УДК 51:681.03

А.Е. Дорошенко, Г.Е. Цейтлин

АЛГЕБРОАВТОМАТНЫЕ СПЕЦИФИКАЦИИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НАД ОБЩЕЙ И РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Освещены вопросы теоретико-автоматных и алгебраических исследований, восходящих от фундаментальных работ В.М.Глушкова, приведен ретроспективный обзор основных результатов, полученных в данной области, а также установлены их внутренние взаимосвязи и перспективные направления развития. Рассмотрены алгебродинамические модели параллельного взаимодействия последовательных программ и алгеброалгоритмические спецификации, ассоциированные с этими моделями. Изложены вопросы дискретных преобразователей над внутренней памятью и конвейерные вычисления, а также теория клонов и инструментальные средства синтеза параллельных алгоритмов и программ.

Введение

Как известно, человеческие коллективы способны эффективно решать сложные задачи посредством разделения на более простые подзадачи с их распределением между исполнителями, входящими в состав коллектива, и организацией взаимодействия между исполнителями (обмена промежуточными результатами, временного согласования этапов разработки в соответствии с общим планом решения поставленной задачи и др.). По аналогии со сказанным современные компьютерные системы параллельного действия (многопроцессорные системы, распределенные вычислительные сети и среды) можно рассматривать как коллективы вычислителей, ориентированные на решение сложных задач при жестких ограничениях на допустимое время решения и другие ресурсы.

Параллельные вычисления в настоящее время являются одной из главных парадигм современного программирования и охватывают чрезвычайно широкий круг вопросов разработки программ. Ввиду значительно более сложной природы параллельных вычислений по сравнению с последовательными большое значение приобретают методы автоматизации разработки параллельного программного обеспечения, основанные на применении техники формальных моделей, спецификаций и преобразований параллельных программ. Фундаментальными проблемами организации параллельных вычислений являются две следующие [1]:

- проблема увеличения производительности и эффективности использования многопроцессорных и распределенных вычислительных систем;
- проблема повышения уровня интеллектуализации программирования параллельных систем.

Они не являются независимыми, ибо организация высокопроизводительных вычислений в многопроцессорной системе современной архитектуры оказывается слишком сложной для попыток ее решения без средств интеллектуализации программирования в такой системе. Трудность решения вопросов программирования параллельных систем определяется тем обстоятельством, что вопросы организации взаимодействий и синхронизации параллельных процессов существенно усложняют разработку параллельных алгоритмов и программ по сравнению с их традиционными, последовательными, вариантами. Имеющиеся средства автоматизации программирования многопроцессорных систем, в основном для языков семейства FORTRAN и C, не могут удовлетворить потребности все возрастающего количества пользователей, так как имеют ограниченные возможности как по предоставляемым средствам параллельного программирования, так и по кругу решаемых задач, характеризуемых в основном регулярными схемами вычислений.

Мощный поток исследований, особенно последнего десятилетия, по созданию языков параллельного программирования высокого уровня, декларативных средств описания и реализации параллельных вычислений, интеграции различных парадигм программирования — функциональной, процедурной, логической, объектно-ориентированной, алгебраической — для проектирования параллельных вычислений привел к созданию целого ряда экспериментальных разработок, технологий и интерфейсов, значительно расширяющих возможности пользователей и разработчиков. Следствием интенсивности исследований в этой области и важности для промышленности стало появление стандарта де-факто на интерфейс параллельного программирования под названием MPI (Message Passing Interface) для архитектуры с распределенной памятью, ориентированной на передачу сообщений. Этот факт характеризует важное направление современного развития параллельных вычислений в сторону их независимости от архитектуры мультипроцессорной платформы, на которую они первоначально намечались к реализации, от топологии связей процессоров и их количества, других особенностей конкретных параллельных систем.

Таким образом, основное назначение современной технологии проектирования программного обеспечения для параллельных систем — способствовать существенному повышению их производительности, надежности, сокращению сроков разработки, облегчению процесса сопровождения программ, их модифицируемости в соответствии с изменчивостью окружающей программной среды и пр. Решение этих и других важных задач методологии и технологии современного параллельного программирования сопряжено с построением моделей, ориентированных на описание (спецификацию) параллельных вычислений, а также на реализацию (синтез) параллельных программ над различными видами памяти. Такие модели обычно представляют собой многокомпонентные формализмы, которые могут охватывать несколько парадигм не только собственно вычислений, но и других аспектов функционирования современных компьютерных систем.

В данной статье освещены вопросы теоретико-автоматных и алгебраических исследований, восходящих от фундаментальных работ В.М. Глушкова по автоматизации проектирования логических структур ЭВМ и программирования [2, 3]. Целью работы является ретроспективный обзор основных результатов, полученных в данной области, а также установление их внутренних взаимосвязей и перспективных направлений развития. Изложение материала осуществляется по мере повышения уровня абстракции в соответствии с принципом "снизу вверх". В разделе 1 обсуждается ряд важных аспектов, определяющих сущность алгебродинамического подхода к формализации семантики параллельной программы. В разделе 2 приводятся основные понятия и результаты по построению аппарата систем алгоритмических алгебр (САА) Глушкова и их модификаций, ориентированных на проектирование и синтез параллельных регулярных схем (ПРС). Особое внимание уделено построению алгебродинамических ПРС, базирующихся на механизмах координации параллельных ветвей с использованием аппарата синхронизаторов и событий, а также РС последовательных компонент. В разделе 3 и 4 рассматриваются вопросы использования автоматных структур с внутренней памятью и концепции двухуровневости алгебры алгоритмики.

1. Алгебродинамические модели параллельных вычислений

Как уже было отмечено во введении, в настоящее время модели параллельных вычислений являются многокомпонентными формализмами, которые могут охватывать несколько парадигм описания не только собственно вычислений, но и других аспектов функционирования многопроцессор-

ных систем [1]. Примерами тому являются так называемые "практические" модели вычислений [4, 5], в которых присутствуют средства трех компонент: алгоритмической, определяющей распараллеливание собственно вычислительных операций; программной, определяющей особенности работы параллельных компонент с представленными в модели различными видами памяти многопроцессорной системы; и координационной, отражающей свойства средств синхронизации и обменов между процессорами, которые могут существенно влиять на принципы организации мультипроцессорной обработки и производительность параллельной системы в целом. В настоящем разделе описан алгебродинамический подход к описанию параллельных вычислений, разработанный в [6], который характеризуется следующими важными достоинствами:

- задание *динамической* (операционной) семантики вычислений, которая собственно и является наиболее общим интерфейсом для разработчиков и пользователей параллельных систем;
- использование в качестве концептуальной основы абстрактного понятия *дискретной динамической системы* [7], которая имеет естественную меру сложности своих процессов, а именно их длину;
- *локальность изменений* и взаимодействий, задаваемых в определении дискретных систем переходов, происходящих в параллельной системе, что облегчает верификацию моделей и направлено на получение эффективных реализаций в распределенных мультипроцессорных системах и компьютерных сетях;
- сохранение характера *исчисления*, присущего декларативным моделям (алгебрам) вычислений, что позволяет вести разработку параллельных систем в трансформационном стиле от начальных спецификаций до конечной реализации, соблюдая корректность и отслеживая эффективность промежуточных преобразований.

1.1. Основные понятия и определения алгебродинамической модели. В основу определения алгебрадинамических моделей параллельных программ положены алгебра алгоритмов Глушкова и теория дискретных динамических систем [1, 2, 7]. Пусть V — множество переменных, а D — область значений переменных. Для простоты будем полагать, что все переменные из V — простые, т. е. их значения являются скалярными. Множество частичных отображений $B = \{b: V \to D\}$, называемых состояниями памяти, образует информационную среду. Алгебра алгоритмов является двухосновной алгеброй $A(Y, U)$, где Y — алгебра операторов, состоящая из частичных преобразований $y : B \to B$ информационной среды; U — алгебра условий, состоящая из частичных предикатов $u : B \to \{\mathbf{0}, \mathbf{1}\}$, заданных на B. Операции алгебры операторов принимают значения в алгебре операторов. Таких операций три:

- последовательная композиция PQ операторов P и Q: $(PQ)(b) = (P(Q(b))$;
- ветвление $u \to (P\ else\ Q)$, где P, Q — операторы, u — условие; значение операции ветвления определяется условным выражением: $u \to (P\ else\ Q) =$ = если $u(b) = \mathbf{1}$ то $P(b)$ иначе если $u(b) = \mathbf{0}$ то $Q(b)$ иначе не определено;
- итерация $while(u, P)$, где u — условие, P — оператор; по определению $while(u, P)(b) =$ если $u(b) = \mathbf{0}$ то b иначе если $u(b) = \mathbf{1}$ то $while(u, P)(P(b))$ иначе не определено.

Пусть $n > 0$ — наименьшее целое число такое, что $u(P^n(b)) = \mathbf{1}$ и $u(P^k(b)) = \mathbf{0}$ для всех $k = 1, \ldots, n-1$. Если такое n существует, то операция итерации определена и ее значение может быть вычислено $while(u, P)(b) = P^n(b)$.

Алгебра операторов содержит константы: ε — тождественный оператор и $\varnothing$ — пустой оператор. В алгебре условий имеются константы: $\mathbf{0}$ — тождественно ложное и $\mathbf{1}$ — тождественно истинное условие; булевы операции, а также операция умножения Pu оператора на условие. По определению

$Pu(b) = u(P(b))$, и тем самым Pu есть условие "u после P". Таким образом, алгебра алгоритмов есть множество операторов и условий, замкнутое относительно операций алгебры алгоритмов. Если в алгебре операторов выделено множество базовых операторов, а в алгебре условий — множество базовых условий таким образом, что они порождают всю алгебру алгоритмов, то каждый элемент алгебры может быть представлен регулярным выражением из элементов выделенного базиса и операций алгебры алгоритмов. Регулярные выражения алгебры операторов называются *регулярными программами*.

Регулярную программу будем называть *программой с двухуровневой памятью*, если множество ее переменных состоит из двух непересекающихся подмножеств: $V = I \cup E$, где I — множество внутренних, E — множество внешних переменных программы, а среди базовых операторов определены операторы чтения из внешней памяти $x: \leftarrow e$ и записи во внешнюю память $x: \rightarrow e$, $x \in I$, $e \in E$. Операторы чтения и записи называются операторами внешнего обмена и имеют смысл, аналогичный оператору присваивания: при выполнении оператора записи значение внутренней переменной становится значением внешней, а при выполнении оператора чтения — наоборот: внутренняя переменная становится обладателем значения внешней переменной.

Если $P = \{P_i\}$ — множество регулярных программ и I_i — множество внутренних переменных программы P_i, то будем говорить, что память $I = \cup I_i$ является *распределенной*, если множества I_i попарно не пересекаются. Пусть теперь $P = \{P_i\}$ — конечное множество регулярных программ с распределенной памятью, K — множество имен, конечное или бесконечное, и $t: K \rightarrow P$ — сюръективное отображение именования. Программы из P называются *сообщающимися в K*, если в их базисе операторов имеются операторы прямого обмена данными: оператор передачи вида $x \rightarrow k$ и оператор приема вида $x \leftarrow k$, где x — внутренняя переменная, k — имя, отличное от имени программы, в которой выполняется данный оператор прямого обмена. Операторы прямого обмена являются парными. При выполнении в программе $t(k_i)$ передачи вида $x \rightarrow kj$ в программе $t(k_j)$ одновременно должен выполняться прием вида $y \leftarrow k$. Обмен данными происходит по правилам присваивания $y := x$. Будем считать, что базис операторов алгебры алгоритмов содержит кроме обычных конструкций последовательных программ указанные выше операторы прямого и внешнего обмена, а также оператор независимого (параллельного) вызова программ вида $(y_1, \ldots, y_m) \leftarrow F(x_1, \ldots, x_n)$, где F есть имя независимо вызываемой программы, а $x_1, \ldots, x_n$ и $y_1, \ldots, y_m$ — соответственно ее фактические входные параметры и результаты. Операторы, не являющиеся операторами обмена и независимого вызова программ, будем называть *элементарными*.

Определение. Параллельной программой называется набор $p = (P, K, t, E)$, где $P = \{P_i\}$ — множество сообщающихся в K регулярных программ с двухуровневой памятью, внутренняя память которых распределена, а внешняя память E является общей. Имена из K называются *именами элементов*, программы из P — *элементными программами* или просто *элементами*, а частичное отображение $t: K \rightarrow P$ — *конфигурацией* программы.

Определенная таким образом параллельная программа представляет собой весьма общую модель параллельных вычислений для системы взаимодействующих регулярных последовательных программ над двухуровневой памятью. Такое определение охватывает все три известных способа взаимодействия в параллельных программах: через общую память, передачей сообщений и с помощью независимого вызова программ. Такое определение параллельной программы

также обобщает, в смысле допущения динамической конфигурации параллельной программы, известные общие формализмы параллелизма, ставшие уже классическими, такие, как взаимодействующие процессы Хоора [8], распределенные процессы Хансена [9] и другие. Конфигурация программы в общем случае зависит от времени и характеризует текущую актуализацию элементных программ, т.е. состояние их активности. Элементы начальной конфигурации, которая может быть не всюду неопределенной, начинают выполняться непосредственно после запуска параллельной программы. В дальнейшем конфигурация может изменяться под воздействием самих элементных программ: при выполнении независимых вызовов новые элементы добавляются в конфигурацию, а по завершении вызванной программы — из конфигурации удаляются. Выполнение параллельной программы начинается запуском элементов из начальной конфигурации и заканчивается по завершении всех вызванных элементных программ. Предполагается, что входные и выходные данные программы находятся во внешней памяти.

Для описания семантики параллельных программ будем использовать общие понятия дискретных динамических систем (называемых по-другому и транзиционными системами [10]). *Дискретная динамическая система* (ДДС) есть тройка (S_0, S, d), где S — множество, называемое пространством состояний; $S_0 \subseteq S$ — множество начальных состояний; d — бинарное отношение переходов в пространстве состояний. Если обозначить $P(S_0)$ множество всех конечных процессов, т.е. последовательностей состояний $p = s_0, s_1, \ldots, s_n, \ldots$, начинающихся в S_0, $s_0 \in S_0$, $n \geq 1$, то d можно представить как отношение $d \subseteq P(S_0) \times S$ такое, что $d(s, S) = \varnothing$ для всех $s \notin S$. Отношение переходов и множество начальных состояний однозначно определяют множество допустимых процессов F системы как объединение $F = \bigcup_{t=0}^{\infty} F_t$, где F_t есть множество допустимых процессов длительностью t. Здесь F_0 — множество начальных состояний, а для $t > 0$ имеет место $ps \in F_t \Leftrightarrow p \in F_{t-1}$ и $ps \in d(p)$, где d рассматривается как многозначное отображение d: $P(S) \to S$. Поэтому систему (S_0, S, d) можно задавать и альтернативным способом в виде пары (S, F). Для обозначения ДДС будем использовать, если это не приводит к двусмысленности, тот же символ, что и для множества ее состояний.

Здесь рассматриваются детерминированные автоматные ДДС. Система (S_0, S, d) называется автоматной, если $(ps, s') \in d \Leftrightarrow (s, s') \in d$. В таком случае пару $(p, s) \in d$, где $p \in P(S_0)$, $s \in S$, естественно записывать $p \Rightarrow s$, если S фиксировано. Система называется детерминированной, если отношение переходов d можно отождествить с отображением $d : P(S_0) \to S$. В автоматных детерминированных системах переходы не зависят от предысторий и поэтому их можно изображать в виде следования $s \Rightarrow s'$. В пространстве состояний может быть выделено подмножество $S_* \subseteq S$, называемое множеством заключительных состояний, такое, что процессы, проходящие через S_*, обязательно завершаются. Такие процессы будем называть *финальными*. Процесс может завершаться также ввиду неопределенности отношения переходов в состояниях, не являющихся заключительными. Такие процессы называются *тупиковыми*, а состояния — *тупиками*. ДДС называется многокомпонентной, если множество ее состояний S содержится в декартовом произведении $S \subseteq S_1 \times \ldots \times S_m$. Множества $S_1 \ldots S_m$, которые составляют наименьшее декартово произведение, содержащее S, называются компонентами системы. Если компоненты системы сами являются многокомпонентными ДДС, то система называется многоуровневой. Многокомпонентные многоуровневые дискретные динамические системы в дальнейшем исполь-

зуются в качестве моделей параллельных программ для определения их операционной семантики.

В соответствии с введенным выше определением параллельная программа определяет объект последовательно-параллельных вычислений, в котором последовательное управление вычислениями внутри элементов сочетается с параллелизмом вычислений на межэлементном уровне, задаваемом конфигурацией параллельной программы. Построение агебродинамической модели проводится в два этапа. Сначала строится базовая дискретная динамическая система С0, в терминах которой определяется концептуально более прозрачная, но зато более ограничительная синхронная операционная семантика взаимодействий параллельных программ. Затем путем преобразований базовой модели получается основная модель, в которой взаимодействия параллельных элементов реализуются асинхронным способом.

1.2. Базовая алгебродинамическая модель параллельных программ. Ключевым в определении общей алгебродинамической модели параллелизма вычислений является случай двухуровневой параллельной программы (который будем называть базовым) — нижний уровень элементов и верхний уровень управления элементами. Базовый уровень является основным для изучения, поскольку к нему может быть сведен общий случай многоуровневых программ [7]. Динамический характер вычислений базовой модели определяется только возможностями независимого вызова программ. Общая память при этом остается одноуровневой, доступной для использования всеми компонентами параллельной программы статически определенными и динамически порожденными. Базовая дискретная динамическая система, в терминах которой задается операционная семантика параллельных программ, детально описана в [6] и выглядит следующим образом.

Пусть $p = (P, K, t, E)$ — параллельная программа в смысле данного выше определения. Будем считать, что внутренние переменные I_i элементов $P_i \in P$ и внешние переменные E программы являются скалярными и принимают значения в некотором универсальном бестиповом множестве данных D. Состояния системы определим как векторы $((k_1, b_1, R_1), \ldots, (k_m, b_m, R_m), M)$, где $b_i : I_i \to D$ — состояния памяти элементных программ P_i; R_i — состояния управления тех же программ (остаточные программы при их выполнении методом интерпретации); $\{k_1, \ldots, k_m\} \subseteq K$ — множество имен элементов текущей конфигурации, задающей соответствие $t(k_i) = P_i$; $M : E \to D$ — состояние внешней памяти. Ко множеству начальных состояний системы отнесем все такие состояния, в которых $R_i = P_i$ и $b_i = \varnothing$, $i = 1, \ldots, m$, т.е. остаточные программы представляют собой исходные элементные программы, а значения всех внутренних переменных неопределены. В заключительных состояниях имеет место $R_i = \varepsilon$, $i = 1, \ldots, m$. Заметим, что внешнюю память также можно считать элементом, но с пустой программой.

Отношения переходов на множестве состояний базовой системы С0 определяются правилами переходов. Для примера в табл. 1 представлены правила переходов А1 — А4, описывающие выполнение операторов прямого обмена данными и основных управляющих конструкций (для удобства представления в правилах фиксируются только изменяемые или используемые составляющие векторов состояний).

В системе С0 правила переходов состояний представлены с точностью до неэлементарных операторов (обменов и независимого вызова). Непосредственно из определения правил переходов следует, что система является автоматной и недетерминированной [7]. Параллелизм выполнения параллельной программы моделируется недетерминированностью порядка, в котором совершаются переходы внутри различных элементов программы. Выполнение всех операторов элементов, вклю-

Таблица 1. **Примеры правил переходов состояний в базовой системе *C*0**

A1. Если y — элементарный оператор, то $(b_i, yR_i) \Rightarrow (y(b_i), R_i)$.	A2. $(b_i,(u \to (R\ else\ Q))T) \Rightarrow \begin{cases}(b_i, RT), \text{ если } u(b_i)=\mathbf{1};\\(b_i, QT), \text{ если } u(b_i)=\mathbf{0}.\end{cases}$
A3. $(b_i, while(u, R)Q) \Rightarrow$ $\Rightarrow \begin{cases}(b_i, Q), \text{ если } u(b_i)=\mathbf{0};\\(b_i, R; while(u,R)Q), \text{ если } u(b_i)=\mathbf{1}.\end{cases}$	A4. $((k_i, b_i, (x \to k_j)R), (k_j, b_j, (y \leftarrow k_i\ Q))) \Rightarrow$ $\Rightarrow ((k_i, b_i, R), (k_j, b'_j, Q))$, где $b'_j(r) = \begin{cases}(b_i(x), \text{ если } r = y;\\(b_j(r) \text{ иначе.}\end{cases}$

чая неэлементарные, является синхронным в том смысле, что в элементной программе, выполняющей данный оператор, не может происходить никаких других вычислений до тех пор, пока данный оператор не будет завершен. Например, для прямых обменов это означает так называемое рандеву (термин, предложенный авторами языка Ада). Именно соответствующие один другому операторы приема и передачи данных согласно правилу A4 могут выполняться только одновременно, а в элементе, где готовность к обмену была достигнута раньше, должно наступить ожидание обоюдной готовности к обмену. Достоинства такого определения семантики обменов заключаются в простоте модели и надежности реализации. Однако простота модели оборачивается потерей эффективности параллельных программ ввиду неконтролируемости ожиданий синхронизации и сложностью программирования для достижения этой эффективности в процессе проектирования.

1.3. Алгебродинамическая модель асинхронных вычислений. В [6] предложена общая схема расширения алгебродинамических моделей путем построения отображений реализации в модели параллельных программ с дополнительными возможностями динамического распараллеливания за счет асинхронных вычислений внутри элементных программ, предназначенных для сглаживания возможных неравномерностей вычислений обменивающихся компонент и повышения таким способом эффективности распараллеливания. Такие возможности динамического совмещения вычислений различных элементов появляются при ослаблении требования строго синхронных вычислений по отношению к памяти. Техника построения такого рода модели продемонстрирована ниже на примере построения модели C1 с локально-асинхронными обменами.

Суть введения асинхронности при выполнении неэлементарного оператора в элементной программе состоит в том, чтобы в случае невозможности немедленного выполнения оператор мог бы быть отложен до момента готовности его входных данных в распределенной памяти элемента, а в промежутке тем временем вычисления могли бы быть продолжены в элементе до тех пор, пока не нарушаются информационные связи отложенного оператора. Например, при откладывании оператора передачи $x \to k$ (приема $x \leftarrow k$) возможно продолжение вычислений вплоть до момента изменения (использования) значения переменной x. Таким образом, готовность к обмену можно проверять не в точке, а в некотором интервале, величина которого определяется объемом вычислений в информационно независимых от неэлементарного оператора фрагментах элементной программы. Чем больше величина такого интервала, тем выше асинхронность оператора обмена и тем меньше потери времени на ожидания синхронизации. Если при этом интервалы готовности обменивающихся элементов перекрываются во времени, то ожидания могут не появляться вовсе. Подходы к определению такого рода моделей асинхронной памяти могут

гут быть различными в зависимости от глубины анализа информационных связей операторов элементной программы.

Для точного формулирования новой модели $C1$ с асинхронной распределенной памятью в терминах дискретных динамических систем понятие состояния системы расширяется путем добавления к нему новых составляющих, отражающих состояния обменов и вызовов в параллельной программе.

Состояниями в $C1$ являются векторы $((k_1, b_1, R_1), \ldots, (k_m, b_m, R_m), H, M, \tau, \mu, \lambda)$, где b_i, R_i и M обозначают то же, что и в модели $C0$, H_i есть множество отложенных неэлементарных операторов в элементе k_i; $H = \bigcup_{i=1}^{m} H_i$, $H_i \cap H_j = \varnothing$, $i \neq j$, τ и μ — пара матриц порядка $m(m+1)$, характеризующая состояния обменов.

Элемент матрицы τ_{ij}, $1 \le i, j \le m$, представляет собой очередь переменных на передачу данных из элемента k_i в k_j, а μ_{ij}, $1 \le i, j \le m$, есть очередь переменных на прием значений в k_i из k_j. Очереди последнего столбца в матрицах τ и μ аналогично представляют состояния обменов с внешней памятью с той лишь разницей, что элементами этих очередей являются пары (x, e), где x — внутренняя, e — внешняя переменная элементной программы. Диагональные элементы этих матриц не используются, так как автопередача и автоприем данных в элементах не допускаются. Операцию постановки в очередь будем изображать конкатенацией * справа, а операцию взятия из очереди — сокращением очереди на один элемент слева. Отметим различие в назначении очередей в матрицах τ и μ, отражающих состояния прямых и внешних обменов параллельной программы. Если очереди τ_{ij} и μ_{ij}, $1 \le i, j \le m$, предназначены сглаживать возможную несогласованность моментов прямых обменов элементов, то очереди последнего столбца этих матриц должны служить буфером для предотвращения ожиданий из-за различия скоростей внутренней и внешней памяти программы.

При таком определении состояний системы $C1$ начальными в ней будут такие состояния, в которых $b_i = H_i = \tau_{ij} = \mu_{ij} = \varnothing$, $R_i = P_i$ для всех k_i, $k_j \in K$, а заключительными — такие, где $R_i = \varepsilon$, $H_i = \tau_{ij} = \mu_{ij} = \varnothing$. Для задания правил переходов в системе $C1$ введем несколько дополнительных определений. Для каждого элементарного оператора y зададим множества его входных $in(y)$ и выходных $out(y)$ переменных. Значения переменных из $in(y)$ используются, а из $out(y)$ — изменяются при вычислении преобразования состояния памяти $y : B \to B$. Для каждого элементарного условия $u \in U$ будем считать заданным множество $in(u)$ его входных переменных, значения которых используются при вычислении предиката $u : B \to \{\mathbf{1}, \mathbf{0}\}$. Например, для операторов обмена множества in и out описываются следующим образом: $in(x \to k) = \{x, k\}$, $out(x \to k) = \varnothing$, $in(x \leftarrow k) = \{k\}$, $out(x \leftarrow k) = \{x\}$, $in(x :\to e) = \{x, e\}$, $out(x :\to e) = \varnothing$, $in(x :\leftarrow e) = \{e\}$, $out(x :\leftarrow e) = \{x\}$. Понятие множеств in и out может быть естественным образом распространено на произвольные операторы элементных программ — выражения в алгебре операторов. Для неэлементарных операторов определяемые ниже множества In и Out совпадают соответственно с множествами in и out этих операторов, а для элементарных операторов будем считать их известными:

$$In(P; Q) = In(P) \cup (In(Q) \backslash Out(P)),$$
$$Out(P; Q) = Out(P) \cup Out(Q),$$
$$In(u \to (P \sim else \sim Q)) = In(u) \cup In(P) \cup In(Q),$$
$$Out(u \to (P\ else\ Q)) = Out(P) \cup Out(Q),$$
$$In(while(u, P)) = In(u) \cup In(P)),$$
$$Out(while(u, P)) = Out(P).$$

Операторы P и Q, как обычно, будем называть информационно независимыми, если $In(P) \cap Out(Q) = In(Q) \cap Out(P) = Out(Q) \cap Out(P) = \varnothing$. Аналогич-

Таблица 2. **Примеры правил переходов состояний системы *C*1**

B1. Если y элементарный оператор и $ind(y, H_i)$, то $(b_i, yR_i) \Rightarrow (y(b_i), R_i)$.	B2. Если u есть условие и $ind(u, H_i)$, то $(b_i, (u \to (R\ else\ Q))T) \Rightarrow \begin{cases}(b_i, RT), \text{ если } u(b_i) = \mathbf{1};\\ (b_i, QT), \text{ если } u(b_i) = \mathbf{0}.\end{cases}$
B3. Если u есть условие и $ind(u, H_i)$, то $(b_i, while(u, R)Q) \Rightarrow \Rightarrow \begin{cases}(b_i, Q), \text{ если } u(b_i) = \mathbf{0};\\ (b_i, R; while(u, R)Q), \text{ если } u(b_i) = \mathbf{1}.\end{cases}$	B4. Если $ind(x \to k_j, H_i)$, то $((b_i, (x \to k_j)R, H_i, \tau_{ij} \Rightarrow ((b_i, R, H_i \cup \{x \to k_j\}, \tau_{ij}{*}x)$.
B5. Если $ind(x \leftarrow k_j, H_i)$, то $((b_i, (x \leftarrow k_j)R, H_i, \mu_{ij} \Rightarrow \Rightarrow ((b_i, R, H_i \cup \{x \leftarrow k_j\}, \mu_{ij}{*}x)$.	B6. Если $\tau_{ij} = x{*}\tau'_{ij}$ и $\mu_{ij} = z{*}\mu'_{ij}$, то $((b_j, R_j), H_i, H_j, \tau_{ij}, \mu_{ji}) \Rightarrow \Rightarrow ((b'_j, R_j), H_i \setminus \{x \to k_j\}, H_j \setminus \{x \leftarrow k_i\}, \tau'_{ij}, \mu'_{ji})$, где $b'_j(r) = \begin{cases}(b_i(x), \text{ если } r = z;\\ (b_j(r) \text{ иначе.}\end{cases}$

но условие u считается информационно независимым от оператора P в случае $in(u) \cap Out(P) = \varnothing$. Если P — оператор, u — условие и S — множество операторов, информационно независимых от P и u, то будем обозначать такое отношение соответственно $ind(P, S)$ и $ind(u, S)$. С учетом введенных обозначений правила переходов в модели $C1$, соответствующие указанным в табл. 1 примерам правил системы $C0$, приведены в табл. 2.

Как видно из табл. 2, обмен данными, который был специфицирован в системе $C0$ единственным правилом A4, в новой системе теперь определяется тремя: B4, B5 и B6. Тем самым асинхронность работы с памятью в модели $C1$ достигается за счет разбиения единого, в случае синхронного режима базовой модели, акта выполнения взаимодействия на две фазы: инициирования и завершения. Инициирование состоит в превращении неэлементарного оператора в *отложенный* (оператор попадает во множество H) и постановке в очередь (в список) соответствующих элементов сигнатуры этого оператора. Состояния, в которые приходит система $C1$ после выполнения перехода по правилам B4, B5, называются *состояниями инициирования*. Завершение обмена или независимого вызова происходит при условии взаимной готовности взаимодействующих элементов и заключается в передаче данных или возвращению результатов, приводящих к изменению состояния памяти элементов и сокращению очередей переменных, списков состояния вызовов и множества H. В промежутке между инициированием и завершением, называемом *интервалом готовности* к взаимодействию, в элементе могут проводиться вычисления операторов, информационно независимых от оператора обмена или независимого вызова. Если время выполнения этих вычислений меньше интервала готовности к взаимодействию, то в элементе, пришедшем в состояние готовности раньше элемента-партнера, наступит ожидание. В противном случае элементы взаимодействуют без ожиданий. Увеличение интервалов готовности для повышения асинхронности памяти при фиксированной элементной программе возможно только путем искусственного разрыва информационных связей неэлементарных операторов взаимодействия. Другие возможные усиления модели вычислений в этом направлении разрабатывались в [6].

Асинхронность выполнения неэлементарных операторов по отношению к распределенной памяти в построенных моделях является важным источником повышения эффективности параллельных программ. Однако

правильность функционирования программ при этом нуждается в обосновании. Вопросы корректности рассмотренной модели асинхронных обменов сводятся к исследованию свойства глобальной детерминированности (конфлюэнтности) [7]. Хотя элементы параллельных программ являются детерминированными динамическими системами, построенные модели асинхронных взаимодействий порождают недетерминированные процессы вычислений. Источником недетерминизма является общая внешняя память элементов. Однако если ограничить обмены с внешней памятью только чтениями значений входных переменных (назовем такие параллельные программы простыми), то процесс вычислений определяется однозначно его начальным состоянием. Это утверждает следующая теорема, являющаяся обобщением теоремы о глобальной детермиированности асинхронных вычислений в [6] (идею доказательства и некоторые дополнительные понятия можно найти там же).

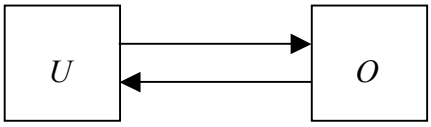
Теорема 1. Пусть C есть модель простых параллельных программ. Пусть также p — финальный (тупиковый) процесс в C с начальным состоянием s_0 и заключительным (тупиковым) состоянием s^*. Тогда любой другой процесс $p' = r(p)$ в системе C', полученный при неукорачивающем реализующем отображении r: $C' \to C$ и начинающийся в $r(s_0)$, также является финальным (тупиковым) и заканчивается в $r(s^*)$ или может быть продолжен до такового.

Результат этой теоремы аналогичен полученному в [10] в терминах транзиционных систем и дает возможность строить путем неукорачивающих (т.е. таких, при которых длины процессов-образов могут только увеличиваться) реализующих отображений все более "тонкие", в смысле совмещения вычислений и обменов, алгебродинамические модели параллельных программ, в которых, например, можно автоматически разрешать некоторые классы коммуникационных тупиков [11].

2. Регулярные схемы алгоритмов в алгебрах Глушкова

Рассмотрим ряд результатов, относящихся к описанию алгебродинамических моделей средствами структурных (регулярных) схем в алгебрах Глушкова.

Особенность функционирования достаточно сложной кибернетической системы, состоящей из управляющего U и операционного O объектов, заключается в наличии наряду с прямым и обратного канала связи между упомянутыми объектами, которые, как правило, представлены в автоматной форме (рисунок).



Рисунок

По прямому каналу (от U к O) выполняются операторы A — действия для целенаправленного изменения состояния автомата O (объекта, подлежащего управлению). В то же время управляющий автомат U по каналу обратной связи получает информацию о текущем состоянии автомата O (управляемого объекта). Эта информация поступает в виде значений логических условий u, с требуемой степенью полноты отражающих состояние объекта O. При этом в классической версии [13] автомат U конечен, тогда как, автомат O бесконечен в силу, обычно, высокой степени сложности объекта O. В соответствии с итеративным взаимодействием с O управляющий автомат U либо переходит в заключительное состояние, завершая процесс управления, либо продолжает взаимодействие с объектом O.

В литературе управляющий автомат был назван дискретным преобразователем (ДП) [12]. В рамках приведенной модели ДП получены следующие фундаментальные результаты:

- произвольно выбранная кибернетическая система представима в форме ДП;

• использование в ДП недетерминированных переходов и выходов приводит к моделированию недетерминированных процессов;

• модель ДП положена в основу алгебродинамических и ряда других известных моделей и обеспечивает возможность многоаспектного интегрированного описания исследуемых процессов [6, 13].

В табл. 3 приведены основные конструкции входящие в сигнатуру САА и их модификаций (САА-М), впоследствии получивших название алгебр Глушкова (АГ) [14]. Под регулярными (структурными) схемами (РС) понимаются представления последовательных алгоритмов в САА; в тоже время параллельные алгоритмы представимы в САА-М в форме параллельных РС (ПРС).

В качестве базиса зафиксируем совокупность I — операторных и U — логических переменных. Справедливы следующие утверждения.

Теорема 2 (о регуляризации — теорема Глушкова [САА,16]). Произвольный алгоритм (программа или микропрограмма) F представим в САА и САА-М эквивалентной интерпретированной РС $F = F_I$; разработана конструктивная процедура регуляризации (сведения к РС F_I) произвольного алгоритма.

Следствие 1. Поведение произвольно выбранного ДП, описанного в алгебродинамической форме, представимо посредством РС в САА.

Следствие 2. В рамках САА осуществимо прогнозирование процесса управления представленного ДП, а также обоснование проверки правильности его функционирования.

Справедливость данного следствия вытекает из наличия в сигнатуре САА операции прогнозирования, а также из двухосновности данной системы. Таким образом, АГ совпадают по своей порождающей мощности с известными алгоритмическими системами [17].

Теорема 3 [18, 19]. Пусть |А| — совокупность операторов, порожденных алгеброй А, тогда справедливы следующие строгие включения: $|АД| \subset \subset |АЯ| \subset |АГ|$, где АД — алгебра Дейкстры; АЯ — алгебра Янова.

Доказательства теорем 2 и 3 базируются на принадлежности операции прогнозирования сигнатуре АГ и производности в АЯ основных операций АД. Важно также отметить трансформационную сводимость неструктурных схем АЯ к эквивалентным РС [18].

К числу распространенных средств алгоритмических спецификаций относятся граф-схемы алгоритмов. Проблема построения соответствующей алгебры алгоритмов была поставлена в [19] и решена в [21, 20]. Пусть АГС — алгебра граф-схем, сигнатура операций которой состоит из графовых представлений операций, входящих в сигнатуру АГ (см. табл. 3).

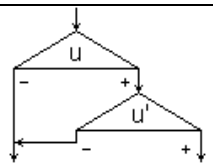
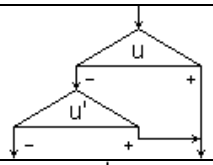
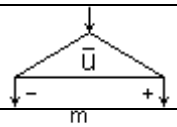
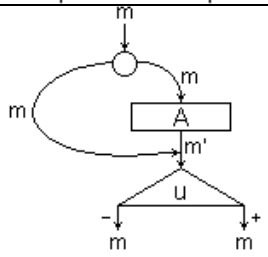
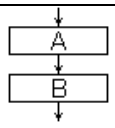
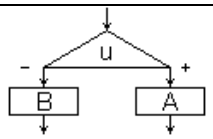
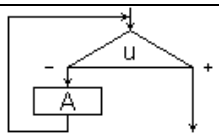
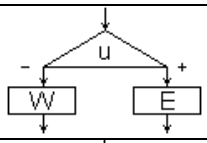
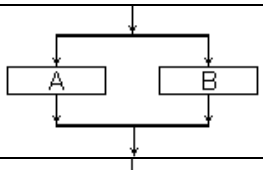
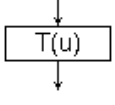
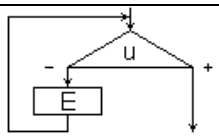
Теорема 4 [19—21]. АГС изоморфна АГ и любой алгоритм A представим эквивалентной интерпретированной граф-схемой G_I, $A = G_I$.

На построенную в [20] АГС могут быть распространены результаты структурной схематологии [22].

Взаимодействие автоматов U и O может носить распределенный характер. Такое разбиение среды представимо в форме распределенной сети ДП, обеспечивающих систематический контроль и реакцию на изменения состояний среды. В результате приходим к распределенным многопроцессорным моделям, а также ассоциированным с ними алгеброавтоматным параллельно взаимодействующим компонентам, рассмотренным в предыдущем разделе. В терминах аппарата САА-М могут быть формализованы приведенные там построения, так что справедливо следующее утверждение.

Теорема 5. Алгебродинамические модели $C0$ и $C1$ над общей и распределенной памятью представимы в АГ посредством ПРС.

Таблица 3. **Основные операции модифицированных систем алгоритмических алгебр (САА-М)**

Тип	Название операции	Форма		
		аналитическая	естественно-лингвистическая	графовая
Логические	Конъюнкция	u ∧ u' u . u'	u И u'	
	Дизъюнкция	u ∨ u' u + u'	u ИЛИ u'	
	Отрицание	$\bar{u}$	НЕ(u)	
	Прогнозирование (левое умножение условия на оператор)	A • u	ПОСЛЕ A условие u	
Операторные	Композиция	A * B	A ЗАТЕМ B	
	Альтернатива	([u] A, B)	ЕСЛИ u ТО A ИНАЧЕ B	
	Цикл	{[u] A}	ПОКА НЕ u ЦИКЛ A	
	Фильтр	$\underline{u}$	ФИЛЬТР(u)	
	Асинхронная дизъюнкция	A $\dot{\vee}$ B A // B	A ПАРАЛЛЕЛЬНО B	
	Контрольная точка	T(u)	КТ u	
	Синхронизатор	S(u)	ЖДАТЬ(u)	

Действительно, последовательные ветви введенных выше параллельных программ представимы в РС САА (см. теорему 2), а обмен данными между элементными программами возможен двух видов: прямыми обменами между областями локальной памяти двух последовательных компонент и внешними — через общую внешнюю память. На уровне ПРС подобные обмены реализуемы в терминах синхронизаторов и контрольных точек, т.е. событийных средств, входящих в сигнатуру САА-М.

3. Дискретные преобразователи с внутренней памятью и асинхронные конвейерные вычисления

К структурному параллельному программированию наряду с обычными программными конструкциями (последовательной, альтернативной, циклической и др.) следует отнести и основные структуры организации параллельных вычислений: конвейер, карусель, стапель и их разнообразные сочетания. Перечисленные основные параллельные структуры концентрируются по двойственным стратегиям мультиобработки: синхронной и асинхронной [13, 23].

Сущность синхронной обработки состоит в разделении времени на равные интервалы (квантование) и в одновременном выполнении за один временной интервал (квант) ряда действий (однотипных или разнотипных) над потоком данных, например одновременное умножение каждого элемента вектора или матрицы на некоторую константу. В случае асинхронной обработки несколько взаимодействующих параллельных ветвей функционируют на распределенных между ними потоках данных, осуществляя в процессе взаимодействия обмены промежуточными результатами и синхронизацию в определенные моменты времени.

Рассмотрим известные разновидности структур памяти: магазин, стек, очередь и их разнообразные сочетания в качестве разновидностей общего понятия эластичная лента [15, 16, 20], которое ориентировано на плотное хранение и обработку данных при проектировании различных процессов, в том числе и динамических, относящихся к асинхронному конвейеру, стапелю и др. [23]. В процессе функционирования конвейерных вычислений для снятия временных ограничений на квант синхронного выполнения команд, свойственного типовому конвейеру, введём в рассмотрение асинхронную модель конвейера K_a, в которой применяется структура памяти типа очередь для передачи данных между соседними процессами [23].

Пусть дана упорядоченная последовательность в общем случае разнотипных процессов $A_1, A_2, ..., A_k$, которые выполняются над потоком перерабатываемых данных $M = (m_1, m_2, ..., m_c)$ так что результат обработки q-го процесса поступает в q-ю очередь (память типа FIFO), работающую по принципу "последним пришел, последним ушел", из вершины которой элементы передаются на обработку $(q + 1)$-му процессу:

входной поток
$$\mathbf{m_c, m_{c-1}, ..., m_2, m_1 \Rightarrow A_1 | \Rightarrow q_1 \Rightarrow A_2 | \Rightarrow q_2 \Rightarrow A_2, ..., q_{k-1} \Rightarrow A_k | \Rightarrow r_1 \Rightarrow R,}$$

где **R** — поток результирующих данных.

Процессы $A_1, A_2, ..., A_k$ по мере поступления к ним на вход данных работают одновременно и асинхронно так, что A_i передает свои выходные данные на вход соседу справа процессу A_{i+1}, используя очередь БС$_i$. Посредст-

вом процесса A_k формируется поток R результирующих данных. Использование очередей обеспечивает независимую по времени параллельную работу процессов A_i, входящих в $К_а$. Проиллюстрируем САА-М спецификации последовательных и параллельных алгоритмов символьной обработки данных на примерах сортировки. С этой целью в терминах разметки [20] формализуются средства доступа — концепция абстрактного типа данных (АТД). Под разметкой понимается последовательность *m*: НУ2У1 a1 a2 ... an К, где Н, К — маркеры, фиксирующие соответственно начало и конец сортируемого массива; У1, У2 — указатели, перемещающиеся по массиву a1 a2 ... an вправо или влево на указанное количество символов ai (i = 1, 2, ..., n). На размеченном массиве *m* определяются базисные предикатные и операторные интерпретации РС, характеризующие доступ к массиву в процессе его сортировки. Предикатные интерпретации: $l > r$ по У2 — логическое условие, истинное, если указанное отношение выполняется для элементов *l* и *r*, расположенных непосредственно слева, справа от указателя У2; d(У1, К) — логическое условие, истинное, когда указатель У1 достиг маркера К. В остальных случаях приведенные предикатные интерпретации ложны. Операторные интерпретации: Р(У2, У1) — сдвиг указателей У2 и У1 на один символ вправо по массиву *m*; L(У2) — сдвиг У2 на один символ влево по массиву *m*; ТРАНСП(l, r|У2) — перестановка элементов *l* и *r*, соседних с указателем У2; УСТ(У2 на У1) — установка указателя У2 на У1.

Пример 1. Алгоритм ЧЕЛНОК, в терминологии Д.Кнута — "прямые вставки" [24], использует описанные выше средства доступа к сортируемым данным:

ЧЕЛНОК ::= {[d(У1, К)] Р(У2, У1) *
{[l > r по У2] ТРАНСП(l, r|У2) *
* L(У2)} * УСТ(У2 на У1) }.

Суть алгоритма состоит в том, что пара указателей перемещается вправо по массиву до обнаружения первой неупорядоченной пары, затем во вложенном цикле элемент *r* перемещается влево, пока он не будет установлен на место с последующим возвратом У2 к У1. Продолжая в цикле данный процесс, вплоть до перемещения указателей к маркеру К, завершаем процесс последовательной сортировки массива.

Пример 2. Конвейерная сортировка альтернативными вставками [23].

Реализуем на $К_а$ алгоритм $АЛЬТ_а$ — параллельная модификация сортировки альтернативными вставками. Пусть М0: НУ2У1a1 a2 ... anК — начальная разметка массива. С указателями У1 и У2 связаны два асинхронных процесса: 1) посредством левостороннего сканирования У1 осуществляется поиск элементов, стоящих "не на месте", которые засылаются в очередь БС; 2) одновременно посредством У2 осуществляется поиск в уже отсортированном фрагменте массива "места" для вставки элемента из вершины БС.

$АЛЬТ_а$ ::= [{[d(У1, К) . ('БС= ∅)] ['БС ≠
≠∅] ВСТАВКА, Е)} * ТК1 * С(ТК2) //
//{[d(У1,К)] ([*i* >*c*] ЗАП(*c*,БС) , Р(У1)) } *
* КТ2 * С(КТ1)] * ИСК(У1,У2),

где 'БС ≠ ∅ — предикат, истинный, если содержимое очереди не пусто; *i* и *c* — элементы непосредственно слева и справа от У1; ЗАП(*c*, БС) — оператор записи элемента *c* в основание очереди БС; ВСТАВКА — циклический оператор поиска (посредством перемещения У2 влево или вправо по уже отсортированному фрагменту массива) места вставки элемента из вершины БС с установкой его в найденную позицию.

Приведенная ПРС представляет асинхронное взаимодействие двух ветвей так, что по правой ветви находятся элементы массива, стоящие не на месте. Эти элементы загружаются в очередь. Параллельно по левой ветви очередь опустошается в результате рас-

становки находящихся в ней элементов по местам в уже отсортированной части массива. Мультиобработка продолжается вплоть до выполнения конъюнктивного циклического условия левого процесса; один из сомножителей этого условия истинен при достижении указателем У1 маркера К по правому процессу. Взаимодействие параллельных ветвей осуществляется посредством синхронизаторов и контрольных точек (событий), что обеспечивает синхронное завершение мультиобработки массива. Посредством ПРС могут быть представлены и другие алгоритмы символьной мультиобработки: конвейерное слияние отсортированных подмассивов, асинхронный распределенный поиск по системам файлов и пр. [20, 23].

4. Алгебра алгоритмики и инструментальные средства синтеза параллельных алгоритмов

Алгебра алгоритмики представляет собой двухуровневую систему. Верхний уровень соответствует неинтерпретированным схемам, а нижний — интерпретациям, ассоциированным с разнообразными приложениями [20].

В зависимости от поставленной задачи, выбранного метода разработки классов алгоритмов, технологической среды программирования осуществляется построение требуемой алгебры алгоритмов (АА) из семейства алгебро-алгоритмических спецификаций, подобных САА-М. Такие семейства описываются в терминах теории алгоритмических клонов [14]. Сигнатура САА из подобного семейства удовлетворяет теореме о функциональной полноте для данного клона, являясь тем самым, его системой образующих (СО). Построенная АА может быть исследована на предмет аксиоматизации, разработки систем оптимизирующих трансформаций, канонизации аналитических представлений и пр.

Следующие основные результаты по теории клонов, относящиеся к полугрупповым грамматическим и алгоритмическим клонам, соответствуют известным семействам АА [25]:

- исследованные клоны являются алгебрами континуального типа;
- каждый из них включает бесконечно порожденные подалгебры (с бесконечным базисом и без базиса);
- для клона Клини и его обобщений построены семейства максимальных подалгебр и решена проблема функциональной полноты;
- построена поверхность клона Дейкстры и его обобщений, имеющих q выходов из цикла ($q = 1, 2, ...$);
- описана совокупность максимальных подалгебр клона Янова, канонический представитель которого – алгебра Янова;
- установлено, что клон Янова – теоретико-множественный предел исследованных обобщений клона Дейкстры;
- описана совокупность максимальных подалгебр для клона Глушкова и его логической компоненты, система образующих которой включает операцию прогнозирования.

Важность распространения перечисленных результатов на алгоритмические клоны граф-схем и их обобщения определяется приложениями теории граф-схем при визуализации объектно-ориентированного программирования [26, 27]. Установлены критерии функциональной полноты и выразимости схем в построенных клонах. Получение таких критериев имеет важное практическое значение для реализации инструментальных (программных и аппаратных) средств проектирования и синтеза алгоритмов и программ, а также их классов, ассоциированных с актуальными предметными областями [20].

Следует заметить также, что приведенные результаты по итеративным алгоритмическим алгебрам и их семействам (клонам) ограничиваются двухосновным случаем, сопряженным с анализом операторных и логических конструкций. Вместе с тем имеется возможность их распространения на многосортные клоны, ассоциированные с описанием структур данных [28].

Разработанный формализм является универсальным в связи с возможностью его применения для проектирования самых различных алгоритмических моделей предметных областей, в частности при решении задач символьной мультиобработки, а также соответствующих языковых и инструментальных средств [26, 27].

Заключение

В данной статье была затронута только часть результатов, возникших под влиянием фундаментальных работ В.М. Глушкова по системам алгоритмических алгебр, которые касаются развития техники алгеброалгоритмических спецификаций для параллельных вычислений. Весомость вклада исследований академика В.М. Глушкова в развитие компьютерной науки трудно переоценить [29] и со временем она будет только возрастать. Об этом свидетельствуют тенденции развития исследований по теоретическим и прикладным аспектам алгоритмики последних лет [30], а также повышенный интерес к применению автоматно-алгебраических методов к автоматизации проектирования и разработки схемного и программного обеспечения компьютерных систем (см., например, [31–33]). Нет сомнения, что завет В.М. Глушкова поверить алгеброй гармонию алгоримики будет находить все новые и новые воплощения в будущих трудах отечественных и зарубежных ученых.

1. *Дорошенко А.Е.,* Структура и средства современных моделей параллельных вычислений // Пробл. программирования. — 1999. — N 1. — С. 38–52.
2. *Глушков В.М.* Теория автоматов и вопросы проектирования структур цифровых машин // Кибернетика. — 1965. — №1. — С. 3–11.
3. *Глушков В.М.* Теория автоматов и формальные преобразования микропрограмм // Там же. — №5. — С. 1–10.
4. *LogP:* Towards a realistic model of parallel computation / D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken // 4-th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP), SIGPLAN Notices. — 1993. — **28**, N 7. — P. 1–12.
5. *Valiant L.G.* A bridging model for parallel computation // Commun. ACM. — 1990. — **33**, N 8. — P. 103 –111.
6. *Дорошенко А.Е.* Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебродинамический подход. — Киев: Наук. думка, 2000. — 177 с.
7. *Капитонова Ю.В., Летичевский А.А.* Математическая теория проектирования вычислительных систем. — М.: Наука, 1988. — 296 с.
8. *Hoare C.A.R.* Communicating sequential processes // Commun. ACM. — 1978. — **21**, N 8. — P. 666–677.
9. *Hansen P.B.* Distributed processes: A concurrent programming concept // ibid. — **21**, N 11. — P. 934–941.
10. *Keller R.M.* A fundamental theorem of asynchronous parallel computation // Lect. Notes Comput. Sci. — 1975. — **24**. — P. 102–112.
11. *Doroshenko A.E.* On asynchronous avoidance of deadlocks in parallel programs // Parallel Processing Letters. — 1992. — **2**, N 2–3. — P. 291 –297.
12. *Глушков В.М., Летичевский А.А.* Теория дискретных преобразователей // Избранные вопросы алгебры и логики. — Новосибирск: Наука, 1973. — С. 5–39.
13. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Методы символьной мультиобработки. — Киев: Наук. думка, 1980. — 252 с.
14. *Цейтлин Г.Е.* Алгебраическая алгоритмика: теория и применения // Кибернетика и системный анализ. — 2003. — № 1. — С. 8–18.
15. *Гладкий А.В.* Формальные грамматики и языки. — М.: Наука, 1973. — 368 с.
16. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. — Киев: Наук. думка. — 1-е изд. — 1974, 327с.; 2-е изд., перераб. — 1978, 318 с.; 3-е изд., перераб. и доп. — 1989, 376 с.
17. *Колмогоров А.Н., Успенский В.А.* К определению алгоритма // УМН. — Т. 13, вып. 4 (82). — С. 3–28.
18. *Цейтлин Г.Е.* Формальные аспекты структурного программирования с GO TO // Программирование. — 1984, — № 1. — С. 3–11.
19. *Калужнин Л.А.* Об алгоритмизации математических задач // Пробл. кибернетики. — 1959. — № 2. — С. 51–69.
20. *Цейтлин Г.Е.* Введение в алгоритмику. — Киев: Сфера, 1998. — 310 с.
21. *Цейтлин Г.Е., Ющенко Е.Л.,* Алгебра алгоритмов и граф-схемы Калужнина // Кибернетика и системный анализ. — 1994, № 2. — С. 3–17.
22. *Цейтлин Г.Е.* Проблема тождественных преобразований схем структурированных программ с замкнутыми логическими условиями: ч. 1–3 // Кибернетика. — 1979. — № 3. — С. 50–57; №4. — С. 10–18; — № 5. — С. 44–51.

23. *Цейтлин Г.Е.* Языковые структуры и процессоры. — М.: Моск. энергет. ин-т, 1979. — 73 с.
24. *Кнут Д.* Искусство программирования для ЭВМ.— М.: Мир, 1978. — Т. 3. — 843 с.
25. *Цейтлин Г.Е.* Алгебры Глушкова и теория клонов // Кибернетика и системный анализ. — 2003 — № 4. — С. 48 – 58.
26. *Цейтлин Г.Е., Теленик С.Ф., Амонс А.А.* Алгебро-логическая формализация в объектно-ориентированных технологиях // Пробл. программирования. — 2002. — № 1 – 2. — С. 136 – 146.
27. *Цейтлин Г.Е., Яценко Е.А.* Элементы алгебраической алгоритмики и объектно ориентированный синтез параллельных программ // Математические машины и системы. — 2003. — №2. — С. 56 – 67.
28. *Редько В.Н., Гришко И.В., Редько И.В.* Дескриптологическая среда программирования // Пробл. программирования. — 2002. — № 1 – 2. — С. 7 – 14.
29. *Сергієнко І.В.* Інформатика в Україні: становлення, розвиток, проблеми. — Київ: Наук. думка, 1999. — 354 с.
30. *Ноден П., Китте К.* Алгебраическая алгоритмика. — М.: Мир, 1999. — 720 с.
31. *Alur R., Dill D.L.* A theory of timed automata // Theoretical Computer Sci. — 1994. — **235**. — P. 126 – 183.
32. *Generating* an Action Notation environment from montages descriptions / M. Anlauff, S. Chakraborty, P. Kutter, A. Pierantonio, L. Thiele // Software Tools and Technology Transfer. — 2001. — N 3. — P. 431 – 455.
33. *Gurevich Y.* Evolving Algebras 1993: Lipari Guide // Specifcation and Validation Methods / Ed. E. Boerger. — Oxford: Oxford University Press, 1995. — P. 9 – 36.

Получено 26.08.03

Об авторах

Дорошенко Анатолий Ефимович,
доктор физ.-мат.наук, профессор, заместитель директора по научной работе

Место работы автора
Институт программных систем НАН Украины,
просп. Академика Глушкова, 40,
Киев-187, 03680, Украина
Тел. 266 1538
E-mail: dor@isofts.kiev.ua

Цейтлин Георгий Евсеевич,
доктор тех. наук, профессор, заведующий кафедрой программного обеспечения автоматизированных систем

Место работы автора
Международный Соломонов университет,
Киев, Украина
Тел. 257 4374
E-mail: tseytlin@vikno.relc.com