13. *Novak V., Perfil'eva I., Mochkrozh I.* Matematicheskie printsipy nechetkoy [Mochkrozh I. Mathematical principles of fuzzy logic]. Moscow: Fizmatlit, 2006, 352 p.
14. *Lapidus V.A.* Sistema Shukharta [Shukhart System]. Nizhniy Novgorod: OOO SMTs «Prioritet», 2004, 65 p. ISBN 5-98366-010-1.
15. *Barabanova O.A.* Sem' instrumentov kontrolya kachestva [Seven quality control tools]. Moscow: ITs «MATI» - RGTU im. Tsiolkovskogo, 2001, 88 p.
16. *Spears W.M.* Adapting crossover in a genetic algorithm. The University of Michigan Press, 1988.
17. *Donald J. Wheeler.* Advanced Topics in Statistical Process Control: The Power of Shewhart's Charts. SPC Press, 1995.
18. *Romanov V.N.* Nechetkie modeli prinyatiya resheniy [Fuzzy models of decision-making], *Al'manakh sovremennoy nauki i obrazovaniya* [Almanac of modern science and education], 2013, No. 5 (72), pp. 144-147.
19. *Grant V.* Evolyutsionnyy protsess: Kriticheskiy obzor evolyutsionnoy teorii [The evolutionary process: A critical review of evolutionary theory]: Transl. from English. Moscow: Mir, 1991, 488 p.
20. *Shmal'gauzen I.I.* Izbrannye trudy. Organizm kak tseloe v individual'nom i istoricheskom razvitii [Selected Works. The organism as a whole in individual and historical development]. Moscow: Nauka, 1982, pp. 348-372.
21. *Goldberg D.E., Sastry K.* A Practical Schema Theorem for Genetic Algorithm Design and Tuning, Illinois Genetic Algorithms Laboratory, 2001.
22. *Herrera F., Lozano M., Verdegay J.L.* Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis, 1996. Department of Computer Science and Artifcial Intelligence University of Granada, Spain.
23. *Rosenbrock H.H.* An automatic method for finding the greatest or least value of a function, *The Computer Journal*, 1960, No. 3, pp. 175-184.
24. *Rastrigin L.A.* Systems of extremal control. Moscow: Mir, 1974.

Статью рекомендовал к опубликованию к.т.н. А.А. Кажаров.

Курейчик Виктор Михайлович – Южный федеральный университет; e-mail: vmkureychik@sfedu.ru; 347?928, г. Таганрог, пер. Некрасовский, 44; тел.: 88634681887; кафедра САПР; г.н.с.; д.т.н.; профессор.

Каплунов Тимофей Геннадьевич – e-mail: tkaplunov@sfedu.ru; тел.: 89515359742; кафедра САПР; аспирант.

Kureichik Viktor Mikhaylovich – Southern Federal University; e-mail: vmkureychik@sfedu.ru; 44, Nekrasovskiy, Taganrog, 347928, Russia; phone: +78634681887; the department of CAD; chief researcher; dr. of eng. sc.; professor.

Kaplunov Timofey Gennadyevich – e-mail: tkaplunov@sfedu.ru; phone: +79515359742; the department of CAD; graduate student.

УДК 004.432.4 DOI 10.23683/2311-3103-2018-5-34-48

И.И. Левин, А.И. Дордопуло, И.В. Писаренко, А.К. Мельников

ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДОМ ЯКОБИ НА ЯЗЫКЕ АРХИТЕКТУРНО-НЕЗАВИСИМОГО ПРОГРАММИРОВАНИЯ SET@L

Для большинства существующих языков программирования характерна проблема архитектурной специализации, которая заключается в необходимости разработки нового кода при портировании параллельных программ между вычислительными системами с разными архитектурами. Эта проблема может быть решена с помощью разработанного языка архитектурно-независимого программирования Set@l, который основан на принципах теоре-

тико-множественного представления исходного кода и аспектно-ориентированного программирования. Программа на языке Set@l состоит из исходного кода, описывающего информационный граф задачи в архитектурно-независимой форме, и аспектов, которые адаптируют алгоритм к архитектуре и конфигурации вычислительной системы. Если аспекты не изменяют алгоритм в процессе адаптации, то решение задачи и особенности его распараллеливания могут быть описаны в рамках теории множеств Кантора-Больцано. В случае модернизации алгоритма некоторые совокупности выделены нечетко и не являются множествами. Для описания подобных объектов в языке Set@l используется альтернативная теория множеств П. Вопенка. Совокупности, тип и структура которых на некотором уровне абстракции определены нечетко, обозначаются в языке Set@l как классы. Если нечеткость является неотъемлемой характеристикой совокупности, то она относится к типу «полумножество». Оперируя классами, множествами и полумножествами, возможно описать различные способы реализации и распараллеливания алгоритма в единой аспектно-ориентированной программе на языке Set@l. В данной работе особенности использования объектов альтернативной теории множеств в языке Set@l рассмотрены на примере алгоритма решения систем линейных алгебраических уравнений методом Якоби, модернизация которого целесообразна при реализации на вычислительных системах с реконфигурируемой архитектурой.

Архитектурно-независимое программирование; язык программирования Set@l; альтернативная теория множеств; аспектно-ориентированный подход.

I.I. Levin, A.I. Dordopulo, I.V. Pisarenko, A.K. Melnikov

DESCRIPTION OF JACOBI ALGORITHM FOR SOLUTION OF LINEAR EQUATION SYSTEM IN ARCHITECTURE-INDEPENDENT SET@L PROGRAMMNING LANGUAGE

Software porting between high-performance computer systems with different architectures requires a considerable code revision due to the architectural limitation of existing programming languages. To solve the problem, we proposed architecture-independent Set@l programming language based on the principles of set-theoretic codeview and aspect-oriented programming. Set@l program consists of a source code, which describes an information graph of a problem, and aspects, which adapt an algorithm to architecture and configuration of a computer system. If an algorithm remains unchanged during its adaptation, calculations and their parallelizing are described by the set theory of Cantor and Bolzano. In the case of algorithm modernization, some collections are indefinite, and we can not treat them as traditional sets. To describe indefinite objects, Set@l applies the alternative set theory developed by P. Vopenka. In Set@l, a class is a collection characterized by indefinite type and structure at some abstraction level. If indefiniteness represents an essential attribute of a collection, it is classified as a semiset. Application of classes, sets and semisets allows to describe various methods of algorithm implementation and parallelizing as an entire Set@l program. In this paper, the Jacobi algorithm for the solution of linear equation systems is considered as an example for the utilization of classes and semisets.

Architecture-independent programming; Set@l programming language; alternative set theory; aspect-oriented approach.

Введение. Портирование параллельных программ между вычислительными системами (ВС) с различными архитектурами, как правило, сводится к разработке нового кода, что обусловлено архитектурной специализацией существующих языков программирования и отсутствием эффективных методов и средств архитектурно-независимого описания алгоритмов. Для решения данной проблемы в работах [1, 2] предложен язык архитектурно-независимого программирования ВС Set@l, являющийся дальнейшим развитием базовых принципов высокоуровневого языка COLAMO [3–5] и теоретико-множественного языка программирования SETL [6–8]. В отличие от других языков программирования на основе теории множеств, Set@l использует типизацию совокупностей по параллелизму, четкости выделения элементов и другим критериям. Язык Set@l основан на парадигме ас-

пектно-ориентированного программирования [9–12], в соответствии с которой программа состоит из исходного кода, описывающего алгоритм решения задачи в архитектурно-независимой форме, и аспектов, определяющих особенности реализации алгоритма на ВС с различными архитектурами.

Исходный код программы на языке Set@l описывает информационный граф решаемой задачи в виде множеств и отношений между ними. Архитектурная независимость исходного кода обусловлена неопределенностью типов некоторых совокупностей алгоритма и их разбиений на подмножества. Система аспектов задает различные варианты декомпозиции совокупностей, дополняет и переопределяет их признаки, адаптируя алгоритм к реализации на ВС с определенными архитектурой и конфигурацией.

Одним из основных критериев классификации совокупностей в языке Set@l является характер параллелизма их элементов. Формируя комбинации из совокупностей с различными типами параллелизма, возможно описать любые способы распараллеливания алгоритмов.

Если аспекты программы на языке Set@l не изменяют алгоритм в процессе его адаптации к архитектуре ВС, то решение задачи может быть описано в рамках классической теории множеств Кантора-Больцано [13], что показано в работе [1]. Однако в ряде случаев аспекты должны не только определять различные варианты распараллеливания вычислений, но и осуществлять модернизацию алгоритма в соответствии с особенностями его реализации на ВС с определенной архитектурой. В таких случаях некоторые совокупности выделены нечетко и не являются множествами, поэтому их невозможно описать на языке теории множеств Кантора-Больцано. Неопределенность типа совокупности по параллелизму, рассмотренная в статье [1], также является проявлением нечеткости, для описания которой введен специальный тип **imp**.

Язык архитектурно-независимого программирования Set@l позволяет описывать разные способы реализации одного и того же алгоритма в единой аспектно-ориентированной программе. Для этого вводится классификация совокупностей по четкости выделения их элементов, предложенная в альтернативной теории множеств П. Вопенка [14–17]. Особенности использования и описания объектов альтернативной теории множеств в языке Set@l рассмотрены на примере алгоритма решения системы линейных алгебраических уравнений (СЛАУ) методом Якоби [18], модернизация которого целесообразна при реализации на ВС с реконфигурируемой архитектурой [19].

Способы реализации алгоритма метода Якоби и их теоретико-множественное описание. Реализация решения СЛАУ методом Якоби на ВС может осуществляться двумя основными способами: с проверкой условия завершения вычислений на каждой итерации (рис. 1,а) и проверкой после нескольких идущих подряд вычислительных итераций (рис. 1,б).

В случае, изображенном на рис. 1,а, каждая итерация алгоритма включает операции пересчета вектора-столбца неизвестных (блок *C*) и проверки условия, определяющего завершение вычислений (блок *V*). Сразу после выполнения условия $err \le \delta$ управляющее устройство (УУ) осуществляет транзит данных через незадействованные аппаратные блоки и обеспечивает сохранение результата в специально выделенную область распределенной памяти. Рассматриваемый вариант реализации полностью соответствует математическому описанию алгоритма решения СЛАУ методом Якоби, однако на практике оказывается эффективным не для всех вычислительных архитектур. Данная особенность обусловлена тем, что каждый блок проверки условия *V* реализует ресурсоемкую операцию расчета невязки *err(k)*, которая занимает аппаратный ресурс, сопоставимый с операцией пересчета матрицы *C*, и характеризуется сравнимыми с пересчетом временными затратами.

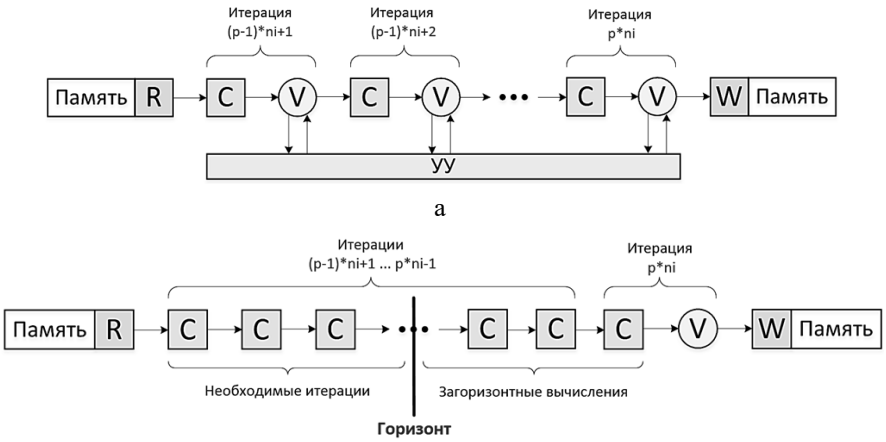

Рис. 1. Способы реализации алгоритма решения СЛАУ методом Якоби на ВС с реконфигурируемой архитектурой: с проверкой на каждой итерации (а) и одной проверкой после ni вычислительных итераций (б)

Для повышения эффективности использования вычислительных ресурсов и сокращения времени решения задачи на реконфигурируемых ВС целесообразно осуществить модернизацию алгоритма решения СЛАУ методом Якоби. Данная модернизация подразумевает однократную проверку условия завершения вычислений в кадре (рис. 1,б). В представленном случае под кадром понимается совокупность аппаратно реализованных операций, объединенных в единую вычислительную структуру, которая выполняет функциональные преобразования входных информационных потоков в выходные [20]. Если в блоке итераций условие завершения вычислений выполнится раньше, чем данные достигнут элемента *V*, то дополнительные итерации не приведут к ухудшению результата. Вместе с тем, рассматриваемый способ реализации алгоритма позволяет сократить время решения задачи, так как освобожденный от блоков проверки *V* аппаратный ресурс будет использован для наращивания числа блоков пересчета *C* в кадре, задаваемого параметром *ni*. Отметим, что оба рассмотренных варианта организации вычислений могут быть реализованы как на реконфигурируемой ВС, так и на ВС с мультипроцессорной архитектурой.

Рассмотрим описание представленных на рис. 1 вариантов реализации алгоритма решения СЛАУ методом Якоби на языке теории множеств.

В случае, который изображен на рис. 1, а, проверка условия $err \le \delta$ осуществляется на каждой итерации алгоритма, что позволяет четко определить номер последней итерации I_m. Несмотря на то, что значение I_m заранее неизвестно, совокупность итераций алгоритма *K* может быть задана как множество с помощью известного условия, определяющего завершение вычислений:

$$K = \mathbf{set}\Big(k \mid k \in \mathbb{N} \ \& \ \big(k = 1 \ \mathbf{or} \ err\big(k-1\big) > \delta\big)\Big), \tag{1}$$

где $\mathbb{N}$ – совокупность натуральных чисел. Итерации с номерами 1 и I_m (если $I_m \neq 1$) в любом случае входят в множество *K*, так как проверка условия всегда реализуется после соответствующего пересчета матрицы. Информационный граф алгоритма *F* описывается присвоением признаков пересчета *C* и проверки *V* каждому элементу множества *K* и, как следствие, также является множеством:

$$F = \mathbf{set}\left(\; k^{\,C,V} \; | \, k \in K \; \right), \tag{2}$$

где $[\![\;]\!]$ – обозначение неопределенности с точки зрения параллелизма. С учетом формул (1) и (2) и разбиения на блоки для распараллеливания по итерациям, множества K и F будут иметь следующий вид:

$$K = \left[\!\!\left[\; 1 \; \ldots \; ni \; , \; ni+1 \; \ldots \; 2 \cdot ni \; , \; \ldots \; , \left[\!\!\left[(T-1) \cdot ni + 1 \; \ldots \; I_m \right]\!\!\right] \right]\!\!\right]; \tag{3}$$

$$F = \left[\!\!\left[\left[\!\!\left[1^{\,C,V} \; \ldots ni^{\,C,V} \; \right]\!\!\right], \left[\!\!\left[(ni+1)^{\,C,V} \; \ldots (2 \cdot ni)^{\,C,V} \; \right]\!\!\right], \ldots, \left[\!\!\left[((T-1) \cdot ni + 1)^{\,C,V} \; \ldots I_m{}^{\,C,V} \; \right]\!\!\right] \right]\!\!\right], \tag{4}$$

где T – номер блока итераций, в котором выполнится условие $err \le \delta$; ni – число итераций в блоке. В соответствии с приведенными выше рассуждениями, вариант реализации алгоритма с проверкой на каждой итерации описывается в рамках классической теории множеств Кантора-Больцано.

Наибольший интерес с точки зрения теоретико-множественного описания представляет вариант реализации алгоритма, рассмотренный на рис. 1,б. С помощью четко заданных множеств, которыми оперирует теория множеств Кантора-Больцано, может быть описан только один частный случай K^*, соответствующий выполнению условия на итерации с номером $T \cdot ni$:

$$K^* = \left[\!\!\left[\; 1 \; \ldots \; ni \; , \; ni+1 \; \ldots \; 2 \cdot ni \; , \; \ldots \; , \left[\!\!\left[(T-1) \cdot ni + 1 \; \ldots \; T \cdot ni \right]\!\!\right] \right]\!\!\right], \tag{5}$$

где T – номер последнего блока итераций, определяемый выполнением условия $err(T \cdot ni) \le \delta$. Во всех остальных случаях множества K и K^* описывают принципиально разные с математической точки зрения объекты, так как совокупность K^* включает не только необходимые, но и избыточные итерации (рис. 1,б), а четко определить положение точки выполнения условия $err \le \delta$ не представляется возможным: можно точно определить, что к итерации $T \cdot ni$ условие выполнится, а на итерации $(T-1) \cdot ni$ оно еще не выполнялось. Последний элемент K^*, в отличие от элемента I_m в множестве K, указывает лишь на границу подмножества, которому принадлежит номер последней итерации. Чтобы не нарушать семантику определения совокупности итераций K и обеспечить унификацию обозначений объектов во всех модулях аспектно-ориентированной программы, необходимы новые методы и средства описания, которые выходят за рамки классической теории множеств.

Одной из областей математики, в которых исследуются нечеткие совокупности, является *альтернативная теория множеств*, разработанная чехословацким математиком П. Вопенка.

Под *множеством* в теории П. Вопенка понимается четко выделенная совокупность объектов, которая характеризуется индивидуальностью и представляется как самостоятельная и целостная единичность [14]. Для множества всегда известен точный ответ на вопрос, принадлежит ли совокупности тот или иной из рассматриваемых объектов. В действительности, многие естественно возникающие совокупности не являются множествами, так как выделены нечетко. Альтернативная теория множеств рассматривает феномен нечетких совокупностей с помощью особых математических объектов – классов и полумножеств.

Класс определяется аналогично множеству, однако не требует четкого выделения соответствующей совокупности объектов [14]. Тем не менее, для любого элемента класса существует понятие принадлежности в его классическом смысле, то есть никакой объект не может одновременно принадлежать и не принадлежать определенному классу. Так как класс выделен нечетко, то не всегда возможно точ-

но определить, принадлежит ли совокупности тот или иной объект или нет. По аналогии с подмножеством, в альтернативной теории множеств существует понятие *подкласса*, отражающее отношение включения между различными нечетко выделенными объектами. Если класс является подклассом некоторого четко выделенного множества, то он представляет собой *полумножество* [14]. Понятие полумножества используется для описания тех случаев, когда нечеткая совокупность выделена из четко заданного множества.

В работах П. Вопенка категория *нечеткости*, лежащая в основе понятий класса и полумножества, тесно связана с понятием горизонта. Под *горизонтом* понимается некоторый виртуальный объект, при движении в направлении которого проявляется феномен нечеткости. В отличие от четко выделенных границ, горизонт не занимает строго определенного положения и может перемещаться. Те объекты, которые лежат перед горизонтом, всегда выделены нечетко, а приближение к горизонту усиливает нечеткость их представления. Нечеткость в выделении полумножества и класса означает, что существует один или несколько горизонтов, ограничивающих наш взгляд на рассматриваемые объекты.

При реализации алгоритма решения СЛАУ методом Якоби с одной проверкой условия в кадре (рис. 1,б) совокупность итераций K является полумножеством – классом, связанным с множеством K^* отношением включения $K \subseteq K^*$. Роль горизонта в рассматриваемом случае играет точка выполнения условия завершения вычислений $err \le \delta$, определить точное положение которой из-за особенностей реализации алгоритма (см. рис. 1,б) не представляется возможным. В зависимости от ряда факторов (например, начального приближения и особенностей исходной матрицы) горизонт может перемещаться, формируя разные варианты заполнения совокупности K. В том случае, когда условие выполняется на итерации с номером $T \cdot ni$, горизонт переходит в четкую границу, а полумножество K – в четко выделенное множество итераций, совпадающее с множеством K^*. Разность множества K^* и полумножества K в общем случае соответствует полумножеству итераций, на которых условие $err \le \delta$ уже выполняется, однако вычисления не прерываются из-за отсутствия возможности его проверки. Данное полумножество описывает *загоризонтные вычисления*, которые не являются необходимыми с математической точки зрения, но не приводят к ухудшению результата и возникают из-за особенностей выбранного способа реализации алгоритма.

Для того, чтобы задать полумножество итераций K как математический объект, необходимо сформировать надмножество K^* и описать соответствующее отношение включения:

$$K^*_{sub}(k) = \mathbf{set}\left(\; k_1 \ldots k_2 \;\mid\; k_1 = (k-1)\cdot ni + 1 \;\;\&\;\; k_2 = k \cdot ni \; \right); \qquad (6)$$

$$K^* = \mathbf{set}\left(\; K^*_{sub}(k) \;\mid\; k \in \mathbb{N} \;\;\&\;\; \left(k = 1 \;\; \mathbf{or} \;\; err\left((k-1)\cdot ni \right) > \delta \right) \; \right); \qquad (7)$$

$$\mathbf{sm}(K) \subseteq \mathbf{set}(K^*), \qquad (8)$$

где $K^*_{sub}(k)$ – k-е подмножество K^*, соответствующее k-му блоку итераций; **sm** – обозначение типа совокупности как «полумножество». Алгоритм F для варианта реализации метода Якоби с одной проверкой условия после ni вычислительных итераций описывается следующей реляционной конструкцией:

$$F = \mathbf{sm}\Big(P(k) \;\mid\; k \in K \;\;\&\;\; \left(\mathbf{mod}(k, ni) = 0 \to P = \; C, V \; \right) \;\;\&\;\; \ldots$$

$$\ldots \;\;\&\;\; \left(\mathbf{mod}(k, ni) != 0 \to P = \; C \; \right), \qquad (9)$$

где **mod** – остаток от деления. Отметим, что совокупность *F* содержит дополненные признаками пересчета *C* и проверки условия *V* элементы совокупности *K*, поэтому также является полумножеством. С учетом разбиения на блоки по итерациям, полумножества *K* и *F* будут иметь следующие структуры:

$$K = \{\!|\ 1 \ldots ni\ ,\ ni+1 \ldots 2 \cdot ni\ ,\ \ldots,\ (T-1) \cdot ni+1 \ldots \mathbf{??};\quad (10)$$

$$F = \{\!|\{\!|1^{\,C} \ldots ni^{\,C,V}\ |\!\},\{\!|(ni+1)^{\,C} \ldots (2 \cdot ni)^{\,C,V}\ |\!\},\ \ldots,\{\!|((T-1) \cdot ni+1)^{\,C} \ldots \mathbf{??}, \quad (11)$$

где **?** – обозначение нечеткости в выделении совокупности, которая обусловлена присутствием горизонта, не позволяющего точно определить номер последней необходимой итерации. Как следует из выражений (10) и (11), *T*-е подклассы совокупностей *K* и *F* являются полумножествами, а их объединения с четко заданными подмножествами полных блоков итераций также образуют полумножества.

Как отмечалось в работе [1], с точки зрения четкости выделения элементов любая совокупность в языке архитектурно-независимого программирования Set@l может относиться к одному из следующих типов: «множество» (**set**), «полумножество» (**sm**) или «класс» (**cls**). Типы «множество» и «полумножество» соответствуют четко и нечетко выделенным совокупностям, а тип «класс» – совокупностям, тип которых не может быть определен однозначно. Использование классов обеспечивает уникальность обозначения объектов во всех модулях аспектно-ориентированной программы на языке Set@l. Например, в рассматриваемом алгоритме решения СЛАУ методом Якоби совокупность итераций *K* и алгоритм *F* могут быть множествами (см. выражения (1)–(4)) или полумножествами (см. выражения (6)–(11)) в зависимости от выбранного способа реализации, что описывается в аспекте метода обработки. При формировании исходного кода программы способ реализации алгоритма еще неизвестен, поэтому совокупность *K* не может быть отнесена к какому-либо конкретному типу и обозначается как класс.

Чтобы описать какую-либо совокупность как класс, необходимо присвоить ей признак **cls** и в любом модуле программы выделить возможные варианты ее типизации с помощью конструкции **typing**:

cls(<*имя класса*>**);**

typing (<*имя класса*>**) :** <*тип 1*> **or** <*тип 2*> **or** … ;

Класс – наиболее общий и универсальный тип совокупностей в языке Set@l. Если в каком-либо модуле программы необходимо ввести совокупность, тип и структура которой на данном уровне абстракции не могут быть четко определены, то достаточно обозначить ее как класс и использовать в коде аналогично классическим множествам. Доопределение в одном из аспектов позволяет конкретизировать тип и заполнение совокупности при трансляции. Фактически, неопределенность по параллелизму, для обозначения которой в языке Set@l введен особый тип **imp** [1], также может быть описана с помощью классов.

Описание алгоритма метода Якоби на языке Set@l. Одной из особенностей языка архитектурно-независимого программирования ВС Set@l является типизация совокупностей по различным критериям. В дополнение к базовой типизации по параллелизму и четкости выделения элементов совокупности [1], в программах на языке Set@l могут использоваться любые пользовательские признаки, объявляемые с помощью синтаксической конструкции **attribute** в следующем формате:

attribute <*имя признака*> (<*множество или элемент*>) **:**
<*описание признака*>;
end (<*имя признака*>);

Признак, присваиваемый совокупности или отдельному элементу, определяет способ обработки либо задает отношения между этим элементом и различными объектами программы. Как и любые другие объекты в языке Set@l, признаки могут образовывать совокупности различных типов.

Так, при описании алгоритма решения СЛАУ методом Якоби на языке Set@l целесообразно ввести следующие признаки, присваиваемые элементам совокупности номеров итераций *K*:

♦ признак пересчета *C*, который связывает номер итерации $k \in K$ с совокупностью, описывающей информационный граф основной вычислительной операции метода Якоби:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \cdot \left(b_i - \sum_{1 \le j \le n,\ j \ne i} a_{i,j} \cdot x_j^{(k)} \right); \quad i = 1, 2, \ldots, n, \tag{12}$$

где *i, j* – индексы номеров строк и столбцов; *n* – размерность СЛАУ; *a* – матрица коэффициентов СЛАУ; *b* – вектор-столбец свободных членов; $x^{(k)}$ – вектор неизвестных на *k*-й итерации;

♦ признак проверки условия завершения вычислений *V*, который устанавливает соответствие между номером итерации и совокупностью, описывающей информационный граф операции расчета невязки *err(k)* с последующей проверкой условия $err(k) \le \delta$, где δ – допустимая погрешность решения СЛАУ, и остановкой вычислительного процесса при выполнении этого условия. Вычисление невязки осуществляется по формуле:

$$err(k) = \max\left(\left| x_i^{(k+1)} - x_i^{(k)} \right| \right), \quad i = 1, 2, \ldots, n. \tag{13}$$

В сущности, признаки *C* и *V* описывают отношение между совокупностью итераций *K* и теоретико-множественным представлением информационного графа алгоритма решения СЛАУ методом Якоби *F*, разбитым на подмножества по итерациям.

Фрагмент кода, описывающий признаки пересчета *C* и проверки *V* на языке Set@l, представлен на рис. 2. Признак пересчета или проверки может быть присвоен номеру итерации алгоритма *k*, который является элементом совокупности итераций *K*, что отражено в строках (1) и (7). В строке (2) задано использование начального приближения, хранящегося в отдельном множестве *x_init*, на первой итерации вычислений. Символ «*» обозначает перебор по всем элементам множества или подмножества, что эквивалентно понятию среза в языке COLAMO [4]. Для формирования условной конструкции в строке (2) использована логическая операция импликации, обозначенная символом «–>». Цикл **forall** (строки 3–5) осуществляет перебор по элементам совокупности номеров строк *I*. В строке (4) для описания суммирования по элементам совокупности номеров столбцов *J* использована следующая конструкция реляционного исчисления:

sum(<*член суммы*> **|** <*предикат для индекса суммирования*>**)**.

В строке (8) выполняется поиск максимального элемента множества, которое задано как совокупность разностей соответствующих элементов векторов неизвестных на (*k*+1)-й и *k*-й итерациях. В строке (9) описаны проверка условия остановки вычислительного процесса и сохранение последнего рассчитанного приближения вектора-столбца неизвестных в отдельное множество *x_res* в случае выполнения условия.

```
(1)   attribute C(element(k) in K) :        // признак операции пересчета вектора неизвестных;
(2)       k=1 –> x(k,*)=x_init(*);
(3)       ( forall i in I ) :
(4)           x(k+1,i)=( b(i)–sum(a(i,j)*x(k,j) | j in J and j!=i) )/a(i,i);
(5)       end (forall);
(6)   end (C);
(7)   attribute V(element(k) in K) :        // признак операции проверки условия;
(8)       err(k)=max( abs( x(k+1,i)–x(k,i) ) | i in I );
(9)       err(k)<=delta –> x_res(*)=x(k+1,*);
(10)  end (V);
```

Рис. 2. Описание признаков пересчета C и проверки условия V на языке Set@l

Фрагмент исходного кода, описывающий алгоритм решения СЛАУ методом Якоби на языке Set@l, приведен на рис. 3. Помимо данного фрагмента, в исходный код входят описания признаков пересчета *C* и проверки *V* (см. рис. 2). В строках (1)–(4) объявлены следующие основные совокупности алгоритма:

- совокупности строк и столбцов матрицы *I* и *J*;
- совокупность итераций обработки матрицы *K*;
- совокупность *F*, являющаяся теоретико-множественным представлением информационного графа рассматриваемого алгоритма.

Совокупности строк (*I*) и столбцов (*J*) – множества, так как входящие в них элементы четко определены размерностью матрицы коэффициентов. Совокупность итераций (*K*) и алгоритм (*F*) в зависимости от выбранного способа реализации могут быть множествами или полумножествами, поэтому в рамках исходного кода они определяются как классы. В строках (5)–(7) информационный граф *F* описан как отношение между тройкой множеств (*x_init*, *a*, *b*) и множеством *x_res*, где *x_init* – множество начального приближения, *a* и *b* – множества коэффициентов и свободных членов СЛАУ, *x_res* – множество, в которое сохраняется результат вычислений. Строки (8)–(10) задают обобщенный алгоритм *F* как объединение (**union**) двух подклассов (**sub**) *CI* и *VI*, которые включают номера итераций пересчета и проверки с соответствующими признаками *C* и *V* (см. рис. 2). В строках (8) и (9) декартовы произведения совокупностей обозначены ключевым словом **prod**.

```
(1)   set(I);                    // множество строк матрицы;
(2)   set(J);                    // множесство столбцов матрицы;
(3)   cls(K);                    // класс итераций обработки матрицы;
(4)   cls(F);                    // класс, описывающий информационный граф алгоритма;
(5)   graph (F) :
(6)       x_res=F(x_init,a,b);   // отношение, описываемое графом F;
(7)   end (F);
(8)   CI=sub(prod(C,K));         // подкласс итераций пересчета;
(9)   VI=sub(prod(V,K));         // подкласс итераций проверки;
(10)  F=union(CI,VI);            //  F как объединение подклассов CI и VI;
```

Рис. 3. Фрагмент исходного кода программы на языке Set@l

На рис. 4 представлен фрагмент кода аспекта метода обработки для программы на языке Set@l, реализующей алгоритм решения СЛАУ методом Якоби. Данный фрагмент включает два основных раздела: формирование разбиений множеств (**set construction**, строки (1)–(5)) и описание способов реализации алгоритма

(**implementation method**, строки (6)–(19)). Способ реализации алгоритма задается переменной **imp_method** (строка (7)), которая может принимать значения ‘**oneC_oneV**’ (реализация с проверкой условия на каждой итерации, рис. 1, а) или ‘**manyC_oneV**’ (реализация с проверкой условия после нескольких вычислительных итераций, рис. 1, б). Класс итераций *K* в зависимости от выбранного способа реализации алгоритма может быть множеством или полумножеством (строка (8)), а соответствующее ветвление программы описано с помощью условного оператора **case** (строки (9) и (14)).

```
(1)  set construction ::
(2)      I=imp( BL(i) | BL(i)={set,imp}( (i-1)*s+1 … i*s ) and i in set(1…N) );
(3)      J=imp( BC(j) | BC(j)={set,imp}( (j-1)*c+1 … j*c ) and j in set(1…M) );
(4)      K*={set,imp}( IT(k) | IT(k)={set,imp}( (k-1)*ni+1 … k*ni ) and ( k=1 or err((k-1)*ni)>delta ) );
(5)  end (set construction);
(6)  implementation method ::
(7)      imp_method=<переменная, задающая способ реализации>;
(8)      typing(K)=’set’ or ‘sm’;
(9)      case (imp_method=’oneC_oneV’) :        // проверка на каждой итерации;
(10)         type(K)=’set’;
(11)         K=( k | k in K* and ( k=1 or err(k-1)>delta ) );
(12)         F=( conc(C,V) (k) | k in K );
(13)     end (case);
(14)     case (imp_method=’manyC_oneV’) :       // проверка каждые ni итераций;
(15)         type(K)=’sm’;
(16)         K=sub(K*);
(17)         F=( P(k) | k in K and ( mod(k,ni)=0 –> P=conc(C,V) ) and ( mod(k,ni)!=0 –> P=C ) );
(18)     end (case);
(19) end (implementation method);
```

Рис. 4. Фрагмент кода аспекта метода обработки

В строках (2) и (3) раздела **set construction** (см. код на рис. 4) задаются множества строк (*I*) и столбцов (*J*) матрицы, используемые при описании признаков операций пересчета (*C*) и проверки (*V*) (рис. 2). Множество строк *I* разбивается на *N* подмножеств *BL(i)*, в каждое из которых входит *s* строк, а множество столбцов *J* – на *M* подмножеств *BC(j)*, в каждое из которых входит *c* строк:

$$I = \left\{\left[\underbrace{1 \ldots s}_{BL(1)}, \underbrace{s+1 \ldots 2 \cdot s}_{BL(2)}, \ldots, \underbrace{\left[\left(N-1\right) \cdot s+1 \ldots N \cdot s\right]}_{BL(N)} \right]\right\}; \quad (14)$$

$$J = \left\{\left[\underbrace{1 \ldots c}_{BC(1)}, \underbrace{c+1 \ldots 2 \cdot c}_{BC(2)}, \ldots, \underbrace{\left[\left(M-1\right) \cdot c+1 \ldots M \cdot c\right]}_{BC(M)} \right]\right\}. \quad (15)$$

Различные варианты распараллеливания по строкам, столбцам или клеткам описываются путем задания соответствующих значений параметров *s*, *c*, *N* и *M* в архитектурном аспекте программы [1]. Четкость выделения совокупности и параллелизм ее элементов являются независимыми критериями типизации, поэтому соответствующие признаки (например, **set** и **imp**) образуют параллельно-независимые множества, которые выделяются в коде фигурными скобками (**{set,imp}**).

В строке (4) раздела **set construction** (см. код на рис. 4) формируется множество K^*, разбиение которого соответствует выражению (5) и описывает распараллеливание по итерациям, характерное для ВС с реконфигурируемой архитектурой. В множестве K^* каждому *k*-му блоку итераций соответствует подмножество *IT(k)*, которое включает *ni* итераций. Общее количество подмножеств определяется через условие завершения вычислений. Множество K^* необходимо для выделения полумножества итераций *K*, возникающего при реализации алгоритма с проверкой условия после нескольких вычислительных итераций (см. выражения (6)–(11) и рис. 1,б). Отметим, что множество итераций *K*, описывающее другой способ реализации алгоритма (см. выражения (1)–(4) и рис. 1, а), также является подмножеством совокупности K^*.

Способ реализации алгоритма с проверкой условия завершения вычислений на каждой итерации (**oneC_oneV**, рис. 1, а) описан в строках (9)–(13) кода на рис. 4. В строке (10) уточняется типизация класса *K*: в данном случае он является четко выделенным множеством (**set**). В строке (11) структура множества *K* формируется с помощью объявленного ранее множества K^*. Для этого из совокупности выбираются только те итерации, на которых условие завершения вычислений еще не выполнено или выполняется впервые. В строке (12) каждому элементу множества *K* присваиваются признаки пересчета *C* и проверки условия *V*, образующие множество параллельно-зависимого типа (**conc**). Совокупность *F*, описывающая информационный граф алгоритма, задается через множество итераций *K*, поэтому наследует его типизацию (**{set,imp}**) и разбиение на блоки по итерациям.

В строках (14)–(18) (см. код на рис. 4) описан способ реализации алгоритма метода Якоби с проверкой условия после нескольких вычислительных итераций (**manyC_oneV**, рис. 1, б). В этом случае класс *K* является полумножеством (строка (15)) и задается как подкласс четко выделенного множества K^* (строка (16)), причем *K* имеет сходную с K^* структуру, подразумевающую распараллеливание по итерациям. В строке (17) формируется описывающая информационный граф совокупность *F*, которая наследует типы (**{sm,imp}**) и разбиение полумножества *K*. В левой части реляционной конструкции в строке (17) используется переменный признак *P*, который в зависимости от остатка (**mod**) от деления номера итерации *k* на *ni* принимает значения **conc(C,V)** (пересчет и последующая проверка условия) или **C** (только пересчет).

На рис. 5 приведен фрагмент кода архитектурного аспекта для программы на языке Set@l, описывающей алгоритм решения СЛАУ методом Якоби.

Если алгоритм реализуется на ВС с реконфигурируемой архитектурой (**RCS**, левый столбец кода на рис. 5), то используется метод обработки по итерациям: совокупность *K* разбивается на подклассы, в каждый из которых входит *ni* итераций. Параметры, описывающие доступный вычислительный ресурс (*R*) и аппаратные затраты на реализацию одной итерации пересчета (*Rc*) и проверки условия завершения вычислений (*Rv*), определены в аспекте конфигурации ВС.

В случае ВС с мультипроцессорной архитектурой (**MP**, правый столбец кода на рис. 5) используется обработка по клеткам, размерность которых определяется числом процессоров $q_1 \times q_2$. Отметим, что в рассматриваемом коде предусмотрена возможность реализации алгоритма с проверкой условия после нескольких идущих подряд итераций. В таком случае значение параметра *ni* больше единицы и осуществляется дополнительное распараллеливание по итерациям.

```
(1)  case (architecture_type='RCS') :                case (architecture_type='MP') :
                          // Строки (I):
(2)      s=1;                                        s=q1;
(3)      N=n;                                        N=n/s;
(4)      type(I)='pipe';                             type(I)='seq';
(5)      type(sub(I))='pipe';                        type(sub(I))='par';
                          // Столбцы (J):
(6)      c=1;                                        c=q2;
(7)      M=m;                                        M=m/c;
(8)      type(J)='pipe';                             type(J)='seq';
(9)      type(sub(J))='pipe';                        type(sub(J))='par';
                          // Итерации (K):
(10)     case (imp_method='oneC_oneV') :             case (imp_method='oneC_oneV') :
(11)         ni=floor(R/(Rc+Rv));                        ni=1;
(12)     end (case);                                 end (case);
(13)     case (imp_method='manyC_oneV') :            case (imp_method='manyC_oneV') :
(14)         ni=floor((R–Rv)/Rc);                        ni=<число итераций>;
(15)     end (case);                                 end (case);
(16)     type(K)='pipe';                             type(K)='seq';
(17)     type(sub(K))='conc';                        type(sub(K))='seq';
(18) end (case);                                     end (case);
```

Рис. 5. Фрагмент кода архитектурного аспекта

Заключение. В отличие от существующих архитектурно-специализированных средств параллельного программирования ВС, язык Set@l позволяет описывать алгоритм решения задачи, способы его модернизации и распараллеливания в виде отдельных программных модулей – исходного кода и аспектов, которые образуют единую архитектурно-независимую программу.

Одной из отличительных особенностей языка программирования Set@l является возможность описания нечетких совокупностей в соответствии с альтернативной теорией множеств П. Вопенка. В дополнение к типизации по параллелизму, в языке Set@l вводится классификация совокупностей по четкости выделения их элементов. Если тип и разбиение совокупности на данном уровне абстракции определены нечетко, то она обозначается как класс и используется в коде аналогично множеству. Структура и типизация класса могут быть уточнены и переопределены в других аспектах программы. Помимо классов, существуют особые совокупности – полумножества, для которых нечеткость является основополагающей характеристикой и не может быть устранена аспектами программы. Оперируя классами, множествами и полумножествами, возможно описать различные способы реализации алгоритма в единой аспектно-ориентированной программе на языке Set@l. Представленный подход к программированию ВС открывает принципиально новые возможности для создания архитектурно- и ресурсно-независимого программного обеспечения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К.* Подход к архитектурно-независимому программированию вычислительных систем на основе аспектно-ориентированного языка Set@l // Известия ЮФУ. Технические науки. – 2018. – № 3 (197). – С. 46-58.
2. *Левин И.И., Дордопуло А.И., Мельников А.К., Писаренко И.В.* Аспектно-ориентированный подход к архитектурно-независимому программированию вычислительных систем // Суперкомпьютерные технологии (СКТ-2018): Материалы 5-й Всероссийской научно-технической конференции. – Ростов-на-Дону, Таганрог: Изд-во ЮФУ, 2018. – Т. 1. – С. 181-183.

3. *Каляев А.В., Левин И.И.* Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. – М.: ООО «Изд-во Янус-К», 2003. – 380 с.
4. *Гузик В.Ф., Каляев И.А., Левин И.И.* Реконфигурируемые вычислительные системы. – Таганрог: Изд-во ЮФУ, 2016. – 472 с.
5. *Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А., Гуленок А.А.* Программирование вычислительных систем гибридного типа на языке программирования COLAMO // Известия ЮФУ. Технические науки. – 2016. – № 11 (184). – С. 39-54.
6. *Stroetmann K., Herrmann T.* SetlX – A Tutorial // Research Gate. – URL: https://www.researchgate.net/publication/236174821_SetlX_-_A_Tutorial (дата обращения: 31.10.2018).
7. *Cantone D., Omodeo E., Policriti A.* Set Theory for Computing: From Decision Procedures to Declarative Programming with Sets. – New-York: Springer-Verlag, 2001. – 409 p.
8. *Dewar R.* SETL and the evolution of programming // From Linear Operators to Computational Biology: Essays in Memory of Jacob T. Schwartz. London: Springer-Verlag, 2013. – P. 39.
9. *Dessi M.* Spring 2.5 Aspect-Oriented Programming. – Birmingham: Packt Publishing Ltd., 2009. – 331 p.
10. *Kurdi H.A.* Review on aspect oriented programming // International Journal of Advanced Computer Science and Applications. – 2013. – Vol. 4. No. 9. – P. 22-27.
11. *Подорожкин Д.Ю., Когай А.Р., Сафонов В.О.* Применение методов аспектно-ориентированного программирования при разработке программных систем // Научно-технические ведомости СПбГПУ: Информатика. Телекоммуникации. Управление. – 2011. – Т. 126. № 3. – С. 166-171.
12. *Rebelo H., Leavens G.T.* Aspect-Oriented Programming Reloaded // SBLP 2017 Proceedings of the 21st Brazilian Symposium on Programming Languages, 2017. – Art. No.10.
13. *Хаусдорф Ф.* Теория множеств. Изд. 6-е, пер. с нем. – М.: ЛЕНАНД, 2018. – 304 с.
14. *Вопенка П.* Альтернативная теория множеств: Новый взгляд на бесконечность: пер. со словац. – Новосибирск: Изд-во Института Математики, 2004. – 612 с.
15. *Holmes M.R., Forster T., Libert T.* Alternative Set Theories // Handbook of the History of Logic: Sets and Extensions in the Twentieth Century. Vol. 6 / Gabbay D.M., Kanamori A., Woods J. (eds.). – Elsevier, 2012. – P. 559-632.
16. *Габрусенко К.А.* Философские основания теорий множеств Георга Кантора и Петра Вопенка // Вестник Томского государственного университета. – 2010. – № 339. – С. 32-35.
17. *Vopenka P.* The Philosophical Foundations of Alternative Set Theory // International Journal of General Systems. – 1991. – Vol. 20, No. 1. – P. 115-126.
18. *Бахвалов Н.С., Жидков Н.П., Кобельков Г.М.* Численные методы. – 7-е изд. – М.: БИНОМ. Лаборатория знаний, 2017. – 636 с.
19. *Ebrahimi A., Zandsalimy M.* Evaluation of FPGA Hardware as a New Approach for Accelerating the Numerical Solution of CFD Problems // IEEE Access. – 2017. – Vol. 5. – P. 9717-9727.
20. *Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И.* Реконфигурируемые мультиконвейерные вычислительные структуры. –2-е изд. – Ростов-на-Дону: Изд-во ЮНЦ РАН, 2009. – 344 с.

REFERENCES

1. *Levin I.I., Dordopulo A.I., Pisarenko I.V., Mel'nikov A.K.* Podkhod k arkhitekturno-nezavisimomu programmirovaniyu vychislitel'nykh sistem na osnove aspektno-orientirovannogo yazyka Set@l [Approach to architecture-independent programming of computer systems in aspect-oriented Set@l language]. *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2018, No. 3, pp. 46-58.
2. *Levin I.I., Dordopulo A.I., Mel'nikov A.K., Pisarenko I.V.* Aspektno-orientirovannyy podkhod k arkhitekturno-nezavisimomu programmirovaniyu vychislitel'nykh sistem [Aspect-oriented approach to architecture-independent programming of computer systems]. *Superkomp'yuternye tekhnologii – 2018: Materialy 5-y Vserossiyskoy nauchno-tekhnicheskoy konferencii* [Proceedings of the 5 th All-Russia Conference on Supercomputer Technologies (SCT-2018)], 17–22 September, 2018, Divnomorskoe. Rostov-on-Don, Taganrog: Izd-vo YuFU, 2018, Vol. 1, pp. 181-183.
3. *Kalyaev A.V, Levin I.I.* Modul'no-narashchivaemye mnogoprocessornye sistemy so strukturno-procedurnoj organizaciej vychislenij [Modular-scalable multiprocessor system with structural-procedural organization of computing]. Moscow: OOO «Izdatel'stvo Janus-K», 2003, 380 p.

4. *Guzik V.F., Kalyaev I.A, Levin I.I. Rekonfiguriruemye vychislitel'nye sistemy* [Reconfigurable computer systems]. Taganrog: Izdatel'stvo YuFU, 2016, 472 p.
5. *Dordopulo A.I., Levin I.I., Kalyaev I.A., Gudkov V.A., Gulenok A.A.* Programmirovanie vychislitel'nykh sistem gibridnogo tipa na yazyke programmirovaniya COLAMO [Programming of hybrid computer systems in the programming language COLAMO]. *Izvestiya YUFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2016, no. 11, pp. 39-54.
6. *Stroetmann K., Herrmann T.* SetlX – A Tutorial. Research Gate. Available at: https://www.researchgate.net/publication/236174821_SetlX_-_A_Tutorial (accessed 31 October 2018).
7. *Cantone D., Omodeo E., Policriti A. Set* Theory for Computing: From Decision Procedures to Declarative Programming with Sets. New-York: Springer-Verlag, 2001, 409 p.
8. *Dewar R.* SETL and the evolution of programming. From Linear Operators to Computational Biology: Essays in Memory of Jacob T. Schwartz. London: Springer-Verlag, 2013, 208 p.
9. *Dessi M.* Spring 2.5 Aspect-Oriented Programming. Birmingham: Packt Publishing Ltd., 2009, 331 p.
10. *Kurdi H.A.* Review on aspect oriented programming. International, *Journal of Advanced Computer Science and Applications*, 2013, Vol. 4, No. 9, pp. 22-27.
11. *Podorozhkin D.Yu., Kogaj A.R., Safonov V.O.* Primenenie metodov aspektno-orientirovannogo programmirovaniya pri razrabotke programmnyh sistem [Application of aspect-oriented programming methods for development of software systems]. *Nauchno-tekhnicheskie vedomosti SPbGPU: Informatika. Telekommunikacii. Upravlenie* [St. Petersburg Polytechnical University Journal. Computer Science. Telecommunication and Control Systems], 2011, Vol. 126, No. 3, pp. 166-171.
12. *Rebelo H., Leavens G.T.* Aspect-Oriented Programming Reloaded. SBLP 2017 Proceedings of the 21st Brazilian Symposium on Programming Languages, 2017, art. no.10.
13. *Hausdorff F.* Teoriya mnozhestv [Set theory], 6-th ed. Moscow: LENAND, 2018, 304 p.
14. *Vopenka P.* Al'ternativnaja teorija mnozhestv: Novyj vzgljad na beskonechnost' [Alternative set theory: a New look at infinity]. Novosibirsk: Izdatel'stvo Instituta matematiki, 2004, 612 p.
15. *Holmes M.R., Forster T., Libert T.* Alternative Set Theories. Handbook of the History of Logic: Sets and Extensions in the Twentieth Century, vol. 6, Gabbay D.M., Kanamori A., Woods J. (eds.). Elsevier, 2012, pp. 559-632.
16. *Gabrusenko K.A. Filosofskie osnovaniya teoriy mnozhestv Georga Kantora i Petra Vopenka* [Philosophical foundations of Georg Cantor and Petr Vopenka set theories]. *Vestnik Tomskogo gosudarstvennogo universiteta* [Tomsk State University Journal], 2010, No. 339, pp. 32-35.
17. *Vopenka P.* The Philosophical Foundations of Alternative Set Theory, *International Journal of General Systems*, 1991, Vol. 20, No. 1, pp. 115-126.
18. *Bahvalov N.S., ZHidkov N.P., Kobel'kov G.M.* Chislennye metody [Numerical methods]. Moscow: BINOM. Laboratoriya znanij, 2017, 636 p.
19. *Ebrahimi A., Zandsalimy M.* Evaluation of FPGA Hardware as a New Approach for Accelerating the Numerical Solution of CFD Problems, *IEEE Access*, 2017, Vol. 5, pp. 9717-9727.
20. *Kalyaev I.A, Levin I.I., Semernikov E.A., Shmoylov V.I.* Rekonfiguriruemye mul'tikonveernye vychislitel'nye struktury [Reconfigurable multipipeline computing structures], 2nd ed. Rostov-on-Don: Izdatel'stvo YUNC RAN, 2009, 344 p.

Статью рекомендовал к опубликованию д.т.н. Э.В. Мельник.

Левин Илья Израилевич – Южный федеральный университет, e-mail: iilevin@sfedu.ru; 347928, г. Таганрог, ул. Петровская, д. 15, кв. 143; тел.: 88634612111; и.о. зав. кафедрой интеллектуальных и многопроцессорных систем; д.т.н.; профессор.

Дордопуло Алексей Игоревич – Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров»; e-mail: dordopulo@superevm.ru; 347902, г. Таганрог, 9-й переулок, д. 44; тел.: 88634477407; начальник отдела математико-алгоритмического обеспечения; к.т.н.

Писаренко Иван Вадимович – e-mail: pisarenko@superevm.ru; 347942, г. Таганрог, ул. Нижняя линия, 8; тел.: 88634477407; м.н.с.

Мельников Андрей Кимович – Закрытое акционерное общество «ИнформИнвестГрупп»; e-mail: ak@iigroup.ru; 115432, г. Москва, Проспект Андропова, 18, Бизнес-Центр Nagatino i-Land, корп. 7 “Ломоносов”, эт. 3; тел.: 84957871109; с.н.с.; к.т.н.

Levin Ilya Izrailevich – Southern Federal University; e-mail: iilevin@sfedu.ru; 15, Petrovskaya str., ap. 143, Taganrog, 347928, Russia; phone: +78634612111; head of the department of Intellectual and Multiprocessor Systems; dr. of eng. sc.; professor.

Dordopulo Alexey Igorevich – “Supercomputers and Neurocomputers Research Center” Co. Ltd; e-mail: dordopulo@superevm.ru; 44, 9th lane, Taganrog, 347902, Russia; phone:: +78634477407; head of the Division of Mathematic and Algorithmic Support; cand. of eng. sc.

Pisarenko Ivan Vadimovich – e-mail: pisarenko@superevm.ru; 8, Nizhnyaya Liniya str., Taganrog, 347942, Russia; phone: +78634477407; junior researcher.

Melnikov Andrey Kimovich – “InformInvestGroup” C. C.; e-mail: ak@iigroup.ru; 18, Andropov Av., Nagatino i-Land Business Centre, building 7 “Lomonosov”, floor 3, Moscow, 115432, Russia; phone: +74957871109; senior researcher; cand. of eng. sc.

УДК 004.422 DOI 10.23683/2311-3103-2018-5-48-56

Л.К. Бабенко, И.А. Писарев

ЭЛЕКТРОННОЕ ГОЛОСОВАНИЕ С ПРИМЕНЕНИЕМ МНОЖЕСТВЕННОГО БРОСАНИЯ БЮЛЛЕТЕНЕЙ*

Применение электронного голосования постепенно вытесняет традиционное. Однако вопрос создания честной и надежной системы электронного голосования все еще открыт. Создание надежной системы электронного голосования, удовлетворяющей всем требованиям безопасности, является сложной задачей. В работе представлена система электронного голосования на основе принципа слепых посредников с применением множественного бросания бюллетеней. Слепые посредники позволяют исключить связь голоса пользователя с какими-либо его аутентификационными данными. Принцип множественного бросания бюллетеней основан на использовании обманных бюллетеней и позволяет верифицировать корректность бюллетеня без применения доказательства с нулевым разглашением, а также обеспечивает выполнение требований к системам электронного голосования. Приведена архитектура системы, описаны компоненты, задействованные в процессе проведения голосовании, и их взаимодействие между собой. Описан процесс голосования, состоящий из нескольких этапов: подготовка, голосование, подсчет результатов. Приведены криптографические протоколы, которые используются при передаче данных между компонентами системы. Описаны принципы заполнения бюллетеней в зависимости от количества кандидатов и типа голосования. Описаны типы голосования один из многих и несколько из многих. Обосновано соблюдение требований безопасности системой. Описана процедура обманного голосования, которая применяется для защиты реального голоса избирателя в случае возможного принуждения к определенному выбору кандидата. Описаны основные атаки, которые возможно провести с помощью вредоносного программного обеспечения на клиентском приложении, обоснованы механизмы защиты от той или иной атаки, а также дальнейшие пути доработки системы для противодействия атакам.

Электронное голосование; криптографическая защита; криптографические протоколы; шифрование.

* Работа поддержана грантом РФФИ № 18-07-01347 А.