ВОПРОСЫ ТЕОРИИ И ПОСТРОЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ
(Вычислительные системы)
1978 год Выпуск 73

УДК 681.322.0:681.3.06

# ЗАДАЧИ ПАРАЛЛЕЛЬНОГО МИКРОПРОГРАММИРОВАНИЯ

О.Л.Бандман, С.В.Шискунов, С.Н.Сергеев

## §1. Предмет параллельного микропрограммирования

1.1. Особенности параллельного микропрограммирования. Микропрограммирование является упорядоченным и удобным способом проектирования ЭВМ, позволяющим более детально проводить моделирование и выбор операторов. При этом предполагается, что ЭВМ имеет традиционную структуру, т.е. в ней явно выражены три части: запоминающая, управляющая и операционная. Микропрограммы хранятся в управляющей части и выдаются в виде последовательности управляющих воздействий в операционную часть, обеспечивая выполнение алгоритма, соответствующего поступившей из памяти команде.

С усложнением вычислительных машин, объединением их в системы и стремлением к максимальному совмещению операторов во времени возникла задача планирования вычислительного процесса в ходе его выполнения. Эта задача решается в системном программировании путем введения специальных переменных (семафоров, меток) и функций над ними [1], обеспечивающих асинхронное согласование работы операторов.

Аналогичная задача возникает на уровне микропрограммирования. Однако здесь основное влияние оказывают особенности организации микроструктуры, которые отражают тенденцию к распараллеливанию процесса вычисления на уровне элементарных операций. Эта тенденция обусловлена бурным развитием технологии БИС и связанным с ним изменением соотношения стоимости аппаратных и программных средств. Появление дешевых и быстродействующих микропроцессорных БИС, программируемых и перепрограммируемых ПЗУ и логических матриц привело к интенсивным разработкам микропроцессорных систем [2] и одно-

родных ассоциативных структур типа памяти с логикой [3,4], клеточных автоматов [5] и вычислительных сред [6], представляющих собой большие коллективы маломощных вычислителей. Для таких структур характерно следующее:

1) стирание граней между тремя основными частями традиционной структуры ЭВМ: памятью, управлением, процессором; часто возможно только условное (но не пространственное) их разделение, в зависимости от функции, которую они выполняют в данный момент времени;

2) структурная и функциональная перестройка в процессе вычисления для получения нужных операционных блоков и обеспечения надежности;

3) проникновение асинхронного управления на самый низкий уровень организации взаимодействий операторов.

Микропрограммный подход к проектированию однородных структур параллельной обработки с перечисленными свойствами определяет предмет микропрограммирования как методологию организации вычислительного процесса в коллективе вычислителей, обеспечивающую все виды согласования их взаимодействий. Такое расширенное понимание микропрограммного способа мы будем называть *параллельным микропрограммированием* и рассматривать его как обобщение традиционного понятия. При этом обычная (последовательная) микропрограмма может рассматриваться как частный случай параллельной.

1.2. *Основные понятия параллельного микропрограммирования.* Не выделяя в проектируемом устройстве управляющую и операционную части, мы будем рассматривать его как систему взаимосвязанных процессорных элементов и элементов памяти и характеризовать множеством элементарных операторов $Q$ и множеством состояний элементов памяти $S$. На основе этих характеристик определим основные понятия параллельного микропрограммирования следующим образом.

*Микрооперация* – выполнение одного элементарного оператора. Например: ⟨содержимое регистра $R_1$ переслать в регистр $R_2$⟩, ⟨слово $W_1$ заменить на слово $W_2$⟩, ⟨сложить ($mod_2$) содержимые $R_1$ и $R_2$, результат переслать в $R_3$⟩.

*Микрокоманда* – выражение вида $\langle \sigma \to u \rangle$, в котором левая часть $\sigma \subset S$ обозначает состояние некоторого подмножества элементов памяти, правая часть – подмножество микроопераций

$u \subset Q$, которые должны быть выполнены при наличии этого состояния. Например: ⟨содержимое входного регистра $R_{ВХ}$ = (0110) → содержимое триггера $T_1$ изменить на противоположное, содержимое $R_2$ переслать в $R_3$ ⟩.

М и к р о п р о г р а м м а – множество микрокоманд, определяющих функцию $\Phi: S \to Q$, соответствующую алгоритму, записанному в терминах заданных множеств состояний и операторов.

Основные особенности, отличающие определения этих понятий от традиционных, заключаются в следующем.

1. Микрооперация не обязательно привязана к отдельному процессорному элементу (исполнителю). Она может, например, восприниматься всеми исполнителями, а выполняться только теми из них, для которых состояния $\sigma$ равны указанным в микрокоманде.

2. Продолжительность выполнения микрокоманды не всегда фиксирована.

3. Порядок выполнения микрокоманд не обязательно определен микропрограммой. Моменты начала их выполнения определяются готовностью операндов и операторов.

1.3. Т р е б о в а н и я к я з ы к у и м е т о д а м п а р а л л е л ь н о г о м и к р о п р о г р а м м и р о в а н и я. Основной задачей параллельного микропрограммирования является создание методологии синтеза параллельных микропрограмм по заданному алгоритму и определение структур памяти и логики, которые будут их реализовать.

Поскольку методы синтеза микропрограмм должны обеспечивать возможность эффективной реализации сложных алгоритмов, они должны быть многоуровневыми. Самый низкий уровень составляют способы реализации вычислений, параллельных на уровне микроопераций. Теоретическую основу этого уровня составляют параллельные алгоритмические системы [7-9]. Наиболее приемлемым для микропрограммной реализации их вариантом являются алгоритмы параллельных подстановок [9,10], поскольку они допускают непосредственную интерпретацию сетями из автоматов. Микропрограммный способ записи таких алгоритмов составляет нижний уровень языка параллельного микропрограммирования. Способы композиции алгоритмов с учетом согласования их взаимодействий составляют второй (верхний) уровень языка. Эти способы опираются на теорию асинхронных взаимодействий и модели параллельных процессов [11] и по возможности используют опыт применения граф-схем алгоритмов и синтеза микропрограммных автоматов [12]. Наиболее приемлемым вариантом моделей параллельных процессов

является аппарат сетей Петри [13], который в сочетании с параллельной граф-схемой позволяет непосредственно переходить к микропрограммному описанию.

Реализация микропрограммных структур параллельной обработки представляется в виде множества взаимосвязанных блоков, интерпретирующих алгоритмы параллельных подстановок и алгоритмы асинхронного управления, реализующие их композицию. При этом возможны любые сочетания структур с разной степенью параллельности: однородные структуры, вычислительные среды, традиционные микропрограммные автоматы управления, сети из микропроцессоров и т.д.

Хотя основы теории параллельных алгоритмов и модели параллельных процессов в достаточной степени разработаны и число публикаций по методам параллельных вычислений непрерывно растет, до сих пор не создано еще пригодной для практики методологии параллельного микропрограммирования. Более того, нет работ, которые рассматривали бы задачу построения микропрограммных структур в целом, в виде взаимосвязанных этапов проектирования устройств с распараллеливанием вычислительного процесса до уровня микроопераций и асинхронным согласованием работы операторов.

В следующих параграфах статьи делаются попытки наметить основные пути создания методов параллельного микропрограммирования на основе оригинальных работ с привлечением теории управления параллельными процессами. §2 посвящен описанию алгоритмов параллельных подстановок и их интерпретации в виде однородных сетей из автоматов. В §3 предложен метод описания и синтеза композиции микропрограмм. Такой набор средств является, по мнению авторов, достаточно полным для построения на его основе практических приемов проектирования микропрограммных структур.

## §2. Алгоритмы параллельных подстановок и их интерпретация

2.1. О б щ а я  х а р а к т е р и с т и к а  а л г о р и т м и ч е с к о й  с и с т е м ы. Основные черты алгоритмической системы определяются ориентацией на микропрограммирование параллельных однородных структур на нижнем уровне. Эта система должна обеспечивать возможность композиции алгоритмов, допускать произвольную структуру данных и отображать следующие наиболее существенные черты переработки информации в однородных структурах [9,10]:

а) получение полного преобразования информации из совокупности локальных преобразований;

б) параллельное (асинхронное) применение всех локальных преобразований.

Кроме того, требуется компактность записи алгоритма, простота базовой микрокоманды и ее непосредственная интерпретируемость в однородной структуре.

Этим требованиям удовлетворяют алгоритмы параллельных подстановок, которые объединяют в себе черты многих известных систем и прежде всего таких, как алгоритмы Маркова и автоматы Неймана, и имеют следующие свойства:

а) в качестве объекта переработки вводится клеточное множество - совокупность клеток, имеющих имя и состояние;

б) в качестве основного средства преобразования информации вводится микрокоманда подстановки, поскольку локальное преобразование однородной структуры всегда можно представить как замену состояний одной совокупности клеток другой, в зависимости от состояний последней;

в) так же как и в алгоритмах Маркова, размеры слов в первой и левой частях микрокоманды не фиксируются и зависят от требуемого преобразования (в отличие от алгоритмов Маркова перерабатываемые слова не обязательно имеют линейную структуру);

г) структура слов задается в каждой команде указанием процедуры вычисления имен клеток;

д) в левой части микрокоманды может быть выделено специальное слово - контекст, обеспечивающее возможность композиции алгоритмов.

2.2. Алгоритм параллельных подстановок. При применении алгоритмов параллельных подстановок объектами преобразования являются клеточные множества - совокупности поименованных клеток $W=\{(a_i, m_i)\}$ $(i = 1,2,\dots)$. Каждая клетка характеризуется состоянием $a_i \in A$, т.е. символом алфавита A, вписанным в ее внутреннюю память, и именем $m_i \in M$, которое может быть ее порядковым номером, $m_i = j$, вектором ее координат в пространстве $(m_i = (i,j,k))$, значением именующей функции $(m_i = \varphi_i(y))$ и т.д. Любое конечное упорядоченное подмножество клеток $W_k \in W$, $W_k = \{(a_0, m_0),\dots,(a_n, m_n)\}$ образует слово. Векторы $Pr_1(W_k) = (a_0 \dots a_n)$ и $Pr_2(W_k) = (m_0 \dots m_n)$ называются первой и второй проекциями слова $W_k$ соответственно.

Конфигурацией S называется множество слов $\{W_j\}$ $(j=1,2,\dots)$, имеющих одинаковые первые проекции, а вторые

проекции являются значениями одинаковых функций от некоторого аргумента $m_0 \in M$, т.е.

$$\forall W_j \in S:\ Pr_1(W_j)=(a_0 \ldots a_n),\ Pr_2(W_j)=(\varphi_0(m_0),\ldots,\varphi_n(m_0)).$$

В пространстве "клеток" конфигурация представляет собой некоторый геометрический образ.

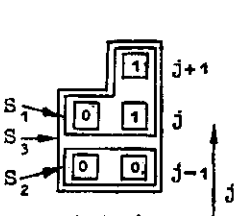



Рис.I

П р о и з в е д е н и е м конфигураций $S_1 = \{(a_0, \varphi_0), \ldots, (a_{n_1}, \varphi_{n_1})\}$ и $S_2 = \{(b_0, \phi_0), \ldots, (b_{n_2}, \phi_{n_2})\}$ называется конфигурация $S_1 \cdot S_2 = \{(a_0, {}^2\varphi_0), \ldots, (a_{n_1}, \varphi_{n_1}), (b_0, \phi_0), \ldots, (b_{n_2}, \phi_{n_2})\}$, где $\varphi_j$ и $\psi_j$ являются функциями от одного и того же аргумента.

Пример геометрических образов конфигураций $S_1 = \{(1,(i,j)),(0,(i+1,j)),(1,(i,j+1))$ и $S_2 = \{(0,(i+1,j+1)),(0,(r,j-1))\}$ и их произведения $S_3 = S_1 * S_2$ показана на рис. I.

Процесс переработки клеточного множества заключается в применении к нему микрокоманд подстановки.

М и к р о к о м а н д о й подстановки является выражение вида

$$\theta: S_1 * S_2 \to S_3 \quad ,$$

которое имеет следующий смысл: если в клеточном множестве W существует слово $W_{12} = W_1 * W_2 (W_{12} \in S_1 * S_2)$, то $W_1$ заменяется на $W_3$ ($W_1 \in S_1$, $W_3 \in S_3$). Эта операция замены для каждого конкретного $W_{12}$ называется м и к р о о п е р а ц и е й подстановки; $S_1 * S_2$ называется левой частью подстановки, $S_2$ – контекстом, $S_3$ – правой частью.

Микрокоманда подстановки называется п р и м е н и м о й к данному клеточному множеству, если в нем существует хотя бы одно слово $W_{12} \in S_1 * S_2$, если такого слова в W не существует, то $\theta$ называется н е п р и м е н и м о й.

Конечное множество операций подстановок для пары (A, M) называется с и с т е м о й, или а л г о р и т м о м п а р а л л е л ь н ы х п о д с т а н о в о к. Неупорядоченное множество соответствующих им микрокоманд

$$\Phi = \begin{cases} \theta_1 : S_{11} * S_{12} \to S_{13} \\ \theta_k : \ldots \\ \theta_m : S_{m1} * S_{m2} \to S_{m3} \end{cases} \qquad (I)$$

образуют м и к р о п р о г р а м м у реализации этого алгоритма для $W \in A \times M$.

Выполнение алгоритма по преобразованию клеточного множества $W_0$ задается следующей итерационной процедурой. Пусть $W^{i-1}$ - результат $(i-1)$-й итерации. Если ни одна из микрокоманд алгоритма неприменима к $W^{i-1}$, то $W^{i-1}$ является р е з у л ь т а т о м выполнения микропрограммы $\Phi$ над $W_0$. Иначе, в соответствии с левыми частями всех применимых микрокоманд подстановок осуществляются микрооперации замены. Полученное $W^i$ является исходным для следующей итерации.

Микропрограмма $\Phi = \{\theta_1, \ldots, \theta_m\}$ называется н е п р о т и в о р е ч и в о й относительно $W$ [9], если на каждом итерационном шаге ее выполнения не возникает клеточное множество, содержащее две клетки с одинаковыми именами, но с различными состояниями. Если микропрограмма $\Phi$ непротиворечива относительно всех $W_i \in A \times M$, то она непротиворечива относительно $(A, M)$.

Для непротиворечивых микропрограмм существенна одновременность изменения состояний во всех клетках для каждого итерационного шага.

На рис.2. приведена микропрограмма сложения трех четырехразрядных чисел и показана работа алгоритма параллельных подстановок.

Алгоритмы параллельных подстановок допускают различные способы их композиции [10]. Простейшими из них являются:

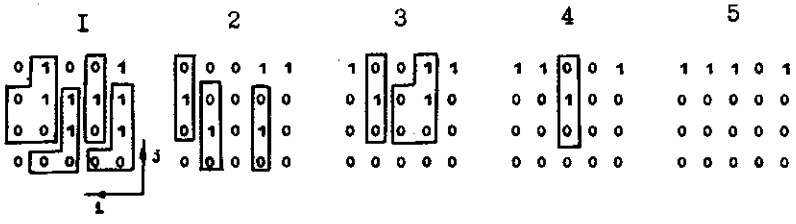
а) суперпозиция: $\Phi(W) = \Phi_1(\Phi_2(W))$,

б) объединение: $\Phi(W) = \Phi_1(W) \cup \Phi_2(W)$,

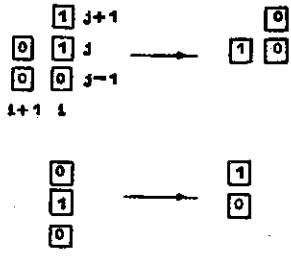
в) условные ветвления:

$$\Phi(W) = \begin{cases} \Phi_2(W), & \text{если } \Phi_1 \text{ неприменим к } W, \\ \Phi_3(W), & \text{если } \Phi_1 \text{ применим к } W. \end{cases}$$

Более сложные виды композиции разрабатываются на основе модели асинхронного управления (п. 3.4). Для включения алгоритма $\Phi_i$ в композицию необходимо перейти к его расширению $\Phi_i'$ (см.[10]), добавив в него несколько подстановок и клеток, среди которых имеется




а)




б)

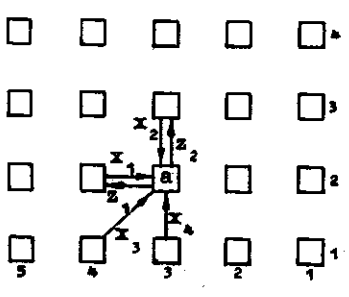
$\Theta_1=\{(1,(i,j)),(0,(i+1,j)),(1,(i,j+1))\}*\{(0,(i,j+1)),$
$(0,(i+1,j-1))\}\rightarrow\{(0,(i,j)),(1,(i+1,j)),(0,(i,j+1))\}.$
$\Theta_2=\{(1,(i,j)),(0,(i,j+1))\}*\{(0,(i,j-1))\}\rightarrow\{(0,(i,j),(1,(i,j+1))\}.$

в)

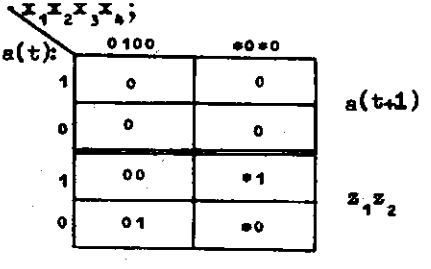
Рис. 2. Реализация алгоритма сложения многих чисел:
а) итерация преобразования клеточных множеств;
б) геометрические образы подстановок; в) микропрограмма.

клетка с именем $a_i$, состояние которой указывает на неприменимость $\Phi_i$. Расширение алгоритма для алфавита {0,I} при включении его в композицию приведено на рис. I0.

2.3. Интерпретация алгоритмов параллельных подстановок сетью из автоматов. Наиболее прямым способом интерпретации алгоритмов параллельных подстановок является отображение клеточных множеств в сеть из одинаковых автоматов. Каждый автомат имеет свое имя $m_j \in M$ и внутреннюю память для хранения символов $a_i \in A$. Структура связей в сети представляется вторыми проекциями слов, задаваемых конфигурациями левых и правых частей подстановок, входящих в алгоритм $\Phi$ (1), причем для каждого автомата $m_j$ сети указываются соседи по входам $M_j^{in}$ и соседи по выходам $M_j^{out}$. Соседи по входам определяются левой частью команды, соседи по выходам – правой частью команды. Например, клетка (I,(3,2)) сети (рис.3,а), реализующей алгоритм сложения многих чисел (рис.2), имеет $M^{in}$={(3,2), (3.3), (3,I), (4,I), (4,2)}, $M^{out}$ = {(3.2), (4.2), (3.3)}. Поведение автоматов сети определяется первыми проекциями микрокоманд алгоритма. Например, для алгоритма сложения (рис.2) таблица переходов приведена на рис.3,б.




а)




б)

Рис. 3

По сути дела, $\Phi$ является микропрограммным описанием автомата, и для получения его логической структуры можно использовать известные методы синтеза микропрограммных автоматов [I2], приводящие в результате к схеме, состоящей из следующих блоков: I – ячейка памяти данных (внутренняя память автомата), 2 – память микропрограмм, 3 – входные регистры (рис.4).

Реализация такой схемы при современных условиях возможна в виде специальной БИС либо в виде соединения из стандартных ПЗУ или ПЛМ (блок 2) и регистровых БИС (блоки I и 3).

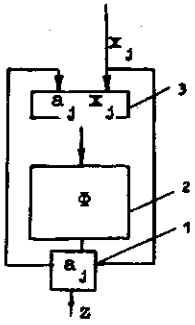



Рис. 4

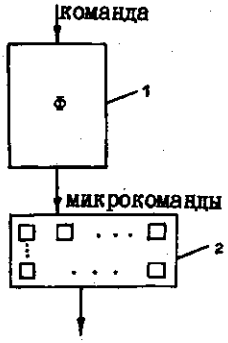



Рис. 5

Значительная экономия по оборудованию может быть достигнута, если блок памяти микропрограмм, повторяющийся в каждом автомате, сделать общим для всех автоматов (рис.5), а в автоматах вместо него поставить схемы сравнения. Полученные устройства состоят из двух однородных частей: I) памяти микропрограмм и 2) памяти данных, где производится их переработка, и называются о д н о р о д н ы м и м а ш и н а м и [IO]. Микрокоманды одна за другой в произвольном порядке поступают на все ячейки памяти данных, в которых производится сравнение их с входными состояниями и запоминание результата сравнения до окончания цикла выдачи микропрограммы. После окончания цикла все ячейки одновременно меняют свое состояние. Циклы повторяются до тех пор, пока микропрограмма применима.

Элементной базой для однородных машин могут служить стандартные БИС памяти с произвольной выборкой (если микропрограммы надо часто менять), программируемые или перепрограммируемые ПЗУ или ПЛМ для реализации памяти микропрограмм и специальные БИС – для памяти данных. Ячейки памяти данных могут быть заменены стандартными микропроцессорами, однако при этом вводится избыточность как по оборудованию, так и по времени.

Очевидно, что экономия оборудования в однородных машинах по сравнению с прямым способом интерпретации достигается за счет последовательной выдачи микрокоманд и тем самым растягивания процесса вычисления во времени. Идеальным вариантом однородной машины является такой, в котором вся микропрограмма одновременно подается на все ячейки памяти данных. Реализацию этого "идеального

варианта" можно, по-видимому, осуществить, используя спектральный метод представления совокупностей микрокоманд.

## §3. Синтез асинхронного управления параллельными микропрограммами

3.I. П о с т а н о в к а  з а д а ч и  и  э т а п ы  с и н т е з а. Задача синтеза асинхронного микропрограммного управления ставится следующим образом. Предполагается, что в вычислительном устройстве реализуется алгоритм, заданный в виде композиции некоторого множества операторов. Следует найти микропрограммную запись условий их выполнения и определить структуру памяти и логики, реализующих выработку управляющих сигналов.

Асинхронный принцип управления требует, чтобы управляющая микропрограмма имела двухстороннюю связь с каждым оператором. От нее оператор получает управляющие сигналы функциональной настройки (код операции, контекст операторов) и синхронизирующий сигнал "готов" (ready), разрешающий оператору начать работу. От операторов управляющая микропрограмма воспринимает сигналы подтверждения о выполнении оператора (acknoledge) и значения предикатов условных переходов.

Процедуру синтеза микропрограммы управления будем подразделять на три следующих этапа.

Э т а п I. Составление параллельной граф-схемы алгоритма (в дальнейшем просто граф-схемы). Ее можно рассматривать как расширение (обобщение) обычных граф-схем алгоритмов [I2] или асинхронный вариант параллельных граф-схем [I4]. Основными трудностями этого этапа является обеспечение детерминированности и отсутствия дэдлоков и зависаний (п. 3.2), возможных при асинхронных параллельных взаимодействиях.

Э т а п II. Переход от граф-схемы к абстрактной модели поведения управляющей структуры (модели управления) в виде сети Петри [I3]. На основе этой модели осуществляется разрешение конфликтов по ресурсам и определение внутренней структуры микропрограммного управления.

Э т а п III. Переход от абстрактной модели (сети Петри) к конкретной микропрограмме, которая либо является управляющей частью микропрограммы - композиции алгоритмов операторов, либо определяет работу автономного устройства управления. Этот этап включает в себя кодирование состояний, оптимизацию логики и вложение в реальные структуры памяти и логики.

3.2. П а р а л л е л ь н а я  г р а ф - с х е м а  а л г о р и т м а. Предлагаемая здесь граф-схема является обобщением широко используемых при проектировании микропрограммных автоматов граф-схем алгоритмов [I2]. В нее добавлены средства отображения распараллеливания процесса, и обеспечивается возможность различать слияние параллельных ветвей и ветвей условного ветвления [I5]. Последнее помогает устранению неприятностей, связанных с асинхронной работой, и способствует формализации синтеза микропрограммного управления.

Граф-схему будем представлять ориентированным графом $G = \langle V, E \rangle$ с двумя типами вершин:

I) вершины $V_\Sigma = \{B, K, \Phi, W, U\}$, обозначающие события, изменяющие состояние системы, которые, в свою очередь, подразделяются на управляющие $\{B, K, W, U\}$ и операторные $\Phi$;

2) вершины условные $v_\Omega = \{\Omega, \Delta\}$, указывающие на выбор предыдущего ($\Delta$) или следующего ($\Omega$) события.

Существование дуги $e_{ij} \in E$ из $v_i$ в $v_j$ обозначает, что событие или условие $v_j$ зависит от результата $v_i$. Обозначения и содержательный смысл вершин и дуг показаны на рис.6.

Существует 4 типа отношений между вершинами параллельной граф-схемы.

I) Строгий порядок: $v_i < v_j$, если от $v_i$ к $v_j$ существует простой направленный путь $p_{ij}$, а от $v_j$ к $v_i$ такого пути нет.

2) Циклический порядок: $v_i \leftrightarrows v_j$, если существует простой контур, проходящий через обе вершины.

3) Неопределенный порядок: $v_i \parallel v_j$, если не существует направленного пути ни от $v_i$ к $v_j$, ни от $v_j$ к $v_i$, но существует $v_k \in W \cup \Phi$, такая что

$$v_k < v_i, \quad v_k < v_j, \quad p_{ki} \cap p_{kj} = v_k.$$

4) Взаимное исключение: $v_i \not\leftrightarrows v_j$, если не существует направленных путей $p_{ij}$ и $p_{ji}$, но существует $v_k \in \Omega$, такая что

$$v_k < v_i, \quad v_k < v_j, \quad p_{kj} \cap p_{ki} = v_k.$$

Граф-схему будем называть д е т е р м и н и р о в а н н о й [2I], если результат вычисления инвариантен последовательности выполнения микрокоманд, соответствующих вершинам граф-схемы, связанных отношением неопределенного порядка. Очевидно, что условие детерминированности не может быть выполнено, если в граф-схеме имеются конфигурации (рис.7), соответствующие следующим от-

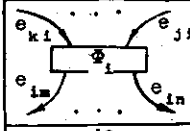
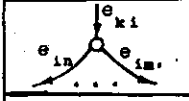
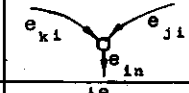
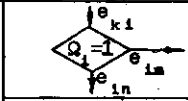
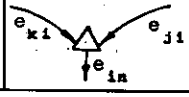
| Тип вершины | Обозначение | | Содержательный смысл |
|---|---|---|---|
| | Графическое | Символьное | |
| События | Начало | $v_i \in B$ | Вход в микропрограмму |
| | Конец | $v_i \in K$ | Выход из микропрограммы |
| |  | $v_i \in \Phi$ | Выполнение оператора $\Phi_i$ : выполняется после $\{v_k \ldots v_j\}$ и раньше, чем $\{v_n \ldots v_m\}$. |
| |  | $v_i \in W$ | Распараллеливание процесса: $\{v_n \ldots v_m\}$ выполняются после $v_k$. |
| |  | $v_i \in U$ | Соединение параллельных ветвей: $v_n$ выполняется после $\{v_k \ldots v_j\}$. |
| Условия |  | $v_i \in \Omega$ | Условное ветвление: после $v_k$ выполняются либо $v_n$ (если $\Omega_i = 1$), либо $v_m$ (если $\Omega_i \neq 1$). |
| |  | $v_i \in \Delta$ | Условное объединение: $v_n$ выполняется либо после $v_k$, либо после $v_j$. |

Рис. 6. Типы вершин параллельной граф-схемы алгоритма.

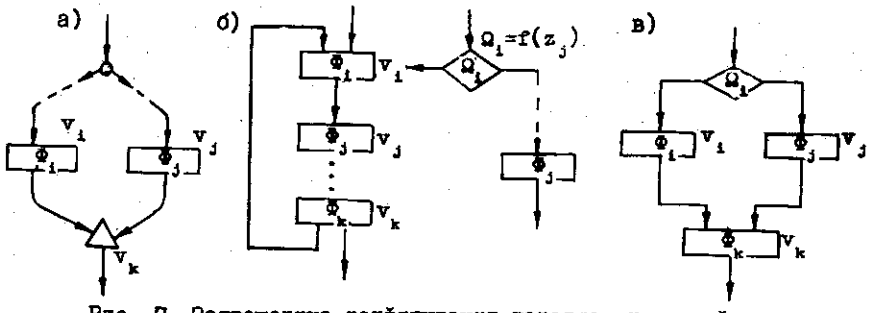



Рис. 7. Запрещенные конфигурации параллельных граф-схем:
а) фрагмент недетерминированной схемы;
б) дэдлок; в) зависание.

ношениям:

$$v_i \| v_j, \quad v_i > v_k, \quad v_j > v_k, \quad p_{ik} \cap p_{jk} = v_k, \quad v_k \in \Delta.$$

Кроме того, в детерминированной граф-схеме недопустимы некоторые конфигурации, которые приводят к тому, что вычисление оказывается невыполнимым. К таким конфигурациям относятся дэдлоки (смертельные объятия) и зависания [I3].

Д э д л о к имеет место (рис.7,б), если в граф-схеме существует пара вершин, связанная одним из следующих соотношений:

1) $v_i \geq v_j$, причем ни $p_{ij}$, ни $p_{ji}$ не содержат вершин, $v_k \in \Delta$.
2) $v_i < v_j$, $v_i \in \Omega$, $\Omega_i = f(z_j)$, $z_j$–результат $\Phi_j$.

З а в и с а н и е имеет место в граф-схеме (рис.7,в), если в ней существуют 3 вершины, связанные следующими соотношениями:

$$v_j < v_k; \quad v_i < v_k; \quad v_j \not< v_i; \quad v_k \in \Phi \cup \Pi.$$

Один из алгоритмов проверки граф-схемы на детерминированность приведен в [2I].

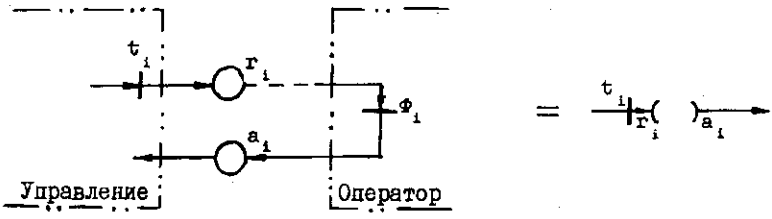
3.3. А б с т р а к т н а я м о д е л ь а с и н х р о н н о г о у п р а в л е н и я. Граф-схема полностью описывает алгоритм асинхронной композиции операторов, однако форма этого описания не дает представления о внутренней структуре управляющей микропрограммы, т.е. об объеме внутренней памяти, условиях разрешения ресурсных конфликтов, логических зависимостях между внутренними и внешними сигналами и т.д. Необходима промежуточная абстрактная модель поведения управляющей микропрограммы, в качестве которой будем использовать сеть Петри [I3,2I]. *)

Если сеть Петри моделирует поведение системы, реализующей композицию операторов $\{\Phi_1, \ldots, \Phi_n\}$, заданную граф-схемой, то множество ее переходов подразделяется на два подмножества:

1) $\Sigma_\Phi$ соответствует множеству операторов системы,
2) $\Sigma_C$ соответствует множеству управляющих микрокоманд.

Пусть, например, выработка сигнала $r_i$, включающего оператор $\Phi_i$ (рис.8,а), моделируется срабатыванием перехода $t_i \in \Sigma_C$. При этом в позиции $r_i \in t_i$ появляется метка. После окончания работы $\Phi_i$ вырабатывает сигнал $a_i$, что моделируется срабатыванием $t(\Phi_i) \in \Sigma_\Phi$

*) Определение и основные свойства сети Петри см. в статье П.А.Анишева [2I].



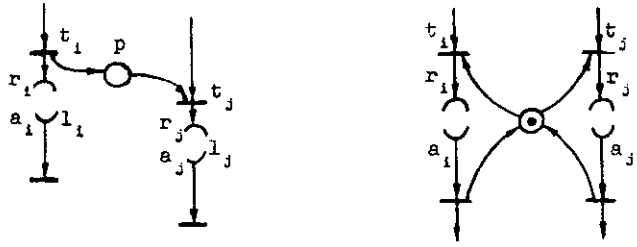

а) б)

Рис. 8

и переходом метки в $a_i$ . Для упрощения фрагмент сети Петри, моделирующий работу оператора, изображается в виде одной разорванной позиции [I8 ] (рис.8,б). Действия, соответствующие операторным вершинам $v_i \in \{ B \cup K \cup W \cup U \}$, моделируются переходами $t_{n+j} \in \Sigma_C$ , реализующими внутреннюю логику управления. Множество позиций сети определяет множество сигналов управления, а множество достижимых маркирований – множество состояний системы.

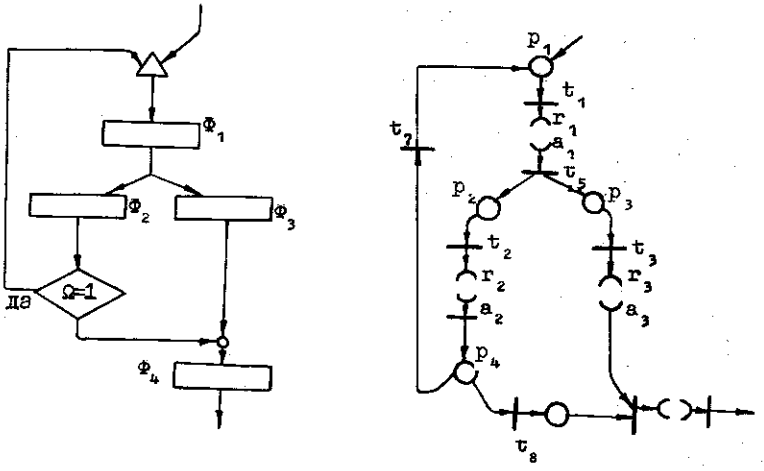
Модель сети Петри позволяет простыми формальными способами ввести условия разрешения р е с у р с н ы х конфликтов.

Два оператора $\Phi_i$ и $\Phi_j$ находятся в ресурсном конфликте, если $v_i \| v_j$ и в соответствующих им микрокомандах участвует один и тот же ресурс. Ресурсные конфликты могут быть разрешены двумя путями: а) установлением очереди, б) арбитражем. Первый предполагает задание приоритетов и реализуется введением в сеть Петри дополнительной позиции (рис.9,а). Второй предоставляет ресурс тому оператору, который раньше окажется готовым (рис.9,б).




а) б)

Рис. 9




а)

$$\Phi_i' = \begin{cases} \{s_{j1}*(1,m_i')\}*\{s_{j2}*(1,r_i)\} \Rightarrow \{s_{j3}*(0,m_i')\} \\ \text{для всех } \theta_j \in \Phi_i; \\ \{(1,m_i)\}*\{(1,r_i)\} \Rightarrow \{0,m_i\}; \\ \{(0,m_i'),(0,m_i)\}*\{(1,r_i)\} \Rightarrow \{(1,m_i'),(1,m_i)\}, \quad i=1,2,3,4; \\ \{(0,m_i),(0,a_i),(1,r_i)\}*\{(1,m_i')\} \Rightarrow \{(1,m_i),(0,r_i),(1,a_i)\}; \end{cases}$$

$$\Phi_c = \begin{cases} \{(0,r_i),(1,p_i)\} \Rightarrow \{(1,r_i),(0,p_i)\}, \quad i=1,2,3 \quad (t_1,t_2,t_3) \\ \{(0,r_4),(1,p_5),(1,a_3)\} \Rightarrow \{(1,r_4),(0,p_5),(0,a_3)\} \quad (t_4) \\ \{(0,p_2),(0,p_3),(1,a_1)\} \Rightarrow \{(1,p_2),(1,p_3),(0,a_1)\} \quad (t_5) \\ \{(0,p_4),(1,a_2)\} \Rightarrow \{(1,p_4),(0,a_2)\} \quad (t_6) \\ \{(1,p_4),(0,p_1)\}*\{(1,\Omega)\} \Rightarrow \{(0,p_4),(1,p_1)\} \quad (t_7) \\ \{(1\ p_4),(0\ p_5)\}*\{(0,\Omega)\} \Rightarrow \{(0\ p_4),(1\ p_5)\} \quad (t_8) \end{cases}$$

б)

Рис. 10. Граф-схема композиции алгоритмов, сеть Петри и микропрограмма управления.

Переход от граф-схемы к соответствующей сети Петри сводится к следующей формальной процедуре: I) замене вершин граф-схемы соответствующими фрагментами сети Петри, 2) выявлению ресурсных конфликтов и введению в сеть дополнительных позиций, обеспечивающих очередь или арбитраж. При этом если полученная сеть Петри живая и безопасная, то граф-схема детерминированная [2I].

На рис.I0,а представлена граф-схема алгоритма композиции операторов $\{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$ и сеть Петри управляющей микропрограммы.

3.4. Синтез асинхронной микропрограммы и способы ее реализации. Форма представления микрокоманд асинхронного управления и синтеза микропрограммы зависит от способа ее реализации. При этом следует различать два случая: I) централизованное управление, когда управляющая микропрограмма находится в специальном управляющем устройстве, 2) распределенное управление, когда управляющие микрокоманды хранятся совместно с микрокомандами операторов, образуя одну микропрограмму, соответствующую композиции алгоритмов операторов. В обоих случаях управляющие микрокоманды могут быть записаны непосредственно по сети Петри. При этом каждому переходу $t \in \Sigma$ ставится в соответствие одна микрокоманда. Левая ее часть отображает условия срабатывания $t$, а правая – изменение маркирований.

Для случая централизованного управления в настоящее время намечается два подхода к синтезу асинхронного управления.

I. Автоматный – с вложением микропрограмм в ЗУ, ПЗУ и ПЛМ [I6,I7].

2. Непосредственное вложение сети Петри в специальную структуру типа памяти с логикой [I8,I9].

Автоматный подход основывается на следующей автоматной интерпретации сети Петри:

I) множеству допустимых маркирований соответствует множество полных состояний автомата;

2) позициям типа $П_a$ и $П_Q$ соответствуют входные переменные, позициям типа $П_r$ и $П_Q$ – внутренние, среди которых $П_r$ – выходные;

3) срабатывание перехода соответствует переходу автомата из состояния в состояние.

Микрокомандой в автоматном подходе является выражение вида

$$\langle м(П) \to м'(П_Q),\ м'(П_r) \rangle, \quad \forall м(П),$$

где левая часть – маркирование сети до срабатывания перехода, правая – маркирование внутренних и выходных позиций после срабатывания перехода. Число микрокоманд в микропрограмме равно числу достижимых маркирований сети Петри. Это число при наличии многих параллельных ветвей очень велико. Поэтому непосредственное вложение такой микропрограммы требует больших объемов управляющих памятей. Однако применение методов логической оптимизации с последующим вложением в ПЛМ [20] или декомпозиции сетей Петри [17] может сделать автоматный подход целесообразным, особенно если учесть его несомненные преимущества: стандартную элементную базу и накопленный опыт синтеза традиционных микропрограммных автоматов.

Вложение живых и безопасных сетей Петри в специальную структуру основано на представлении микрокоманд в виде локальных условий срабатывания для каждого перехода $t$:

$$\langle M(p\in{}^{\cdot}t) = 1 \rightarrow M'(p\in{}^{\cdot}t)=0; \quad M'(p\in t^{\cdot})=1\rangle. \qquad (2)$$

Число микрокоманд в такой микропрограмме равно числу переходов в сети. Предложено два типа таких структур [18,19], состоящих из достаточно простых ячеек с перестройкой. Формальный синтез такой структуры может быть выполнен путем построения сети из автоматов, интерпретирующей алгоритм параллельных подстановок $\Phi_c$, в котором каждая микрокоманда является выражением вида (2), записанным в терминах подстановок (I), где

$$\Phi_c = \{\langle\{(1,p\in{}^{\cdot}t),(0,p\in t^{\cdot})\} \rightarrow \{(0,p\in{}^{\cdot}t),(1,p\in t^{\cdot})\}\rangle\}, \qquad (3)$$

$$\forall t\in\Sigma .$$

В любом случае централизованное устройство управления является сложной схемой асинхронного типа, которая, кроме всех перечисленных требований, должна удовлетворять условиям отсутствия гонок, состязаний и потерь сигналов, что вызывает необходимость ставить специальные триггеры [16], синхронизаторы [16] и задержки на входы и выходы структуры.

Случай распределенного управления предполагает использование методов асинхронного управления для композиции $\Phi$ алгоритмов $\{\Phi_1,\ldots,\Phi_m\}$ параллельных подстановок. При этом операторами являются микропрограммы $\Phi_i'$, расширенные для композиции, как показано на рис.I0,б, причем имена клеток $r_i$ и $a_i$, определяющие включение $\Phi_i$ в композицию и конец работы $\Phi_i$, имеют смысл позиций $r_i$ и $a_i$

в сети Петри. Кроме того, клеточное множество, соответствующее композиции, дополняется клетками с именами, соответствующими позициям $p_k \in \Pi_Q \cup \Pi_\Omega$. Микропрограмма $\Phi$ композиции операторов $\{\Phi_i\}$, заданной сетью Петри $\{\Sigma, \Pi\}$, объединяет следующие множества микрокоманд:

$$\Phi = \bigcup_{i=1}^{m} \Phi_i' \cup \Phi_C ,$$

где $\Phi_C$ – множество микрокоманд вида (3), $\forall t \in \Sigma$.

На рис.10,б приведена микропрограмма асинхронной композиции алгоритмов параллельных подстановок, заданная граф-схемой и сетью Петри (рис.10,а). На рис.11 приведена схема однородной машины, реализующей композицию $\Phi$. Схема построена в соответствии с алгоритмом (рис.10,б) и состоит из памяти данных, образованной клеточными множествами $W_1, W_2, W_3$ и $W_4$, хранящими перерабатываемые данные для алгоритмов $\Phi_1, \Phi_2, \Phi_3$ и $\Phi_4$ клеточных множеств, образованных ячейками $m_i, m_i', r_i, a_i, p_i$ $(i = 1, 2, \ldots, 5)$, а также из памяти программ, хранящей совокупность микрокоманд. Возможно, например, такое устройство этих памятей. Память данных состоит из совокупности одинаковых клеток, функциональный блок каждой клетки содержит столько схем сравнения, сколько строк в $\Phi$, память программ – блок, выдающий одну микрокоманду на одну схему сравнения каждой ячейки. Такая сеть реализует алгоритм с предельным распараллеливанием и однородностью. Разбивая алгоритм $\Phi$ на части, хранимые в различных блоках памяти данных, и варьируя числом схем сравнения в ячейках, можно получить более простые сети, правда, с меньшей степенью параллельности.

## Заключение

Предложенный набор средств описания и методов синтеза параллельных микропрограмм и их композиций составляют основу методологии создания микропрограммных структур параллельной обработки информации. Разработка на этой основе рабочих методов практического проектирования требует решения еще ряда сложных задач, основными среди которых являются:

1) исследование способов эквивалентных преобразований и оптимизации параллельных микропрограмм и их композиций;

2) разработка языка и формальных методов записи алгоритмов в терминах параллельных подстановок;

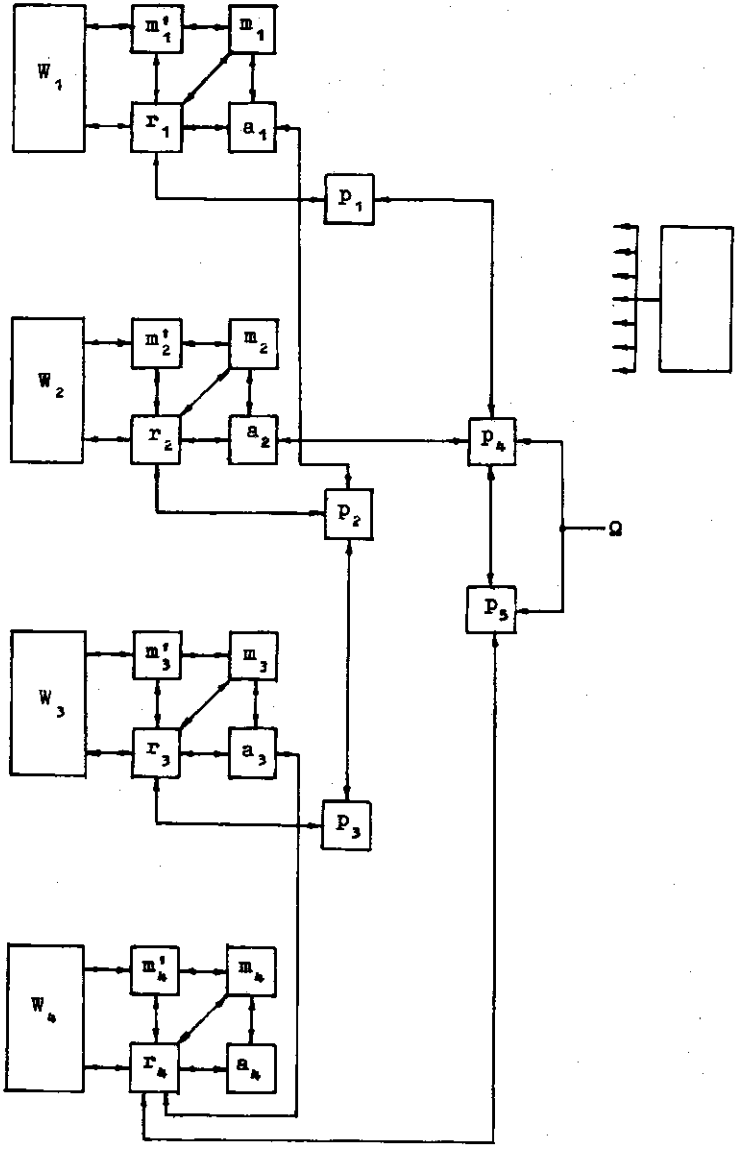



Рис.II. Сеть, реализующая композицию Φ.

3) создание машинной модели преобразования информации в параллельных микропрограммных структурах, позволяющей отлаживать параллельные микропрограммы, оценивать и сравнивать эффективность вариантов;

4) определение условий и методов проверки непротиворечивости параллельных микропрограмм и бесконфликтности их композиций;

5) разработка эффективных методов автоматизированного синтеза микропрограммных структур обработки и управления.

Современный уровень развития технологии БИС уже сейчас позволяет реализовать микропрограммные структуры параллельного типа и делает разработку методологии их проектирования насущной необходимостью.

## Литература


I. ДИЙКСТРА Э. Взаимодействие последовательных процессов. -В кн.: Языки программирования. М., "Мир", 1972.

2. БАНДМАН О.Л., ЕВРЕИНОВ Э.В., КОРНЕЕВ В.В., ХОРОШЕВСКИЙ В.Г. Однородные вычислительные системы на основе микропроцессорных БИС. -В кн.: Вопросы теории и построения вычислительных систем. (Вычислительные системы, вып.70.) Новосибирск, 1977, с. 3-28.

3. Однородные микроэлектронные ассоциативные процессоры. М., "Сов.радио", 1973.

4. ФЕТ Я.И. Массовая обработка информации в специализированных однородных процессорах. Новосибирск, "Наука", 1976.

5. YAMADA H., AMOROSO S. Tesselation Automata.-"Inform. and Control",1969,v.14, N 3,p.299-317.

6. ЕВРЕИНОВ Э.В., ПРАНГИШВИЛИ И.В. Цифровые автоматы с настраиваемой структурой. М., "Энергия", 1974.

7. БАРЗДИНЬ Я.М. Проблема универсальности в теории растущих автоматов. -"Докл. АН СССР", 1964, т.157, № 3, с.542-545.

8. КОЛМОГОРОВ В.Н., УСПЕНСКИЙ В.А. К определению алгоритма. -"Успехи мат.наук", 1958, т.13, вып.482, с.3-13.

9. КОРНЕЕВ Ю.Н., ПИСКУНОВ С.В., СЕРГЕЕВ С.Н. Алгоритмы обобщенных подстановок и их интерпретация сетями автоматов и однородными машинами. -"Изв. АН СССР, Техническая кибернетика",1971, №6, с. 131-142.

10. КОРНЕЕВ Ю.Н., ПИСКУНОВ С.В., СЕРГЕЕВ С.Н. Вопросы построения алгоритмов обобщенных подстановок с выделенным контекстом. -В кн.: Вычислительные системы. Вып.47, Новосибирск, 1971, с.117-129.

11. LIPTON R.J.,SNYDER L., ZALZSTEIN Y. A comparative study of Models of Parallel Computation.- 15th Annual Symposium on Switching and Automata Theory, 1974, p.145-155.

12. БАРАНОВ С.И. Синтез микропрограммных автоматов.Л., "Энергия". 1974.

13. HACK M. Analysis of Production Schemata by Petry Nets. MAC, TR-94, MJT, 1972.

I4. ПИЙЛЬ Е.И. Особенности синтеза центрального блока управления при наличии параллельных участков в алгоритме. -В кн.: Синтез автоматов и управление на сетях связи. М., "Наука", I973, с.5-I5.

I5. RUMBAUGH J.E. A Parallel Asynchronous Computer Architechture for Data Flow Programs. MIT,June 1975, AD A012-822.

I6. CORSINI P.,FROSINI G. A model for Asynchronous Control Networks.-"Digital Processe",1976, N 1,p.47-62.

I7. ANDRÉ C.,BOERI F.,MARIN J. Synthése et realisation des systemes logiques a evolutions simultanees.-"Revue Francaise d'Automatique Informatique.Recherche operationell",1976,v.10,N 4,p.67-87.

I8. PATIL S.S. An asynchronous Logic Array. Project MAC, MJT, May 1975.

I9. JUMP J. Asynchronous Control Arrays.- "IEEE Trans. Computers",1974,v.23, N 10,p.1020-1029.

20. АЧАСОВА С.М.,БАНДМАН О.Л. Проектирование автоматов управления на основе БИС. -В кн.: Автоматизация проектирования в микроэлектронике. (Вычислительные системы. Вып. 64.) Новосибирск, I975, с.26-53.

2I. АНШЕВ П.А. О детерминированности параллельных граф-схем. -Настоящий сборник, с.40-52.

Поступила в ред.-изд.отд.
5 сентября I977 года