чем 1 нГн. К примеру, импеданс такой пары с индуктивностью 1 нГн на частоте 100 МГц составляет около 0,6 Ом.

Все вышесказанное означает, что какие операции ни были бы сделаны на уровне печатной платы (добавление конденсаторов, изменение конфигурации платы), выше эффективной частоты нельзя добиться импеданса ниже, чем импеданс, определяемый выводами корпуса. Снижения импеданса в области, выше эффективной частоты, можно добиться добавлением конденсаторов непосредственно на корпус микросхемы, а также изменением конфигурации самого корпуса.

4. Конденсаторы

Одним из самых эффективных способов уменьшения импеданса является установка конденсаторов [3]. Импеданс идеального конденсатора с емкостью C обратно пропорционален частоте:

$$Z = \frac{1}{2\pi f C}, \qquad (2)$$

где C – это емкость конденсатора. Но поведение импеданса реального конденсатора отличается от идеального случая, поскольку реальный конденсатор обладает паразитными параметрами. Реальный конденсатор может быть аппроксимирован RLC-моделью (рисунок 3).

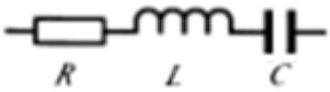


Рис. 3. Эквивалентная RLC-модель конденсатора.

R или ESR на данном рисунке есть ничто иное, как последовательное эквивалентное сопротивление конденсатора (ПЭС), L или ESL – последовательная эквивалентная индуктивность конденсатора (ПЭИ).

На рисунке 10 видно, что поведение импеданса этой модели довольно точно совпадает с поведением импеданса реального конденсатора вплоть до очень высоких частот.

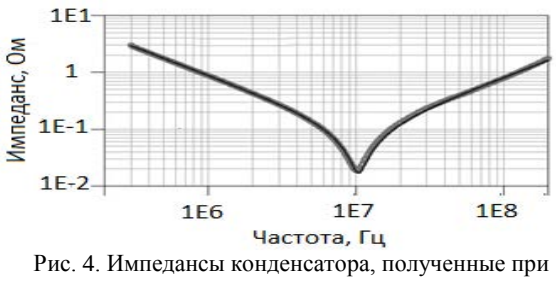


Рис. 4. Импедансы конденсатора, полученные при помощи моделирования и измерения

Частота, на которой импеданс конденсатора достигает наименьшего значения, называется частотой собственного резонанса (ЧСР) и определяется следующим образом:

$$f_{sr} = \frac{1}{2\pi\sqrt{LC}}, \quad (3)$$

где f_{sr} – частота собственного резонанса, L – последовательная эквивалентная индуктивность конденсатора, C – емкость конденсатора.

Выше частоты собственного резонанса основной вклад в импеданс конденсатора вносит индуктивность, а ниже этой частоты – емкость. Вблизи частоты собственного резонанса поведение импеданса конденсатора отличается от поведения идеальной емкости и индуктивности и зависит от величины последовательного эквивалентного сопротивления. Именно частота собственного резонанса и определяет ту область частот, в которой данный конденсатор может быть эффективно использован. Однако кроме собственной индуктивности конденсатора, существенное влияние на его работу оказывает индуктивность, связанная с его монтажом. На самом деле, величина индуктивности монтажа может даже превышать значение собственной индуктивности конденсатора. ПЭИ связанная с конденсатором и его соединением с корпусом может быть разделена на 4 части:

- индуктивность контура дорожек металлизации на поверхности платы от конденсатора до переходных отверстий;
- индуктивность контура переходных отверстий до слоев питания и земли;
- распределенная индуктивность между переходными отверстиями конденсатора и корпуса;
- индуктивность контура переходных отверстий от слоев питания и земли до выводов корпуса микросхемы.

Также следует отметить, что когда на плате установлено небольшое количество конденсаторов и токи, текущие от конденсаторов к микросхемам, не накладываются друг на друга, ПЭИ каждого конденсатора есть индуктивность его полного пути. В этом случае каждый конденсатор ведет себя независимо от других.

Однако если токи накладываются, как в случае если имеется скопление конденсаторов в одной области печатной платы или используется большое количество конденсаторов вокруг одного корпуса, распределенная индуктивность является сложной функцией от расположения и величин конденсаторов и расположения выводов питания и земли в корпусе.

Наиболее эффективными методами уменьшения ПЭИ являются:

- располагать слои питания/земли как можно ближе к поверхности печатной платы;
- уменьшать толщину диэлектрика;
- использовать широкие дорожки металлизации;
- минимизировать длину дорожек металлизации;

располагать конденсаторы как можно ближе к нагрузке.

5. Параллельное соединение конденсаторов

Стратегия по достижению профиля импеданса разводки питания ниже целевого в рабочем диапазоне частот заключается в подборе и установке нужного количества конденсаторов с необходимыми параметрами [2]. При параллельном соединении идентичных конденсаторов результирующий импеданс является все той же RLC-моделью, но с параметрами R,L и C, отличающимися от параметров в одиночном случае. При увеличении количества конденсаторов величина импеданса во всем диапазоне частот уменьшается, но при этом частота собственного резонанса параллельного соединения любого количества идентичных конденсаторов совпадает с частотой собственного резонанса для одиночного конденсатора. Поэтому один из способов снижения импеданса – это добавление идентичных конденсаторов в параллель. В том случае, если два конденсатора, соединенных параллельно, имеют разные значения емкости (C) или ПЭИ (ESL), то профиль их суммарного импеданса не так прост. В этом случае в профиле импеданса все так же наблюдаются минимумы на частотах собственного резонанса каждого из конденсаторов. Конденсатор с большей емкостью характеризуется меньшей частотой собственного резонанса и наоборот. Но помимо этих минимумов здесь наблюдается некий пик импеданса, который возникает на частоте параллельного резонанса (ЧПР). На рисунке 5 изображен импеданс параллельного соединения двух конденсаторов различной емкости, но с одинаковыми значениями ПЭИ (ESL) и ПЭС (ESR).

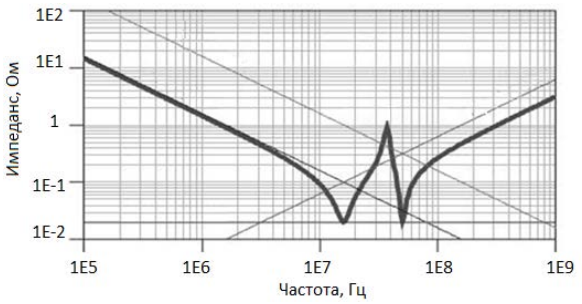


Рис.5 Импеданс параллельного соединения двух конденсаторов с разными емкостями, но одинаковыми ПЭИ и ПЭС

Величину ЧПР сложно вычислить точно, т.к. она зависит от ПЭИ большего конденсатора, емкости меньшего конденсатора и от ПЭС обоих конденсаторов. В случае если частоты собственного резонанса разнесены, ЧПР может быть вычислено согласно следующей аппроксимации:

$$PRF \approx \frac{1}{2\pi\sqrt{C_2 ESL_1}}, \quad (4)$$

где PRF - частота параллельного резонанса, C_2 – емкость меньшего конденсатора, ESL_1 – ПЭИ большего конденсатора. ЧПР является очень важной величиной при проектировании схемы разводки питания, потому что именно на ней возникают нежелательные скачки импеданса, с которыми

необходимо бороться. Существуют несколько способов борьбы с параллельным резонансом:

- понижение ПЭИ большего конденсатора;
- увеличение емкости меньшего конденсатора;
- увеличение ПЭС обоих конденсаторов.

Последовательное эквивалентное сопротивление конденсатора зависит от структуры самого конденсатора. Чем больше емкость конденсатора, тем количество слоев в параллель, из которых он состоит, больше и тем самым меньше его последовательное эквивалентное сопротивление. Поэтому для снижения резонансного пика можно использовать конденсаторы с большим значением ПЭС, т.е. с меньшим значением емкости.

Снижение ПЭИ тоже является эффективным методом по снижению резонансного пика. Выбор большего конденсатора с меньшим значением последовательной эквивалентной индуктивности и увеличение емкости меньшего конденсатора могут привести к почти полной нейтрализации пика импеданса.

Еще один способ снижения резонансного пика, образованного двумя конденсаторами с разными частотами собственного резонанса, заключается в добавлении третьего конденсатора с частотой собственного резонанса, находящейся между их частотами собственного резонанса. Оптимальное значение этой частоты для третьего конденсатора зависит от емкостей, ПЭИ и ПЭС всех трех конденсаторов и посему сложно определяется без помощи каких-либо САПР. Но существует два подхода по выбору данной частоты: выбрать третий конденсатор так, что его частота собственного резонанса совпадает с ЧПР, или выбрать частоту собственного резонанса третьего конденсатора посередине между ЧСР двух конденсаторов.

6. Ограничение по плотности тока.

Помимо просадок и пульсаций напряжения в цепях питания на печатных платах необходимо следить за тем, чтобы плотность тока в них не превышала предельно допустимого значения. Ввиду того, что токи потребления различных микросхем могут достигать значений порядка десятков ампер, и эти токи могут протекать по участкам схемы разводки питания малой площади, то для предотвращения выхода из строя (выгорания) межсоединений необходимо следить за распределением плотности тока по слоям питания и земли и в случае обнаружения критических значений плотности тока принимать меры по их устранению.

7. Моделирование.

Как уже упоминалось ранее, современные системы постоянно усложняются, частоты и токи потребления постоянно растут, так что моделирование начинает играть все более важную роль в процессе проектирования. Для моделирования цепей питания в печатных платах в настоящее время создано уже довольно большое количество разных инструментов. В целом весь маршрут моделирования цепей питания можно разбить на следующие этапы:

1) **Получение исходных данных для моделирования**
   Сюда входят топология печатной платы, свойства материалов печатной платы, модели конденсаторов, индукторов и резисторов, информация по конкретным параметрам нагрузок, таким как токи потребления и рабочий диапазон частот.
2) **Настройка САПР перед моделированием**
   Установка необходимых параметров для моделирования, загрузка моделей, выбор необходимых цепей и нагрузок;
3) **Моделирование по постоянному току и анализ результатов моделирования**
   На данном этапе моделирования происходит проверка уровней просадок напряжения на пути от источника к нагрузкам и анализ плотностей токов в цепях питания;
4) **Внесение корректировок в проект**
   На этом этапе ведется тесная совместная работа с разработчиком печатной платы по внесению корректировок в проект исходя из результатов моделирования.
5) **Повторное моделирование и возврат в случае необходимости к пункту 4**
6) **Моделирование цепей питания по переменному току и анализ результатов моделирования**
   На этом этапе происходит создание профилей импеданса цепей питания, подсчет целевого импеданса в области влияния печатной платы. Кроме этого, проведя моделирования на более высоких частотах (выше 100МГц) можно найти наиболее шумящие области в печатной плате, таким образом, данная проверка также затрагивает и вопрос электромагнитной совместимости.
7) **Внесение корректировок в проект**
   На этом этапе ведется тесная совместная работа с разработчиком печатной платы по внесению корректировок в проект исходя из результатов моделирования.
8) **Повторное моделирование и возврат в случае необходимости к пункту 7.**

Изменения в проект вносятся на основании теоретического материала, данного в разделах 2-5.

Благодаря моделированию существует возможность на ранних этапах проектирования, когда еще нет физического образца печатной платы, выявлять ошибки, связанные с цепями питания. Такой подход к проектированию печатных плат позволяет избежать дополнительных затрат времени и ресурсов, а так же позволяет облегчить и ускорить процесс создания печатных плат. Также достигается снижение трудоемкости процесса выявления ошибок, связанных с цепями питания за счет простоты процесса моделирования.

Simulation of power delivery networks in design of printed circuit boards

A.S. Mukhin

Abstract: There is an overview of power delivery networks in printed circuit boards. Frequency range in which the printed circuit boards make the largest contribution to the impedance of the power supply circuits is determined. The properties of capacitors and the characteristics of their connection to the power delivery networks are discussed. A simple model of the capacitor. Route of simulation of power delivery networks in the design of the printed circuit boards is presented.

Keywords: modeling, PCB, capacitor.

Литература

1. Eric Bogatin. Signal and Power Integrity – Simplified (2nd Edition) – Prentice Hall, 2009.
2. Stephen H. Hall, Howard L. Heck. Advanced signal integrity for high-speed digital designs – John Wiley & Sons, Inc, 2009.
3. Л.Н. Кечиев. Проектирование печатных плат для цифровой быстродействующей аппаратуры - М.: ООО «Группа ИДТ», 2007.

Химические реакции окисления нефти при её добыче с закачкой в пласт воздуха

В.А. Юдин[1], А.В. Королёв[2], И.В. Афанаскин[2], С.Г. Вольпин[2]

1 – кандидат физико-математических наук, 2 – кандидат технических наук

Аннотация: Приведен краткий обзор литературы по окислению и термолизу пластовой нефти при её добыче с закачкой в пласт воздуха; рассмотрены экспериментальные данные о типичных реакциях. Приведены примеры моделирования данных реакций с разделением нефти на SARA-фракции. Показана необходимость проведения комплекса экспериментальных исследований окисления и термолиза углеводородов с использованием нефтей и пород предполагаемых к разработке объектов.

Ключевые слова: термогазовое воздействие, внутрипластовое горение, окисление пластовой нефти, термолиз пластовой нефти, баженовская свита.

Введение

По данным большинства аналитиков [1] производство и потребление энергии в мире в ближайшие 2–3 десятилетия будет расти.

Хотя доля нетрадиционных и возобновляемых источников энергии будет возрастать, их вклад в общее производство энергии всё же будет составлять только около 30% [2].

На ближайшие несколько десятилетий роль не возобновляемых углеводородных источников энергии - нефти, газа, угля - будет оставаться определяющей в снабжении человечества энергией. При этом около 60% от общего производства первичной энергии придётся на нефть и газ [1-3].

На территории России, составляющей 12.8% территории Земли, сосредоточено 12-13% *прогнозных ресурсов* и около 12% *разведанных запасов* нефти [1-5].

Однако, *доказанные запасы* нефти в России не велики, и при сохранении нынешних темпов её добычи их хватит менее чем на 25 лет [1]. В России сложилось весьма тревожное положение с обеспечением запасами, т.е. с соотношением между разведанными запасами и уровнем годовой добычи, ввиду чего многие специалисты прогнозируют возможное значительное падение нефтедобычи в России в недалёком будущем [4, 5]. Помимо этого, наблюдается и ухудшение качества *остаточных доказанных запасов*. В них велика доля так называемых *трудноизвлекаемых запасов (ТИЗ)*, на долю которых приходится не менее 55-58% *разведанных запасов* России [3-5].

Однако и *благоприятные запасы* характеризуются в России высокой степенью выработанности, которая на эксплуатируемых месторождениях превысила 50%, и высокой обводненностью пластов - в среднем на 70% [4, 5].

Ситуация с запасами нефти в России требует принятия срочных мер для поддержания нефтедобычи на приемлемом уровне в текущем столетии.

1. Термогазовое воздействие. Некоторые вопросы моделирования

Значительным стратегическим резервом поддержания (или увеличения) нефтедобычи в России могут являться совершенно нетрадиционные коллектора: кремнисто-глинистые и карбонатно-кремнисто-глинистые высокобитумонасыщенные породы, в первую очередь, баженовской свиты Западной Сибири, доманиковых отложений Волго-Уральской провинции, хадумского горизонта Предкавказья [6 - 9].

Согласно некоторым оценкам, суммарные *ресурсы* нефти только в баженовской свите оцениваются в размере 0.8–2.1 трлн.т. [10-13]. Достаточно консервативная оценка *прироста добычи* составляет 10-15 млн.т. нефти в год в течение 20 лет, т.е. 2.0-3.0 млрд.т. накопленной добычи [14]. В литературе приводятся даже экстремальные оценки - 30 млрд.т., 35-50 млрд.т. [10-15].

Наиболее перспективным методом для полномасштабной разработки таких месторождений является метод с закачкой в пласт воздуха и созданием в нефтенасыщенном пласте подвижного очага низкотемпературного окисления – термогазовое воздействие **(ТГВ)** [10-17].

Особенностью ТГВ является то, что при инициировании в пласте подвижного очага окисления путём закачки воздуха происходит и оттеснение нефти к добывающим скважинам газами горения, и повышается температура пласта, что делает возможным начало пиролиза керогена, входящего в состав скелета нефтесодержащих пород. Получаемая in-situ дополнительная нефть может быть добыта.

Физические и технологические особенности ТГВ предъявляют повышенные требования к подготовке соответствующих проектных документов разработки, даже на стадии опытно-промышленных работ [18, 19]. В обязательном порядке необходимо многовариантное численное моделирование всего процесса, прежде всего, для оценки экономической рентабельности

проекта до начала его реализации, выбора оптимального варианта ТГВ - как по расположению скважин, так и по режиму закачки агентов в пласт [18, 19].

На этапе же эксплуатации участка (или месторождения) методом ТГВ совершенно необходим контроль и регулирование внутрипластовых процессов, в первую очередь, перемещения фронта окисления и снижение риска прорыва воздуха к добывающим скважинам [18, 19].

Главной же составной частью процедур контроля и регулирования ТГВ является численное гидродинамическое моделирования пластовых процессов, по результатам которого и должны приниматься решения по изменению тех или иных технологических параметров [20-23].

За последние десятилетия в нефтегазовой отрасли были разработаны и используются несколько больших пакетов программ **(симуляторов)** для различных задач гидродинамического моделирования разработки, принципы которых создавались ещё в 60-70-х годах прошлого века.

Но, несмотря на значительные усилия, затраченные на разработку таких программ, имеющиеся симуляторы обладают значительным количеством недостатков и упрощений [24]. Они не позволяют эффективно использовать существующие симуляторы для проектирования и регулирования ТГВ в разнообразных геолого-технологических условиях.

Создание усовершенствованного, отечественного термогидросимулятора, предназначенного для отечественных суперкомпьютеров является одним из важнейших элементов в освоении стратегических резервов углеводородного сырья нашей страны.

Очевидно, что при численном моделировании ТГВ блок моделирования химических реакций окисления нефти является одним из основных, поскольку именно эти реакции являются источником тепла и газов горения, что и способствует более эффективному вытеснению нефти из пласта.

Корректное моделирование этих химических процессов определяет, в значительной мере, достоверность прогноза результатов ТГВ, целесообразность его применения на конкретном месторождении [25, 26].

Методология создания такого блока окончательно ещё не установилась [25, 26], поэтому **целями данной работы** являются:

- краткий анализ имеющихся литературных данных о характере и особенностях реакций, происходящих при закачке в пласт воздуха, и инициировании в нём окисления углеводородов;
- формулировка основных подходов для создания блока химических реакций при разработке термогидросимулятора для моделирования ТГВ.

2. Состав нефти. Разделение нефти на фракции

Каждая нефть содержит сотни различных углеводородных и неуглеводородных соединений самого разнообразного молекулярного строения. Разделение нефти на эти индивидуальные соединения представляет собой совершенно неподъёмную задачу, а, соответственно, невозможно изучить преобразование нефти при ТГВ через анализ индивидуального поведения этих веществ при окислении и термолизе.

Обычно нефти разделяют на какие-то составные части, **фракции**, каждая из которых состоит также из множества соединений, но свойства и поведение соединений в пределах одной фракции по каким-то параметрам ближе друг к другу, по сравнению с соединениями, входящими в другие фракции.

Подобное разделение нефти на фракции было начато более ста лет тому назад, но способ выделения фракций диктовался интересами нефтепереработки и получения тех или иных топлив или иных нефтепродуктов.

Для моделирования процессов, происходящих при внутрипластовом горении или ТГВ, все подобные разбиения пластовой нефти, конечно, могут быть использованы при составлении системы уравнений, описывающих процесс окисления нефти. Но основано такое фракционирование на различии температуры кипения разных фракций, а не на характере или кинетике химических реакций.

Поэтому для анализа процессов, происходящих при внутрипластовом окислении нефти, такое разделение на фракции практически не используется.

Весьма популярно при исследованиях свойств и составов нефтей другое разбиение нефти на фракции, именуемое в зарубежной литературе **SARA** (saturates, aromatics, resins, asphaltenes), т.е. разделение нефти на насыщенные УВ (алканы), ароматические соединения (арены), смолы и асфальтены [27]. В этом методе, предложенном ещё в начале XX века, разделение идёт по растворимости различных фракций нефти (см. рис. 1 [27]). Асфальтены отделяются от остальных фракций путём добавления в нефть какого-нибудь алканового растворителя, например, н-гептана, в котором асфальтены не растворяются и выпадают в осадок. Оставшийся раствор пропускают через колонку с сорбентом (силикагелем или оксидом алюминия, например), на котором и отлагаются оставшиеся три фракции. На следующем шаге каждую из них вымывают из сорбента соответствующим растворителем. Алканы вымываются раствором н-алкана, арены – толуолом и т.п.

Всё большее число исследователей склоняются к использованию такого разделения нефти на фракции при численном моделировании химических процессов, происходящих при ТГВ [26-34]. Это даёт некоторую более-менее ясную основу и для лабораторного изучения поведения нефти при термолизе и окислении, и для их последующего численного моделирования.

Достоинство подхода, основанного на выделении SARA-фракций, состоит в том, что:

- они могут быть выделены реально, по известной химической методике, как из начальной нефти, так и из продуктов реакций крекинга, окисления; это даёт некоторую «основу» и при конструировании модели химических превращений

нефти при ВГ и ТГВ, и при лабораторных исследованиях этих превращений;

Рис. 1. Схема разделения нефти на фракции растворителями и сорбентом (SARA) [27]

- введение этих фракций вполне совместимо с известными методами введения псевдокомпонентов при моделировании фазовых переходов [26] и имеется ряд примеров успешного моделирования фазового поведения углеводородов при использовании SARA–фракций [31-34];
- реальное выделение этих фракций из пластовых углеводородных смесей может быть выполнено с помощью достаточно простых химических процедур и посильно обычной химической лаборатории, специализирующейся в органической химии;
- при некоторых условиях число фракций может быть уменьшено с четырёх до двух – мальтены и асфальтены [26], хотя такой подход может быть оправдан лишь в ограниченном числе случаев [30].

Поэтому на сегодняшний день подход к моделированию окисления и термолиза нефти, основанный на выделении SARA-фракций, представляется более предпочтительным и логичным.

3. Подход к моделированию химических реакций, основанный на выделении SARA-фракций

Этот подход принят в значительном (и всё возрастающем) числе публикаций за последние два десятилетия. Рассмотрим, для примера, некоторые из них.

Пример 1. В работе [35] рассмотрено моделирование как химических реакций, так и всего процесса высокотемпературного окисления битумов Атабаски.

В качестве жидких компонентов битума рассматривались мальтены (объединение фракций *saturates, aromatics и resins)* и *асфальтены*; приняты следующие схемы их термического крекинга:

мальтены → асфальтены,
асфальтены → кокс,
асфальтены → газ.

Для низкотемпературного окисления рассматривались следующие схемы реакций:

мальтены + кислород → асфальтены,
асфальтены + кислород → кокс.

Естественно, используя формулу Аррениуса, определили константы скоростей всех этих реакций.

Используя как результаты определения плотностей исследуемого битума, его мальтенов и асфальтенов, а также литературные данные по молекулярным массам этих фракций, определили стехиометрические коэффициенты указанных реакций крекинга:

мальтены → 0.372 асфальтены,
асфальтены → 83.223 кокс,
асфальтены → 37.683 газ.

На основании литературных данных о потреблении кислорода при окислении мальтенов и асфальтенов рассматриваемых битумов, а также о значениях их молекулярных масс, сконструировали следующие уравнения реакций окисления:

мальтены + 3.431 кислород → 0.4726 асфальтены,
асфальтены + 7.513 кислород → 101.539 кокс.

Для горения кокса использовано уравнение однокомпонентного полного горения, а также предположение, что отношение CO_2/CO равно 8.96, на том основании, что конечные результаты моделирования мало зависят от величины этого параметра. В этом случае уравнение горения кокса (обозначаемого как СК) приняло вид:

$$CK\ (CH=1.13) + 1.232\ O_2 \rightarrow CO_x + 0.565\ H_2O.$$

Энтальпия реакций термического крекинга предполагалась равной нулю, а теплоты реакций окисления взяты из литературных данных.

Температурная зависимость вязкости битума и мальтенов оценивалась экспериментально:

$$\mu_{bitumen} = 0.48267 \times 10^{-6}\, exp(7685.2/T),$$

$$\mu_{malt} = 0.19359 \times 10^{-4}\, exp(5369.2/T),$$

а вязкость асфальтенов определялась, исходя из правила смешения:

$$\mu_{bitumen} = (\mu_{malt})^{x_{malt}} \cdot (\mu_{asp})^{x_{asp}},$$

где x – молекулярные доли соответствующих фракций, откуда:

$$\mu_{asp} = 4.892 \times 10^{-25}\, exp(33147/T).$$

Плотность асфальтенов и мальтенов оценена равной 1.1580 и 0.9832 г/см^3 соответственно; плотность кокса – 1.380 г/см^3. Снижение летучести в результате окисления учитывалось выбранным значением нулевой летучести асфальтенов.

При этих предположениях и с использованием оценённых значений иных необходимых параметров - теплоёмкостей, теплопроводностей, константы равновесия для мальтенов, относительных фазовых проницаемостей численно моделировали данные экспериментов в реакторах длиной от 25 до 183 см, диаметром от 5 до 10 см. Число расчётных блоков составляло от 3 (в минимальном случае) до 36.

На рис. 2, 3 показаны результаты расчёта для реактора малой длины и минимального диаметра, в котором проводился эксперимент по методике *ramped temperature oxidation [RTO]*.

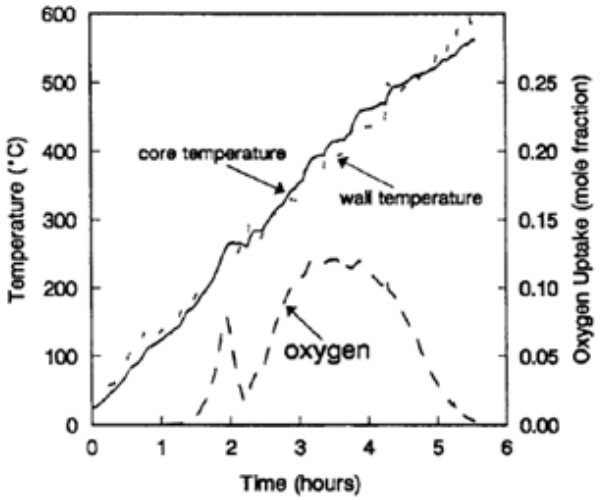


Рис. 2. Потребление кислорода и температура в зависимости от времени в эксперименте RTO [35]

Первый пик потребления кислорода при малых временах связан с окислением мальтенов, а более поздний – с окислением асфальтенов и горением кокса.

Ситуация кардинально меняется, если реакции низкотемпературного окисления не учитываются, как следует из рис. 4 и 5.

Эти расчёты показали крайнюю важность введения низкотемпературного окисления SARA-фракций в число реакций при моделировании.

Пример 2. В работе [36] описана идея расчётной схемы для моделирования ТГВ, основанная на использовании SARA-фракций.

Вводятся 4 фазы в поровом пространстве породы – жидкая водная фаза (*W*), жидкая углеводородная фаза – (*L*), газовая фаза – (*G*), и твёрдая фаза в порах – (*CK*).

Считается, что компоненты фаз *W, L, G* не могут содержаться в фазе *CK*, и наоборот.

Жидкая углеводородная фаза состоит из четырёх компонентов: насыщенные углеводороды (парафины) – *SA*, ароматические углеводороды – *AR*, смолы – *RE*, асфальтены – *AS*. Твёрдая фаза *CK* или «кокс» - считается конечным продуктом крекинга, окисления, и в дальнейшем - в результате горения - переходит в воду и оксиды углерода.

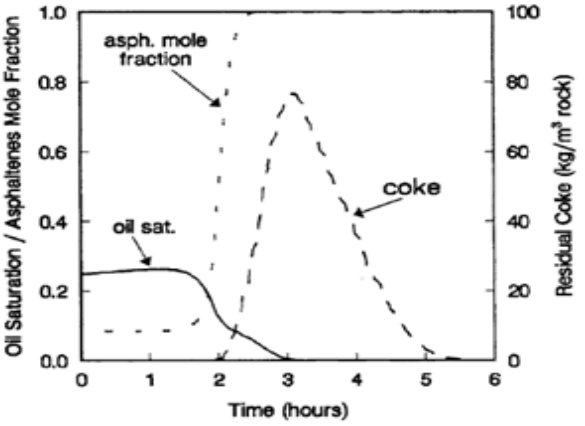


Рис. 3. Содержание нефти, асфальтенов и кокса в зависимости от времени в эксперименте RTO [35]

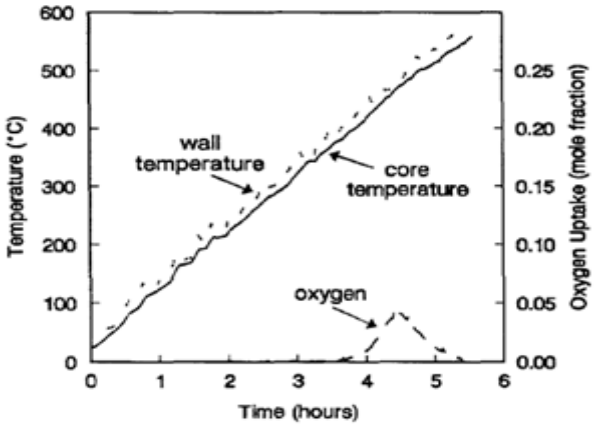


Рис. 4. Потребление кислорода и температура в зависимости от времени в эксперименте RTO без учёта низкотемпературного окисления [35]

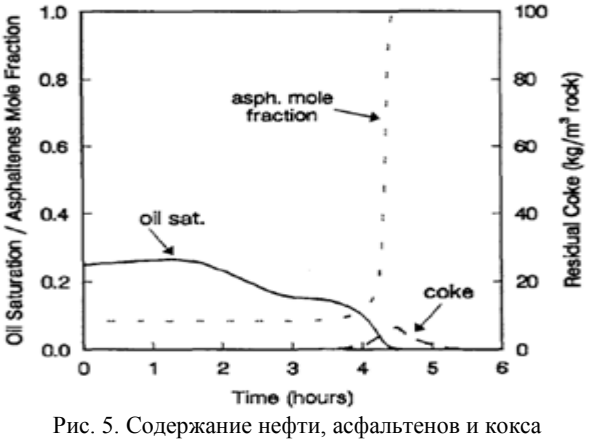


Рис. 5. Содержание нефти, асфальтенов и кокса в зависимости от времени в эксперименте RTO без учёта низкотемпературного окисления [35]

В газовой фазе выделяется метан (CH_4) и отдельный компонент – «другие углеводородные газы» (*HCG*), который вступает в реакции окисления и горения.

Со ссылкой на работу [35] принимаются следующие реакции преобразования фаз:

- термический крекинг:

$$AR \rightarrow 0.203\ AS,$$
$$AS \rightarrow 153.05\ CK,$$
$$AS \rightarrow 16.24\ HCG + SA,$$

- низкотемпературное окисление нефти в жидкой фазе:

$$AR + 1.71\ O_2 \rightarrow 0.481\ RE,$$
$$RE \rightarrow 0.4723\ AS,$$
$$0.48\ RE + 1.71\ O_2 \rightarrow 0.258\ AS,$$
$$AS + 13.78\ O_2 \rightarrow 0.94\ SA + 141.13\ CK + 2.39\ HCG,$$
$$SA + 17.62\ O_2 \rightarrow 14.29\ CO_2 + 17.56\ H_2O,$$

- высокотемпературное окисление «кокса», т.е. остаточного топлива:

$$CK\ (CH=1.13) + 1.232\ O_2 \rightarrow CO_x + 0.565\ H_2O,$$

- горение летучих УВ компонентов:

$$CH_4 + 2\ O_2 \rightarrow CO_2 + 2\ H_2O,$$
$$HCG + O_2 \rightarrow 0.9695\ CO + CO_2 + 2\ H_2O.$$

Создание вычислительной схемы, программного продукта, настройка данной системы на результаты лабораторных экспериментов целью данной, чисто постановочной, работы не являлись.

Предложение авторов работы [36] вызывает ряд серьёзных вопросов. В самой работе [35], так и во многих других, исследовались конкретные углеводороды – битумы канадской провинции Атабаска. В то же время, уже известно, что характер реакций зависит от состава нефти и от минерального состава твёрдой фазы породы. Поэтому использование приведенной в [36] системы реакций для иных условий в качестве сколько-нибудь универсального инструмента вряд ли допустимо. Соответственно, для условий баженовской свиты и аналогичных отложений в РФ конкретный вид именно этой, жёстко определённой, системы уравнений вряд ли приемлем.

Пример 3. Краткий обзор ранних исследований реакций окисления и пиролиза SARA-фракций приведен в работе [37], в первую очередь для асфальтенов.

Отмечено, что асфальтены участвуют в реакциях низкотемпературного окисления, а экзотермические же реакции с этой фракцией идут лишь при очень высоких температурах.

В низкотемпературной области в асфальтенах не получено никакой потери массы за счёт дистилляции или низкотемпературного окисления, тогда как во фракции *saturates* в области низкотемпературного окисления наблюдалась значительная потеря массы.

В низкотемпературной области какая-то часть тепловыделения обусловлена реакциями с асфальтенами, но в высокотемпературной области именно они являются основным генератором тепла.

В ряде экспериментов получено, что тепловыделение реакций с *saturates* и *aromatics* выше, чем реакций со смолами и асфальтенами.

При исследовании чистых парафинов и их смесей с нефтями получено, что образцы парафинов обнаруживают наибольшую активность в низкотемпературной области, и именно в этой области при окислении выделяется наибольшее количество тепла. При калориметрии температурные пики в низко- и высокотемпературной областях с ростом молекулярной массы парафинов сдвигаются в сторону более высоких температур и тепловыделение возрастает.

Пример 4. Наиболее полное исследование реакций термолиза нефти было выполнено в работе [29] с использованием разделения жидких углеводородов на SARA-фракции. **Исследование кинетики проводили как для исходных нефтей, так и для каждой из четырёх SARA-фракций.**

Исследования проводили для двух, достаточно тяжёлых, нефтей: одна – плотности 0.9595 г/см^3 и вязкостью 1700 мПа·с при 20 ^{0}C, вторая – плотности 1.0115 г/см^3 и вязкости 319000 мПа·с при 20 ^{0}C.

Все эксперименты проводили на смесях углеводородных компонентов с керновым материалом. Последний был получен из реальных кернов после тщательной промывки растворителями, прокаливания (для удаления остаточной нефти) и промывки от солей.

Обычно исследования проводили до температур 400-425 ^{0}C и при давлениях 195-200 кПа. Особое внимание было обращено на исключение различного рода экспериментальных погрешностей.

При анализе результатов предполагался первый порядок реакции по отношению к массе исследуемой фракции и формула Аррениуса для температурной зависимости скоростей всех реакций.

При исследовании рассматривался пиролиз каждой SARA-фракции каждой из двух нефтей. В продуктах пиролиза выделялись следующие компоненты: газ, лёгкие углеводороды (с температурой кипения ниже 220 ^{0}C), алканы (*saturates*), арены (*aromatics*), смолы (*resins*), асфальтены (*asphaltenes*), кокс (остаток, не растворимый в толуоле), вода.

Анализ газового компонента показал, что он состоял из следующих газов: метан, этилен, этан, пропилен, пропан, C_{4+}, CO_2, CO, H_2S, H_2.

Обозначим SARA-фракции через *SA*, *AR*, *RE*, *AS*, кокс – через *CK*, а отличающиеся от них компоненты продуктов пиролиза через *G* (газ), *LEN* (лёгкие углеводороды), *W* (вода). При экспериментальном разделении нефти на SARA-компоненты часто наблюдается утечка летучих соединений. Для коррекции такой утечки и введен специальный компонент *LEN*, содержащий лёгкие углеводороды с температурами кипения ниже 220 ^{0}C; согласно дополнительным исследованиям он содержал в основном лёгкие соединения фракции *SA* и *AR*.

С учётом указанного разделения флюидов на компоненты, для каждой SARA-фракции можно записать уравнение реакции пиролиза. Окончательный вид этих уравнений находится из анализа результатов эксперимента.

Например, для одной из исследованных нефтей получена следующая окончательная система

уравнений реакций пиролиза (коэффициенты даны не в мольных, а в весовых долях):

SA → 0.08 G + 0.62 LEN + 0.3 AR,

AR → 0.01 G + 0.18 LEN + 0.03 SA + 0.78 RE,

RE → 0.05 G + 0.16 LEN + 0.22 SA + 0.28 AR + 0.29 AS,

AS → 0.04 G + 0.02 LEN + 0.10 SA + 0.04 AR +
+ 0.03 RE + 0.77 CK.

Для каждой из SARA-фракций были определены предэкспоненциальный множитель и энергия активации в формуле Аррениуса для скорости реакции пиролиза.

Пример 5. В работе [28] аналогичное исследование проведено для изучения реакций низкотемпературного окисления SARA-фракций нефтей и нефтей в целом.

Качественное исследование реакций окисления проводилось с помощью дифференциальной сканирующей калориметрии высокого давления (ДСК). Пробы от 3 до 5 мг изучались при постоянном давлении, начальная температура составляла 40 ^{0}C и поднималась до 580 ^{0}C со скоростью 10 ^{0}C/мин. Измерялась скорость тепловыделения как функция температуры.

Количественное изучение окисления выполняли с помощью трубного реактора, аналогичного тому, который использовался для исследования пиролиза. Теми же были и образцы исследованных нефтей.

С помощью ДСК-измерений исследовалось, можно ли окисление нефти представить как сумму реакций окисления её SARA-фракций.

В целом, было получено вполне удовлетворительное совпадение кривой калориметрии для нефти и кривой, полученной суммированием кривых для разных фракций.

Оказалось, что для смесей фракций *AR, RE, AS* (без *SA*) эта аддитивность практически справедлива. Но окисление взятой отдельно фракции *SA* происходит иначе, чем в смеси с другими фракциями.

При этом если фракция *SA* смешивается с 10% фракции *AR*, то окисление фракции *SA* имеет место, хотя происходит и слабее, чем чистой фракции *SA*. Но при наличии в смеси 50% или более фракции *AR* окисление фракции *SA* практически прекращается. Тот же эффект отмечен в смеси алканов и смол (*SA* и *RE*). Возможно, такой эффект связан с тем, что окисление фракции *SA* происходит, в основном, в газовой фазе (как в работе [25]), а испарение этой фракции в смеси с более высокомолекулярными фракциями затруднено по сравнению со случаем чистой фракции *SA*.

В тщательных экспериментах по изучению окисления фракции *saturates* установлено наличие индукции. При окислении более тяжёлых фракций индукция не наблюдалась, что качественно соответствует другим сведениям [25, 38].

При исследованиях на трубном реакторе окисления чистых фракций *SA*, *RE*, *AS* были определены скорости реакций окисления и параметры в формуле Аррениуса.

Было получено, что порядок реакций по кислороду не является постоянной величиной, а растёт примерно с 0.5 до 1 для всех трёх указанных фракций с ростом температуры от 150 до 230 ^{0}C.

Также наблюдался период индукции для реакций окисления фракции *SA* (алканы).

Пример 6. В работе [39] выполнено исследование способов описания окисления SARA-фракций в низкотемпературной области для битумов Атабаски. Использованы ранее полученные в Университете Калгари результаты изучения в изотермических условиях зависимости концентраций продуктов реакций и отложения кокса от температуры в интервале 60–150 ^{0}C, а также исследования RTO, включая оценку теплоты реакций при низкотемпературном окислении.

Предполагалось, что: горение является полным – до CO_2; газовая фаза *G*, возникающая при крекинге асфальтенов, состоит из углеводородов $C_6 – C_8$ и имеет молекулярную массу, равную 100.

В качестве начальной модели использовалась модифицированная система реакций, предложенная в работе [40]. Вводились 4 реакции:

$$AR + 4.79\ O_2 \rightarrow 0.28\ AS,$$

$$RE + 6.01\ O_2 \rightarrow 0.57\ AS,$$

$$AS \rightarrow 0.78\ SA + 116.14\ CK + 1.99\ G,$$

$$SA + 38.34\ O_2 \rightarrow 24.25\ CO_2 + 30.06\ H_2O.$$

Стехиометрические коэффициенты даны в молярных единицах. Молекулярные массы и плотности SARA-фракций были взяты из литературных данных.

По оценённым молекулярным массам с помощью программы WINPROP был рассчитан компонентный состав как самих SARA-фракций, так и газов (O_2, N_2, CO_2). Кокс (*CK*) принимался твёрдым веществом молекулярной массы 13 и плотности 1.4 г/см^3. Значения вязкостей SARA-фракций оценивались по величинам вязкости нефтяных фракций близкой молекулярной массы, опубликованным в литературе.

На основании этих данных и симулятора STARS компании CMG была сформулирована модель, позволявшая численно смоделировать результаты экспериментов по низкотемпературному окислению SARA-фракций.

Для моделирования «задержки» в образовании кокса из асфальтенов использовали опцию в программе STARS, именуемую Partial Equilibrium Reactions. Именно эта опция позволяет вычислить пороговую концентрацию, с которой начинается моделируемая реакция.

В результате значительного числа расчётов было выяснено, что первоначальная система реакций должна быть изменена и в неё необходимо ввести низкотемпературные реакции с присоединением кислорода и исключить реакцию с горением фракции *SA*. Выяснилось также, что необходимо явно ввести новый, промежуточный, компонент – RS_1, образованный при реакциях присоединения

кислорода, и предваряющий реакцию полного окисления ароматических углеводородов и смол.

Итоговая, уточнённая, модель химических реакций выглядела следующим образом:

a.) низкотемпературное окисление фракции *AR* с образованием промежуточных кислородсодержащих соединений

$$AR + 0.26\ O_2 \rightarrow 0.10\ RS1 + 0.85\ AR,$$

b.) и c.) реакции низкотемпературного окисления

$$AR + 0.10\ RS_1 + 11.7\ O_2 \rightarrow 0.30\ AS + 3.1\ CO_2 + \\ + 3.78\ H_2O,$$

$$RS + 0.10\ RS_1 + 25.92\ O_2 \rightarrow 0.67\ AS + 7.17\ CO_2 + \\ + 8.77\ H_2O,$$

d.) термокрекинг асфальтенов

$$AS \rightarrow 0.78\ SA + 116.4\ CK + 4.52\ CO_2.$$

В результате было достигнуто полное соответствие экспериментальных и расчётных профилей температуры вдоль реактора, характера и количеств полученных продуктов.

Результаты работы [39] свидетельствуют о том, что:

- **реальная модель химических реакций может быть определена только в результате многопланового экспериментального изучения реакций термолиза и окисления для конкретной нефти,**
- необходимо сочетать лабораторные эксперименты с их численным моделированием, и добиться согласования расчётных и экспериментальных величин на установках малого пространственного масштаба, до того, как переходить к расчётам в масштабе межскважинного пространства,
- фазовое поведение отдельных SARA-фракций нефти может быть охарактеризовано с помощью существующих программ (STARS, WINPROP) и известных подходов описания фазового поведения пластовых углеводородных систем,
- при описании реакций нефти с закачиваемым воздухом нельзя пренебрегать рассмотрением реакций низкотемпературного окисления; весьма вероятно, что для описания низкотемпературного окисления нефтей необходимо вводить промежуточные продукты – пероксиды, образуемые в реакциях присоединения кислорода,
- существующие опции программы STARS позволяют учесть «задержку» в образовании кокса из асфальтенов, и этот параметр также является одним из варьируемых, наряду с кинетическими параметрами выбранных реакций.

Пример 7. Разумеется, в действительности все реакции с ростом температуры участка пласта происходят с каждой из фракций нефти как последовательно, так и параллельно. Например, в работе [141] экспериментально получено, что для асфальтеновой фракции вязкой нефти одного из месторождений в провинции Саскачеван (Saskatchewan, Canada) при повышении температуры наблюдалось протекание следующих реакций:

- ✓ испарение + перегонка + НТО, 40–290 ^{0}C,
- ✓ перегонка + висбрейкинг + НТО, 290–400 ^{0}C,
- ✓ термический крекинг + СТО, 400–480 ^{0}C,
- ✓ термический крекинг + горение, 480–560 ^{0}C,
- ✓ коксование + горение + разложение некоторых минералов твёрдой части породы, 560-750 ^{0}C,

здесь НТО – низкотемпературное окисление; висбрейкинг – низкотемпературный крекинг, реакции расщепления парафиновых и нафтеновых углеводородов с образованием углеводородных газов, бензиновых фракций и фракций кипящих при 200–450 ^{0}C, вторичных асфальтенов; СТО – среднетемпературное окисление.

Как следует из приведенных литературных данных, введение реальных SARA-фракций является достаточно перспективным методом фракционирования нефти для описания химических реакций окисления и термолиза нефти при ТГВ по следующим причинам:

1. При таком подходе мы получаем реальные жидкости, для которых можно экспериментально определить молекулярную массу и плотность, что позволяет оценить константы равновесия или выбрать подходящие уравнения состояния; возможно и прямое проведение PVT-экспериментов, по крайней мере, с тремя фракциями, кроме асфальтенов. Это позволяет корректно моделировать фазовое поведение рассматриваемой нефти. Вязкости фракций и их зависимости от температуры также могут быть определены экспериментально, а не рассчитаны или взяты по аналогии, что также повышает достоверность описания фильтрации нефти в целом.

2. Разделение на указанные фракции соответствует и общим сведениям о характере химических реакций. Если фракция *saturates* более других ответственна за поведение нефти при сравнительно низких температурах, то фракция *asphaltenes* определяет характер химических реакций в высокотемпературном диапазоне. Таким образом, проведя исследования с чистыми фракциями, мы можем лучше понять и достовернее моделировать реакции крекинга и окисления нефти во всём интервале температур, характерных для внутрипластового горения и ТГВ.

3. Хотя есть данные, что скорости реакций фракции *saturates* в смеси с другими фракциями отличаются от значений для чистой фракции, для остальных трёх фракций такого не наблюдалось. Даже при наличии этого фактора, он, в принципе, может быть учтён при моделировании, если есть достаточное количество экспериментальных данных по изучению реакций этой фракции в смеси с другими. Это возможно, так как при гидродинамическом моделировании на каждом временном шаге в каждой точке пласта рассчитывается содержание каждой из фракций.

4. Тем не менее, даже при введении SARA-фракций, остаётся значительный произвол в выборе системы происходящих реакций. Хотя в большинстве работ исследовались сходные образцы, битумы Атабаски, например, - разные исследователи приходили к конструированию достаточно отличающихся систем реакций.

Заключение

1. Реально сконструировать подходящую систему реакций для численного моделирования можно только на основе результатов многопланового экспериментального исследования термолиза и окисления конкретной нефти и её четырёх SARA–фракций, для конкретного состава твёрдой фазы породы изучаемого объекта – в нашем случае, для отложений типа баженовской свиты.

2. Эти эксперименты обязательно должны охватывать как область низких, так и высоких температур (т.е. от 20 до 500 ^{0}C), а также давлений от 15 до 40 МПа, поскольку в обязательном порядке должны включать и низко- и высокотемпературное окисление.

3. Экспериментальное исследование должно включать изучение следующих реакций, как самой нефти, так и её SARA–фракций:

- высокотемпературное горение,
- термический крекинг, с учётом наличия дистилляции и коксования,
- низкотемпературное окисление.

4. Блок химических реакций должен быть универсален и предоставить исследователю возможность учесть различные варианты реакций окисления и горения нефти и её фракций, которые возникнут в результате эксперимента**.**

Oil chemical reactions in HPAI oil field development

V.A. Yudin, A.V. Korolev, I.V. Afanaskin, S.G. Volpin

Abstract: A brief review of oil chemical reactions in HPAI oil field development is presented. Some typical oxidation and thermolysis models of oil and its SARA-fractions are described. The need of special wide scale experimental investigation of oxidation and thermolysis is emphasized, using oil, its SARA-fractions and mineral rock matrix of formations under development.

Keywords: HPAI, in-situ combustion, formation oil oxidation, formation oil thermolysis, bazen suite.

Литература

1. BP Energy Outlook 2035. January 2014. http://www.slideshare.net/BP_plc/bp-energy-outlook-2035-2014-booklet
2. Paul Chefurka. World Energy and Population, 2007. http://www.wprr.ru/mirovaya-energiya-i-naselenie-perspektivy-s-2007-po-2100-gg
3. А.М. Мастепанов. Топливно-энергетический комплекс России на рубеже веков: состояние, проблемы и пути решения. Том 1. М., ИАЦ «Энергия», 2009.
4. М.Д. Белонин, Ю.В. Подольский. Состояние сырьевой базы и прогноз возможных уровней добычи нефти в России до 2030 г. «Минеральные ресурсы России», 2006, № 5. http://www.vipstd.ru/gim/content/view/88/279
5. В.П. Якуцени, Ю.Э. Петрова, А.А. Суханов. Динамика доли относительного содержания трудноизвлекаемых запасов нефти в общем балансе. «Нефтегазовая геология. Теория и практика», 2007(2). www.ngtp.ru
6. А.М. Брехунцов, И.И. Нестеров. Нефть битуминозно-глинистых и карбонатно-кремнисто-глинистых пород. «Инновационные технологии оценки, моделирования и разработки залежей нефти баженовской свиты: Научно-практическая конференция им. Н.Н. Лисовского - 28 сентября 2010».
7. Ю.Е. Батурин, В.П. Сонич, А.Г. Малышев, А.А. Кошелева. О возможном пути интенсификации нефтеизвлечения из отложений баженовской свиты. «Освоение ресурсов трудноизвлекаемых и высоковязких нефтей: Международная конференция – Краснодар, 1999».
8. Ю.Е. Батурин, В.П. Сонич, А.Г. Малышев, О.Г. Зарипов, В.Г. Шеметило. Проблемы и перспективы освоения баженовской свиты. «Нефтяное хозяйство», 2001, № 9.
9. В.В. Ананьев, В.М. Смелков, Н.В. Пронин. Прогнозная оценка ресурсной базы мендым-доманиковых отложений как основного источника углеводородного сырья центральных районов Волго-Уральской нефтегазоносной провинции. «VIPStudio ИНФО», 2007. http://www.vipstd.ru/gim/content/view/459/
10. В.И. Грайфер, А.А. Боксерман, Н.М. Николаев, В.И. Кокорев, О.В. Чубанов. Интеграция тепловых и газовых методов увеличения нефтеотдачи – основа технико-технологического комплекса разработки месторождений нетрадиционных ресурсов и трудноизвлекаемых запасов нефти. «Rusnanotech: Международный форум по нанотехнологиям – Москва, 2010».

11. В.И. Кокорев. Технико-технологические основы инновационных методов разработки месторождений с трудноизвлекаемыми и нетрадиционными запасами нефти. Дисс. на соиск. уч. степени докт. техн. наук. М., ИПНГ РАН, 2010.
12. А.А. Боксерман, В.А. Савельев, И.С. Джафаров, А.Г. Соломатин, Д.Т. Миронов. Термогазовое воздействие – инновационная технология разработки месторождений Сибири. «Энрекон-2010: Международная конференция – Москва, 2010».
13. М.Ф. Ямбаев. Основные особенности термогазового метода увеличения нефтеотдачи применительно к условиям сложнопостроенных коллекторов (на основе численного моделирования). Дис. насоиск. уч. степени канд. техн. наук. М., ОАО «ВНИИнефть им. ак. А.П. Крылова», 2006.
14. А. Шпильман. Ввод в разработку таких месторождений, как Имилорское, им. В.И. Шпильмана и Гавриковское, может стабилизировать добычу нефти в Югре. «Агенство нефтегазовой информации», 29.11.2011. http://www.angi.ru/news.shtml?oid=2782534
15. А.А. Боксерман, В.И. Грайфер, В.И. Кокорев, О.В. Чубанов. Термогазовый метод увеличения нефтеотдачи. «Интервал», 2008, № 7, 6 - 33.
16. В.В. Плынин. Термогазовый метод и баженовская свита. «Интернет-портал сообщества ТЭК EnergyLand.info». http://www.energyland.info/analitic-show-50375
17. А. Соломатин. Термогазовое воздействие и месторождения Сибири. «Интернет-портал сообщества ТЭК EnergyLand.info». http://www.energyland.info/analitic-show-52541
18. Partha S. Sarathi. In-situ combustion handbook - principles and practices. Final Report, November 1998. Performed Under Contract No. DE-AC22-94PC91008 (Original Report Number NIPER/BDM-0374). «BDM Petroleum Technologies», BDM-Oklahoma, Inc. Bartlesville, Oklahoma, National Petroleum Technology Office U. S. DEPARTMENT OF ENERGY, Tulsa, Oklahoma.
http://repository.icse.utah.edu/dspace/bitstream/123456789/5336/2/DOE-PC-91008-0374-OSTI_ID-3175-.pdf
19. Д.Г. Антониади, А.Р. Гарушев, Б.Г. Ишханов. Настольная книга по термическим методам добычи нефти. Краснодар, «Советская Кубань», 2000.
20. С.Г. Вольпин, В.А. Юдин, Р.М. Кац, И.В. Афанаскин, В.А. Галкин. Применение суперкомпьютерных технологий – ключ к решению проблем повышения нефтеотдачи на месторождениях России. «Наука и общество: Новые технологии для новой экономики России: С.-Петербургский научный форум, СПб., 30.09 – 04.10.2014».
21. В.Б. Бетелин. Экзафлопные вычисления и энергетическая безопасность США в период 2010–2020–2030 гг. «Энергия», № 3, 2011.
22. В.Б. Бетелин. «Цифровое месторождение» — путь к трудноизвлекаемым запасам углеводородов. «Наука и общество: Новые технологии для новой экономики России: С.-Петербургский научный форум, СПб., 2013».
23. В.Б. Бетелин, А.А. Боксерман, В.Е. Костюков, В.А. Савельев. Проблемы управления процессами повышения нефтеотдачи на основе моделирования на супер-ЭВМ. «НефтеГазоПромысловый Инжиниринг», 3 кв., 2010.
24. И.В. Афанаскин. Повышение технологической эффективности метода направленной закачки воздуха в нефтяные пласты на основе численного моделирования и результатов гидродинамических исследований скважин. Дисс. на соиск. уч. степени канд. техн. наук. М., ОАО «ВНИИнефть им. ак. А.П. Крылова», 2013.
25. Y. Barzin, R.G. Moore, S.A. Mehta, M.G. Ursenbach, F. Tabasinejad.A Comprehensive Kinetics Model for Light Oil Oxidation/Combustion Reactions under High Pressure Air Injection Process (HPAI). SPE 166483, 2013.
26. D. Gutierrez, R.G. Moore, M.G. Ursenbach, S.A. Mehta. The ABCs of In-Situ-Combustion Simulations: From Laboratory Experiments to Field Scale. SPE- 148754, 2012.
27. К. Акбарзаде, А. Хаммами, А. Харрат и др. Асфальтены: проблемы и перспективы. «Нефтегазовое обозрение», 2007. http://www.slb.ru/userfiles/file/Oilfield%20Review/2007/summer/3%20Asphaltenes.pdf
28. N.P. Freitag, B. Vercoczy. Low-Temperature Oxidation of Oils in Terms of SARA Fractions Why Simple Reaction Models Don't Work. «Journal of Canadian Petroleum Technology», Vol.44, №2.
29. N.P. Freitag, D.R. Exelby. A SARA-Based Model for Simulating the Pyrolysis Reactions That Occur in High-Temperature EOR Process. «Journal of Canadian Petroleum Technology», Vol.45, №3.
30. M.R. Kristensen, M.G. Gerittsen, P.G. Thomsen, M.L. Michelsen, E.H. Stenby. Impact of Phase Behavior Modeling in In-Situ Combustion Process Performance. SPE – 113947, 2008.
31. I. Anaya, R.E.L. Cruz, A.J. Alvarez, D. Gutierrez, F.A. Skoreyko, C. Card. Simulation study for Designing an In-Situ Combustion Pilot in the Orinoko Belt in Venezuela: From Laboratory Studies to the Field Scale. SPE – 137491, 2010. http://dx.doi.org/10.2118/137491-MS.
32. M.R. Kristensen, M.G. Gerittsen, P.G. Thomsen, M.L. Michelsen, E.H. Stenby. An Equation-of-State Compositional In-Situ Combustion Model: A Study of Phase Behavior Sensitivity. «Transport Porous Media», 76 (2), 219 - 246. http://dx.doi.org/10.1007/s11242-008-9244-6
33. M. Greaves et all. Estimation of SARA Fraction Properties With the SRK EOS. «J. Can. Technol.», 43(9), 31 - 39. PETSOC-04-09-02. http://dx.doi.org/10.2118/04-09-02

34. O. Sabbaghet all. Applying the PR-EoS to Asphaltene Prescipitation from n-Alkane Diluted Heavy Oils and Bitumens. «Energy Fuels», 20 (2), 625 - 634. http://dx.doi.org/10.1021/ef0502709
35. M.G. Ursenbach, D.W. Bennion, J.D.M. Belgrave, R.G. Moore. A comprehensive approach to in-situ combustion modeling. «SPE Advanced Technology Series», 1993, V.1, № 1, 98 - 107.
36. В.Е. Борисов и др. Композиционная неизотермическая модель фильтрации в пористой среде с учетом химических реакций и активной твердой фазы. «Препринты ИПМ им. М.В.Келдыша», 2013, № 91. http://library.keldysh.ru/preprint.asp?id=2013-91
37. Shyamol Chandra Das. A study of oxidation reaction kinetics during an air injection process. Australian School of Petroleum Faculty of Engineering, Computer and Mathematical Science, The University of Adelaide, Adelaide, Australia, 2009. http://digital.library.adelaide.edu.au/dspace/bitstream/2440/60192/1/02whole.pdf
38. Химия горения / под ред. У. Гарднера мл. М., «Мир», 1988.
39. B. Sequera, R.G. Moore, S.A. Mehta, M.G. Ursenbach. Numerical Simulation of In-Situ Combustion Experiments Operated Under Low Temperature Conditions. SPE-132486-PA.
40. N. Jia, R.G. Moore, S.A. Mehta, M.G. Ursenbach, M. Hancock. Kinetic Modelling of Thermal Cracking and Low Temperature Oxidation Reactions. «Petroleum Society's 4th Canadian International Petroleum Conference, 2003».

Оценка фильтрационных и энергетических параметров нефтяных пластов с помощью гидродинамических исследований скважин на двух режимах: теория, моделирование и практика

И.В. Афанаскин[1], П.В. Крыганов[1], С.Г. Вольпин[1], Ю.М. Штейнберг, И.А. Вольпин

1 – кандидат технических наук

Аннотация: Рассмотрены вопросы гидродинамических исследований скважин на двух режимах при различных моделях течения. С помощью интерпретации модельных кривых изучено влияние длительности первого режима работы скважины на погрешность определения параметров пласта. Приведены реальные примеры исследований скважин. Сопоставлены результаты исследований скважин на двух режимах с результатами исследований скважин методом восстановления давления.

Ключевые слова: гидродинамические исследования скважин, исследования скважин на двух режимах.

Введение

Гидродинамические исследования скважин и пластов позволяют с большой степенью достоверности определять важные параметры: текущую фазовую проницаемость, абсолютную проницаемость, пластовое давление, скин-фактор, коэффициент продуктивности скважин, расстояние до границ пласта и пр. Преимуществом гидродинамических исследований скважин является тот факт, что в ходе этих исследования изучают непосредственно процесс фильтрации (замеряют изменение давления и дебита) в отличие от геофизических (изучают акустические, радиоактивные, электромагнитные и тепловые поля) и сейсмических исследований. Следует отметить, что наиболее полную информацию о строении нефтяных пластов можно получить лишь с помощью комплексирования различных видов исследований скважин и пластов, а также многовариантного численного моделирования.

Наиболее информативным видом гидродинамических исследований являются исследования методом восстановления (падения) давления. Этот метод позволяет определять наибольшее количество фильтрационно-емкостных параметров пласта. Кроме того, при анализе исследования методом восстановления давления с помощью диагностических графиков Бурде [1-3] можно с высокой степенью достоверности определить модель пласта (тип пласта коллектора, наличие границ, особенности потока в прискважинной зоне пласта и пр.).

Однако для исследования скважины методом восстановления давления ее необходимо остановить, что приводит к потере в добыче нефти. Кроме того, для корректной интерпретации этого вида исследований необходимо, чтобы скважина «вышла» на радиальный или псевдорадиальный (для горизонтальных скважин и скважин с трещиной гидроразрыва пласта) режим фильтрации. В низкопроницаемых коллекторах для этого необходимо остановить скважину на длительное время.

В последние годы для получения информации о фильтрационно-емкостных свойствах и уточнении геологического строения пласта, наряду с традиционными гидродинамическими исследованиями, все чаще применяются альтернативные методы исследований, позволяющие минимизировать потери в добыче нефти. Эти методы часто не только предоставляют дополнительную информацию о пласте, но и в ряде случаев являются необходимым условием для однозначного решения обратной задачи подземной гидромеханики при интерпретации других более точных видов исследований.

При исследованиях, например, неоднородного коллектора нередко возникает ситуация, когда на диагностическом графике Бурде, построенном по кривой восстановления давления, не выделяется однозначно участок радиальной фильтрации, который позволил бы достоверно определить проницаемость, скин-фактор и пластовое давление. При этом обработка материалов гидродинамических исследований методом наилучшего совмещения с помощью специализированной программы показывает, что получить однозначное решение в этом случае затруднительно. Хорошее совмещение расчетной и фактической кривых забойного давления может быть получено при различных сочетаниях искомых параметров. В частности, диапазон изменения проницаемости может составлять, например, от нескольких десятков до нескольких тысяч мД. Дополнительная информация о любом параметре позволила бы сузить поиск, и соответственно, приблизиться к решению проблемы неоднозначности интерпретации результатов гидродинамических исследований скважин.

В связи с этим предлагается рассмотреть возможность применения методов гидродинамических

исследований с переменным дебитом (без остановки работы скважины). В данной работе будем рассматривать наиболее удобный для интерпретации метод – исследование нефтяной скважины на двух режимах.

1. Теоретические аспекты гидродинамических исследований нефтяных скважин на нескольких режимах

Когда скважину исследуют только на двух режимах с разными дебитами, то как само исследование, так и его интерпретация упрощаются. Исследование на двух режимах позволяет получить информацию о параметрах проницаемость *k*, скин-фактор *s* и пластовое давление p_i без остановки скважины. Часто считают, что при таком виде исследований эффект влияния объема ствола скважины минимизируется или устраняется полностью. На самом деле эффект влияния объема ствола скважины при исследовании на двух режимах длится столько же, сколько и при обычном исследовании скважины методами восстановления давления, падения давления, падения уровня и т.д. [4]. Однако исследование скважины на двух режимах позволяет уменьшить влияние изменения коэффициента объема ствола скважины, что делает этот метод пригодным даже в тех случаях, когда другие виды исследований неприменимы [4].

На рис. 1 схематически показана динамика изменения дебита и давления при исследовании скважины на двух режимах [4, 5]. Допускается использовать последовательность как увеличивающихся, так и уменьшающихся дебитов [4]. Тогда уравнение, описывающее радиальный неустановившийся приток жидкости к скважине в однородном бесконечном пласте на втором режиме работы имеет вид [4, 5]:

$$p_{wf} = m_1^{/}\left[log\left(\frac{t_1+\Delta t}{\Delta t}\right)+\frac{q_2}{q_1}log(\Delta t)\right]+p_{int}, \quad (1)$$

где p_{wf} - забойное давление при работе скважины на втором режиме с дебитом q_2 на момент времени Δt после изменения режима работы скважины, t_1-продолжительность первого режима работы скважины с дебитом q_1, $m_1^{/}$ и p_{int} - постоянные коэффициенты.

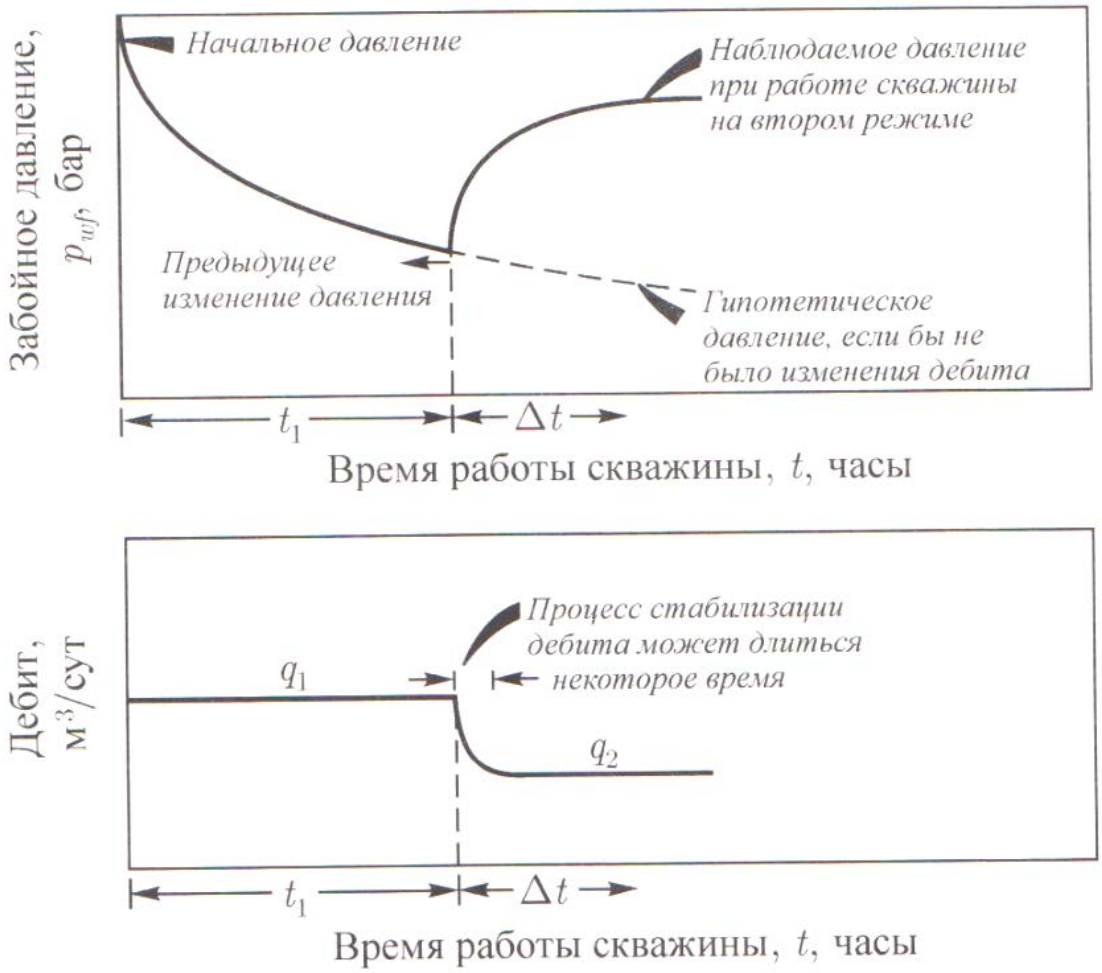


Рис. 1. Схема изменения дебита и забойного давления при исследовании скважины на двух режимах [4, 5]

В общем случае исследования скважины с переменным дебитом уравнение, описывающее радиальный неустановившийся приток жидкости к скважине в однородном бесконечном пласте на *N*-ом режиме работы имеет вид [4]:

$$\frac{p_i - p_{wf}}{q_N} = m' \sum_{j=1}^{N} \left[-\frac{q_j - q_{j-1}}{q_N} log\left(t - t_{j-1}\right) \right] + b' , \quad (2)$$

где

$$m' = \frac{21.5 B\mu}{kh},$$

$$b' = m' \left[\log\left(\frac{k}{\phi \mu c_t r_w^2} \right) - 3.0923 + 0.86859\, s \right],$$

k - проницаемость (мД), h - толщина (м), ϕ - пористость (д.ед.), μ - вязкость (мПа·с), c_t - полная сжимаемость (1/бар), r_w - радиус скважины (м), B - объемный коэффициент ($м^3/м^3$), s - скин-фактор (б/р), p_i - пластовое давление (бар), p_{wf} - давление на N-ом режиме (бар), q_j - дебит на j-ом режиме ($м^3$/сут), t - время (ч). Для интерпретации исследований скважины с переменным дебитом (когда количество дебитов более двух) необходимо знать пластовое давления p_i, поэтому далее они рассматриваться не будут.

Вернемся к исследованию скважины на двух режимах. Из уравнения (1) следует, что график p_{wf} от

$$x = \left[log\left(\frac{t_1 + \Delta t}{\Delta t} \right) + \frac{q_2}{q_1} log\left(\Delta t\right) \right] \quad (3)$$

имеет вид прямой линии с наклоном m_1' и точкой пересечения с осью ординат p_{int}, рис. 2.

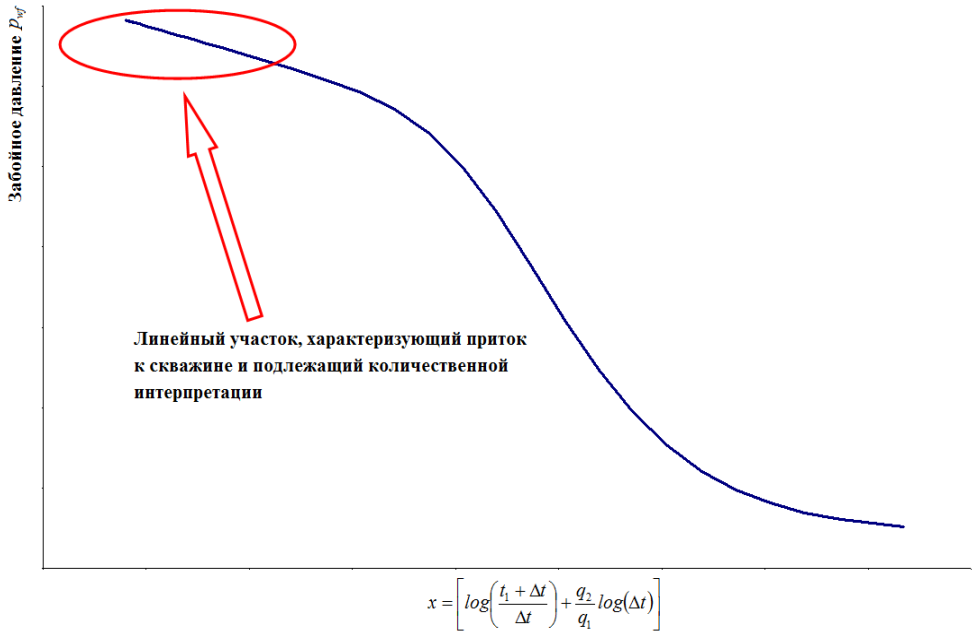


Рис. 2. Схематический график исследования скважины на двух режимах при $q_1 > q_2$ с выделением участка, подлежащего количественной интерпретации. Модель притока жидкости к вертикальной скважине в однородном бесконечном пласте

Определив из указанного графика наклон прямой можно оценить проницаемость пласта [4]:

$$k = -\frac{21.5 q_1 B\mu}{m_1' h}. \quad (4)$$

Тогда скин-фактор скважины можно оценить из выражения [4]:

$$s = 1.1513 \left[\frac{q_1}{q_1 - q_2} \left(\frac{p_{wf}\left(\Delta t = 0\right) - p_{1hr}}{m_1'} \right) - \log\left(\frac{k}{\phi \mu c_t r_w^2} \right) + 3.0923 \right] \quad (5)$$

где

$$p_{1hr} = m_1' \, log\left(t_1 + 1\right) + p_{int}. \quad (6)$$

Пластовое давление можно оценить следующим образом [4]:

$$p_i = p_{int} - \frac{q_2}{q_2 - q_1} \left[p_{wf}\left(\Delta t = 0\right) - p_{1hr} \right]. \quad (7)$$

Напомним, что формулы (4), (5), (7) верны для притока жидкости к вертикальной скважине в однородном бесконечном пласте.

Для случая притока жидкости к вертикальной скважине в однородном пласте с непроницаемой границей, на графике p_{wf} от x появится второй прямолинейный участок с наклоном $2m_1'$. В этом случае проницаемость и скин-фактор определяются из первого прямолинейного участка, а пластовое давление – из второго.

Для случая притока жидкости к вертикальной скважине в однородном пласте с границей постоянного давления, на графике p_{wf} от x после прямолинейного участка появляется участок, параллельный оси x (горизонтальный участок). В этом случае проницаемость и скин-фактор так же определяются из первого прямолинейного участка. Пластовое давление определяется по формуле:

$$p_i = p_2 - 0.87 m_1' \frac{q_2}{q_1} \left[ln\left(\frac{2L}{r_w} \right) + s \right], \quad (8)$$

где L - расстояние до границы (м), p_2 - давление на горизонтальном участке графика p_{wf} от x. Переходный процесс на графике p_{wf} (x) от прямолинейного участка (характеризующего приток к скважине без влияния границы постоянного давления) к горизонтальному участку (характеризующему приток к скважине под

влиянием границы постоянного давления) может быть весьма продолжительным по времени. Определение p_2 может быть затруднено, т.к. по различным причинам исследование скважины может быть завершено раньше, чем давление окончательно стабилизируется и появится искомый горизонтальный участок. Тогда имея переходную зону на графике p_{wf} (x) от прямолинейного участка к горизонтальному участку можно оценить значение p_2 аппроксимировав забойное давление на переходном участке по формуле:

$$p_{wf} = p_2 - ae^{-bx}, \qquad (9)$$

где p_2, a, b - коэффициенты аппроксимации, полученные с помощью метода наименьших квадратов. Имея данные о коэффициенте продуктивности скважины PI (м^3/сут/бар) можно оценить скин-фактор скважины:

$$s = -\frac{1.19q_1}{m_1' PI} - ln\left(\frac{2L}{r_w}\right). \qquad (10)$$

Скин-факторы, оцененные по формулам (5) и (10) должны быть равны.

Расстояние до непроницаемой границы либо до границы постоянного давления можно определить по формуле [4, 7-9]:

$$L = 0.029\sqrt{\frac{kt_x}{\phi\mu c_t}}, \qquad (11)$$

где t_x - время начала отклонения линии p_{wf} (x) от прямой.

В условиях обработки реальных зашумленных данных, для уверенной идентификации модели притока жидкости к скважине по аналогии с исследованием скважины методом восстановления давления [1-3], следует использовать диагностический график, на котором строить две кривые $\Delta p_{wf} = p_{wf}(x(\Delta t)) - p_{wf}(x(\Delta t = 0))$ от x и $\left|d(\Delta p_{wf})/dx\right|$ от x, рис. 3. Появление скачков на левой части производной изменения давления рис. 4 связано с тем, что аргумент дифференцирования x зависит от Δt и график $x(\Delta t)$ имеет минимум, см. формулу (3) и рис. 4.

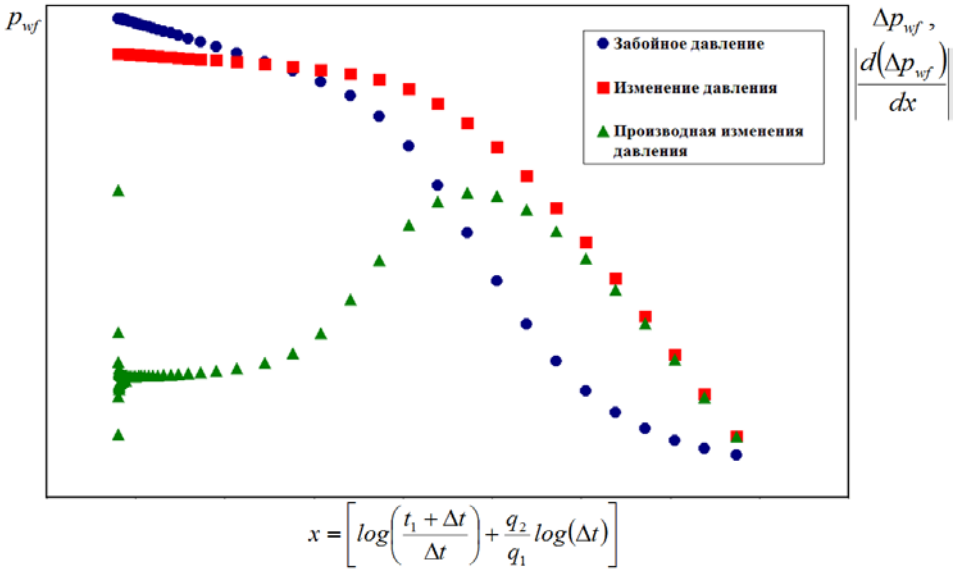


Рис. 3. Схематический график исследования скважины на двух режимах при $q_1 > q_2$ совмещенный с диагностическим графиком. Модель притока жидкости к вертикальной скважине в однородном бесконечном пласте

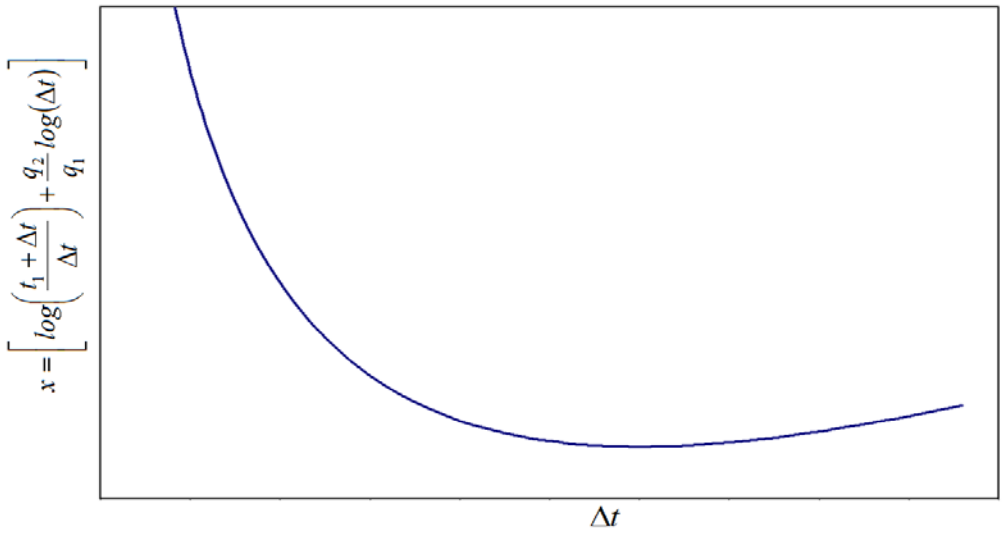


Рис. 4. Схематический график зависимости $x(\Delta t)$ при $q_1 > q_2$

2. Гидродинамические исследования на двух режимах. Обработка модельных кривых изменения давления

Для оценки погрешности приведенной методики исследования с помощью ПО TESTAR [6] на основании точного решения уравнения пьезопроводности с учетом влияния различных условий на границах пласта и влияния ствола скважины были получены гипотетические кривые изменения забойного давления скважины при заданном дебите (решена прямая задача подземной гидромеханики). Для этого приняты следующие исходные данные: длительность первого режима 120 ч., дебит первого режима 150 $м^3$/сут, длительность второго режима 120 ч., дебит второго режима 120 $м^3$/сут, вязкость жидкости 1 сПз, пористость 10%, радиус скважины 0.09 м., объемный коэффициент 1 $м^3/м^3$, толщина пласта 9 м., полная сжимаемость $4.35 \cdot 10^{-5}$ 1/бар, начальное пластовое давление 343.2 бар, значения проницаемости, скин-фактора, а так же модели пласта варьировались. Всего было рассмотрено 4 варианта:

1. Модель однородного бесконечного пласта. Проницаемость 10 мД, скин-фактор -2 б/р, рис. 5.
2. Модель однородного бесконечного пласта. Проницаемость 100 мД, скин-фактор +2 б/р рис. 6.
3. Модель однородного пласта с линейной непроницаемой границей. Проницаемость 100 мД, скин-фактор +2 б/р, расстояние до границы 200 м, рис. 7.
4. Модель однородного пласта с линейной границей постоянного давления. Проницаемость 100 мД, скин-фактор +2 б/р, расстояние до границы 200 м, рис. 8.

В табл. 1 приведены результаты интерпретации гипотетические кривых изменения забойного давления в работающей скважине при гидродинамических исследованиях на двух режимах. Для моделей однородного бесконечного пласта и однородного пласта с линейной непроницаемой границей получено хорошее совпадение заданных при решении прямой задачи и полученных при решении обратной задачи параметров пласта. Для модели однородного пласта с линейной границей постоянного давления совпадение удовлетворительное.

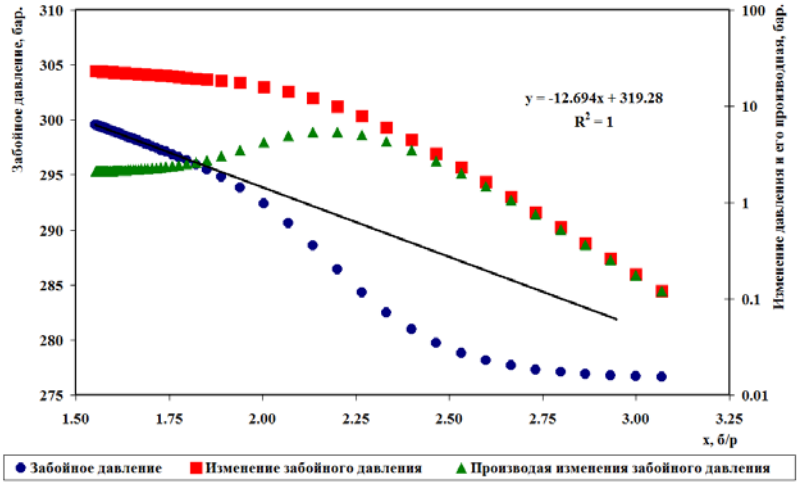


Рис. 5. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Модель однородного бесконечного пласта. Проницаемость 10 мД, скин-фактор -2 б/р

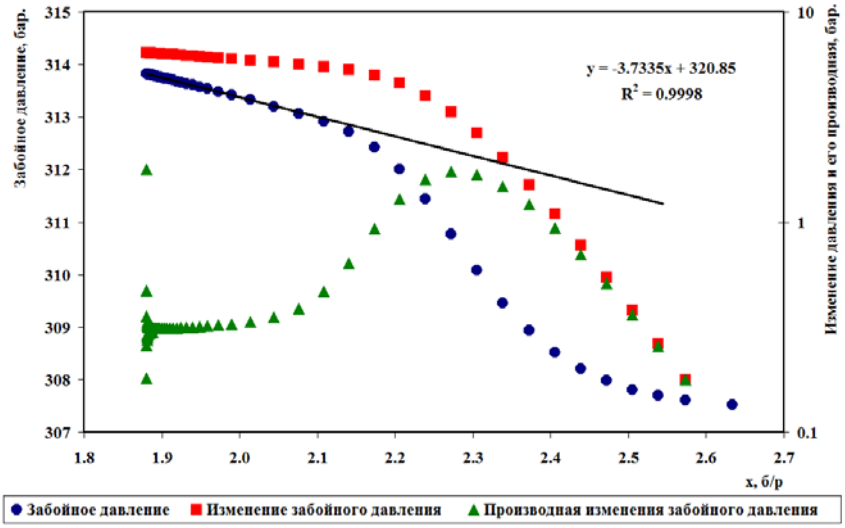


Рис. 6. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Модель однородного бесконечного пласта. Проницаемость 100 мД, скин-фактор +2 б/р

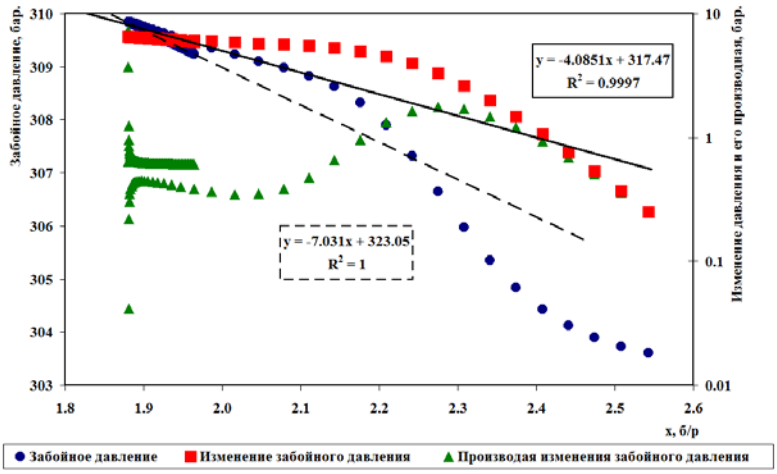


Рис. 7. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Модель однородного пласта с линейной непроницаемой границей. Проницаемость 100 мД, скин-фактор +2 б/р, расстояние до границы 200 м.

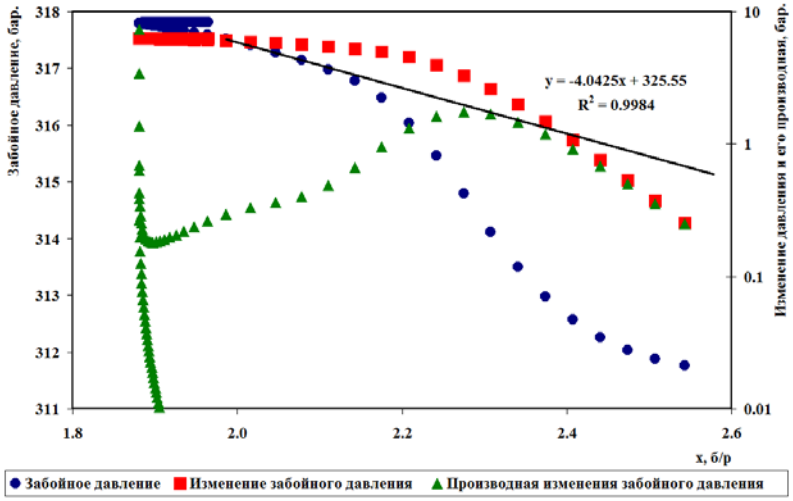


Рис. 8. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Модель однородного пласта с линейной границей постоянного давления. Проницаемость 100 мД, скин-фактор +2 б/р, расстояние до границы 200 м.

Таблица 1

Результаты интерпретации гипотетических кривых изменения забойного давления в работающей скважине при гидродинамических исследованиях на двух режимах

Модель	Проницаемость, мД		Скин-фактор, б/р		Пластовое давление, бар.		Расстояние до границы, м.	
	Факт*	Расчет**	Факт	Расчет	Факт	Расчет	Факт	Расчет
Однородный бесконечный пласт	10	9	-2.0	-2.4		343.8	--	--
Однородный бесконечный пласт		94		+1.4		343.4	--	--
Однородный пласт с линейной непроницаемой границей	100	86	+2.0	+0.7	343.2	343.2	200	178
Однородный пласт с линейной границей постоянного давления		87		+1.1		344.0		165

* - Параметры, которые задавались при решении прямой задачи (получении гипотетических кривых)

** - Параметры, полученные в результате решения обратной задачи

3. Гидродинамические исследования на двух режимах. Влияние длительности первого режима на точность определения параметров пласта

Уравнения для интерпретации исследования скважины на двух режимах, приведенные в разделе 1, получены для случая запуска скважины в работу на первом режиме в невозмущенном пласте. Возникает закономерный вопрос, можно ли пользоваться рассматриваемым методом в случае, когда скважина перед исследованием на двух режимах работала, а не стояла? Ведь рассмотренное решение уравнения пьезопроводности предполагает учет влияния на забойное давление на втором режиме только первого режима и отсутствие влияния всей предыдущей истории работы скважины. При этом с точки зрения метода суперпозиции влияние на забойное давление в скважине оказывает вся предыдущая история ее работы. Для практических целей принято учитывать историю в 6 раз превышающую длительность периода исследования [2, 3].

Для проверки применимости этого правила в рассматриваемом случае с помощью ПО TESTAR [6] были рассчитаны 2 гипотетические кривые забойного давления для модели однородного бесконечного пласта. Параметры пласта: проницаемость 100 мД, скин-фактор +2 б/р, пластовое давление 343.2 бар. Режимы работы скважины приведены в табл. 2 и на рис. 9, 10. Скважина запущена в невозмущенном пласте и работает на 7 режимах. В первом варианте все режимы одинаковой длительности 48 ч. Во втором варианте режимы №№ 1-5 и № 7 имеют длительность 48 ч., а режим № 6 в 6 раз длиннее остальных и составляет 288 ч.

Кривые изменения забойного давления, приведенные на рис. 9 и 10, были представлены, как гидродинамические исследования скважины на двух режимах и соответствующим образом интерпретированы. При этом за первый режим принимался режим № 6, а за второй режим - № 7. Результаты интерпретации приведены на рис. 11, 12 и в табл. 3. Видно, что результаты второго варианта, когда длительность первого режима в 6 раз больше длительности второго, намного лучше согласуются с истинными значениями параметров пласте, чем результаты первого варианта. Кроме того, слишком короткий режим 1 в первом варианте приводит к искажению диагностического графика, см. рис. 11, что может привести к неверному выбору интерпретационной модели фильтрации.

Таблица 2

Режимы работы скважины, использованные при генерации гипотетических кривых забойного давления для проверки влияния учета истории работы скважины на результаты определения параметров пласта

Номер режима	Дебит, $м^3$/сут	Время работы на каждом режиме, ч.	
		Вариант 1	Вариант 2
1	120	48	48
2	130		
3	150		
4	140		
5	120		
6	150		288
7	120		48

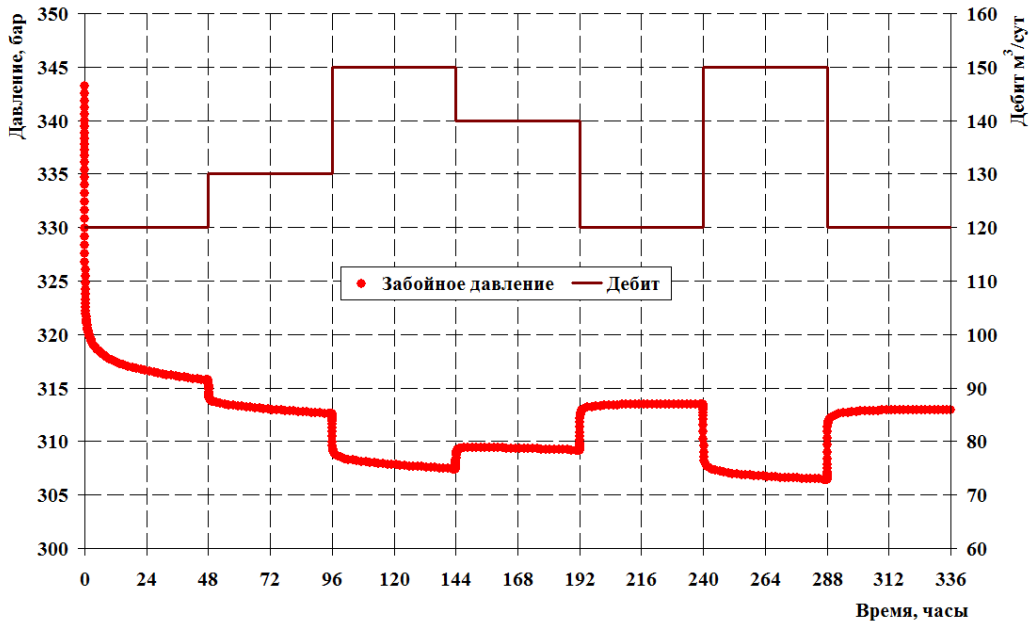


Рис. 9. Гипотетическая кривая забойного давления для изучения влияния учета истории работы скважины на результаты определения параметров пласта. Вариант 1

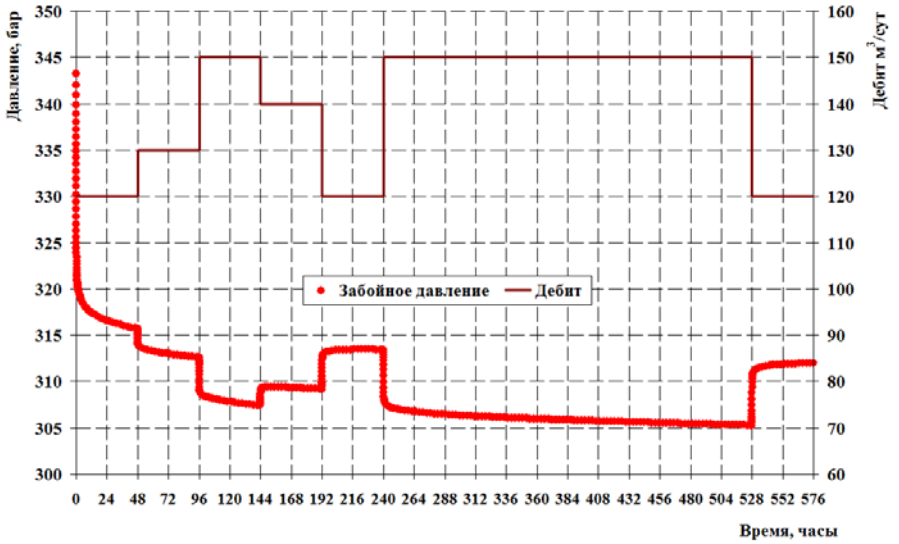


Рис. 10. Гипотетическая кривая забойного давления для изучения влияния учета истории работы скважины на результаты определения параметров пласта. Вариант 2

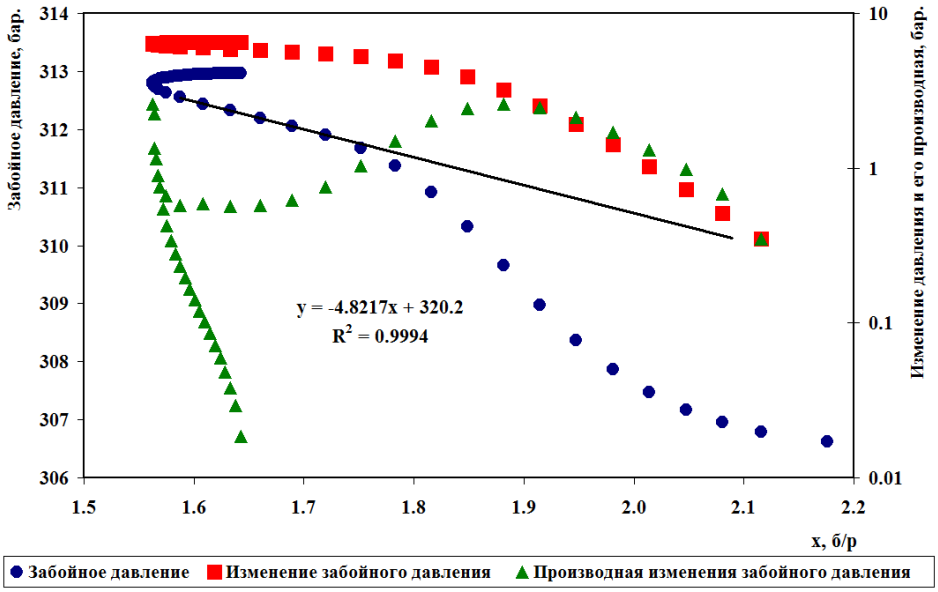


Рис. 11. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Изучение влияния учета истории работы скважины на результаты определения параметров пласта. Вариант 1

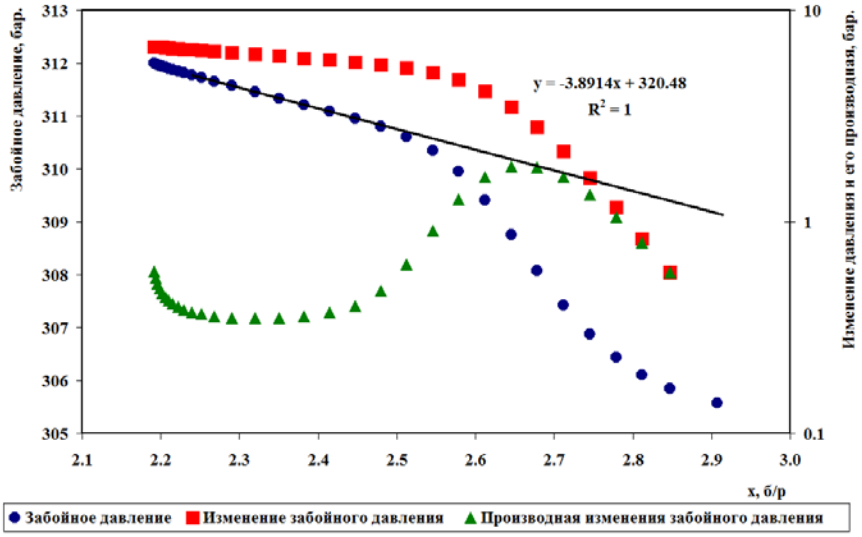


Рис. 12. Изменение забойного давления при исследовании скважины на двух режимах, совмещенное с диагностическим графиком. Изучение влияния учета истории работы скважины на результаты определения параметров пласта. Вариант 2

Таблица 3

Влияние длительности первого режима работы на определяемые методом исследования на двух режимах параметры пласта

	Проницаемость, мД	Скин-фактор, б/р	Пластовое давление, бар.
Фактические параметры*	100	+2	343.2
Вариант 1	73	-0.5	342.6
Вариант 2	91	+1.1	343.0

* - Параметры, которые задавались при решении прямой задачи (получении гипотетических кривых)

4. Примеры обработки материалов гидродинамических исследований методом двух режимов

Для оценки применимости данного подхода были рассмотрены примеры на основе фактических материалов комплексных гидродинамических исследований, проведенных в скважинах различных отечественных месторождений методами восстановления давления и установившихся отборов. Основными требованиями к исходным данным были следующие: «гладкие» кривые забойного давления при работе скважины на режимах, наличие участка радиальной фильтрации (желательно) на диагностическом графике, построенном по КВД.

Обработка кривых изменения давления на каждом из режимов осуществлялась по методике, описанной выше. Кроме того, были обработаны кривые восстановления давления и проведен анализ индикаторных диаграмм, что позволило сравнить фильтрационные параметры.

На рис. 13, 16, 19 представлены технологические схемы исследований в скв. 214 местрождения К Восточно-Сибирской НГП (нефтегазоносной провинции), скв. 1 месторождения Л Волго-Уральской НГП, скв. 42 месторождения ЗХ Тимано-Печерской НГП. На этих рисунках приведены кривые изменения забойного давления, дебита жидкости и диаметры штуцеров, через которые скважина работала на различных режимах. Овалами выделены обрабатываемые участки.

По кривым восстановления давления, зарегистрированным в этих скважинах, были построены диагностические графики Бурде, рис. 14, 17, 20. Участки радиальной фильтрации четко выделяются на кривых производных изменения давления, построенных по КВД, зарегистрированных в скв. 214 (рис. 14) и 42 (рис. 20). Это позволяет достоверно определить фильтрационные параметры пласта. При обработке КВД были использованы модель однородного бесконечного пласта (скв. 214), модель пласта с двойной проницаемостью и границей постоянного давления (скв.1, рис. 17) и модель однородного пласта с границей постоянного давления (скв. 42, рис. 20).

На рис. 3, 6, 9 и 10 представлено графическое отображение обработки кривых изменения давления методом 2-х режимов и результаты этих обработок в сравнении с результатами обработки КВД. На этих рисунках показаны диагностические графики, которые состоят из кривой изменения давления и кривой производной от изменения давления. Так же на рисунках представлены кривые забойного давления. Основное предназначение диагностического графика заключается в уточнении положения прямолинейного участка на обрабатываемой кривой забойного давления по горизонтальному участку производной. Более того, по виду диагностического графика можно получить информацию о модели пласта.

4.1 Месторождение К, Восточно-Сибирская НГП, скв. 214

На рис. 13 приведена технологическая схема комплексных гидродинамических исследований скв. 214.

При обработке КВД была использована модель однородного бесконечного пласта, рис. 14.

Для обработки кривой изменения забойного давления, зарегистрированного на режиме 7 мм, был построен график (круглые точки) забойного давления p_{wf} от x, на котором четко выделяется прямолинейный участок, рис. 15. По тангенсу угла наклона прямой и точке пересечения с осью ординат были оценены проницаемость (5.0 мД), скин-фактор (7.1 б/р) и пластовое давление (202.0 бар).

На рис. 15 приведен диагностический график, который состоит из кривой изменения давления (квадратные точки) и кривой производной изменения давления (треугольные точки) по аргументу x. Основное предназначение диагностического графика заключается в уточнении положения прямолинейного участка на обрабатываемой кривой забойного давления по горизонтальной составляющей производной.

Сравнение результатов обработки КВД и кривых режимов, приведенное на рис. 15 в таблице, показало, что проницаемость и пластовое давление, определенные согласно разным способам исследования, оказались близки. Относительная погрешность определения проницаемости составила 13%, а абсолютная погрешность пластового давления – 3.6 бар. Значительное отличие скин-фактора (7.1 и 0 б/р) обусловлено изменением количества свободного газа на разных режимах, что характерно на месторождении К, на котором начальное давление близко к давлению насыщения нефти газом.

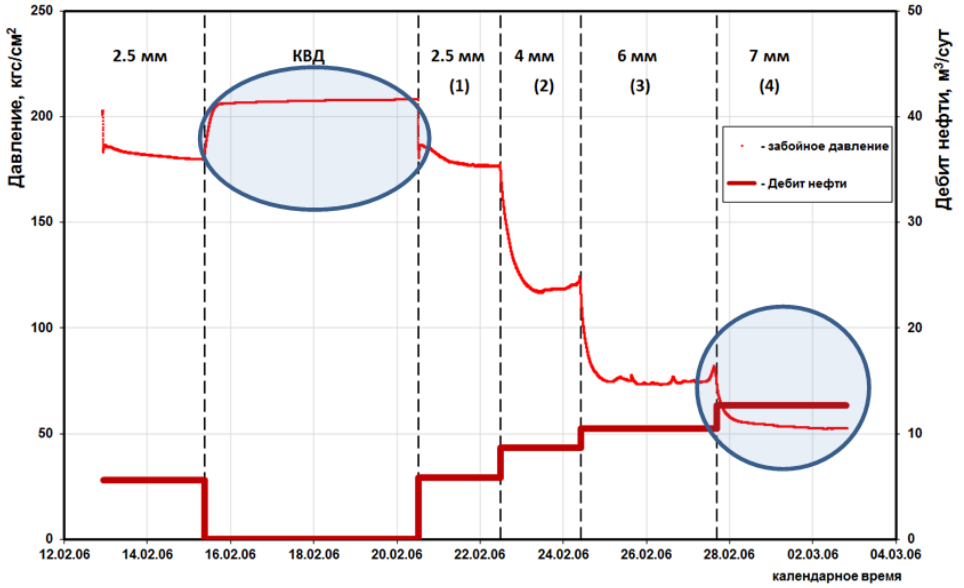


Рис. 13. Технологическая схема гидродинамических исследований. Месторождение К, скв. 214

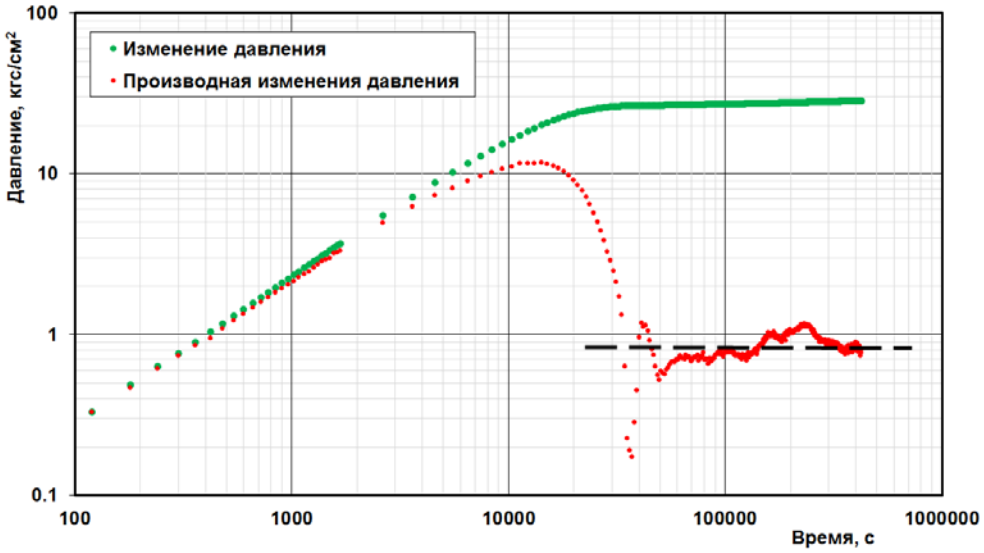


Рис. 14. Диагностический график Бурде, построенный по КВД. Месторождение К, скв. 214

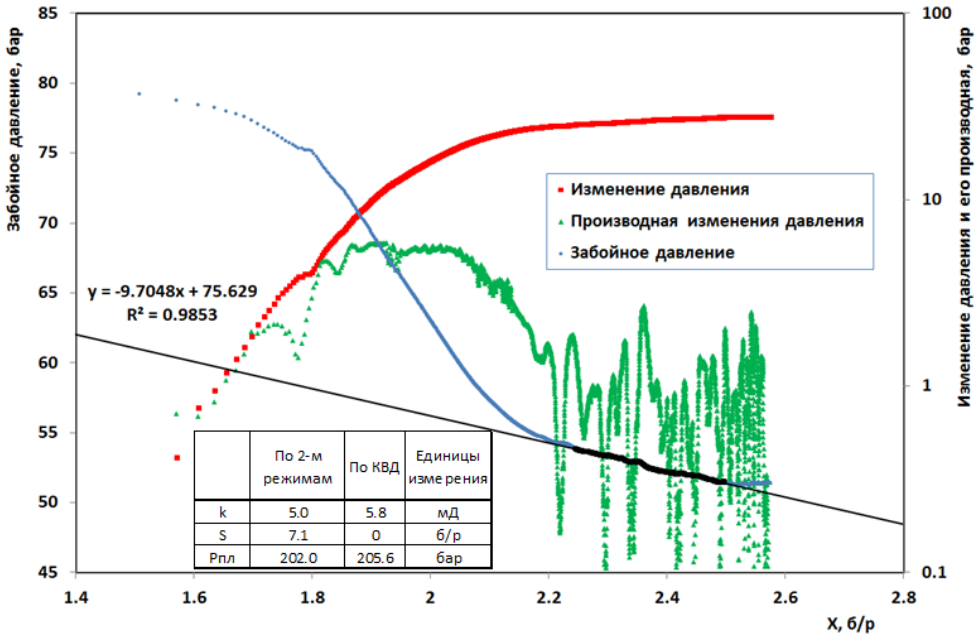


Рис. 15. Обработка кривой изменения давления на режиме 7 мм методом 2-х режимов. Месторождение К, скв. 214

4.2 Месторождение Л, Волго-Уральская НГП, скв. 1

На рис. 16 приведена технологическая схема комплексных гидродинамических исследований скв. 1.

При обработке КВД была использована модель пласта с двойной проницаемостью и границей постоянного давления, рис. 17.

Для обработки кривой изменения забойного давления, зарегистрированного на режиме 5 мм был построен график (круглые точки) забойного давления p_{wf} от x, рис. 18. На графике четко выделяется участок перегиба и 2 прямолинейных участка. Причем более поздний участок находится выше раннего, что характерно для случая пласта с границей постоянного давления. Проницаемость, скин-фактор и пластовое давление были определены по формулам, соответственно (4), (10), (8). Кроме того, произведена оценка расстояния до границы согласно формуле (11), которое составило 130м.

Сравнение результатов обработки КВД и кривых режимов, приведенное на рис. 18, показало, что проницаемость и пластовое давление согласно разным способам исследования оказались близки. Относительная погрешность определения проницаемости составила 16%, абсолютная погрешность пластового давления – 2.4 бар. Следует отметить, что при сравнении результатов исследований была принята общая проницаемость, полученная по КВД методом наилучшего совмещения по модели пласта с двойной проницаемостью.

На рис. 18 также приведен диагностический график, который состоит из кривой изменения давления (квадратные точки) и кривой производной изменения давления (треугольные точки) по аргументу x. В данном случае кривая производной оказалась малоинформативной.

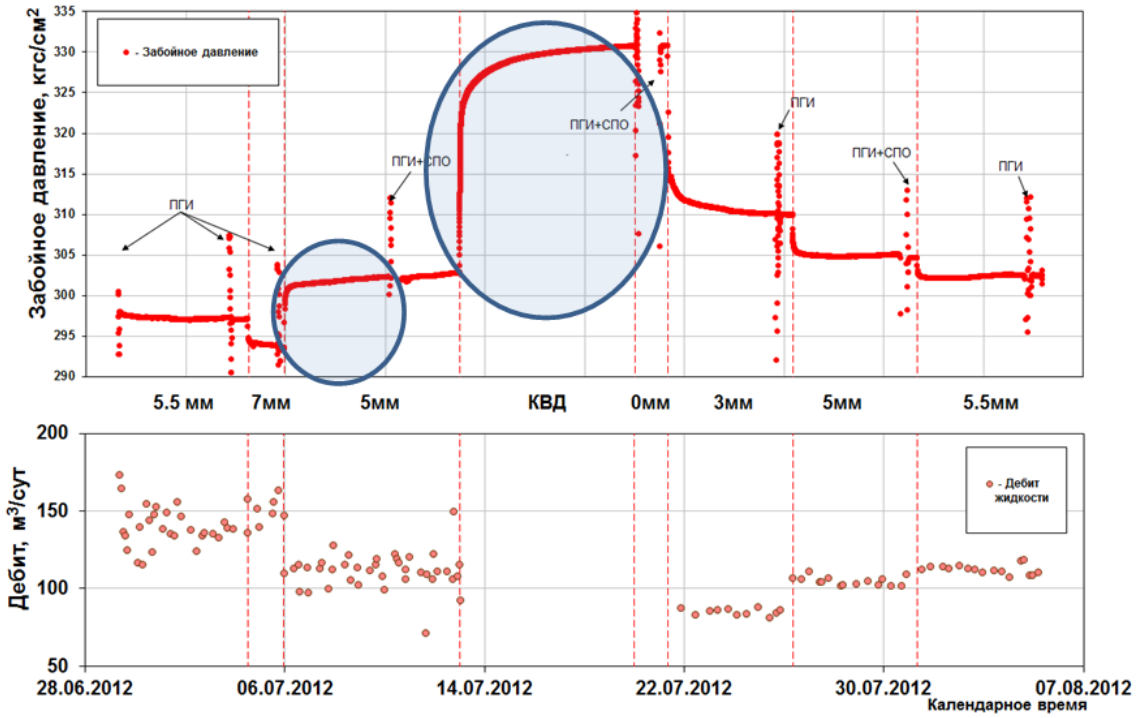


Рис. 16. Технологическая схема гидродинамических исследований (ПГИ - промыслово-геофизические исследования, СПО – спускоподъемные операции). Месторождение Л, скв. 1

4.3 Месторождение ЗХ, Тимано-Печерская НГП, скв. 42.

На рис. 19 приведена технологическая схема комплексных гидродинамических исследований.

При обработке КВД была использована модель однородного пласта с границей постоянного давления, рис. 20.

Для обработки кривой изменения забойного давления, зарегистрированного на режиме 6 и 8 мм были построены графики (круглые точки) забойного давления p_{wf} от x, рис. 21, 22. На этих рисунках также приведены диагностические графики, которые состоят из кривых изменения давления (квадратные точки) и производных изменения давления (треугольные точки) по аргументу x. На рис. 21-22 видно, что четко выделить прямолинейный участок на графиках забойного давления (круглые точки) затруднительно. Наличие горизонтальных участков на производных позволило уточнить положение прямолинейных участков, по которым и была произведена оценка проницаемости, скин-фактора и пластового давления (результаты приведены в таблицах на рис. 21-22).

Сравнение результатов обработки КВД и кривых режимов, приведенное на рис. 21-22 показало, что проницаемость, скин-фактор и пластовое давление согласно разным способам исследования оказались достаточно близки. Относительная погрешность определения проницаемости составила 3 и 8%, соответственно в первом случае и во втором случае, абсолютная погрешность пластового давления – 1.5 и 6.1 бар.

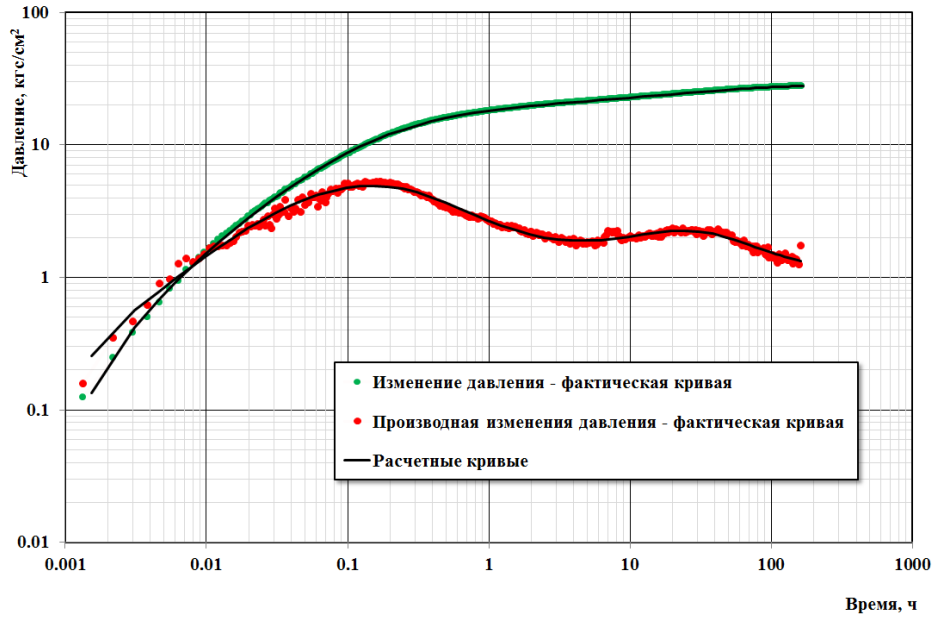


Рис. 17. Обработка КВД методом наилучшего совмещения на диагностическом графике Бурде. Месторождение Л, скв. 1

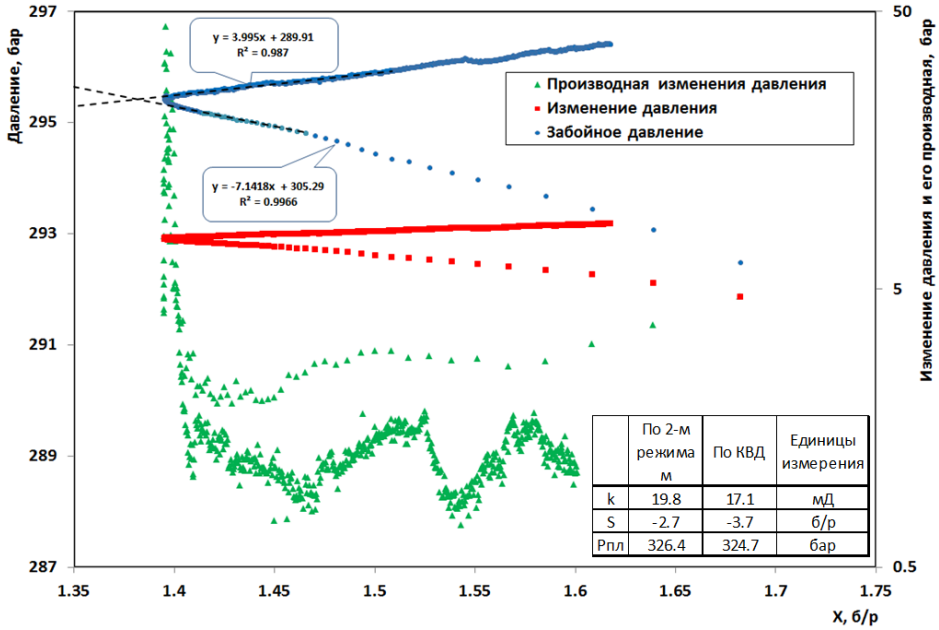


Рис. 18. Обработка кривой изменения давления на режиме 5 мм методом 2-х режимов. Месторождение Л, скв. 1

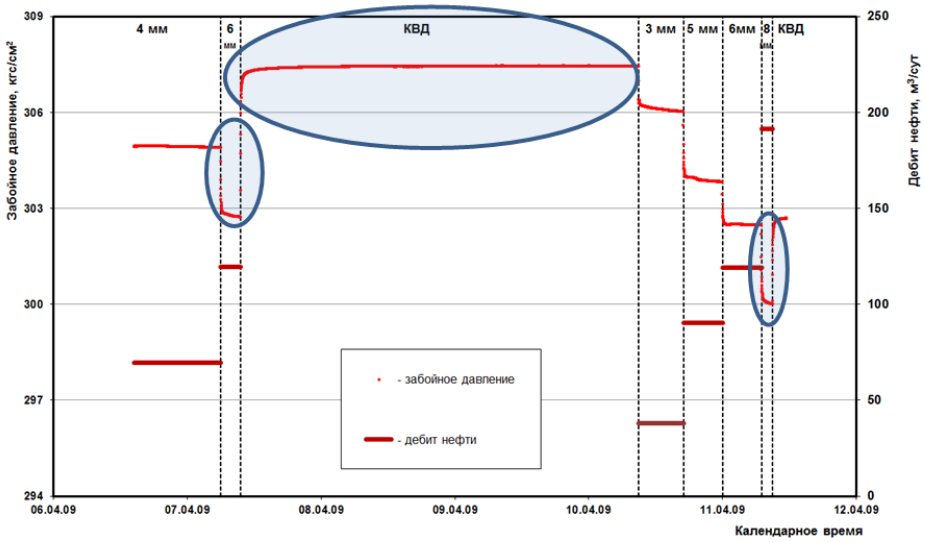


Рис. 19. Технологическая схема гидродинамических исследований. Месторождение ЗХ, скв. 42

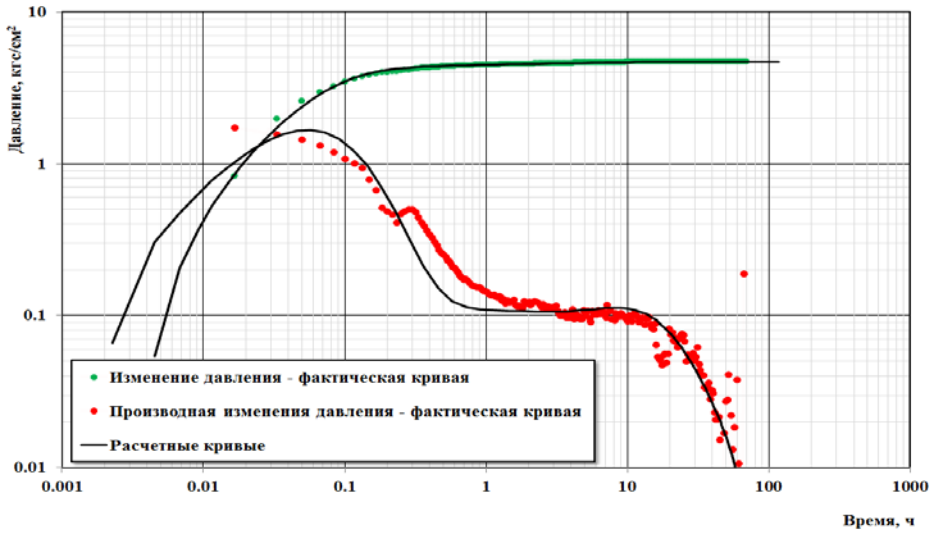


Рис. 20. Обработка КВД методом наилучшего совмещения на диагностическом графике Бурде. Месторождение ЗХ, скв. 42

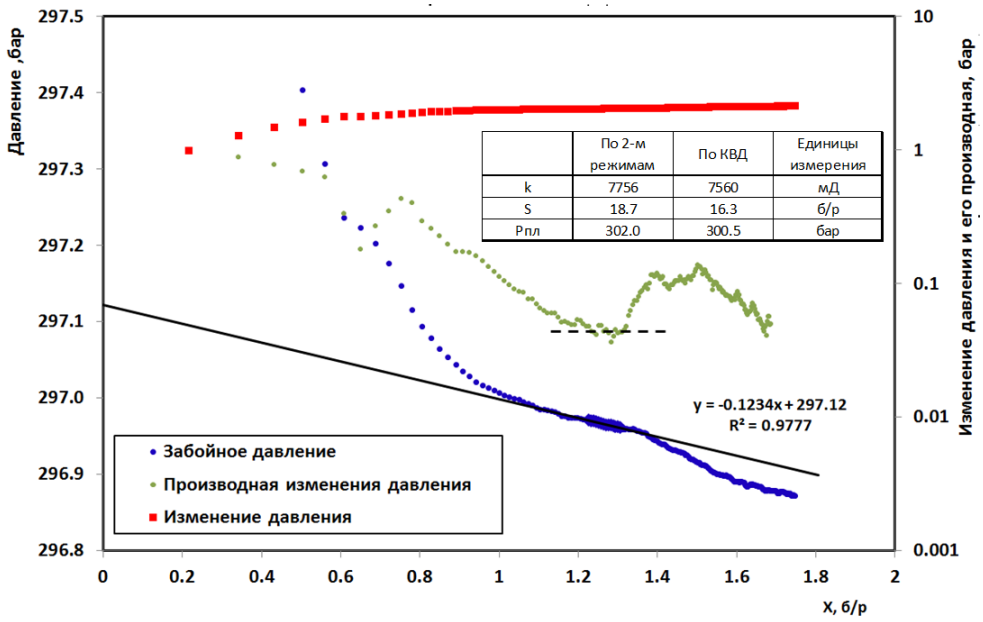


Рис. 21. Обработка кривой изменения давления на режиме 6 мм методом 2-х режимов. Месторождение ЗХ, скв. 42

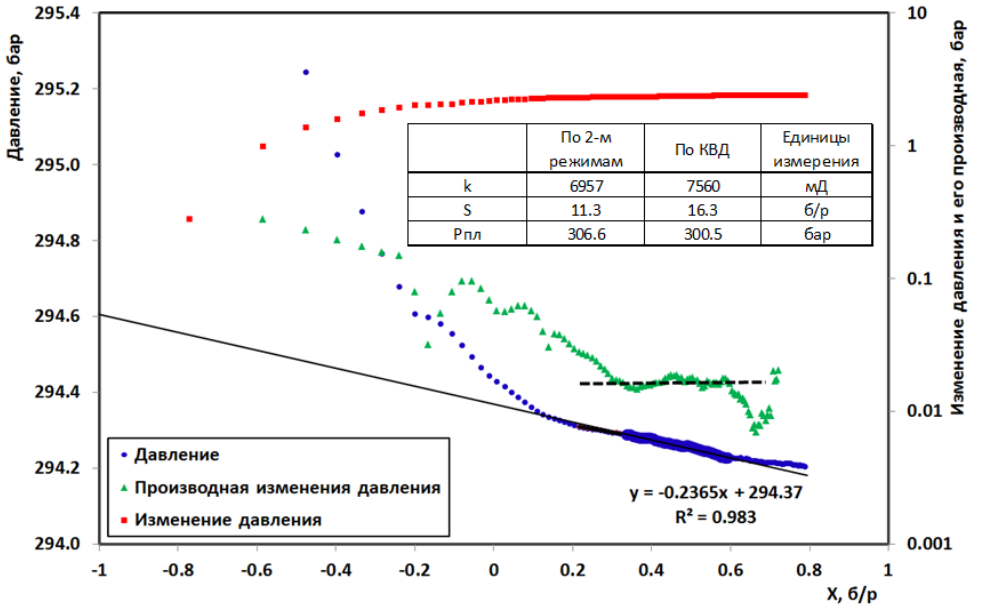


Рис.22. Обработка кривой изменения давления на режиме 8 мм методом 2-х режимов. Месторождение ЗХ, скв. 42

Заключение

1. Рассмотренный метод гидродинамических исследований скважина на двух режимах имеет удовлетворительную для практических целей точность. Он позволяет оценивать такие параметры, как пластовое давление, проницаемость, скин-фактор, расстояние до границ. Это показано для моделей однородного бесконечного пласта, однородного пласта с линейной непроницаемой границей и однородного пласта с линейной границей постоянного давления.
2. Вероятно, возможна оценка и других параметров при использовании более сложных моделей. Однако в случае сложных моделей течения, идентификация модели может быть усложнена в результате существенных колебаний давления и (в некоторых случаях) дебита при работе скважин на режимах.
3. Для идентификации модели течения может быть использован аппарат диагностических графиков Бурде в несколько модифицированном виде, как показано в работе.
4. Для уверенной оценки параметров пласта скважина должна исследоваться на двух режимах после восстановления давления до пластового, либо первый режим должен быть существенно длиннее второго.
5. Были проанализированы фактические материалы по 10-ти комплексным исследованиям, включающим КВД и исследования на режимах в фонтанирующих скважинах и скважинах, оборудованных электроцентробежным насосом. Основными требованиями к исходным данным были следующие: «гладкие» кривые забойного давления при работе скважины на режимах, наличие участка радиальной фильтрации (желательно) на диагностическом графике, построенном по КВД.
6. Обработка кривых изменения давления на каждом из режимов осуществлялась по методу 2-х режимов. Кроме того, были обработаны кривые восстановления давления и проведен анализ индикаторных диаграмм, что позволило сравнить полученные с помощью разных методов фильтрационные параметры. КВД обрабатывались методом наилучшего совмещения или обобщенным дифференциальным методом.
7. В настоящей статье приведена оценка фильтрационных параметров и уточнение информации о модели пласта по 3-м наиболее информативным исследованиям с помощью рассматриваемого подхода.
8. При обработке кривых изменения давления методом 2-х режимов были построены диагностические графики. В ряде случаев горизонтальная составляющая производной позволила уточнять положение прямолинейного участка на обрабатываемой кривой забойного давления, по которой и была произведена оценка проницаемости, скин-фактора и пластового давления. В связи с этим подтверждается необходимость построения такого графика по аналогии с диагностическим графиком Бурде, используемого при интерпретации КВД.
9. Сравнение обработок материалов разных исследований показало, что при обработке методом 2-х режимов наиболее точно определена проницаемость и пластовое давление, точность определения скин-фактора оказалась удовлетворительной.
10. Результаты обработки по данной методике оказались неудовлетворительными в следующих случаях: зашумленные кривые (особенно в случаях большого количества свободного газа); наличие скачков давления по причине техногенного характера; знакопеременная динамика давления, связанная, возможно, с влиянием соседних скважин; высокая погрешность определения дебита жидкости (выявлена большая чувствительность погрешности расчетного пластового давления от точности замера дебита).
11. Полученные результаты говорят о высокой эффективности и достаточной для практических задач точности рассматриваемого подхода. При наличии исследования методом восстановления давления данный подход является дополнительным источником информации. При отсутствии такого исследования он позволяет оценивать фильтрационно-емкостные свойства и пластовое давление. Наиболее полную и точную картину строения пласта и его свойств можно получить только при комплексировании различных видов исследований.

Reservoir properties and reservoir energy parameters evaluation by two-rate testing

I.V. Afanaskin, P.V. Kryganov, S.G. Volpin, U.M. Shteynberg, I.A. Volpin

Abstract: Two-rate testing aspects observed for different reservoir models. First stage rate duration influence on reservoir parameters investigated by interpretation of theoretical curves. Real data examples of two-rate testing provided. Pressure build-up and two-rate testing results comparison provided.

Keywords: Well test, two-rate testing.

Литература

1. Dominique Bourdet et al. A new set of type curves simplifies well test analysis // World Oil, 1983, May. - pp. 95–106.
2. Dominique Bourdet. Well Test Analysis: The Use of Advanced Interpretation Models. Amsterdam: Elsevier Science B.V., 2002. - P. 425.
3. Dominique Bourdet. Интерпретация результатов исследований скважин // Материалы лекций. Petroleum Engineering and Related Management Traning Gubkin Academy – Moscow, 1994. – 109 с.
4. Роберт Эрлогер мл. Гидродинамические методы исследования скважин / Под ред. докт. техн. наук, проф. М.М. Хасанова. М.-Ижевск: Институт компьютерных исследований, 2007. – 512 с.
5. D.G. Russell. Determination of Formation Characteristics From Two rate Flow Test // J. Pet. Tech. Dec 1963. – pp. 1347-1355.
6. С.Г.Вольпин, Ю.А.Мясников, Н.П.Ефимова. и др. TESTAR – пакет программ для обработки материалов гидродинамических исследований нефтегазовых пластов // Нефтяное хозяйство. - №5, 2002. – с. 58-60
7. Аманат Чодри. Гидродинамические исследования нефтяных скважин / Под ред. канд. техн. наук С.Г.Вольпина: ООО «Премиум Инжиниринг», 2011. – 687 с.
8. С.Н. Бузинов, И.Д. Умрихин. Исследование нефтяных и газовых скважин и пластов. М.: Недра, 1984. – 269 с.
9. Л.Г. Кульпин, Ю.А. Мясников. Гидродинамические методы исследования нефтегазоводоносных пластов. М.: Недра, 1974. – 200 с.

Влияние периода и коэффициента металлизации в многополосковом волноводе на возбуждение плазмонного резонанса

А.Н.Палагушкин, С.А.Прокопенко, А.П.Сергеев

Аннотация: В многочисленных публикациях рассмотрены вопросы связанные с распространением поверхностных электромагнитных волн оптического диапазона в плазмонных волноводах. При этом, основное внимание уделяется волноводам с характерными соизмеримыми с длиной оптической волны и субволновыми размерами, а также двумерным и более сложным плазмонным наноструктурам. В то же время границы перехода от сплошного двумерного слоя к одномерному, разделенному на систему полос с зазорами, ограничивающими колебания электронного газа в слое, практически не изучены. В работе приводятся результаты расчетов, выполненных по программе RODIS для системы многополосковых волноводов, образованных на поверхности плазмонного металлического покрытия. Варьируется ширина полосок и зазоров между ними, при характерных размерах значительно больших длины оптической волны, но уже отличающихся от сплошного слоя по форме поверхностного плазмонного резонанса. Результаты выполненных расчетов проверены экспериментально.

Ключевые слова: металлизация, волновод, плазменный резонанс.

Научные и технические приложения плазмоники активно используются для исследования ближнего поля в планарных оптических наноструктурах, лежат в основе нанофотоники [1-8]. Разработка новых современных систем связи направлена на увеличение скорости передачи больших объемов данных и преодоление существующих физических ограничений в используемых для этого электронных и оптических технологиях. Поверхностные электромагнитные волны (ПЭВ) оптического диапазона, связанные с электронной плазмой металла (поверхностные плазмоны поляритоны-ППП) и распространяющиеся вдоль границы металла с диэлектриком, позволяют преодолеть дифракционный предел и обеспечить расширение полосы пропускания оптических схем, согласовать компактность электронных чипов с огромной пропускной способностью оптических схем и объединить эти схемы в нанометровом масштабе размеров.

В этой связи, значительный интерес представляет переход от сплошного слоя металла к периодической системе полосковых волноводов. Оценки длины пробега ПЭВ для сплошного слоя Ag оптимальной толщиной 50 нм на длине волны ($\lambda_{вак}$ = 633, $\lambda_{ПЭВ}$ = 390 нм) составляют порядка 4.5 мкм, скин слой 29 нм. Плазмонная волна связана с двумерным электронным газом (2D) в тонком слое Ag. При разделении сплошного слоя на систему периодических узких полосок с зазорами, с определенного их размера наступает ограничение волны в поперечном направлении (1D) и она приобретает направленность распространения. Такой набор полосок металла можно рассматривать как систему связанных волноводов. Причем, модовый состав волновода определяется шириной полоски, а электромагнитная связь между ними - шириной зазора. Максимальная ширина полоски была выбрана примерно на порядок больше длины распространения ПЭВ, минимальная - близка к ней, а зазор задавался коэффициентом металлизации K (долей периода, приходящейся на полосу металлизации).

Структура освещалась лучом лазера (p-поляризация) вдоль полос металлизации со стороны подложки по схеме Кречмана. С помощью некоммерческого программного пакета RODIS выполнено моделирование зависимости углового спектра отражения такой структуры от периода и коэффициента металлизации. Расчеты показали значительное влияние ширины металлизированных полосок и зазоров между ними на угловой спектр отражения и поверхностный плазмонный резонанс.

Исходными данными для расчетов служили:

длина волны лазера 633nm, поляризация TM, излучение направлено вдоль полосков,

параллельный луч падал на границу с металлом из стекла с n = 1.514,

оптические параметры Ag использованы по данным «SOPRA» [9], n = 0.135, k = 3.978

толщина слоя Ag 50 нм, полоски полностью разделены зазором,

рассчитывался коэффициент отражения для нулевого порядка дифракции в дальней зоне в зависимости от угла падения луча Θ,

расчеты проводились для сплошного слоя (используемого в качестве начального отсчета и эталона для контроля правильности расчета) и бесконечной системы полосок, что определялось возможностями программы,

геометрия расчета показана на Рис.1.

Расчеты выполнены для сплошного слоя, а также для периодов 5, 10, 25, 50 мкм при коэффициентах металлизации K 0.50, 0.80, 0.85, 0.90, 0.95 (Рис.2). Минимум отражения соответствует углу SPR.

Заметные на графиках (Рис.2) биения в области углов, меньших угла полного отражения (т.е.

соответствующих пропусканию), являются артефактом программы RODIS и могут не приниматься во внимание. Расчеты с помощью независимо разработанной нами программы SPR16, предназначенной для расчетов сплошных многослойных плазмонных покрытий и тестированной на экспериментальных образцах, показали совпадение результатов расчетов с результатами RODIS для сплошных покрытий (за исключением биений).

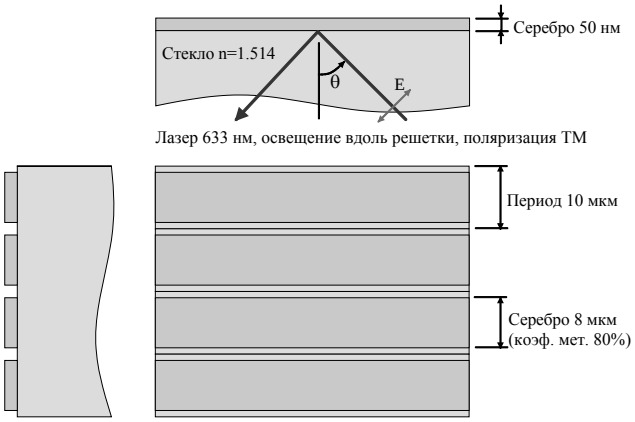
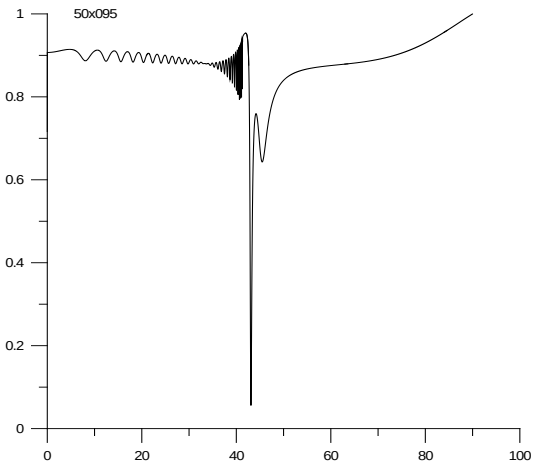
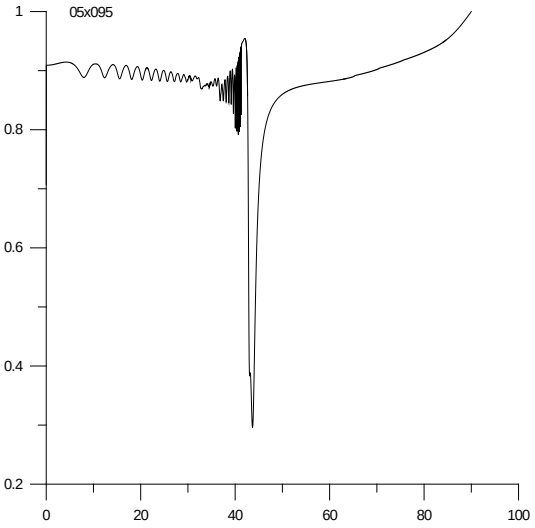


Рис.1. Условия проведения расчетов.
Показана структура с периодом 10 мкм и коэффициентом металлизации 0.8.

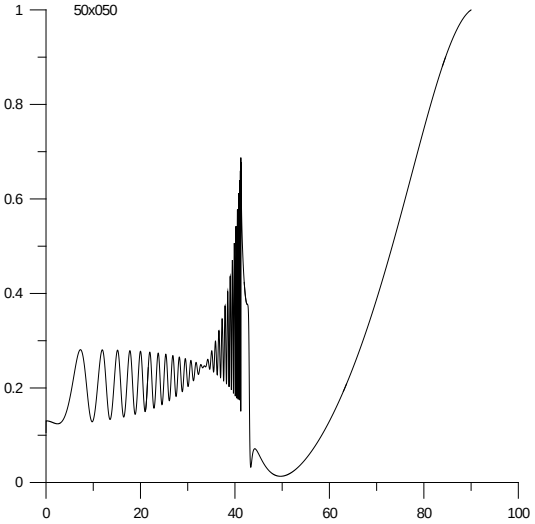
Анализ полученных результатов расчетов показывает, что при разделении сплошного слоя на отдельные, периодические полоски, при уменьшении их ширины и увеличении разделительного зазора форма зависимости отражения существенно изменяются, хотя угловое положение SPR резонанса остается неизменным. Появляется дополнительный минимум отражения, положение и форма которого сильно зависит от относительной ширины зазора между полосками, и в меньшей степени от ширины самих полосок. Поскольку он связан с геометрией решетки, то его условно можно назвать «резонансом» решетки. При К = 0.95 SPR и «резонанс» решетки близки, но с уменьшением ширины полоски от 50 до 5 мкм сливаются в один достаточно узкий (но шире, чем SPR) минимум отражения. Для К = 0.5 (меандровая решетка) - «резонанс» решетки более широкий и глубокий, чем SPR, и с уменьшением ширины полосок SPR–резонанс также с ним сливается (Рис.2).



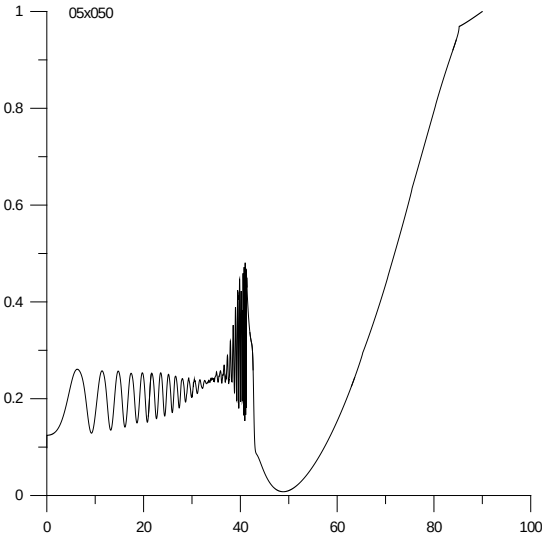
(а) К = 0.95 (период 50 мкм, зазор 2.5 мкм)



(б) К = 0.95 (период 5 мкм, зазор 0.25 мкм)



(в) К = 0.5 (период 50 мкм, зазор 25 мкм)



(г) К = 0.5 (период 5 мкм, зазор 2.5 мкм)

Рис.2. Расчетные угловые спектры отражения для периодов 5 (б,г) и 50 (а,в) мкм, К 0.95 и 0. 5.

Выполненные расчеты показывают, что геометрия (1D) периодической полосковой системы сильно влияет на форму угловой зависимости отражения при SPR. Причем, это влияние проявляется при достаточно большой ширине полосков и зазоров (значительно большей длины плазмонной поверхностной волны).

Для экспериментальной проверки результатов расчетов был разработан специальный измерительный стенд и изготовлен ряд полосковых структур с периодами 5 и 50 мкм и коэффициентами металлизации 0.5 и 0.95, выполненных на слоях серебра толщиной 50 нм, нанесенных на поверхности подложек из плавленого кварца и ниобата лития. Были проведены экспериментальные измерения изготовленных макетов на автоматизированной гониометрической установке по схеме Кречмана, которые показали хорошее совпадение с теоретическими расчетами.

В дальнейшем, приложив к полоскам внешнее напряжение и нанеся на поверхность такой структуры слой электрооптического материала можно попытаться управлять величиной коэффициента отражения. Это направление является предметом дальнейших исследований.

Работа выполнена по программе фундаментальных исследований ОНИТ РАН №1.8.

Influence of the period and the coefficient of metallization in multistrip waveguide on plasmon resonance excitation

A.N.Palagushkin, S.A.Prokopenko, A.P.Sergeev

Abstract: There are many publications discussing optical surface waves propagation in plasmon waveguides. The main interest is focusing on waveguides with characteristics comparable to the optical wavelength and sub-wavelength dimensions, as well as two-dimensional and more complex plasmonic nanostructures. At the same time, the boundaries between solid and one-dimensional and two-dimensional layers, divided by the system of bands with gaps limiting the oscillations of the electron gas in the layer, actually have not been studied. The paper presents the results of calculations performed by the program RODIS of multilayer waveguides formed on the surface of plasmon metal coating. The width of the strips and the gaps between them were varying, with characteristic dimensions considerably larger then optical wave length, but already having different form of the surface plasmon resonance curve as compared to continuous layer. The results of the calculations are experimentally verified.

Keywors: plating, waveguide, plasmon resonance.

Литература

1. «Surface Plasmon Resonance, Methods and Protocols», Edited by Nico J. de Mol, Marcel J. E. Fischer, Humana Press is part of Springer Science+Business Media, 286 p., 2010.
2. «Handbook of Surface Plasmon Resonance», Edited by R.B.M. Schasfoort and Anna J. Tudos, The Royal Society of Chemistry, 2008.
3. Dieter W. Pohl. «Near-Field Optics and the Surface Plasmon Polariton», Springer Berlin , Heidelberg, ISSN, 1437-0859, Том 81, 2001.
4. Pieter G. Kik and Mark L.Brongersma. «Surface Plasmon Nanophotonics», 2007 Springer,
5. Stefan A. Maier. «Plasmonics: Fundamentals and Applications», Springer Science+Business Media LLC, p.177, 2007.
6. «Plasmonic Nanoguides and Circuits», Editor Sergey Boszevolnyi, Pan Stanford Publishing Pte. Ltd., ISBN-13 978-981-4241-32-8, 441p., 2009.
7. Igor Tsukerman. "Computational Methods for Nanoscale: Applications, Particles, Plasmons and Wave" , Sprindger, ISBN: 978-0-387-74777-4, 520p., (2005).
8 В.В. Климов. "Наноплазмоника", М., Физматлит, 480с., 2009 г.
9. SOPRA , http://www.sspectra.com/files/misc/win/SOPRA.EXE
10. American Institute of Physics Handbook, McGraw-Hill, NY (1957) pp. 6–104
11. A.N.Palagushkin, S.A.Prokopenko, A.P.Sergeev. “Plasmonic Holographic Nanostructures”, Optical Memory and Neural Networks (Information Optics), v.16, 288-294 (2007); ibid v.18, 156–163 (2009).

Газовый сенсор SPR с коронным разрядом

А.Н.Палагушкин, С.А. Прокопенко, А.П. Сергеев

Аннотация: В настоящее время сенсоры на основе поверхностного плазмонного резонанса (SPR) широко применяются в экспресс анализе жидких биологических сред в микробиологии, медицине и генной инженерии. Они также используются для определения примесей в газовых средах. При этом основная проблема их разработки заключается в подборе селективных сорбирующих сред наносимых на поверхность сенсора для определения конкретной примеси. Свойства сорбента в сочетании с высокой чувствительностью SPR определяют достижимые параметры сенсора. При нанометровой толщине сорбентов основной вклад в их свойства вносит состояние поверхности. Воздействие плазмы положительного коронного разряда приводит к эффективной десорбции посторонних примесей блокирующих активные центры адсорбции. Кроме того, поверхность сенсора облучается радикалами, возникающими в плазме разряда при ионизации примесей. Возможность использования эффектов коронного разряда для создания SPR сенсоров, определяющих микропримеси в газовых средах, рассматриваются в данной статье.

Ключевые слова: газовый сенсор, коронный разряд, резонанс

Введение

В работе приведены результаты предварительных исследований по разработке и оценке параметров сенсоров, выполненных на основе поверхностного плазмонного резонанса (SPR) с применением коронного разряда [1,2]. Они предназначены для анализа присутствия микропримесей в газовой среде.

Принцип работы SPR сенсоров основан на измерении в реальном времени изменения оптических параметров среды в очень малом объеме вблизи и на чувствительной поверхности сенсора [3], в области распространения поверхностной оптической волны [4-6]. Такие сенсоры универсальны, имеют высокую чувствительность, но малую селективность. Поэтому для определения конкретных микропримесей на их поверхность наносятся нанослои селективных сорбентов и измеряются изменения их оптических характеристик и толщины [7,8]. Это проявляется в смещении положения минимума угловой зависимости нарушенного полного отражения, соответствующего SPR. Отслеживая положение минимума в реальном времени можно определить кинетику сорбции и десорбции микропримеси на сорбенте и исходное состояние его поверхности. Последнее очень важно, т.к. активные центры сорбента блокируются молекулами посторонних примесей и воды. Для их активации мы использовали облучение положительными ионами коронного разряда.

1. Эксперимент

Проводился на специально разработанном стенде (Рис.1) для измерения (in situ) динамических временных смещений минимума SPR-резонанса при воздействии положительного коронного разряда в кварцевой ячейке продуваемой осушенным воздухом, в который вводились разбавленные и дозированные примеси паров органических соединений.

Для возбуждения коронного разряда применялась кварцевая герметичная ячейка объемом ~ 1 $см^3$, установленная на поверхность сенсора (Рис.2), через которую продувался осушенный воздух с постоянным расходом ~ 1 $см^3/с$. Расход воздуха измерялся электронным датчиком AWM3200V.

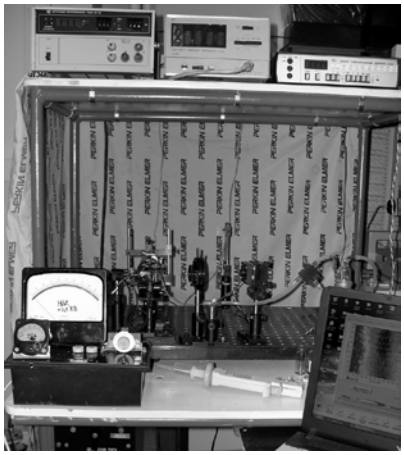


Рис.1 Измерительный стенд.

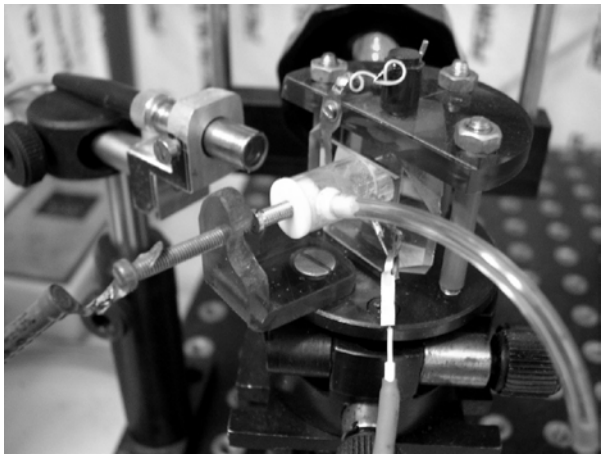


Рис.2 Измерительная ячейка с сенсором SPR.

Воздух подавался микро компрессором и осушался цеолитом. Перед измерительной ячейкой установлено устройство для ввода в поток воздуха проб примесей из иглового микрошприца или дозатора.

Положительный коронный разряд зажигался от стабилизированного источника высокого напряжения между вольфрамовой иглой радиуса ~20 мкм и сеткой, расположенной у поверхности сенсора. Сетка применялась для ослабления потока частиц плазмы на поверхность сенсора. Использовался положительный коронный разряд, который, по сравнению с отрицательным. не приводит к необратимым изменениям поверхности сенсора. Ток разряда составлял 2…10 мкА при напряжении 3…5 кВ. Для исключения пробоев ток разряда ограничивался резистором 22 МОм.

Схема работы измерительного стенда показана на Рис.3.

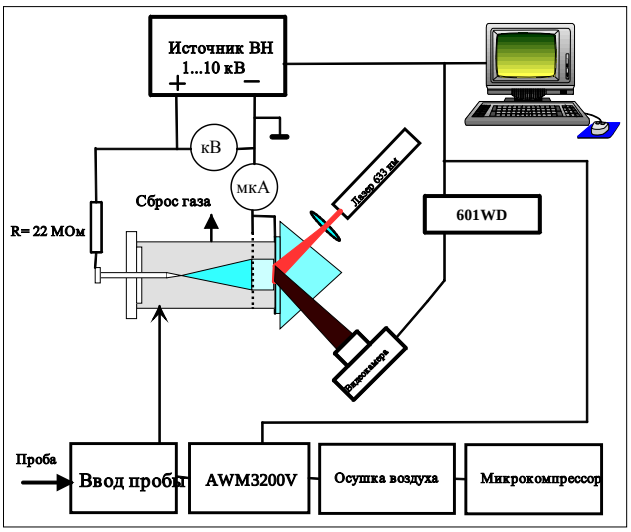


Рис.3 Схема стенда.

На этом экспериментальном стенде, для измерения угла SPR в реальном времени (~ 10 измерений в секунду), применялась схема Кречмана с п/п лазером KLM-B635-1-5-FA и КМОП видеокамерой SK-1004C, USB конвертором видеосигнала 601WD и компьютером, записывающим изменение углового положения минимума SPR резонанса во времени.

Посредством этого стенда изучались кинетические зависимости изменения свойств чувствительной поверхности SPR-сенсора при сорбции и десорбции ионизированных в коронном разряде газов и продуктов разложения органических примесей.

2. Полученные результаты.

Первоначально изучалось воздействие плазмы коронного разряда в сухом воздухе на поверхность сенсора. Для этого сенсор длительно выдерживался на атмосферном воздухе (в присутствии паров воды при естественной относительной влажности ~ 50%), а затем, без включения разряда, продувался осушенный воздух. Зависимость десорбции влаги с поверхности сенсора записывалась во времени (Рис.4).

На графике Рис.4 шкала оси Y – относительная, ее масштаб постоянный для всех последующих графиков. Смещение вверх показывает относительное увеличение угла SPR, вниз – уменьшение. Увеличение угла соответствует осаждению какого либо вещества на поверхности сенсора, уменьшение – его удалению и очистке поверхности.

Начальная точка отсчета не соответствует абсолютному значению угла SPR и выбиралась при настройке аппаратуры перед измерением.

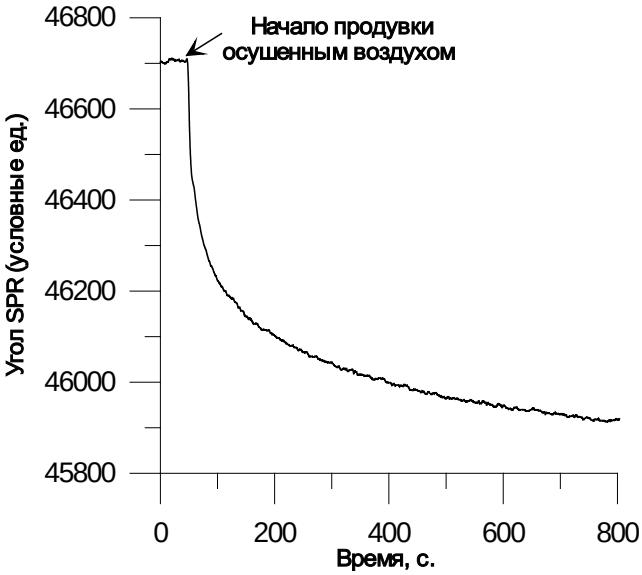


Рис.4. Продувка осушенным воздухом.

Изменение угла SPR во времени близко к экспоненциальному, что характерно для процесса десорбции влаги с поверхности сенсора. В данном эксперименте применялся сенсор на основе нанослоя Ag с защитным слоем Al_2O_3 толщинами 65 нм и 7 нм.
В дальнейшем, продувка сухим воздухом не прекращалась, и включался коронный разряд при напряжении 3 кВ и токе 2 мкА. Изменение угла SPR показано на Рис.5.

Рис.5. Действие продувки сухим воздухом и коронного разряда на изменение угла SPR.

При включении разряда всегда наблюдался выброс, вероятно вызванный десорбцией влаги с поверхности кварцевой ячейки и попадающей на поверхность сенсора. Затем следовало значительное уменьшение угла SPR, связанное с десорбцией остаточных следов воды с поверхности сенсора. При этом наблюдалось явно выраженное (Рис.5) или не столь явное разделение на два различных временных процесса, зависящих от толщины защитного слоя Al_2O_3. Скорость десорбции увеличивалась при уменьшении его толщины, а величина общего изменения угла SPR несколько уменьшалась, что, возможно, вызвано меньшей сорбционной емкостью более тонкого слоя.

Положение начального угла до продувки сухим воздухом зависело от влажности окружающей среды и времени установления сорбционного равновесия с поверхностью сенсора. Положение конечного угла от внешних условий практически не зависело.

При увеличении напряжения до 5 кВ и тока коронного разряда до 10 мкА выброс при его включении возрастал, а скорость спада увеличивалась. При больших напряжениях и токах, после спада положение конечного угла SPR несколько увеличивалось (Рис.6), что может быть связано с осаждением на поверхности сенсора заряженных радикалов, присутствующих в плазме коронного разряда.

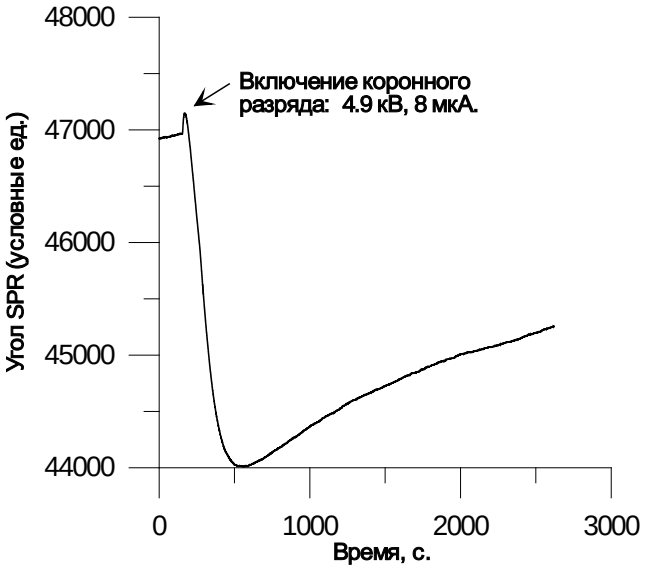


Рис.6. Изменение угла SPR при больших токах КР.

После выключения коронного разряда (протекающего при токах 2…5 мкА), при включенной продувке, и достижения минимального угла SPR, происходило длительное и линейное во времени увеличение угла до исходного значения. Повторное включение разряда приводило к достижению того же значения минимального угла.

Эксперименты, проведенные на сенсорах с покрытием из слоя золота толщиной 53 нм показали аналогичные результаты. Изменения угла SPR при продувке сухим воздухом были несколько меньшими, а воздействие коронного разряда примерно таким же.

В дальнейшем, проведены эксперименты с вводом в поток сухого воздуха примеси паров гексана. Использовался гексан ОСЧ. Для дозирования паров примеси гексана в осушенном воздухе использовалось последовательное разбавление. Доза жидкого вещества объемом 0.1 мкл микрошприцом Hamilton 7101N (1 мкл) вводилась и испарялась в объеме 25 мл сухого воздуха в модернизированном пипеточном дозаторе для жидкостей ДПОПц-1-10-5000. Затем она последовательно 4-х кратно разбавлялась сухим воздухом до нужной концентрации. Это приближенный метод, поскольку вероятна не учитываемая сорбция примеси на стенках дозатора при сильном разбавлении смеси. Пробная смесь объемом от 0.5 мл до 2.5 мл вводилась в поток сухого воздуха поступающего в измерительную ячейку.

Расчет концентраций вводимой примеси в зависимости от разбавления приведен в Таблице 1.

Таблица 1. Концентрация примеси гексана в воздухе.

Разбавление 1 : 4 – раз:	Концентрация		
	в %	в ppm (об.)	в мг/м^3
1	$1.71*10^{-2}$	171	613
2	$4.29*10^{-3}$	42.9	154
3	$1.07*10^{-3}$	10.7	38.3
4	$2.68*10^{-4}$	2.68	9.61
5	$6.69*10^{-5}$	0.67	2.40
6	$1.67*10^{-5}$	0.17	0.61
7	$4.18*10^{-6}$	0.042	0.15

На Рис.7 показана реакция сенсора на последовательный ввод проб паров гексана в объемах от 0.5 до 2.5 мл в сухом воздухе с концентрациями ~0.61 и ~9.6 мг/м^3, а также чистого сухого воздуха. Ввод пробы приводит к изменению давления и расхода основного постоянного потока воздуха через ячейку сенсора. Это влияние можно оценить, вводя в тех же условиях, чистый сухой воздух.

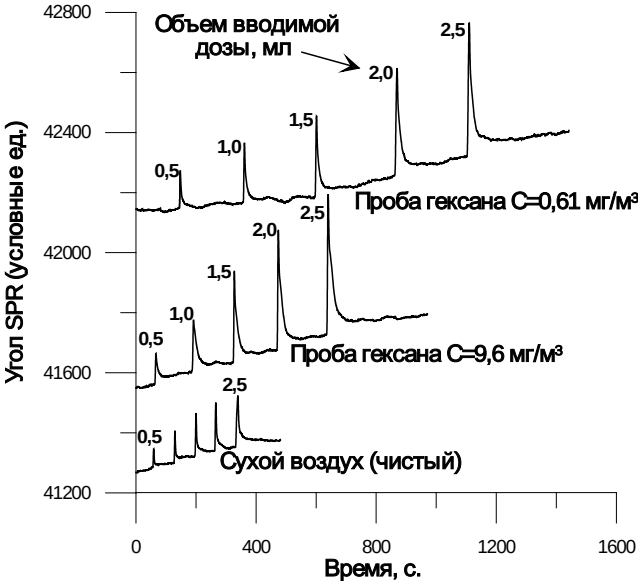


Рис.7. Реакция сенсора на пробы паров гексана в сухом воздухе.

Применялся SPR сенсор на основе нанослоя Ag с защитным слоем Al_2O_3 толщинами 65 нм и 7 нм. Расход воздуха составлял 1 $см^3/с$, коронный разряд был включен постоянно при напряжении 4 кВ и токе 4 мкА.

На графиках наблюдается постоянный подъем «нулевой» линии, что вероятно связано с накоплением на поверхности сенсора заряженных радикалов, образующихся в плазме коронного разряда.

Обнаружено, что отключение разряда, продувка сухим воздухом в течении определенного времени, и последующее включение разряда приводит к очистке поверхности сенсора, аналогично приведенному выше.

Такая процедура была проведена между вводами проб гексана с концентрациями 9.6 $мг/м^3$ и 0.61 $мг/м^3$. При этом реакция сенсора на концентрации примеси различающиеся на ~ 1.5 порядка была примерно одинакова. Это может быть связано как с повышением чувствительности сенсора, так и с погрешностями сильного разбавления пробной смеси.

Далее эксперименты были проведены на образце с SPR покрытием Ag (67 нм) и защитным слоем Al_2O_3 (~ 1 нм). Чувствительность этого образца была значительно ниже (Рис.8-10). Также наблюдались меньшие изменения угла SPR при продувке и очистке поверхности коронным разрядом в сухом воздухе. При больших концентрациях пробной смеси наблюдалась меньшее смещение «нулевой» линии и более быстрая реакция сенсора на ввод пробы. При меньших концентрациях возрастали шумы (Рис.10).

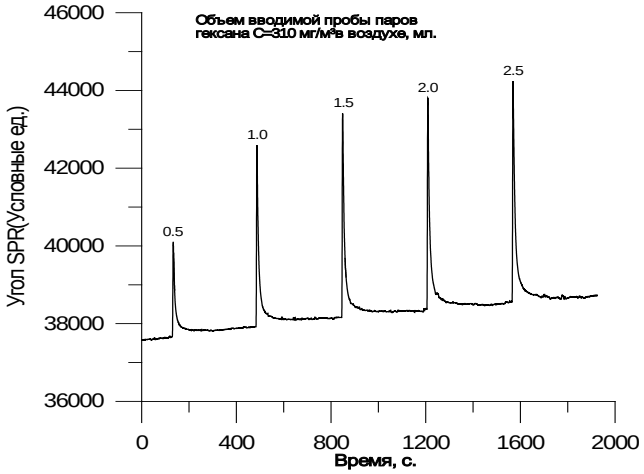


Рис.8. Реакция сенсора на пробы 0.5…2.5 мл гексана с концентрацией С=310 $мг/м^3$ в сухом воздухе.

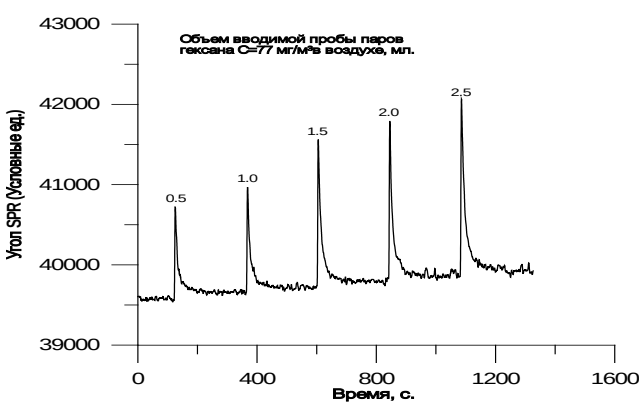


Рис.9. Реакция сенсора на пробы 0.5…2.5 мл гексана с концентрацией С=77 $мг/м^3$ в сухом воздухе.

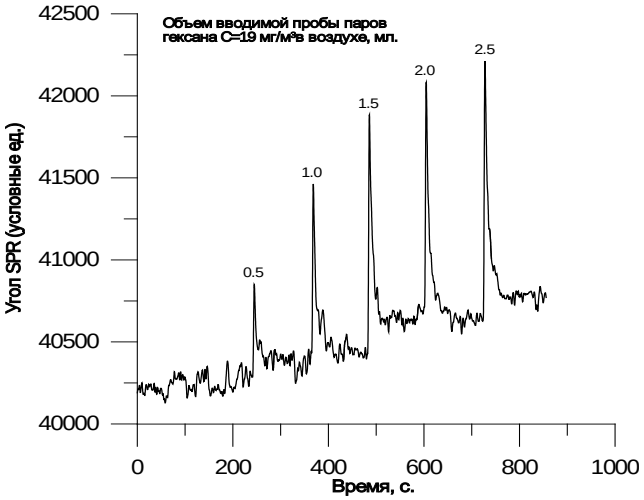


Рис.10. Реакция сенсора на пробы 0.5…2.5 мл гексана с концентрацией С=19 $мг/м^3$ в сухом воздухе.

Можно предположить, что это связано с меньшей сорбционной емкостью более тонкого слоя Al_2O_3 и более быстрой разрядкой ионов и радикалов, удерживаемых электрическим полем на поверхности сенсора и затем удаляемых потоком сухого воздуха.

Объективно сравнить чувствительность SPR сенсора при включенном коронном разряде и без него на одном и том же образце не удалось, так как она сильно зависит от исходного состояния его поверхности.

После только продувки сухим воздухом чувствительность к парам гексана достаточно низкая. Применение коронного разряда значительно (на порядки) повышает чувствительность SPR сенсора. Ввод пробы высокой концентрации приводит к насыщению сенсора и необходимости длительной его продувки, а последующее включение разряда - к медленной очистке его поверхности (требуется несколько циклов продувки – очистки).

3. Выводы.

По результатам предварительных исследований и оценке параметров сенсоров, выполненных на основе поверхностного плазмонного резонанса (SPR) с применением коронного разряда, можно сделать следующие выводы:

- Применение коронного разряда оказывает сильное влияние на параметры сенсора,
- Значительно, на порядки, увеличивается его чувствительность, но она зависит от исходного состояния поверхности сенсора и недостаточно стабильна,
- Чувствительность также возрастает при увеличении толщины слоя Al_2O_3 с 1 до 7 нм, но при этом снижается время реакции сенсора,

- Наблюдается обратимое смещение «нулевой» линии, вызванное осаждением на поверхности продуктов плазмы коронного разряда.

Использование коронного разряда при разработке новых типов высокочувствительных SPR сенсоров для анализа микропримесей в газовых средах может быть весьма перспективным направлением с широкой областью применения в технике, медицине, экологии.

Для этого необходимо провести более детальные исследования в данной области:

- Оптимизировать конструкцию разрядной ячейки,
- Исследовать влияние высокочастотного и импульсного коронных разрядов,
- Оптимизировать толщины и материалы сорбционных защитных покрытий SPR сенсора,
- Доработать и усовершенствовать измерительную аппаратуру,
- Исследовать более широкий спектр анализируемых примесей и диапазон их концентраций,
- Исследовать возможность получения дополнительной информации из спектров излучения плазмы коронного разряда и токов разрядки ионов на поверхности сенсора.

Перечисленные направления являются целью продолжения наших дальнейших исследований.

Данная работа выполнена при поддержке программ фундаментальных исследований ОНИТ РАН №1.8 и №2.1.

Gas sensor SPR with corona discharge

A.N.Palagushkin, S.A.Prokopenko, A.P.Sergeev

Abstract. Currently sensors based on the surface plasmon resonance (SPR) are widely used in express analysis of liquid biological environments in microbiology, medicine and genetic engineering. They are also used for the determination of impurities in gas environments. However the main problem of their development is the selection of selective sorption environments applied to the surface of the sensor to determine the specific impurities. Properties of sorbent in combination with high sensitivity SPR define achievable sensor parameters. Having nanometer thickness of sorbents, the surface properties make the main contribution in sensitivity. Impact of plasma positive corona discharge leads to effective desorption of impurities blocking active centers of adsorption. In addition, the surface of the sensor is exposed by radicals arising in the discharge plasma during ionization of impurities. The use of effects of corona discharge for creating SPR sensors for detection traces of contaminants in gas environments that is discussed in this article.

Keywords: gas sensor, corona discharge, resonance

Литература

1. Ю.П. Райзнер. «Физика газового разряда», М., «Наука», 1992 г.,536с.
2. K. Sekimoto. M. Takayama. "Negative ion evolution and formation in atmospheric pressure corona discharges between point-to-plane electrodes". 30th ICPIG. August 28th – September 2nd 2011. Belfast. Northern Ireland. UK.
3. «Surface Plasmon Resonance Based Sensors». Springer Series on Chemical Sensors and Biosensors. 1612-7617. Volume 4. 2006.
4. «Handbook of Surface Plasmon Resonance». Edited by R.B.M. Schasfoort and Anna J. Tudos. The Royal Society of Chemistry. 2008.
5. Stefan A. Maier. «Plasmonics fundamentals and applications».
6. Springer Science+Business Media LLC. p.177. 2007.
7. Dror Sarid, William A. Challener. “Modern introduction to surface plasmons: Theory, Mathematica, Modeling and Applications”, Cambridge University Press, 2010.
8. A.L.Mikaelian, B.V.Kryzhanovsky, A.N.Palagushkin, S.A.Prokopenko, A.P.Sergeev, F.A.Yudkin and A.N.Arlamenkov. “Sensors Using Plasmon Nanostructures”. Optical Memory and Neural Networks (Information Optics). Vol.14. No.4. 2005. pp.229-244.
9. A.N.Palagushkin. S.A.Prokopenko. A.P.Sergeev. Optical Memory and Neural Networks (Information Optics). v.16. 288-294 (2007); ibid v.18. 156–163 (2009).

Библиотека параллельной обработки сигналов

Г.О. Райко

Аннотация: В статье рассматривается библиотека, реализующая инфраструктуру для распараллеливания задач обработки сигналов. Библиотека предоставляет унифицированное окружение для прикладного программного обеспечения многопроцессорных вычислительных комплексов цифровой обработки сигналов, реализуя примитивы декомпозиции задачи по процессорам и обеспечивая обмены распределенными многомерными объектами данных.

Ключевые слова: параллельная обработка сигналов, ПОС

1. Введение

Процесс цифровой обработки информации в современных радиолокационных и гидроакустических комплексах включает в себя разнообразные операции над многомерными данными: быстрое преобразование Фурье, операции линейной алгебры, адаптивная фильтрация, формирование характеристик направленности, корреляционный и спектральный анализ. Выполнение каждой из таких операций может требовать до нескольких десятков процессоров, связанных сложными внутригрупповыми и межгрупповыми информационными потоками. При этом общее число требуемых процессоров в указанных комплексах может достигать нескольких сотен.

Включая в состав комплексов сотни процессоров, мы сталкиваемся с тем, что интенсивность обменов между ними возрастает, и сложность решения задачи обмена начинает превалировать над арифметической сложностью вычислений. При этом возникает проблема программирования совокупности всех необходимых обменов общим числом более тысячи с обеспечением необходимой ритмичности обработки данных. Для решения указанной задачи в НИИСИ РАН разработана библиотека параллельной обработки сигналов.

2. Характеристики библиотеки

Библиотека параллельной обработки сигналов (ПОС) обеспечивает унифицированное окружение для программ цифровой обработки сигналов (ЦОС), предназначенных для выполнения на многопроцессорных вычислительных комплексах. Библиотека может эффективно применяться, как на ЭВМ, состоящих из нескольких процессоров, так и на комплексах с одной-двумя сотнями процессоров.

Библиотека предоставляет разработчику программные интерфейсы на языке программирования Си, скрывающие как особенности аппаратуры, так и особенности используемой операционной системы и другого общего программного обеспечения. В частности, программный код, разработанный с использованием библиотеки, может быть без изменений запущен и на эмуляторе, функционирующем на инструментальной ЭВМ под управлением ОС общего назначения, и на целевом многопроцессорном комплексе, работающем под управлением ОС реального времени. Тем самым, использование библиотеки позволяет облегчить переход от этапа разработки и отладки отдельных фрагментов кода (на эмуляторе или на малопроцессорной установке) к этапу построения единой задачи, решаемой на многопроцессорном вычислительном комплексе.

Библиотека предоставляет средства для быстрой адаптации разработанных ранее алгоритмов обработки сигналов к изменениям объема обрабатываемых данных и числу процессоров, выполняющих алгоритм. Это повышает степень повторной используемости кодов, разработанных с помощью библиотеки.

Использование библиотеки позволяет разработчику систем ЦОС сконцентрироваться непосредственно на реализации алгоритмов цифровой обработки, сводя к минимуму усилия по распределению задачи между процессорами в многопроцессорной системе и учету деталей передачи информации между процессорами, как на уровне архитектуры прикладной задачи, так и на уровне топологии межпроцессорных коммуникаций.

Библиотека обеспечивает независимость разрабатываемого приложения от используемой целевой платформы – аппаратного обеспечения и операционной системы. В настоящий момент библиотека ПОС функционирует на следующих платформах: целевые системы на базе процессорных элементов К64 и К128 под управлением операционных систем ОСРВ Багет 2.0, ОСРВ Багет 3.0, ПИ «Оболочка-64» и ОС Linux, инструментальная платформа x86 под управлением ОС Linux. Для большинства вышеперечисленных платформ обеспечивается работа как в 32-разрядном, так и в 64-разрядном окружении. Для целевых систем библиотека использует примитивы передачи данных, предоставляемые операционными системами, в частности, в ОС Linux на целевых платформах используется механизм сокетов (socket) и протокол Janeiro, обеспечивающий эффективную передачу больших объемов данных в рамках коммуникационной среды RapidIO. [1] На инструментальной платформе x86 используется пакет MPI. [2]

3. Модель вычислений библиотеки

Модель вычислений, принятая в библиотеке, существенно опирается на следующие особенности задач цифровой обработки данных:

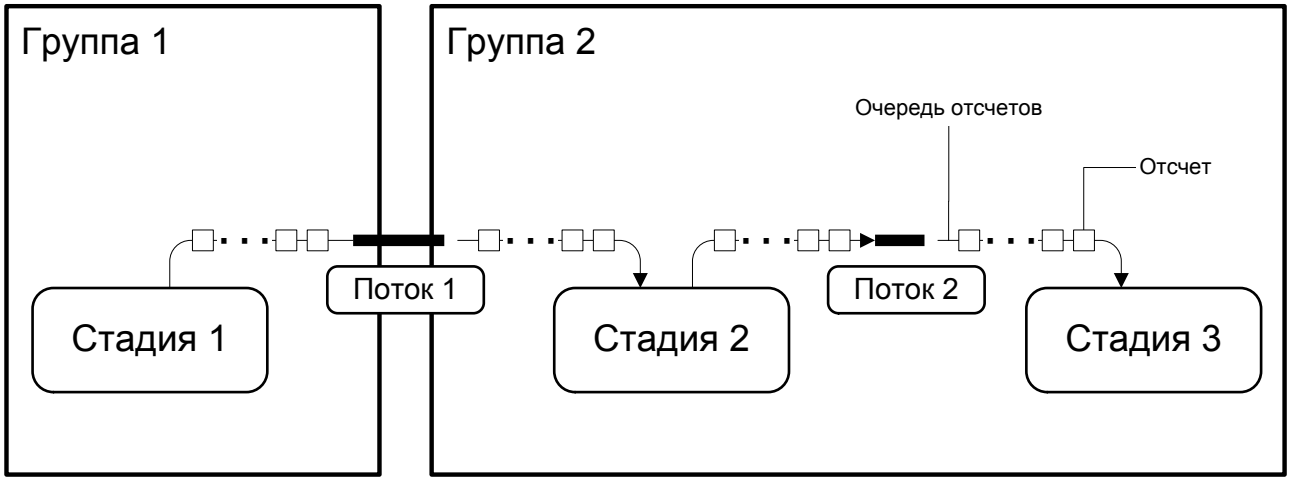


Рис. 1. Основные понятия вычислительной модели.

– тракты цифровой обработки допускают эффективное представление в виде вычислительного конвейера с выделенными стадиями обработки;
– подавляющее большинство вычислительных стадий такого конвейера не могут быть выполнены на одном процессоре и требуют распараллеливания;
– типовые алгоритмы цифровой обработки данных могут быть распараллелены простейшим образом, а именно, SIMD-методом, при котором процессоры выполняют (логически) один и тот же программный код;
– варианты пересылки данных между стадиями вычислительного конвейера описываются несколькими типовыми шаблонами;
– данные пересылаются между стадиями преимущественно в одном направлении, от входа конвейера к его выходу;
– работа вычислительного конвейера – тракта цифровой обработки – представляет собой итерационный процесс, разворачивающийся во времени. На каждой конкретной стадии периодически выполняется одна и та же последовательность действий (характерная для данной стадии). Эта последовательность действий, как правило, состоит из трех этапов: получение входных данных, их обработка – формирование выходных данных по специфическому вычислительному алгоритму, отправка выходных данных следующей стадии (или нескольким стадиям) в конвейере.

В силу этого в рамках библиотеки ПОС приложение цифровой обработки сигналов представляется в виде набора стадий, объединенных в единый вычислительный конвейер. Данные для обработки поступают на вход (входы) этого конвейера (обычно с некоторой заданной частотой). Предполагается, что требуется обеспечить производительность конвейера, достаточную для бесконечно долгой обработки поступающих данных. Еще одно предположение состоит в том, что этого всегда можно достичь, выделив достаточное число аппаратных ресурсов и выбрав подходящий алгоритм цифровой обработки. На каждом временном интервале, стадия конвейера принимает один или несколько массивов данных фиксированного размера (входные отсчеты), выполняет обработку этих входных данных, формируя массивы выходных данных фиксированного размера (выходные отсчеты), и передает выходные данные одной или нескольким последующим стадиям в конвейере. Эти действия, возможно с небольшими вариациями, выполняются и в следующих временных интервалах. Такой набор действий называется «итерацией стадии».

Передача отсчетов между стадиями происходит в рамках так называемых «потоков». Поток соединяет две стадии между собой, отсчеты по потоку всегда текут в одном направлении от стадии-источника к стадии-приемнику. В рамках модели вычислений отсчет всегда рассматривается как трехмерный массив данных фиксированного размера по каждому из измерений. Для буферизации обменов на входе и выходе потока задаются так называемые «очередь входных отсчетов» и «очередь выходных отсчетов», размеры которых фиксированы и не обязаны совпадать. Поток гарантирует доставку отсчетов без потерь и без переупорядочивания, т.е. отсчеты будут приняты стадией-приемником в том же порядке, в каком были отправлены стадией-источником.

Для организации параллельной работы в модели вычислений определяется понятие «группа процессоров» фиксированного размера Вычислительные стадии привязываются к группам. К одной группе может быть привязано несколько стадий. Если к группе привязана ровно одна стадия, то каждый процессор в группе выполняет один и тот же код – итерацию стадии. Если к группе привязано несколько стадий, скажем, первая, вторая, …, последняя, то по-прежнему каждый процессор будет выполнять один и тот же код: итерацию первой стадии, итерацию второй стадии, …, итерацию последней стадии.

На рисунке 1 схематически изображены основные понятия вычислительной модели – группы процессоров («Группа 1» и «Группа 2»), стадии («Стадия 1», «Стадия 2» и «Стадия 3») и потоки («Поток 1» и «Поток 2», причем последний соединяет стадии, расположенные на одной группе процессоров).

Введение понятий «стадия» и «поток» позволяет описать вычислительный конвейер с логической точки зрения, абстрагируясь от того факта, что стадии выполняются на нескольких процессорах, каждый

отсчет распределен по памятям этих процессоров, а передача отсчета требует выполнения нескольких коммуникационных транзакций с различными отправителями и получателями. Таким образом, библиотека позволяет программисту работать над созданием программного кода, руководствуясь подобными «укрупненными» схемами и прилагая минимальные усилия на решение проблем, связанных с распараллеливанием.

Например, в предположении, что в примере выше «Группа 1» состоит из двух, а «Группа 2» – из четырех процессоров, передача данных между процессорами может быть организована следующими способами. В ситуации, изображенной на рисунке 2, отсчет стадии 1 состоит из двух частей (т.к. стадия 1 привязана к группе из двух процессоров), а на стадии 2 – из четырех. При выбранном способе передачи с каждого процессора стадии 1 на каждый процессор стадии 2 необходимо передать четверть отсчета.

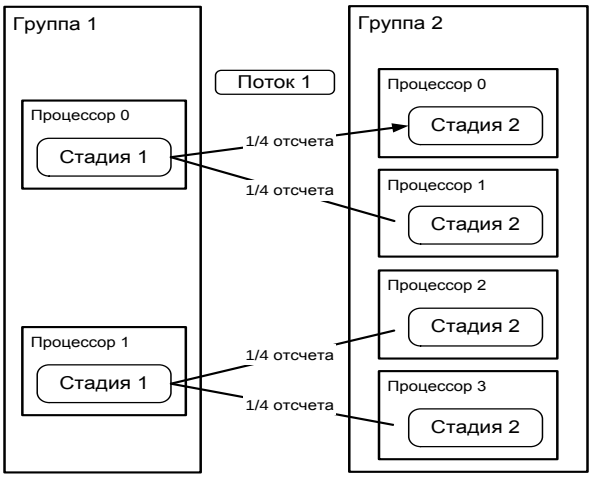


Рис. 2. Передача отсчета.

В ситуации, изображенной на рисунке 3, требуется передать данные так, чтобы каждый процессор приемной стадии 2 получил порции отсчета от каждого процессора стадии 1 (т.е. переупорядочить данные при передаче). В этом случае с каждого процессора стадии 1 на каждый процессор стадии 2 необходимо передать восьмую часть отсчета.

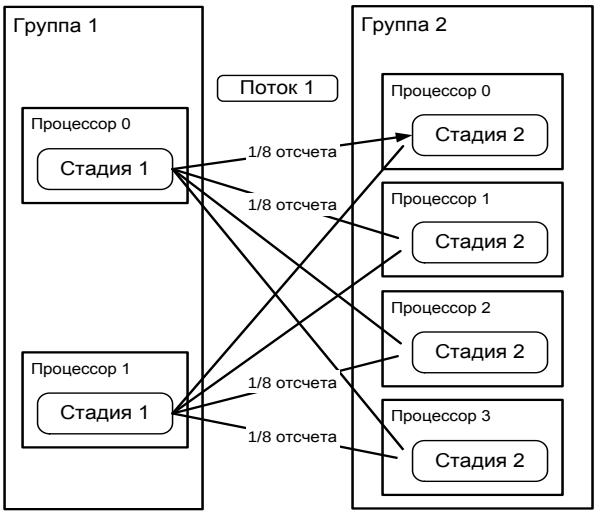


Рис. 3. Передача отсчета с транспонированием.

Прикладной программист указывает, какой из способов следует избрать, после чего библиотека реализует его автоматически. Помимо способов, разобранных в примерах выше, библиотека поддерживает и другие, они описываются ниже при обсуждении понятия «разбиение потока».

Группы процессоров, стадии и потоки прикладной задачи, описываются в специальных структурах данных библиотеки, называемых «конфигурация задачи». В конфигурации задается число процессоров в каждой группе, привязка стадий к группам, привязка потоков к стадиям, размеры очередей отсчетов и некоторые другие параметры. Таким образом, редко изменяемый после завершения автономной отладки код алгоритмов, реализуемых стадией, отделен от часто изменяемых параметров, таких как число процессоров, выделяемых для стадии, размеры очередей отсчетов и т.д.

Ниже описаны основные особенности понятий, на которых строится библиотека ПОС: «группа процессоров», «вычислительная стадия», «поток» и «отсчет».

4. Основные понятия библиотеки

Группа процессоров – набор процессоров, на которых исполняется одна или несколько вычислительных стадий. Каждый процессор многопроцессорной системы по определению принадлежит какой-либо группе, процессор не может принадлежать нескольким группам одновременно. Таким образом, вся совокупность процессоров многопроцессорной системы разбивается на непересекающиеся группы процессоров.

С помощью понятия «группа процессоров» производится привязка вычислительных стадий к процессорам целевой системы для обеспечения вычислительной мощности, требуемой стадией, а также для выполнения нескольких вычислительных стадий на одном множестве процессоров.

Например, на начальном этапе разработки программного кода вычислительная стадия может быть привязана к группе из одного процессора. По мере усложнения задачи, решаемой данной стадией, число процессоров в этой группе может быть увеличено. На более поздних этапах разработки, код стадии может быть оптимизирован, и размер процессорной группы (число процессоров в ней) уменьшится. Понятие «группы процессов» позволяет варьировать число процессоров, отведенных стадии, не меняя при этом программный код самой стадии.

В простейшем случае, к одной группе процессоров привязывается ровно одна вычислительная стадия, использующая все процессоры группы. Однако к группе можно привязать и несколько стадий. Например, если в задаче возникает несколько «мелких» стадий (не требующих большого вычислительного ресурса), целесообразно привязать их к одной процессорной группе, состоящей из одного процессора. Позже, по мере развития кода, может оказаться, что вычислительная нагрузка одной из «мелких» стадий возросла и целесообразно привязать эту стадию к отдельной процессорной группе. Как уже

отмечалось ранее, подобного рода перепривязки не влекут никаких изменений в программном коде стадий.

Библиотека определяет вычислительную стадию в виде набора из нескольких входов для приема данных, нескольких выходов для отправки данных и алгоритма переработки входных данных в выходные. Входы для приема данных и выходы для отправки называются входными и выходными потоками.

Прикладной программист реализует алгоритм обработки сигналов в виде набора функций - методов стадии - с определенной семантикой и определенного набора типов (структур), описывающих характеристики алгоритма с точки зрения библиотеки: структуру стадии, определяющую глобальные переменные, требуемые алгоритму, и структуру конфигурации стадии, задающую поведение стадии (обеспечивающую параметризацию стадии).

Разделение описания стадии на две структуры – собственно структуру стадии и структуру конфигурации стадии – обусловлено тем, что для адаптации к специфике конкретных задач переменные алгоритма, как правило, делят на два класса: внутренние переменные алгоритма и конфигурационные параметры, задающие поведением алгоритма.

Указанные структуры и набор методов стадии задают тип стадии, полностью идентифицирующий конкретную стадию в рамках библиотеки. Основным методом стадии, содержащим собственно вычислительный алгоритм, является метод `iter` – «выполнить одну итерацию стадии». Предполагается, что при каждом вызове метода происходит извлечение готовых отсчетов с данными из входных потоков, переработка входных данных в выходные вычислительным алгоритмом, т.е. формирование отсчетов выходных потоков и отправка отсчетов на следующую стадию. Задача остальных методов стадии - начальная инициализация алгоритма, настройка выходных потоков, завершение работы алгоритма и некоторые другие вспомогательные действия.

Поток данных осуществляет прозрачную для программиста передачу отсчетов – трехмерных массивов данных – от стадии-источника к стадии-приемнику. Таким образом, поток является однонаправленным каналом передачи информации.

Поток характеризуется следующими свойствами:

– тип элементов передаваемого отсчета (массив целых, действительных или комплексных чисел);

– размеры передаваемого отсчета по трем измерениям;

– тип исходного разбиения данных на стадии-источнике, т.е. как распределен выходной отсчет по процессорам группы, к которой привязана стадия-источник;

– тип требуемого разбиения данных на стадии-приемнике, т.е. как следует распределить входной отсчет по процессорам группы, к которой привязана стадия-приемник,

– размер выходной очереди отсчетов (очереди отсчетов на стадии-источнике)

– размер входной очереди отсчетов (очереди отсчетов на стадии-приемнике).

Библиотека поддерживает четыре типа разбиений: нет разбиения (`N`), т.е. на каждом процессоре имеется копия всего отсчета, разбиение по процессорам вдоль измерения `X`, разбиение вдоль измерения `Y` и разбиение вдоль измерения `Z`. Разбиение самого потока определяется как пара разбиений; например, разбиение потока `ZxN` означает, что на источнике разбиение проводится вдоль измерения `Z`, а на приемнике разбиение отсутствует, т.е. на каждом процессоре стадии-приемника нужно сформировать полный отсчет, собрав его по частям с процессоров стадии-источника. Хотя теоретически могут существовать 16 типов разбиений потока, библиотека допускает лишь некоторые разбиения:

– тривиальные разбиения (`NxN`, `XxX`, `YxY`, `ZxZ`);

– с переупорядочиванием (`XxZ`, `YxZ`);

– размножение (`ZxN`).

Кроме того, привязывая стадию-приемник или стадию-источник к группе из одного процессора и используя одно из тривиальных разбиений, можно получить стандартные мультиплексор (многие-к-одному) и демультиплексор (один-ко-многим).

Рисунок 4 иллюстрирует некоторые типовые разбиения потока.

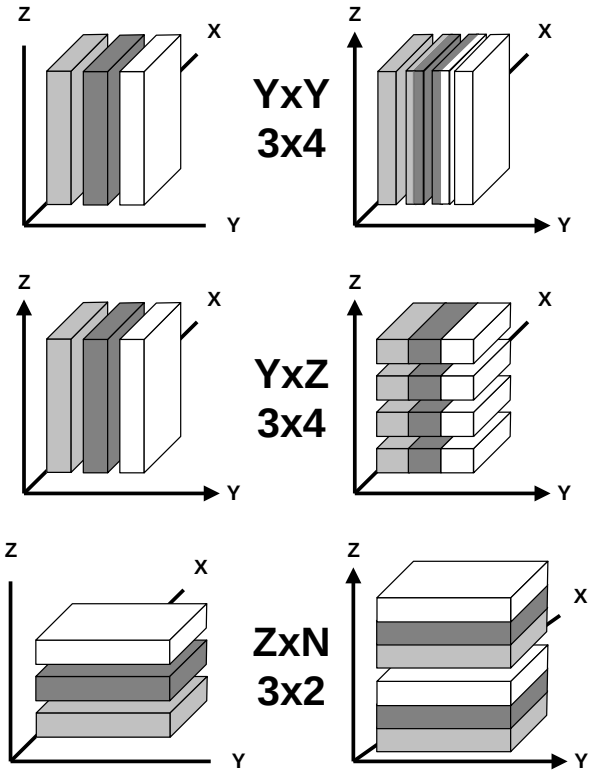


Рис. 4. Типовые разбиения потока.

Тип передаваемых данных, размерности отсчета и исходное разбиение задаются в коде стадии-источника. Размеры очередей и требуемое разбиение на стадии-приемнике указываются в конфигурации задачи. Дело в том, что многие алгоритмы цифровой обработки сигналов нечувствительны к разбиению входных данных. Для некоторых классов алгоритмов, например, допускается разбиение данных как по размерности `X`, так и по размерности `Y`. Реализации таких алгоритмов

могут динамически определять разбиение входного потока и перестраивать функции обработки соответствующим образом. В этом случае можно говорить о том, что реализация стадии дополнительно параметризована входным разбиением. При включении такой стадии в задачу ЦОС, управляя входным разбиением, можно повысить эффективность передачи данных, например, добиваясь тривиального разбиения потока.

При передачах по потоку библиотека также учитывает, локален ли поток, т.е. выполняются ли стадия-источник и стадия-приемник этого потока на одной группе процессоров. Если поток не локален, то потребуется фактически передавать данные по коммуникационной сети. Если же поток локален, то достаточно передавать от стадии к стадии указатели на данные, избегая копирования самих данных.

Если разбиение отсчета на стадии-источнике не совпадает с разбиением на стадии-приемнике, т.е. разбиение потока не является тривиальным, поток осуществляет сборку данных во входном отсчете на стадии-приемнике. Сборка выполняется также, если при тривиальном разбиении, размер группы процессоров стадии-источника не делит размер группы стадии-приемника, поскольку в этом случае на каких-то (или даже всех) процессорах группы необходимо собрать данные с нескольких процессоров источника. Например, если передача происходит от 2 процессоров к 4, то при тривиальном разбиении каждый процессор из четверки получает данные только от одного процессора пары, и сборка данных не нужна. Увеличение числа передающих процессоров до 3, приведет к тому, что каждому процессору из четверки придется принимать данные от 2 процессоров (из 3). В этом случае потребуется сборка данных.

При сборке пришедшие на каждый процессор данные копируются из промежуточных буферов в результирующий массив. Если сборка не нужна, библиотека не создает промежуточных буферов, а данные принимаются сразу в результирующий массив.

Передачи по потоку являются синхронными с точки зрения программиста, т.е. соответствующие вызовы блокируются, если данные не готовы. Передачи являются асинхронными с точки зрения коммуникационной сети, т.е. прикладной программист не знает и не контролирует, в какой момент времени происходит передача того или иного отсчета (или его частей) по физическим каналам коммуникационной сети. При отправке отсчет (или его часть – порция данных, сформированная на данном процессоре) попадает в выходную очередь отсчетов и не будет отправлен получателю, пока не будут переданы все отсчеты, поступившие в очередь ранее. Однако функция отправки не будет блокироваться до тех пор, пока у потока существует свободный для передачи отсчет, т.е. пока число отсчетов в выходной очереди не превышает ее размера (заданного для потока в конфигурации задачи). Аналогично, если во входной очереди отсчетов уже находятся принятые отсчеты, функция, извлекающая отсчеты из входной очереди, не будет блокироваться. Программист обязан возвратить извлеченный из входной очереди отсчет обратно в поток, иначе библиотека не сможет принимать новые отсчеты.

Отсчет, передаваемый в потоке, помимо самих данных содержит также признак корректности этих данных. Библиотека никак не интерпретирует признак корректности, доставляя такой отсчет стадии точно также, как и корректные отсчеты. Прикладной программист должен проанализировать признак корректности перед обработкой отсчета. Впрочем, как правило, алгоритмы цифровой обработки сигналов, находящиеся в начале и середине вычислительного конвейера, нечувствительны к некорректным данным, поэтому, обычно, в таких стадиях просто переносят признак корректности из входного отсчета в выходной и отправляют такой отсчет на следующую стадию. А реакцию на некорректные отсчеты реализуют в конечных стадиях конвейера. Такой подход, в частности, позволяет обеспечить равномерную производительность вычислительного конвейера вне зависимости от корректности поступающих на вход данных (которая определяется либо приемной аппаратурой, либо начальной стадией конвейера).

Помимо признака корректности каждый отчет может иметь свой номер. Номер отсчета задается прикладным программистом в стадии-отправителе, библиотека никак не интерпретирует значения номера. Предполагается, что номер будет сформирован начальной стадией в конвейере по номеру своей итерации, последующие стадии будут устанавливать значение номера по входному отсчету. Допустимы, конечно, и иные правила нумерации отсчетов.

В целях отладки поток можно перевести в неактивный режим. Отправка отсчетов по такому потоку не производится (функция отправки возвращает управление, не передавая отсчет в выходную очередь). Функция приема всегда возвращает специальный недействительный отсчет с признаком некорректных данных. Режим работы потока задается в конфигурации.

Поток можно проредить, т.е. передавать или принимать не все отсчеты в потоке без изменения кода стадии. Параметры прореживания задаются в конфигурации и определяют шаг и интервал передачи. При прореживании функция отправки возвратит управление, не передавая отсчет в выходную очередь, функция приема возвратит специальный некорректный отсчет, даже если входная очередь не пуста.

ParT: A Parallel DSP Framework

G.O. Raiko

Abstract: In this paper, a framework for parallelizing of digital signal processing (DSP) software is described. The framework provides a unified environment for multiprocessor DSP computing by implementing primitives for application partitioning among the processor sets and by supporting the transfers of distributed multidimensional data objects.

Keywords: ParT, DSP, SIMD, RapidIO

Литература

1. S. Fuller. RapidIO: The Embedded Interconnect. – John Wiley & Sons Ltd., 2004.
2. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 3.0. – High Performance Computing Center Stuttgard, 2012.

Инструментальный комплекс «РИО-оптимизатор» для разработки прикладного ПО с использованием Библиотеки Параллельной Обработки Сигналов

А. Н. Павлов

Аннотация: В статье рассматривается инструментальный комплекс для разработки прикладного ПО с использованием Библиотеки Параллельной Обработки Сигналов (БПОС). Рассмотрены языки, используемые в инструментальном комплексе для описания прикладного ПО и для описания аппаратуры. Предложен технологический процесс разработки прикладного ПО с использованием БПОС.

Ключевые слова: цифровая обработка сигналов, БПОС, RapidIO, РИО-оптимизатор

1. Введение

Разработанная в ФГУ ФНЦ НИИСИ РАН Библиотека Параллельной Обработки Сигналов (БПОС, [1,2]) позволяет прикладному программисту проектировать код многостадийной обработки сигналов, абстрагируясь от

количества и расположения на физической установке микропроцессоров, необходимых для выполнения вычислительной работы;

выбора физических каналов передачи информации от микропроцессора к микропроцессору и схемы маршрутизации установки.

После того, как такой БПОС-код многостадийной обработки сигналов вчерне отлажен, прикладной программист должен выбрать схему маршрутизации целевой многопроцессорной установки, определить количество процессоров, выделяемых на каждую стадию обработки и, наконец, расположить «логические» процессоры, выделяемую на каждую стадию, на конкретном многопроцессорном целевом комплексе.

Инструментальный комплекс «РИО-оптимизатор» предназначен для создания и предварительной отладки целевого многостадийного БПОС-кода на инструментальном комплексе, привязки виртуальных объектов библиотеки БПОС, созданных в процессе реализации кода, к реальной физической установке (целевому комплексу), на котором этот код будет выполняться и окончательной отладке и оптимизации кода на целевом вычислительном комплексе. Задача оптимальной привязки подобна задачам оптимальной маршрутизации, и скорее всего NP-трудна, поэтому процедуры оптимизации с использованием комплекса «РИО-оптимизатор» носят эвристический, «человеко-машинный», итеративный характер: «РИО-оптимизатор» позволяет разработчику системы быстро перестраивать варианты привязки, отбрасывая заведомо неудовлетворительные и получая информацию об узких местах подходящих вариантов, с целью их оптимизации.

В процессе привязки человек имеет дело несколькими графовыми структурами:

структура потоков и стадий БПОС-кода обработки сигналов, структура многопроцессорного целевого комплекса с физическими РИО-каналами передачи информации; структура РИО-маршрутизации целевого РИО-комплекса. Пока сложности перечисленных графовых структур невелики, человек в состоянии справиться с задачей отладки и оптимизации программно-аппаратного комплекса без формализации этих графовых структур и автоматизации манипуляций с ними. Когда же число узлов РИО-сети и число стадий прикладного кода переваливает за десяток, без формализации и автоматизации работы с описаниями аппаратуры и прикладного кода не обойтись. РИО-оптимизатор как раз и является инструментом такой автоматизации.

Применение РИО-оптимизатора к решению конкретной задачи по привязке уже созданного БПОС-кода к уже собранному целевому РИО-комплексу, начинается с «загрузке» в Оптимизатор формальных описаний БПОС-кода, целевого РИО-комплекса и таблиц РИО-маршрутизации. В идеальном случае, в процессе оптимизации привязки, эти формальные описания не меняются. На практике, загруженные в РИО-оптимизатор описания в ходе отладки приходится перестраиваться, поэтому в РИО-оптимизатор должны быть включены средства редактирования этих формальных описаний и проверки их корректности. С учетом высокой сложности целевого комплекса, эти средства редактирования не могут быть чисто текстовыми, а должны сочетать текстовый и графический интерфейсы. Наконец, в РИО-оптимизатор должны быть включены средства проверки соответствия целевого РИО-комплекса с загруженными таблицами маршрутизации, формальным описаниям, с которыми оперирует РИО-оптимизатор.

2. Формальное описание БПОС-кода: UML-модель

На рисунке 1 изображена UML-модель вычислений БПОС.

Вычислительный конвейер задачи ЦОС состоит из нескольких стадий, передача данных между которыми осуществляется при помощи потоков.

Каждая стадия получает на вход блок данных

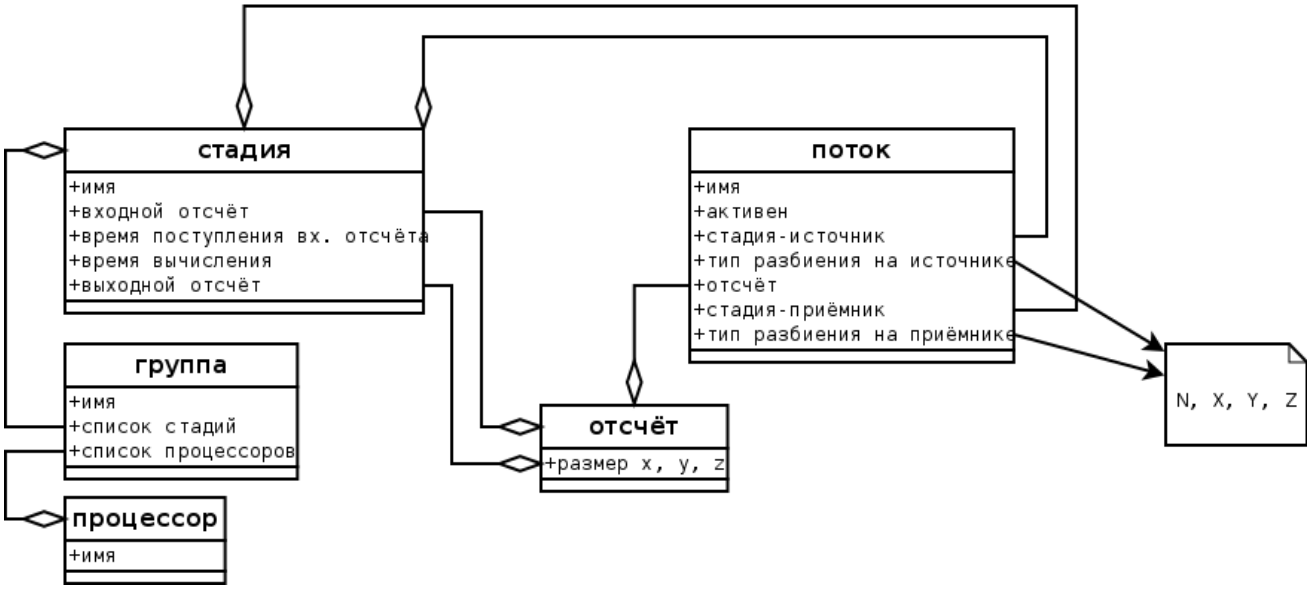


Рис. 1. UML-модель вычислений БПОС

определённого размера (входной отсчёт), производит его обработку, формирует блок выходных данных (выходной отсчёт), который передаётся на вход следующей стадии вычислительного конвейера.

Для указания того, как стадии связаны между собой, использовано понятие потока, моделирующего одностороннюю передачу данных с выхода одной Для явного описания распараллеливания вычислений по нескольким микропроцессорам в модель введено понятие группа, которое указывает привязку вычислительных стадий к конкретным процессорам вычислительного комплекса. Манипуляции с параметрами процессорных групп позволяют оперативно перераспределить вычислительную нагрузку между процессорами комплекса не внося явных изменений в код стадии

Введение отдельных понятий стадия и поток, а также процессор и группа процессоров позволило выполнять задачу написания и отладки вычислительного конвейера от задачи адаптации готового отлаженного вычислительного конвейера к конкретной аппаратной конфигурации вычислительного комплекса, в частности от задачи распараллеливания кода.

Формальное описание аппаратуры РИО-комплекса: NIISI-HDL

Для того, чтобы оперативно и с минимальным ручным трудом оценить возможность запуска программных компонент на выбранной аппаратной конфигурации необходима фиксация формального языка описания аппаратуры комплекса. Этот основанный на XML формальный язык (NIISI-HDL) является входным языком для РИО-оптимизатора.

Конструктивно, АПК ЦОС на основе разработок НИИСИ РАН, могут быть описаны как четырёхуровневые иерархические системы с иерархией понятий, изображенных на рисунке 2.

Этот набор понятий достаточно полон для решения тех задач оптимизации, которые поддерживает РИО-оптимизатор.

На нижнем уровне иерархии находятся БИС и СБИС, для которых характерна конструктивная неделимость. Для соединения СБИС в рамках процессорного модуля (печатной платы) используются наплатные печатные соединения. Для соединения плат с другими компонентами комплекса используются наплатные разъёмы.

На следующем уровне иерархии располагаются процессорные и периферийные модули, выполненные в виде отдельных печатных плат и соединённые между собой по RapidIO при помощи кросс-платы прибора ЦОС либо при помощи внутриприборных кабельных межмодульных соединений. Номенклатура процессорных и периферийных модулей также довольно ограничена. Возможности по коммутации модулей при помощи внутриприборных кабельных межмодульных соединений довольно велики.

На высшем уровне иерархии находится комплекс, который представляет собой несколько приборов ЦОС, вместе с кабельными межприборными соединениями. В частном случае комплекс может состоять из одного прибора ЦОС.

Понятия языка NIISI-HDL

Законченное описание аппаратной компоненты АПК на языке NIISI-HDL называется проектом. РИО-оптимизатор использует не только описание аппаратуры комплекса, но также описание программной компоненты и её размещения на аппаратуре.

Понятия языка NIISI-HDL формализуют описанную ранее иерархию, изображенную на рисунке 2.

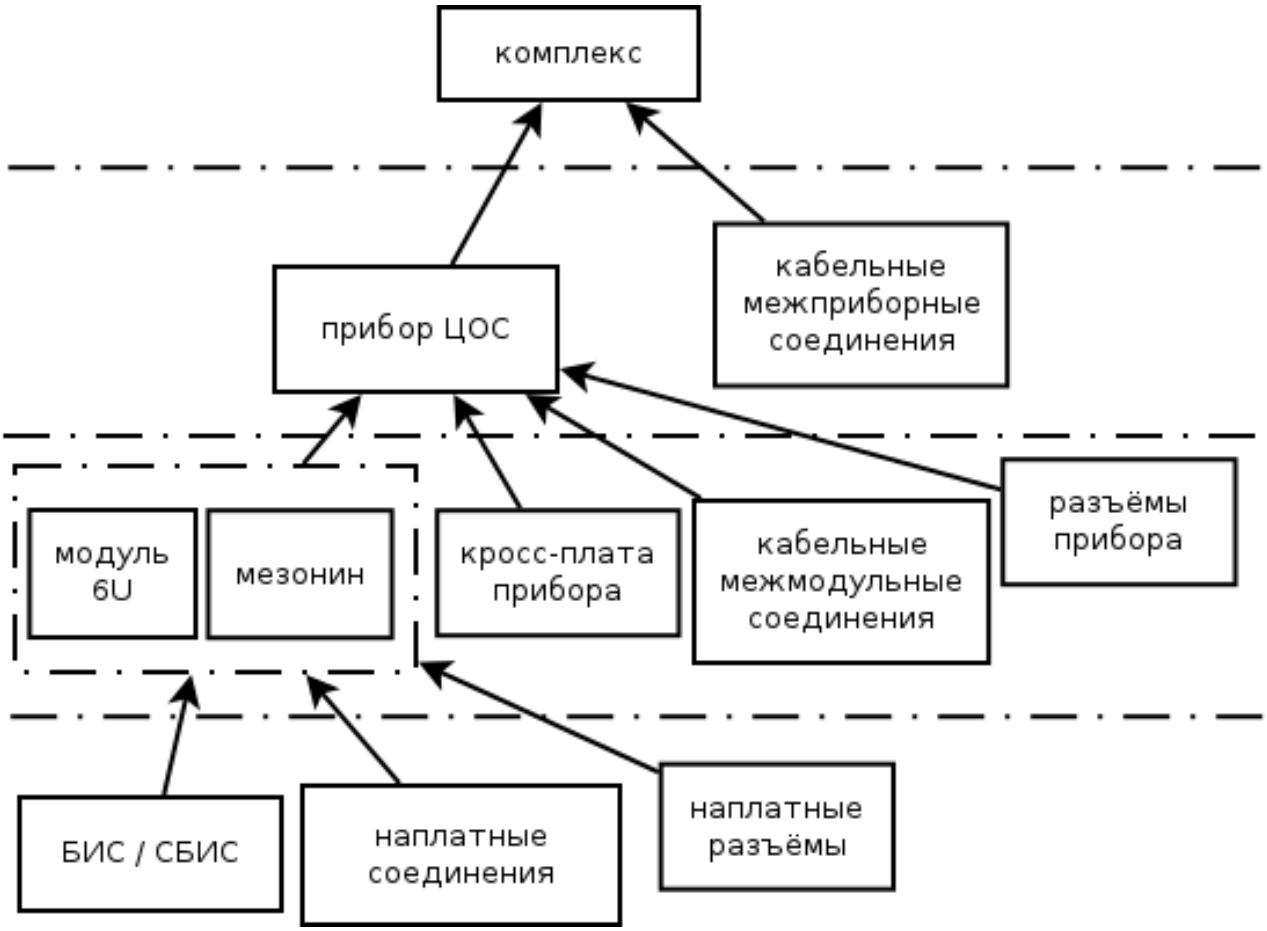


Рис. 2. Схема АПК ЦОС как четырёхуровневой иерархической системы

Можно выделить следующие три класса сущностей:

композитные сущности (далее блоки) — процессорные и периферийные модули, приборы; частным случаем блока является СБИС, хотя иерархия рисунка 2 не предусматривает у СБИС субблоков;

каналы — наплатные каналы RapidIO, каналы на кросс-плате, внутриприборные и межприборные кабельные каналы;

порты — точки в которых возможно подключение канала передачи данных к блоку.

Блоки, входящие в состав блока А будем называть субблоками блока А. Блоки, которые не содержат субблоков будем называть атомарными. Так как описание комплекса иерархическое, то комплекс описывается как совокупность блоков-приборов с явно указанными портами для подключения межприборных каналов, и собственно межприборных кабельных каналов. Каждый из блоков-приборов в свою очередь также описывается как совокупность блоков-процессорных модулей, блоков-кросс-плат и соединяющих блоки каналов и т.д. Описание заканчивается на уровне блоков-СБИС, описания для которых являются атомарными, то есть не содержат субблоков более низкого уровня, а имеют только порты для подключения каналов.

Описание на языке NIISI-HDL использует следующие понятия:

тип канала — упорядоченная пара (имя_типа_канала, множество_атрибутов) (об атрибутах см. ниже);

тип порта — упорядоченная пара (имя_типа_порта, множество_атрибутов)

тип блока — состоит из двух частей:

интерфейса — в этой части описываются порты (т.е. экземпляры портов) для данного типа блока, для каждого порта указывается имя и тип;

реализации — в этой части описывается внутренняя структура блока:

описываются субблоки данного блока (т.е. экземпляры субблоков, из которых состоит блок), для каждого субблока указывается имя и тип.

описываются каналы, соединяющие субблоки между собой и с портами блока (и даже одни порты блока с другими портами этого же блока) (каналы-экземпляры).

атрибут — упорядоченная пара (имя_атрибута, значение_атрибута).

Для каждого проекта, типа блока, типа канала, типа порта, а также экземпляра блока, экземпляра канала и экземпляра порта считается заданным множество атрибутов. Атрибуты позволяют хранить произвольные дополнительные параметры элементов описания (например, для СБИС — версию СБИС, для процессорного модуля — географический адрес, для кабельного канала — длину кабеля и т.д.).

Проект содержит описание всех типов применяемых блоков. Особо оговаривается тип блока верхнего уровня. Блок верхнего уровня может не иметь портов в интерфейсной части (в этом случае блок верхнего уровня не может являться субблоком блока ещё более высокого уровня) — в этом случае будем говорить о закрытом проекте. На практике удобно отлаживать ПО на отдельной части комплекса, поэтому может быть удобно выбрать в качестве блока верхнего уровня блок, являющийся только частью реального комплекса, который содержит порты для подключения к другим частям комплекса — в этом случае будем говорить об открытом проекте.

Таким образом, для блока верхнего уровня известна структура и субблоки, для которых в свою очередь также известны структура и субблоки и т. д. вплоть до субблоков самого нижнего уровня. Всё это даёт возможность на основании иерархического описания комплекса восстановить полное описание домена RapidIO комплекса.

Изложенный способ описания АПК ЦОС выработан для работы в первую очередь с коммуникационной системой RapidIO комплекса, нет никаких препятствий для:

- неограниченного увеличения уровней иерархии описываемого комплекса (если в дальнейшем изменится количество уровней иерархии; например, комплекс будет состоять не непосредственно из приборов, а из стоек внутри которых установлены приборы, то новый комплекс также будет возможно описать при помощи текущей версии NIISI-HDL);
- при желании можно учесть не только каналы RapidIO комплекса, но и интерфейсы ВСК, Ethernet, Манчестер-2 и PCI (описание для данных каналы не используется при оптимизации обменов по RapidIO при помощи ПО РИО-оптимизатор, но могут быть использованы для передачи параметров ОС, работающей на комплексе).

Часто используемый стандартный (библиотечный) тип блока NIISI-HDL вместо того, чтобы каждый раз непосредственно цитироваться в проекте может быть вынесен в отдельный файл, который может подключаться сразу к нескольким проектам.

При этом, если потребуется обновить этот библиотечный тип блока, то сделать это придётся только один раз, причём сразу после внесения обновления новая версия типа блока будет доступна всем проектам.

XML-схема для языка NIISI-HDL, может быть использована для проверки синтаксической корректности описаний на языке NIISI-HDL помощи свободно-распространяемых программных средств xmlstarlet и xmllint.

Трансформация описания на NIISI-HDL

Описание на NIISI-HDL позволяет компактно описать аппаратуру вычислительного комплекса, а затем, путём трансформации компактного иерархического описания получить полное (и возможно довольно громоздкое) описание домена RapidIO для ПО РИО-оптимизатор. Язык описания домена RapidIO на основе XML (далее rioml), который в настоящее время используется ПО РИО-оптимизатор в качестве входного, описан в статье [3].

Технологический маршрут разработки, отладки и оптимизации БПОС-кода с помощью РИО-оптимизатора

Технологический маршрут разработки, отладки и оптимизации БПОС-кода при помощи РИО-оптимизатора изображен на рисунке 3.

Исходными данными для работы является ТЗ на код.

На первом этапе (Эмулятор) пользователь реализует описанный в ТЗ алгоритм обработки сигналов при помощи БПОС. Непосредственно эмулятор используется отладки БПОС-кода в однопроцессорной конфигурации. По окончании этапа Эмулятор пользователь получает отлаженный БПОС-код, правильной работающий в однопроцессорной конфигурации и готовый для распараллеливания.

На следующем этапе (Оптимизатор) производится распараллеливание и оптимизация БПОС-кода. Предварительная оценка распараллеливания получается при помощи программного средства РИО-калькулятор, входящего в состав РИО-оптимизатора. РИО-калькулятор на основе описания аппаратуры вычислительного комплекса, описания конфигурации БПОС и описания привязки процессорных групп БПОС к процессорам реального комплекса позволяет выявить заведомо неудачные варианты распараллеливания до запуска на реальной аппаратуре, а также указать на узкие места, сдерживающие рост производительности при распараллеливании.

После устранения узких мест при помощи РИО-калькулятора производится тестовый запуск оптимизированного кода на реальной аппаратуре вычислительного комплекса. Тестовый запуск позволяет определить реальные показатели производительности, полученные при конкретной конфигурации. По результатам тестового запуска принимается решение о необходимости дальнейших работ.

Если продемонстрированный уровень производительности недостаточен, то у пользователя имеется несколько возможностей:

- скорректировать техническое задание;
- вернуться на этап Эмулятор с целью внесения изменений в БПОС-код;
- исправить входные данные этапа Оптимизатор:
  - скорректировать конфигурацию БПОС (изменить число процессоров, выделенных конкретной стадии);
  - скорректировать привязку процессорных групп БПОС к процессорам реального комплекса;
  - изменить маршрутизацию коммуникационной среды RapidIO;
  - скорректировать конфигурацию аппаратуры вычислительного комплекса (например, увеличить число доступных процессоров).

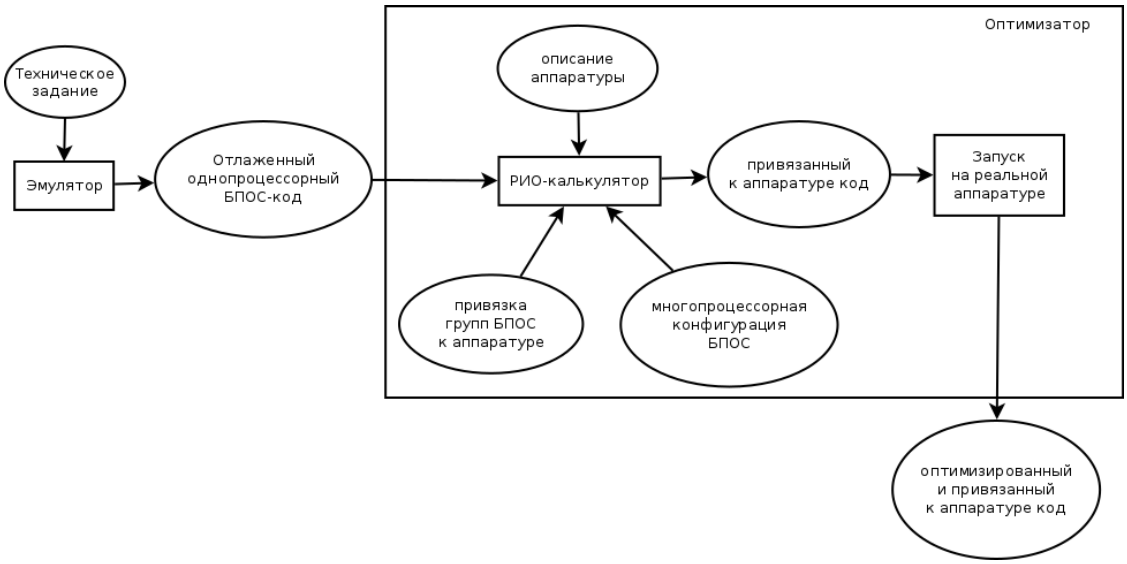


Рис. 3. Технологический маршрут

RIO-optimizer: CAD for ParT-based DSP software development

A.N. Pavlov

Abstract: In this paper, a CAD to assist in creation and optimization of ParT library based DSP software is described. RIO-optimizer provides software and hardware description languages to describe DSP system. Proposed parallel DSP software creation process is used to increase productivity of DSP system developer.

Keywords: digital signal processing, ParT, RapidIO, RIO-optimizer

Литература

1. Г. О. Райко, Ю. А. Павловский, В. С. Мельканович. Технология программирования многопроцессорной обработки гидроакустических сигналов на вычислительных устройствах семейства «КОМДИВ» // Научно-технический сборник Гидроакустика. Вып. 20 (2) СПб.: ОАО «Концерн «Океанприбор», 2014 С. 85-92.

2. Г. О. Райко. Библиотека параллельной обработки сигналов // Труды НИИСИ РАН, 2013. Т.5. №1. С. 22-25.

3. А. Н. Павлов. Формальная модель RapidIO. Сборник Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО / Сборник статей под редакцией академика РАН В. Б. Бетелина .— М.: НИИСИ РАН, 2009.— 215 с. ISBN 978-5-93838-038-7

Реализация алгоритма MG пакета NPB для многопроцессорного вычислительного комплекса на базе микропроцессора КОМДИВ128-РИО

О.Ю. Сударева

Аннотация: В статье рассматривается алгоритм геометрического многосеточного метода из пакета NAS Parallel Benchmarks (NPB) и его реализация на многопроцессорном комплексе на отечественной элементной базе, разрабатываемой в ФГУ ФНЦ НИИСИ РАН. Предлагается схема реализации алгоритма и устанавливается зависимость производительности вычислений по такой схеме от различных характеристик как самого процессора, так и высокопроизводительной коммуникационной среды и других аппаратных узлов. Приводится описание предварительной реализации алгоритма и результаты частичных замеров её производительности.

Ключевые слова: геометрический многосеточный метод, КОМДИВ128-РИО, К128

1. Введение

В данной статье рассматривается вычислительный комплекс на базе разрабатываемой в ФГУ ФНЦ НИИСИ РАН линейки микропроцессоров КОМДИВ (в частности КОМДИВ128-РИО, или сокращённо К128, со встроенным математическим сопроцессором СР2), а также контроллеров и коммутаторов высокопроизводительной коммуникационной сети RapidIO. Такой комплекс может быть использован для распределённых вычислений – например, следующим образом: на каждом процессоре вычисления выполняются при помощи арифметического сопроцессора СР2, а обмены между процессорами осуществляются через программный интерфейс MPI ([2]). В рамках этой схемы возможна эффективная реализация многих алгоритмов, применяемых для решения различных практических задач.

В статье предлагается возможная реализация алгоритма многосеточного метода (MG), входящего в состав набора тестов NAS Parallel Benchmarks ([3]). Описывается общая схема вычислений, которая включает распределение работы между вычислительными узлами, организацию вычислений на СР2 на каждом процессоре и организацию обменов через MPI над RapidIO. Оценивается теоретический максимум производительности для такой реализации, в зависимости от ряда параметров процессора и комплекса в целом. Далее приводится описание разработанной автором предварительной реализации алгоритма и результаты замеров производительности вычислительных ядер на К128. Рассматриваются возможные пути дальнейшей оптимизации вычислений по алгоритму.

По итогам проделанной работы вносится ряд предложений по архитектуре перспективного микропроцессора и вычислительного комплекса на его основе, нацеленных на повышение производительности вычислений по алгоритму MG и сходным алгоритмам.

2. Многопроцессорный вычислительный комплекс на базе К128

В состав К128 входят ядро универсального процессора с архитектурой КОМДИВ64 (развитие RISC-архитектуры MIPS) и 128-разрядный математический сопроцессор СР2. Сопроцессор имеет SIMD-архитектуру, позволяющую выполнять до 40 операций с 32-разрядными вещественными числами за один такт рабочей частоты. К128 имеет контроллер высокопроизводительной коммуникационной сети (ВКС) RapidIO.

На рисунке (1) представлена упрощённая схема К128, на которой изображены структурные блоки, наиболее существенные с точки зрения прикладного программиста, их взаимосвязь по данным (сплошные стрелки) и по управлению (пунктирные стрелки).

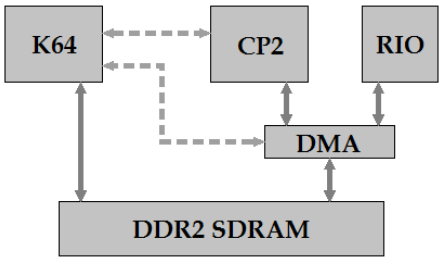


Рис. 1. Структурная схема К128

Полное описание архитектуры см. в [1].

Сопроцессор СР2 содержит 4 вычислительных секции, работающих параллельно. Каждая секция обладает собственным набором из 64 64-разрядных регистров и локальной памятью (LMEM) объёмом 64 Кбайт. Единицей адресации является 64-разрядное слово, которое может рассматриваться как пара вещественных 32-разрядных чисел, одно комплексное число, и др..

Обмен данными между локальной памятью СР2 и глобальной памятью процессора (DDR) осуществляется посредством контроллера DMA. DMA формирует единую очередь для операций загрузки и выгрузки; единицей передачи является 128-разрядное слово.

Локальная память СР2 разделена на два банка, что позволяет совмещать вычисления на СР2 с данными в одной половине памяти и обмены по DMA с другой половиной памяти, другими словами, осуществлять двойную буферизацию вычислений. Использование такого совмещения – двойной буферизации, – наряду с написанием эффективных вычислительных ядер для СР2, – основной приём, позволяющий добиться высокой эффективности процедур на К128.

Микропроцессоры при помощи коммутаторов RapidIO, также разработанных в ФГУ ФНЦ НИИСИ РАН, могут быть объединены в сеть, и таким образом сформировать многопроцессорный вычислительный комплекс. Стандартом передачи данных для таких систем является стандарт передачи сообщений (MPI) [2]. Подробно о портировании процедур MPI на ВКС RapidIO см. в [3].

В процессоре К128, помимо совмещения вычислений на СР2 и пересылок данных между LMEM и DDR, имеется также возможность совмещения вычислений на процессоре и обменов данными через RapidIO. Нужно, однако, иметь в виду, что на обслуживание запросов от RapidIO отводится часть тактов DMA, и кроме того, между DMA и RapidIO могут возникать конфликты доступа к DDR. Это может отрицательно сказаться на производительности вычислительной процедуры на К128.

Пиковая производительность К128 на вещественных операциях одинарной точности, при работе на частоте 200 МГц, составляет 8 GFLOP/s (на инструкции «бабочка Фурье»).

Пиковая скорость передачи данных между LMEM и DDR через DMA — 3,2 Гбайт/с. С учётом того, что часть тактов отводится на технические нужды, фактический максимум скорости составляет порядка 2,7 Гбайт/с.

Пиковая скорость внешнего интерфейса RapidIO — 1 Гбайт/с (500 Мбайт/с на приём и 500 Мбайт/с на передачу). С учётом накладных расходов, по результатам замеров, фактическая максимальная скорость составляет примерно по 420 Мбайт/с на приём и на передачу.

3. Алгоритм многосеточного метода

3.1. NAS Parallel Benchmarks.

Алгоритм геометрического многосеточного метода (MG), который рассматривается в данной работе, входит в состав NAS Parallel Benchmarks (Numerical Aerodynamics Simulation Parallel Benchmarks, или NPB) – см. [4]. Это набор тестов, разработанных в центре NASA для оценки производительности параллельных суперкомпьютеров. Тесты представляют собой некоторые упрощённые базовые алгоритмы, на основе задач вычислительной гидродинамики, которые позволяют оценить характеристики вычислительной системы, существенные при решении реальных задач. Первая версия тестов вышла в 1994; набор тестов продолжает пополняться и уточняться, на текущий момент последняя версия – 2012 года.

Все алгоритмы из NPB имеют так называемое «бумажное» описание: они полностью описаны в тексте, и программисту предлагается реализовать их для конкретной системы любым удобным способом. Накладывается ряд ограничений на используемые средства: например, не разрешается использовать язык ассемблера и подключать нестандартные библиотеки. Цель тестов состоит в том, чтобы проверить, какой производительности вычислений сможет достичь на той или иной системе рядовой программист, при решении своих задач.

В тексте также задаются входные данные и эталонные результаты, для проверки корректности реализации алгоритмов. Все вычисления требуется производить с двойной точностью.

В реализации алгоритма на К128, представленной в данной работе, есть несколько отступлений от требований, заявленных NPB. Во-первых, при реализации использовались библиотека обработки сигналов (dsplib) для К128 и вычислительные ядра для СР2 на языке ассемблера. Во-вторых, в настоящее время сопроцессор СР2 не поддерживает вычислений с двойной точностью. Поэтому везде в работе рассматривается реализация MG для вещественных чисел одинарной точности. Соответственно, погрешность результата вычислений будет существенно больше, чем требуется в тестах NPB. Отметим, однако, что описываемую реализацию в будущем можно путём незначительных правок адаптировать к вычислениям с двойной точностью – общая схема работы и принципиальные оценки эффективности останутся без изменений.

В NPB вводится понятие классов задач – от S до F (последнего на настоящий момент). Различные классы соответствуют различным объёмам входных данных, количествам итераций внутренних циклов в алгоритмах и т.п.. С увеличением класса объём вычислительной работы возрастает экспоненциально. Соответственно, разные классы рассчитаны на вычислительные системы разных масштабов:

- S – маленькие задачи, для тестирования и отладки;
- W – задачи для рабочих станций 90х годов;
- A, B, C – типичные размеры задач для современных персональных компьютеров;
- D, E, F – задачи для распределённых высокопроизводительных вычислительных комплексов.

Имеются готовые реализации алгоритмов, доступные на сайте NPB ([5]), последняя редакция – NPB 3.3.1. Представлены различные версии кодов: в основном на языке FORTRAN, частично на C; последовательная версия и параллельные с использованием моделей программирования MPI и OpenMP; версии на Java и HPF, и др.. Коды сделаны как образец для облегчения работы программисту; никакая совместимость с ними не требуется.

3.2. Алгоритм MG.

Многосеточный метод в настоящее время широко применяется для решения многих задач, в частности, для предобуславливания СЛАУ с разреженными матрицами. В NPB рассматривается простая версия алгоритма: классический геометрический многосеточный метод. Задача тестирует регулярные обмены данными на локальных и межузловых подсистемах передачи.

Формулировка задачи: найти приближённое решение уравнения Пуассона $\nabla^2 u = v$ на сетке $N \times N \times N$, с периодическими граничными условиями. Алгоритм, который требуется использовать для решения, полностью описан в документации NPB. На каждой итерации осуществляется коррекция приближённого решения на наборе сеток от самой мелкой ($N \times N \times N$) до самой грубой ($2 \times 2 \times 2$): производится последовательность проекций, на всё более и более грубые сетки; на самой грубой сетке вычисляется поправка к приближению, после чего эта поправка последовательно интерполируется на более мелкие сетки (вычисляется интерполяция, невязка и сглаживание). Начальное приближение – нулевое. Также для каждого класса задачи задаётся следующее:

- размер сетки N;
- функция в правой части, v – равна 0 везде, кроме нескольких точек, в которых принимает значение 1 или -1;
- количество итераций алгоритма IT;
- норма невязки для искомого решения;
- допустимая погрешность.

Алгоритм, в сущности, представляет собой последовательность трилинейных операторов на сетке (далее Tlin). Tlin переводит трёхмерный массив в трёхмерный массив, каждое новое значение в точке получается в общем случае как сумма 27 соседних старых значений с коэффициентами (см. рисунок (2)).

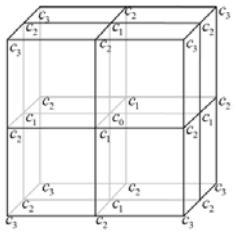


Рис. 2. Tlin: схема вычисления нового значения в точке

Всего используется 4 различных оператора, их коэффициенты заданы и не зависят от уровня:

- A (невязка) – оставляет сетку $M \times M \times M$ неизменной. Его коэффициенты:

$$c_0 = -\frac{8}{3}, \quad c_1 = 0, \quad c_2 = \frac{1}{6}, \quad c_3 = \frac{1}{12};$$

- P (проекция) – уменьшает сетку: $M \times M \times M \mapsto M/2 \times M/2 \times M/2$. Коэффициенты:

$$c_0 = \frac{1}{2}, \quad c_1 = \frac{1}{4}, \quad c_2 = \frac{1}{8}, \quad c_3 = \frac{1}{16};$$

- Q (интерполяция) – увеличивает сетку: $M \times M \times M \mapsto 2M \times 2M \times 2M$. Коэффициенты:

$$c_0 = 1, \quad c_1 = \frac{1}{2}, \quad c_2 = \frac{1}{4}, \quad c_3 = \frac{1}{8};$$

- S (сглаживание) – оставляет сетку $M \times M \times M$ неизменной. Коэффициенты:

$$c_0 = -\frac{3}{8}, \quad c_1 = \frac{1}{32}, \quad c_2 = -\frac{1}{16}, \quad c_3 = 0.$$

При вычислении значений на границе требуется учитывать периодические граничные условия: по каждому из трёх измерений $u_N = u_1$, $u_0 = u_{N-1}$ – соответственно, на входе необходимы значения с противоположной грани обрабатываемого трёхмерного массива. Иллюстрация для различных операторов (для простоты, в двумерном случае) приводится на рисунке (3).

В ходе вычислений каждый следующий оператор применяется к результату предыдущего. В некоторых случаях результат поэлементно прибавляется или вычитается из одного из полученных ранее массивов, поэтому в ходе работы требуется хранить целый набор промежуточных массивов, а все вычисления выполнять out-of-place.

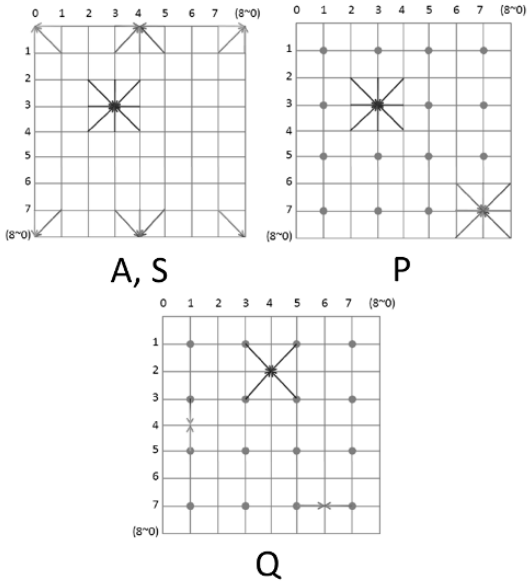


Рис. 3. Tlin: учёт граничных условий

3.3. Схема вычислений для многопроцессорного комплекса на базе К128.

Замеры для референсных кодов показали, что порядка 70% общего времени вычислений в алгоритме занимает вычисление невязки (с оператором A) и полностью аналогичное вычисление сглаживания (с оператором S) на самой мелкой сетке, то есть для массива порядка $N \times N \times N$. Поэтому в качестве базовой операции будет рассматриваться упрощённый Tlin на самой мелкой сетке: все коэффициенты ненулевые, и сетка остаётся неизменной.

Вычисление одного Tlin на нескольких процессорах можно организовать следующим образом. Обрабатываемый массив делится на блоки, каждый процессор обрабатывает свой блок: вычисляет Tlin для его точек. При этом для вычисления значений в граничных точках блока требуется иметь на каждом процессоре 6 дополнительных граничных плоскостей (по одной для каждой грани блока). Точки этих граничных плоскостей являются граничными точками соседних блоков, обрабатываемых на других процессорах, либо копиями противоположных граней того же блока (если по какому-то направлению разбиение массива не производилось). Процедура Tlin на каждом процессоре К128 получает на вход блок с дополнительными границами и вычисляет Tlin, давая на выходе новый блок. Чтобы можно было вычислять следующий Tlin в ходе алгоритма, необходимо произвести обмен граничными плоскостями с соседними процессорами – тогда на каждом процессоре окажется блок нового массива с актуальными дополнительными границами.

При выборе способа разбиения массива для описываемой реализации было решено остановиться на классической схеме: массив разбивается на блоки, по возможности равные по всем измерениям – делится пополам поочерёдно по всем направлениям. Для определённости, сначала осуществляется разбиение по *z* плоскостью, параллельной *Oxy*, на блоки размера *N*×*N*×*N/2*, затем аналогичное разбиение каждого полученного блока по *y*, по *x*, снова по *z* и т.д. до тех пор, пока число блоков не станет равно числу процессоров. Эта схема подходит для случая, когда число процессоров (обозначим его *np*) является степенью 2. Преимущества этого способа:

- общий объём пересылаемых данных минимально возможный;
- на одном процессоре выполняется максимальное возможное количество уровней алгоритма MG;
- схема реализована в референсном коде NPB.

Рассмотрим теперь процедуру Tlin на одном К128, когда входные и выходные данные располагаются в его глобальной памяти. Имеется несколько общих приёмов, позволяющих добиться высокой производительности вычислений на CP2:

1) Написать как можно более «плотное» ядро для CP2, в котором арифметические операции запускаются по возможности на каждом такте.
2) Организовать вычисления так, чтобы на каждый запуск ядра приходилось как можно меньше загрузок данных по DMA из DDR в LMEM и выгрузок обратно – другими словами, загрузив порцию данных в LMEM, выполнить над нею как можно больше вычислений, прежде чем выгружать результат.
3) Оставшиеся пересылки данных скрыть за вычислениями в ядре: два банка LMEM позволяют совмещать вычисления на CP2 с данными в одной половине памяти и обмены по DMA с другой половиной.

Для того чтобы минимизировать количество загрузок и выгрузок данных по DMA, можно было бы распределить работу между четырьмя секциями сопроцессора CP2 аналогично описанному выше распределению между несколькими процессорами на верхнем уровне: массив делится на 8 блоков, каждый блок обрабатывается в половине памяти одной из секций, после чего производится обмен граничными плоскостями. Архитектура К128, однако, не приспособлена для эффективного обмена данными между секциями. Кроме того, объёма локальной памяти недостаточно для хранения всех необходимых промежуточных массивов в ходе вычислений по алгоритму MG, даже для задачи класса S. С другой стороны, на К128 канал доступа к памяти DMA обладает высокой пропускной способностью: время пересылок данных сопоставимо с временем вычислений в ядре даже для таких нагруженных пересылками алгоритмов, как БПФ длинных векторов (см. [6]).

С учётом этих соображений, для вычисления Tlin на К128 предлагается следующая схема. Входной массив разбивается на небольшие блоки, каждый из которых может быть обработан в половине памяти одной секции CP2. Каждый блок, вместе с дополнительными границами, загружается в LMEM, для него в ядре вычисляется Tlin, и полученный блок, без дополнительных границ, выгружается в GMEM. При этом четыре блока обрабатываются в четырёх секциях параллельно, а выгрузка и загрузка блоков в одной половине памяти совмещается с вычислениями над блоками в другой. После обработки всех блоков мы получаем в DDR выходной массив без дополнительных граничных плоскостей. Обмен границами можно произвести одним из двух способов:

1) Если на верхнем уровне по какому-либо направлению разбиение массива между процессорами не производилось, то обе соответствующие граничные плоскости принадлежат полученному блоку, и можно их обменять посредством DMA – такое копирование оказывается более эффективным, чем копирование стандартными средствами непосредственно в DDR.
2) Если по данному направлению разбиение производилось, то обмен с соседними процессорами производится через RapidIO – например, с помощью интерфейса MPI.

На выходе процедуры Tlin на К128 получается блок выходного массива, со всеми необходимыми граничными плоскостями.

В конце алгоритма MG вычисляется норма невязки полученного массива, по формуле:

$$\|u\|_2 = \sqrt{\frac{\sum_{i,j,k} u_{i,j,k}^{\ 2}}{N^3}}. \qquad (1)$$

На каждом процессоре, в ядре на CP2, вычисляется сумма квадратов элементов соответствующего блока, после чего на одном процессоре собирается общая сумма и вычисляется окончательный результат.

4. Теоретические оценки производительности

Чтобы получить ориентировочную предварительную оценку производительности реализации алгоритма MG по описанной схеме, для простоты рассмотрим последовательность базовых операций – Tlin на самой мелкой сетке. Для определённости будем рассматривать в качестве Tlin вычисление невязки.

Все оценки будем выводить в общем виде – в виде формул, зависящих от размеров задачи (т.е. размера сетки $N\times N\times N$), от количества используемых процессоров (np) и параметров одного процессора и вычислительного комплекса. Это позволит получить конкретные оценки для имеющейся в настоящий момент аппаратуры, и кроме того, судить о влиянии той или иной характеристики комплекса на производительность алгоритма. Рассмотрим следующие параметры:

- F – тактовая частота сопроцессора CP2;
- K – количество секций сопроцессора;
- $BW = BW_{DMA}$ – пиковая скорость обменов данными между DDR и LMEM через DMA;
- R – пиковая скорость пересылки данных по RapidIO (суммарная для приёма и передачи);
- $bw = bw_{DDR}$ – пиковая скорость копирования данных в DDR.

Ещё один условный параметр – α, $0 \le \alpha \le 1$ – характеризует влияние топологии соединения процессоров на фактическую скорость пересылки по RapidIO. Подробнее этот параметр будет обсуждаться в разделе 4.3.

Для последней на настоящий момент версии процессора, при его работе на частоте F = 200 МГц, параметры имеют следующие значения:

K = 4;

BW = 2,7 Гбайт/с;

R = 840 Мбайт/с;

bw = 90 Мбайт/с, при копировании посредством управляющего процессора K64.

Значение α определяется экспериментально. В данной статье для получения конкретных оценок будем условно полагать α = 1.

4.1. Производительность ядра Tlin на CP2.

Рассмотрим ядро Tlin на CP2: входные и выходные данные располагаются в LMEM, в K секциях.

В каждой секции вычисляется Tlin для точек блока порядка $N_1\times N_2\times N_3$. При вычислении невязки, $r = v - Au$, значение r в каждой точке представляет собой разность значения v и суммы 27 соседних значений u с четырьмя заданными коэффициентами, как показано на рисунке (2). Коэффициенты могут быть переданы в ядро, например, через регистры GPR (см. [1], [6]); загружать их из LMEM при вычислении очередного значения не требуется. Таким образом, для вычисления каждого выходного значения необходимо, как минимум:

- загрузить 54 входных значения из LMEM (27 точек массива u 27 точек массива v);
- выполнить 27 операций вычитания с накоплением;
- выгрузить полученное значение в LMEM.

Итого, на 55 операций загрузки/выгрузки приходится 27 арифметических операций. Система команд CP2 позволяет запустить за один такт операцию загрузки или выгрузки двух 64-разрядных слов (т.е. четырёх вещественных чисел); также за один такт может быть запущена операция вычитания с накоплением для пары вещественных чисел. Таким образом, на вычисление одной точки требуется, по меньшей мере, $\lceil 54/4 \rceil + 1 = 15$ команд загрузки/выгрузки в LMEM и $\lceil 27/2 \rceil = 14$ арифметических команд. При совместном вычислении четырёх точек потребуется, соответственно, $(54\cdot 4 + 4)/4 = 55$ команд загрузки/выгрузки и $27\cdot 4/2 = 54$ арифметических команды.

На каждом такте CP2 может быть запущена на выполнение VLIW-инструкция, включающая одну команду загрузки/выгрузки и одну арифметическую команду. Отсюда, в идеальном случае, когда отсутствуют вспомогательные операции и различные накладные расходы, общее количество тактов на выполнение ядра будет оцениваться как максимальное из двух: количество команд загрузки/выгрузки и количество арифметических команд. В нашем случае получаем оценку в 55 тактов на 4 точки, или $Tcs = 55N_1N_2N_3/4$ тактов на блок.

Количество арифметических операций с вещественными числами, необходимое на вычисление Tlin одного блока, составляет $FLOP = 30N_1N_2N_3$, причём K блоков обрабатываются в K секциях параллельно. Отсюда получаем формулу для оценки производительности ядра:

$$Perf_{Kern} = \frac{K\cdot FLOP}{Tcs/F} = \frac{24}{11}KF. \qquad (2)$$

Для последней версии процессора, с приведёнными выше значениями параметров, получается оценка

$$Perf_{Kern} = \frac{24}{11}\text{флоп}\cdot 4\cdot 200\text{МГц} \approx 1745\text{Мфлоп/с}.$$

4.2. Производительность процедуры Tlin на K128.

Пусть теперь мы имеем входной блок размера $N_1\times N_2\times N_3$, с дополнительными границами, в DDR процессора K128. Требуется вычислить Tlin и записать выходной блок (без дополнительных границ) на другое место в DDR.

Согласно описанной в разделе 3.3 схеме, блок делится на меньшие блоки, размера $M_1\times M_2\times M_3$. Эти блоки обрабатываются на CP2 постранично: вычисления производятся над K блоками параллельно в половине памяти секций CP2, и одновременно из другой половины в DDR выгружаются K предыдущих блоков-результатов и загружаются K следующих входных блоков. В предположении максимально возможного совмещения вычислений с пересылками время работы процедуры будет определяться большим из двух:

- временем вычислений в ядре на CP2 с одной страницей блоков;
- суммарным временем выгрузки и загрузки одной страницы блоков по DMA.

Время вычислений в ядре с одной страницей определяется из количества арифметических операций и производительности, полученной в разделе 4.1:

(3) $$t_{exec} = \frac{K \cdot FLOP}{Perf_{Kern}} = \frac{Tcs}{F} = \frac{55 M_1 M_2 M_3}{4F}.$$

Общее количество страниц составляет $N_1 N_2 N_3 / (K M_1 M_2 M_3)$, соответственно, время вычислений в ядре для всего блока оценивается как

(4) $$T_{exec} = \frac{N_1 N_2 N_3}{K M_1 M_2 M_3} \cdot \frac{55 M_1 M_2 M_3}{4F} = \frac{55 N_1 N_2 N_3}{4KF}.$$

Для приведённых значений параметров

$$T_{exec} = 55 N_1 N_2 N_3 / (4 \cdot 4 \cdot 200 \cdot 10^6) \text{с} \approx$$
$$\approx 1{,}719 \cdot 10^{-2} N_1 N_2 N_3 \text{мкс}.$$

Теперь оценим время на пересылки по DMA. На вход загружаются блоки с дополнительными граничными плоскостями: всего по $(M_1+2)(M_2+2)(M_3+2)$ элементов на каждый. Для максимально возможных блоков, которые могут быть обработаны в половине памяти одной секции (16×8×8), эту величину можно для простоты приблизительно оценить сверху как $2M_1M_2M_3$. Выгружаются блоки без дополнительных плоскостей – по $M_1M_2M_3$ элементов. Т.о., общий объём данных, передаваемых через DMA для одной страницы, составляет приблизительно

$$K \cdot (2 M_1 M_2 M_3 + M_1 M_2 M_3) = 3 K M_1 M_2 M_3.$$

Отсюда находим, что время на пересылки по DMA для одной страницы оценивается снизу как

(5) $$t_{ldst} = \frac{3 K M_1 M_2 M_3}{BW}.$$

Соответственно, время на пересылки при обработке всего блока составляет

(6) $$T_{ldst} = \frac{N_1 N_2 N_3}{K M_1 M_2 M_3} \cdot \frac{3 K M_1 M_2 M_3}{BW} = \frac{3 N_1 N_2 N_3}{BW},$$

где BW – пиковая скорость обменов данными через DMA.

Подставляя значения параметров для последней версии процессора, получаем

$$T_{ldst} = 3 N_1 N_2 N_3 \text{байт} / 2{,}7 \text{Гбайт/с} \approx$$
$$\approx 1{,}11 \cdot 10^{-3} N_1 N_2 N_3 \text{мкс}.$$

Таким образом, в настоящий момент на процессоре К128, при работе на частоте 200 МГц, $T_{ldst} < T_{exec}$. Это означает, что общее время вычислений для процедуры Tlin можно оценить как T_{exec}, поэтому и производительность оценивается как производительность вычислений в ядре:

$$Perf_{Tlin} = Perf_{Kern} \approx 1745 \text{ Мфлоп/с}.$$

Если же параметры процессора таковы, что $T_{ldst} > T_{exec}$, то время вычислений оценивается как T_{ldst}, откуда можно найти и производительность.

4.3. Производительность базовой операции на нескольких процессорах.

Рассмотрим случай, когда вычисления производятся на не менее чем 8 процессорах. Тогда разбиение входного массива, согласно выбранной схеме, производится по всем трём направлениям. Соответственно, после вычисления Tlin на каждом процессоре К128 для соответствующего блока требуется произвести обмен граничными плоскостями с соседними процессорами. В случае, когда общее количество процессоров менее 8, некоторые из обменов каждый процессор осуществляет с самим собой; эти обмены можно заменить на обмен граничных плоскостей через локальную память СР2, посредством DMA. Такой обмен будет происходить быстрее, чем при использовании MPI, поскольку не требует дополнительных копирований, и кроме того, производительность DMA выше, чем производительность RapidIO. Соответственно, общая производительность процедуры несколько увеличится.

Теоретически, архитектура К128 позволяет совмещать вычисления на СР2 с передачами данных по RapidIO. Однако, выбранная схема вычислений для алгоритма MG требует, чтобы обработка всех блоков на СР2 была завершена до того, как начинается обмен границами. Это решение, помимо простоты реализации, связано отчасти и с тем, что при одновременной работе RapidIO и DMA скорость передач данных по DMA, а значит и вычислений на СР2 в целом, снижается.

Вычислим общий объём пересылаемых по RapidIO данных для каждого процессора. Применяется следующая схема обменов: сначала пересылаются вычисленные границы по направлению x (т.е. параллельные плоскости Oyz) – массивы порядка $N_2 \times N_3$; затем границы по y – массивы порядка $(N_1+2) \times N_3$, включающие два ребра, полученных при обмене по x; наконец, границы по z со всеми уже полученными рёбрами – массивы порядка $(N_1+2) \times (N_2+2)$. По каждому направлению границ две – процессор отправляет каждую из них и получает соответствующую дополнительную границу от соседнего процессора. Таким образом, общий объём пересылаемых данных составляет

$$4\,(N_2 N_3 + (N_1+2) N_3 + (N_1+2)(N_2+2)).$$

Для простоты формул, при достаточно большом размере блока, временем на пересылку дополнительных рёбер можно пренебречь.

При оценке времени на пересылку данных по RapidIO следует иметь в виду общее количество и взаимное расположение процессоров, одновременно осуществляющих обмены. Мы условно обозначим эту зависимость коэффициентом α, $0 \le \alpha \le 1$: будем считать фактическую скорость обменов равной αR. Выбор наилучшей возможной топологии соединения процессоров является, вообще говоря, отдельной непростой задачей. Во-первых, имеется ряд конструктивных ог-

раничений, таких как количество слотов для процессоров на одной плате. Во-вторых, сам кристалл процессора имеет фиксированное количество внешних выходов.

С точки зрения алгоритма MG, оптимальной топологией является трёхмерный тор: каждый процессор соединён с двумя соседними по каждому из трёх направлений. На рисунке (4) приводится двумерный тор для 16 узлов.

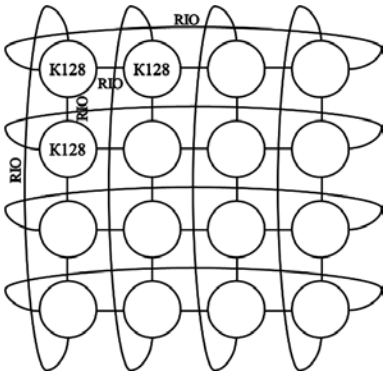


Рис. 4. Соединение процессоров в сеть: двумерный тор

Здесь каждый процессор имеет четыре внешних выхода. Для построения трёхмерного тора каждый процессор должен обладать не менее чем шестью внешними выходами. При такой топологии соединения, в ходе обменов границами при вычислении Tlin, по каждому из каналов в каждый момент времени будет передаваться не более одного сообщения. Другими словами, $\alpha = 1$, пиковая скорость пересылки равна R и не зависит от количества процессоров.

В настоящее время, однако, K128 имеет лишь один выход, т.е. по одному каналу на приём и передачу. В следующей версии процессора планируется 4 выхода. Кроме того, существующие платы имеют по 5 слотов для процессоров. Все эти факторы затрудняют формализацию оценки скорости обменов данными, поэтому для каждого количества процессоров предлагается определять оптимальную топологию и, соответственно, значение α, экспериментально.

С учётом сказанного, получаем оценку времени на пересылку данных по RapidIO:

$$T_{RIO} = \frac{4(N_2N_3 + N_1N_3 + N_1N_2)}{\alpha R}. \quad (7)$$

Для отправки MPI-сообщений пересылаемые данные должны быть расположены в непрерывных областях памяти. Поскольку граничные плоскости в трёхмерном массиве представляют собой данные с шагом, перед отправкой требуется произвести копирование. После обменов полученные данные необходимо скопировать обратно в трёхмерный массив. Время на все копирования, осуществляемые при обменах, оценивается по формуле

$$T_{copy} = \frac{4(N_2N_3 + N_1N_3 + N_1N_2)}{bw},$$

где bw – пиковая скорость копирования данных в DDR.

Общее время на MPI обмены составляет

$$T_{MPI} = 4(N_2N_3 + N_1N_3 + N_1N_2)\left(\frac{1}{\alpha R} + \frac{1}{bw}\right). \quad (8)$$

Для приведённых значений параметров это время равно

$$T_{MPI} = 4(N_2N_3 + N_1N_3 + N_1N_2)\cdot$$
$$\cdot 4\text{байт}\cdot\left(\frac{1}{840}\text{Мбайт/с} + \frac{1}{90}\text{Мбайт/с}\right)\approx$$
$$\approx 0{,}206(N_2N_3 + N_1N_3 + N_1N_2)\text{мкс}.$$

Мы получили оценки для времени, которое каждый из np процессоров тратит на само вычисление Tlin и на обмены с соседними процессорами. Вычисления выполняются на всех процессорах параллельно. Если в задаче входной массив имеет порядок $N\times N\times N$, и мы обрабатываем его на np процессорах, то $N_1N_2N_3 = N^3/np$. Для выбранной схемы разбиения массива можно условно считать, что $N_1 = N_2 = N_3 = N/\sqrt[3]{np}$. Тогда общая формула для оценки времени вычисления одного Tlin на всех процессорах, включая время на обмены, выглядит следующим образом:

$$\begin{aligned} T_{Tlin} &= \max\{T_{exec}, T_{ldst}\} + T_{MPI} = \\ &= \max\left\{\frac{55N_1N_2N_3}{4KF}, \frac{3N_1N_2N_3}{BW}\right\} + \\ &+ 4(N_2N_3 + N_1N_3 + N_1N_2)\left(\frac{1}{\alpha R} + \frac{1}{bw}\right) = \\ &= \frac{N^3}{np}\max\left\{\frac{55}{4KF}, \frac{3}{BW}\right\} + \frac{12N^2}{\sqrt[3]{N^2}}\left(\frac{1}{\alpha R} + \frac{1}{bw}\right). \end{aligned} \quad (9)$$

Здесь F – рабочая частота, K – количество секций математического сопроцессора, BW – пиковая скорость обменов данными между DDR и LMEM через DMA. N – размер входного массива, зависящий от класса задачи.

Зная T_{Tlin}, можно найти теоретическую оценку производительности для базовой операции, а значит и для алгоритма в целом:

$$Perf_{MG} \le \frac{30N^3}{T_{Tlin}}. \quad (10)$$

Рассмотрим в качестве примера задачу класса A и 8 процессоров K128, с приведёнными значениями параметров. Будем считать, что $\alpha = 1$. Подставляя $N=256$, $np=8$ и полученные выше T_{exec}, T_{ldst} и T_{MPI}, находим:

$$T_{Tlin} \approx 1{,}719 \cdot 10^{-2} \cdot \frac{256^3}{8}\text{мкс} + 0{,}206 \cdot \frac{256^2}{4}\text{мкс} \approx$$
$$\approx 36{,}05\text{мс} + 3{,}37\text{мс} = 39{,}42\text{мс}.$$

Соответственно,

$$Perf_{MG} \le \frac{30 \cdot 256^3 \text{флоп}}{39{,}42 \cdot 10^{-3}\text{с}} = 12768\text{Мфлоп/с}.$$

4.4. Производительность алгоритма в целом.

В предыдущих разделах производительность алгоритма MG оценивалась как производительность базовой операции Tlin – вычисления невязки или сглаживания на самой мелкой сетке. Следует иметь в виду, что полученная оценка является лишь ориентировочной оценкой сверху, поскольку на производительности отрицательно сказывается ряд факторов:

- алгоритм включает вычисление и других трилинейных операторов, в том числе на более мелких сетках, с меньшей производительностью, а также вычисление нормы невязки;
- оценки сделаны в предположении, что во всех случаях данные пересылаются с пиковой скоростью. В реальности, особенно при пересылке данных с шагом, скорость может быть ниже;
- любая реализация включает ряд накладных расходов на организацию вычислений (например, формирование дескрипторов для пересылки данных по DMA). Время, затрачиваемое на эти операции, существенно зависит от качественных характеристик управляющего процессора;
- немалую роль играет качество самого кода – как ядер на ассемблере CP2, так и управляющей процедуры.

В реальности можно ожидать производительность порядка 60-70% от полученного теоретического максимума. Производительности выше, чем этот максимум, добиться заведомо не удастся. В ходе реализации оценку можно уточнять, замеряя производительность ядра, пересылки данных по DMA, копирования в DDR и передачи по RapidIO и подставляя полученные значения в формулу (9). Однако предварительные оценки позволяют ещё до начала работ сделать ряд выводов о влиянии той или иной характеристики процессора на производительность выбранного алгоритма.

В частности, характеристики последней существующей в настоящий момент версии микропроцессора К128 таковы, что для алгоритма MG:

1) Производительность вычислений на каждом процессоре определяется производительностью ядер – трилинейных операторов на CP2. DMA достаточно производителен, чтобы обеспечить загрузку и выгрузку из LMEM данных, необходимых при этих вычислениях.

2) При реализации на нескольких процессорах время на MPI-обмены граничными данными между процессорами составит в идеальном случае порядка 10% от времени вычислений. Для того чтобы обмены не сказались критически на общей производительности вычислений, важен тщательный выбор топологии соединения процессоров.

3) Кроме того, при организации MPI-обменов на вспомогательные копирования средствами управляющего процессора тратится существенно большее время, чем на саму передачу данных через RapidIO. Поэтому для достижения большей производительности необходимо разработать альтернативный способ копирования разреженных данных в непрерывную область – например, посредством DMA и вспомогательной процедуры на CP2.

5. Предварительная реализация алгоритма

Автором разработана предварительная реализация алгоритма MG для многопроцессорного комплекса на основе К128.

Разработан набор вычислительных ядер для CP2 и соответствующих процедур верхнего уровня для процессора К128. При их реализации использовалась инфраструктура библиотеки цифровой обработки сигналов (dsplib).

Реализация алгоритма в целом основана на референсном коде NPB на языке FORTRAN. Фрагменты этого кода заменены вызовами соответствующих процедур для К128.

В целом все вычисления строятся по описанной выше схеме, однако в ходе работы над процедурами программист сталкивается с рядом технических и архитектурных ограничений, о которых пойдёт речь ниже.

5.1. Формат хранения данных.

В каждой процедуре Tlin входные и выходные данные представляют собой трёхмерный массив 32-разрядных вещественных чисел, размера $(2^{n_1}+2)\times(2^{n_2}+2)\times(2^{n_3}+2)$, включающий дополнительные граничные плоскости. Для вычислений на CP2 данный формат неудобен по двум причинам:

1) На CP2 арифметические операции выполняются над 64-разрядными словами, и такое же слово является единицей адресации в LMEM. 64-разрядное слово может рассматриваться как два вещественных числа. Однако при вычислении трилинейного оператора на каждый выходной элемент требуется 27 входных, соседних в трёхмерном массиве, причём все они обрабатываются по-разному. Поэтому использование пар вещественных чисел потребовало бы одновременного вычисления как минимум двух выходных элементов и дополнительной перегруппировки половин слов в регистрах, после чтения из LMEM и перед записью.
2) Контроллер DMA пересылает данные 128-разрядными словами, т.е. по 4 вещественных числа. Имеется возможность маскирования записи половины слов в секции LMEM, однако в настоящее время в библиотеке dsplib она не поддерживается. Это требует написания отдельного кода для обработки небольших массивов, для которых $n_1 = 1$. Кроме того, при пересылке адрес соответствующей области в памяти DDR, а также шаг и длина каждого непрерывного фрагмента должны быть

кратны 16 байтам, т.е. тем же 4 вещественным числам. При вычислении Tlin требуется загружать в LMEM блок с дополнительными граничными плоскостями, а выгружать – без дополнительных плоскостей, т.е. адреса различаются на 1 элемент по каждому направлению, однако в обоих случаях должно соблюдаться выравнивание.

Исходя из этих соображений, для упрощения реализации было принято решение временно использовать другой, более удобный формат данных. Во-первых, данные разреживаются нулями, и таким образом, каждый элемент массива становится 64-разрядным словом – парой (вещественное число, 0). Это облегчает реализацию ядра на CP2, и кроме того, требуется отдельное рассмотрение только случая $n_1=0$, т.е. массива из 1 элемента.

Во-вторых, массив дополняется ещё двумя нулевыми граничными плоскостями по x, по одной с каждой стороны. При обработке каждый блок загружается в LMEM также с двумя лишними плоскостями, в которых могут содержаться нули или элементы соседнего блока, но в ходе вычислений эти плоскости не используются. Поскольку элементы массива в нашем формате 64-разрядные, мы добиваемся нужного выравнивания как при загрузке, так и при выгрузке.

На рисунке (5) приводится массив (для наглядности – двумерный) размера 4×4 и его разбиение по x на два блока, в исходном формате и в новом. Белые клетки соответствуют «внутренним», т.е. основным элементам массива, серые – дополнительным граничным плоскостям. В клетках указаны индексы основных элементов массива, пустые клетки соответствуют нулям.

Отметим, что описанное решение является временным, и его использование в реальных расчётах невыгодно, так как влечёт потерю 50% эффективности как вычислений, так и пересылок по DMA.

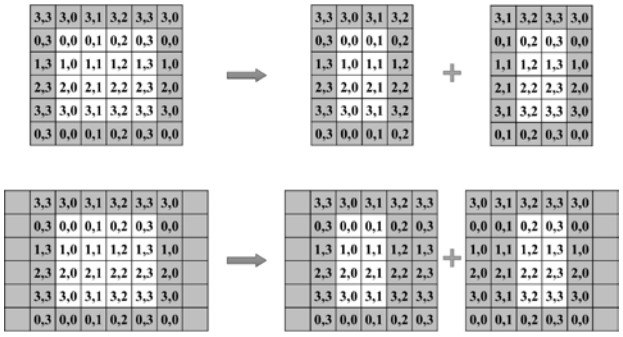


Рис. 5. Формат хранения данных.

5.2. Вычислительные ядра для CP2.

Всего для алгоритма MG разработано 7 вычислительных ядер:

1) resid (невязка): $r = v - Au$;
2) proj (проекция): $v = Pu$;
3) interp (интерполяция): $v = v + Qu$;
4) interp0 (интерполяция без накопления): $v = Qu$;
5) smooth (сглаживание): $w = v + Su$;
6) smooth0 (сглаживание без накопления): $w = Su$;
7) norm (норма): $res = \|u\|_2$.

Первые 6 ядер – это различные трилинейные операторы. На вход каждому из них подаётся трёхмерный блок (в каждой секции LMEM) с дополнительными граничными плоскостями: по одной с каждой стороны по направлениям y и z, по две по направлению x (для счёта используется только по одной из них). На выходе получается блок без дополнительных плоскостей. В ядрах resid, interp и smooth второй входной массив, v, также не имеет дополнительных плоскостей.

Массивы из одного элемента, для которых $N_1=N_2=N_3=1$ (либо 2 для proj), обрабатываются на управляющем процессоре, в коде на языке C, без участия CP2.

Расчёты в разделе 4.1 показали, что в ядрах Tlin арифметические операции должны преобладать над операциями загрузки/выгрузки. Однако в текущей версии ядер используются команды загрузки и выгрузки одного слова (lw/sw), а не пары слов (ld/sd), как более удобные для работы с «кубиками» из 3×3×3 точек. Поэтому операции обменов с LMEM преобладают в основном цикле.

Вычисления в ядре, например, resid, организованы в виде трёх вложенных циклов:

- внешний цикл – по слоям блока (плоскостям, параллельным Oxy),
- средний цикл – по рядам слоя (параллельным Ox),
- внутренний цикл – по парам выходных элементов.

Ядро можно было бы сделать более плотным, не разбивая элементы одного ряда на пары. Но для параллельной обработки большего количества точек не хватает плавающих регистров (FPR): для вычисления одного выходного элемента требуется как минимум 28 регистров, а всего регистров 64. Частично нехватка регистров связана с выбранным форматом данных: если бы в каждом регистре обрабатывалась пара элементов, всего регистров потребовалось бы вдвое меньше. В текущей реализации ядра resid на 163 инструкции в основном цикле приходится 108 арифметических операций, т.е. эффективность вычислений в ядре составляет 66%. Остальные ядра Tlin устроены аналогично.

В ядре norm вычисляется только сумма квадратов элементов входного блока. Деление результата на размер массива и взятие квадратного корня осуществляется на CPU. Первоначально сумма квадратов накапливалась в одном регистре в каждой секции, причём значение сохранялось при последовательных вызовах ядра – таким образом накапливалась сумма для всего массива, обрабатываемого на K128. Тестирование показало, однако, что вычисление суммы по такой простой схеме даёт слишком большую погрешность. Поэтому в текущей версии суммирование осуществляется по схеме двоичного дерева: вычисляются суммы квадратов пар элементов, затем попарно суммируются полученные пары, четвёрки и т.д.. Суммы для блоков, на которые делится массив, накапливаются во временном буфере. После обработки всех блоков массива результаты снова суммируются по той же схеме. Результаты

из четырёх секций CP2 собираются в одной секции посредством регистров GPR, и производится окончательное суммирование. Данное решение требует разбиения работы на несколько этапов и целого набора ядер для различных случаев, но даёт хорошую погрешность вычислений.

5.3. Пересылки данных по DMA.

С каждым ядром Tlin связана процедура верхнего уровня для K128, вычисляющая соответствующую функцию для массива в DDR и дающая на выходе аналогичный массив в DDR. Все массивы – и входные, и выходные – содержат дополнительные граничные плоскости, но на CP2 вычисляются только внутренние части массивов. Процедура вычисления нормы получает на вход массив и даёт на выходе одно число.

Небольшие массивы обрабатываются в половине LMEM одной секции, целиком. Если для обработки массива половины памяти секции недостаточно, он разбивается на блоки, которые обрабатываются постранично, в соответствии со схемой, описанной в разделе 3.3.

Контроллер DMA имеет так называемую «2,5-мерную» архитектуру: аппаратно поддерживается пересылка 2-мерных массивов, и ещё одно измерение в ограниченном объёме. В рамках библиотеки dsplib реализованы процедуры, за счёт программных средств позволяющие переслать трёхмерный массив из DDR в память одной из секций CP2, или обратно. Эти процедуры и используются для загрузки/выгрузки страниц блоков: загрузка блоков в четыре секции осуществляется последовательно, и аналогично для выгрузки.

5.4. Обмены граничными данными.

Помимо процедур, вычисляющих Tlin и норму, для алгоритма MG могут потребоваться также процедуры, производящие обмен граничных плоскостей для массива в DDR: каждая вычисленная граничная плоскость (грань массива порядка $N_1 \times N_2 \times N_3$) копируется в противоположную дополнительную граничную плоскость (с учётом дополнительных рёбер, как описано в разделе 4.3). Всего таких процедур три: для обмена граничных плоскостей, соответственно, по *x, y* и *z*. Копирование плоскостей по трём направлениям проиллюстрировано на рисунке (6). Обмен производится посредством DMA: каждая плоскость загружается в LMEM, а затем выгружается на новое место в массиве в DDR.

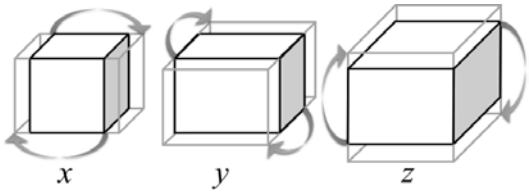


Рис. 6. Копирование границ на одном процессоре.

Граничные плоскости по z представляют собой непрерывные области в памяти. Граничные плоскости по *y* – двумерные массивы с шагом между началами строк. Пересылка их по DMA не составляет труда. Что касается граничных плоскостей по *x*, они состоят из отдельных элементов с шагом. Как говорилось выше, минимальная непрерывная область памяти, которую можно переслать по DMA – 128 бит, т.е. два элемента. Поэтому процедура устроена таким образом, что фактически обмениваются пары плоскостей: плоскости, соответствующие $x=2$ и $x=3$, копируются в плоскости $x=N_1+2$ и $x=N_1+3$, а плоскости, соответствующие $x=N_1$ и $x=N_1+1$ – в плоскости $x=0$ и $x=1$. В результате «фиктивные» плоскости $x=0$ и $x=N_1+3$ содержат мусорные данные, но нигде в вычислениях они не используются, поэтому не влияют на результат.

5.5. Организация процедуры MG в целом.

Реализация алгоритма MG представляет собой синтез разработанных автором процедур для K128 и референсного кода NPB на языке FORTRAN, предварительно приведённого к вычислениям с одинарной точностью: процедуры генерации данных, замера производительности и проверки результата в референсном коде остаются без изменений, а вся вычислительная часть алгоритма заменяется на вызов разработанной процедуры на C. Кроме того, добавляются вызовы процедур инициализации и завершения вычислений на K128.

Процедура инициализации, в числе прочего, приводит входные данные к формату, описанному в разделе 5.1. Это копирование осуществляется медленно, однако выполняется всего один раз перед началом вычислений и не включается в замеры времени, поэтому на производительность вычислений не влияет. Основная процедура на C организует вызов описанных выше процедур вычисления Tlin и нормы, а также MPI-обменов, в нужной последовательности.

Процедуры обмена границами через MPI были портированы на C из референсного кода и в сущности осуществляют те же обмены, однако работают с данными в выбранном специальном формате. Обмены парами плоскостей по *x, y* и *z* осуществляются отдельно: для каждой пары, в зависимости от общего количества процессоров, вызывается либо процедура копирования через DMA (см. раздел 5.4.), либо «настоящая» процедура обмена через MPI. Каждая процедура обмена через MPI выполняет следующие действия, одинаковые для всех процессоров:

1) Граничные плоскости (для *y* и *z* – с дополнительными рёбрами) для пересылки копируются из трёхмерного массива в два непрерывных буфера. Дополнительные нули при этом не копируются.
2) Два сформированных буфера пересылаются соседним процессорам, а два новых буфера принимаются – на это требуется по 2 вызова процедур MPI_Send, MPI_Irecv и MPI_Wait.
3) Данные из принятых буферов раскладываются в трёхмерные массивы, в дополнительные граничные плоскости.

В настоящее время копирования на шаге 1 и 3 осуществляются средствами центрального процессора. Как показано в разделе 4.3, такое копирование неэффективно и требует больше времени, чем сами обмены, поэтому данное решение является временным и должно быть впоследствии заменено на копирование посредством DMA и дополнительного ядра на CP2.

Наша реализация рассчитана на вычисления на 1, 2, 4 или 8 процессорах, каждому процессору соответствует 1 MPI-процесс. Ограничение сверху связано с тем, что на нижнем уровне алгоритма MG обрабатывается массив размера 2×2×2 – в случае 8 процессов на каждый процесс приходится по одному элементу. Реализация может быть расширена на большее количество процессов: для этого можно, например, на нижних уровнях многосеточного метода собрать все данные на одном процессоре и произвести необходимые вычисления на нём, а затем снова распределить данные по всем процессорам.

В конце вычислений сумма квадратов элементов невязки полученного решения вычисляется на каждом процессоре отдельно, для соответствующего блока. Общая сумма собирается на одном из процессоров при помощи вызова процедуры MPI_Gather, после чего находится значение нормы.

6. Результаты замеров

В ходе работы над алгоритмом разработанные процедуры Tlin были протестированы на реальном процессоре К128. Данные процедуры работают на одном процессоре и включают в себя постраничное вычисление на CP2 и копирование границ, как описано в разделе 5.4. Входные и выходные данные располагаются в памяти DDR, обмены данными по MPI отсутствуют. По производительности этих процедур можно судить о качестве написанных ядер для CP2 и об ожидаемой производительности всего алгоритма MG при работе на одном процессоре. Также была протестирована процедура вычисления нормы, однако её производительность в меньшей степени сказывается на производительности всего алгоритма, т.к. данная процедура вызывается всего один раз.

В таблице 1 приводятся результаты замеров производительности для всех семи процедур, перечисленных в разделе 5.1. Рассматривается массив максимального размера, соответствующего задачам классов A и B: 256×256×256 (для процедур интерполяции – 128×128×128). Частота работы процессора – 200 МГц.

Таблица 1. Производительность Tlin и нормы на К128.

Процедура	Производительность (Мфлоп/с)
resid	620
smooth	620
smooth0	635
proj	308
interp	318
interp0	492
norm	472

В разделе 4.2 мы получили оценку производительности для процедуры resid на одном процессоре: 1745 Мфлоп/с. Таким образом, эффективность предварительной разработанной процедуры составляет 36% от теоретического максимума.

Время на копирование границ, которое учтено в замерах и не учитывалось в разделе 4.2, не значительно по сравнению со временем вычислений, как следует из таблицы 2. В данной таблице приводится время на выполнение 100 итераций каждой процедуры: суммарное, а также время только на вычисления на CP2, только на обмены через DMA (которые по возможности совмещаются с вычислениями) и время на копирование границ. В скобках для каждой величины приводится доля от общего времени, в процентах. Рассматриваются те же размеры массивов, что и в таблице 1, и частота процессора 200 МГц.

Таблица 2. Время на процедуры Tlin и нормы на К128.

Проц-ра	Общее время (с)	CP2	DMA	Обмен границ
resid	81,15728	67,80572 (84%)	33,21575 (41%)	0,65814 (1%)
smooth	81,13797	67,71053 (84%)	33,12527 (41%)	0,65711 (1%)
smooth0	76,65288	67,63675 (88%)	24,3359 (32%)	0,65838 (1%)
proj	19,72275	9,8688 (50%)	19,03402 (97%)	0,1667 (1%)
interp	19,81385	7,66731 (39%)	18,91858 (96%)	0,65502 (3%)
interp0	12,3527	7,61186 (62%)	10,82938 (88%)	0,65531 (5%)
norm	7,11625	3,47934 (49%)	6,95383 (98%)	-

Мы видим, что, как и предсказывали теоретические оценки, для процедуры вычисления невязки, а также сглаживания, которые занимают наибольшее время в алгоритме MG, время на вычисления в ядре более чем вдвое превышает время на пересылки по DMA. При этом фактическое совмещение пересылок с вычислениями составляет порядка 60% от времени пересылок. Время на копирование границ через DMA составляет менее 1% от общего времени вычислений.

Процедура MG в целом была протестирована на эмуляторе К128, на универсальном процессоре Intel Xeon E5-2670, для 1, 2, 4 и 8 MPI-процессов. На том же процессоре для сравнения был протестирован и референсный код, в котором вычисления с двойной точностью были заменены на вычисления с одинарной точностью. В таблице 3 приводится погрешность результат (нормы невязки), для референсного кода и для процедуры на эмуляторе, а также объём памяти DDR, требуемый для вычислений на К128.

Из таблицы видно, в частности, что разработанные процедуры на К128 дают в среднем лучшую точность вычислений, чем референсный код.

Таблица 3. Тестирование процедуры MG на эмуляторе.

Класс задачи	Погрешность (реф.)	Погрешность (эмул.)	Память на процесс (Мбайт)
1 MPI-процесс			
S	$0{,}48 \cdot 10^{-5}$	$0{,}12 \cdot 10^{-5}$	1,102
W	$0{,}86 \cdot 10^{-3}$	$0{,}27 \cdot 10^{-5}$	56,724
A	$1{,}00 \cdot 10^{-3}$	$0{,}18 \cdot 10^{-5}$	436,958
B	$0{,}17 \cdot 10^{-2}$	$0{,}46 \cdot 10^{-5}$	436,958

2 MPI-процесса			
S	$0{,}48 \cdot 10^{-6}$	$0{,}12 \cdot 10^{-5}$	0,597
W	$0{,}57 \cdot 10^{-3}$	$0{,}27 \cdot 10^{-5}$	28,98
A	$0{,}61 \cdot 10^{-3}$	$0{,}18 \cdot 10^{-5}$	220,882
B	$0{,}11 \cdot 10^{-2}$	$0{,}46 \cdot 10^{-5}$	220,882
4 MPI-процесса			
S	$0{,}20 \cdot 10^{-5}$	$0{,}12 \cdot 10^{-5}$	0,32
W	$0{,}36 \cdot 10^{-3}$	$0{,}27 \cdot 10^{-5}$	14,758
A	$0{,}45 \cdot 10^{-3}$	$0{,}17 \cdot 10^{-5}$	111,466
B	$0{,}84 \cdot 10^{-3}$	$0{,}46 \cdot 10^{-5}$	111,466
8 MPI-процессов			
S	$0{,}18 \cdot 10^{-5}$	$0{,}37 \cdot 10^{-2}$	0,181
W	$0{,}23 \cdot 10^{-3}$	$0{,}29 \cdot 10^{-4}$	7,638
A	$0{,}28 \cdot 10^{-3}$	$0{,}10 \cdot 10^{-5}$	56,724
B	$0{,}53 \cdot 10^{-3}$	$0{,}18 \cdot 10^{-4}$	56,724

Замеры, проведённые для референсной реализации MG на процессоре Intel Xeon E5-2670, показали, что эффективность вычислений на одном ядре процессора, для задачи класса B, составляет 28% от пика. При этом 74% времени работы занимает выполнение базовых операций. Зная это и производительность базовой операции на К128, из таблицы 1 (620 Мфлоп/с), можно сделать вывод, что ожидаемая производительность всего алгоритма MG на одном процессоре К128, в текущей реализации, составит порядка 450 Мфлоп/с.

В дальнейшем планируется тестирование процедур Tlin и алгоритма в целом на прототипе будущего многопроцессорного комплекса на базе К128.

7. Выводы

7.1. Характеристики алгоритма MG.

Проведённые теоретические оценки показали, что архитектура К128 в целом позволяет эффективную реализацию алгоритма MG и подобных ему алгоритмов.

Данный алгоритм является нагруженным с точки зрения вычислений: основное время работы занимают непосредственно вычисления в ядре на арифметическом сопроцессоре. Пересылки данных по DMA могут быть эффективно скрыты за этими вычислениями.

«Бутылочным горлышком» алгоритма могут оказаться обмены данными между процессорами через RapidIO, поэтому особенно важными становятся задачи оптимального выбора топологии соединения процессоров и эффективного портирования процедур MPI на RapidIO.

В настоящий момент теоретическая оценка эффективности алгоритма – 22% от пиковой производительности многопроцессорного комплекса.

7.2. Дальнейшая работа над реализацией.

В данной работе была описана предварительная разработанная процедура многосеточного метода. Эта процедура реализует выбранную схему вычислений, включая совмещение вычислений на СР2 с пересылками по DMA, и показывает хорошую погрешность вычислений. Однако эффективность вычислений составляет лишь 36% от теоретического максимума. Для повышения эффективности вычислений на существующей в настоящий момент аппаратуре можно предпринять следующие шаги:

1. Избавиться от избыточных нулей в обрабатываемых массивах. Для этого потребуется усложнить ядро и реализовать набор частных случаев для пересылки данных по DMA. Ожидаемый прирост производительности от данной оптимизации – до 2 раз.
2. Заменить копирование разреженных данных в непрерывную область при помощи центрального процессора на копирование при помощи DMA. Для этого требуется вспомогательное ядро для СР2. Ожидаемый прирост производительности – до 8%.

Описанные оптимизации позволят добиться для базовой операции эффективности вычислений порядка 70% от теоретического максимума.

Производительность всего алгоритма на одном процессоре К128, на частоте 200 МГц, составит в этом случае не менее 900 Мфлоп/с, т.е. 15% от пиковой производительности на операции «умножение с накоплением». Данный результат сопоставим по порядку с эффективностью процедуры MG на одном ядре универсального процессора Intel Xeon – 28 %,

7.3. Предложения к архитектуре перспективного вычислительного комплекса.

Возможности оптимизации вычислений ограничены архитектурными особенностями процессора. Из полученных в разделе 4 формул можно сделать ряд выводов о значениях параметров процессора, необходимых для эффективных вычислений. Основные выводы:

1. Чтобы производительность процедуры MG на К128 определялась производительностью ядра на СР2, для канала доступа к памяти (DMA) должно выполняться следующее ограничение снизу (здесь BW – пропускная способность DMA, K – количество секций сопроцессора, F – рабочая частота):

$$BW > \frac{12KF}{55}.$$

2. Чтобы при обработке массива размера $N \times N \times N$ на np процессорах время на обмены данными между процессорами составляло не более γ% от времени вычислений, должно выполняться следующее соотношение (здесь R – пропускная способность RapidIO, α – коэффициент, отражающий влияние топологии соединения процессоров в комплексе)

$$\frac{1}{\alpha R} < \gamma \cdot \frac{55N}{48KF \cdot \sqrt[3]{np}}.$$

Теоретический максимум эффективности, при любых значениях параметров, можно дополнительно повысить, внеся ряд усовершенствований в архитектуру процессора и вычислительного комплекса в целом.

Во-первых, чтобы избежать избыточности при пересылке данных по DMA, требуется ослабить требование к выравниванию данных: необходима возможность пересылки данных 32-разрядными словами, причём адрес в DDR должен быть выравнен только на 4 байта.

Во-вторых, для эффективных обменов данными между процессорами необходима возможность соединения процессоров в трёхмерный тор – соответственно, каждый процессор должен обладать как минимум 6 внешними выходами.

В-третьих, увеличение объёма накристальной памяти сопроцессора (LMEM) позволит обрабатывать бóльшие блоки, благодаря чему повысится эффективность вычислений для больших классов задач.

Наконец, чтобы повысить точность вычислений, и в частности, удовлетворить требованиям NBP, необходима поддержка вычислений на CP2 с двойной точностью.

Выражаю благодарность Богданову Павлу Борисовичу за техническую поддержку и ценные обсуждения в ходе работы над реализацией алгоритма.

The NPB MG algorithm implementation on a multi-processor computing system based on the KOMDIV128-RIO microprocessor

O.J. Sudareva

Abstract: In this paper we consider the multigrid (MG) algorithm of the NAS Parallel Benchmarks and its implementation on a multi-processor computing system based on Russian hardware components. We offer an implementation scheme for this algorithm and deduce the correlation between computation performance and a number of computing system characteristics. We also describe out initial algorithm implementation and show the results of some performance tests.

Keywords: Multi-Grid, MG, NAS Parallel Benchmarks, NPB, KOMDIV128-RIO, K128

Литература

1. Микросхема интегральная 1890ВМ7Я (КОМДИВ128-РИО), указания по применению. // Москва, НИИСИ РАН, 2009.
2. MPI: A message-passing interface standard, version 3.0. // Message Passing Interface Forum, 2012. URL: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
3. Поддержка интерфейса MPI в ядре ОС Linux для многопроцессорных модулей на базе высокоскоростных каналов RapidIO. // Москва: НИИСИ РАН, 2015
4. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon,V.Venkatakrishnan and S.Weeratunga. The NAS Parallel Benchmarks // RNR Technical Report RNR-94-007, March 1994.
5. NAS Parallel Benchmarks // URL: https://www.nas.nasa.gov/publications/npb.html
6. О.Ю. Сударева. Эффективная реализация алгоритмов быстрого преобразования Фурье и свёртки на микропроцессоре КОМДИВ128-РИО. – Москва: НИИСИ РАН, 2014.

Задача оптимального управления для квазилинейных дифференциальных уравнений с нелокальными краевыми условиями

Г.В. Меладзе, Д. Ш. Девадзе, В. Ш. Беридзе, М.Ш. Сургуладзе, П.И. Кандалов

Аннотация. В данной работе рассматривается нелокальная краевая задача Бицадзе-Самарского для квазилинейных дифференциальных уравнений первого порядка на плоскости. Доказывается теорема о существования и единственности обобщенного решения в пространстве $C_\alpha(\overline{G})$; Для линейной краевой задачи доказывается существования решения в пространстве $C^p_\alpha(\overline{G})$ и получена априорная оценка; Поставлена задача оптимального управления для квазилинейных дифференциальных уравнений первого порядка с интегральным критерием качества. Получены необходимые условия оптимальности в форме принципа максимума Понтрягина; Для линейной задачи оптимального управления доказывается теорема о необходимом и достаточном условии оптимальности. Рассматривается задача оптимального управления для уравнений Гельмогольца с краевыми условиями Бицадзе-Самарского. Приведена теорема о необходимом и достаточном условии оптимальности. Представлен алгоритм решения задачи оптимального управления средствами пакета Mathcad.

Ключевые слова: нелокальная краевая задача, задача Бицадзе-Самарского, обобщенное решение, оптимальное управление, условия оптимальности, принцип максимума.

.

1. Существование обобщенного решения краевой задачи Бицадзе-Самарского для квазилинейных дифференциальных уравнений первого порядка на плоскости

Пусть G ограниченная область комплексной плоскости E с границей Γ, которая является замкнутой простой кривой Ляпунова (т.е. угол, которую составляет с постоянным направлением касательная к этой кривой, непрерывен в смысле Гельдера).

Обозначим через γ часть границы Γ, представляющую собой разомкнутую кривую Ляпунова с параметрическим уравнением $z = z(s)$, $0 \le s \le \delta$. Пусть γ_0 диффеоморфный $z_0 = I(z)$ образ γ, лежащий в области G, с параметрическим уравнением $z_0 = z_0(s)$, $0 \le s \le \delta$. Предположим, что γ_0 пересекается с Γ, но не касательно к ней, $z = x + iy \in G$, $w = w_1 + iw_2$. $\partial_{\overline{z}} = \frac{1}{2}\left(\frac{\partial}{\partial x} + i\frac{\partial}{\partial y}\right)$ - обобщенная производная Соболева [1, 2]. $C(\overline{G})$ - банахово пространство, состоящие из всех непрерывных на $\overline{G}$ функций. Норма в $C(\overline{G})$ определяется равенством $\|f\|_{C(\overline{G})} = \underset{z\in\overline{G}}{\mathrm{Max}}|f(z)|$. $C_\alpha(\overline{G})$ - множество всех ограниченных функций удовлетворяющие условию Гельдера с показателем α. Норма в $C_\alpha(\overline{G})$ определяется равенством $\|f\|_{C_\alpha(\overline{G})} = \underset{z\in\overline{G}}{\mathrm{Max}}|f(z)| + \underset{z_1,z_2\in\overline{G}}{\mathrm{Sup}}\frac{|f(z_1)-f(z_2)|}{|z_1-z_2|^\alpha}$. $L_p(\overline{G})$ - банахово пространство, состоящие из всех измеримых на $\overline{G}$ функций, суммируемых по $\overline{G}$ со степенью $p \ge 1$. Норма в $L_p(\overline{G})$ определяется равенством $\|f\|_{L_p(\overline{G})} = \left(\int_{\overline{G}} |f|^p\, dz\right)^{1/p}$.

В области $\overline{G}$ рассмотрим краевую задачу Бицадзе-Самарского [3] для квазилинейных дифференциальных уравнений первого порядка:

$$\partial_{\overline{z}} w = f(z, w, \overline{w}), \quad z \in G, \tag{1.1}$$

$$\begin{aligned} &\mathrm{Re}[w(z)] = \varphi(z), \quad z \in \Gamma \setminus \gamma, \\ &\mathrm{Im}[w(z^*)] = c, \quad z^* \in \Gamma \setminus \gamma, \end{aligned} \tag{1.2}$$

$$\begin{aligned} &\mathrm{Re}[w(z(s))] = \sigma\,\mathrm{Re}[w(z_0(s))], \\ &z(s) \in \gamma, \ z_0(s) \in \gamma_0, \ 0 < \sigma = \mathrm{const}. \end{aligned} \tag{1.3}$$

Будем предполагать, что выполнены следующие условия:

(A1). Функция $f(z, w, \overline{w})$ определена при $z \in G$, $|w| < R$, $f(z, 0, 0) \in L_p(\overline{G})$, $p > 2$ и $|f(z, w, \overline{w}) - f(z, w_0, \overline{w}_0)| \le L(|w - w_0| + |w - \overline{w}_0|)$.

(A2). $\varphi(z) \in C_\alpha(\Gamma \setminus \gamma)$, $\alpha > 1/2$.

(A3). Существует такое число $R_1 > 0,\ R_1 \le R$, такое, что выполняются неравенство

$$\|\psi_n\|_{C_\alpha(\bar G)} + \left(C_1 + \|T_G\|_{L_p(\bar G), C_\alpha(\bar G)}\right)\left(2L\,|G|^{1/p}\ R_1\right) \le R_1,$$

где $|G| = \mathrm{mes}G$.

(A4). $2\,|G|^{1/p}\ L\left(C_1 + \|T_G\|_{L_p(\bar G), C_\alpha(\bar G)}\right) < 1,$

где $L, R, C_1 > 0$ - постоянные,

$$T_G[z, f] = -\frac{1}{\pi}\iint_G \frac{f(t)}{t - z}d\xi d\eta,\ t = \xi + i\eta\ [1, 4].$$

Справедлива следующая теорема:

Теорема1. Пусть выполнены условия (A1)-(A4), тогда решение задачи (1.1)-(1.3) существует в пространстве $C_\alpha(\bar G)$ и единственно.

Для доказательства существования решения задачи (1.1)-(1.3), рассматривается итерационный процесс:

$$\partial_{\bar z} w_n = f(z, w_n, \bar w_n),\quad z \in G, \tag{1.4}$$

$$\begin{aligned} &\mathrm{Re}[w_n(z)] = \varphi(z),\quad z \in \Gamma \setminus \gamma, \\ &\mathrm{Im}[w_n(z^*)] = c,\ z^* \in \Gamma \setminus \gamma, \end{aligned} \tag{1.5}$$

$$\begin{aligned} &\mathrm{Re}[w_n(z(s))] = \sigma\,\mathrm{Re}[w_{n-1}(z_0(s))], \\ &z(s) \in \gamma,\ z_0(s) \in \gamma_0, n = 1, 2, 3, \ldots \end{aligned}, \tag{1.6}$$

где $w_0(z)$ - любая функция из $C_\alpha(\gamma),\ \alpha > 1/2$, непрерывно примыкающая к значениям функции $\varphi(z)$ в концах контура γ.

2. Линейная задача

Рассмотрим в области $\bar G$ краевую задачу Бицадзе-Самарского для линейного дифференциального уравнения первого порядка

$$\begin{aligned} &\partial_{\bar z} w = A(z)w + B(z)\bar w + d(z),\quad z \in G, \\ &\mathrm{Re}[w(z)] = 0,\ z \in \Gamma \setminus \gamma, \\ &\mathrm{Im}[w(z^*)] = 0,\ z^* \in \Gamma \setminus \gamma, \\ &\mathrm{Re}[w(z(s))] = \sigma\,\mathrm{Re}[w(z_0(s))], \\ &z(s) \in \gamma,\ z_0(s) \in \gamma_0. \end{aligned} \tag{2.1}$$

Предположим, что $A(z),\ B(z),\ d(z) \in L_p(\bar G),\ p > 2,\ |A|, |B| \le N$.

Обозначим через $C^p_\alpha(\bar G)$ множество функций $w(z) \in C_\alpha(\bar G)$ таких, что на Γ выполняются условия (2.1) и обладают конечной нормой

$$\|w\|_{C^p_\alpha(\bar G)} = \|w\|_{C_\alpha(\bar G)} + \|\partial_{\bar z} w\|_{L_p(\bar G)} < +\infty. \tag{2.2}$$

Теорема 2. Для любой функции $d(z) \in L_p(\bar G),\ p > 2$, решение $w(z)$ задачи (2.1) существует, принадлежит пространству $C^p_\alpha(\bar G)$ и для него справедлива следующая априорная оценка:

$$\|w\|_{C^p_\alpha(\bar G)} \le \lambda\,\|d\|_{L_p(\bar G)}, \tag{2.3}$$

где λ - положительная постоянная, зависящая лишь от p, N и $|G| = \mathrm{mes}G$.

3. Постановка задачи оптимального управления

Пусть U - некоторое подмножество из E. Каждую функцию $u(z) : G \to U$ будем называть управлением. Функцию $u(z)$ назовем допустимым управлением, если $u(z) \in L_p(G),\ p > 2$. Множество всех допустимых управлений обозначим через Ω.

Для каждого фиксированного $u \in \Omega$ в области $\bar G$ рассмотрим следующую краевую задачу Бицадзе-Самарского

$$\begin{aligned} &\partial_{\bar z} w = f(z, w, \bar w, u),\ z \in G, \\ &\mathrm{Re}[w(z)] = \varphi_1(z),\ z \in \Gamma \setminus \gamma, \\ &\mathrm{Im}[w(z^*)] = c,\ z^* \in \Gamma \setminus \gamma,\ c = \mathrm{const}, \\ &\mathrm{Re}[w(z(s))] = \sigma\,\mathrm{Re}[w(z_0(s))], \\ &z(s) \in \gamma,\ z_0(s) \in \gamma_0,\ 0 < \sigma = \mathrm{const}. \end{aligned} \tag{3.1}$$

Будем предполагать, что выполнены условия **(A1)-(A4).** В этом случае для каждого фиксированного $u \in \Omega$ решение задачи (3.1) существует в пространстве $C_\mu(\bar G)$ и единственно (Теорема 1.).

Далее, для получения условия оптимальности дополнительно будем предполагать что:

(A5). Функция $f(z, p, q, u)$ непрерывна по $p,\ q,\ u$, имеет непрерывные частные производные f'_p и f'_q, которые также непрерывны по тем же аргументам, а результат суперпозиции $f(z, w(z), \bar w(z), u(z)) \in L_p(\bar G)$, $p > 2$.

(A6). Для любых $(p, q) \in S^M_{pq}$ в области G справедливы следующие оценки:

$$\begin{aligned} &|f'_p|,\ |f'_q| \le N_1(M) < +\infty, \\ &|f| \le N_2(M) < +\infty, \\ &S^M_{pq} = \{(p, q) : |p| < M_1,\ |q| < M_2\}, \end{aligned}$$

M_1 и M_2 - положительные постоянные, а $M = \min\{M_1, M_2\}$.

Рассмотрим функционал

$$I(u) = \iint_G F(x, y, w_1, w_2, u_1, u_2)dxdy \tag{3.2}$$

где $x + iy = z,\ w_1 + iw_2 = w,\ u_1 + iu_2 = u$, а вещественная функция F непрерывно дифференцируема по w_1, w_2, u_1, u_2 и принадлежат пространству $L_p(G)$. $p > 2,\ (x, y) \in G$.

Поставим следующую задачу оптимального управления [5, 6]: Найти функцию $u_0(z) \in \Omega$, при которой решение $w_0(z)$ краевой задачи Бицадзе-Самарского (3.1) придает функционалу (3.2) минимальное значение. Функцию $u_0(z) \in \Omega$ назовем оптимальным управлением, а соответствующее решение $w_0(z)$ - оптимальным решением.

Справедлива следующая теорема:

Теорема 3. Пусть выполнены условия (A1)-(A6), $u_0(z)$ - оптимальное управление, $w_0(z)$ - соответствующее ему решение задачи (3.1), $\psi_0(z)$ - решение сопряженной задачи:

$$\begin{gathered}\partial_{\bar z}\left(\psi(z)\right)+\frac{\partial f(u_0)}{\partial w}\psi(z)+\frac{\partial \overline{f}(u_0)}{\partial w}\overline{\psi}(z)= \\ =-2\partial_w F(u_0),\ \ z\in G, \\ \operatorname{Re}[\psi(z)]=0,\ \ z\in\Gamma,\end{gathered} \tag{3.3}$$

$$\operatorname{Re}[\psi(z_0^+)-\psi(z_0^-)]=\sigma\cdot\operatorname{Re}[\psi(z)],\ \ z_0\in\gamma_0,\ \ z\in\gamma,$$

тогда почти всюду на G выполняется соотношение

$$\begin{gathered}\operatorname{Re}[(f(z,w_0(z),\overline{w}_0(z),u_0(z))\psi_0(z)+ \\ +F(x,y,w_{01},w_{02},u_{01},u_{02})]= \\ \inf_{u\in U}\operatorname{Re}[(f(z,w_0(z),\overline{w}_0(z),u)\psi_0(z)+ \\ +F(x,y,w_{01},w_{02},u_1,u_2)].\end{gathered}$$

4. Необходимые и достаточные условия оптимальности

Пусть, область $\overline{G}$ прямоугольник $\{z=x+iy:\ 0\le x\le 1,\ 0\le y\le 1\}$, Γ - граница области G, а $\gamma_0=\{z_0=x_0+iy:\ 0\le y\le 1\}$, $\gamma=\{z=1+iy:\ 0\le y\le 1\}$, $z^*\in\Gamma\setminus\gamma$, $c(z),\ d(z),\ B(z),\ f(z)\in L_p(\overline{G}),\ p>2,\ g(z)\in C_\alpha(\Gamma\setminus\gamma)$, $0<\alpha<1$. Для каждого фиксированного $u\in\Omega$ в области $\overline{G}$ рассмотрим следующую краевую задачу

$$\begin{gathered}\partial_{\bar z}w+B(z)\overline{w}=f(z)u,\ z\in G, \\ \operatorname{Re}[w(z)]=g(z),\ z\in\Gamma\setminus\gamma,\ \operatorname{Im}[w(z^*)]=\text{const}, \\ \operatorname{Re}[w(z)]=\sigma\operatorname{Re}[w(z_0)], \\ z_0\in\gamma_0,\ \ z\in\gamma,\ \ 0<\sigma=\text{const}.\end{gathered} \tag{4.1}$$

Рассмотрим функционал

$$I(\omega)=\operatorname{Re}\iint_G[c(z)w(z)+d(z)u(z)]dxdy \tag{4.2}$$

и поставим следующую задачу оптимального управления: Найти функцию $u_0(z)\in\Omega$, при которой решение краевой задачи Бицадзе-Самарского (4.1) придает функционалу (4.2) минимальное значение.

Теорема 4. Пусть $\psi(z)$ - решение сопряженной задачи:

$$\begin{gathered}\partial_{\bar z}\psi(z)-\overline{B}(z)\overline{\psi}(z)=c(z),\ \ G\setminus\gamma_0, \\ \operatorname{Re}[\psi(z)]=0,\ \ z\in\Gamma, \\ \operatorname{Re}[\psi(z_0^+)-\psi(z_0^-)]=\sigma\operatorname{Re}[\psi(z)], \\ z_0\in\gamma_0,\ z\in\gamma_1,\end{gathered} \tag{4.3}$$

тогда, для оптимальности $u_0(z)$, $w_0(z)$ необходимо и достаточно почти всюду на G выполнение следующего равенства:

$$\begin{gathered}\operatorname{Re}[(d(z)-\psi(z)f(z))u_0(z)]= \\ =\inf_{u\in U}\operatorname{Re}[(d(z)-\psi(z)f(z))u(z)]\end{gathered}. \tag{4.4}$$

5. Задача оптимального управления для одной нелокальной краевой задачи

Пусть область $\overline{G}$ прямоугольник, $\overline{G}=[0,1]\times[0,1]$, Γ - граница области G, $0<x_0<1$, $\gamma_0=\{(x_0,y):\ 0\le y\le 1\}$, $\gamma=\{(1,y):\ 0\le y\le 1\}$, $a(x,y),\ b(x,y),\ c(x,y),\ d(x,y)\in L_2(\overline{G})$, $0\le q(x,y)\in L_\infty(\overline{G})$, $U=[-1,1]$, Ω - множество всех допустимых функций управления $\omega(x,y):G\to U$.

Для каждого фиксированного $\omega\in\Omega$ в области $\overline{G}$ рассмотрим краевую задачу Бицадзе-Самарского для уравнений Гельмогольца (описывающих, в частности, процессы теплопереноса в бесконечно тонкой пластине с внутренним источником тепла и теплообменом в среду с её поверхностей [12]):

$$\begin{gathered}\frac{\partial^2u}{\partial x^2}+\frac{\partial^2u}{\partial y^2}-q(x,y)u= \\ =a(x,y)\omega(x,y)+b(x,y),\ \ (x,y)\in G, \\ u(x,y)=0,\ \ \ (x,y)\in\Gamma\setminus\gamma, \\ u(1,y)=\sigma u(x_0,y),\ \ 0\le y\le 1,\ \sigma>0.\end{gathered} \tag{5.1}$$

Рассмотрим функционал

$$I(u)=\iint_G[c(x,y)u(x,y)+d(x,y)\omega(x,y)]dxdy \tag{5.2}$$

и поставим задачу оптимального управления: Найти функцию $\omega_0(x,y)\in\Omega$, при которой решение краевой задачи (5.1) придает функционалу (5.2) минимальное значение.

Теорема 5. Пусть ψ_0 - решение сопряженной задачи

$$\begin{gathered}\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}-q(x,y)\psi=-c(x,y),\ \ (x,y)\in G\setminus\gamma_0, \\ \psi(x,y)=0,\ \ (x,y)\in\Gamma, \\ \frac{\partial\psi(x_0^+,y)}{\partial x}-\frac{\partial\psi(x_0^-,y)}{\partial x}=\sigma\frac{\partial\psi(1,y)}{\partial x},\ \ 0\le y\le 1,\end{gathered} \tag{5.3}$$

тогда для оптимальности (u_0, ω_0) необходимо и достаточно выполнение принципа минимума

$$\inf_{\omega \in U}[d(x,y) + a(x,y)\psi_0(x,y)]\omega = \\ = [d(x,y) + a(x,y)\psi_0(x,y)]\omega_0 \qquad (5.4)$$

почти всюду на G [6, 7].

6. Алгоритм решения задачи оптимального управления

Сначала решаем сопряженную задачу (5.3) для нахождения решения $\psi_0(x,y)$; При помощи функций $\psi_0(x,y)$ из (5.4) строим оптимальное управление $\omega_0(x,y)$; Решаем задачу (5.1) для нахождения оптимального решения $u_0(x,y)$.

Для решения задачи (5.1) рассмотрим следующий итерационный процесс [3,8]:

$$\frac{\partial^2 u^{k+1}}{\partial x^2} + \frac{\partial^2 u^{k+1}}{\partial y^2} - q(x,y)u^{k+1} = \\ = a(x,y)\omega + b(x,y), \quad (x,y) \in G, \\ u^{k+1}(x,y) = 0, \qquad (x,y) \in \Gamma \setminus \gamma, \qquad (6.1) \\ u^{k+1}(1,y) = \sigma u^k(x_0, y), \\ 0 \le y \le 1, \quad \sigma > 0, \quad k = 0,1,2,\ldots .$$

Решение сопряженной задачи (5.3) представим в виде $\psi = w + w^*$, где w^* – решение задачи Дирихле:

$$\frac{\partial^2 w^*}{\partial x^2} + \frac{\partial^2 w^*}{\partial y^2} - q(x,y)w^* = -c(x,y), \quad (x,y) \in G, \qquad (6.2)$$

$$w^*(x,y) = 0, \qquad (x,y) \in \Gamma,$$

а w – решение неклассической краевой задачи:

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} - q(x,y)w = 0, \quad (x,y) \in G \setminus \gamma_0, \\ w(x,y) = 0, \quad (x,y) \in \Gamma, \\ \frac{\partial w(x_0^+, y)}{\partial x} - \frac{\partial w(x_0^-, y)}{\partial x} = \qquad (6.3) \\ = \sigma \frac{\partial w(1,y)}{\partial x} + \sigma \frac{\partial w^*(1,y)}{\partial x}, \quad 0 \le y \le 1.$$

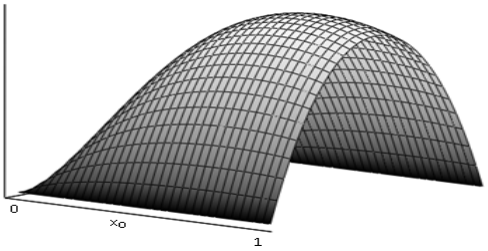


Рис. 1. Результат численного решения задачи (6.1) в графическом виде.

Задача (6.2) имеет единственное решение и принадлежит пространству $\overset{o}{W}{}_2^2(G)$ [9], а решение задачи (6.3) пространству $w(x,y) \in \overset{0}{W}{}_2^2(G \setminus \gamma_0) \cap \overset{0}{W}{}_2^1(G)$ [7, 10]. Для решения задачи (6.3) рассмотрим итерационный процесс [10]:

$$\frac{\partial^2 w^{k+1}}{\partial x^2} + \frac{\partial^2 w^{k+1}}{\partial y^2} - q(x,y)w^{k+1} = 0, \quad (x,y) \in G \setminus \gamma_0, \\ w^{k+1}(x,y) = 0, \quad (x,y) \in \Gamma, \\ \frac{\partial w^{k+1}(x_0^+, y)}{\partial x} - \frac{\partial w^{k+1}(x_0^-, y)}{\partial x} = \qquad (6.4) \\ = \sigma \frac{\partial w^k(1,y)}{\partial x} + \varphi(y), \quad 0 \le y \le 1, \quad k = 0,1,2,\ldots,$$

где $\varphi(y) = \sigma \dfrac{\partial w^*(1,y)}{\partial x}$, $w^0(x,y)$ - начальное приближение, которое можно взять равным нулю.

Для численного решения задачи (5.1) и (5.3) на каждом шаге итераций (6.1) и (6.4) будем пользоваться встроенной функций **relax(a, b, c, d, e, f, u, rjac)** на Mathcad'e [10,11].

Для примера рассмотрим краевые задачи (5.1) и (5.3), где $q(x,y) = x + y$, $a(x,y) = 1$, $b(x,y) = 0$, $c(x,y) = 1$, $x_0 = 1/2$. Результаты численного решения задачи (6.1) и (6.4) графическом виде представлены на рис.1 и рис.2.

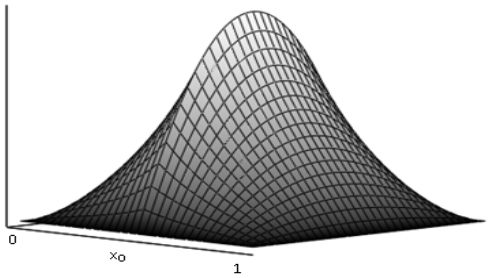


Рис. 2. Результат численного решения задачи (6.4) в графическом виде.

The optimal control problem for quasi-linear differential equations with nonlocal boundary conditions

G. Meladze, D. Devadze, V. Beridze, M. Surguladze, P.I. Kandalov

Abstract: The aim of this paper is to consider nonlocal boundary value problem of the Bitsadze-Samarskiy for quasi-linear first-order differential equation on a space. The authors prove a theorem of the existence and uniqueness of a generalized solution in the space $C_\alpha(\overline{G})$. For linear boundary value problem is proved the existence of solutions in the space $C_\alpha^p(\overline{G})$ and obtained a priori estimate. The problem of optimal control has been formulated for quasi-linear differential equations of the first order with integral performance criterion. The necessary conditions of optimality in the form of the Pontryagin's maximum principle have been obtained. The authors prove necessity and sufficiency of optimal conditions for linear optimal control problem. The problem of optimal control are considered for Helmholtz equations using Bitsadze-Samarskiy's boundary condition. The theorem about necessity and sufficiency conditions of optimality is presented. An algorithm for solving the problem of optimal control using the Mathcad package, is presented.

Keywords: nonlocal boundary value problem, the problem of Bitsadze-Samarskii, generalized solution, optimal control, optimality conditions, maximum principle.

Литература

1. И.Н. Векуа. Обобщенные аналитические функции. М: Физматгиз. 1988. 512 с.
2. В.С. Владиров. Уравнения математической физики. М.: Наука 1981. 512с.
3. А.В. Бицадзе, А.А. Самарский // Докл. АН СССР. 1969. –Т.185. - #2. с. 739-740.
4. Г.Ф. Манджавидзе, В. Тучке //ТГУ, Ин-т ПМ. И.Н. Векуа. 1983. 123с.
5. Л.С. Понтрягин, В.Г. Болтянский, Р.В. Гамкрелидзе, Е.Ф. Мищенко. Математическая теория оптимальных процессов. М.: Наука, 1983. 390с.
6. Г.В. Меладзе, Т.С. Цуцунава, Д.Ш. Девадзе // Деп. ГрузНИИНТИ 01.04.87. №300-187. 51с.
7. D. Devadze, V. Beridze. //Proc. A. Razmadze Math. Inst. 2013. Vol.161. pp. 47-53.
8. D.G. Gordeziani // Inst, Prikl. Math.Tbilisi Gos. Univ., Dokl., No (1970), pp.39-41.
9. О.А. Ладыженская, Н.Н. Уральцева. Линейные и квазилинейные уравнения эллиптического типа. М. Наука.1973. 576 с.
10. D. Devadze, V. Beridze //Bull. Georgian Academy Sciences. Vol. 7, no.1. (2013), 44-48.
11. D.V. Kiryanov. Mathcad 14, St. Petersburg.: BHV-Petersburg, 2007. 680p.
12. А.Г. Мадера. Моделирование теплообмена в технических системах. – М.: Науч. фонд «Первая исслед. лаб. им. акад. В.А. Мельникова», 2005.

Об использовании явных разностных схем для интегрирования параболических уравнений.

М.Л. Бахмутский

кандидат физико-математическх наук

Аннотация: Рассмотрены возможности ослабления требований устойчивости явных разностных схем. Предложены модификации явных схем значительно ослабляющие эти требования. Приведены результаты численных расчетов модельной задачи и их сопоставления с аналитическим решением.

Ключевые слова: уравнение теплопроводности, явные схемы, условия устойчивости.

В задачах математического моделирования пластовых систем широко применяется IMPES- метод – явный по насыщенности и неявный по давлению. Это вызвано тем, что условия устойчивости явных схем решения параболического уравнения для давления налагают слишком жесткие ограничения на временной шаг интегрирования. Вместе с тем, для решения ряда практических задач как при поиске систем разработки месторождений, так и при мониторинге разработки необходимо использовать громадные сетки и моделировать процессы фильтрации, протекающие длительное время. Это требует распараллеливания вычислений. Вместе с тем известно, что абсолютно устойчивые неявные схемы плохо распараллеливаются.

Известно [1], что только первые гладкие собственные элементы разностной аппроксимации аппроксимируют дифференциальный оператор и, поэтому, необходимо правильно рассчитывать эволюцию во времени несколько первых низкочастотных компонент и подавлять рост высокочастотных. Оценивая условия устойчивости явных разностных схем, легко заметить, что если исходить из условия устойчивости первых компонент, то временной шаг может быть значительно больше обычно используемого, при котором не допускается рост высокочастотных компонент. Если внешним образом, на каждом временном шаге, подавлять рост высокочастотных компонент, то шаг по времени можно значительно увеличить и его будут ограничивать не столько соображения устойчивости, сколько требования точности счета. В качестве ограничителя роста высокочастотных компонент, предлагается использовать сингулярное разложение [2] матрицы решения или траекторной матрицы [3] и использование нескольких первых при отбрасывании старших компонент. В данной работе приведены примеры численных экспериментов для линейных параболических уравнений, имеющих аналитическое решение. Нелинейные уравнения будут рассмотрены позже. Были проведены эксперименты по использованию традиционной явной разностной схемы плюс сингулярный анализ и схемы типа «классики». При этом, шаг по времени существенно увеличивается. Рассмотрим простейшее уравнение теплопроводности. Начально-краевая задача для него имеет вид (1).

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}\ ;\ 0\le x\le 1\ ; u(0,t)=u(1,t)=0\ ; u(x,0)=f(x) \qquad (1)$$

Решение этой задачи дается формулой:

$$u(x,t)=\sum_{k=1}^{\infty} c_k\, e^{-\pi^2 k^2 t}\sin(\pi k x)$$

Здесь $\gamma_k=-\pi^2k^2$ k-е собственное значение оператора второй пространственной производной, а $\sin(\pi kx)$ $k-$ я собственная функция этого оператора. Начальное условие имеет вид:

$$f(x)=\sum_{k=1}^{\infty} c_k \sin(\pi k x)$$

Введем пространственную сетку из N узлов с постоянным шагом h. Уравнение (1) в узлах этой сетки примет вид (2):

$$\frac{d\tilde{u}_i}{dt}=\left(\tilde{u}_{i+1}-2\tilde{u}_i+\tilde{u}_{i-1}\right)\Big/_{h^2}\ ; i\in[2,N-1]\ ;\ \tilde{u}_1=\tilde{u}_N=0;\ \ \tilde{u}_i(0)=f(x_i) \qquad (2)$$

Решение задачи (2) дается формулой (3)

$$\tilde{u}(x_i,t)=\sum_{k=1}^{N-1}\tilde{c}_k\, e^{-\mu_k^2 t}\sin(\pi k x_i);\ \mu_k=2(N-1)\sin\left(\frac{\pi k}{2(N-1)}\right); \qquad (3)$$

где

$$x_i=h(i-1)={}^{(i-1)}\!/_{(N-1)};$$

$$f(x_i)=\sum_{k=1}^{N-1}c_k\sin(\pi k x_i)$$

а

$\tilde{\gamma}_k=-\mu_k^2$ - k-е собственное значение разностной аппроксимации второй производной, $\sin(\pi k x_i)$ – k –я собственная функция.

На *Рис.1* нанесены собственные значения второй производной из задачи (1) (верхняя линия) и собственные значения ее разностной аппроксимации (нижняя линия). N=101.

$$\tau\le\frac{h^2}{2\left(\sin\frac{\pi k}{2(N-1)}\right)^2} \qquad (4)$$

Известное же соотношение $\tau\le\frac{h^2}{2}$ ограничивает все гармоники.

Проделаем некоторые простейшие оценки. Предположим, что все гармоники кроме первой равны нулю. Тогда условие устойчивости схемы принимает вид:

$$\tau_1\le\frac{2h^2(N-1)^2}{\pi^2}=\frac{2}{\pi^2}\approx 0.2$$

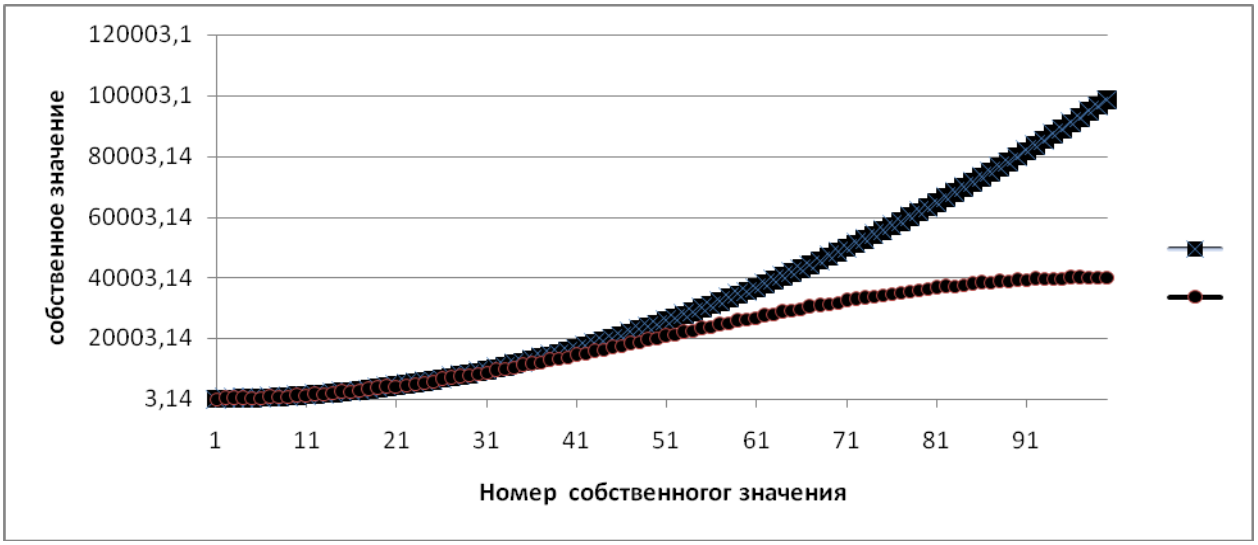


Рис. 1

График демонстрирует тот факт, что менее 1/5 от всех конечно-разностных спектральных компонент аппроксимируют дифференциальный оператор. Поэтому желательно правильно рассчитать вклад нескольких первых, наиболее гладких компонент и подавлять более высокие. Если предположить , что в решении на n-м временном слое, все гармоники, кроме к-й равны нулю, то условие устойчивости при переходе к n+1-му слою будет выглядеть так:

Если все гармоники, кроме двух первых равны нулю, то условие устойчивости равно

$$\tau_2\le\frac{h^2(N-1)^2}{2\pi^2}=\frac{1}{2\pi^2}\approx 0.05$$

Отношение условий устойчивости при h=0.01 $\frac{\tau_1}{\tau}=4*10^3$, $\frac{\tau_2}{\tau}=10^3$,т.е. при вычислении эволюции первой или двух первых гармоник условие устойчивости (допустимый шаг по времени) возрастает в 4000 или 1000 раз. Предположим, что $\tau=0.1h$ и

оценим количество гармоник, удовлетворяющих условию устойчивости. Тогда:

$$0.1h \approx \frac{h^2}{2\left(\sin\left(\frac{\pi k}{2(N-1)}\right)\right)^2}; \qquad k \approx \sqrt{2(N-1)}$$

Если N=101, то k=14, что согласуется с графиком на Рис.1. В таблице 1 приведены оценки числа гармоник и отношения условий устойчивости для этих гармоник в зависимости от шага сетки при $\tau = 0.1h$.

Таким образом, при подавлении высших «паразитных»

Элементы аппроксимирующей матрицы имеют вид:

$$a_{i,j} = \sum_{\sigma_p \geq \sigma_0} \sigma_p u_p(i) v_p(j)$$

где $\sigma_0 = \sigma_{\max} \varepsilon$.

После этого получаем решение на следующем временном слое:

Таблица 1

Пространственный шаг сетки	*Число гармоник*	*Отношение временных шагов*
0.01	*14*	*20*
0.001	*44*	*200*
0.0001	*141*	*2000*
0.00001	*447*	*20000*

гармоник можно дробить пространственные шаги и не получать чрезмерно жесткие ограничения на временные.

Для подавления высших гармоник предлагается следующий алгоритм. Разностная аппроксимация уравнения (1) берется в стандартном виде:

$$\frac{\tilde{u}_i^{n+1} - u_i^n}{\tau} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}, \qquad (5)$$

(здесь u_i^n - значение функции в узле i на n-м временном слое, а $\tilde{u}_i^{n+1}$ - значение функции в узле i на n+1 – м временном слое, которое полагаем промежуточным).

По полученным значениям функции на n+1 –м временном слое, строится траекторная матрица размерностью $m \times n$, где

$$m = \left[\frac{N+1}{2}\right], n = N + 1 - m .$$

Элементы матрицы $\tilde{A} = \{\tilde{a}_{i,j}\}$ находятся как $\tilde{a}_{i,j} = \tilde{u}_{i+j-1}^{n+1}$. Находится сингулярное разложение

$$\tilde{A} = U \sum V^T$$

этой матрицы.

Затем матрица приближается отбрасыванием сингулярных чисел меньших некоторого граничного.

$$u_j^{n+1} = \begin{cases} \frac{1}{j}\sum_{i=1}^{j} a_{i,j-i+1}, 1 \leq j \leq m \\ \frac{1}{m}\sum_{i=1}^{m} a_{i,j-i+1}, m \leq j \leq n \\ \frac{1}{N-j+1}\sum_{i=}^{N-j+1} a_{i+j-n,n-i+1}, n \leq j \leq N \end{cases} \qquad (6)$$

Временной шаг считается завершенным и вычисляется следующий временной шаг.

Рассмотрим для иллюстрации следующую задачу:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \sin\left(\frac{\pi x}{2l_x}\right)$$

$$u(0,t) = 0; \ \frac{du}{dx}(l_x, t) = 0; \ \ u(x, 0) = 0 \qquad (7)$$

Эта задача имеет аналитическое решение:

$$u(x,t) = \left(\frac{2l_x}{\pi}\right)^2 \left[1 - e^{-\left(\frac{\pi}{2l_x}\right)^2 t}\right] \sin\left(\frac{\pi x}{2l_x}\right) \qquad (8)$$

Рис 2

Численный счёт велся по алгоритму (5),(6) при $l_x = 2.$ На графике (*Рис.2*) приведены результаты численного счета и аналитического решения по формуле (7) для момента t=2. Шаг интегрирования по времени равен 0.1h, h=0.002, $\varepsilon = 0.01.$ Если исходить из обычного условия устойчивости, то шаг по времени должен быть меньше 0.000002. Несмотря на то, что шаг по времени в 100 раз превышал максимально допустимый по традиционному условию устойчивости, графики точного и численного решения практически сливаются. На *Рис.3* приведен график относительной ошибки в % численного решения в сравнении с аналитическим для момента t=2.

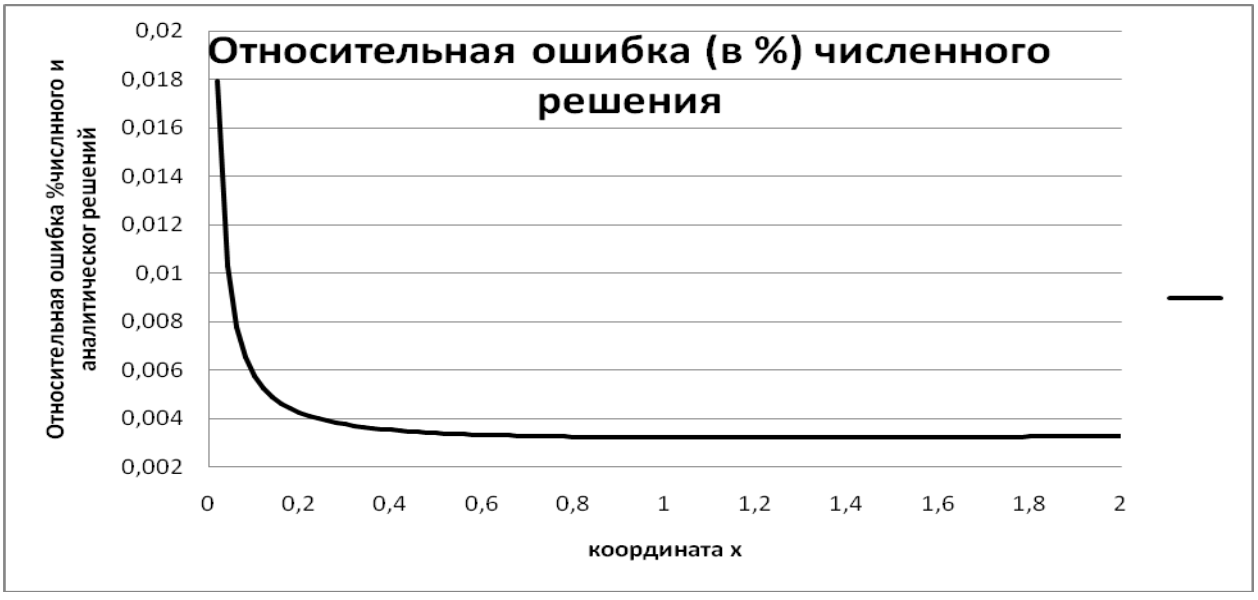


Рис. 3

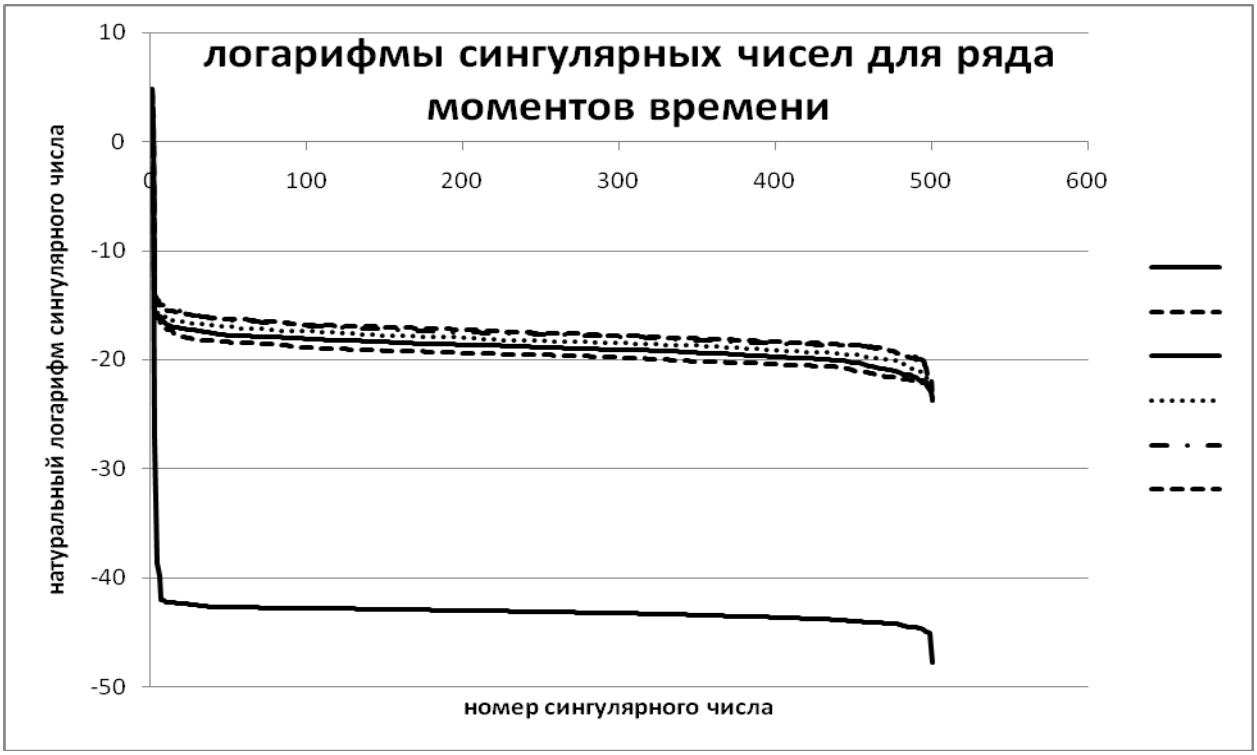


Рис 4

На *Рис.4* и *Рис.5* приведены графики натуральных логарифмов сингулярных чисел траекторных матриц для ряда различных моментов времени. Из графиков видно, что надо использовать 2 сингулярных числа, остальные отбрасываются.

Если уравнение теплопроводности аппроксимируется на неравномерной сетке, то шаг по времени должен удовлетворять условию $\tau \leq \frac{1}{2}\min h_i^2$.

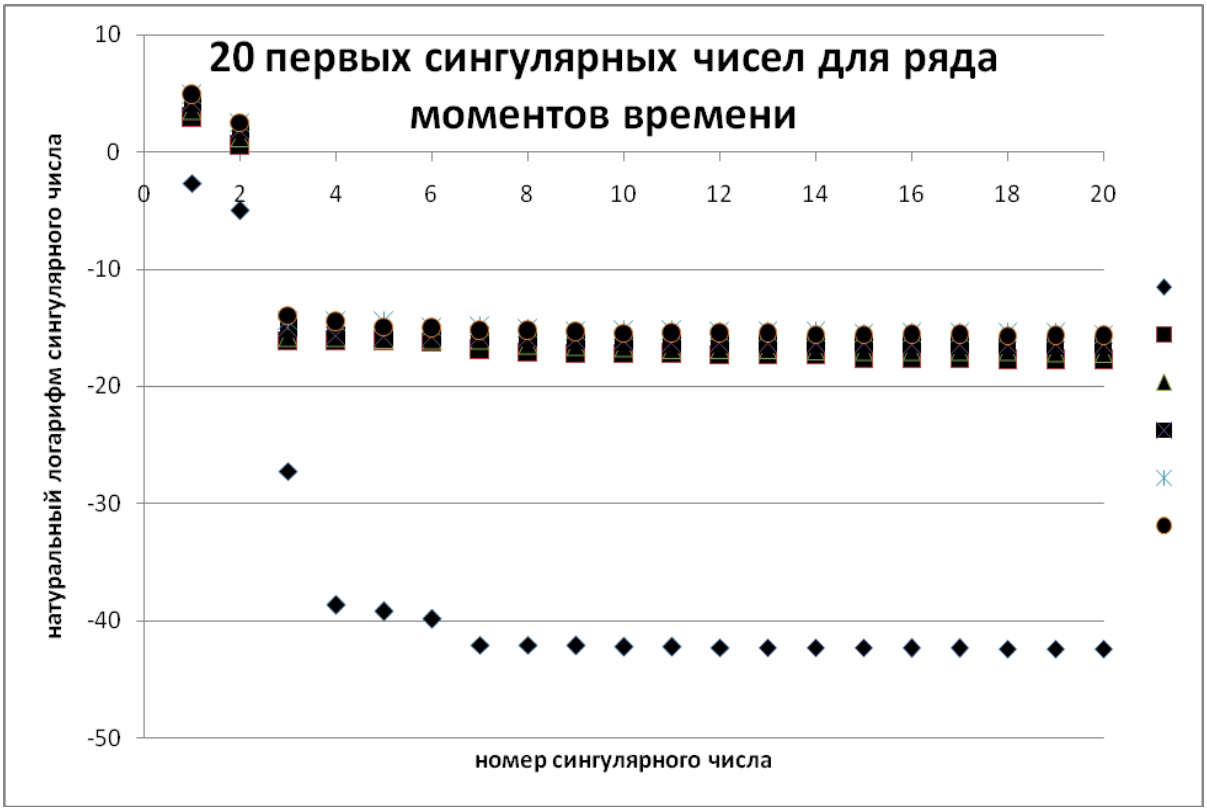


Рис 5

Рассмотрим задачу (7),(8) на неравномерной сетке. Приведем три примера расчёта

$$l_0 = 0.02;\ N_0 = 11; q = 1.05; N = 201;\ l_x = 445.856\ ; h = 0.002;\ \tau_0 = 2 * 10^{-6}$$

$$l_0 = 0.002;\ N_0 = 11; q = 1.05; N = 201;\ l_x = 44.585\ ; h = 0.0002;\ \tau_0 = 2 * 10^{-8}$$

$$l_0 = 0.0002;\ N_0 = 11; q = 1.05; N = 201;\ l_x = 4.4585\ ; h = 0.00002;\ \tau_0 = 2 * 10^{-10}$$

где $h = l_0/(N_0 - 1)$, если $0 \le x \le l_0$

$h_{i+1} = h_i q$, если $l_0 \le x \le l_x$

Брались сингулярные тройки, сингулярные числа которых больше $\sigma_{max} * 0.01$. Шаг по времени в приведенных графиках брался равным h, т.е. отношение шага к стандартному равнялось 1000,10000,100000. Графики построены для момента времени t=2. На графиках нанесены аналитическое решение и решение задачи (7) по алгоритму (5) и (6).

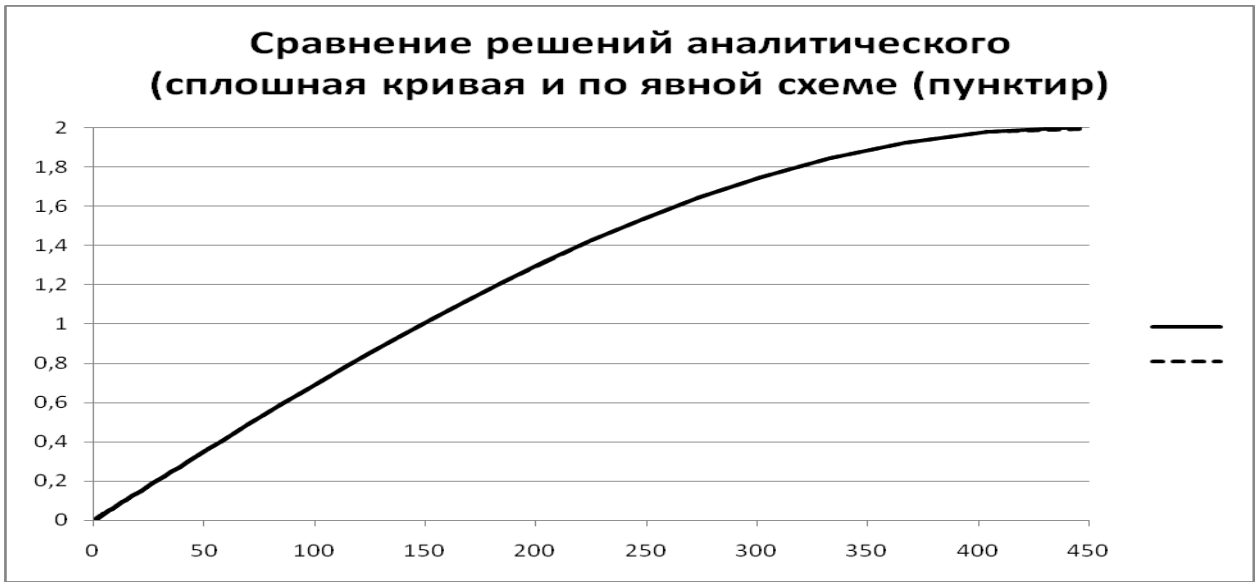


Рис 6

Рис 7

1)

Рис 8

Существуют абсолютно устойчивые явные схемы, такие как схема В.К. Саульева [4], Дюфорта – Франкела [5], и схема «классики» [6],[7]. Схемы Саульева и Дюфорта-Франкела все - таки не очень хорошо аппроксимируют уравнение теплопроводности, схема же «классики» в наших экспериментах показала прекрасные результаты для постоянных пространственных шагов и неустойчивость в расчетах с переменными. Однако небольшие изменения, превращение ее в трехслойную схему устранили эти трудности.

Рассмотрим уравнение:

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+f$$

Проведя пространственную дискретизацию этого уравнения, получим

$$\frac{du_i}{dt}=\frac{1}{\bar{h}_i h_i}(u_{i+1}(t)-u_i(t))-\frac{1}{\bar{h}_i h_{i-1}}(u_i(t)-u_{i-1}(t))+f_i(t);\quad \bar{h}_i=0.5*(h_i+h_{i-1}).$$

Интегрируя эти уравнения по t от t_{n-1} до t_{n+1}, на первом полушаге находим

$$u^n_{2j}-u^{n-1}_{2j}=\frac{\tau}{\bar{h}_{2j}h_{2j}}\left(u^{n-1}_{2j+1}-u^{n-1}_{2j}\right)-\frac{\tau}{\bar{h}_{2j}h_{2j-1}}\left(u^{n-1}_{2j}-u^{n-1}_{2j-1}\right)+f^{n-1}_{2j}\tau,\quad j\in 1\ldots\frac{N-1}{2}$$

$$u^n_{2j-1}-u^{n-1}_{2j-1}=\frac{\tau}{\bar{h}_{2j-1}h_{2j-1}}(u^n_{2j}-u^n_{2j-1})-\frac{\tau}{\bar{h}_{2j-1}h_{2j-2}}(u^n_{2j-1}-u^n_{2j-2})+f^n_{2j-1}\tau,\quad j\in 1\ldots\frac{N-1}{2}$$

На втором полушаге, соответственно, имеем::

$$u^{n+1}_{2j-1}-u^n_{2j-1}=\frac{\tau}{\bar{h}_{2j-1}h_{2j-1}}\left(u^n_{2j}-u^n_{2j-1}\right)-\frac{\tau}{\bar{h}_{2j-1}h_{2j-2}}\left(u^n_{2j-1}-u^n_{2j-2}\right)+f^n_{2j-1}\tau,\quad j\in 1\ldots\frac{N+1}{2};$$

$$u^{n+1}_{2j}-u^n_{2j}=\frac{\tau}{\bar{h}_{2j}h_{2j}}\left(u^{n+1}_{2j+1}-u^{n+1}_{2j}\right)-\frac{\tau}{\bar{h}_{2j}h_{2j-1}}\left(u^{n+1}_{2j}-u^{n+1}_{2j-1}\right)+f^{n+1}_{2j}\tau,\quad j\in 1\ldots\frac{N-1}{2}..$$

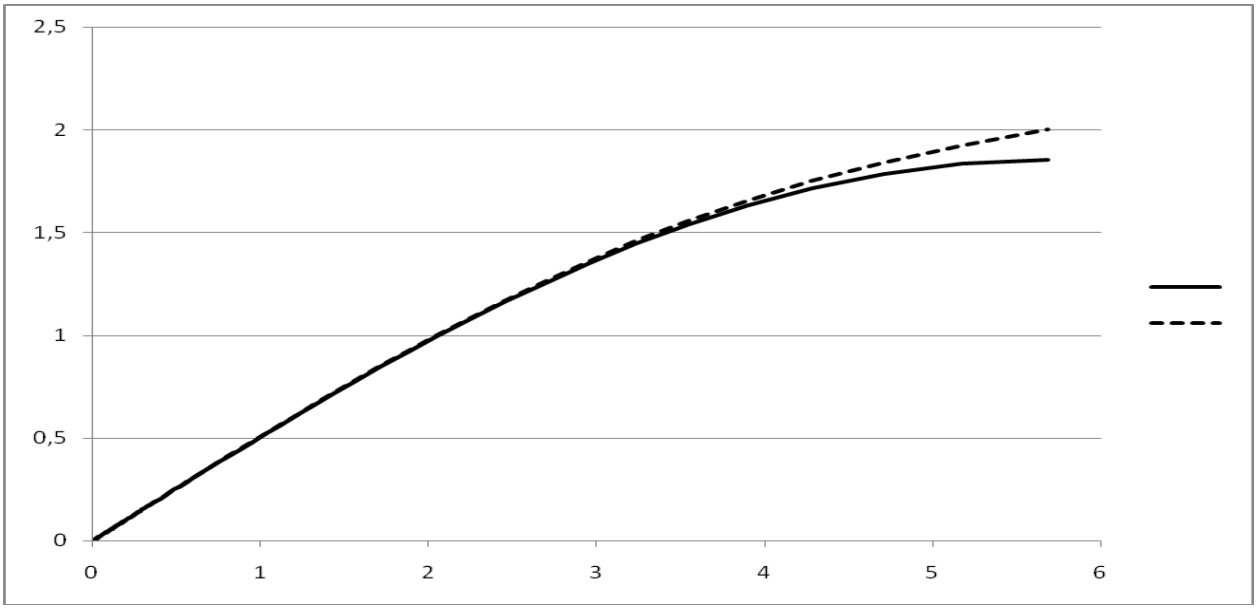


Рис 9

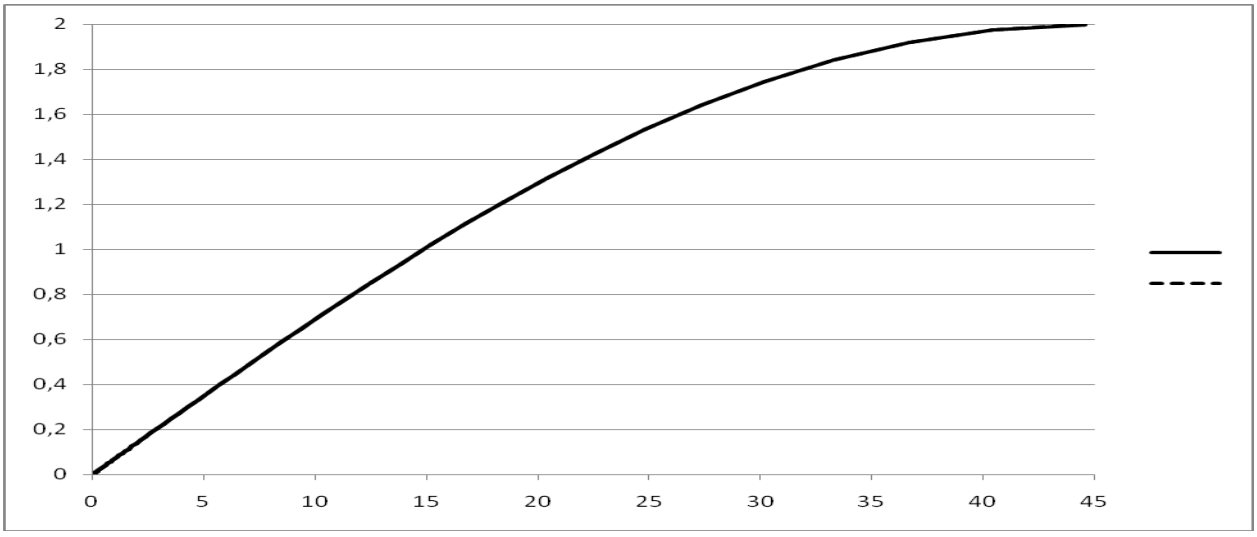


Рис 10

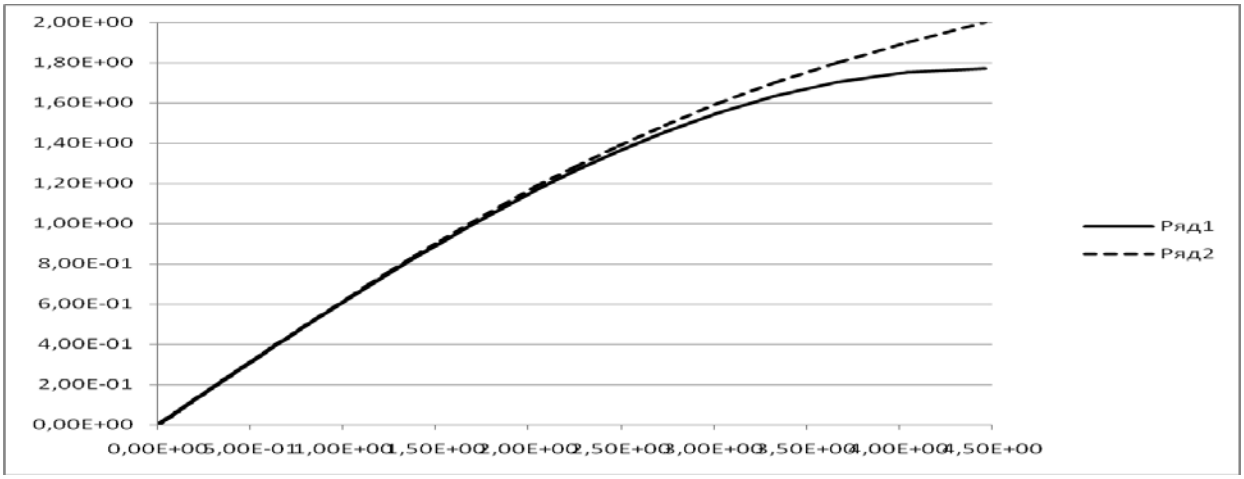


Рис 11

Эта схема имеет второй порядок аппроксимации по времени и напоминает хорошо известную схему Кранка – Николсона. На *Рис*. 9,10,11 приведены графики результатов расчета трех вышеприведенных вариантов по этой схеме и аналитического решения. Видно, что счет устойчив. Сплошная линия – аналитическое решение. Точность расчета по модифицированной схеме «классики» сравнивалась с точностью по схеме Кранка-Николсона. На *Рис.* 12,13,14 приведены кривые относительных ошибок в % по сравнению с аналитическим решением для модифицированной схемы «классики» и схемы Кранка-Николсона. Сплошная линия соответствует счету по Кранку –Николсону.

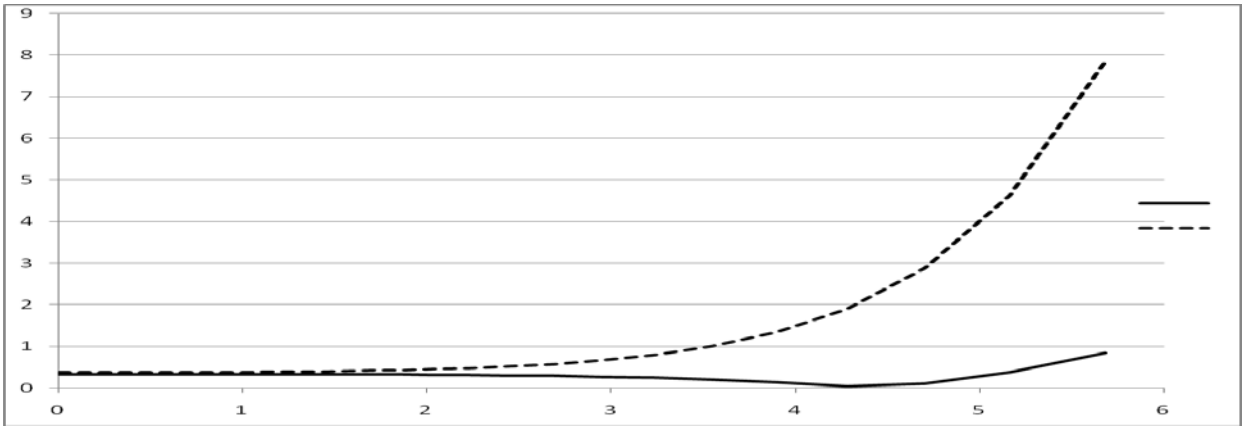


Рис 12

.

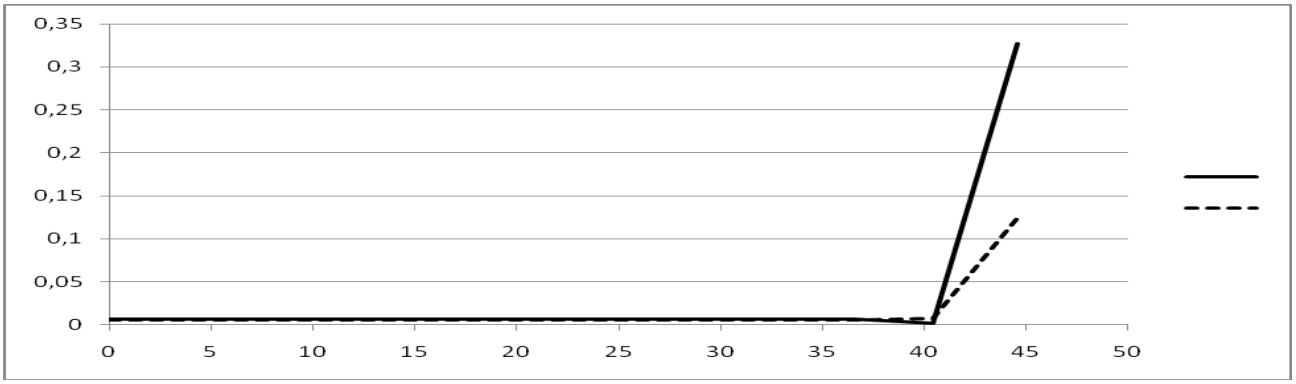


Рис 13

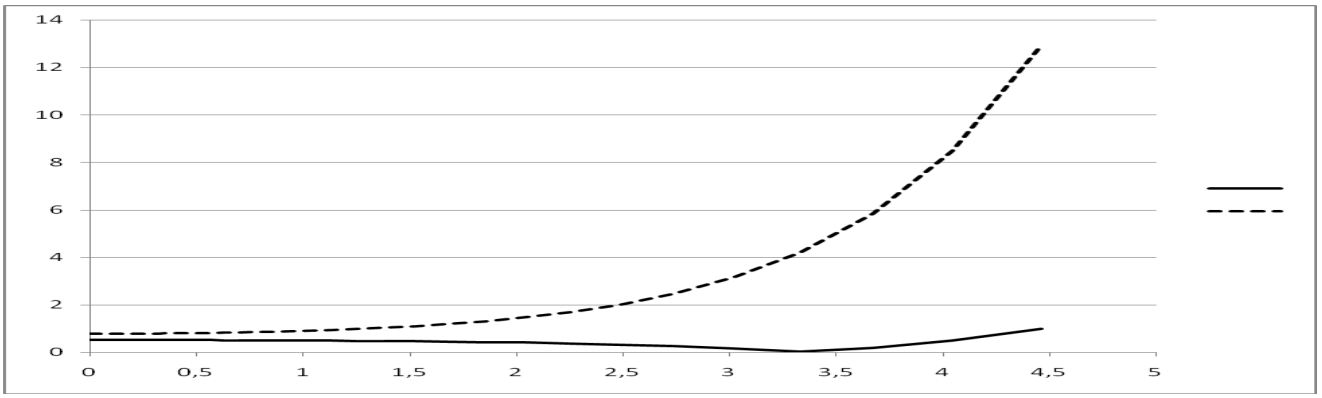


Рис 14

Заключение.

Можно утверждать, что ценой некоторых дополнительных вычислений удается значительно ослабить жесткие условия устойчивости для явных схем. Это открывает определенные перспективы в их использовании для параллельных вычислений.

On application explicit difference scheme to the diffusion equations.

M.L. Bakhmutsky

Abstract: Ability relaxation requirements to stable for explicit difference scheme for diffusion equation are shown. Suggest an algoritm with more weak stable condition. The results of numerical computation and matching with analytical results for model problem are presented.

Keywods: heat equation, explicit schemes, the conditions of stability.

Литература

1. Л.А. Люстерник. О разностных аппроксимациях оператора Лапласа //Успехи матем. наук, 1954, т.IX, вып. 2(60), с. 3-66.
2.Ч.Лоусон, Р.Хенсон. Численное решение задач метода наимньших квадратов, М.,Наука, 1986.
3. R.Vautard, P.Yiou, M.Ghil. Physica D.,1992, v.58, p.95-126
4. В.К.Саульев. Интегрирование уравнений параболического типа методом сеток, М., Физматгиз,1960
5.Д.Поттер. Вычислительные методы в физике, М., Мир, 1975
6.A.R.Gourlay. J.Inst.Math.Appl., n.6, 1970, pp375-90
7. A.R.Gourlay and G.R.McGuire. J.Inst.Math.Appl., n. 7, (1971), pp.216-27

Программная конвейеризация циклов с выбором команд

Н.И. Вьюкова, В.А. Галатенко[1], С.В. Самборский

[1] – *доктор физико-математических наук*

Аннотация: В статье представлен метод программной конвейеризации циклов путем точного совместного решения задач выбора и планирования инструкций для циклических участков программ. Предложен подход, в котором задачи выбора и планирования команд представляются как единая задача целочисленного линейного программирования (ЦЛП). Совмещение выбора команд с их планированием позволяет оптимальным образом учесть наличие циклических зависимостей, возможности параллельного исполнения команд, наличие специализированных вычислительных команд, наличие общих подвыражений, ограничения по числу аппаратных регистров. Предложенный метод может быть полезен при разработке аппаратных ускорителей, а также в генераторах высокопроизводительного кода для микропроцессоров, используемых во встроенных системах.

Ключевые слова: генерация кода, программная конвейеризация циклов, выбор команд, планирование команд, целочисленное линейное программирование (ЦЛП).

1. Введение

Эффективность кода, генерируемого компилятором для приложений вычислительного характера, в высокой степени зависит от набора применяемых оптимизаций циклов. Одним из важных методов оптимизации циклов, используемых в современных компиляторах, является программная конвейеризация циклов, которая позволяет совмещать выполнение операций, относящихся к разным итерациям цикла. Рис. 1 иллюстрирует временные характеристики выполнения исходного (*а*) и конвейеризованного (*б*) циклов. В рамке выделено тело конвейеризованного цикла, состоящее из команд нескольких соседних итераций исходного цикла

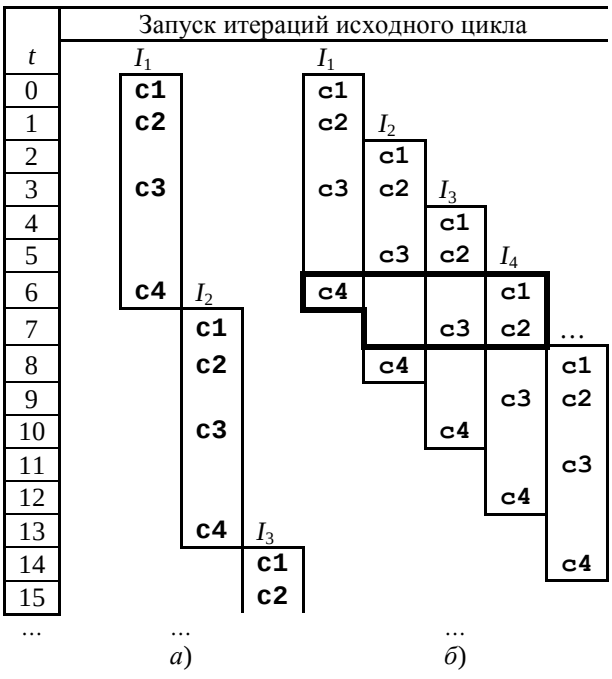


Рис. 1. Выполнение (*а*) исходного и (*б*) конвейеризованного циклов

Поскольку эта оптимизация существенно зависит от архитектуры целевого процессора, включая такие факторы, как набор функциональных устройств, латентности команд, организация конвейеров, то она обычно выполняется на стадии генерации кода после выбора команд, но до распределения регистров.

Этот общепринятый подход позволяет упростить реализацию генератора кода, но обладает рядом недостатков с точки зрения оптимальности генерируемого кода. С одной стороны, выбор команд может ограничить возможности эффективной конвейеризации циклов. С другой стороны, прирост производительности, полученный в результате конвейеризации цикла, может быть сведен на нет из-за вставки спилл-кода во время распределения регистров.

Ещё одно ограничение традиционной схемы – использование эвристических алгоритмов для выбора команд, их планирования и распределения регистров. Массовые компиляторы должны работать быстро, поэтому в них применяют, в основном, алгоритмы, время работы которых линейно зависит от размера входных данных. Такие алгоритмы, при грамотном подборе эвристик, позволяют получать код, близкий к оптимальному, для большинства современных универсальных микропроцессорных архитектур. Однако при компиляции для специализированных микропроцессорных архитектур, используемых во встроенных системах, применение эвристических методов может приводить к значительным потерям в эффективности результирующего кода по сравнению с ручным ассемблерным программированием. Наиболее ощутимо эти недостатки традиционной схемы проявляются при генерации кода для циклических участков программ.

Идеальным с точки зрения оптимальности генерируемого кода мог бы быть подход, представленный в докладе [1], в котором задачи оптимизации и генерации кода рассматриваются как единая задача математического программирования. В данной работе предлагается подход, в котором задачи выбора и планирования команд решаются совместно

как задача целочисленного линейного программирования (ЦЛП).

Совместное рассмотрение задач подбора и планирования команд позволяет получать более эффективный код за счет того, что выбор команд осуществляется с учетом множества факторов, включая возможность параллельного исполнения команд, межитерационные зависимости по данным, дефицит регистров, наличие общих подвыражений. Хотя распределение регистров в этом процессе не осуществляется, при выборе и планировании команд учитываются ограничения по числу доступных регистров, так что на стадии их распределения не требуется генерация спилл-кода (см. [2, 3]).

Для экспериментов, связанных с развитием метода генерации кода путем совместного решения задач выбора и планирования команд, был реализован прототип кодогенератора ISched (Instruction selection & Scheduling) для линейных и циклических участков программ. ISched позволяет по описанию системы команд и входного участка программы сгенерировать формулировки ЦЛП-задач, реализующих генерацию кода, запустить один из доступных солверов ЦЛП-задач и вывести полученный результат в виде расписания, то есть в виде последовательности команд с указанием номеров тактов, на которых они должны запускаться. Использование генератора кода ISched для линейных участков представлено в [4].

Настоящая работа посвящена применению указанного подхода к генерации кода для циклических участков программ, то есть к реализации программной конвейеризации циклов с подбором команд. Следует отметить, что планирование с подбором команд для циклических участков программ имеет ряд сложностей по сравнению с использованием этого метода для линейных участков. В частности, это связано с необходимостью рассмотрения версий переменных, соответствующих разным итерациям исходного цикла.

Дальнейшее содержание статьи построено по следующему плану. В разделе 2 вводятся основные понятия, связанные с программной конвейеризацией циклов, и рассматриваются особенности конвейеризации циклов с подбором команд. Раздел 3 содержит описание входных данных для прототипа генератора кода ISched и описание системы команд, используемой в примерах. В разделе 4 рассмотрены примеры программной конвейеризации циклов с выбором команд. В разделе 5 обсуждаются формулировки ЦЛП-задач программной конвейеризации циклов с выбором команд, сложность этих задач и солверы, применяемые для их решения. В заключении суммируется опыт проделанной работы и рассматриваются перспективы возможных дальнейших исследований.

2. Планирование с выбором команд для циклических участков кода

2.1. Основные понятия, связанные с программной конвейеризацией циклов

Программная конвейеризация циклов – это метод оптимизации циклов, который может давать такой же эффект, как внеочередное исполнение (out-of-order execution) команд процессором, с той разницей, что переупорядочение выполняется не аппаратурой, а компилятором (или программистом, при программировании на ассемблере).

Результатом программной конвейеризации является цикл, который выполняет те же вычисления, что исходный, и совмещает в своем теле команды, относящиеся к разным итерациям исходного цикла. За счет такого совмещения обеспечивается скрытие латентностей команд и достигается эффективное использование параллелизма на уровне команд, присущего современным микропроцессорным архитектурам. Интервал между запусками последовательных итераций конвейеризованного цикла (в тактах) называют основным интервалом или интервалом запуска и обозначают *II* (Initiation Interval). Для того чтобы обеспечить эквивалентность конвейеризованного цикла исходному, перед входом в цикл и после выхода из него добавляются пролог и эпилог, в которых исполняются команды из, соответственно, начальных и конечных итераций исходного цикла. Кроме того, должно быть скорректировано число повторений цикла.

Размер диапазона соседних итераций исходного цикла, команды которых совмещаются в теле конвейеризованного цикла, называется глубиной конвейеризации. Например, если в теле конвейеризованного цикла (в его первой итерации) выполняются команды из 1-й, 3-й и 4-й итераций исходного цикла, то значение глубины составляет 4. От глубины конвейеризации зависит размер пролога и эпилога конвейеризованного цикла. Она также косвенно влияет на общее число аппаратных регистров, требуемых для конвейеризованного цикла.

Подробнее об этих и других аспектах генерации кода для конвейеризованных циклов см. обзор [5].

В большинстве современных компиляторов применяют метод программной конвейеризации, называемый методом планирования по модулю (modulo scheduling). Суть метода состоит в том, что для значений *II* из некоторого диапазона [II_{min}, II_{max}] последовательно делаются попытки построить расписание цикла, укладывающееся в *II* тактов на итерацию. Расписание для минимального *II*, для которого это удалось сделать, и будет использовано при генерации кода.

Как правило, построение расписаний для фиксированного *II* осуществляется при помощи эвристических алгоритмов, обзор которых можно найти в [5]. Это быстрые алгоритмы, которые для универсальных целевых архитектур позволяют успешно находить расписания при значениях *II*, близких к оптимальным. Однако для специализированных архитектур они далеко не всегда дают приемлемые результаты. Поэтому для таких архитектур, в условиях, когда имеются высокие требования к эффективности генерируемого кода, оправдано применение точных подходов, основанных на использовании методов математического программирования, в частности методов ЦЛП, см. [6, 7].

2.2. Программная конвейеризация циклов с выбором команд

Процесс программной конвейеризации циклов в генераторе кода ISched включает следующие основные шаги.

1. Считывание и обработка входных данных, состоящих из описания системы команд целевого микропроцессора (см. разд. 3) и тела исходного цикла в виде набора поддеревьев, представленных в списочной ЛИСП-подобной форме (см. разд. 4). В традиционной схеме компиляции программная конвейеризация осуществляется, когда уже имеется фиксированный набор команд, реализующих циклическое вычисление. В данном случае на входе имеется лишь некоторое абстрактное представление циклического вычисления в виде набора деревьев.

2. Выделение уникальных поддеревьев и отождествление общих подвыражений в теле исходного цикла. В данной реализации отождествление выполняется на уровне линейного участка, составляющего тело цикла; не делается попыток проводить отождествление с учетом циклических зависимостей.

Каждому уникальному поддереву v_i соответствует значение, равное результату вычисления, описываемого поддеревом. Результат вычисления может быть помещен в разные виды хранилищ, имеющихся в процессоре. В данной работе мы рассматриваем четыре вида хранилищ: `D` – регистры общего назначения, `CC` – регистр кода условия, `M` – память, `V` – псевдохранилище для результатов типа `void`, которые могут «порождаться», например, операциями записи в память.

В формулировке ЦЛП-задачи приходится иметь дело с результатами, порождаемыми на нескольких соседних итерациях исходного цикла. Каждому такому результату соответствует переменная ЦЛП-задачи. Поскольку число переменных должно быть фиксированным, в формулировку задачи был введен дополнительный параметр *DP*, соответствующий максимальной допустимой глубине конвейеризации.

Таким образом, каждому уникальному поддереву v_j, выделенному на шаге 2, ставится в соответствие набор переменных $D_j[i,k]$, $M_j[i,k]$, $V_j[i,k]$, $CC_j[i,k]$, где i принимает значения от 0 до II - 1, а k – от 0 до DP - 1. Общее число переменных ЦЛП-задачи, описывающих значения, которые вычисляются в конвейеризованном цикле, составляет $N \cdot II \cdot DP \cdot 4$, где N – число уникальных подвыражений в теле исходного цикла.

3. Первичный выбор команд, то есть формирование множества команд *Com*, которые в принципе могут участвовать в реализации заданного циклического вычисления. Для этого корень каждого уникального поддерева v_i, сформированного на шаге 2, сопоставляется с древовидными шаблонами в описаниях команд процессора.

В процессе первичного выбора команд сохраняются все команды, для которых оказалось успешным сопоставление хотя бы одного из ее шаблонов с вершиной какого-либо поддерева v_i. Не делается попытка выбрать наиболее выгодный набор команд, как это имеет место в традиционных декларативных методах выбора команд, таких как BURG [8,9]. Окончательный выбор команд откладывается до шага 4.

4. Формирование и поиск решения ЦЛП-задач выбора и планирования команд для заданных значений основного интервала (*II*) процессора и глубины конвейеризации *DP*. При этом *II* пробегает значения 1, 2, … и для каждого *II* значение *DP* варьируется от 1 до DP_{max}, где DP_{max} – константа, задающая максимальную возможную глубину конвейеризации. Процесс завершается, когда для очередной пары значений *II, DP* удается найти решение ЦЛП-задачи.

Ограничения, включаемые в формулировку ЦЛП-задач, обсуждаются в разд. 5.

5. Вывод результатов. Решение ЦЛП-задачи, если оно было найдено, выводится в виде расписания выполнения конвейеризованного цикла. Примеры представлены в разд. 4.

3. Система команд, используемая в примерах

Описание системы команд и описание входных циклических участков для генератора кода ISched представляют собой программы на языке Python [11], чем и обусловлен их синтаксис. Каждый элемент описания является присваиванием, правая часть которого представляет литеральное значение некоторой структуры данных языка Python: список, словарь и т. д. Символ «#» является символом комментария; комментарий продолжается до конца строки.

Описание системы команд и ресурсов целевого процессора для генератора ISched состоит из разделов, содержание которых представлено далее.

Ресурсы (вычислительные устройства, регистры) процессора. Архитектура, описание которой использовано в примерах в разд. 4, включает ресурсы, перечисленные в таблице 1.

Таблица 1. Ресурсы процессора

Имя ресурса	Назначение	Число единиц
`iss`	Устройство выдачи команд на исполнение	2
`iadd`	Устройство сложения	2
`imul`	Устройство умножения	1
`idiv`	Устройство деления	1
`idivsqrt`	Устройство деления и извлечения квадратного корня	1
`dreg`	регистры общего назначения	32
`cc`	код условия	1

Типы команд процессора, различающиеся по набору потребляемых ресурсов. Используются 8 типов команд, резервирующих различные наборы ресурсов. Все команды на 0-м такте резервируют устройство выдачи команд на исполнение `iss`. Таким образом, рассматриваемый процессор способен выдавать на исполнение до двух команд на такт. Ниже приведены сведения о потреблении прочих ресурсов для каждого типа команд.

`c_simple` – не потребляют ресурсов кроме `iss`.

`c_div1` – используют `idivsqrt` на тактах 0 – 4.

`c_div2` – используют `idiv` на тактах 0 – 3.

`c_sqrt` – используют `idivsqrt` на тактах 0 – 6.

`c_mem` – не потребляют ресурсов кроме `iss`.

`c_iadd` – используют `iadd` на такте 0.

`c_imul` – используют `imul` на тактах 0, 1.

`c_imadd` – используют `imul` на тактах 0, 1 и `iadd` на такте 2.

Типы хранилищ данных (регистры, память).

Для целевого процессора поддерживаются хранилища данных следующих видов:

`"D"` – общие регистры в количестве 32.

`"M"` – память, может использоваться без ограничений.

`"V"` – фиктивные значения (типа void), порождаемые, в частности, командами записи в память. Могут использоваться без ограничений.

`"CC"` – регистр кода условия (один).

Набор поддерживаемых операций. В описаниях входных циклических участков программы могут использоваться следующие операции:

`"add"` – сложение,
`"sub"` – вычитание,
`"mul"` – умножение,
`"div"` – деление,
`"neg"` – смена знака числа,
`"sqrt"` – извлечение квадратного корня,
`"if"` – условное копирование значений,
`"cond"` – тестирование значения,
`"var"` – чтение переменной из памяти,
`"const"` – чтение константы из памяти,
`"load"` – чтение из памяти по адресу, находящемуся в регистре,
`"store"` – запись в память по адресу, находящемуся в регистре.

В описании входного участка могут также использоваться следующие ключевые слова:

`"label"` – определяет метку поддерева, по которой можно ссылаться на данное поддерево,
`"look"` – ссылка на поддерево по метке,
`"out"` – отмечает выходные значения.

Команды процессора и правила оптимизации. В таблице 2 приведена сводка команд целевого процессора и правил оптимизации, которые использовались при генерации кода для примеров разд. 4.

Таблица 2. Команды процессора и правила оптимизации.

№	Вычисление/ Комментарий	Команда/тип/ латентность/CC
1	`D0=D1+D2` `Сложение`	**`add`** `D0,D1,D2` `c_add, 2, CC`
2	`D0=D1–D2` `Вычитание`	**`sub`** `D0,D1,D2` `c_add, 2, CC`

3	`D0=D0+D1`
	`D1=D0-D1`
	`Сложение и вычитание`
		`addsub` `D0,D1`
		`c_add, 2`

4	`D0=D1*D2` `Умножение`	**`mul`** `D0,D1,D2` `c_mul, 3, CC`
5	`D0=D0+(D1*D2)` `Умножение с накоплением`	**`madd`** `D0,D1,D2` `c_madd, 3`
6	`D0=(D1*D2)+D0` `Умножение с накоплением`	**`madd`** `D0,D1,D2` `c_madd, 3`
7	`D0=D0-(D1*D2)` `Умножение с вычитанием`	**`msub`** `D0,D1,D2` `c_madd, 3`
8	`D0=-D1` `Смена знака`	**`neg`** `D0,D1` `c_add, 2, CC`
9	`D0=D1/D2 Деление на устройстве divsqrt`	**`div1`** `D0,D1,D2` `c_div1, 5`
10	`D0=D1/D2 Деление на устройстве div`	**`div2`** `D0,D1,D2` `c_div2, 4`
11	`D0=sqrt(D1) Извлечение квадратного корня`	**`sqrt`** `D0,D1` `c_sqrt, 7`
12 13	`D0=2*D1` `D0=D1*2 Оптимизация умножения на 2`	**`add`** `D0,D1,D1` `c_add, 2, CC`
14	`M=D` `Сохранение регистра`	**`save`** `D,M` `c_mem, 1`
15	`D=M Восстановление регистра`	**`restore`** `D,M` `c_mem, 2`
16	`L=D` `Запись переменной`	**`store`** `D,L` `c_mem, 2`
17	`D=L; D=C Чтение переменной или константы`	**`load`** `D,L/C` `c_mem, 2`
18	`D0=D1` `Копирование регистра`	**`move`** `D0,D1` `c_simple, 2`
19	`C=cond(D0) Выработка кода условия`	**`test`** `D0` `c_simple, 1, CC`
20	`D0=CC Копирование кода условия в регистр`	**`move_cc`** `D0` `c_simple, 2`
21	`D0=if`*`U`*`(C,D2,D3)` `Условное копирование`	**`if`***`U`* `D0,D2,D3` `c_simple, 2`
22	`D0=ifU(D1,D2,D3)` `Условное копирование по CC, сохраненному в D1`	**`if`***`U`* `D0,D1,D2,D3` `c_simple, 2`
23	`D1=D1+4; *D1=D0` `Запись с преинкремент.`	**`store`** `D0,++(D1)` `c_mem, 3`
24	`D1=D1+4; D0=*D1` `Чтение с преинкремент.`	**`load`** `D0,++(D1)` `c_mem, 2`
25	`*D1=D0; D1=D1+4` `Запись с постинкремент.`	**`store`** `D0,(D1)++` `c_mem, 3`
26	`D0=*D1; D1=D1+4` `Чтение с постинкремент.`	**`load`** `D0,(D1)++` `c_mem, 2`

Примечания к таблице 2.

1. Вторая строка третьей колонки содержит тип команды и ее латентность. Символ `CC` означает, что команда записывает код условия.

2. Команды 14, 15 (`save`, `restore`) служат для реализации спиллинга – сохранения значения из регистра в память и последующего восстановления. Генератор кода может автоматически генерировать спилл-код при дефиците регистров и учитывать команды сохранения/восстановления при планировании. Эти команды являются синонимами команд `store` и `load`.

3. Команда 18 (`move`) может генерироваться автоматически, если необходимо сделать копию значения, например, когда оно используется как входное несколькими командами, среди которых есть команды, разрушающие его.

Набор команд и правил такой же, как в [4], с добавлением команд чтения из памяти и записи в память с пре- и постинкрементацией адресного регистра.

Рассмотрим пример описания команды `iload` (№ 24 в таблице 2) на языке генератора кода ISched.

```
# Чтение из памяти с преинкрементацией
# адресного регистра
   ("iload", "c_mem",
   #    Операнды команды:
   { "A":(InOut,"D",0,1),
     "R":(Out,"D",2)},
   #    Шаблоны сопоставления
   [(("A",), ("add","A",("const",4))),
    (("R",), ("load",("add","A",
                      ("const",4))))],
   #    Шаблон печати команды
   ("load ","R","++(","A",")"),
   ),
```

В первой строке приведено название команды (`"iload"`) и тип команды (см. выше в этом разделе **Типы команд**). Далее следуют описания двух операндов команды:

`"A"` – адрес в памяти, это входной и выходной операнд (`InOut`), размещающийся на регистре общего назначения (хранилище типа `"D"`); этот операнд считывается на такте `0`, а результат в него записывается на такте `1`.

`"R"` – выходной операнд (`Out`), значение, считанное из памяти, помещается в регистр общего назначения; запись в него происходит на такте `2`.

В разделе «Шаблоны сопоставления» приведены древовидные шаблоны, с которыми сопоставляются операнды `"A"` и `"R"` команды. Если в представлении входного участка программы присутствуют данные шаблоны, то команда `iload` может быть применена для реализации соответствующих вычислений. Заметим, что во втором шаблоне подразумевается исходное значение операнда `"A"`, а не результат действия из первого шаблона.

В разделе «Шаблон печати» приведен шаблон для печати команды в отладочных выдачах и при выводе результата (тела конвейеризованного цикла и его расписания).

4. Примеры генерации кода

В этом разделе рассмотрены примеры программной конвейеризации с подбором команд для нескольких простых циклов. Демонстрируется оптимизация подбора команд, обеспечивающая минимизацию основного интервала (времени выполнения одной итерации). Показано автоматическое использование сложных команд, порождающих более одного результата (чтение из памяти с пре- и постинкрементацией адресного регистра, команда `addsub`), приведены примеры генерации спилл-кода.

Генерация кода поддерживается для циклов с числом повторений, известным до входа в цикл. Тело цикла представлено одним линейным участком (в котором условные вычисления могут быть реализованы при помощи условных пересылок). Предполагается, что процессор содержит команду организации циклов (DO), так что команды инкрементации счетчика цикла, сравнения и условного перехода не генерируются и не учитываются в плане выполнения.

4.1 Скалярное произведение векторов

Рассмотрим пример цикла, реализующего вычисление скалярного произведения векторов:

```
for (i=0; i<N; i++) {
  a = *(++pa)
  b = *(++pb)
  c = c+a*b
}
```

Рассмотрим теперь запись тела цикла на языке генератора кода ISched. Здесь в виде комментариев показаны соответствующие вычисления на C-подобном псевдоязыке. Через `pa'`, `pb'`, `c'` обозначены значения переменных `pa`, `pb`, `c` на предыдущей итерации.

```
Scal_prod_test = (

#   c = c' + a*b
("label", "c", ("out", ("D","M"),
    ("add", ("look", "c", 1),
            ("mul",("look", "a", 0),
                   ("look", "b", 0)))),
#   a = load(pa)
("label", "a",
    ("load", ("look", "pa", 0))),
#   b = load(pb)
("label", "b",
    ("load", ("look", "pb", 0))),

#   pa = pa' + 4
("label", "pa", ("add",
    ("look", "pa", 1), ("const",4))),
#   pb = pb' + 4
("label", "pb", ("add",
    ("look", "pb", 1), ("const",4)))
)
```

Поясним на этом примере смысл элементов синтаксиса, используемых в записи входного циклического участка.

`("label", "c", <дерево>)` – метка и древовидное представление выражения для вычисления значения переменной `c`. По метке `"c"` можно сослаться на это выражение в этом же или другом дереве.

Запись `("look", "c", 1)` в первом поддереве означает ссылку на значение выражения с меткой `"c"` на 1 итерацию раньше. Записи `("look", "a", 0)` и `("look", "b", 0)` в том же дереве означают ссылки на деревья с метками `"a"`, `"b"`, соответствующие значениям `a`, `b` в той же итерации. Таким образом, значения `a`, `b` должны быть вычислены до вычисления `c`.

Деревья с метками `"a"`, `"b"`, в свою очередь, ссылаются на деревья с метками `"pa"`, `"pb"`, то есть значения `pa`, `pb` должны быть вычислены раньше чем `"a"`, `"b"` соответственно. Деревья с метками `"pa"`,

"pb" ссылаются на значения pa, pb на предыдущей итерации.

В записи ("out", ("D","M"), <дерево>) ключевое слово "out" означает, что значение выражения, заданное деревом, является выходным, то есть оно должно быть обязательно вычислено. Список ("D","M") указывает, что результат выражения может быть вычислен на регистре общего назначения либо в памяти.

Поддеревья с головными элементами "load", "mul", "add" описывают операции загрузки из памяти, умножения, сложения с соответствующими операндами.

Решение ЦЛП-задачи для данного цикла было получено при значениях $II = 2$ и $DP = 4$. Соответственно, расписание конвейеризованного цикла требует 2 такта на итерацию, и в его теле совмещаются команды из 4 соседних итераций исходного цикла. В листинге 1 приведен результат планирования с выбором команд для этого цикла.

Листинг 1. Тело конвейеризованного цикла для вычисления скалярного произведения векторов.

```
0: load Db,++(Dpb<<Dpb')  [4]
   add Dc,Dc',Da*b         [1]
1: mul Da*b,Da,Db          [3]
   load Da,++(Dpa<<Dpa')   [4]
```

Здесь в именах виртуальных регистров подстрочными индексами показаны программные переменные или выражения, которым соответствуют регистры. Например, регистр D_{a*b} содержит результат умножения a*b. Штрих указывает на значение переменной от предыдущей итерации. Запись D_{pb}<<D_{pb}' указывает, что команда записывает новое значение регистра на место прежнего. В квадратных скобках для каждой команды показан номер итерации исходного цикла, к которой относится команда в первой итерации конвейеризованного цикла.

На такте 0 выполняется загрузка значения b для итерации 4 с преинкрементацией адресного регистра и сложение $D_c = D_c + D_{a*b}$ для первой итерации. На такте 1 выполняется умножение $D_{a*b} = D_a * D_b$ для 3-й итерации и загрузка значения a для итерации 4, с преинкрементацией адресного регистра. Нетрудно проверить, что ограничения по готовности значений выполняются.

Здесь не показаны пролог и эпилог конвейеризованного цикла, в которые выносятся «недостающие» команды из начальных и конечных итераций. Также при генерации выходного машинного кода потребуется развертка цикла, поскольку время жизни некоторых значений на регистрах составляет более одной итерации. Мы не рассматриваем здесь эти аспекты генерации кода для конвейеризованных циклов, подробное обсуждение которых можно найти в обзоре [5].

Может возникнуть вопрос, почему в этом примере для вычисления значения c=c+a*b не использована команда умножения с накоплением madd? Время ее выполнения (3 такта) меньше чем суммарное время выполнения команд mul (3 такта) и add (2 такта), поэтому традиционный метод подбора команд сгенерировал бы вместо add и mul команду madd.

Однако в данном примере использование madd привело бы к снижению эффективности кода до трех тактов на итерацию вместо двух. Это объясняется тем, что вычисление выражения c=c+a*b при помощи madd создает циклическую (межитерационную) зависимость с латентностью 3, поэтому интервал запуска *II* в этом случае не может быть меньше 3 (подробнее о нижней границе *II* см. в [5]). Использование mul и add вместо madd приводит к тому, что циклическая зависимость переносится на команду сложения, имеющую латентность 2, соответственно, ограничение снизу на *II* также снижается до двух тактов.

4.2 Скалярный квадрат вектора

Рассмотрим пример генерации кода для следующего цикла, демонстрирующий использование команды madd:

```
for (i=0; i<N; i++) {
  a = *(++pa)
  c = c+a*a
```

Если в описании системы команд изменить характеристики команды madd, задав для нее латентность 1 и использование ресурсов только на одном такте, то порождается расписание с $II = 1$ (один такт на итерацию) и $DP = 3$. Тело конвейеризованного цикла включает команду загрузки значения a от 3-й итерации и умножение с накоплением madd $D1_c$<<$D1_c$', $D2_a$,$D2_a$ от первой итерации:

```
0: load Da,++(Dpa<<Dpa')      [3]
   madd Dc<<Dc',Da,Da  [1]
```

Этот и предыдущий примеры демонстрируют важное преимущество предлагаемого метода программной конвейеризации циклов, заключающееся в том, что при подборе команд учитывается не только суммарная стоимость команд, но и возможность эффективного планирования, которая может определяться наличием циклических зависимостей, конфликтами по использованию ресурсов, зависимостями по данным, латентностями команд.

4.3 Вычисление корней квадратных уравнений

Рассмотрим теперь более сложный цикл, реализующий вычисление корней квадратных уравнений, заданных массивами коэффициентов. В данном примере для вычисления корней применяются следующие формулы (см. [4, п.2.2]):

$$D = b^2 - 4ac$$
$$y_1 = b + \sqrt{D}$$
$$y_2 = b - \sqrt{D}$$

$$y = \text{if } (b \geq 0)\, y_1 \text{ else } y_2$$
$$x_1 = (-y)/2a$$
$$x_2 = 2c/(-y)$$

Здесь предполагается, что D ≥ 0. Формула для x_2 получается из стандартной формулы домножением числителя и знаменателя на y_1 или y_2 в зависимости от знака b.

На языке генератора кода ISched запись цикла выглядит следующим образом:

```
Cycle_Big_test = (
#   D = b^2-4ac
 ("label", "D",
  ("sub", ("mul",("look", "b", 0),
                 ("look", "b", 0)),
          ("mul",("const",4),
              ("mul",("look", "a", 0),
                     ("look", "c", 0))))),
#   y1 = b + sqrt(D)
("label", "y1",
  ("add", ("look", "b", 0),
          ("sqrt",("look","D",0)))),
#   y2 = b - sqrt(D)
("label", "y2",
  ("sub", ("look", "b", 0),
          ("sqrt",("look","D",0)))),
#   y = (b>=0) ? y1 : y2
("label", "y",
  ("if","GE",("cond",("look", "b", 0)),
      ("look","y1",0),
      ("look","y2",0))),
#   x1 = (-y)/2a
("label", "x1",
  ("div",("neg",("look","y"s,0)),
         ("mul",("const",2),
                ("look", "a", 0)))),
#   x2 = 2c/(-y)
("label", "x2",
  ("div",("mul",("const",2),
     ("look", "c", 0)),
     ("neg",("look","y",0)))),
#   pa = pa + 4
("label", "pa", ("add", ("look", "pa", 1),
                        ("const",4))),
#   pb = pb + 4
("label", "pb", ("add", ("look", "pb", 1),
                        ("const",4))),
#   pc = pc + 4
("label", "pc", ("add", ("look", "pc", 1),
                        ("const",4))),
#   a = *pa
("label", "a", ("load", ("look", "pa",
0))),
#   b = *pb
("label", "b", ("load", ("look", "pb", 0))),
#   c = *pc
("label", "c", ("load", ("look", "pc", 0))),
#   px1 = px2 + 4
("label", "px1", ("add", ("look", "px2", 1),
                         ("const",4))),
#   px2 = px1 + 4
("label", "px2", ("add", ("look", "px1", 0),
                         ("const",4))),
#   *px1 = x1
 ("label", "outx1", ("out", ("V"),
  ("store", ("look", "px1", 0),
            ("look", "x1", 0)))),
#   *px2 = x2
("label", "outx2", ("out", ("V"),
  ("store", ("look", "px2", 0),
            ("look", "x2", 0)))),
)
```

Общее число изначально сгенерированных команд – 143. Результат конвейеризации с подбором команд получен при II = 9, DP = 4 (листинг 2).

Листинг 2. Конвейеризованный цикл вычисления корней квадратных уравнений.

```
0:  load Da,++(Dpa<<Dpa',)          [4]
    load Dc,++(Dpc<<Dpc',)          [4]
1:  ifGE,Dy = Dy1,Dy1               [2]
    sqrt Dsqrt(D),DD                [3]
2:  mul Dac,Da,Dc                   [4]
    load Db,++(Dpb<<Dpb')           [4]
3:  div2 Dx2,D2c,D-y                [1]
    neg D-y,Dy                      [2]
4:  mul Db^2,Db,Db                  [4]
    add D2a,Da,Da                   [4]
5:  add D2c,Dc,Dc                   [2]
    move D4,4                       [4]
6:  test Db                         [3]
    store Dx1,++(Dpx1<<Dpx2')       [1]
7:  msub DD<<Db^2,D4,Dac            [4]
    store Dx2,++(Dpx2<<Dpx1)        [1]
8:  div2 Dx1,D-y,D2a                [2]
    addsub Dy1<<Db,Dy2<<Dsqrt(D)    [3]
```

Можно видеть, что тело конвейеризованного цикла состоит из 18 команд, и на каждом такте выполняется по 2 команды. При этом в максимальной степени были использованы «составные» команды, вычисляющие 2 результата: чтение и запись с преинкрементацией, умножение с вычитанием `msub`, сложение с вычитанием `addsub`. Единственной «лишней» командой здесь является загрузка константы 4 на регистр в такте 5. Она могла бы быть вынесена из тела цикла, однако в настоящее время эта возможность в генераторе ISched не реализована. В данном случае, впрочем, это не позволило бы добиться ускорения, поскольку для выполнения 17-ти команд все равно потребовалось бы 9 тактов.

Для сравнения было вычислено расписание с подбором команд для линейного участка, соответствующего одной итерации цикла (листинг 3).

Представленное в листинге 3 расписание соответствует расписанию для исходного цикла без конвейеризации. Время выполнения одной итерации исходного цикла – 29 тактов (три последних такта на завершение записи в память можно не учитывать) более чем втрое превышает основной интервал конвейеризованного цикла. Этот пример показывает, что рост производительности кода в результате программной конвейеризации циклов может быть весьма значительным.

Листинг 3. Расписание линейного участка для вычисления корней одного квадратного уравнения.

```
0:  load Db,++(Dpb<<Dpb')
1:  load Da,++(Dpa<<Dpa')
2:  mul Db^2,Db,Db
```

```
        load Dc,++(Dpc<<Dpc')
   3:
   4:   mul Dac,Da,Dc
        add D2c,Dc,Dc
   5:   move D4,4
   6:
   7:   msub DD<<Db^2,D4,Dac
   8:
   9:
   10: sqrt Dsqrt(D),DD
   11:
   12: add D2a,Da,Da
   13:
   14:
   15:
   16:
   17: addsub Dy1<<Db,Dy2<<Dsqrt(D)
   18: test Db
   19: ifGE,Dy=Dy1,Dy2
   20:
   21: neg D-y,Dy
   22:
   23: div2 Dx1,D-y,D2a
        div1 Dx2,D2c,D-y
   24:
   25:
   26:
   27: store Dx1,++(Dpx1<<Dpx2')
   28: store Dx2,++(Dpx2<<Dpx1)
   29:
   30:
   31:
```

4.4 Вектор минимумов из элементов двух векторов

В этом разделе продемонстрированы примеры автоматической генерации спилл-кода. Рассмотрим цикл для вычисления вектора, содержащего минимумы соответствующих элементов двух входных векторов:

```
for (i=0; i<N; i++) {
  a = *(++pa)
  b = *(++pb)
  c = a > b ? b : a;
  *(++pc) = c;
}
```

Запись цикла на языке генератора кода ISched:

```
vec2min_test = (
("label", "a", ("load", ("look", "pa", 0))),
("label", "b", ("load", ("look", "pb", 0))),
("label", "pa", ("add", ("look", "pa", 1),
                        ("const",4))),
("label", "pb", ("add", ("look", "pb", 1),
                        ("const",4))),
("label", "pc", ("add", ("look", "pc", 1),
                        ("const",4))),
("label", "outc", ("out", ("V"),
  ("store", ("look", "pc", 0),
            ("look", "c", 0)))),
("label", "c",
  ("if","GE",("cond",
    ("sub",("look","a",0),("look","b",0))),
  ("look","b",0),("look","a",0)) ),
)
```

Рассмотрим расписания, полученные для этого цикла при различных ограничениях на число доступных регистров.

При наличии 6 или более регистров класса "D" дефицит регистров отсутствует, и мы получаем результат при *II* = 3, *DP* = 3 (листинг 4).

Листинг 4. Результат конвейеризации (6 регистров).

```
0: load Da,++(Dpa<<Dpa')       [3]
   load Db,++(Dpb<<Dpb')       [3]
  Da[2], Db[2], Dpa[2], Dpb[2], Dpc[0], Dc[1]
1: ifGE,Dc=Db,Da               [2]
   store Dc,++(Dpc<<Dpc')      [1]
  Da[2], Db[2], Dpa[3], Dpb[3], Dpc[0],
  Voutc[0], CCa-b[2], Dc[1]
2: sub Da-b,Da,Db              [3]
  Da[3], Db[3], Dpa[3], Dpb[3], Dpc[1]
```

В этом листинге помимо команд для каждого такта показаны также списки занятых регистров. В квадратных скобках показано, к какой итерации относится значение (0 соответствует значению до входа в цикл). Нетрудно видеть, что на каждом такте занято не более 6 регистров класса D.

Если число доступных регистров сократить до 5, то возникает дефицит регистров. Решение для этого случая получено при *II* = 4, *DP* = 2 (листинг 5).

Листинг 5. Результат конвейеризации (5 регистров).

```
0:  load Da,++(Dpa<<Dpa')    [2]
    load Db,++(Dpb<<Dpb')    [2]
  Db[1], Dpa[1], Dpb[1], Dpc[0], CCa-b[1]
1:  ifGE,Dc=Db,Da            [1]
  Da[1], Db[1], Dpa[2], Dpb[2], Dpc[0],
  CCa-b[1]
2:  sub Da-b,Da,Db           [2]
  Da[2], Db[2], Dpa[2], Dpb[2], Dpc[0],
  Voutc[0]
3:  load Da,Dpa              [2]
    store Dc,++(Dpc<<Dpc')   [1]
  Db[2], Dpa[2], Dpb[2], Dpc[0], Dc[1]
```

В этом расписании присутствует спиллинг значения a. Поскольку a хранится в памяти, то спиллинг реализуется как повторная загрузка этого значения (команда load в такте 3). Значение a используется дважды: в команде вычитания (такт 2) и в условном присваивании (такт 1). Повторная загрузка позволяет высвободить соответствующий регистр в промежутке между двумя использованиями значения a.

При наличии 4 регистров используется спиллинг регистра Dpa. Решение получено при *II* = 5, *DP* = 2 (листинг 6).

Листинг 6. Результат конвейеризации (4 регистра).

```
0:  restore Dpa,Mpa          [1]
   Da[1], Db[1], Mpa[1], Dpb[1], Dpc[0]
1:  ifGE,Dc=Db,Da            [1]
  Da[1], Db[1], Dpb[1], Dpc[0], Vcout[0],
  CCa-b[1]
2:  load Da,++(Dpa<<Dpa')    [2]
    load Db,++(Dpb<<Dpb')    [2]
  Dpa[1], Dpb[1], Dpc[0]
```

```
3:  save Dpa,Mpa            [2]
    store Dc,++(Dpc<<Dpc')      [1]
  Dpa[2], Dpb[2], Dpc[0]:1   Dc[1]
4:  sub Da-b,Da,Db          [2]
  Da[2], Db[2], Mpa[2], Dpb[2], Dpc[1]
```

При наличии 3 регистров осуществляется спиллинг двух регистров – D_{pa} и D_{pb}; решение получено при $II = 7$, $DP = 1$. Если доступно только 2 регистра, то спиллинг выполняется уже для 3 регистров: D_{pa}, D_{pb} и D_{pc}, решение получено при $II = 8$, $DP = 1$. Глубокая программная конвейеризация увеличивает потребность в регистрах, и при их нехватке оказывается невыгодной, поэтому она и не была выполнена при наличии 3-х и менее регистров ($DP = 1$ означает, что совмещения команд из разных итераций исходного цикла нет). Тем не менее, были получены оптимальные расписания цикла, автоматически выбраны регистры для спиллинга, сгенерированы и включены в расписание соответствующие команды `save/restore`.

5. Формулировки ЦЛП-задач и их сложность

Рассмотрим основные аспекты метода программной конвейеризации циклов с выбором команд и ограничения, используемые в формулировках ЦЛП-задач, на примере цикла из разд. 4.1 (вычисление скалярного произведения векторов)

```
for (i=0; i<N; i++) {
  a = *(++pa)
  b = *(++pb)
  c = c+a*b
}
```

На шаге 3 (см. п. 2.2) для этого примера было сгенерировано 38 команд. Напомним, что генерируются все возможные команды из набора команд процессора, которые в принципе могут быть использованы для вычисления значений. Рассмотрим, например, набор команд, который генерируется для вычисления значения переменной `a`:

```
load Da,Dpa
iload Da,Dpa
save Da,Ma
restore Da,Ma
move Da,Da'
```

Здесь идентификаторы D_a, D_{pa} обозначают виртуальные регистры, соответствующие хранению `a` и `pa` на регистрах общего назначения, M_a – виртуальный регистр хранения `a` в памяти). Здесь `load` – обычная загрузка значения `a` из памяти, `iload` – загрузка с преинкрементацией. Команда `save` позволяет получить копию `a` в памяти, `restore` восстанавливает `a` из памяти в регистр, эти команды реализуют спиллинг. Команда `move` позволяет получить копию `a` в другом регистре. В результирующем коде для вычисления значения `a` может быть использована любая из этих команд. В этом наборе все команды, кроме `save`, вычисляют `a` в регистре (D_a); `save` вычисляет `a` в памяти (M_a). На практике в этом примере для вычисления `a` была использована команда `iload` (см. листинг 1 в разд. 4.1).

Построение ЦЛП-задач для конвейеризации циклов с подбором команд во многом схоже с построением ЦЛП-задач для планирования линейных участков с выбором команд [3]. Одна итерация исходного цикла рассматривается как линейный участок, и делается попытка спланировать его, используя количество тактов, равное $II * DP$.

На Рис. 2 показан граф зависимостей для некоторого возможного набора команд, реализующего данный цикл. В скобках для каждой зависимости показаны ее латентность и дистанция.

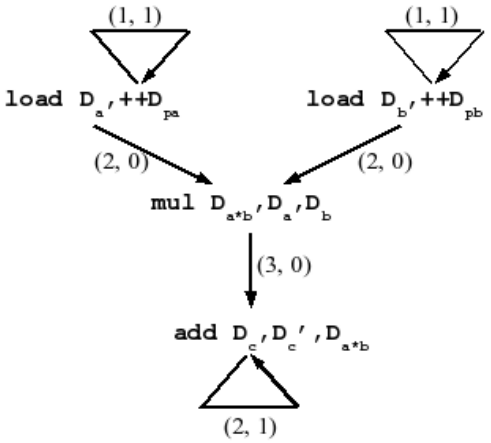


Рис. 2. Граф зависимостей для цикла, вычисляющего скалярное произведение векторов

На рис. 3 в столбце I_1 показано расписание одной итерации исходного цикла, удовлетворяющее зависимостям по данным. В рамке выделено тело конвейеризованного цикла, полученное совмещением команд из 4 соседних итераций исходного цикла (см. также листинг 1 в разд. 4.1). Здесь показан только один из возможных наборов команд для этого цикла и одно возможное расписание. В действительности во время решения ЦЛП-задачи происходит перебор возможных наборов команд и расписаний для них. При этом проверяются условия, заданные в формулировке ЦЛП-задаче: наличие команд, вычисляющих все требуемые значения, соблюдение зависимостей по данным, а также ограничений по ресурсам и числу регистров.

В ЦЛП-задаче генерации кода для циклов ограничения, связанные с учетом зависимостей по данным, отличается от аналогичных ограничений в задаче для линейного участка тем, что приходится учитывать циклические зависимости (с ненулевой дистанцией). В частности, дистанция 1 означает, что команда, вычисляющая входное значение, относится к предыдущей итерации, и это должно быть учтено при записи ограничений. Из-за этого приходится увеличивать размер массива переменных, описывающих готовность значений.

Еще одно отличие формулировки ЦЛП-задачи для циклов от формулировки аналогичной задачи для линейного участка касается ограничений по потреблению ресурсов (функциональных устройств и

аппаратных регистров). При суммировании потребностей в каждом виде ресурсов нужно учитывать, что команды исходного цикла, время запуска которых отличается на *DP*, выполняются одновременно, а значит, их потребности в ресурсах должны суммироваться.

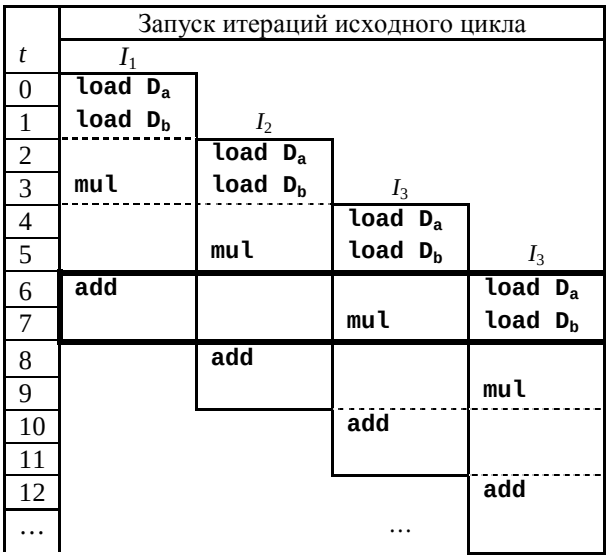


Рис. 3. Планирование циклического участка кода «по модулю»

Сложность ЦЛП-задач зависит от размера входного цикла. Для небольших циклов, таких как примеры из разд. 4.1, 4.2, 4.4, были использованы солверы свободно доступные в исходных текстах. А именно, пакеты GLPK [12] и более эффективный CBC [13], входящий в состав проекта COIN-OR [14]. Для цикла из разд. 4.3 (решение квадратных уравнений) мощности этих солверов оказалось недостаточно. Сформированные для этого цикла задачи содержат 5000 – 7000 переменных, 7000 – 9000 ограничений с 30000 – 40000 ненулевых элементов.

Для этого и других сложных циклов были использованы возможности сервера NEOS [15], предоставляющего сервис решения различных видов задач математического программирования, включая задачи ЦЛП, с использованием ряда коммерческих солверов. Наилучшие результаты продемонстрировали солверы Gurobi [16] и особенно XpressMP, являющийся частью пакета FICO™ Xpress Optimization Suite [17]. С их помощью для цикла из разд. 4.3, решения при $II = 9$, $DP = 4$ вычислялись за время 3-5 минут.

Для сравнения можно рассмотреть две более простые задачи генерации кода.

Первая задача – планирование с подбором команд для линейного участка [4]. Решение этой задачи для программы вычисления корней одного квадратного уравнения представлено в разд. 4.3 (листинг 3). Построенное расписание занимает 32 такта. Хотя 32 такта не намного меньше чем число тактов, рассматриваемых в аналогичной задаче программной конвейеризации цикла ($II * DP = 9 * 4 = 36$, см. листинг 2), ЦЛП-задача для линейного участка решается во много раз быстрее. Это говорит о том, что задача планирования линейного участка по своей природе существенно проще чем задача планирования циклического участка.

Вторая задача – программная конвейеризация цикла без выбора команд (в случае, когда выбор команд выполнен заранее). Соответствующая ЦЛП-задача оказывается проще уже потому, что в несколько раз сокращается размер рассматриваемого набора команд. Кроме того, в этом случае используется требование о том, что каждая команда должна быть запланирована в цикле ровно один раз. Это позволяет построить существенно более эффективную формулировку ЦЛП-задачи [10], в которой количество переменных и неравенств зависит только от *II*, но не от глубины конвейеризации *DP*. Глубина конвейеризации определяется из решения ЦЛП-задачи, так что к тому же отпадает необходимость в переборе ЦЛП-задач для разных значений *DP*.

Обе эти более простые задачи для данного примера (нахождение корней квадратного уравнения) легко решаются с использованием свободно доступных солверов.

6. Заключение

В работе рассмотрен метод программной конвейеризации циклов с выбором команд путем сведения этой задачи к задаче математического программирования. Разработана формулировка ЦЛП-задачи выбора и планирования команд с ограничениями по числу регистров для циклических участков кода при фиксированных значениях интервала запуска *II* и глубины конвейеризации *DP*.

Решением данной ЦЛП-задачи, если оно существует, является набор команд (из системы команд целевого микропроцессора), реализующих заданный программный цикл, и план их выполнения, обеспечивающий запуск последовательных итераций с заданным интервалом *II* тактов, при соблюдении ограничений на число аппаратных регистров. При этом в результирующем цикле могут использоваться команды и значения из диапазона не более чем *DP* соседних итераций исходного цикла. Поиск оптимального кода для заданного входного цикла осуществляется путем перебора по числу тактов $II = 1, 2, \ldots$, а для каждого *II* также по величине $DP = 1, 2, \ldots DP_{max}$, до тех пор, пока для некоторой пары значений *II', DP'* не будет найдено решение соответствующей ЦЛП-задачи.

Предложенный метод позволяет осуществлять выбор и планирование команд для циклических участков кода с учетом различных факторов:

- возможность параллельного исполнения команд;
- стоимость (латентность) команд;
- наличие межитерационных зависимостей;
- наличие общих подвыражений;
- наличие специализированных вычислительных команд;
- число доступных аппаратных регистров.

При этом в случае дефицита аппаратных регистров автоматически генерируются и включаются в расписание команды спиллинга.

Развитием предложенного подхода может быть дополнение представленного в разд. 2.2 процесса генерации кода применением математических тождеств, включая законы коммутативности, ассоциативности и дистрибутивности, формулу разности квадратов и т.п. Как показали эксперименты с применением математических тождеств в генерации кода для линейных участков, это позволяет получить значительный прирост производительности кода. Однако использование тождественных преобразований для циклических участков может привести к чрезмерному усложнению задач ЦЛП, поэтому на практике при конвейеризации циклов возможно лишь ограниченное применение тождественных преобразований.

В силу сложности реализации рассмотренного подхода и высоких затрат вычислительных ресурсов, области его применимости ограничиваются разработкой генераторов высокопроизводительного кода для процессоров, используемых во встроенных системах (как, например, в [7]), а также аппаратных ускорителей.

Software Pipelining of Loops with Instruction Selection

N. I. Viukova, V.A. Galatenko, S.V. Samborskij

Abstract: The paper presents a method of software pipelining of loops by means of exact joint solution of the tasks of instruction selection and instruction scheduling for cyclic blocks of program code. We propose an approach of treating the above tasks as a single Integer Linear Programming (ILP) task. Combining instruction selection with instruction scheduling allows one to take into account the factors of loop dependencies, possibility of parallel execution of instructions, availability of specialized computing instructions, presence of common subexpressions, restrictions on the number of available hardware registers etc. The method proposed may be useful in development of hardware accelerators and in generation of high performance code for processors used in embedded systems.

Keywords: code generation, software pipelining, instruction selection, instruction scheduling, integer linear programming (ILP).

Литература

1. V.A. Galatenko, S.V. Samborskij, N.I. Viukova. Code generation as a mathematical programming task. // ADVANCED MATHEMATICS, COMPUTATIONS AND APPLICATIONS – 2014. (AMCA-2014). http://conf.nsc.ru/files/conferences/amca14/241897/AMCA_abstracts.pdf, p. 56.

2. Н.И. Вьюкова. Генерация кода с отложенным выбором инструкций. – М.:НИИСИ РАН, 2010, С. 80-96.

3. Н.И. Вьюкова, В.А. Галатенко, С.В. Самборский. Совместное решение задач выбора и планирования команд в условиях дефицита регистров. // Программная инженерия, № 2, 2012, С. 35-41.

4. Н.И. Вьюкова, В.А. Галатенко, С.В. Самборский. Генерация кода методом точного совместного решения задач выбора и планирования команд. – // Программная инженерия, № 6, 2014, С. 8-15.

5. Н.И. Вьюкова, В.А. Галатенко, С.В. Самборский. Программная конвейеризация циклов методом планирования по модулю. М.: – // Программирование, № 6, 2007г., С. 14-25.

6. A. Stoutchinin. An Integer Linear Programming Model of Software Pipelining for the MIPS R8000 Processor. // Proceedings of the 4th International Conference on Parallel Computing Technologies (PaCT '97). September, 1997. P. 121-135.

7. Н.И. Вьюкова, В.А. Галатенко, С.В. Самборский. Программная конвейеризация циклов для ускорителя плавающей арифметики в составе процессора Комдив128-РИО. // Программные продукты и системы № 4, 2013, С. 35-43.

8. C.W.Fraser, R.R.Henry, T.A.Proebsting. BURG – Fast optimal instruction selection and tree parsing. // SIGPLAN Notices 27, 4 (Apr. 1992), p. 68-76.

9. Н.И. Вьюкова, В.А. Галатенко, С.В. Самборский. К проблеме выбора машинных инструкций в генераторах кода. – // Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО. Сборник статей – М.:НИИСИ РАН, 2009, С. 3-31.

10. С.В. Самборский. Формулировка задачи планирования линейных и циклических участков кода. // Программные продукты и системы. № 3, 2007, С. 12-16.

11. Язык программирования Python. http://www.python.ru/

12. GNU Linear Programming Kit. http://www.gnu.org/software/glpk.

13. CBC User Guide. http://www.coin-or.org/Cbc/cbcuserguide.html.

14. COmputational INfrastructure for Operations Research, open source for the operations research community, http://www.coin-or.org/.

15. NEOS Server: State-of-the-Art Solvers for Numerical Optimization. http://www.neos-server.org/neos .

16. Gurobi Optimization. www.gurobi.com.

17. FICO™ Xpress Optimization Suite. http://www.fico.com/en/products/fico-xpress-optimization-suite.

Описание функциональных возможностей среды FTStudio для разработки кроссплатформенных функциональных тестов СБИС

К.К. Смирнов

Аннотация: Рассматриваются вопросы автоматизации подготовки функциональных тестов в среде FTStudio (Functional Test Studio). Применение предложенного в статье подхода позволяет существенно снизить трудоемкость подготовки комплекта функциональных тестов микросхем под различное измерительное оборудование.

Ключевые слова: среда FTStudio, автоматизация тестов, оборудование

1. Введение

Разработка функциональных тестов – одна из основных задач в процессе проектирования и производства полупроводниковых СБИС. Функциональные тесты позволяют выявить случайные скрытые дефекты микросхем и чувствительность элементов СБИС к воздействию внешних факторов. Функциональный контроль позволяет проводить проверку СБИС в условиях, близких к эксплуатационным на основе совокупности тестовых таблиц, проверяющих основные функциональные параметры микросхем и составленных по заданному алгоритму.

Функциональные тесты используются как для проверки функционирования микросхем так и для задания режимов при проведении отбраковочных и периодических испытаний. На ряде операций, таких как электротермотренировка или испытание на влагоустойчивость применяется групповое тестирование микросхем, при котором входные воздействия для микросхем совпадают (рисунок 1).

Часто разработчики функциональных тестов сталкиваются с проблемами переноса тестов или алгоритмов тестирования на различные тестирующие платформы. Это связано с большим разнообразием форматов файлов тестовых воздействий под различное оборудование, а также наличием различных требований и ограничений, обусловленных возможностями оборудования и разработанной оснасткой. Решить подобные проблемы позволяет кроссплатформенное формирование функциональных тестов.

Кроссплатформенный (межплатформенный) функциональный тест — функциональный тест, работающий более чем на одной аппаратной платформе. Типичным примером является функциональный тест, работающий на различном оборудовании и предназначенный для проведения функционального контроля, контроля в процессе проведения электротермотренировки и других испытаниях одновременно.

Разработанная среда разработки программ функционального контроля FTStudio [1] содержит в своем составе программные инструменты по созданию, адаптации, конвертации и отладке функциональных тестов.

Благодаря модульной архитектуре, FTStudio является расширяемой средой с возможностью быстрого включения поддержки новых форматов файлов тестовых воздействий и возможности работы с различным типом испытательного оборудования.

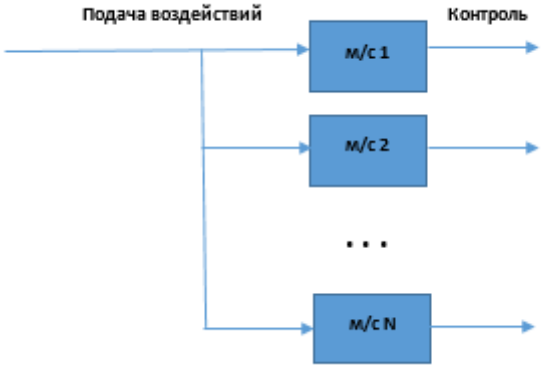


Рисунок 1. Функциональная схема процесса тестирования микросхем при проведении операции электротермотренировки и испытания на влагоустойчивость

В настоящее время в состав FTStudio включена поддержка логического анализатора Agilent 93000, оборудования для проведения электротермотренировки AEHR MAX-3B, Блоков задания режима для проведения испытаний СБИС СОЗУ 4Мбит, 8Мбит, 16Мбит и блока функционального контроля СБИС СОЗУ.

Для добавления поддержки нового оборудования с целью обеспечения возможности отладки разработанных или адаптированных функциональных тестов в среде FTStudio имеются встроенные возможности для разработки драйверов нового оборудования. Разработанный драйвер совместим с Тест-Мониторной системой KernelTest [2], используемой в качестве универсальной среды для проведения функционального и диагностического контроля микросхем.

Для включения поддержки новых форматов файлов тестовых воздействий для FTStudio разработан специальный язык описания, позволяющий непосредственно в редакторе кода описывать структуру файла с тестом.

При подготовке тестов в виду различных методик тестирования[3,4] и различного числа тестируемых каналов на оборудовании, разработчик нередко сталкивается с задачами параллельного тестирования нескольких микросхем или модульного разбиения функционального теста по числу поддерживаемых каналов. Автоматизация решения подобных задач может быть максимально гибко выполнена в среде FTStudio.

Описание алгоритма в среде FTStudio ведется на языке описания тестовых воздействий STeeL, разработанном автором. Данный язык включает в себя инструменты языка описания аппаратуры и языка программирования, основанного на синтаксисе C++ и C#. Данный язык поддерживает как модульную так и объектные модели создания функциональных тестов. При модульном подходе разработчик может формировать для последующего использования библиотеки алгоритмов, а при объектном подходе импортировать как функциональный элемент ранее созданный тест на языке STeeL либо тестовую таблицу в формате ASCII. При работе с функциональными элементами разработчик имеет возможность исключать некоторые сигналы, комбинировать, а также совершать с ними логические операции (инверсия, конъюнкция и проч.) и устанавливать различные сложные зависимости, такие как циклы ожидания.

Рассмотрим процесс подготовки кроссплатформенного теста на примере теста памяти для СБИС СОЗУ с различным числом шин адреса(AD) и данных (DATA):

СОЗУ	AD (разрядов)	DATA (разрядов)
1Мбит	17	8
4Мбит	19	8
8Мбит	19	16
16Мбит	19	32

Для каждой СБИС создаются следующие файлы тестовых воздействий:

- для проведения функционального контроля на логическом анализаторе Agilent 93000;
- для проведения электротермотренировки на оборудовании AEHR MAX-3B;
- для проведения испытаний при помощи блока задания режимов без контроля тока потребления;
- для проведения испытаний при помощи блока задания режимов с контролем тока потребления;
- исследовательский тест микросхемы СОЗУ 16Мбит на оборудовании, обеспечивающем контроль только 16 цифровых сигналов.

При разработке тестов необходимо учесть следующие особенности испытательного оборудования:

1. для Agilent 93000 – режимы чтения/записи на контрольных выводах (nce/noe/nwe) кодируются специальными символами для реализации специфических форм сигнала (waveform);
2. для MAX-3B –режим тестирования одновременно 10 микросхем, с контролем не более трех сигналов, подачей на микросхему не более 12 сигналов и ограничением на количество тестовых циклов, равным 1000000;
3. Для решения данной задачи потребуется разработать 4 файла с описанием формата файлов тестовых воздействий для блоков задания режима, 1 файл с описанием алгоритма на языке STeeL, 1 файл на языке STeeL, описывающий подключение микросхем во время проведения электротермотренировки и 2 файла описывающих схемы разбиения функционального теста по шине данных (в первом будут контролироваться младшие 16 бит, а во втором – старшие).

2. Методика подготовки файлов тестовых воздействий

Огромное разнообразие форматов файлов тестовых воздействий и отсутствие универсальных методик для отладки тестов на различном оборудовании приводит к тому, что разработчик вынужден либо разрабатывать отдельную программу тестирования для каждого оборудования либо разрабатывать конвертеры из одного формата в другой. В виду индивидуальных особенностей тестового оборудования, такие как максимальный размер файла, наличие или отсутствие поддержки управляющих команд, разработка комплекта функциональных тестов становится весьма трудоемкой задачей (рисунок 2).

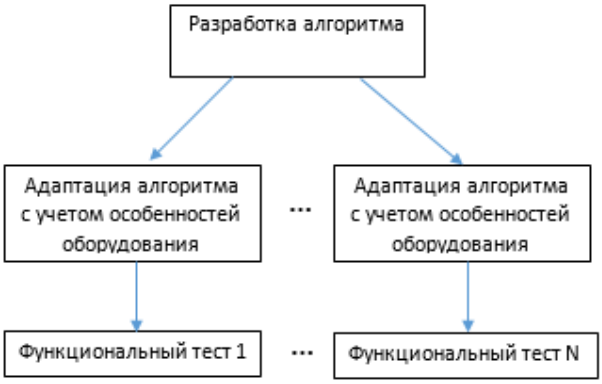


Рисунок 2. Традиционный подход при подготовке комплекта функциональных тестов

Предлагаемая методика предполагает единовременную подготовку данных, описывающих все имеющиеся на предприятии тестовое оборудование, используемое для функционального тестирования и использование возможностей среды FTStudio по разработке кроссплатформенных функциональных тестов (рисунок 3). При этом описание всех требований к функциональному тесту описываются в самом алгоритме, что упрощает внесение последующих изменений.

В качестве исходных данных для разработки набора функциональных тестов выступают:

1. файл описания формата тестовых воздействий;
2. алгоритм теста на языке STeeL либо импортированная тестовая таблица;
3. файл соответствия каналов тестового оборудования с названиями сигналов микросхемы. Для оборудования у которого названия каналов соответствуют названиям сигналов данный файл не создается.

Алгоритм теста на языке STeeL может быть разработан в редакторе среды FTStudio, который содержит в своем составе инструменты ускоренной разработки:

Подсветка синтаксиса, механизмы автозавершения кода, подсказки по ключевым словам и средства отладки и симуляции функциональных тестов.

Рисунок 3. Кроссплатформенный подход при разработке комплекта функциональных тестов в FTStudio

Для создания набора тестов в среде FTStudio создается файл проекта в который импортируются все исходные данные, указываются все необходимые зависимости и какие типы файлов выходных данных необходимо получить (рисунок 4).

На следующем этапе проект компилируется. Результатом компиляции является объектный двоичный файл в формате FTO (FTStudio Objects). В этот файл компилятор помещает созданную на основе алгоритма тестовую таблицу, информацию о сигналах и метаданные, необходимые для создания тестов под конкретное оборудование.

Полученный FTO-файл можно открыть в FTStudio в режиме симуляции (рисунок 5) или в режиме. просмотра тестовой таблицы (рисунок 6).

На основе полученного FTO-файла создаются исследовательские и расширенные тесты для функциональ-

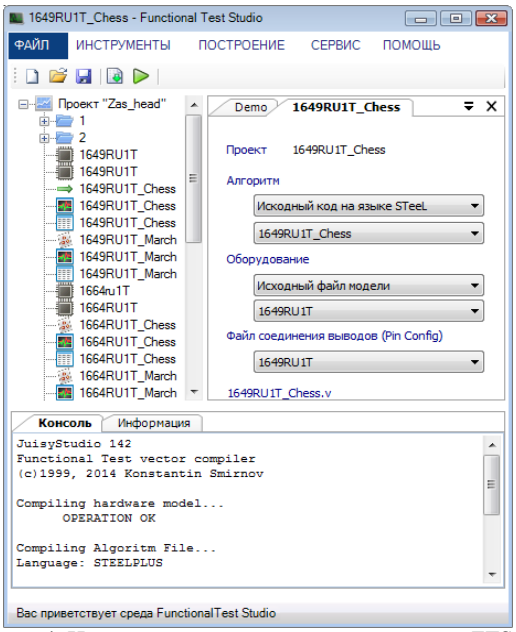
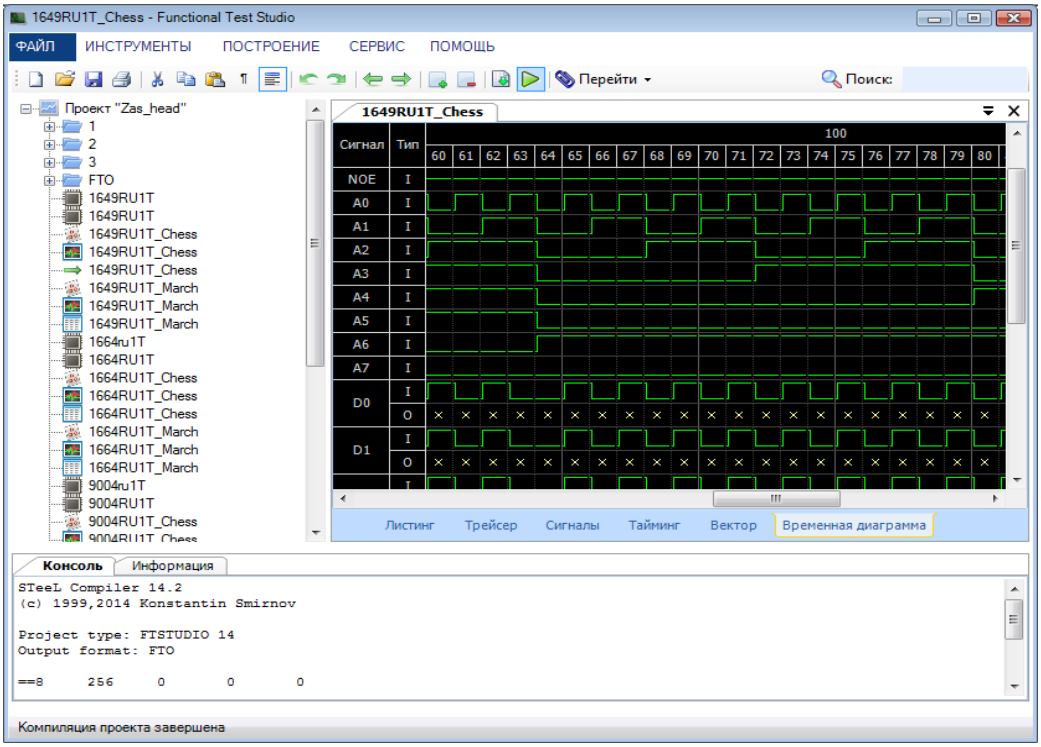


Рисунок 4. Настройка параметров проекта в среде FTStudio

ного контроля СБИС в процессе проведения электротермотренировки а также производится сборка всего проекта по правилам, описанным в файле-сборки(make file).

На последнем этапе разработчик теста имеет возможность запуска созданных файлов тестовых воздействий на оборудовании и оценки корректности их работы. Запуск теста возможен только на оборудовании, для которого в FTStudio установлен соответствующий драйвер. Драйвер может быть разработан как



.Рисунок 5. Режим симуляции в среде FTStudio

средствами FTStudio, так и в любой среде разработки .NET приложений.

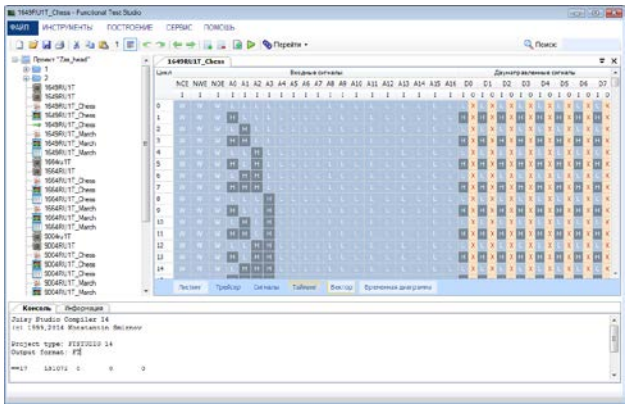


Рисунок 6. Режим просмотра тестовой таблицы в среде FTStudio

3. Описание формата файла тестовых воздействий

Файл, описывающий формат тестовых воздействий представляет собой текстовый файл и в терминологии FTStudio называется моделью вектора. Данный файл состоит из четырех разделов.

1. Раздел настроек формата. В этом разделе разработчик тестов может указать параметры компиляции алгоритма тестирования при создании теста в описываемом формате, а также максимальный размер файла, возможность автоматического разбиения теста на несколько файлов и задать поддержку заголовка.

   Пример описания:

   ```
   ; ключ условной компиляции теста
   .define BZR
   ; формат выходного файла (stv-двоичный,
   ASCII –текстовый)
   .vectortype stv
   ;максимальный размер поддерживаемого
   файла
   .maxfilesize 100
   ;Разделять текст на несколько файлов?
   .splitvector NO
   ;включение заголовка в файл
   .header YES
   ;Необязательный параметр указывает
   ;какое оборудование  использовать при
   ;отладке данного теста
   .driver  BZR_SOZU4M
   ```

2. Раздел описания заголовка. Данный раздел игнорируется компилятором, если в разделе описания формата не включена поддержка заголовка. В заголовке разработчик может задать любые произвольные данные в различных форматах (Символ, Строка, Число) и вычисляемые данные. Числа могут быть заданы в десятичной, двоичной и шестнадцатеричной системах. В качестве вычисляемых данных компилятор может указать вычисленные значения размера заголовка, размер тестового цикла, число требуемых логических каналов оборудования, формат тестового цикла, число сигнальных каналов и каналов инструкций.

   В рамках решения задачи подготовки комплекта тестов памяти, заголовок в файле тестовых воздействий требуется только для блока задания режимов.

   Пример описания заголовка:

   ```
   ;вставляем строковый идентификатор
   chars STV
   ;вставляем последовательность байт
   bytes 12, 100, 121, 55
   ;вставляем 8 нулевых байт
   nop 8
   ;вставляем размер заголовка
   var header.size
   ;вставляем размер структуры
   ;описания цикла
   var cycle.format.size
   ;вставляем структуру описания цикла
   var cycle.format
   ; вставляем последовательность байт
   bin 00001010, 01011001
   ; вставляем последовательность байт
   hex FF, A1, DA
   ```

3. Раздел перечисления каналов оборудования. В этом разделе разработчик указывает в каком порядке в цикле идут данные для каждого сигнала и формат этих данных. Формат данных может быть задан как для группы сигналов, так и индивидуально для каждого. Каналы задаются либо группами либо по одному с указанием имен.

   Пример:

   ```
   ;Каналы объявляются в той
   ;последовательности, как они будут
   ;следовать в цикле тестовой
   ;последовательности

   ; Канал, предназначенный только для
   ;входных сигналов
   channel ch1, IN
   ; Канал, предназначенный только для выходных сигналов
   channel ch2, OUT
   ; Канал, участвующий в формировании
   управляющих  команд
   channel code1, opcode
   ;Канал для работы с двунаправленными
   сигналами
   ```

```
channel ch3, IO
; Задаем для всех последующих сигналов
;новый формат
; n – направление (вход=1, выход=0),
; x - Отсутствие контроля
;или Z-состояние, s -значение сигнала
attribute nxs
;Канал, участвующий в формировании
;управляющих  команд
channel code2, opcode
;Канал, участвующий в формировании
;управляющих  команд
channel code3, opcode
;Будет создано 20 каналов AD0-AD19
group 20, AD*, IO
```

4. Раздел описания управляющих команд. В данном разделе описываются все управляющие команды и соответствие подачи логических уровней на управляющие каналы, объявленных в предыдущем разделе. Допускается два способа объявления соответствия – по значениям или по названиям управляющих каналов. В первом случае перечисляются значения логических уровней для всех управляющих каналов в порядке их объявления, во втором – перечисление имен управляющих каналов, для которых необходимо установить логическую единицу.

Пример описания управляющих команд:

```
;Первый способ – задание значений
;управляющих каналов
opcode Break=001
;Второй способ – перечислить через
;запятую имена каналов, для
;которых нужно установить
;единицу
opcode ICCmeas={code1, code3}
;равносильно написанию =101
```

Все описанные управляющие команды автоматически доступны из алгоритма функционального теста на языке STeeL, а их вызов равносилен вызову функции. Так, при создании файла тестовых воздействий из алгоритма могут быть вызваны две функции Break() и ICCmeas(), которые будут автоматически транслированы в установку соответствующих логических уровней на управляющих каналах тестового оборудования. Так как файлы моделей создаются под каждую единицу оборудования, разработчик имеет возможность задать под каждое оборудование свой алгоритм трансляции управляющих команд, что дает максимальную гибкость разработки кроссплатформенных функциональных тестов.

4. Разработка функционального теста памяти

Для разработки кроссплатформеннных функциональных тестов для рассмотренных выше трех СБИС СОЗУ, отличающихся размером шины данных необходимо задать внешнюю константу DATASIZE, которая при компиляции алгоритма будет принимать значения, указанные в файле сборки тестов.

Пример разработки теста памяти описан в [1], поэтому для простоты изложения воспользуемся встроенным в FTStudio универсальным тестом Chess (Checkerboard [5]), расположенным в библиотеке MemoryTools. Далее опишем входные управляющие сигналы NCE, NWE, NOE, шины AD и DATA. Далее формируем тестовые циклы функционального теста путем последовательного перехода по всем циклам теста Chess. При подобном подходе разработчик тестов имеет возможность объединения различных алгоритмов тестирования в рамках одного теста либо возможность формировать единый тест для одновременного тестирования нескольких функциональных блоков СБИС. Внутри тестового цикла разработчику доступна переменная *cycle*, в которой содержится номер текущего цикла. Анализируя значение этой переменной разработчик имеет возможность вносить дополнительные задержки в тест, варьировать состояния сигналов на определенных циклах или выполнять различные действия, например вызов управляющей команды для измерения тока на определенном цикле.

Пример реализации кроссплатформенного теста памяти:

```
#usingcomp “MemoryTools”
#steel
  //указываем компилятору использовать константы,
  // объявленные в файле-сборки (makefile)
  ext ADSIZE
  ext DATASIZE
  //Задаем управляющие сигналы и объединяем
  //их в группу CTRLS
  group "UPR"
  pin ["NCE", "NWE","NOE"], IN, CTRLS
  //Задаем шины адреса и данных
  bus "AD","A",IN, ADSIZE
  bus "DATA","D",IO, DATASIZE
#end
//алгоритм теста
function Test()
   //создаем экземпляр теста памяти из библиотеки
   //MemoryTools и подключаем к нему сигналы
   //тестируемой микросхемы
   #comp chess from
            MemoryTools.ChessTest(AD,  DATA, CTRLS )
      //проходим все циклы теста памяти “chess”
      #testcycle(chess.CyclesCount)
          //переходим на очередной цикл в тесте
          //памяти и меняем состояние сигналов шин AD,
          // DATA, CTRLS в соответствии с алгоритмом
          //теста
          next chess
```

```
        ifdef BZR
          //данный код будет выполнен только для БЗР
            ifdef ICCMEAS
              //если задан ключ условной компиляции
              //для измерения тока, то сформировать
              //управляющую команду,
              //объявленную в файле модели БЗР
              #if (cycle==1000) ICCmeas();
            endif
        endif
        ifdef ETT
         //в виду ограничения на длинну теста ЭТТ,
         //равной 1000000 выполним принудительный
         //выход из цикла на цикле №100000
         #if (cycle==1000000) break;
        endif
    #end
TestEnd
```

В данном примере, в случае запуска теста на блоке задания режимов, при достижении 1000 цикла будет сформированы высокие логические уровни на сигналах code1 и code2, разрешающих запуск функции измерения тока.

Разработчик имеет возможность задания более сложных условий запуска управляющих команд:

1. посредством цикла ожидания, при котором указываются состояния определенных сигналов либо сигналов шины;

2. анализа состояний конкретного сигнала. В этом случае команда будет выполнена при последовательной установке сигнала в определенные состояния.

Продемонстрируем задание указанных сложных условий на следующем примере:

```
//Вызов управляющей команды Break (code1=0,
//code2=0, code3=1) и установка на выводе test низкого
//логического уровня при //последовательном появлении
//на младшем разряде шины DATA состояний “01011”

#wait DATA[0], “01011”

   #Break();
       form "test" =  $L

#end

//Установка значения номера текущего тестового
//цикла при установке  адреса на шине AD равным
//11110000

#wait AD, “11110000”
        form DATA = cycle.ToSerial(cycle)

#end
```

5. Разработка теста для проведения электротермотренировки

Рассмотрим пример создания нового теста для функционального контроля 10 СБИС СОЗУ при проведении электротермотренировки на основе разработанного функционального теста.

В этом тесте, ввиду ограничения оборудования на число тестовых каналов, на каждую СБИС будут подаваться только 9 адресных разрядов. На разрядах 10..18 будут установлены высокие уровни непосредственно на тестирующей оснастке. Ввиду особенностей применяемого алгоритма теста Checkerboard [5] с целью обращения теста ко всем банкам тестируемой памяти выбираем разряды 5..9, адресующие банки памяти, младшие два разряда, адресующие часть столбцов и разряды 10,11, адресующие часть строк. На остальных разрядах будут установлены высокие уровни непосредственно на тестирующей оснастке. Для упрощения, в тексте алгоритма разрядам, отвечающим за адресацию строк и банкам присвоим номера 0..6, а разрядам, адресующим столбцы – номера 7 и 8. Контролировать будем только сигналы D0, D5 и D7. С целью предотвращения появления “холостых” циклов и правильности работы алгоритма необходимо при импорте ранее созданного теста явно указать размерность шины адреса.

Представим задачу в виде функциональной схемы, приведенной на рисунке 7.Для решения подобной задачи в FTStudio необходимо импортировать старый тест, задать для него имя и описать на языке STeeL функциональную схему.

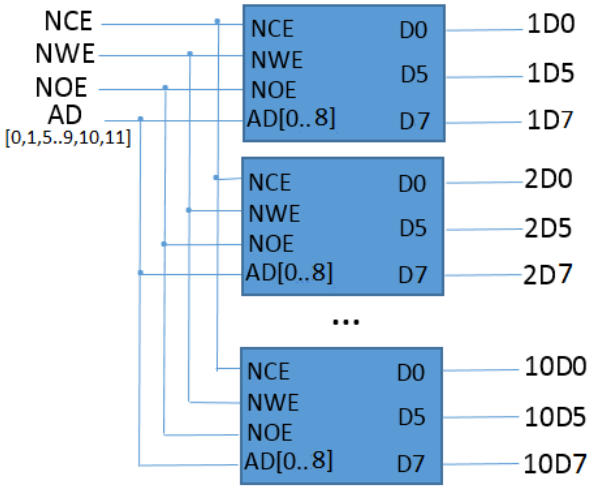


Рисунок 7. Функциональная схема теста для проведения электротермотренировки

Пример реализации этого теста приведен в следующем примере:

```
pin ["NCE", "NWE","NOE"], IN
  //Задаем шину адреса
  bus "AD","A",IN,9
  //импортируем ранее созданный тест памяти
  //(memory.fto) и указываем для него параметр
  // ADSIZE равным 9
  using alg::memory(ADSIZE=9)
```

```
  //соединяем сигналы исходного теста с вновь
  //создаваемым
  connect NCE, memory.NCE
  connect NWE, memory.NWE
  connect NOE, memory.NOE
  connect AD, memory.AD
#end
function Test()
  //формируем 10 двунаправленных групп D0,D5,D7
  for (var i=1; i<=10; i++){
     //номера подключаемых разрядов шины данных
     var digit ={0,5,7};
     for (var k=0;k< digit.Length;k++){
       var name = "_"+i+"D"+raz[k];
       #pin name, IO
       #connect name, memory["D"+ digit [k]]
     }
  }
  //копируем все циклы исходного теста
  #testcycle(memory.CyclesCount)
    next memory
  #end
TestEnd
```

6. Разработка исследовательских тестов

В виду того, что у СБИС СОЗУ 16Мбит необходимо контролировать 32 сигнала, а оборудование содержит всего 16 цифровых канала, то исследовательский функциональный тест предлагается разделить на два. В первом тесте будем контролировать младшие разряды шины данных, во втором – старшие (рисунок. 8).

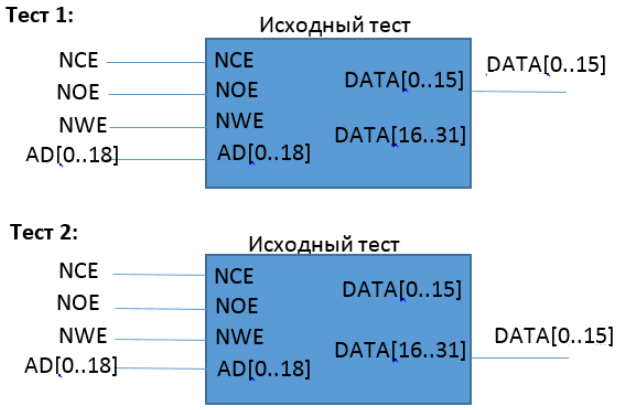


Рисунок 8. Функциональная схема исследовательского теста

```
#steel
  //Задаем управляющие сигналы
  pin ["NCE", "NWE","NOE"], IN
 //Задаем шины адреса и данных
  bus "AD","A",IN,18
  bus "DATA","D",IO, 16
  //импортируем ранее созданный тест памяти
  // (memory.fto)
  using "memory"
  //соединяем сигналы исходного теста с вновь
  //создаваемым
  connect NCE, memory.NCE
  connect NWE, memory.NWE
  connect NOE, memory.NOE
  connect AD, memory.AD
#end
function Test()
  //выполняем подключение младших или старших
 //битов шины данных
  for (var i=0; i<15; i++){
       #var delta = 0
       #ifdef test2
          delta =16;
       #endif
       #connect DATA[i], memory.DATA[i+delta]
  }
  //копируем все циклы исходного теста
  #testcycle(memory.CyclesCount)
    next memory
  #end
TestEnd
```

7. Сборка функциональных тестов

Для разработки кроссплатформенных тестов с изменяющимися параметрами в среду FTStudio встроен язык макросов, позволяющий описать алгоритм компиляции набора функциональных тестов. Следующий пример демонстрирует создание необходимого и достаточного набора тестов памяти для рассмотренных трех СБИС СОЗУ, отличающихся размером шин данных:

```
variable("ADSIZE", 19);
var datasize={8,16,32};
for(var i=0; i< datasize.Length;i++){
   variable("DATASIZE", datasize[i]);
   var Name="SOZU"+ datasize[i]+"T";
   compile ("memory.ftp", Name,
              format=V93000);
   compile ("ett.ftp", Name +"_ett",format=MAX3);
   compile ("memory.ftp", Name+"_bzr",
              format="BZR_SOZU4M");
    compile ("memory.ftp", Name+"_icc",
              format="BZR_SOZU4M",
              define="ICCMEAS");
}
//компиляция исследовательских тестов
compile ("isledovat.ftp", "SOZU16_isl1",
        format="NEW_HARWARE");
compile ("isledovat.ftp", "SOZU16_isl2",
        format="NEW_HARWARE" , define="test2");
```

В новой версии среды разработки функциональных тестов FTStudio планируется включить поддержку среды Lorenz, предназначенной для проведения анализа результатов функционального контроля, основанной на языке Python [2]. Данная поддержка позволит непосредственно из среды Lorenz запускать разработанные в FTStudio функциональные тесты с указанными параметрами и выполнять сложный функциональный анализ результатов.

Выводы

Разработанная система автоматического проектирования функциональных тестов позволяет существенно сократить время при подготовке и отладке кроссплатформенных тестов.

Использование встроенного языка описания алгоритмов тестовых воздействий автоматизирует процесс разработки функциональных тестов, а концептуально заложенная в среду гибкость позволяет добавлять поддержку нового измерительного оборудования и форматов тестов функционального контроля.

.

Development of Cross-platform functional test for VLSI

K. K. Smirnov

Abstract. Issues regarding the process of development of cross-platform functional tests in FTStudio (Functional Test Studio) are discussed in this paper. The application of the proposed approach allows significantly reduce labor costs for the development of set of functional tests targeting different measurement equipment.

Keywords: Wednesday FTStudio, automation test, equipment

Литература

1. К.К. Смирнов, М.Д. Бубнова. Среда для подготовки программ функционального контроля // Труды НИИСИ РАН, 2014, том 4, №1, стр. 32-39.

2. К.К. Смирнов, Е.Н. Ефимов. Организация прослеживаемости предметов труда при проведении функционального контроля СБИС // Труды НИИСИ РАН, 2014, том 4, №1, стр. 40-44.

3. Н.К. Жердев, Б.П. Креденцев, Р.Н. Белоконь. Контроль устройств на интегральных схемах // Техника 1986.

4. С.Г. Бобков. Методика тестирования микросхем для компьютеров серии «багет», Программные продукты и системы, 2007, № 3, стр. 2-5.

5. Wu Cheng-wen, Wen Xiaoqing, Wang Laung-terng. VLSI Test Principles and Architectures: Design for Testability. Morgan Kaufmann Publishers, 2006.

Процедурный сшитый код для обеспечения нисходящей разработки структурированных программ в ДССП для троичной машины

А.А. Бурцев

кандидат физико-математических наук

Аннотация: В НИЛ троичной информатики ВМК МГУ созданы троичная виртуальная машина ТВМ и кросс-система ДССП-ТВМ разработки программ для неё на языке ДССП-Т – троичном варианте языка ДССП, а также диалоговый интерпретатор ДССП/ТВМ, функционирующий на ТВМ как резидентное ПО. В статье объясняется, как решена проблема обеспечения нисходящей разработки структурированных программ при реализации кросс-компилятора и диалогового интерпретатора ДССП для ТВМ.

Ключевые слова: структурированное программирование, сшитый код, нисходящая разработка, троичный компьютер, ДССП.

Введение

В НИЛ троичной информатики ВМК создан программный комплекс ТВМ (Троичная Виртуальная Машина) [1], имитирующий функционирование современного варианта троичного процессора двухстековой архитектуры с поддержкой структурированного программирования на уровне машинных команд, аналогичной той, что была обеспечена в троичной ЦМ “Сетунь-70”.

Для программирования ТВМ сначала была построена кросс-система ДССП-ТВМ [2], позволяющая создавать программы для ТВМ на языке ДССП-Т с помощью кросс-компилятора, а затем был создан диалоговый интерпретатор ДССП/ТВМ [3], способный функционировать на троичной машине (ТВМ) в качестве её резидентной системы программирования.

Чтобы обеспечить возможности для структурированного программирования в полном объёме, при создании этих систем потребовалось решать проблему обеспечения нисходящей разработки программ. Результаты решения этой проблемы применительно к ДССП для ТВМ и являются предметом данной статьи.

1. Структурированное программирование как метод разработки программ и характеризующие его принципы

Вводя в употребление термин «структурированное программирование» (“structured programming”) Эдсгер Дейкстра предлагал [4] называть им такой метод разработки программы, в результате которого создаваемая программа приобретает свойственную ей (как правило, древовидную) структуру, отражающую динамическую сущность заложенного в неё алгоритма. И ведь действительно, если такая структура сама характеризует динамику поведения программы, т.е. порядок следования в ней ветвлений, циклов, вызовов подпрограмм, а также их иерархию, и явно отражена в её исходном тексте, то такую структурированную программу становится намного легче понять, модифицировать и сопровождать. В результате это и позволяет, в конечном счёте, радикально решить проблему снижения трудоёмкости программирования, т.е. существенно снизить затраты труда и облегчить работу не только одного программиста, но и всего программистского коллектива в целом

Согласно замыслу автора этого термина Дейкстры структурировать программу означает:

1) разрабатывать её путём пошагового уточнения (step wise refinement) и нисходящего анализа согласно так называемому принципу «сверху-вниз» (top-down);

2) выделять обособленные смысловые её части в отдельные программные компоненты – модули;

3) использовать для управления ходом исполнения программы только известные хорошо понимаемые управляющие конструкции (структуры).

Применительно к языкам программирования того времени для оформления отдельных модулей предлагалось использовать подпрограммы (процедуры и функции), а для последовательного сочленения и организации ветвлений и циклов было предложено использовать только такие управляющие конструкции, которые соответствуют правилу «один вход - один выход»: **begin…end, if…then…else, while…do…, repeat…until.**

Существенно, что из набора средств, рекомендованных к употреблению в программе для управления ходом её исполнения, Дейкстра предложил исключить оператор перехода (**goto**), объясняя это тем, что его применение может существенно повредить структуру программы и ухудшить тем самым её читабельность.

Излишне заострив на этом внимание (например, в своей статье “Goto considered harmful”), Дейкстра спровоцировал полемику среди ведущих программистов того времени по вопросу целесообразности отказа от оператора перехода.

Некоторые защитники оператора **goto** (например, Дональд Кнутт [5]) приводили примеры программ, структура которых, по их мнению, наоборот, ухудшалась, когда не разрешалось использовать этот оператор. Как правило, в приводимых программах требова-

лось обеспечить обработку чрезвычайных ситуаций с прекращением исполнения всей программы или какой-то её обособленной части в случае возникновения подобной ситуации. А в рекомендованном Дейкстрой наборе подходящей управляющей конструкции (на тот момент) не было, так что для обеспечения экстренного выхода (из программы или отдельной её компоненты) действительно приходилось употреблять оператор перехода или его разновидности (**exit**, **return**), либо усложнять программу, предусматривая излишние дополнительные проверки по окончании отдельных её участков, ветвлений и циклов, при исполнении которых подобная чрезвычайная ситуация могла возникнуть.

Но, к счастью, такая полемика оказалась даже полезной. В результате она привела к появлению в языках программирования специальных структурированных управляющих конструкций для обработки так называемых исключительных ситуаций, например, в форме защищённого оператора: **begin-except-end**.

К сожалению, излишний акцент на запрете оператора перехода допустил возможность упрощенно трактовать структурированное программмирование всего лишь как «программирование без goto». А такая упрощённая трактовка выхолащивает весь глубокий смысл, заложенный в этот термин его основоположником. Появились даже публикации, в которых нисходящая разработка (top-down) программы и структурированное программирование рассматривались как два разных отдельных метода [6, п 2.14–2.16].

Ещё большее сожаление вызывает недопонимание истинного смысла термина “structured programming”, когда встречаешь в русско-язычных изданиях его не совсем верный перевод в виде фразы «структурное программирование», что, увы, конечно же, не отражает полностью его истинный смысл. Ведь английское слово “structured” означает не прилагательное от слова «структура» (“structure”), а причастие, образованное от глагола “to structure”, что в переводе означает «структурировать», т.е. придавать структуру подлежащему воздействию объекту, а в нашем конкретном случае, программе.

Поэтому ещё раз подчеркнём, что структурированное программирование как метод разработки компьютерных программ, предложенный Дейкстрой, подразумевает совокупное соблюдение трёх принципов:

1) пошаговая нисходящая разработка программы;

2) дробление программы на обособленные компоненты - модули;

3) употребление только структурированных управляющих конструкций при составлении программы.

2. Языки для поддержки структурированного программирования

Успешность применения структурированного программирования как метода разработки программ во многом зависит того, какую поддержку применению этого метода могут оказать используемые для разработки языки и системы программирования.

Среди широко распространённых к тому времени (на момент публикации известных заметок Дейкстры о структурированном программировании) языков, пожалуй, лишь Алгол и ПЛ/1 хоть в какой-то мере могли быть признаны пригодными для составления структурированных программ. Эти языки предоставляли достаточно полный ассортимент управляющих конструкций, рекомендованных Дейкстрой. Но при этом они и никак не запрещали программировать «неструктурированно», ибо допускали возможность употребления оператора перехода **goto** без каких-либо ограничений.

А вот применять подпрограммы в этих языках оказывалось не совсем выгодным. Во-первых, по соображениям эффективности: вызов подпрограммы (особенно в ПЛ/1) сопровождался значительными накладными расходами, что приводило к существенному замедлению программы. А во-вторых, из-за досадных неудобств, связанных с оформлением исходного текста программы: объявление подпрограммы требовалось поместить в текст программы заранее, по крайней мере, до её первого вызова.

Всё это в совокупности приводило к тому, что программисты, как правило, стали оформлять в виде подпрограмм лишь те фрагменты алгоритма, которые приходилось бы записывать повторно, т.е. оформлять подпрограммы (процедуры и функции) считалось целесообразным ради сокращения исходного текста всей программы.

А чтобы отразить задуманную структуру алгоритма в исходном тексте программы, старались использовать так называемую «лесенку»: одинаковыми отступами от начала строки отмечали начало и конец соответствующих управляющих конструкций, вложенных друг в друга при составлении текста программы. Более того, именно наличие такой «лесенки» стало впоследствии считаться чуть ли не главным признаком хорошо структурированной программы.

Употребление «лесенки», конечно же, способствует пониманию структуры программы, но только в том случае, когда эта «лесенка» видна вся сразу целиком. В том же случае, когда текст программы не удаётся разместить на одной странице листа бумаги (при печати) или целиком в окне на одном экране (при просмотре программы на мониторе), «лесенка» уже не помогает увидеть созданную структуру всей программы. В таких случаях разбиение всей программы на отдельные модули (процедуры и функции) становится просто необходимым. При этом (в идеале) хорошо бы потребовать, чтобы размер отдельного модуля (подпрограммы) был таким, чтобы текст каждого модуля умещался целиком в просматриваемом окне экрана или странице печати.

И хотя впоследствии было создано много языков с учётом потребностей структурированного программирования (Паскаль, Алгол-68, Си, Forth, Модула, Модула-2, Ада и др.), следует признать, что лишь немногие из них поощряют выработку строгого стиля структуризации программ и позволяют отразить в исходном тексте создаваемой программы применение нисходящего (top-down) метода её разработки.

Почти во всех широко распространённых языках (Паскаль, Си, Forth и выросших из них многочисленных разновидностях) требуется сначала объявить подпрограмму (всю целиком или хотя бы характеризующий её заголовок), а уж потом разрешается употреблять её вызов. Это досадное ограничение затрудняет

сочинение исходного текста программы в полном соответствии с планом построения программы, выработанным путём нисходящего анализа (по принципу «сверху-вниз»).

В самом деле, чтобы лучше быть понятной человеку, всю программу надо изображать по принципу «сверху-вниз», т.е. сначала представить её головную часть, а уж затем, спускаясь вниз, охарактеризовать детали реализации отдельных её частей. Поскольку программа представляет собой текст, предназначенный для чтения слева направо и сверху вниз, значит, сначала в этом тексте следует разместить тело основной (головной) процедуры программы, а затем уже тела вызываемых в нём подпрограмм. И далее так же следует поступить с каждой подпрограммой: сначала разместить в тексте её тело, а следом за ним уже помещать объявления (и тела) тех процедур, которые использует в своём определении эта подпрограмма.

Но составленный таким способом исходный текст не способен принять транслятор (компилятор или интерпретатор) используемого языка. Вот поэтому и приходится для имеющегося транслятора представлять программу в несколько изменённом, перевёрнутом виде, т.е. образно говоря ставить программу «с ног на голову». При этом приходится сначала разместить в исходном тексте объявления всех используемых подпрограмм, а потом уже помещать в текст тела процедур, их вызывающих. И конечно же, при таком вынужденном способе записи программы основное тело программы окажется, увы, не в начале, как хотелось бы, а только в самом конце исходного текста программы.

Возникает очевидный вопрос: почему трансляторы столь несовершенны, почему не могут «прочитать» и проанализировать программу, записанную в том же естественном порядке («сверху-вниз»), как она создавалась методом пошагового нисходящего уточнения?

Для ответа на такой вопрос необходимо глубже изучить проблемы построения трансляторов. Оказывается, чтобы «научить» транслятор воспринимать программу, сочинённую и записанную методом «сверху-вниз», необходимо решить (при его создании) так называемую «проблему неопределённых ссылок вперёд».

Эта проблема заключается в следующем. Допустим, в программе определяется тело процедуры P, содержащее вызовы ещё неопределённых процедур P1, P2 … Pn. Чтобы сформировать код тела процедуры P, необходимо знать адреса размещения всех вызываемых процедур Pi (i=1…n), но эти адреса станут известны позднее, после того, как транслятор, проанализировав их определения, сформирует для них код и разместит их тела (в памяти или в выходном файле).

Проще всего такая проблема решается, если транслятору разрешается повторно прочитать исходный текст анализируемой программы, т.е. в итоге прочитать его дважды. Такой двухпроходной транслятор на первом проходе фиксирует для каждой процедуры, где она будет размещена, и запоминает её адрес, а на втором проходе уже формирует тела всех процедур, расставляя в них команды вызова подпрограмм с адресами, намеченными на первом проходе.

Однако, необходимость повторного прохода существенно ограничивает возможности применения транслятора. Это не только замедляет его работу, но и в ряде случаев делает её практически невозможной (например, если большой файл читается с устройства последовательного доступа). Поэтому в целях эффективности их применения трансляторы стараются создавать однопроходными.

В однопроходном трансляторе для разрешения «проблемы ссылок вперёд» требуется для каждой процедуры, которая определяется позже, чем встречается её вызов, сначала формировать список тех позиций кода программы, где требуется поместить ссылку на неё (в команде вызова такой процедуры), а затем после того, как будет обработано объявление этой процедуры и сформировано и размещено её тело, потребуется проставить ссылки на неё во всех тех позициях кода, которые были запомнены в сформированном для неё списке. Создание таких списков, конечно же, существенно усложняет однопроходной транслятор и требует дополнительных расходов памяти.

Проблема ещё более усложняется, если такая позже определяемая процедура наделяется параметрами. Чтобы знать заранее, какие у неё параметры, трансляторы как раз и «требуют» предварительно задать хотя бы её заголовок, с тем, чтобы знать, какие команды для подготовки нужных параметров поместить в формируемый код программы непосредственно перед самой командой вызова этой процедуры.

Видимо, все эти возникающие сложности, связанные с разрешением «проблемы ссылок вперёд», и служат причиной, почему не всякий транслятор наделяют способностью «понимать» программу, составленную методом «сверху-вниз» и потому имеющую «ссылки вперёд» на процедуры, определяемые позже их вызова.

3. ДССП и сшитый код

Диалоговая система структурированного программирования (ДССП) была создана [7] в начале 80-х годов XX века в проблемной лаборатории ЭВМ на факультете ВМК МГУ под руководством Брусенцова Н.П. ДССП была призвана существенно облегчить разработку ПО для широкого класса малых цифровых машин – мини- и микрокомпьютеров.

ДССП создавалась в качестве программного эмулятора новой двоичной цифровой минимашины, в которой предполагалось воплотить ценные идеи двухстековой архитектуры и структурированных команд управления, унаследованные от модернизированной троичной ЦМ «Сетунь-70».

ДССП относится к классу языков, построенных на основе интерпретации так называемого сшитого (threaded) кода. К такому же классу относится и язык Форт, известный своей словарной организацией и интерактивным режимом работы.

ДССП создавалась для поддержки структурированной разработки программ и потому способствует соблюдению всех трёх принципов такой разработки. Этим она выгодно отличается от других языков того же класса, в том числе и от Форта.

ДССП не только воспринимает написанную «сверху-вниз» структурированную программу, но ещё и стимулирует дробление такой программы на более мелкие процедуры. Для организации ветвлений и цик-

лов в ДССП предлагается использовать команды условного вызова процедуры и многократно повторяющегося вызова процедуры, принуждая тем самым оформлять тела ветвлений и циклов в виде отдельных процедур.

В ДССП каждая из команд ветвлений вместо перехода по телу сшитого кода (как это обычно делается в системе Форт) по сути осуществляет вызов процедуры слова, выбранного по условию, а каждая команда цикла организует многократный вызов процедуры слова, пока такой вызов допускается условием цикла. Таким образом, в ДССП поддержка структурированного программирования осуществляется непосредственно на уровне сшитого кода аналогично тому, как такая поддержка была обеспечена в ЦМ «Сетунь-70» на уровне машинных команд.

Определение нового слова в ДССП и в языке Форт синтаксически выглядят одинаково:

: P P1 P2 ... Pn ; [определение слова с именем P]

Однако Форт допускает подобное определение, только если все слова P1 P2 ... Pn уже были определены и стали известны системе. А ДССП допускает, что в этой последовательности слов P1 P2 ... Pn могут встречаться в том числе и такие слова, которые еще не были определены и предполагается, что будут определены впоследствии. Именно благодаря такой особенности ДССП и позволяет разрабатывать программу методом нисходящего анализа: сначала определить главное слово, затем слова следующего уровня, из которых оно определяется, и так постепенно спуститься до детального определения самых простых слов.

При создании ДССП в качестве основной требовалось, прежде всего, решить «проблему неопределённых ссылок вперёд». Далее рассмотрим, как решалась эта проблема в версиях ДССП для двоичных машин.

Но прежде вспомним простой приём создания списка позиций употребления неопределённых ссылок, который уже не раз применялся для решения подобной проблемы. Для сохранения списка адресов таких позиций все эти позиции связывались в цепной список так, что в каждой из них помещалась ссылка на предыдущую позицию, где встречался вызов той же не определённой пока процедуры. А начальная ссылка такого списка, указывающая на позицию последнего вызова этой процедуры, помещалась в специальном заголовке таблицы неопределённых процедур. Как только встречалось определение этой процедуры и становился известен адрес её размещения, требовалось пройти по сформированному для неё списку позиций и проставить во всех этих позициях только что назначенный для неё адрес.

Такой приём, в частности, применялся в однопроходном ассемблере ЦМ «Сетунь-70». Почему же этим известным приёмом нельзя было воспользоваться при создании ДССП?

Дело в том, что ДССП предлагает составлять и отлаживать программу в режиме диалога и допускает, что создаваемая программа может быть запущена на пробное исполнение, будучи ещё не до конца реализованной. В ней могут встречаться вызовы слов–процедур, которые ещё не были определены (предполагается, что они будут определены поздне).

Поэтому необходимо позаботиться о том, чтобы при попытке вызова такой не определённой пока процедуры диалоговая система смогла бы продолжить своё нормальное функционирование, сообщив лишь о встреченном ею вызове неопределённой процедуры. А это значит, в тех позициях, где остаются неразрешённые ссылки, требуется на некоторое время поместить особые команды для вызова пустой процедуры–заглушки, а ещё лучше — встроенного отладчика (если он имеется), который бы позволил в режиме диалога исполнить вместо вызванной процедуры разумную последовательность заменяющих её действий.

Вот почему при создании интерпретатора ДССП для двоичных машин не представлялось возможным применить описанный выше прием, применённый в ассемблере для ЦМ «Сетунь-70»: ведь чтобы построить такой цепной список, потребовалось бы разместить на таких позициях ссылки для его продолжения, а они могли бы помешать нормальному функционированию интерпретатора.

Для решения «проблемы неопределённых ссылок вперёд» в первой версии ДССП (ДССП-НЦ [8]) был специально предложен новый вид сшитого кода, получивший название «сшитый код двойной косвенности». Чтобы объяснить его суть, сначала познакомимся с разновидностями [9] сшитого кода, которые до этого применялись при построении Форт-подобных систем.

1. *Процедурный сшитый код* (*Subroutine Threaded Code* — STC). Так согласно принятой классификации (см. [9]) называют код, состоящий из машинных команд вызова подпрограмм (процедур):

call adrP1; call adrP2; … call adrPn;

который непосредственно может исполняться базовой машиной, т.е. той, на которой функционирует интерпретатор Форт-подобной системы.

2. *Прямой сшитый код* (*Direct Threaded Code* — DTC). Такой код, в основном, состоит только из адресов вызываемых подпрограмм:

IntProlog; adrP1; adrP2; … adrPn; adrRet;

Но впереди у него помещается код-пролог, который как раз и организует (запускает) необходимую последовательность вызовов процедур, задаваемыми этими адресами. В конце списка адресов помещается адрес особой процедуры, которая завершает последовательность таких вызовов.

Команда вызова подпрограммы, как правило, складывается из двух частей: кода команды и адреса подпрограммы. Поэтому хранить программу в DTC-коде оказывается выгодным, так как для её размещения потребуется значительно меньший объём памяти.

3. *Косвенный сшитый код* (*Indirect Threaded Code* — ITC). В таком коде размещаются ссылки на тела, в первой позиции которых задаётся либо адрес процедуры, реализующей слово-примитив, либо адрес процедуры, которая запускает интерпретацию нового уровня вызовов процедур из стоящего следом списка ссылок:

ptrInt; ptrP1; ptrP2; … ptrPn; ptrRet

В отличие от DTC-кода, который начинает исполняться как машинный код, для ITC-кода предполагается, что он начинает исполняться интерпретатором.

4. *Косвенный знаковый сшитый код* (*Indirect Token Threaded Code* — ITTC). В отличие от ITC-кода в этом виде кода вместо обычных ссылок применяются ин-

дексы, по которым из специально заготовленной таблицы выбираются адреса подпрограмм, подлежащих вызову:

indInt; indP1; indP2; … indPn; indRet

Индексы – это числа небольшого диапазона, поэтому список из них занимает гораздо меньше места в памяти, чем список ссылок, состоящий из обычных машинных адресов.

Сшитый код двойной косвенности строится аналогично ITTC-коду, только для ссылки на вызываемую процедуру вместо индексов используются обычные указатели на помещённые в словарь специальные заголовки слов, в первой позиции которых задаётся ссылка на тело реализующей это слово процедуры, либо ссылка на процедуру вызова встроенного отладчика в случае, когда определение для такого слова ещё не встретилось. При последующем определении такого слова, потребуется исправить лишь ссылку на тело этого слова в его заголовке, и ничего не потребуется изменять в тех позициях сформированного кода ДССП-программы, где встречался вызов этого слова, поскольку в таких позициях всегда проставляется указатель на заголовок этого слова. А этот указатель получает значение только однажды и далее уже не меняется. Оно фиксируется в момент, когда в словарь добавляется заголовок этого слова, если оно опознано как новое, неизвестное ранее (т.е. не найденное в словаре). Такой момент возникает либо когда это слово определяется (т.е. задаётся его тело), либо когда оно впервые употребляется (вызывается) в теле другого слова.

Реализованный на микрокомпьютере «НЦ-03Д» интерпретатор ДССП-НЦ, чтобы вызвать исполнение очередного слова, на которое ссылался текущий указатель по телу сшитого кода двойной косвенности, вынужден был потратить 7 машинных команд вместо 5-ти, если бы он был реализован для косвенного сшитого кода (как для Форта). Таким образом, возможность программирования «сверху-вниз» достигалась в ДССП-НЦ ценой лишь незначительного снижения производительности интерпретатора.

Но в последующей версии ДССП-80 для микрокомпьютеров унифицированной архитектуры (PDP-11) снижение производительности обещало уже быть более весомым. Ведь основной цикл внутреннего интерпретатора удавалось реализовать всего одной командой базовой машины (JSR @R5+), если для внутреннего представления ДССП-программы использовать прямой сшитый код. Поэтому ради эффективности интерпретатора было предложено решить «проблему неопределённых ссылок вперёд» другим способом.

Подробно этот способ описан в [10]. Основная идея этого решения заключается в том, что в словаре вместе с заголовком неопределённого слова хранится в форме особой таблицы список адресов тех позиций тела формируемого кода ДССП-программы, где встречался вызов этого неопределённого слова. В самих же этих позициях временно помещается ссылка на встроенный отладчик. А когда встречается определение этого слова, то ссылка на тело этого слова проставляется во всех позициях, запомненных в такой таблице, а сама таблица расформировывается.

Конечно же, формирование и хранение такой таблице в заголовке каждого слова существенно усложняет структуру ДССП-словаря. Но с таким усложнением пришлось согласиться ради быстродействия ДССП-интерпретатора. В дальнейшем словарь такой сложной структуры использовался во всех последующих версиях ДССП, созданных для различных операционных сред и разнообразных архитектур двоичных машин.

4. Обеспечение нисходящей разработки в ДССП для ТВМ

При разработке ДССП-ТВМ предстояло решить проблему, какую же разновидность сшитого кода применять для внутреннего представления, чтобы ТВМ могла напрямую (т.е. без специального интерпретатора) вызывать процедуры по встречающимся в этом коде адресам. Для решения этой проблемы было предложено применить в ТВМ такую кодировку команд, чтобы значение адреса воспринималось ею и как команда вызова процедуры с этого адреса. В результате для внутреннего представления программ в ДССП-ТВМ стал использоваться так называемый процедурный сшитый код (STC согласно классификации, приведённой в [9]), который может непосредственно исполняться троичной машиной.

Кросс-компилятор формирует такой код на языке ассемблера ТВМ. В нём для каждого слова-примитива ДССП-ядра размещается одна машинная команда, которая реализует действие этого примитива. Для тех примитивов, действие которых невозможно исполнить одной машинной командой, заготовлены ассемблерные процедуры, их реализующие, а в сшитый код вставляется адрес (или команда вызова) соответствующей ей процедуры. Совокупность таких ассемблерных процедур образует ассемблерное ядро, которое добавляется к каждой откомпилированной ДССП-программе.

Поскольку кросс-компилятор ДССП-ТВМ формирует код ДССП-программы на ассемблере ТВМ, то он тем самым просто перекладывает решение проблемы «неопределённых ссылок вперёд» на транслятор языка ассемблера, который разрешает неопределённые ссылки путём осуществления двойного прохода текстового файла, подаваемого ему на обработку. Интерпретатор же ДССП/ТВМ должен решать такую проблему самостоятельно.

Ранее были рассмотрены возможные варианты решения данной проблемы в версиях ДССП, реализованных для двоичных машин: либо путём применения для внутреннего представления ДССП-программы сшитого кода двойной косвенности (в ДССП-НЦ]), либо за счёт усложнения словаря путём хранения в нём для каждого встреченного (но не определённого ещё) слова специальных таблиц, содержащих адреса позиций тел, в которых было использовано это слово.

Но такие варианты решения этой проблемы при создании интерпретатора ДССП/ТВМ потребовали бы серьёзной переделки и самого ДССП-компилятора, и построенного для него ассемблерного ядра. Ведь интерпретатор должен уметь работать с тем же строением словаря и с тем внутренним представлением ДССП-программ, которые поддерживаются кросс-

компилятором и ассемблерным ядром системы разработки ДССП-ТВМ.

В ходе разработки интерпретатора ДССП/ТВМ удалось найти иное решение данной проблемы, не потребовавшее каких-либо модификаций ни словаря, ни внутреннего представления ДССП-программы. Рассмотрим предлагаемый способ разрешения неопределённых ссылок и объясним, почему он сделан именно таким.

Для каждого встреченного неопределённого слова ДССП-интерпретатор должен уметь, по крайней мере, запоминать адреса всех позиций в телах, где такое слово было вызвано. А встретив определение этого слова, проставить во всех запомненных позициях ссылку на сформированное тело этого слова.

Для сохранения адресов таких позиций можно связать все эти позиции в цепной список, записав в каждой из них ссылку на предыдущую позицию, где встречался вызов этого неопределённого слова. А начальную ссылку такого списка, указывающую на позицию последнего вызова этого слова, можно разместить в заголовке неопределённого слова (в поле команды).

Такой известный приём построения списка позиций употребления неопределённых слов нами ранее уже рассматривался. Существенный недостаток такого приёма сохранения списка позиций проявляется при попытке запускать в среде интерпретатора ДССП-программу, содержащую вызовы неопределённых слов. Такие вызовы в прежних реализациях ДССП-интерпретаторов могли бы вызывать непредсказуемые последствия, поэтому разработчики ДССП вынуждены были отказываться от такого привлекательного приёма сохранения списка позиций.

Но в создаваемом интерпретаторе ДССП/ТВМ описанный приём сохранения списка позиций можно выгодно использовать, если поставить в конце созданного списка (т.е. в той позиции тела, где неопределённое слово встретилось первый раз) адрес специальной процедуры, назначенной для общей реакции на вызов неопределённого слова (см. рис. 1, вариант А).

Воспользуемся тем, что ТВМ воспринимает 26-тритный адрес как команду вызова процедуры с этого адреса. Значит, при вызове такой команды с одной из позиций, используемой в построенном цепном списке, произойдет череда вызовов процедур по адресам из этого списка и, в конечном итоге, будет вызвана назначенная спецпроцедура реакции.

Предполагается, что в такой процедуре реакции интерпретатор может предложить разработчику ДССП-программы выполнить действия, имитирующие эффект исполнения не определённой пока процедуры, и затем вернуться к продолжению прерванной программы. (В дальнейшем такая спецпроцедура реакции, возможно, станет компонентой специализированного отладчика в составе интерпретатора).

Поэтому спецпроцедура реакции должна, во-первых, уметь распознать, какое неопределённое слово было вызвано, т.е. получить адрес его словарного заголовка, и во-вторых, обеспечить продолжение прерванной программы с той точки, где было вызвано неопределённое слово.

По адресу возврата из спецпроцедуры реакции можно узнать адрес позиции, откуда она была вызвана. В этой позиции должен завершаться цепной список, построенный для вызванного неопределённого слова. Далее потребуется осуществить обход словаря и для каждого неопределённого слова (помеченного флагом U) проверить, завершается ли его цепной список именно в этой позиции. Так можно узнать, какое неопределённое слово было вызвано.

Но к сожалению, в ДССП не во всех случаях по адресу возврата можно правильно определить позицию, откуда была вызвана процедура (у нас это спецпроцедура реакции). Например, её нельзя однозначно определить, если процедура вызывалась управляющими командами: BR+ BRS BR DW EXEC .

Чтобы обеспечить возможность правильного определения позиции конца цепного списка по адресу возврата из спецпроцедуры, потребуем, чтобы он заканчивался на позиции, где поставлена команда явного вызова спецпроцедуры реакции на неопределённое слово. Такая дополнительная позиция требуется для каждого неопределённого слова на время работы с его цепным списком. Для таких позиций будем выделять память динамически, используя для этой цели свободные (нулевые) ячейки специально организованной таблицы, размещённой в памяти так, чтобы не мешать формированию тела и словаря ДССП-программы (см. рис. 1 вариант Б).

При определении тела неопределённого слова, когда во все позиции его цепного списка заносится адрес формируемого тела, сам цепной список расформировывается, а отведённая ему ячейка спецтаблицы освобождается (она помечается нулевым значением) и может быть распределена снова.

Рис. 1. Схемы вариантов решения проблемы неопределённых слов.

Предложенный вариант (Б) позволяет не только однозначно распознать, какое неопределённое слово было вызвано, но и вернуться из спецпроцедуры реакции к продолжению прерванной программы. Заметим, что для обеспечения правильного возврата (именно в ту точку, где было вызвано неопределённое слово), процедура реакции должна будет очистить стек возвратов от “лишних” адресов, накопившихся в нём в результате промежуточных вызовов процедур по адресам из цепного списка.

В данном варианте для распознавания неопределённого слова требуется осуществлять обход словаря и

цепных списков. Можно значительно ускорить процесс поиска неопределённого слова, если в спецтаблице рядом с позицией конца цепного списка сохранять ссылку на заголовок неопределённого слова, для которого этот список построен (см. Рис. 1 с вариантом В). Но тогда потребуется усложнить процедуру передвижения словаря, т.к. ей придётся ещё и корректировать такие ссылки, сохраняемые в спецтаблице.

Если даже в качестве таких ссылок на узлы словаря использовать не абсолютные, а относительные адреса, то их всё равно пришлось бы корректировать: если не при передвижении словаря, так при его очистке, выполняемой операцией CLEAR .

Усложнение операций передвижения и очистки словаря может замедлять регулярную работу интерпретатора по построению ДССП-программы. А длительный поиск неопределённого слова может проявиться лишь в виде паузы при реакции интерпретатора на вызов неопределённого слова. Поскольку такая реакция, как правило, сама предусматривает приостановку исполнения ДССП-программы с выходом в режим диалога, то данная пауза вряд ли будет заметна. Поэтому в настоящее время для решения проблемы неопределённых слов в интерпретаторе ДССП/ТВМ используется вариант Б, изображённый на рисунке 1 как основной.

7. Заключение

Для полноценной поддержки структурированного программирования необходимо не только предусмотреть в языке программирования управляющие конструкции для структурированного кодирования создаваемой программы, но и обеспечить возможность её пошаговой нисходящей разработки. ДССП является одной из немногих оригинальных систем, которая обеспечивает такую поддержку.

При реализациях ДССП для двоичных машин проблема обеспечения нисходящей разработки требовала либо усложнения строения сшитого кода, либо усложнения структуры словаря. Но при реализации ДССП для троичной машины (ТВМ) удалось найти такое решение проблемы, которое позволило и упростить ДССП-словарь, и применить для внутреннего представления ДССП-программ такой вариант процедурного сшитого кода, который может напрямую непосредственно исполняться созданной виртуальной троичной машиной.

Subroutine threaded code to support top-down development of structured programs in DSSP for ternary computer

A.A. Burtsev

Abstract: In research laboratory of ternary informatics leaded by Brusentsov N. P. at the Computer Science (CS) department of the Moscow State University (MSU) the ternary virtual machine (TVM) and cross-system (DSSP-TVM) for development of programs for it using the DSSP-T language are created. Ternary virtual machine (TVM) is a simulator of ternary computer, which architecture has two stacks (data stack and control stack) and machine commands for structured programming like «Setun-70». Two variants of Dialogue System of Structured Programming (DSSP) for TVM have been constructed. DSSP-TVM lets it possible to create DSSP-program for TVM by means of cross-compiler. DSSP/TVM is a dialogue interpreter, which can run on TVM as its resident soft-ware. Programming language DSSP-T used in both systems is a ternary version of language of the DSSP.

The main problem during realization of the DSSP-TVM cross-compiler and the DSSP/TVM dialogue interpreter is how to support top-down development of structured program. Variants of solving this problem in DSSP for TVM are explained.

Keywords: structured programming, threaded code, top-down development, ternary computer, DSSP.

Литература

1. С.А.Сидоров, Ю.С.Владимирова. Троичная виртуальная машина // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ,2011. С.46-55.

2. А.А.Бурцев, Х.Рамиль Альварес. Кросс-система разработки программ на языке ДССП для троичной виртуальной машины // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ,2011. С.183-193.

3. А.А.Бурцев, М.А.Бурцев. ДССП для троичной виртуальной машины // Труды НИИСИ РАН, т. 2, № 1.— М.: Изд-во НИИСИ РАН, 2012.— С. 73–82.

4. У.Дал, Э.Дейкстра, К.Хоор. Структур(ирован)ное программирование.— М.: Изд-во Мир, 1975.— 247 с.

5. D.Knuth. Structured programming with GOTO statements // ACM Computer Survey, 1974, v.6, p.261-301.

6. Д.Ван Тассел. Стиль, разработка, эффективность, отладка и испытание программ.— М.: Изд-во Мир, 1981.

7. А.А.Бурцев, С.А.Сидоров. История создания и развития ДССП: от "Сетуни-70" до троичной виртуальной машины. // Вторая Международная конференция "Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР" SORUCOM-2011 (12-16 сентября 2011 г., г. Великий Новгород, Россия): Труды конференции. В.Новгород: Изд-во НовГУ, 2011. с. 83-88.

8. В.Б.Захаров, Г.В.Златкус, И.А.Руднев, С.А.Сидоров. Реализация диалоговой системы структурированного программирования на микрокомпьютере «Электроника НЦ-03Д» // Архитектура и программное оснащение цифровых систем.— М.: Изд-во МГУ, 1984, с. 10–17.

9. T.Ritter, G.Walker. Varieties of threaded code for language implementation // BYTE, 1980, v. 5, No. 9, p.206.

10. С.А.Сидоров. Программирование сверху вниз и организация словаря ДССП // Вопросы кибернетики. Сборник статей / ред. В. Б. Бетелин. М.: Изд-во НИИСИ, 1999.— С. 32–44.

Ограничения на параметры относительного движения для основных видов шарниров

М.В. Михайлюк[1], Е.В. Страшнов

1 – доктор физико-математических наук

Аннотация: В статье на примерах основных видов шарниров описана процедура вывода ограничений на параметры относительного движения. Эти ограничения используются в методе последовательных импульсов для моделирования динамики системы многих тел.

Ключевые слова: система шарнирно связанных тел, параметры относительного движения, неудерживающие связи, якобианы ограничений.

Введение

Для моделирования динамики шарнирно связанных тел требуется описать функциональные соотношения, которые определяют поведение разных типов шарниров. Этими соотношениями являются связи, которые ограничивают относительное движение в системе многих тел. Каждый шарнир ограничивает определённое число степеней свободы системы (например, сферический шарнир ограничивает три степени свободы). Оставшиеся (неограниченные шарнирами) степени свободы системы описываются с помощью наборов параметров, которые определяют движение всей системы. В данной статье рассматриваются такие шарниры, на параметры которых накладываются дополнительные ограничения (например, в осевом шарнире заданы пределы изменения угла от -45° до 45°). Эти ограничения в дальнейшем используются в методе последовательных импульсов, описанного в статье [1], для моделирования динамики системы многих тел.

1. Ограничения шарниров системы многих тел

Рассматривается система из N тел, связанных с помощью M шарниров (рис. 1). Каждое тело определяется с помощью координат $\bar{X}_i$, $i=1,...,N$. Данный набор координат является избыточным, поэтому дополнительно координаты тел должны удовлетворять набору связей. Для k-го шарнира, соединяющего i-ое и j-ое тело должны быть выполнены голономные удерживающие связи следующего вида

$$\bar{G}_k(\bar{X}_i,\bar{X}_j)=\bar{0},\quad k=1,...,M,$$

где $\dim(\bar{G}_k)=m_k$ - число степеней свободы, которые ограничивает k-ый шарнир.

На оставшиеся $l_k=6-m_k$ степеней свободы дополнительно могут быть наложены неудерживающие связи следующего вида

$$\bar{\chi}_k(\bar{X}_i,\bar{X}_j)\ge\bar{0},\quad k=1,...,M. \qquad (1)$$

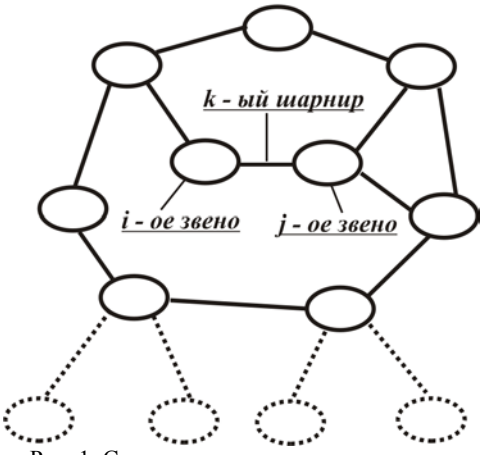


Рис. 1. Система шарнирно-связанных тел

Для описания движения i-го твёрдого тела используется линейная скорость $\bar{v}_i$ и угловая скорость $\bar{\omega}_i$, которые объединим в общий вектор скоростей $\bar{V}_i=\begin{pmatrix}\bar{v}_i\\ \bar{\omega}_i\end{pmatrix}$. Кинематика твёрдого тела описывается с помощью соотношений (см. [1])

$$\dot{\bar{X}}_i=H_i(\bar{X}_i)\bar{V}_i.$$

Дифференцируя функцию $\bar{\chi}_k$ из (1) по времени, получим

$$\dot{\bar{\chi}}_k=\frac{\partial\bar{\chi}_k}{\partial\bar{X}_i}\dot{\bar{X}}_i+\frac{\partial\bar{\chi}_k}{\partial\bar{X}_j}\dot{\bar{X}}_j=$$
$$\left(\frac{\partial\bar{\chi}_k}{\partial\bar{X}_i}H_i(\bar{X}_i)\right)\bar{V}_i+\left(\frac{\partial\bar{\chi}_k}{\partial\bar{X}_j}H_j(\bar{X}_j)\right)\bar{V}_j= \qquad (2)$$
$$J_{ki}\bar{V}_i+J_{kj}\bar{V}_j,$$

где $J_{ki}=\frac{\partial\bar{\chi}_k}{\partial\bar{X}_i}H_i(\bar{X}_i)$ и $J_{kj}=\frac{\partial\bar{\chi}_k}{\partial\bar{X}_j}H_j(\bar{X}_j)$ - якобианы ограничений.

С помощью соотношений (2) выписываются ограничения, которые используются в методе последовательных импульсов [1]. Выпишем ограничения (1) и

соотношения (2) для сферического, осевого и призматического шарнира.

2. Описание ограничений на параметры в сферическом шарнире

Рассмотрим сферический шарнир с ограничениями на углы поворота звеньев и выпишем для него ограничения (1) и соотношения (2).

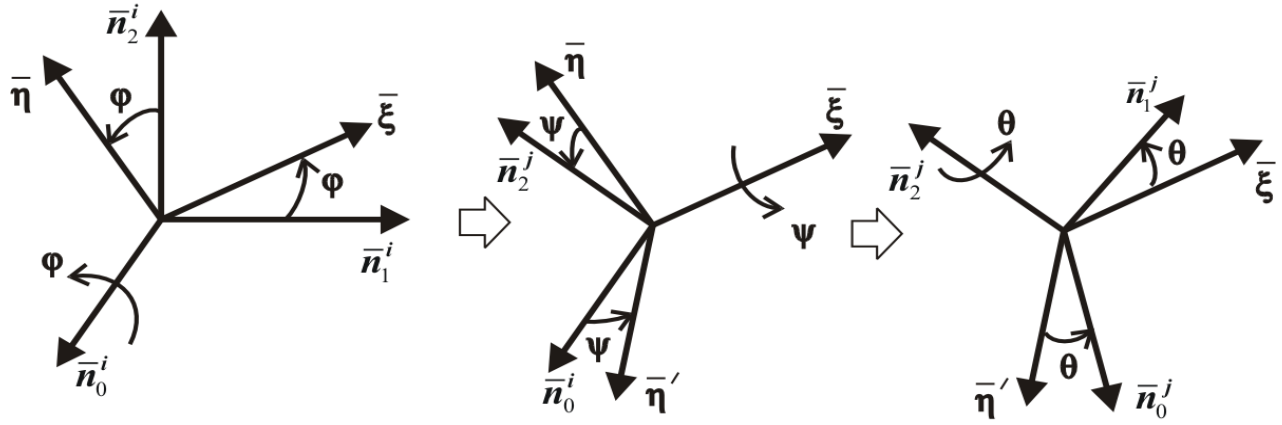


Рис. 2. Преобразование систем координат с помощью углов Эйлера

Введем две системы координат шарнира, которые жёстко связаны с соединяемыми телами и имеют общее начало в точке шарнира. Обозначим орты первой системы координат через $\left\{\bar{n}_0^i, \bar{n}_1^i, \bar{n}_2^i\right\}$, а второй – через $\left\{\bar{n}_0^j, \bar{n}_1^j, \bar{n}_2^j\right\}$. Предполагая, что системы координат шарнира в начальный момент времени совпадают, движение в сферическом шарнире можно описать с помощью трех углов Эйлера, осуществляя последовательные повороты j-го тела (относительно i-го) сначала на угол φ вокруг оси $\bar{n}_0^i$ (изначального положения оси $\bar{n}_0^j$), затем на угол ψ вокруг оси $\bar{\xi}$ для совмещения $\bar{\eta}$ с $\bar{n}_2^j$ и на угол θ вокруг оси $\bar{n}_2^j$ для совмещения всех осей координат. Здесь через $\bar{\xi}$ и $\bar{\eta}$ обозначены новые положения осей $\bar{n}_1^j$ и $\bar{n}_2^j$ после поворота на угол φ. Каждый поворот осуществляется против часовой стрелки, если смотреть с конца вектора, вокруг которого производится вращение (см. рис. 2).

Мы рассматриваем сферический шарнир с ограничениями следующего вида:

$$\begin{aligned} &-\pi < \varphi_{\min} \le \varphi \le \varphi_{\max} < \pi; \\ &-\pi/2 < \psi_{\min} \le \psi \le \psi_{\max} < \pi/2; \\ &-\pi < \theta_{\min} \le \theta \le \theta_{\max} < \pi. \end{aligned} \qquad (3)$$

Повороты вокруг осей $\bar{n}_0^i$, $\bar{\xi}$ и $\bar{n}_2^j$ задаются следующими соотношениями:

$$\begin{aligned} &\bar{n}_0^i = \bar{n}_0^i; && \bar{\eta}' = -\bar{\eta}\sin\psi + \bar{n}_0^i\cos\psi; \\ &\bar{\xi} = \bar{n}_1^i\cos\varphi + \bar{n}_2^i\sin\varphi; && \bar{\xi} = \bar{\xi}; \\ &\bar{\eta} = -\bar{n}_1^i\sin\varphi + \bar{n}_2^i\cos\varphi; && \bar{n}_2^j = \bar{\eta}\cos\psi + \bar{n}_0^i\sin\psi; \end{aligned}$$

$$\begin{aligned} &\bar{n}_0^j = \bar{\eta}'\cos\theta + \bar{\xi}\sin\theta; \\ &\bar{n}_1^j = -\bar{\eta}'\sin\theta + \bar{\xi}\cos\theta; \\ &\bar{n}_2^j = \bar{n}_2^j. \end{aligned} \qquad (4)$$

Матрица R перехода из шарнирной системы координат j - го тела в шарнирную систему координат i - го тела равна транспонированному произведению матриц вышеописанных поворотов вокруг осей координат. Используя сокращённые обозначения $c_* = \cos(*)$ и $s_* = \sin(*)$, получим:

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\varphi & -s_\varphi \\ 0 & s_\varphi & c_\varphi \end{pmatrix} \begin{pmatrix} c_\psi & 0 & s_\psi \\ 0 & 1 & 0 \\ -s_\psi & 0 & c_\psi \end{pmatrix} \begin{pmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} c_\psi c_\theta & -c_\psi s_\theta & s_\psi \\ s_\varphi s_\psi c_\theta + c_\varphi s_\theta & -s_\varphi s_\psi s_\theta + c_\varphi c_\theta & -s_\varphi c_\psi \\ -c_\varphi s_\psi c_\theta + s_\psi s_\theta & c_\varphi s_\psi s_\theta + s_\varphi c_\theta & c_\varphi c_\psi \end{pmatrix}.$$

Углы Эйлера выражаются через компоненты полученной матрицы следующим образом

$$\begin{aligned} &\varphi = \operatorname{atan2}\left(-r_{23}, r_{33}\right); \\ &\psi = \operatorname{atan2}\left(r_{13}, \sqrt{r_{11}^2 + r_{12}^2}\right); \\ &\theta = \operatorname{atan2}\left(-r_{12}, r_{11}\right). \end{aligned}$$

Функция $\operatorname{atan2}\left(y, x\right)$ вычисляет арктангенс от y/x (в радианах) в диапазоне от $-\pi$ до π. Если $x = 0$, то $\operatorname{atan2}\left(y, x\right) = \pm\pi/2$ в зависимости от знака y. Если оба аргумента равны 0, то значение функции также равно 0 (в нашем случае эта ситуация не возникнет в силу (3)).

Из (4) следует, что $\bar{n}_0^i \cdot \bar{n}_2^j = \sin\psi$ и, в силу монотонного роста синуса на интервале от $-\pi/2$ до $\pi/2$, ограничения на угол ψ можно записать в виде

$$\sin\psi_{\min} \le \bar{n}_0^i \cdot \bar{n}_2^j \le \sin\psi_{\max},$$

или в форме ограничения вида (1)

$$\chi_{k1} = \operatorname{sgn}(\psi)(\sin\psi^* - \bar{n}_0^i \cdot \bar{n}_2^j) \ge 0,$$

где $\psi^* = \psi_{\min}$ для ограничения снизу (т.е. $\psi > \psi_{\min}$) и $\psi^* = \psi_{\max}$ для ограничения сверху (т.е. $\psi < \psi_{\max}$).

Аналогично, для угла φ на основе соотношения $\bar{n}_2^i \cdot \bar{n}_2^j = \cos\varphi\cos\psi$ формируется ограничение вида (1) следующим образом:

$$\chi_{k2} = \bar{n}_2^i \cdot \bar{n}_2^j - \cos\varphi^* \sqrt{1-(\bar{n}_0^i \cdot \bar{n}_2^j)^2} \geq 0,$$

где $\varphi^* = \varphi_{\min}$ для ограничения снизу (т.е. $\varphi > \varphi_{\min}$) и $\varphi^* = \varphi_{\max}$ для ограничения сверху (т.е. $\varphi < \varphi_{\max}$).

Используя соотношение $\bar{n}_0^i \cdot \bar{n}_0^j = \cos\theta\cos\psi$, для угла θ формируется ограничение вида (1) следующим образом:

$$\chi_{k3} = \bar{n}_0^i \cdot \bar{n}_0^j - \cos\theta^* \sqrt{1-(\bar{n}_0^i \cdot \bar{n}_2^j)^2} \geq 0,$$

где $\theta^* = \theta_{\min}$ для ограничения снизу (т.е. $\theta > \theta_{\min}$) и $\theta^* = \theta_{\max}$ для ограничения сверху (т.е. $\theta < \theta_{\max}$).

Дифференцируя χ_{kr}, $r = 1,2,3$, получим соотношения (2) в виде

$$\dot{\chi}_{kr} = \left[\bar{0}^{\mathrm{T}} \quad -\bar{w}_r^{\mathrm{T}}\right]\bar{V}_i + \left[\bar{0}^{\mathrm{T}} \quad \bar{w}_r^{\mathrm{T}}\right]\bar{V}_j,$$

где $r = 1,2,3$; $J_{ki,r} = \left[\bar{0}^{\mathrm{T}} \quad -\bar{w}_r^{\mathrm{T}}\right]$, $J_{kj,r} = \left[\bar{0}^{\mathrm{T}} \quad \bar{w}_r^{\mathrm{T}}\right]$,

$$\bar{w}_1 = -\mathrm{sgn}(\psi)(\bar{n}_2^j \times \bar{n}_0^i),$$

$$\bar{w}_2 = (\bar{n}_2^j \times \bar{n}_2^i) + \cos\varphi^* \tan\psi(\bar{n}_2^j \times \bar{n}_0^i),$$

$$\bar{w}_3 = (\bar{n}_0^j \times \bar{n}_0^i) + \cos\theta^* \tan\psi(\bar{n}_2^j \times \bar{n}_0^i).$$

При дифференцировании были использованы формула Бура [2] $\dot{\bar{a}} = \bar{\omega} \times \bar{a}$ (для дифференцирования в мировой системе координат), свойство смешанного произведения $\bar{a}(\bar{b} \times \bar{c}) = \bar{b}(\bar{c} \times \bar{a}) = \bar{c}(\bar{a} \times \bar{b})$ и следующее соотношение

$$\frac{d}{dt}\sqrt{1-(\bar{n}_0^i \cdot \bar{n}_2^j)^2} =$$

$$\frac{-(\bar{n}_0^i \cdot \bar{n}_2^j)}{\sqrt{1-(\bar{n}_0^i \cdot \bar{n}_2^j)^2}} \frac{d}{dt}(\bar{n}_0^i \cdot \bar{n}_2^j) = -\tan\psi \frac{d}{dt}(\bar{n}_0^i \cdot \bar{n}_2^j)$$

3. Описание ограничений на параметры в осевом шарнире

Параметром относительного движения в осевом шарнире является угол поворота φ вокруг оси осевого шарнира $\bar{n}_0^i$. Для получения угла поворота рассмотрим относительный единичный кватернион $\bar{q} = \{\bar{q}_v, q_w\}$, который определяет ориентацию одного соединяемого тела относительно другого. С помощью этого кватерниона из соотношений $\bar{q}_v = \sin\frac{\alpha}{2}\bar{u}$ и $q_w = \cos\frac{\alpha}{2}$ можно определить единичную ось мгновенного вращения $\bar{u}$ и угол поворота α вокруг этой оси. Для угла α справедливы следующие неравенства

$$0 \leq \alpha = 2\,\mathrm{atan}\,2\left(\|\bar{q}_v\|, q_w\right) =$$

$$2\,\mathrm{atan}\,2\left(\left|\sin\frac{\alpha}{2}\right|, \cos\frac{\alpha}{2}\right) < \pi.$$

Угол $\alpha = 2\,\mathrm{atan}\,2\left(\|\bar{q}_v\|, q_w\right)$ является модулем угла поворота φ. Если угол $\varphi < 0$, то ось мгновенного вращения противоположна оси осевого шарнира $\bar{n}_0^i$ (то есть $\bar{u} = -\bar{n}_0^i$). Таким образом, угол φ можно вычислить на интервале от $-\pi$ до π следующим образом

$$\varphi = \begin{cases} 2\,\mathrm{atan}\,2\left(\|\bar{q}_v\|, q_w\right), & \bar{q}_v \cdot \bar{n}_0^i \geq 0; \\ -2\,\mathrm{atan}\,2\left(\|\bar{q}_v\|, q_w\right), & \bar{q}_v \cdot \bar{n}_0^i < 0. \end{cases}$$

Мы рассматриваем осевой шарнир с ограничениями следующего вида

$$\varphi_{\min} \leq \varphi \leq \varphi_{\max}.$$

Обозначая орты координат шарнира для i-го тела через $\left\{\bar{n}_0^i, \bar{n}_1^i, \bar{n}_2^i\right\}$, а для j-го тела через $\left\{\bar{n}_0^j, \bar{n}_1^j, \bar{n}_2^j\right\}$, движение j-го относительно i-го тела можно описать поворотом j-го тела на угол φ вокруг оси $\bar{n}_0^i$ (см. рис. 3).

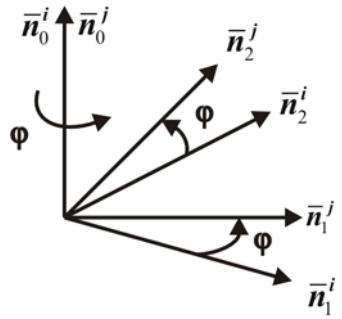


Рис. 3. Системы координат осевого шарнира

Поворот вокруг оси $\bar{n}_0^i$ (против часовой стрелки, если смотреть с конца вектора) задаётся следующими соотношениями

$$\bar{n}_1^j = \bar{n}_1^i \cos\varphi + \bar{n}_2^i \sin\varphi;$$

$$\bar{n}_2^j = -\bar{n}_1^i \sin\varphi + \bar{n}_2^i \cos\varphi.$$

Отсюда следует, что $\bar{n}_1^i \cdot \bar{n}_1^j = \cos(\varphi)$. Тогда ограничения (1) можно сформировать в следующем виде

$$\chi_k = \bar{n}_1^i \cdot \bar{n}_1^j - \cos(\varphi^*) \geq 0,$$

где $\varphi^* = \varphi_{\min}$ для ограничения снизу (т.е. $\varphi > \varphi_{\min}$) и $\varphi^* = \varphi_{\max}$ для ограничения сверху (т.е. $\varphi < \varphi_{\max}$).

Дифференцируя χ_k по времени, получим соотношение (2) в виде

$$\dot{\chi}_k = \left[\bar{0}^{\mathrm{T}} \quad -\bar{w}^{\mathrm{T}}\right]\bar{V}_i + \left[\bar{0}^{\mathrm{T}} \quad \bar{w}^{\mathrm{T}}\right]\bar{V}_j,$$

где $J_{ki} = \left[\bar{0}^{\mathrm{T}} \quad -\bar{w}^{\mathrm{T}}\right]$, $J_{kj} = \left[\bar{0}^{\mathrm{T}} \quad \bar{w}^{\mathrm{T}}\right]$, $\bar{w} = (\bar{n}_1^j \times \bar{n}_1^i)$.

4. Описание ограничений на параметры в призматическом шарнире

Пусть векторы $\bar{r}_P^i$ и $\bar{r}_P^j$ задают в мировой системе координат положения двух систем координат шарнира, которые жёстко связаны с i-м и j-м телом. Параметром, характеризующим относительное движение в призматическом шарнире, является величина перемещения $d = (\bar{r}_P^j - \bar{r}_P^i)\bar{n}_0^i$, где $\bar{n}_0^i$ - ось призматического

шарнира. Мы рассматриваем призматический шарнир с ограничениями следующего вида

$$d_{\min} \le d \le d_{\max},$$

где для простоты будем рассматривать такие ограничения на параметры, что $d_{\min} < 0$ и $d_{\max} > 0$.

Ограничением (1) на параметр d будет соотношение следующего вида

$$\chi_k = \mathrm{sgn}(d)(d^* - d) = \mathrm{sgn}(d)(d^* - (\bar{r}_P^{\,j} - \bar{r}_P^{\,i})\bar{n}_0^i) \ge 0,$$

где $d^* = d_{\min}$ для ограничения снизу (т.е. $d > d_{\min}$) и $d^* = d_{\max}$ для ограничения сверху (т.е. $d < d_{\max}$).

Дифференцируя χ_k, получим соотношение (2) в виде

$$\dot{\chi}_k = \begin{bmatrix} \mathrm{sgn}(d)\bar{n}_0^{i\mathrm{T}} & \bar{\xi}_i^{\mathrm{T}} \end{bmatrix} \bar{V}_i + \begin{bmatrix} -\mathrm{sgn}(d)\bar{n}_0^{i\mathrm{T}} & -\bar{\xi}_j^{\mathrm{T}} \end{bmatrix} \bar{V}_j,$$

где $J_{ki} = \begin{bmatrix} \mathrm{sgn}(d)\bar{n}_0^{i\mathrm{T}} & \bar{\xi}_i^{\mathrm{T}} \end{bmatrix}$, $J_{kj} = \begin{bmatrix} -\mathrm{sgn}(d)\bar{n}_0^{i\mathrm{T}} & -\bar{\xi}_j^{\mathrm{T}} \end{bmatrix}$, $\bar{\xi}_i = \mathrm{sgn}(d)(\bar{r}_P^{\,j} - \bar{r}_C^{\,i}) \times \bar{n}_0^i$, $\bar{\xi}_j = \mathrm{sgn}(d)(\bar{r}_P^{\,j} - \bar{r}_C^{\,j}) \times \bar{n}_0^i$, $\bar{r}_C^{\,i}$ и $\bar{r}_C^{\,j}$ - радиус-векторы центров масс двух тел.

Заключение

В данной статье на примерах сферического, осевого и призматического шарнира была рассмотрена процедура вывода ограничений на параметры относительного движения в системе многих тел. Ограничения (1) и соотношения (2) используются для моделирования динамики шарнирно связанных тел методом последовательных импульсов.

Использование углов Эйлера в качестве параметров сферического шарнира не является единственным [3, 4], и в связи с тем, что не любые ограничения могут быть наложены на углы Эйлера, в качестве дальнейшего исследования данной проблематики может быть выбрана формулировка ограничений (1) и соотношений (2) для других параметров, описывающих сферический шарнир.

Restrictions on the parameters of relative motion for main types of joints

M.V.Mikhaylyuk, E.V.Strashnov

Abstract: The paper describes the procedure of receiving the restrictions on relative motion parameters for main types of joints. These constraints are used in the method of sequential impulses for simulation of dynamics of multibody system.

Keywords: articulated rigid body system, relative motion parameters, unilateral constraints, Jacobians of restrictions.

Литература

1. Е.В. Страшнов, М.В. Михайлюк. Моделирование ограничений на относительное движение шарнирно связанных тел. Мехатроника, автоматизация, управление 2015 (в печати).

2. Курс теоретической механики: Учебник для вузов/ В.И. Дронг, В. В. Дубинин, М.М. Ильин и др.; Под общ. ред. К. С. Колесникова, 3-е изд., стереотип. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005. – 736 с.

3. F.S. Grassia. Practical Parameterization of Rotations using Exponential Map, Journal of Graphics Tools, 3(3), pp. 29-48, 1998.

4. P. Baerlocher, R. Boulic. Parameterization and range of motion of the ball-and-socket joint. Proceedings of AVATARS’2000 Conference, Lausanne, pp. 180-190, 2000.

Знакомим дошкольников и младших школьников с азами алгоритмики с помощью систем ПиктоМир и КуМир

А.Г.Кушниренко[1], А.Г.Леонов[1], М.А.Ройтберг[2]

1 – кандидат физико-математических наук, 2 – доктор физико-математических наук .

Аннотация. Современные информационные технологии становятся рутинным элементом общей культуры школьников и дошкольников. В статье предлагается методика организации занятий по овладению начальной алгоритмической грамотности в младших классах и дошкольных учреждениях. В качестве педагогических программных средств используются системы ПиктоМир и КуМир, позволяющие начать осваивать азы алгоритмики в игровой форме.

Ключевые слова: школьная информатика, педагогическое программное средство, программирование, ПиктоМир, КуМир, раннее обучение информатике.

Сегодня, в первой четверти 21 века, на каждого жителя Земли приходится несколько микропроцессоров, число переключательных элементов в каждом из которых приближается к числу нейронов в человеческом мозге. Построенные на базе таких микропроцессоров компьютеры начинают превосходить человека в тех областях, которые ранее считались не поддающимися автоматизации: игра в шахматы, узнавание человека по фотографии, вождение автомобиля, игра «Что-где-когда?», финансовые спекуляции.

Экономисты и социологи говорят, что человечество переходит от индустриального уклада к информационному. Бытовые устройства

становятся «умными» и интегрируются в Интернет. Дети, еще не умея читать и писать, успешно осваивают «умные» телевизоры, родительские смартфоны и планшеты, получают в подарок радиоуправляемые игрушки-роботы, играют в компьютерные игры на родительских или собственных планшетах.

Неудивительно, что под воздействием этих радикальных изменений в образе жизни человечества меняется и традиционное понятие грамотности. Заголовок одной из статей в газете Нью-Йорк Таймс – «Чтение, письмо, счет, а теперь и программирование» - метко описывает суть происходящих изменений [1].

По поводу традиционных компонент грамотности – чтения, письма и счета – и, в профессиональном образовательном сообществе, и в обществе в целом имеется консенсус. Ожидания по поводу уровня освоения навыков чтения, письма и счета, которые, по мнению родителей, их детям должна обеспечить начальная школа, совпадают с тем, что школа в состоянии обеспечить, и, как правило, обеспечивает.

По поводу новой грамотности – алгоритмической – сегодня даже намека на такой консенсус нет ни в профессиональном педагогическом сообществе, ни в родительских кругах, ни среди работодателей в сфере информационных технологий. Авторы настоящей статьи – принадлежа одновременно к трем

перечисленным сообществам, ответственным за выработку такого консенсуса – сделали в настоящей статье попытку описать тот уровень всеобщей алгоритмической грамотности, достижение которого к концу первого-второго года школьного обучения реально было бы обеспечить и, по мнению авторов, стоило бы обеспечить в России.

Мы предлагаем организовать постижение начальной алгоритмической грамотности в три этапа-уровня. Уровни 1 и 2 могут быть освоены в подготовительной группе детского сада, уровень 3 – в первом классе начальной школы. В зависимости от обстоятельств можно ограничиться только освоением уровня 1 или уровней 1 и 2. На освоение дошкольных уровней 1 и 2 мы отводим суммарно 24 занятия по 30 – 40 минут каждое и на освоение школьного уровня 3 – еще 12 занятий по 45 минут. Первая половина каждого занятия – посвящена коллективным бескомпьютерным играм. Вторая половина каждого занятия посвящается индивидуальным или кооперативным компьютерным играм. Прохождение уровней каждой такой игры требует от ребенка составления все более интересных программ по управлению виртуальными и реальными роботами. Хотя без реальных роботов на уровнях 1 и 2 можно и обойтись, их использование радикальным образом улучшает мотивацию и глубину освоения материала.

На уровнях 1 и 2 программы составляются на планшетах на бестекстовом (пиктограммном) языке программирования, доступном дошкольникам-шестилеткам.

На уровне 3 используется текстовый язык с национальной лексикой, требуемое подмножество которого доступно первоклассникам-семилеткам. В это подмножество входят понятия *программа, подпрограмма, циклические и условные управляющие конструкции*, но не входят понятия *переменная, выражение, присваивание*, а числовые значения-константы используются только в качестве повторителя и как аргументы команд виртуального или реального робота. Для решения ряда задач управления роботами в качестве суррогата целочисленной переменной во внешнюю обстановку вводится исполнитель Счетчик.

В опробованных с участием авторов методиках используются языки программирования разработанных в НИИСИ РАН учебных программных систем ПиктоМир и КуМир [2–6]. На школьном уровне Пиктомир и Кумир методически представляют собой единую систему: можно начать составлять пикто-программу в ПиктоМире, затем автоматически перевести ее в текстовую форму и продолжить работу в КуМире.

По мнению авторов, начальный курс алгоритмической грамоты естественно разбивается на следующие три уровня.

Уровень 1 (12 занятий).

1.1. Парадигма программного управления исполнителями. Понятия:

- *робот – исполнитель команд*; система команд исполнителя; обстановка, в которой «работает» исполнитель; возможность аварии при исполнении данной команды в данной обстановке;
- *алгоритм* – пошаговый план будущих действий по управлению исполнителем с целью достижения определенного результата;
- *исполнение алгоритма* – процесс последовательной выдачи команд исполнителю в соответствии с заранее выработанным планом;
- *программа* – алгоритм, представленный в такой форме, которая позволяет поручить исполнение алгоритма компьютеру или другому автоматическому устройству;
- разделение обязанностей: *робот* – исполнитель команд, *компьютер* – исполнитель программ; *программист* – составитель программ;
- *язык программирования* – конкретный набор правил составления программ для исполнения компьютерами определенного типа.

1.2. Правила составления программ (без обратной связи) в пиктограммном языке программирования: *повторители и подпрограммы (вспомогательные алгоритмы)*.

1.3. *Реальный* робот и его *виртуальный* партнер. Сборка (из готового робототехнического комплекта, например Lego EV3) реального исполнителя-робота без обратной связи. Расхождение результатов выполнения отдельных команд и последовательностей команд реальным и виртуальным роботами. Составление пиктопрограмм для управления *без обратной связи* реальным роботом и его виртуальным партнером (рекомендуется виртуальный робот из компьютерной игры Сокобан).

Уровень 2 (12 занятий)

2.1. Методика коллективного выполнения одной работы двумя (несколькими) программистами: этап деления общей работы на части, этап утверждения договоренности о разделе работы, этап составления и отладки программ

(компьютеры партнеров контролируют соблюдение утвержденных договоренностей).
2.2. *Команды-вопросы*. *Обратная связь* при управлении исполнителем.
Возможность придумывания одного алгоритма с обратной связью, позволяющего достигать аналогичных целей для серии аналогичных обстановок.
Правила составления программ (с обратной связью) в пиктограммном языке программирования: *цикл «пока»* и *ветвление*.
2.3. Придумывание алгоритмов и составление программ с обратной связью
2.4. Примеры алгоритмов управления роботом-исполнителем, требующие подсчета числа шагов. Исполнитель «счетчик». Алгоритмы и программы последовательного управления несколькими исполнителями с использованием обратной связи.
2.5. *Параллельное управление* несколькими однотипными исполнителями с помощью одной и той же программы без обратной связи.

Уровень 3 (12 занятий)
3.1. Пиктограммные (детские) и текстовые (взрослые) языки программирования. Демонстрация автоматического перевода программ управления виртуальным исполнителем с пиктограммного языка на текстовый.

3.2. Правила перевода программ из пиктограммного представления в текстовое:
- общие правила текстовой записи программ: запись команд управления исполнителями, освоенными на этапе 1; запись нескольких команд в одной строке;
- синтаксические ошибки в программах на текстовых языках, правила их обнаружения и исправления;
- правила записи цикла «пока» и ветвлений;
- «заклинания» для использования целочисленной переменной в качестве счетчика (например, в языке КуМир:цел а; а:=а+1; а:=а-1; а=0; а<>0.)
3.3. Сборка (из готового робототехнического комплекта, например, Lego EV3) реального исполнителя-робота с командами обратной связи и составление простейших программ управления реальным роботом с обратной связью.

Подробная методическая разработка для занятий уровня 1 размещена на сайте www.piktomir.ru [7]. Занятия по этой методичке и ее ранней версии успешно проводились в ДОУ Москвы и Сургута. Отдельные темы уровней 2 и 3 обкатывались авторами в кружковой работе.

Для освоения отдельных элементов перечисленных уровней может быть использован ряд зарубежных методик и программных систем: игры lightbot и robologic, миниатюрный робот Ozobot, робототехнический комплект Primo (робот Cubette), система программирования и методики с сайта code.org, система программирования и игры Google-проекта blockly и, наконец, широкораспространенные системы ЛогоМиры и Scratch.

В заключение перечислим основные, по мнению авторов, достоинства комплекта учебных систем ПиктоМир-КуМир и предлагаемой методики для воспитания алгоритмической грамотности дошкольников и первоклассников.

Достоинства Пиктомира. Невозможность допустить синтаксическую ошибку.
Организация выполняемых учениками заданий в виде уровней компьютерной игры. Возможность задания на каждом уровне жесткого шаблона программы, которую должен составить ученик. Возможность включения в игру уровней-подсказок, для выравнивания темпа освоения материала детьми с разным уровнем подготовки без выдачи дополнительных заданий.

Достоинства КуМира. Русская лексика, непрерывная индикация синтаксических ошибок. Возможность задания шаблона программы, защищенного от случайного искажения. Возможность организации автоматической проверки правильности выполнения заданий. Совместимость с ПиктоМиром. Возможность использования языка и системы в курсе информатики основной школы и при подготовке к ГИА.

Достоинства методики. Легкость освоения воспитателями ДОУ, не имеющими специальной подготовки. Большой объем (до 50%) некомпьютерных коллективных игр. Отсутствие необходимости хранить результаты работы учеников от занятия к занятию. Интеграция работы с виртуальными и реальными роботами. Компьютерная поддержка процедур кооперативной работы по составлению программ.

Teaching Programming Concepts to Preschoolers and Firstgraders with Programming Environments PiktoMir and KuMir

A.G. Kushnirenko, A.G. Leonov, M.A. Roytberg

Abstract. Modern information technologies have become an ordinary feature of the general culture of preschoolers and firstgraders. The paper proposes a method to foster algorithmic literacy with the pedagogical programming environments PiktoMir and Kumir.

Keywords: informatics, pedagogic programming environment, PiktoMir, KuMir, early studing of computer science.

Литература

1. http://www.nytimes.com/2014/05/11/us/reading-writing-arithmetic-and-lately-coding.html
2. I.B. Rogozhkina, A.G. Kushnirenko. PiktoMir: Teaching Programming Concepts to Preschoolers with a New Tutorial Environment. — // World Conference of Educational Technology and Researches, July, 2011
3. А.Г. Кушниренко, А.Г. Леонов. Программирование для дошкольников и младших школьников. — // Информатика. — М.: Первое сент., 2011, N15. — стр.20–23
4. А.Д. Кисловская, А.Г. Кушниренко. Методика обучения алгоритмической грамоте дошкольников и младших школьников — // Информационные технологии в обеспечении федеральных государственных образовательных стандартов: Материалы Международной научно-практической конференции. 16-17 июня 2014 года. — Елец: ЕГУ им. И. А. Бунина, 2014. — Т. 2. — стр. 3–7.
5. В.В. Яковлев. ПиктоМир: опыт использования и новые платформы. Презентация к выступлению на 6-ой конференции «Свободное программное обеспечение в высшей школе», январь 2011, Переславль-Залесский, http://www.gosbook.ru/node/32747
6. В.В. Яковлев. Кумир 2.0. Компилятор и среда выполнения (доклад на OSEDUCONF-2013), http://talks.rosalab.com/Kumir-20
7. А.Г. Кушниренко, М.В. Райко, И.Б. Рогожкина. Методические указания по проведению цикла занятий «Алгоритмика», http://www.piktomir.ru/m.pdf

ПиктоМир как кооперативная среда для обучения основам программирования дошкольников и младших школьников.

Н.О. Бесшапошников, А.Н. Дедков, Д.Б. Ерёмин, А.Г. Леонов[1]

1 – кандидат физико-математических наук.

Аннотация. Использование планшетных компьютеров и современные информационные технологий стали обычным элементом общей культуры дошкольников и школьников. Предлагается игровая кооперативная методика обучения основам программирования в младших классах и дошкольных учреждениях. В качестве педагогического программного средств для кооперативной работы предлагается система ПиктоМир. В статье описаны сетевые возможности системы ПиктоМир.

Ключевые слова: школьная информатика, педагогическое программное средство, программирование, ПиктоМир, раннее обучение информатике.

Вот уже 2 года существует в Нидерландах проект под названием «SteveJobsSchools» [1]. В пилотных школах королевства учителей «заменили» на планшеты iPad. Минусы этого спорного проекта общеизвестны, однако есть и позитивные стороны. Планшетные компьютеры для детей, обучающихся в младшей школе или в дошкольном образовательном учреждении, гораздо удобнее в использовании, чем лаптопы и настольные ЭВМ. Эргономические параметры и интерфейс планшетов комфортнее для ребенка.

Маленький ученик, скорее всего, уже хорошо знаком с таким типом компьютера, и он уже знает, что при помощи этого устройства можно не только смотреть любимые мультфильмы, но и играть.

Игра - это деятельность, в которой ребенок берет на себя «взрослые» роли и в игровых условиях воспроизводят деятельность взрослых и отношения между ними. Выбирая определенную роль, ребенок имеет и соответствующий этой роли образ – водителя, инженера и даже программиста. Из этого образа вытекают и игровые действия ребенка: игры детей, как правило, наполнены социальным содержанием и служат средством погружения в полноту человеческих отношений.

Один из видов игры этого дошкольного периода - образно-ролевая игра. В ней ребенок воображает себя кем-то и действует в соответствии с этим образом. Ребенка может привлечь бытовой предмет, явление природы, и он может стать ими на короткий промежуток времени. Обязательное условие для развертывания такой игры - яркое, запоминающееся впечатление, которое и вызвало у ребенка эмоциональный отклик.

В течении последних лет в детских садах и школах Москвы и России проходит эксперимент по обучению программированию детей 5-7 лет. Для этого педагоги используют новационный программный продукт – ПиктоМир [1]. Сейчас уже можно с уверенностью констатировать, что ПиктоМир прошел успешную апробацию и зарекомендовал себя как удобное средство для обучения детей азам последовательного программирования [2].

Первые шаги в программировании сложны для ребенка, ведь алгоритмический стиль мышления не впитывается с молоком матери. Однако, алгоритмический стиль мышления можно сформировать, это вполне решаемая задача даже в столь раннем возрасте.

В ПиктоМире учащийся видит знакомого по мультфильмам и кинофильмам персонажа – Робота. Робот для ребенка понятен и прост. Его даже нечего изучать. Ребенок привык управлять героями в компьютерных играх, поэтому, ассоциируя себя с Роботом, умеющим выполнять простейшие действия (шагать вперед, поворачиваться и «красить» - ремонтировать площадку, по сценарию игры) управлять им, используя кнопки на экране планшета (так называемое, ручное управление), для ученика привычно: нажимаешь на кнопки пульта, и Робот выполняет команды играющего. Да и задача, которую требуется решить, – тривиальна.

Осознав поставленную задачу, ученик методом проб и ошибок ведет Робота к финишу. Меняется задача, меняется обстановка для Робота, меняется и последовательность действий приводящая к успеху.

---

[1] ПиктоМир – это система бестекстового, пиктограммного программирования, которая позволяет ребенку «собрать» из пиктограмм на экране компьютера несложную программу, управляющую виртуальными исполнителями-Роботами. ПиктоМир, в первую очередь, ориентирован на дошкольников, еще не умеющих писать или на младшеклассников, не очень любящих писать [3,4].

В конечном итоге меняется и персонаж игры: сначала ребенок играет с одним роботом, потом с другим или с третьим. Однако все это еще далеко от программирования, пока это просто ручное, пультовое управление.

Как же перейти от ручного управления к составлению программ? Как объяснить ребенку необходимость замены непосредственного, кнопочного управления на составление последовательности действий Робота?

Можно предложить детям кооперативную бескомпьютерную игру, в которой, один ребенок, назовем его Командиром, дает команды другому ребенку-Роботу. Ученик-Робот умеет выполнять только элементарные действия, именно такие, которые выполнял Робот на экране.

Задача детям ставится та же самая: отремонтировать поле космодрома. Воображаемый космодром (лабиринт) можно легко изготовить при помощи бумаги и фломастеров.

Игра у детей начинается с договора, с установления правил. Дети договариваются о начале игровой деятельности, совместно составляют карту Космодрома, распределяют между собой роли и выстраивают свои действия и поведение в соответствии с выбранной ролью.

Взяв на себя роль, ребенок начинает принимать и понимать четкость ролевых прав и обязанностей. Так, например, Командир, который управляет Роботом, требует от игрока-Робота сделать шаг вперед, повернуться, отремонтировать клетку Космодрома, то есть потребовать, чтобы Робот четко выполнял его указания.

Действуя с предметами-заместителями (поле Космодрома, испорченные или отремонтированные клетки), ребенок начинает оперировать в мыслимом, условном пространстве. Предмет-заместитель становится опорой для мышления. Постепенно игровые действия сокращаются, и ребенок начинает действовать во внутреннем, умственном плане. Таким образом, игра способствует тому, что ребенок переходит к мышлению в образах и представлениях.

В продолжении игры, учитель немного меняет условие, рассказывая, что Робот находится на далекой-далекой планете, радио-сигнал до нее идет дни и месяцы, поэтому Командир не может командовать непосредственно. Это попросту невозможно. Командир пишет последовательность действий на листочке бумаги. А игрок-Робот, читая записи, последовательно выполнят их. И ребята вместе проверяют правильность составленной последовательности команд – программу. Так начинается программирование.

В игре, выполняя различные роли, ученик становится на разные точки зрения и начинает видеть предмет с разных сторон, что способствует развитию важнейшей, мыслительной способности человека, позволяющей представить другой взгляд и другую точку зрения.

Осознание правил (команд и поведения Робота) наиболее ярко проявлялось в замечаниях, которые дети начинали делать друг другу в случае неправильного исполнения. Ребенок охотно отмечает промахи других. Контроль за действиями других детей создает внутреннюю готовность к выполнению тех же действий.

Игру с правилами можно перенести уже на виртуального Робота. Например, по выбору педагога дети объединяются в группу. Каждый ребенок со своего планшета, самостоятельно, программирует своего Робота, для решения поставленной перед ним задачи. Учащийся может индивидуально выполнить свою программу, посмотреть на результат, отредактировать составленную программу. Когда, программа готова, ребенок отправляет ее своим партнерам по игре (запускает игру). Таким же образом он получает результаты программирования товарищей. При этом оценивается не только успешность решения поставленной задачи, но и, например, время, затраченное ребенком на решение задачи.

В этих состязательных играх всегда появляются лидеры, которые быстрее и легче выполняют поставленные задачи, осваивают новую обстановку, нового Робота. Стихийно появившиеся лидеры с удовольствием контролируют отстающих детей, помогая им с составлением программы, освоением нового Робота. Эти процессы происходят под наблюдением педагога, но без непосредственного участия последнего в процессе обучения. Эффективность кооперативного обучения дошкольников подчас сильно превышает эффективность индивидуальных занятий.

Кооперативные (как и состязательные игры) с одной стороны позволяют всем ученикам активно участвовать в изучении программирования, с другой стороны являются сильным стимулом к развитию алгоритмического мышления, так как дети стараются победить в игре и (или) совместными усилиями решить поставленную задачу.

Поскольку в игру включены несколько планшетных компьютеров, то для их функционирования необходима локальная сеть. Для

работы в сети используется специальная, сетевая часть системы ПиктоМир.

Задача сетевой части ПиктоМира обеспечить взаимодействие между учениками в классе и позволить преподавателю посредством его устройства следить за работой учеников, их прогрессом, давать задания и т.п.[5]

В основе сетевой системы лежит библиотека Enet (http://enet.bespin.org), которая реализует протокол RUDP (Reliable UDP – основанный на UDP, протокол для гарантированной передачи данных). Для задач ПиктоМира не является критичным и необходимым наличие TCP соединения. В системе ПиктоМир используются лимитированные объемы данных, однако при необходимо одновременно поддерживать большое количество "подключений". Использование UDP протокола с контролем доставки и очередности пакетов позволило значительно снизить нагрузку на сеть в процессе кооперативной или состязательной игр.

В предлагаемой нами сетевой реализации нет разделения на выделенный управляющий экземпляр ПиктоМира (учительского) и остальные игровые (ученические) ПиктоМиры. Обе роли исполняет одно и тоже приложение. С точки зрения ученика приложение ПиктоМир всегда находится в режиме «Ученик». Сетевое общение начинается с автономной работы «Учеников», как ведомых устройств. Эти приложения позволяют автономно составлять программы для Роботов, находясь одновременно, в состоянии ожидания появления «Учителя», распорядителя. «Учитель» – это режим приложения ПиктоМир, в который перевести свое устройство может только преподаватель. Для этого используется аппарат аутентификации.

Для упрощения администрирования принято, что в одной сети не может быть более одного учителя, в таком случае при активации «Учителя», ведомые устройства («Ученики») находят и подключаются к нему автоматически.

Для реализации этой функции каждое неподключенное ведомое устройство с заданным интервалом отправляет широковещательное сообщения поиска «Учителя». «Учитель» получив такое сообщение отправляет в ответ «Ученику» свои данные для подключения.

Даже в режиме индивидуальной (неавтономной) работы, у преподавателя есть возможность наблюдать со своего планшета (или компьютера) за прогрессом в решении задач учениками, в том числе, одновременно за всеми.

Учитель может ставить каждому отдельному ученику индивидуальное задание без непосредственного взаимодействия педагога с планшетом ребенка.

В режиме многопользовательских игр, чтобы снизить нагрузку на устройство «Учитель», для каждой группы учеников случайным образом выбирается мастер-устройство каждой конкретной игры, и ему отправляется информация для коллективной игры, подключения друг к другу, как бы создавая временную подсеть на игру. "Учитель" при этом поддерживает соединение и собирает статистику только с мастер-устройств таких подсетей. Кооперативные игры в ПиктоМире не только помогают учащимся эффективнее осваивать основы программирования, но и упрощают работу педагогу в классе, позволяя планировать и распределять задания и контролировать процесс обучения.

PiktoMir as the cooperative environment for studying the basic of programming for preschoolers and firstgraders

N.O. Beshaposhnikov, A.N. Dedkov, D.B. Eremin, A.G. Leonov

Abstract. Tablet computers and modern information technologies have become an ordinary feature of the general culture of preschoolers and firstgraders. Here is proposed a methods of teaching the basics of programming via cooperative computers games. As an educational software for cooperative work, a system PiktoMir is proposed. This article describes the networking capabilities of PiktoMir.

Keywords: informatics, computer game, pedagogic programming environments, programming, PiktoMir, early studying of computer science.

Литература

1. O4NT foundation (Education for a New Era). (2015, Aug.). http://www.educationforanewera.com. [Электронный ресурс], http://stevejobsschool.nl/founder-maurice-de-hond-and-his-daughter-daphne-speak-at-dclassconference-berlin/ (дата обращения: 01.08.2015).
2. А.Г.Кушниренко, А.Г.Леонов, К.А.Пронин, М.А.Ройтберг, В.В.Яковлев. Свободное программное обеспечение в высшей школе. - // ПиктоМир: опыт использования и новые платформы,. Переславль, 29-30 января 2011.
3. А.Г.Кушниренко, А.Г.Леонов. Программирование для дошкольников и младших школьников. — // Информатика. — М.: Первое сент., 2011, N15. — стр.20–23
4. А.Г.Кушниренко, И.Б.Рогожкина, А.Г.Леонов. Большой московский семинар по информатизации начального и дошкольного образования - // ПиктоМир: пропедевтика алгоритмического языка (опыт обучения программированию старших дошкольников), Москва, 2012.
5. А.Г.Леонов. Тенденции объектно-ориентированного программирования в разработке системы КуМир - // Программные продукты и системы, Тверь, 2012 - № 4, с. 245-249

Система программирования Кумир 2.х

А.Г.Кушниренко[1], М. А. Ройтберг[2], Д.В.Хачко, В. В. Яковлев[1]

1- кандидат физико-математических наук, 2 - доктор физико-математических наук.

Аннотация. Кумир 2.х – система программирования, предназначенная для поддержки курса программирования в начальной и средней школе – является развитием системы Кумир, созданной в России во второй половине 1980-х годов и использует школьный алгоритмический язык – алголоподобный язык с русской лексикой и встроенными командами управления программными исполнителями. Отличительной особенностями Кумир 2.х являются возможность использования практикумов и существенно возросшая по сравнению с предыдущими версиями скорость выполнения программ.

Ключевые слова: алгоритмический язык, система, программа.

1. Введение

В конце 1980-х годов были разработаны как система КУМИР, предназначенная для работы на всех доступных в то время архитектурах персональных и школьных ЭВМ, так и методические материалы, необходимые для поддержки курса школьной информатики. Ввиду быстрого развития компьютерной техники, к началу 2000-х готов встала задача создания новой версии системы Кумир, которая должна была работать под управлением операционных систем семейств Windows и Linux и при этом обладать всеми возможностями исходной системы Кумир. Эта задача была, в основном, решена к 2008 г., и на конференции «Свободное программное обеспечение в высшей школе» [1] была анонсирована система Кумир 1, предназначенная для работы в Windows и Linux.

В ходе разработки и последующего взаимодействия с педагогическим сообществом были выявлены направления дальнейшего совершенствования системы. Во-первых, стало ясно, что наибольшие нарекания пользователей вызывает низкая скорость выполнения КуМир-программ, обусловленная их интерпретацией во время выполнения. Во вторых, выяснилась необходимость расширения набора включенных в систему исполнителей. В третьих, оказалась успешной так называемая методика практикумов, позволяющая педагогу снабдить каждое задание на программирование средствами автоматизированной проверки правильности выполнения этого задания. С учетом накопленного опыта и возникновения новых потребностей, система Кумир была перепроектирована и переписана для решения следующих задач:

1. Возможность простого расширения функциональности системы путем реализации самодостаточных модулей.
2. Возможность создания на базе системы Кумир различных конфигураций, адаптированных под специфичные применения, например, для проведения олимпиад.
3. Радикальное увеличение скорости выполнения программ.
4. Поддержка системы практикумов с возможностью самопроверки (для учеников) и отслеживания хода работы учеников (для учителей).

Результатом работы является платформа Кумир 2.х [2], на базе которой создан комплект из компилятора-выполнителя (версия 2.0) и полнофункциональной версия обучающей системы Кумир (версия 2.1; в настоящее время комплект находится в стадии бета-тестирования).

2. Архитектура системы версий 2.х

Платформа Кумир 2.х представляет собой набор динамически загружаемых программных модулей, реализованных в виде DLL- или SO-библиотек, и небольшой программы для запуска.

Список используемых модулей и параметры их инициализации определяются конфигурацией запуска, которая может быть, как указана в явном виде, через аргументы командной строки, так и статически на этапе компиляции. На этапе компиляции системы из исходных текстов создается несколько типовых конфигураций запуска:

1. Система для поддержки учебника [3];
2. Система для поддержки учебников [4];
3. Система, предназначенная для подготовки заданий учителем;
4. Отдельные инструменты командной строки, предназначенные для работы в пакетном режиме.

Каждый модуль системы реализует один или несколько интерфейсов взаимодействия, а зависимости между модулями построены на использовании интерфейсов, а не реализаций. Это позволяет создавать экспериментальные или специфичные версии программных продуктов, основанных на платформе Кумир 2.х, с помощью добавления новых модулей и новых конфигураций запуска.

В частности, таким образом реализованы два варианта компилятора: один из них компилирует программу в выполнимый байт-код, а другой (в настоящее время имеет статус экспериментального) – в выполняемый машинный код.

Другим примером построения произвольной конфигурации является разрабатываемая среда для языка Python, которая использует ровно те же модули, что и среда для языка Кумир, но с заменой языковых моду-

лей Кумир на модуль языка Python. Аналогичным образом можно создавать среды для разработки на других языках программирования, которые будут иметь общие с системой Кумир пользовательский интерфейс и особенности работы.

3. Быстродействие программ

Платформа Кумир 2.x содержит два варианта реализации поддержки школьного алгоритмического языка:

1. Анализатор программ, генератор компактного байт-кода, и выполнитель полученного байт-кода. Этот вариант реализован в конфигурациях с пользовательским интерфейсом, и позволяет получить полный контроль над выполнением программы.

2. Анализатор программ, точно такой же, как в варианте использования байт-кода, и генератор биткода LLVM [5], который затем транслируется в машинный код для непосредственного выполнения на процессоре. Этот вариант в настоящее время имеет реализацию, разработанную, в первую очередь, для системы Linux. Имеется экспериментальная поддержка версии для Windows версии не ниже Vista, поскольку для использования LLVM требуется компилятор не старее Visual Studio 2012.

3.1. Выполнение с использованием байт-кода

В отличии от предыдущих (1.x) версий системы Кумир, которые интерпретировали каждую выполняемую строку кода, в новой версии выполняется трансляция исходной программы в набор простейших команд – байт-код. Размер каждой команды – фиксированный, равный 4-м байтам, что позволяет реализовать эффективный стековый выполнитель.

Набор команд байт-кода покрывает необходимый и достаточный функциональный минимум для выполнения на виртуальной машине, состоящей из арифметико-логического блока, стековой памяти и памяти-кучи.

Помимо базового функционального набора команд, предусмотрены команды для управления работой самой виртуальной машины. Они используются при создании «отладочных» версий программ, которые сообщают номера строк, осуществляют вывод значений на поля и т. д. Таким образом, при выполнении программы без показа на полях, и при выполнении программы по шагам, выполняется разный байт-код; в первом случае, выполняется более короткая программа, которая не содержит инструкций, предназначенных для управления самой виртуальной машиной.

3.2. Генерация биткода LLVM

Более радикального ускорения программ можно добиться за счет компиляции в машинный код либо перед выполнением программы, либо во время ее выполнения.

Оба механизма можно реализовать с помощью набора библиотек LLVM, которые также используются для генерации машинного кода в реализации CLang компилятора Си/Си++, и эталонной реализации языка Rust.

Поставка Кумир 2.1.0 для Linux и экспериментальная сборка MSVC2012 для Windows включаются в себя консольный компилятор kumir2-llvmc, который принимает на вход программу на школьном алгоритмическом языке, а результатом его работы является самая обычная программа для запуска без использования виртуальной машины.

Ранее мы уже экспериментировали [7] с генерацией машинного кода, используя язык Си в качестве промежуточного представления. Тогда мы столкнулись с проблемой медленной работы компилятора GCC (а особенно – MinGW, его реализации для Windows). Для ускорения процесса компиляции было принято решение отказаться от использования GCC в пользу CLang для генерации кода, который во-первых компилирует программы заметно быстрее, а во-вторых может быть использован в виде библиотеки, а не отдельно запускаемого процесса. Поскольку программный интерфейс CLang оказался тесно связанным с LLVM, то оказалось более удобным использовать в качестве промежуточного языка при генерации машинного кода именно LLVM, а не Си.

3.3. Эквивалентность выполнения байт-кода и программ в машинном коде

Имея две различные реализации выполнения: байт-код в конфигурации на компьютере пользователя, и биткод LLVM, например, в системе автоматизированного тестирования программ, необходимо гарантировать эквивалентность результатов выполнения одних и тех же исходных программ. Для решения этой задачи, помимо стандартной библиотеки алгоритмического языка, реализованной на Си++, в отдельный модуль были вынесены некоторые части виртуальной машины, выполняющей байт-код. Все это оформлено в виде программного модуля, который один раз компилируется в биткод LLVM, и в дальнейшем связывается с результатом компиляции пользовательских программ. Это позволяет гарантировать одинаковое выполнение программ, что особенно критично при выполнении учебных программ, содержащих ошибки выполнения.

Недостатком такого подхода является нерациональное использование системных ресурсов: для большинства простейших операций приходится вызывать через стек внешние процедуры, изначально предназначенные для реализации виртуальной машины. Эти процедуры не только выполняют требуемые операции, но и осуществляют контроль корректности их выполнения.

3.4. Сравнение производительности Кумир-LLVM с другими языками программирования

Методика оценки. Были реализованы три алгоритма, которые могут встречаться в рамках олимпиадных заданий:

1. Алгоритм Флойда-Уоршелла для оценки скорости работы с массивами;

2. Алгоритм сортировки слиянием для оценки скорости вызова подпрограмм;

3. Вычисление коэффициентов ряда Фурье для оценки скорости выполнения простых программ и операций с плавающей точкой.

Для языка Паскаль использовалась реализация

FreePascal, опции компилятора '-Cr -Co -Ct'. Для языка Си использовался компилятор CLang с с опцией '-O0'. Python – стандартная реализация версии 2.7.

За единицу измерения было взято время выполнения программы на языке Си, скомпилированной без оптимизации компилятором, поскольку машинный (точнее – ассемблерный) код именно этой реализации является легко интерпретируемым для дальнейшего анализа производительности.

Результаты приведены на рис.1. Как видно, в задачах с частым обращением к элементам массива, программы на языке Кумир на порядок медленнее, чем на Си или Паскале. Это связано, в первую очередь, с проверками на определенность элемента массива, которой нет в других компилируемых языках программирования. Отставание Кумир от Си и Паскаля в задаче на рекурсивный вызов алгоритма связан с использованием «вариантных» типов переменных, заимствованных из реализации виртуальной машины (причины использования см. выше). Аналогичная ситуация наблюдается и в простых вычислительных задах.

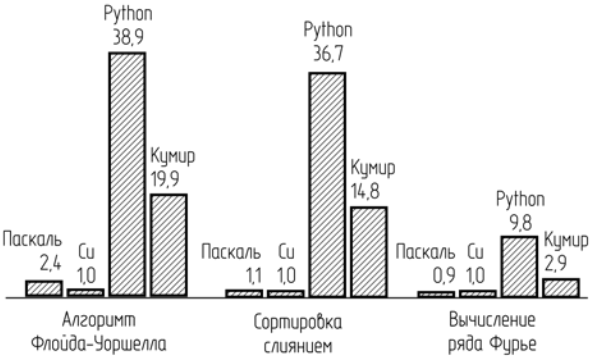


Рис. 1. Сравнение времени выполнения программ на различных языках программирования. За единицу измерения принято время выполнения эквивалентной программы на языке Си

Теоретически возможная скорость работы программ – на уровне языка Паскаль, так что, при необходимости, скорость работы может быть улучшена. В настоящее время это не является для нас первоочередной задачей, поскольку допустимым языком программирования при проведении олимпиад или при выполнении вычислительных задач, как правило может быть и Python. Как видно, реализация Кумир-LLVM является, несмотря на отсутствие каких-либо оптимизаций, в два раза более высокопроизводительной.

4. Система с точки зрения пользователя

4.1. Пользовательский интерфейс

Пользовательский интерфейс системы, в целом, остался очень похожим на интерфейс предыдущих версий системы «Кумир»: центральное место занимает редактор с полями для отображения ошибок и промежуточных значений переменных, а также область ввода-вывода.

В процессе взаимодействия с сообществом пользователей, а также по результатам опытных эксплуатаций, были проведены улучшения интерфейса для улучшения эргономики и адаптации к реальным условиям.

В частности, более гибкой стала система управления вспомогательными окнами: в Кумир 2.1 появилась возможность прикрепления (см. рис. 2) окна исполнителя, или любого другого вспомогательного окна, в фиксированную область главного окна без перекрытия. Это оказалось необходимым для работы в компьютерных классах с ноутбуками, у которых диагональ экрана не превышает 15 дюймов.

В то же время, остается возможность произвольного размещения вспомогательных окон, если используется большой экран, или экран совместно с проектором. Переключение режима отображения окна осуществляется одной кнопкой в заголовке. Для реализации этой возможности пришлось отказаться от использования стандартного оформления окон операционной системой, и реализовать немного не стандартный вид заголовка и кнопок.

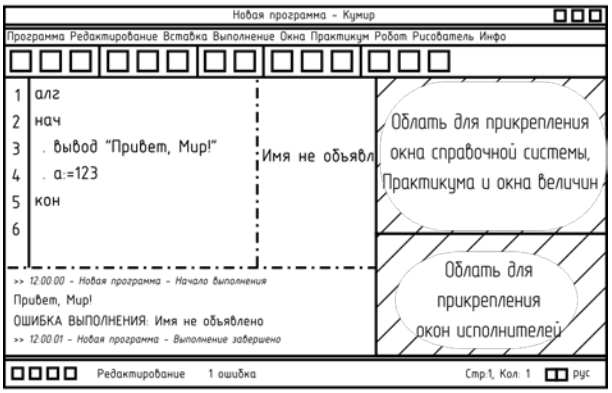


Рис. 2. Пользовательский интерфейс системы, адаптированный для экранов ноутбуков

Не обошли стороной и пользователей с ограниченными физическими возможностями. Система Кумир следует настройкам операционной системы в случае использования высококонтрастных тем: она может инвертировать цвета редактора при использовании темного фона. Значки на панели инструментов реализованы монохромными, и их цвет подстраивается во время запуска под настройки операционной системы.

Помимо удобства использования системы слабовидящими пользователями, мы стараемся сделать интерфейс Кумира доступным и для людей с нарушениями моторики: работать с системой, в большинстве ситуаций, можно и без мыши, используя только клавиатуру.

4.2. Практикум

Модуль поддержки практикумов в системе Кумир 2.x наследует все возможности аналогичного модуля, реализованного в системе Кумир 1.9. Практикум содержит набор заданий, имеющий древовидную структуру (разбитый на разделы, подразделы и т.д.); в каж-

дом задании ученик должен написать программу. Задание представляет собой заготовку, которая включает заголовок нужной программы, необходимые комментарии и, возможно, отдельные элементы программы; ученик не может изменять строки заготовки, он может только дополнить их своими строками.

Подготовив программу – решение задания, ученик может проверить ее, используя встроенную систему тестов; по результатам проверки за задание выставляется оценка. Все программы, отправлявшиеся на проверку, сохраняются и (при желании) доступны для последующей проверки учителем.

Для подготовки заданий практикумов используется специальная учительская конфигурация системы; создание своего практикума и редактирование уже готовых практикумов, как показывает опыт, доступно квалифицированному учителю [8].

4.3. Инструменты для пакетной обработки

Важной особенностью новой платформы является наличие конфигураций, предназначенных для выполнения на сервере без использования графического интерфейса пользователя. Реализованы консольные инструменты: компилятор в байт-код, соответствующий ему выполнитель байт-кода, а также компилятор в машинный код. Эти инструменты являются необходимыми для организации автоматизированного тестирования программ. В отличии от ранней реализации Кумир для КТС-ЕГЭ, данные инструменты не являются специально подготовленной версией, а используют ровно те же модули системы, что и версия с пользовательским интерфейсом.

5. Дальнейшие направления разработки

5.1. Упрощение разработки новых исполнителей

Сообщество пользователей системы Кумир – это не только учителя информатики, но и разработчики оригинальных методик и собственных программных разработок [9]. Мы хотим дать им возможность самостоятельной реализации необходимых им исполнителей. Исходные тексты платформы Кумир 2.x открыты; они, в частности, содержат вспомогательный скрипт и документацию разработчика, облегчающие создание новых исполнителей. В дальнейшем планируется создание специализированной поставки (SDK), ориентированной на разработчиков дополнений.

5.2. Использование платформы для реализации сред на других языках программирования

Модульная архитектура платформы Кумир 2.x позволяет создавать решения на базе существующих компонент. В частности, мы разрабатываем аналогичную систему для языка программирования Python 3.x. Как и система Кумир для школьного алгоритмического языка, среда для Python на базе Кумир ориентирована на обучение программированию, а не на профессиональное программирование. Эта среда использует ряд компонентов системы Кумир: модуль пользовательского интерфейса, систему поддержки практикумов, и может даже использовать «исполнители» системы Кумир (правда, в случае с Python, это не для всех исполнителей имеет смысл из-за отличающейся методики обучения).

Помимо реализации среды для Python, мы экспериментировали с созданием аналогичной среды для языка Паскаль.

5.3. Поддержка курса робототехники

У нас есть опыт в программировании с помощью системы Кумир 1.9 обучающего робота Lego NXT [10]. Эксплуатация этого робота подразумевает использование канала радиосвязи Bluetooth для телеметрии, при этом сама программа выполняется на компьютере. Опыт показал, что такое взаимодействие накладывает ограничения на возможности использования из-за временных задержек радиоканала. Например, реализация алгоритма обхода лабиринта по правилу правой руки надежно работает только в том случае если робот двигается медленно и осторожно, иначе он просто не успевает обмениваться данными с компьютером и допускает неточности при поворотах.

Решением проблемы с задержками в радиосвязи является перенос выполнения программы со стороны компьютера на сторону микроконтроллера, установленного в роботе. Недавнее обновление линейки Lego NXT – модель Lego EV3 обладает достаточными ресурсами для выполнения интерпретатора байт-кода системы Кумир 2.x, а сам байт-код является более компактным, чем выполняемый код, поэтому его проще передавать по радиоканалу.

Не имея возможности эксплуатации Lego EV3, мы обеспечили работоспособность интерпретатора байт-кода Кумир на одноплатном модуле Raspberry Pi, который основан на процессоре с ARM-архитектурой и обладает скромными техническими характеристиками.

В дальнейшем мы планируем портировать интерпретатор для выполнения на процессоре Lego EV3 и реализовать связь интерпретатора с компьютером.

5.4. Перемещение в «облако»

Современные тенденции в программном обеспечении ведут к тому, что скоро на персональных компьютерах останется только одна программа – браузер, а привычные нам приложения станут Web-приложениями. Сегодня в виде Web-приложений существуют не только офисные приложения (Google Docs, Microsoft Office 365 и др.), но даже графические редакторы и CAD-системы. Целесообразно создать Web-версию системы Кумир. Это достаточно трудоемкий, но необходимый процесс, который предстоит нам в недалеком будущем.

System KUMIR 2.X

A.G. Kushnirenko, M.A. Roytberg, D.V. Hachko, V.V. Yakovlev

Abstract: KUMIR 2.x – is a programming environment designed to support the course of programming in primary and secondary schools. It is a reincarnation and development of KUMIR, created in the second half of the 1980s and uses the so-called russian algorithmic language – Algol-like language with Russian vocabulary. Otherness KUMIR 2.x is the support of automatic check of students programm correctness (both syntactic and semantic) and significantly increased (compared to previous versions) speed of execution.

Keywords: Kumir, algorithmic language, system, program.

Литература

1. А. Г. Кушниренко, А. Г. Леонов, М. А. Ройтберг, Д. В. Хачко, В. В. Яковлев, А. В. Карпов, Н. М. Субоч. «Система программирования КУМИР – интегрированная поддержка начальных курсов информатики и программирования». III Конференция «Свободное программное обеспечение в высшей школе», М.: AltLinux, 2008.
2. https://gitlab.com/niisi/kumir2
3. Информатика: 7–9 кл.: Учеб. для общеобразоват. учр. А. Г. Кушниренко, Г. В. Лебедев, Я. Н. Зайдельман. М.: Дрофа, 2003. 335 с.
4. Информатика. 10 класс. Углубленный уровень. В 2 частях (комплект из 2 книг). К. Ю. Поляков, Е. А. Еремин. М.: Бином, 2014. 648 с.
5. "Introduction to the LLVM Compiler System" Chris Lattner Plenary Talk, ACAT 2008: Advanced Computing and Analysis Techniques in Physics Research, Erice, Sicily, Italy, Nov. 2008.
7. М. А. Ройтберг, В. В. Яковлев. «Конвертор КУМИР – C++: поддержка перехода от учебных к профессиональным системам программирования». III Конференция «Свободное программное обеспечение в высшей школе», М.: AltLinux, 2008
8. http://kpolyakov.spb.ru/school/kumir.htm
9. http://www.vinforika.ru/index.php/mnu-umki
10. http://www.lego.com/ru-ru/mindstorms/